HLS for “Driver Drowsiness Detection” Model

Group Number: 28

Group members:

Ritam Mohanty , 234101043
Amiya Ranjan Panda , 234101010
Alexander Jamatia , 234101007
Chemat Wengail , 234101012
Niranjan Kesavan , 234101033

1. Model Description :

The "**Driver Drowsiness Detection**" model is a type of system designed to monitor a driver's state of wakefulness or drowsiness while they are operating a vehicle. This technology is crucial for ensuring road safety, as drowsy driving can be as dangerous as driving under the influence of alcohol or drugs.

This model is a convolutional neural network (CNN) with some dropout layers for regularization.

Different Layers and how these layers help in a Driver Drowsiness Model:

- **Convolutional Layers:** These layers extract features from the input images, such as facial expressions or eye movements, which are crucial for detecting drowsiness.
- **MaxPooling Layers:** They reduce the spatial dimensions, focusing on the most important features while reducing computation.
- **Flatten Layer:** .Converts the 3D output from the convolutional and pooling layers into a 1D vector.Prepares the data for input into the dense layers.
- **Dropout Layers:** Regularize the model by randomly setting a fraction of input units to 0 during training. This helps prevent overfitting.
- **Dense Layers:** These fully connected layers learn complex patterns in the extracted features and make the final decision about the driver's state (drowsy or not).

Task of model	Number of layers	Type of Layers	Parameters
Model aims to enhance road safety by alerting drivers when signs of drowsiness are detected, potentially preventing accidents caused by fatigue while driving.	17	conv2d (Conv2D)	640
		max_pooling2d (MaxPooling2D)	0
		conv2d_1 (Conv2D)	18464
		max_pooling2d_1 (MaxPooling2D)	0
		conv2d_2 (Conv2D)	9248
		max_pooling2d_2 (MaxPooling2D)	0
		conv2d_3 (Conv2D)	4624
		max_pooling2d_3 (MaxPooling2D)	0
		conv2d_4 (Conv2D)	2320
		max_pooling2d_4 (MaxPooling2D)	0
		flatten (Flatten)	0
		dropout (Dropout)	0
		dense (Dense)	2176
		dropout_1 (Dropout)	0
		dense_1 (Dense)	8256
		dropout_2 (Dropout)	0
		dense_2 (Dense)	65

2. Changes to Keras2c :

Changes made to the **CNN_module.c** and **CNN_test_suite.c** files are as as follows:-

- **Function declarations and structure movements:**

Initially, all function declarations, such as **void k2c_flatten()** and **void k2c_dense()**, were declared and defined in separate header files. However, to prevent compilation errors, we consolidated the definitions of all necessary functions within our source file **CNN_module.c**.

```
void k2c_bias_add(float A_array[], size_t A_numel, const float b_array[], const size_t b_numel) {


  #pragma HLS array_partition variable=A_array cyclic factor=16 dim=1
  #pragma HLS array_partition variable=b_array cyclic factor=16 dim=1
  for (size_t i=0; i<A_numel; i+=b_numel) {         //OPTIMIZATION
      for (size_t j=0; j<b_numel; ++j) {
          #pragma HLS unroll factor=8
          A_array[i+j] += b_array[j];
      }
  }
}
```

- **Removing memcpy() and memset() function:**

The **memcpy()** function, originally employed to copy values from one array to another, caused a Bus Pointer error due to lack of support. To circumvent this issue, we replaced it with a for loop to iterate through each element and copy values individually:

```
// memcpy(output->array, input->array,
input->numel*sizeof(input->array[0]));
    for (i = 0; i < input_numel; ++i) {
        output_array[i] = input_array[i];
    }
```

Similarly, the **memset()** function, previously used to initialize all values of an array to zero, also led to the same issue. To resolve this, we substituted it with a for loop to manually set each element to zero:

```
// memset(C, 0, outrows*outcols*sizeof(C[0]));
    for (i = 0; i < outrows*outcols; ++i) {
        C[i] = 0;
    }
```

- **<u>Removing Function pointers:</u>**

  Initially function pointers were defined and declared, and were being passed as parameters inside other functions.

```
typedef void k2c_activationType(float * x, const size_t size);

void k2c_relu_func(float * x, const size_t size) {

    for ( i=0; i < size; ++i) {
        if (x[i] <= 0.0f) {
            x[i] = 0.0f;
        }
    }
}
k2c_activationType * k2c_relu = k2c_relu_func;
```

  For example, **k2c_relu** was being passed inside **k2c_dense()** function -

```
k2c_dense1(dense_output_array,dense_output_numel,

dropout_output_array,dropout_output_ndim,dropout_output_numel,dropout_output_shape,

dense_kernel_array,dense_kernel_ndim,dense_kernel_numel,dense_kernel_shape,

dense_bias_array,dense_bias_ndim,dense_bias_numel,dense_bias_shape,
            k2c_relu, dense_fwork);

```

  But we have replaced all the functions pointers with their function bodies because

function pointers are not supported and were throwing errors.

```
// activation(output->array,output->numel);
    for ( i=0; i < output->numel; ++i) {
        if (output->array[i] <= 0.0f) {
            output->array[i] = 0.0f;
        }
    }
```

```
k2c_dense1(dense_output_array,dense_output_numel,

dropout_output_array,dropout_output_ndim,dropout_output_numel,dropou
t_output_shape,

dense_kernel_array,dense_kernel_ndim,dense_kernel_numel,dense_kernel
_shape,

dense_bias_array,dense_bias_ndim,dense_bias_numel,dense_bias_shape,
            dense_fwork);

```

- **<u>Breaking down the Structure</u>** - Initially a structure named k2c_tensor was being used for inputs and outputs into the functions.

```
struct k2c_tensor
{
    float array[1000000];
    size_t ndim;
    size_t numel;
    size_t shape[K2C_MAX_NDIM];
};
typedef struct k2c_tensor k2c_tensor;
```

```
k2c_tensor c_dense_2_test1 = {{0},1,1,{1,1,1,1,1}};
```

But in order to avoid any Variable sized array errors, we have broken down the structure into individual arrays of fixed sizes and individual variables with fixed values.

```
float c_dense_2_test2_array[1] = {0};
```

```
// k2c_tensor c_dense_2_test2 =
{&c_dense_2_test2_array[0],1,1,{1,1,1,1,1}};
//c_dense_2_test2 tensor uses c_dense_2_test2_array array as its
first parameter
size_t c_dense_2_test2_ndim=1;
size_t c_dense_2_test2_numel=1;
size_t c_dense_2_test2_shape[]={1,1,1,1,1};
```

Also while passing parameters into the functions we have passed each array and variable separately.

```
void k2c_dot(k2c_tensor* C, const k2c_tensor* A, const k2c_tensor*
B, const size_t * axesA,
             const size_t * axesB, const size_t naxes, const int
normalize, float * fwork);
```

---

3. Changes to HLS4ML :

There are two issues encountered while running ***hls_model.build()***.

Problem 1 : The first is a common problem resulting in the error:

```
    ERROR: [HLS 200-101] 'config_schedule': Unknown option
'-enable_dsp_full_reg=false'.

FORMAT
  config_schedule [OPTIONS]
    -effort ( high | medium | low )
    -relax_ii_for_timing
    -verbose
```

Solution : The recommended solution for this is to comment out the existing line (line number 167) in *build_prj.tcl*. This action does not impact the main objective of our project.

167: #config_schedule -enable_dsp_full_reg=false

Problem 2 : The second problem is due to the following errors:

```
ERROR: [XFORM 203-504] Stop unrolling loop 'ReuseLoop:'
(/home/alexanderj13/VLSI/amiya/model_proj/firmware/nnet_utils/nnet_dense_r
esource.h:79) in function 'nnet::conv_1d<ap_fixed<16, 6, (ap_q_mode)5,
(ap_o_mode)3, 0>, ap_fixed<16, 6, (ap_q_mode)5, (ap_o_mode)3, 0>,
config2>' because it may cause large runtime and excessive memory usage
due to an increase in code size. Please avoid unrolling the loop or form
sub-functions for code in the loop body.
ERROR: [HLS 200-70] Pre-synthesis failed.
```

Solution : To address this, we have removed all the pragmas in the given module(Reuse Loop) since it is causing large runtime and excessive memory usage .

---

4. Dependencies and Versions :

1. **python :** 3.10.12
2. **tensorflow :** 2.13.1
   - Requires : h5py, keras, numpy
   - Required By : hls4ml
3. **keras :** 2.13.1
   - Required By : tensorflow
4. **numpy :** 1.24.3
   - Required By : h5py, hls4ml, tensorflow
5. **h5py :** 3.10.0
   - Requires : numpy
   - Required By : hls4ml, tensorflow
6. **hls4ml :** 0.8.1
   - Requires : h5py, numpy, tensorflow

5. Optimizations :

Optimization 1:

Code before optimization:

```
size_t idx = 0;
size_t temp = 0;
for (size_t i=0; i<ndim; ++i) {
    temp = sub[i];
    for (size_t j=ndim-1; j>i; --j) {
        temp *= shape[j];
    }
    idx += temp;
}
return idx;
```

Code after optimization:

```
size_t idx = 0;
size_t temp=1;
for(size_t i=0;i<ndim;++i)
{
    temp*=shape[i];
}
for(size_t j=0;j<ndim;++j)
{
    temp/=shape[j];
    idx=idx+(sub[j]*temp);
}
return idx;
```

Impact: Earlier, there were O(ndim^2) multiplications. After optimization, there are approximately O(ndim) multiplications. Therefore, we reduced the area(of the MULT devices) and the time taken to execute this block of code.

Optimization 2:

Code before optimization:

```
for (i=0; i < naxes; ++i) {
    prod_axesA *= A_shape[axesA[i]];
}
for (i=0; i < naxes; ++i) {
    prod_axesB *= B_shape[axesB[i]];
}
```

Code after optimization:

```
for (i=0; i < naxes; ++i) {
    prod_axesA *= A_shape[axesA[i]];
    prod_axesB *= B_shape[axesB[i]];
}
```

Here we performed loop merging.

Impact: It resulted in improved resource utilization, area and timing.

Optimization 3:

Code before optimization:

```
for (i=ndimA-naxes, j=0; i<ndimA; ++i, ++j) {
    permA[i] = axesA[j];
}
for (i=0; i<naxes; ++i) {
    permB[i] = axesB[i];
}
```

Code after optimization:

```
for (i=ndimA-naxes, j=0; i<ndimA; ++i, ++j) {
    permA[i] = axesA[j];
    permB[j] = axesB[j];
}
```

Again we performed loop merging.
Impact : It resulted in improved resource utilization, area and timing.

Optimization 4:

Code before optimization:

```
void k2c_affine_matmul(float C[], const float A[], const float B[], const float d[],
                       const size_t outrows,const size_t outcols, const size_t innerdim) {

   // make sure output is empty
   //memset(C, 0, outrows*outcols*sizeof(C[0]));
   for (i = 0; i < outrows*outcols; ++i) {
   C[i] = 0;
   }


   for (size_t i = 0 ; i < outrows; ++i) {
      const size_t outrowidx = i*outcols;
      const size_t inneridx = i*innerdim;
      for (size_t j = 0;  j < outcols; ++j) {
         for (size_t k = 0; k < innerdim; ++k) {
            C[outrowidx+j] += A[inneridx+k] * B[k*outcols+j];
         }
         C[outrowidx+j] += d[j];
      }
   }
}
```

Code after optimization:

```
void k2c_affine_matmul(float C[], const float A[], const float B[], const
float d[],
                       const size_t outrows,const size_t outcols, const
size_t innerdim) {

    // make sure output is empty
    //memset(C, 0, outrows*outcols*sizeof(C[0]));
    for (size_t i = 0; i < outrows*outcols; ++i) {
    C[i] = 0;
    }

    size_t ii,jj,kk,temp;
```

```
    for ( i = 0 ; i < outrows/4; ++i) {
        // const size_t outrowidx = i*outcols;
        // const size_t inneridx = i*innerdim;
        for (size_t j = 0;  j < outcols/4; ++j) {
            for (size_t k = 0; k < innerdim/4; ++k) {
                //C[outrowidx+j] += A[inneridx+k] * B[k*outcols+j];
                for(size_t ii=0;ii<4;ii++)
                {
                    const size_t outrowidx = ii*outcols;
                    const size_t inneridx = ii*innerdim;
                    for(size_t jj=0;jj<4;jj++)
                    {
                        temp=outrowidx+jj;
                        float sum=d[jj];
                        for(size_t kk=0;kk<4;kk++)
                        {
                            //#pragma HLS PIPELINE II=1;
                            float A_temp=A[inneridx+kk];
                            float B_temp=B[kk*outcols+jj];
                            sum+=A_temp*B_temp;
                        }
                        C[temp]=sum;
                    }
                }
            }
            // C[outrowidx+j] += d[j];
        }
    }
}
```

Here we performed loop tiling.

Impact:

Old stats:

- **Latency (clock cycles)**
  - **Summary**

Latency		Interval		Type
min	max	min	max	
134792573	1893851609	134792573	1893851609	none

New stats:

- **Timing (ns)**
  - **Summary**

Clock	Target	Estimated	Uncertainty
ap_clk	50.00	33.093	6.25

- **Latency (clock cycles)**
  - **Summary**

Latency		Interval		Type
min	max	min	max	
134775605	1893834641	134775605	1893834641	none

Optimization 5:

Code before optimization:

```
const size_t out_rows = output_shape[0];
    const size_t out_cols = output_shape[1];
    const size_t out_channels = output_shape[2];
    const size_t in_channels = 64;

    for (size_t x0=0; x0 < out_rows; ++x0) {
        for (size_t x1=0; x1 < out_cols; ++x1) {
            for (size_t z0=0; z0 < kernel_shape[0]; ++z0) {
                for (size_t z1=0; z1 < kernel_shape[1]; ++z1) {
                    for (size_t q=0; q < in_channels; ++q) {
                        if(q>=input_shape[2]) break;              //OPTIMIZATION
                        for (size_t k=0; k < out_channels; ++k) {
                            #pragma HLS unroll factor=16
                            output_array[x0*(output_shape[2]*output_shape[1])
                                         + x1*(output_shape[2]) + k] +=
                                kernel_array[z0*(kernel_shape[3]*kernel_shape[2]*kernel_shape[1])
                                             + z1*(kernel_shape[3]*kernel_shape[2])
                                             + q*(kernel_shape[3]) + k]*
                                input_array[(x0*stride[0]
                                             + dilation[0]*z0)*(input_shape[2]*input_shape[1])
                                            + (x1*stride[1] + dilation[1]*z1)*(input_shape[2]) + q];
                        }
                    }
                }
            }
        }
    }
```

Code after optimization:

```
/*APplying array partitioning*/
 #pragma HLS array_partition variable=output_array cyclic factor=16 dim=1
 #pragma HLS array_partition variable=kernel_array cyclic factor=16 dim=1
 #pragma HLS array_partition variable=input_array cyclic factor=16 dim=1
/*ReWriting code to reduce array access*/

size_t k0=kernel_shape[0];
size_t k1=kernel_shape[1];
size_t k2=kernel_shape[2];
size_t k3=kernel_shape[3];
size_t s0=stride[0];
size_t d0 = dilation[0];
size_t i2=input_shape[2];
size_t i1=input_shape[1];
size_t s1=stride[1];
size_t d1=dilation[1];

/*common sub expresion elimination*/
size_t var1=out_channels*out_cols;
size_t var2=k3*k2*k1;
size_t var3=k3*k2;
size_t var4=i2*i1;

for (size_t x0=0; x0 < out_rows; ++x0) {
    for (size_t x1=0; x1 < out_cols; ++x1) {
        for (size_t z0=0; z0 < k0; ++z0) {
            for (size_t z1=0; z1 < k1; ++z1) {
                for (size_t q=0; q < in_channels; ++q) {
                    if(q>=input_shape[2]) break;              //OPTIMIZATION
                    for (size_t k=0; k < out_channels; ++k) {
                        #pragma HLS unroll factor=16
                        output_array[x0*(var1)
                                        + x1*(out_channels) + k] +=
                                            kernel_array[z0*(var2)
                                                            + z1*(var3)
                                                            + q*(k3) + k]*
                                            input_array[(x0*s0
                                                            + d0*z0)*(var4)
                                                           + (x1*s1 + d1*z1)*(i2) + q];
                    }
                }
            }
        }
    }
}
```

Impact:

Old stats:

Latency		Interval		Type
min	max	min	max	
134775605	1893834641	134775605	1893834641	none

New stats:

Latency		Interval		Type
min	max	min	max	
17557987	77631007	17557987	77631007	none

Optimization 6:

```
void k2c_maxpool2d1(float output_array[], size_t output_ndim, size_t output_numel, size_t output_shape[],
                    const float input_array[], const size_t input_ndim, const size_t input_numel, const size_t input_shape[],
                    const size_t pool_size[],
                    const size_t stride[]) {



   #pragma HLS array_partition variable=output_array cyclic factor=16 dim=1
   #pragma HLS array_partition variable=input_array cyclic factor=16 dim=1
   const size_t channels = input_shape[2];
   // i,j,l output indices
   /// i, k, m input indices
   for (size_t i=0; i< channels; ++i) {                //
       for (size_t j=0, k=0; j<output_shape[1]*channels;        //
               j+=channels, k+=channels*stride[1]) {
           for (size_t l=0, m=0; l<output_numel; l+=channels*output_shape[1],
                   m+=channels*input_shape[1]*stride[0]) {
               output_array[l+j+i] = input_array[m+k+i];
               for (size_t n=0; n<pool_size[1]*channels; n+=channels) {
                   for (size_t p=0; p<pool_size[0]*channels*input_shape[1];
                           p+=channels*input_shape[1]) {
                       #pragma HLS unroll factor=8
                       if (output_array[l+j+i] < input_array[m+k+i+n+p]) {
                           output_array[l+j+i] = input_array[m+k+i+n+p];
                       }
                   }
               }
           }
       }
   }
}
```

Here we applied loop unrolling and array partition.

Impact:

Old stats:

Latency		Interval		Type
min	max	min	max	
17557987	77631007	17557987	77631007	none

New stats:

Latency		Interval		Type
min	max	min	max	
16513620	76586640	16513620	76586640	none

Optimization 7:

```
    void k2c_bias_add(float A_array[], size_t A_numel, const float
b_array[], const size_t b_numel) {



        #pragma HLS array_partition variable=A_array cyclic factor=16
dim=1
        #pragma HLS array_partition variable=b_array cyclic factor=16
dim=1
        for (size_t i=0; i<A_numel; i+=b_numel) {         //OPTIMIZATION
            for (size_t j=0; j<b_numel; ++j) {
                #pragma HLS unroll factor=8
                A_array[i+j] += b_array[j];
            }
        }
    }
```

Here, we performed both array partitioning and partial loop unroll.

Impact:

Old stats:

- **Latency (clock cycles)**
  - **Summary**

Latency		Interval		Type
min	max	min	max	
16513620	76586640	16513620	76586640	none

New stats:

- **Latency (clock cycles)**
  - **Summary**

Latency		Interval		Type
min	max	min	max	
16075756	76148776	16075756	76148776	none

Optimization 8:

```
float conv2d_input_array[10000];
#pragma HLS array_partition variable=conv2d_input_array cyclic factor=16 dim=1
#pragma HLS array_partition variable=conv2d_input_input_array cyclic factor=16 dim=1
for(i=0; i<10000; i++)
{
    #pragma HLS unroll factor=16
    conv2d_input_array[i] = conv2d_input_input_array[i];
}
```

Here we are applying array partitioning.

Old stats:

Latency		Interval		Type
min	max	min	max	
16075756	76148776	16075756	76148776	none

New stats:

Latency		Interval		Type
min	max	min	max	
16047006	76120026	16047006	76120026	none

Optimization 9:

```
#pragma HLS array_partition variable=output_array cyclic factor=16 dim=1
    for ( i = 0; i < output_numel; ++i) {
        #pragma HLS unroll factor=16
        output_array[i] = 0;
    }
```

This optimization was done in the k2c_conv2d1().

Impact:

Old stats:

Latency		Interval		Type
min	max	min	max	
16047006	76120026	16047006	76120026	none

New stats:

Latency		Interval		Type
min	max	min	max	
15390244	75463264	15390244	75463264	none

6. Results :

- **Latency and area overhead table for Baseline (Unoptimized)**

Design	LUT	FF		BRAM	Latency	Latency	Clock period

			DSP		(min)	(max)	
cnn__model	16828	6334	44	3172	134792573	1893851609	33.093ns

- **Latency and area overhead table for Optimized (if there are multiple versions like various tradeoffs, give all of them)**

Design	LUT	FF	DSP	BRAM	Latency (min)	Latency (max)	Clock period
cnn__model	114026	56320	366	3186	15390244	75463264	42.905ns

- **HLS4ML generated Latency and area overhead table**

Design	LUT	FF	DSP	BRAM	Latency (min)	Latency (max)	Clock period
cnn_model	1312988	94986	0	1559	88483224	88520074	4.352

- **Finally, comparison report of both Optimized and HLS4ML generated report**

In HLS4ML's report, the latency is a minimum of 88483224 clocks and the clock period is 4.3 ns. In our optimized model, the latency is a minimum of 15390244 clocks and the clock period is 42.905 ns.
Therefore Total time taken in HLS4ML = 38,50,78,990.848 ns
Time taken in our optimized model = 66,03,18,418.82 ns