ASoC Final Project Report

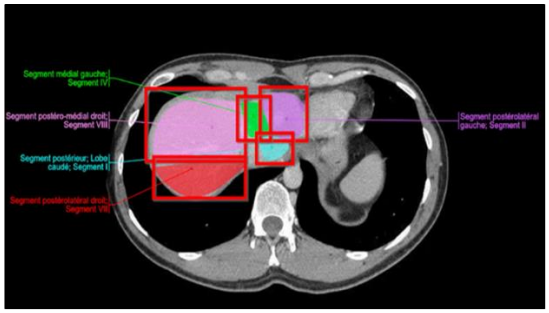
R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

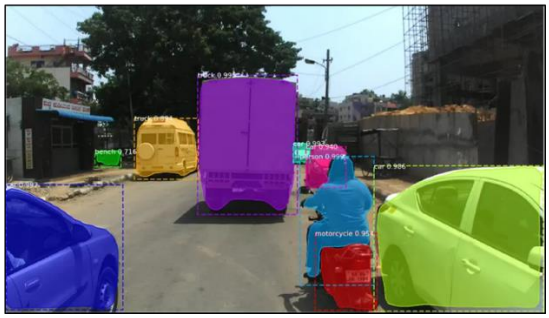
1. Image Semantic Segmentation

Image semantic segmentation is a computer vision task that groups pixels into meaningful or perceptually similar regions. Unlike image classification, which assigns a single label to an entire image, semantic segmentation classifies each pixel in an image into a predefined class (i.e. pixel-level classification). That is, each pixel in the image is assigned a label from a set of predefined categories. For example, in the following image, pixels might be labeled as “table”, “chair”, and “background”.

Applications:

- Medical Imaging: identifying and segmenting anatomical structures or pathological regions in medical scans
- Autonomous Driving: understanding and interpreting the environment by segmenting road scenes
- Agriculture: land use classification, urban planning, and disaster management





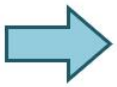
R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

2. Dataset

Pascal VOC 2012 Segmentation Dataset

- Input: RGB image
- Output: Mask with 21 possible class labels for each pixel

Image

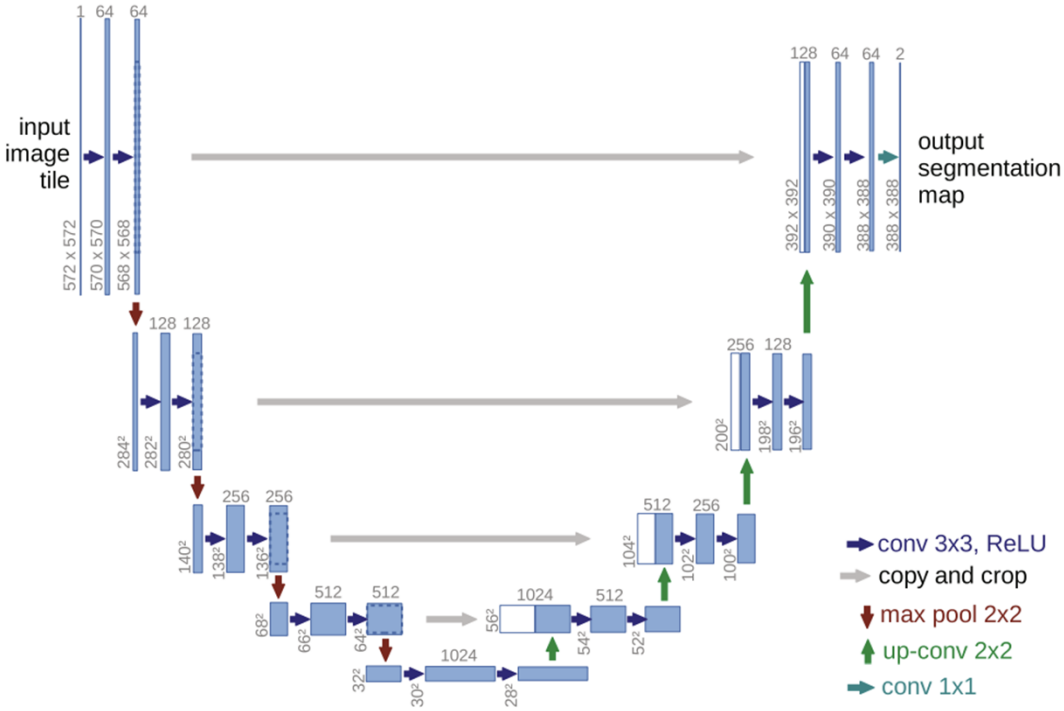


Mask

```
'background',
'aeroplane',
'bicycle',
'bird',
'boat',
'bottle',
'bus',
'car',
'cat',
'chair',
'cow',
'dining table',
'dog',
'horse',
'motorbike',
'person',
'potted plant',
'sheep',
'sofa',
'train',
'tv/monitor'
```

3. U-Net Architecture

U-Net is a popular architecture for semantic segmentation. It consists of a contracting path (encoder) and an expansive path (decoder), with skip connections that combine feature maps from corresponding layers of the encoder and decoder. The key concept of the U-Net architecture is to consider features across different scales.



R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

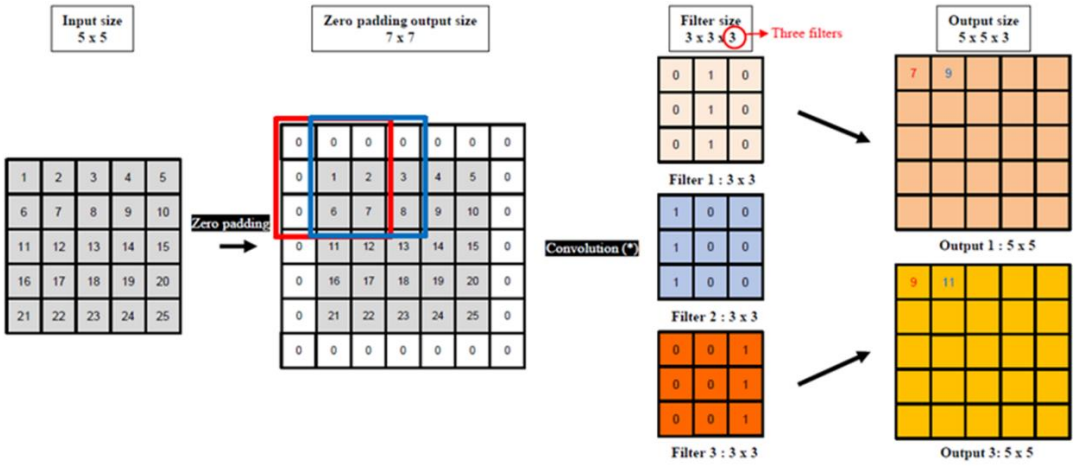
4. Model Training

- Develop and train the model by PyTorch
  - PyTorch is an open-source framework developed by Meta's AI research group
  - It offers an easy-to-use API and integrates seamlessly with the Python data
- Analyze the model structure and implement it
  - Down-sampling
    `nn.Conv2d()`
  - Concatenate path
    `torch.cat()`
  - Up-sampling
    `nn.Upsample()`
    `nn.functional.interpolate()`
    `nn.ConvTranspose2d()`

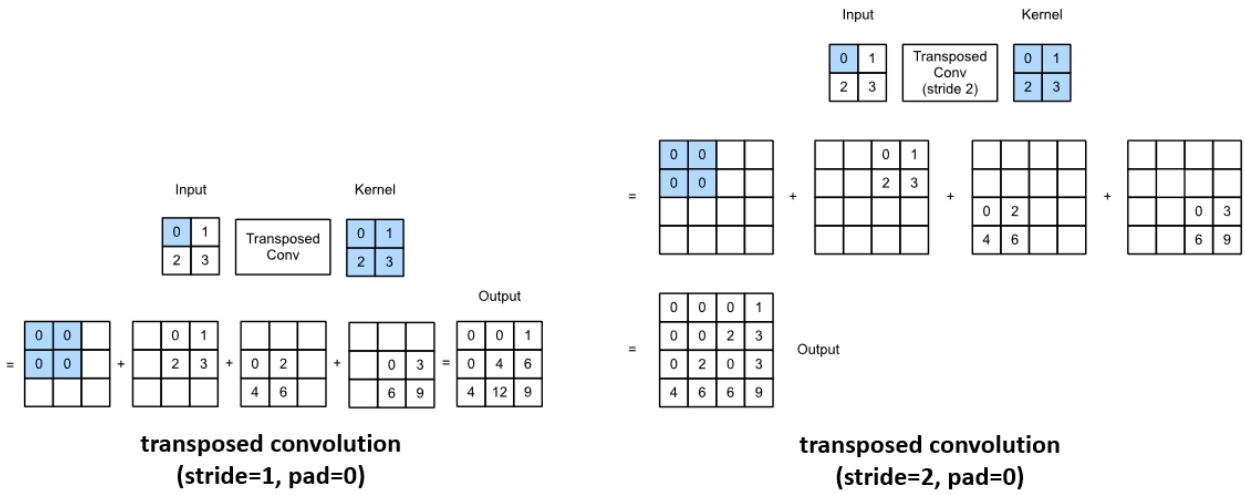
We use transposed convolution for up-sampling because it is a trainable layer.

Operations on U-Net

- Down-sampling: use convolution to extract high-level information



- Up-sampling: use transposed convolution to increase the spatial resolution



R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

Model Profiling

- Parameters: 486,813
- MAC operations: 55,787,520
- Input image size: 3 x 64 x 64
- Output image size: 21 x 64 x 64

```
==========================================================================
Total params: 486,613
Trainable params: 486,613
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 6.84
Params size (MB): 1.86
Estimated Total Size (MB): 8.75
--------------------------------------------------------------------------
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.ConvTranspose2d'>.
macs:55787520.0, params:486613.0
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 8, 64, 64]	216
BatchNorm2d-2	[-1, 8, 64, 64]	16
ReLU-3	[-1, 8, 64, 64]	0
Conv2d-4	[-1, 8, 64, 64]	576
BatchNorm2d-5	[-1, 8, 64, 64]	16
ReLU-6	[-1, 8, 64, 64]	0
Conv2d-7	[-1, 16, 32, 32]	1,152
BatchNorm2d-8	[-1, 16, 32, 32]	32
ReLU-9	[-1, 16, 32, 32]	0
Conv2d-10	[-1, 16, 32, 32]	2,304
BatchNorm2d-11	[-1, 16, 32, 32]	32
ReLU-12	[-1, 16, 32, 32]	0
Conv2d-13	[-1, 32, 16, 16]	4,608
BatchNorm2d-14	[-1, 32, 16, 16]	64
ReLU-15	[-1, 32, 16, 16]	0
Conv2d-16	[-1, 32, 16, 16]	9,216
BatchNorm2d-17	[-1, 32, 16, 16]	64
ReLU-18	[-1, 32, 16, 16]	0
Conv2d-19	[-1, 64, 8, 8]	18,432
BatchNorm2d-20	[-1, 64, 8, 8]	128
ReLU-21	[-1, 64, 8, 8]	0
Conv2d-22	[-1, 64, 8, 8]	36,864
BatchNorm2d-23	[-1, 64, 8, 8]	128
ReLU-24	[-1, 64, 8, 8]	0
Conv2d-25	[-1, 128, 4, 4]	73,728
BatchNorm2d-26	[-1, 128, 4, 4]	256
ReLU-27	[-1, 128, 4, 4]	0
Conv2d-28	[-1, 128, 4, 4]	147,456
BatchNorm2d-29	[-1, 128, 4, 4]	256
ReLU-30	[-1, 128, 4, 4]	0
ConvTranspose2d-31	[-1, 64, 8, 8]	32,768
Conv2d-32	[-1, 64, 8, 8]	73,728
BatchNorm2d-33	[-1, 64, 8, 8]	128
ReLU-34	[-1, 64, 8, 8]	0
Conv2d-35	[-1, 64, 8, 8]	36,864
BatchNorm2d-36	[-1, 64, 8, 8]	128
ReLU-37	[-1, 64, 8, 8]	0
ConvTranspose2d-38	[-1, 32, 16, 16]	8,192
Conv2d-39	[-1, 32, 16, 16]	18,432
BatchNorm2d-40	[-1, 32, 16, 16]	64
ReLU-41	[-1, 32, 16, 16]	0
Conv2d-42	[-1, 32, 16, 16]	9,216
BatchNorm2d-43	[-1, 32, 16, 16]	64
ReLU-44	[-1, 32, 16, 16]	0
ConvTranspose2d-45	[-1, 16, 32, 32]	2,048
Conv2d-46	[-1, 16, 32, 32]	4,608
BatchNorm2d-47	[-1, 16, 32, 32]	32
ReLU-48	[-1, 16, 32, 32]	0
Conv2d-49	[-1, 16, 32, 32]	2,304
BatchNorm2d-50	[-1, 16, 32, 32]	32
ReLU-51	[-1, 16, 32, 32]	0
ConvTranspose2d-52	[-1, 8, 64, 64]	512
Conv2d-53	[-1, 8, 64, 64]	1,152
BatchNorm2d-54	[-1, 8, 64, 64]	16
ReLU-55	[-1, 8, 64, 64]	0
Conv2d-56	[-1, 8, 64, 64]	576
BatchNorm2d-57	[-1, 8, 64, 64]	16
ReLU-58	[-1, 8, 64, 64]	0
Conv2d-59	[-1, 21, 64, 64]	189

R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

Model Training Summary

- Testing accuracy: 94.5%

```
Downloading http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar to ./dataset/VOCtrainval_11-May-2012.tar
100%|██████████████| 1999639040/1999639040 [00:18<00:00, 108398463.22it/s]
Extracting ./dataset/VOCtrainval_11-May-2012.tar to ./dataset
Evaluation Time: 8.15 sec(s), Acc: 0.94508, Loss: 0.67111
```

```python
def forward(self, x):
    # Encoder
    enc1 = self.enc1(x)
    enc2 = self.enc2(F.max_pool2d(enc1, 2))
    enc3 = self.enc3(F.max_pool2d(enc2, 2))
    enc4 = self.enc4(F.max_pool2d(enc3, 2))

    # Bottleneck
    bottleneck = self.bottleneck(F.max_pool2d(enc4, 2))

    # Decoder
    dec4 = self.upconv4(bottleneck)
    dec4 = torch.cat((dec4, enc4), dim=1)
    dec4 = self.dec4(dec4)

    dec3 = self.upconv3(dec4)
    dec3 = torch.cat((dec3, enc3), dim=1)
    dec3 = self.dec3(dec3)

    dec2 = self.upconv2(dec3)
    dec2 = torch.cat((dec2, enc2), dim=1)
    dec2 = self.dec2(dec2)

    dec1 = self.upconv1(dec2)
    dec1 = torch.cat((dec1, enc1), dim=1)
    dec1 = self.dec1(dec1)

    return self.outconv(dec1)
```

```python
super(UNet, self).__init__()
self.channel_size1 = 8
self.channel_size2 = 16
self.channel_size3 = 32
self.channel_size4 = 64
self.channel_size5 = 128
# Encoder
self.enc1 = self.conv_block(in_channels, self.channel_size1)
self.enc2 = self.conv_block(self.channel_size1, self.channel_size2)
self.enc3 = self.conv_block(self.channel_size2, self.channel_size3)
self.enc4 = self.conv_block(self.channel_size3, self.channel_size4)

# Bottleneck
self.bottleneck = self.conv_block(self.channel_size4, self.channel_size5)

# Decoder
self.upconv4 = nn.ConvTranspose2d(self.channel_size5, self.channel_size4, kernel_size=2, stride=2, bias=False)
self.dec4 = self.conv_block(self.channel_size5, self.channel_size4)
self.upconv3 = nn.ConvTranspose2d(self.channel_size4, self.channel_size3, kernel_size=2, stride=2, bias=False)
self.dec3 = self.conv_block(self.channel_size4, self.channel_size3)
self.upconv2 = nn.ConvTranspose2d(self.channel_size3, self.channel_size2, kernel_size=2, stride=2, bias=False)
self.dec2 = self.conv_block(self.channel_size3, self.channel_size2)
self.upconv1 = nn.ConvTranspose2d(self.channel_size2, self.channel_size1, kernel_size=2, stride=2, bias=False)
self.dec1 = self.conv_block(self.channel_size2, self.channel_size1)

# Output
self.outconv = nn.Conv2d(self.channel_size1, out_channels, kernel_size=1)
```

5. Program Translation: from Python to C

- Conv2d() in PyTorch

  `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)`

```c
void conv2d(float* input, float* output, float* filters, int in_channels, int out_channels, int height, int width, int kernel_size, int padding) {
    int padded_height = height + 2 * padding;
    int padded_width = width + 2 * padding;
    int filter_size = kernel_size * kernel_size;

    // Copy input to padded input with padding
    for (int c = 0; c < in_channels; c++) {
        for (int h = 0; h < height; h++) {
            memcpy(
                padded_input + c * padded_height * padded_width + (h + padding) * padded_width + padding,
                input + c * height * width + h * width,
                width * sizeof(float)
            );
        }
    }

    // Initialize output to 0.0
    memset(output, 0, out_channels * height * width * sizeof(float));

    // Define the offset due to the kernel size
    int offset = kernel_size / 2;

    // Apply the convolution
    for (int out_c = 0; out_c < out_channels; out_c++) {
        for (int in_c = 0; in_c < in_channels; in_c++) {
            for (int i = offset; i < height + offset; i++) {
                for (int j = offset; j < width + offset; j++) {
                    for (int x = -offset; x <= offset; x++) {
                        for (int y = -offset; y <= offset; y++) {
                            int in_idx = in_c * padded_height * padded_width + (i + x) * padded_width + (j + y);
                            int filter_idx = out_c * in_channels * filter_size + in_c * filter_size + (x + offset) * kernel_size + (y + offset);
                            int out_idx = out_c * height * width + (i - offset) * width + (j - offset);
                            output[out_idx] += padded_input[in_idx] * filters[filter_idx];
                        }
                    }
                }
            }
        }
    }
}
```

R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

- ConvTranspose2d() in PyTorch

  `torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros', device=None, dtype=None)`

```
void conv_transpose2d(float* input, float* output, float* filters, int in_channels, int out_channels, int in_height, int in_width, int kernel_size, int stride) {
    int out_height = in_height * stride;
    int out_width = in_width * stride;
    int filter_size = kernel_size * kernel_size;

    // Perform the transposed convolution
    for (int out_c = 0; out_c < out_channels; out_c++) {
        for (int in_c = 0; in_c < in_channels; in_c++) {
            for (int i = 0; i < in_height; i++) {
                for (int j = 0; j < in_width; j++) {
                    for (int x = 0; x < kernel_size; x++) {
                        for (int y = 0; y < kernel_size; y++) {
                            int in_idx = in_c * in_height * in_width + i * in_width + j;
                            int filter_idx = out_c * in_channels * filter_size + in_c * filter_size + x * kernel_size + y;
                            int out_i = i * stride + x;
                            int out_j = j * stride + y;
                            int out_idx = out_c * out_height * out_width + out_i * out_width + out_j;
                            output[out_idx] += input[in_idx] * filters[filter_idx];
                        }
                    }
                }
            }
        }
    }
}
```

- ReLU() in PyTorch

  `torch.nn.ReLU(inplace=False)`

- BatchNorm2d in PyTorch

  `torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, device=None, dtype=None)`

```
void batch_norm(float* input, float* output, float* gamma, float* beta, int channels, int height, int width, float epsilon) {
    int num_elements = height * width;

    for (int c = 0; c < channels; c++) {
        float mean = 0.0;
        float var = 0.0;

        // Calculate mean
        for (int i = 0; i < num_elements; i++) {
            mean += input[c * num_elements + i];
        }
        mean /= num_elements;

        // Calculate variance
        for (int i = 0; i < num_elements; i++) {
            var += (input[c * num_elements + i] - mean) * (input[c * num_elements + i] - mean);
        }
        var /= num_elements;

        // Normalize, scale, and shift
        for (int i = 0; i < num_elements; i++) {
            int idx = c * num_elements + i;
            output[idx] = gamma[c] * ((input[idx] - mean) / (float)sqrt((float)(var + epsilon))) + beta[c];
        }
    }
}

void relu(float* input, float* output, int channels, int height, int width) {
    int num_elements = channels * height * width;

    for (int i = 0; i < num_elements; i++) {
        output[i] = (float)fmax((float)0, (float)input[i]);
    }
}
```

R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

6. Hardware Implementation by HLS

Parameter and Data Quantization

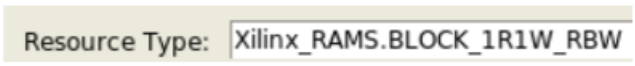
- Original python model use float32 for compute
- Need to quantize as fixed point to reduce hardware complexity and storage
- Parameter: 5 bits, Pixel: 8 bits
- Can reduce 32 bits to 5, 8 bits (fixed point, W5A8)

```
float filerType;
float bufType;
```

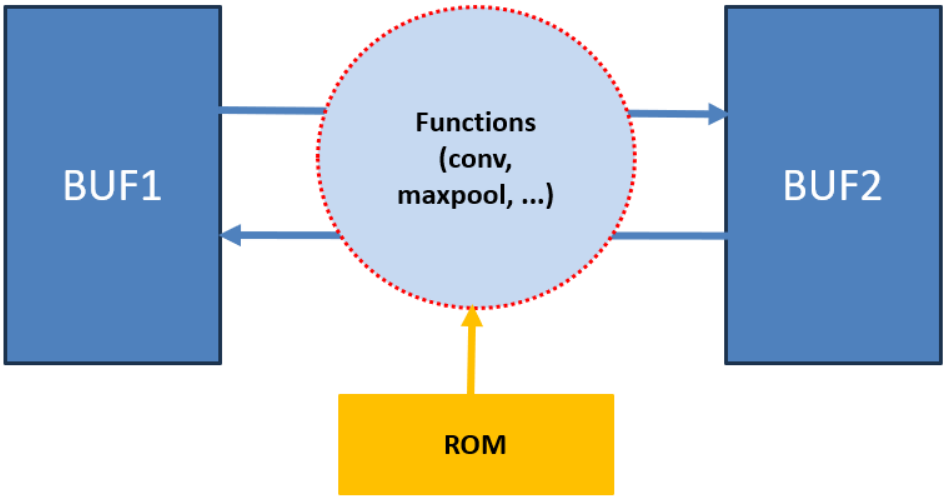
```
typedef ac_fixed<5, 1, true>  filterType;
typedef ac_fixed<8, 6, true>  bufType;
```

HLS Implementation

- Use RAM as data buffer, ROM as parameter storage



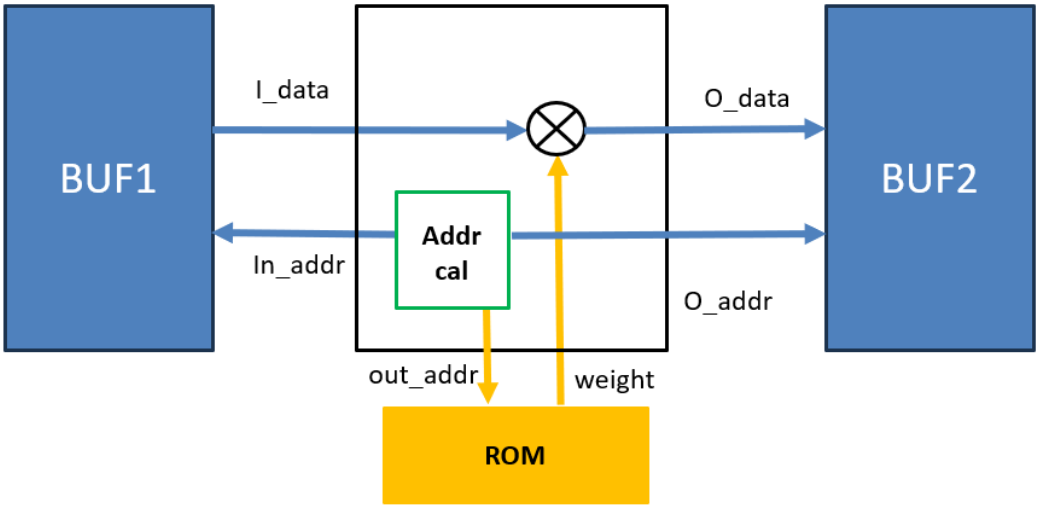
- 2 set of data buffer, ping-pong style



Implementation of Conv2d

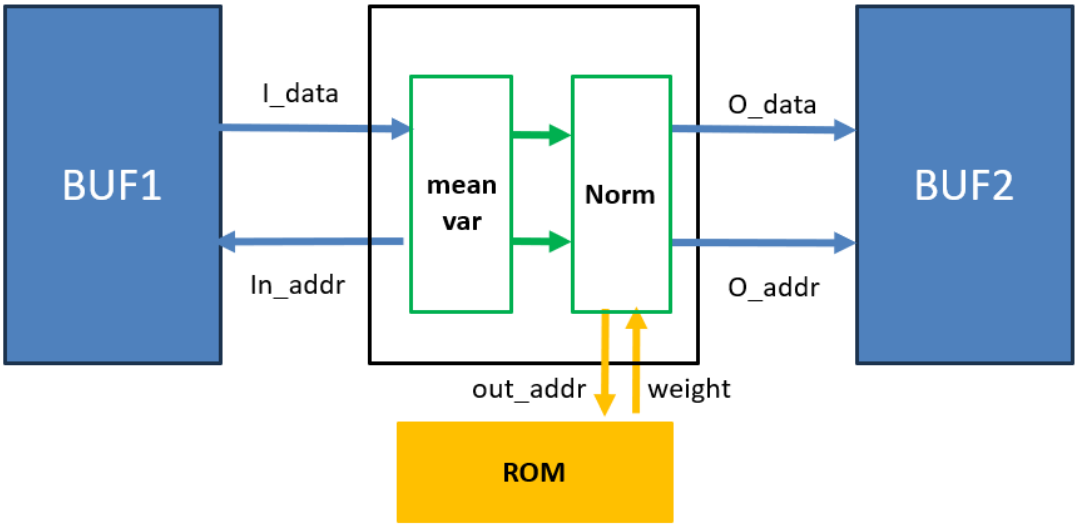
- Minimum hardware resource
- Compute address every cycle, communicate with storage

R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢



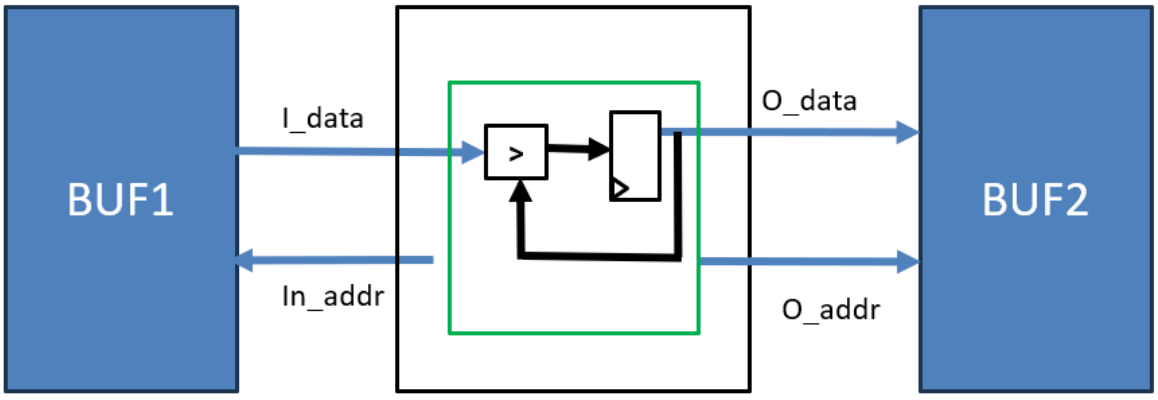
Implementation of Batchnorm

- Compute mean, variance for each channel
- Multiply by gamma and beta



Implementation of Maxpool

- Compare with the candidate max value in the filter



R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢

HLS Implementation Results

- Hardware resources by modules

conv2d	batchnorm	maxpool	transconv	ROM	top
6647	9894	3650	7194	393216	423251

- ROM occupies a large portion
- Reported RAM areas are 0

```
Bill Of Materials (Datapath)
  Component Name                                  Area Score Area(DSP) Area(LUTs) Area(MUX_CARRYs) Delay Post Alloc Post Assign
  ----------------------------------------------- ---------- --------- ---------- --------------- ----- ---------- -----------
  [Lib: Xilinx_RAMS]
  BLOCK_1R1W_RBW(10,12,12,4096,1,4096,12,1)            0.000     0.000      0.000           0.000 0.100          0           1
  BLOCK_1R1W_RBW(4,15,12,32768,1,32768,12,1)           0.000     0.000      0.000           0.000 0.100          0           1
```

7. FSIC Integration

- AXI-Lite configuration register address map

Base Address	Offset	Description
0x3000_0000	0x000	{ap_start, ap_done, ap_idle}
	0x010	{height, width}
	0x020	{kernel_size, padding}
	0x030	{in_channels, out_channels}

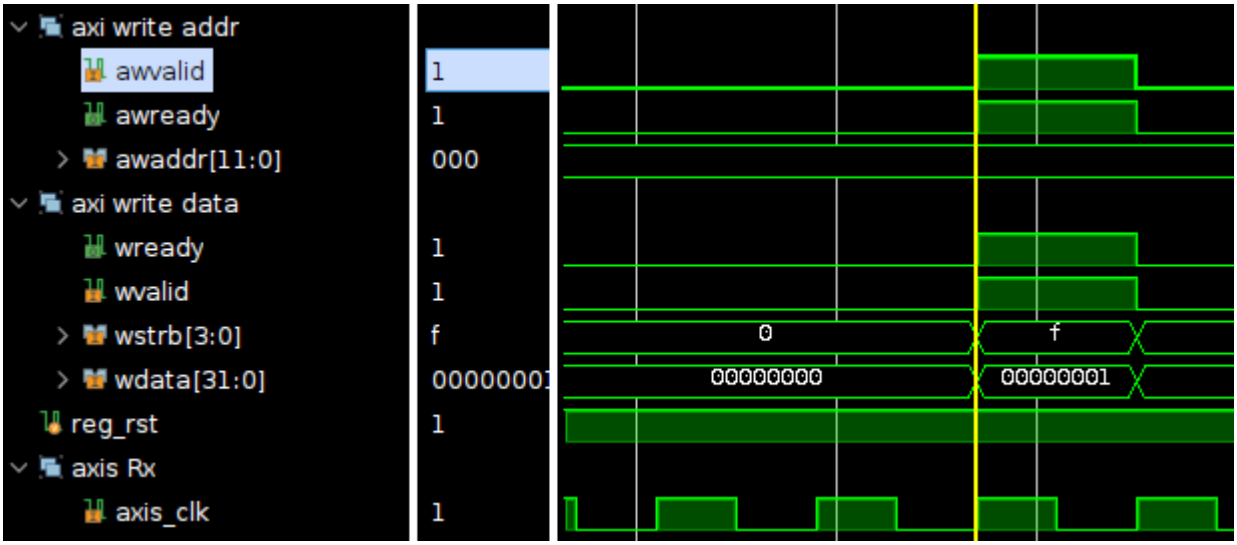
- AXI-Stream
  - Input image to data RAM
  - Output results from data RAM

Integrate a UNET_IP into the user project 0:

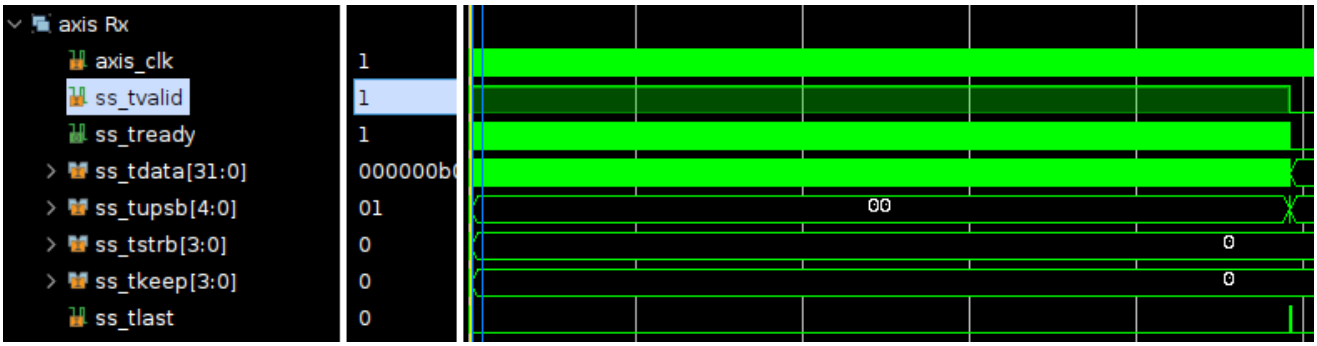
```
UNET_IP_main UNET_IP_main_inst (
    .clk            (axi_clk        ),  // I
    .rst            (reg_rst        ),  // I
    .input_rsc_dat  (input_rsc_dat  ),  // I
    .input_rsc_vld  (input_rsc_vld  ),  // I
    .input_rsc_rdy  (input_rsc_rdy  ),  // O
    .output_rsc_dat (output_rsc_dat ),  // O
    .output_rsc_vld (output_rsc_vld ),  // O
    .output_rsc_rdy (output_rsc_rdy )   // I
);
```

Program ap_start:

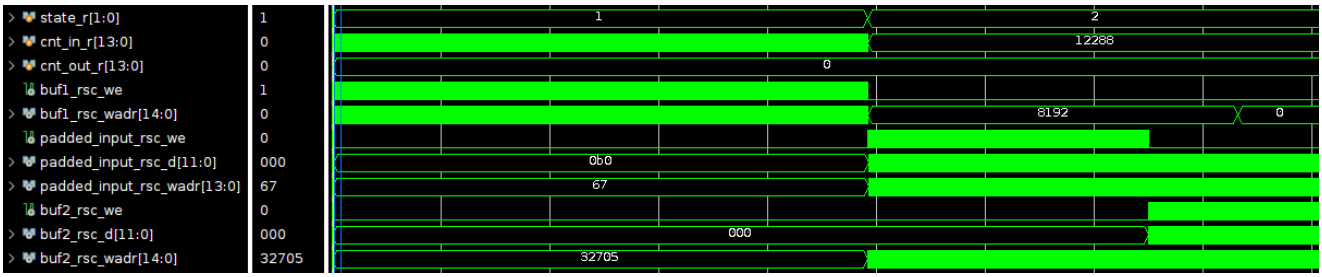
R12943003 劉冠亨　R12943016 謝言鼎　R11943006 張力元　R11525063 李佳螢



Input stream:



Pad input and perform calculation:



8. GitHub Link

https://github.com/alexding1226/ASOC_final