U-Net training and deployment on FPGA

Taehee Jeong*, Elliott Delaye, Paolo D'Alberto, Manasa Bollavaram, Sean Settle

*taeheej@xilinx.com

Abstract

U-Net is a segmentation network for biomedical image. Since it was published in 2015, using this network a lot of application have been produced. We built U-net network and trained it from scratch and successfully deployed on FPGA dpuv1. In this paper, the network architecture and its dataset are reviewed. Then, Data preparation and Procedure to build U-Net architecture and its training procedure are explained. Also, implementing procedure of the network on fpga is described.

1. Introduction

Original U-Net paper was published in 2015[1]. It was developed for segmentation for Biomedical image. The network architecture consists of a down-sampling path and an up-sampling path. Thus, the network shape looks like a U-shape, so the network was named as U-Net. Thereafter, many semantic segmentation applications for Biomedical image have been developed based on U-Net. Therefore, it is worth to implement U-net on Xilinx FPGA.

2. Network Architecture

The U-Net architecture is based on fully convolutional network [2] which is as known as FCN. Like the FCN, U-Net network does not have any fully connected layers. The FCN architecture was modified and extended to work with very few training images and yield more precise segmentations in U-Net [1]. The FCN network is based on VGG16 network. In FCN, the last two fully connected layers were converted to fully convolution layers. There are three up-sampling operations and two combining operations. The final feature map is matched with the original image size.

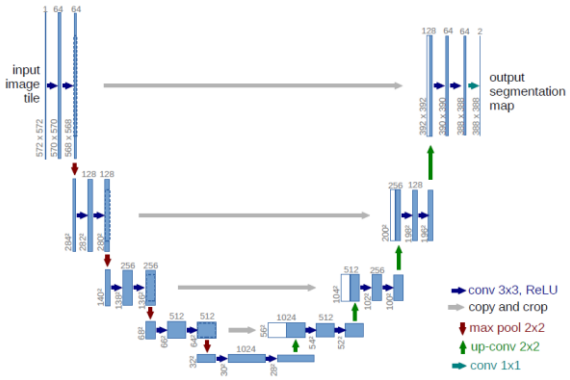

Fig. 1. U-net architecture [1]

The U-net architecture is illustrated in Fig.1. It consists of a down-sampling path (left side) or encoding block and an up-sampling path (right side) or a decoding block. The left side consists of the repeated block of two 3x3 convolutions with each a rectified linear unit (ReLU), and a 2x2 max pooling with stride 2 for down-sampling. These down-sample blocks are repeated 5 times. At each down-sampling block, the number of feature channels are doubled. Also, drop out layers are added at the end of the down-sampling path to prevent overfitting. Meanwhile the right side consists of the repeated block of a 2x2 up-sampling and its following convolution, concatenation with the corresponding feature map from the left side, and two 3x3 convolutions with each a ReLU activation. These up-sampling blocks are repeated 4 times and the final layer is 1 x 1 convolution layer with feature vector to match the number of predicting classes.

The all convolution operation was applied without padding, which is called as 'valid' padding. This made the dimension of every activation layer to be changed every convolution operation in addition to pooling operation.

2.1. Architecture comparison with FCN

The original architecture of U-Net was built using Caffe Framework. The U-Net architecture was based on FCN [2]

architecture. The FCN architecture was based on VGG16 architecture. However, U-Net network removed the last pooling layer (pool5) and two fully connected layers (fc6 and fc7) from VGG16 network, which is different from the FCN. Also, U-Net used concatenate operation instead of 'Add' operation to combine two activation layers. This way, the FCN architecture was modified and extended to work with very few training images and yield more precise segmentations in U-Net [1].

The detailed feature map and its feature size for each layer are illustrated in Fig.2. The one to notice is that the pixel size of the final prediction layer is not matched with the pixel size of the input image. The pixel size of the input image was 572 x 572, but the pixel size of the output is 388 x 388. Also, the pixel size of feature maps to combine between the left side and the right side are not matched each other. Since the pixel size of the feature map does not match each other, crop operations of the left side were added to match the size of feature map from the right side.

layer	image size or feature map	channel or feature size				
input image	572 x 572	1				
block1_conv1	570 x 570	64				
block1_conv2	568 x 568	64		crop1	392 x 392	64
block2_pool	284 x 284	64				
block2_conv1	282 x 282	128				
block2_conv2	280 x 280	128		crop2	200 x 200	128
block3_pool	140 x 140	128				
block3_conv1	138 x 138	256				
block3_conv2	136 x 136	256		crop3	104 x 104	256
block4_pool	68 x 68	256				
block4_conv1	66 x 66	512				
block4_conv2	64 x 64	512		crop4	56 x 56	512
block5_pool	32 x 32	512				
block5_conv1	30 x 30	1024				
block5_conv2	28 x 28	1024				
up6	56 x 56	512				
concat6 (up6 + crop4)	56 x 56	1024				
block6_conv1	54 x 54	512				
block6_conv2	52 x 52	512				
up7	104 x 104	256				
concat7 (up7 + crop3)	104 x 104	512				
block7_conv1	102 x 102	256				
block7_conv2	100 x 100	256				
up8	200 x 200	128				
concat8 (up8 + crop2)	200 x 200	256				
block8_conv1	198 x 198	128				
block8_conv2	196 x 196	128				
up9	392 x 392	64				
concat9 (up9 + crop1)	392 x 392	128				
block9_conv1	390 x 390	64				
block9_conv2	388 x 388	64				
output	388 x 388	2				

Fig. 2. Layer feature map and its pixel size for original U-net architecture. The pixel size of the encoding block and that of the deconding block does not match. To match this, crop operations are included in the encoding block.

2.2. Updated U-Net architecture

We updated U-Net architecture to match the size of feature map of the decoding block and the corresponding encoding block including the final prediction layer without crop operation. For this, every convolution operation in encoding block is added zero padding, which is called as 'same' padding. The up-sampling and its following convolution operation in decoding block are replaced with conv2d_transpose operation with 'same' padding. The input image size is 512 x 512 and the size of its corresponding predicted output is also 512 x 512. The detailed layer information of the updated U-net architecture is illustrated in Fig.3.

layer	image size or feature map	channel or feature size	
input image	512 x 512	3	
block1_conv1	512 x 512	64	
block1_conv2	512 x 512	64	
block2_pool	256 x 256	64	
block2_conv1	256 x 256	128	
block2_conv2	256 x 256	128	
block3_pool	128 x 128	128	
block3_conv1	128 x 128	256	
block3_conv2	128 x 128	256	
block4_pool	64 x 64	256	
block4_conv1	64 x 64	512	
block4_conv2	64 x 64	512	
block5_pool	32 x 32	512	
block5_conv1	32 x 32	1024	
block5_conv2	32 x 32	1024	
conv2d_transpose_1	64 x 64	512	
concatenate_1	64 x 64	1024	<-- combine conv2d_transpose1 and block4_conv2
block6_conv1	64 x 64	512	
block6_conv2	64 x 64	512	
conv2d_transpose_2	128 x 128	256	
concatenate_2	128 x 128	512	<-- combine conv2d_transpose2 and block3_conv2
block7_conv1	128 x 128	256	
block7_conv2	128 x 128	256	
conv2d_transpose_3	256 x 256	128	
concatenate_3	256 x 256	256	<-- combine conv2d_transpose3 and block2_conv2
block8_conv1	256 x 256	128	
block8_conv2	256 x 256	128	
conv2d_transpose_4	512 x 512	64	
concatenate_4	512 x 512	128	<-- combine conv2d_transpose4 and block1_conv2
block9_conv1	512 x 512	64	
block9_conv2	512 x 512	64	
output	512 x 512	21	

Fig. 3. Layer feature map and its pixel size for modified U-net architecture. The pixel size of the encoding block and that of the decoding block match each other. There is no need of crop operation.

In original U-Net network, dropout layers were attached after block4_conv2 and block5_conv2 layer to prevent overfitting. In the updated U-Net network, we experimented to apply dropout layer in many different places both in encoding and decoding block during training. But the dropout layers are removed for inference.

3. Dataset

The original paper experimented three different data sets. One was a set of serial section transmission electron microscopy of the Drosophila first instar larva ventral nerve cord (VNC) from EM segmentation challenge [3]. The other two datasets were light microscopic images. The first dataset was PhC-U373 which contains Glioblastoma-astrocytoma U373 cells on a polyacrylamide substrate recorded by phase contrast microscopy. The

second dataset was DIC-HeLa which are HeLa cells on a flat glass recorded by differential interference contrast (DIC) microscopy. Among these three datasets, we chose PhC-U373 dataset to train our U-Net network.

3.1. PhC-U373 dataset

Glioblastoma-astrocytoma U373 cells on a polyacrylamide substrate were provided by Dr. S. Kumar. Department of Bioengineering, University of California at Berkeley, Berkeley CA (USA). The dataset can be downloaded from http://celltrackingchallenge.net/2d-datasets/. The dataset has two subsets: 01 and 02. The original paper and we used 01 subset. The training dataset has 115 light microscopic images and its segmentation maps.

3.2. Data processing

The segmentation image is TIFF format and two-dimensional pixel array with the class of [0, 1, 2, 3, 4, 5, 6, 7, 8]. The '0' class is background. The photo image is also TIFF format and gray-scale image. The size of the photo image and its corresponding segmentation maps were resized as (256, 256) or (512, 512).

4. Training and its results

The input images and their corresponding segmentation maps are used to train the network with the Adam optimizer with a learning rate as 0.0001 and activation function as ReLU, and the number of epochs as 50.

From the total 115 images, we randomly split the dataset 80/35 for train/validation dataset. The scale of the photo image is set (0,1) and one channel (grayscale image) as the input shape. We experimented to use three channels (RGB scale image) as the input shape for training, but the result was not good as that of one channel. Since dpuv1 accepts only three channels as input shape, the input shape and its corresponding weight were updated after training.

For loss function, the cross entropy loss function was used. The kernel weights were initialized as 'he normal'. The cross entropy was calculated per pixel with 9 classes. For this, we trained each image once at a time instead of the batch training. Also, the output shape of segmentation maps was changed to the shape of (256*256,9) and (512*512,9) instead of (256,256,9) and (512, 512, 9).

The metric to evaluate the model performance was mean IOU (intersection over union) for each pixel and class. To improve mean IOU for validation dataset, we tried to apply drop out layers after all or certain convolutional layers such as block4_conv2 and block5_conv2 layer. However, it turned out that the drop out layers did not help to improve mean IOU for validation dataset. Also, we experimented to use data augmentation methods such as mirroring horizontally or vertically, zooming, and elastic deformation (change the aspect ratio of image). Among the augmentation methods, only mirroring helps a little bit to improve mean IOU for validation dataset.

TABLE I. SEGMENTATION RESULTS (IOU) ON THE PHC-U373/01

Image size	***Mean IOU for train dataset***	***Mean IOU for validation dataset***
Original paper		0.9203
(256,256)	0.9650	0.9370
(512,512)	0.9740	0.9220

Table 1 shows the segmentation results (mean IOU) on PhC-C2DH-U373/01. The validation mean IOU of our result of (256,256) input size is slightly higher than that of the original paper.

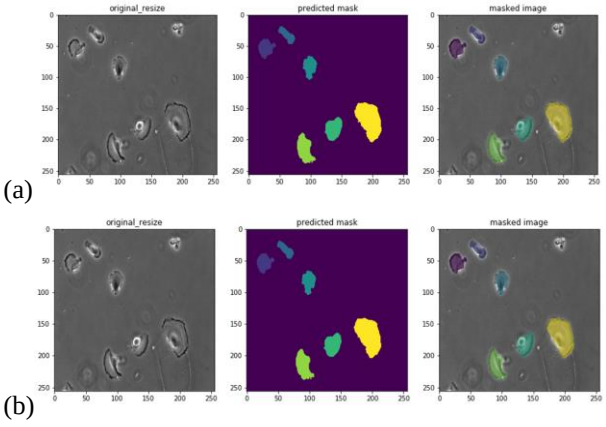

Fig. 4. The result of U-Net on the PhC-U373 as the size of input image as (256,256). (a) sample output from float model on CPU (b) sample output from quantized model on FPGA

5. Inference

We built and trained U-Net model using keras Framework. Then, we converted the model Caffe Framework since currently we cannot directly execute keras model on FPGA. Then, we quantized, compiled, and partitioned the model.

The detail of the tutorial is available in

/workspace/alveo/apps/U-Net/README.md.

The sample outputs from the models are illustrated in Fig.4 and 5.

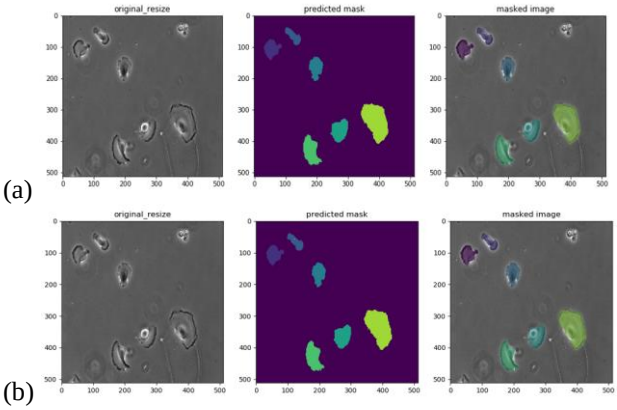

Fig. 5. The result of U-Net on the PhC-U373 as the size of input image as (512,512). (a) sample output from float model on CPU (b) sample output from quantized model on FPGA

5.1. Converting keras model to Caffe model

After we got the trained keras model, the next step was to convert the model to Caffe model. We used *xfdnn_compiler_keras.py*, which needs TensorFlow, Keras, Caffe packages. You need to install Caffe in vitis-ai-tensorflow conda environment. Instead, you might use another keras-to-caffe converter which are available on internet or train the model directly on Caffe framework. After converting, we verified the Caffe model to get same mean IOU over training and validation dataset on CPU.

5.2. Quantization

We kept the Caffe model at 'float' sub folder. We quantized the model using the following command.

```
DECENT_DEBUG=1 vai_q_caffe quantize -model
./float/dummy.prototxt -weights ./float/unet.caffemodel
-input_blob "input_1" -method 1 -calib_iter 100
```

After 100 iteration of calibrating input images, *quantize_info.txt* , *deploy.caffemodel*, and *deploy.prototxt* would be generated at 'quantize_results' sub folder. In addition, quantize_train_test.prototxt and quantize_train_test.caffemodel would be generated in 'quantize_results' sub folder. These files are a kind of simulation version of quantization which can be exectued on cpu. The vai_q_caffe file is included in vitis-ai-caffe conda environment. '-method' is the method for quantization, 0: Non-Overflow, 1: Min-Diff. default is 1. The dummy.prototxt is a copy of the float version of prototxt. The input block of the dummy.prototxt was updated as the following. Comment the input block and created the uncomment block. 'U373_list.txt' is the text file with the list of the file name.

```
#layer {
#  name: "input_1"
#  type: "Input"
#  top: "input_1"
#  input_param {
#    shape {
#      dim: 1
#      dim: 3
#      dim: 256
#      dim: 256
#    }
#  }
#}
layer {
  name: "input_1"
  type: "ImageData"
  top: "input_2"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.0078431
    crop_size: 256

  }
  image_data_param {
    source: "./PhC-C2DH-U373/U373_list.txt"
    batch_size: 1
    shuffle: false
    root_folder: ". /PhC-C2DH-U373/train/Img/"
  }
}
```

5.3. Compilation and Partitioning

We compiled and partitioned the quantized caffemodel on vitis-ai-caffe conda environment using the following commands.

```
vai_c_caffe \
  --net_name deploy.compiler \
  --prototxt ./quantize_results/deploy.prototxt \
  --caffemodel ./quantize_results/deploy.caffemodel \
```

```
  --options "{'quant_cfgfile': './quantize_info.txt'}" \
  --arch
/opt/vitis_ai/compiler/arch/dpuv1/ALVEO/ALVEO.json
\
  -o ./work

python -m
vai.dpuv1.rt.scripts.framework.caffe.xfdnn_subgraph \
  --inproto ./quantize_results/deploy.prototxt \
  --outproto ./xfdnn_deploy.prototxt \
  --cutAfter input_1 \
  --xclbin /opt/xilinx/overlaybins/xdnnv3 \
  --netcfg ./work/compiler.json \
  --quantizecfg ./work/quantizer.json \
  --weights ./work/weights.h5
```

After compilation, all compiler files would be generated in 'work' sub folder. After partitioning, xfdnn_deploy.prototxt would be generated at root folder. We deployed the xfdnn_deploy.prototxt instead of deploy.prototxt to execute the deploy.caffemodel on FPGA.

6. Conclusion

We successfully developed U-Net model and deployed on FPGA. All codes are available at /workspace/alveo/apps/U-Net.

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation (2015), arXiv:1505.04597

[2] Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation (2014), arXiv:1411.4038.

[3] Web page of the em segmentation challenge, http://brainiac2.mit.edu/isbi_challenge/