CrossLink-NX Scene Segmentation Reference Design

Reference Design

FPGA-RD-02256-1.0

June 2022

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CKPT	Checkpoint
CNN	Convolutional Neural Network
EVDK	Embedded Vision Development Kit
FPGA	Field-Programmable Gate Array
LED	Light-emitting diode
MLE	Machine Learning Engine
NN	Neural Network
NNC	Neural Network Compiler
SD	Secure Digital
SDHC	Secure Digital High Capacity
SDXC	Secure Digital extended Capacity
SPI	Serial Peripheral Interface
VIP	Video Interface Platform
USB	Universal Serial Bus
VVML	Voice & Vision Machine Learning Board

1. Introduction

This document describes the Semantic Segmentation design process using Crosslink-NX Voice and Vision Board. As application this document will refer to background blurring demo.

1.1. Design Process Overview

The design process involves the following steps:

1. Training the model
   - Setting up the basic environment.
   - Preparing the dataset.
   - Training the machine.
     - Training the machine and creating the checkpoint data
   - Creating the frozen file (*.pb)
2. Compiling Neural Network: Creating the filter and firmware binary files with Lattice SensAI 5.0 program.
3. FPGA design: Creating the FPGA Bit stream file.
4. FPGA Bit stream and Quantized Weights and Instructions: Flashing the binary and bit stream files to Crosslink-NX VnV hardware.

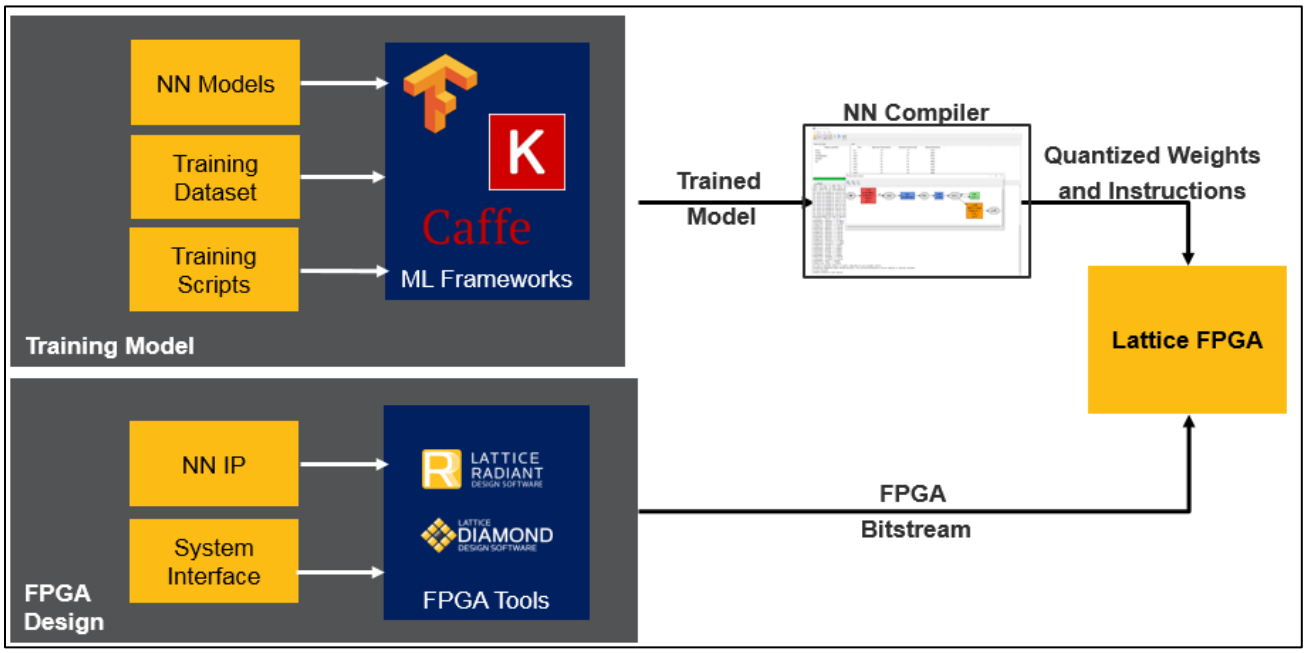


Figure 1.1. Lattice Machine Learning Design Flow

2. Setting Up the Basic Environment

2.1. Tools and Hardware Requirements

This section describes the required tools and environment setup for training and model freezing.

2.1.1. Lattice Tools

- Lattice Radiant Tool v3.1.1 – Refer to http://www.latticesemi.com/latticeradiant.
- Lattice Radiant Programmer v3.1.1 – Refer to http://www.latticesemi.com/latticeradiant.
- Lattice SensAI Compiler v 5.0 – Refer to https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler.

2.1.2. Hardware

This design uses the Crosslink-NX Voice and Vision board as shown in **Figure 2.1**.

- CrossLink-NX Voice and Vision Machine Learning (VVML) Board, Rev B Board

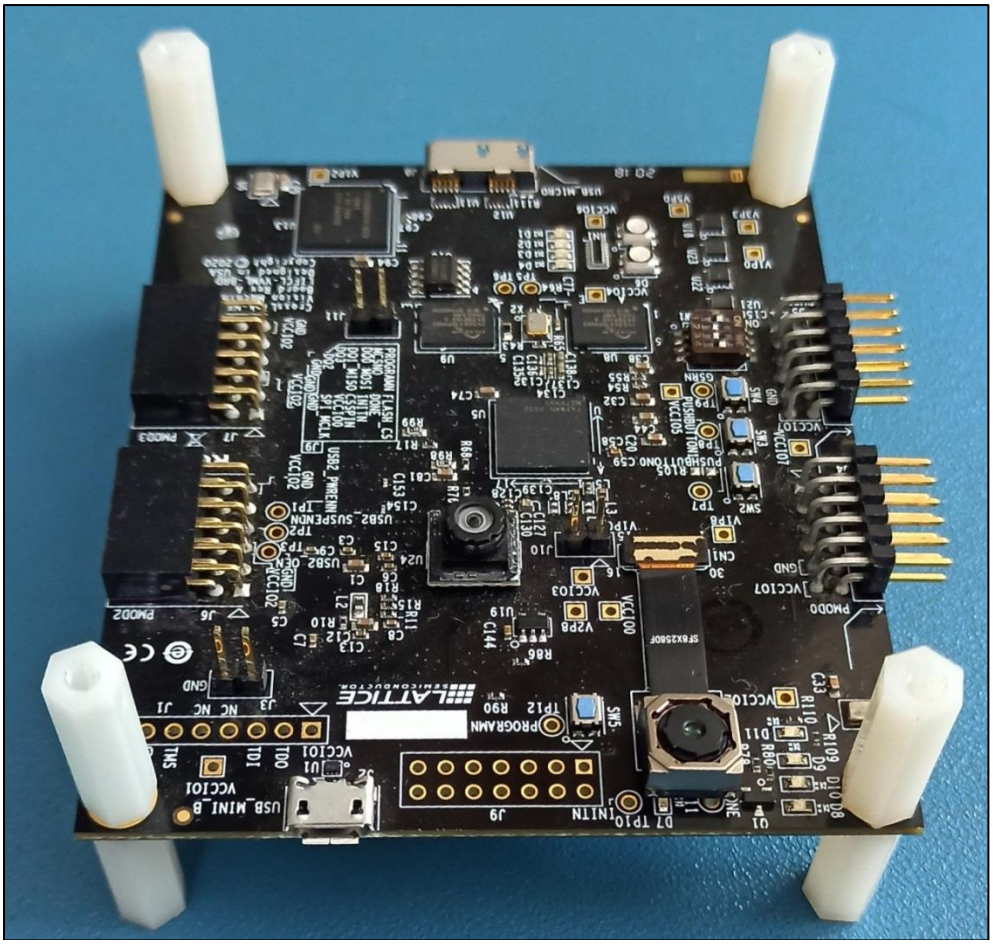


Figure 2.1. Lattice CrossLink-NX Voice and Vision Machine Learning (VVML) Board, Rev B

2.2. Setting Up the Linux Environment for Machine Training

This section describes the steps for NVIDIA GPU drivers and/or libraries for 64-bit Ubuntu 16.04 OS. NVIDIA library and TensorFlow version is dependent on PC and Ubuntu/Windows version.

2.2.1. Installing the NVIDIA CUDA and cuDNN Library for Machine Learning Training on GPU

2.2.1.1. Installing the CUDA Toolkit

To install the CUDA toolkit, run the following commands in the order specified below:

```
$ curl -O
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-
repo-ubuntu1604_10.1.105-1_amd64.deb
```

```
$ curl -O https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2832  100  2832    0     0   2204      0  0:00:01  0:00:01 --:--:--  2205
```

Figure 2.2. Download CUDA Repo

```
$ sudo dpkg –I ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
```

```
$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
Selecting previously unselected package cuda-repo-ubuntu1604.
(Reading database ... 5287 files and directories currently installed.)
Preparing to unpack .../cuda-repo-ubuntu1604_10.1.105-1_amd64.deb ...
Unpacking cuda-repo-ubuntu1604 (10.1.105-1) ...
Setting up cuda-repo-ubuntu1604 (10.1.105-1) ...

The public CUDA GPG key does not appear to be installed.
To install the key, run this command:
sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
```

Figure 2.3. Install CUDA Repo

```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.
pub
```

```
$ sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
Executing: /tmp/tmp.a2QZZnTMUX/gpg.1.sh --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
gpg: key 7FA2AF80: public key "cudatools <cudatools@nvidia.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Figure 2.4. Fetch Keys

```
$sudo apt-get update
```

```
$ sudo apt-get update
Ign:1 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Release [697 B]
Get:4 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Release.gpg [836 B]
Hit:5 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:6 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:7 http://archive.ubuntu.com/ubuntu xenial-backports InRelease
Ign:8 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Packages
Get:8 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Packages [428 kB]
Fetched 429 kB in 1s (386 kB/s)
Reading package lists... Done
```

Figure 2.5. Update Ubuntu Packages Repositories

```
$ sudo apt-get install cuda-9-0
```

```
$ sudo apt-get install cuda-9-0
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 2.6. CUDA Installation

2.2.1.2. Installing the cuDNN

To install the cuDNN:

1. Create Nvidia developer account: https://developer.nvidia.com.
2. Download cuDNN lib: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.1.4/prod/9.0_20180516/cudnn-9.0-linux-x64-v7.1
3. Execute below commands to install cuDNN

```
$ tar xvfcudnn-9.0-linux-x64-v7.1.tgz
$ sudo cp cuda/include/cudnn.h/usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
```

4. $ sudochmoda+r/usr/local/cuda/include/cudnn.h/usr/local/cuda/lib64/libcudnn*

```
$ tar xcf cudnn-9.0-linux-x64-v7.1.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.1.4
cuda/lib64/libcudnn_static.a

$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Figure 2.7.cuDNN Library Installation

2.2.2. Setting Up the Environment for Training and Model Freezing Scripts

This section describes the environment setup information for training and model freezing scripts for 64-bit Ubuntu 16.04. Anaconda provides one of the easiest ways to perform machine learning development and training on Linux.

2.2.2.1. Installing the Anaconda Python

To install the Anaconda and Python 3:

1. Go to https://www.anaconda.com/products/individual#download-section
2. Download Python3 version of Anaconda for Linux.
3. Run the command below to install the Anaconda environment:

```
$   sh   Anaconda3-2019.03-Linux-x86_64.sh
```

Note: Anaconda3-**<version>**-Linux-x86_64.sh, version may vary based on the release

```
(base) $ sh Anaconda3-2020.07-Linux-x86_64.sh

Welcome to Anaconda3 2020.07

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

Figure 2.8.Anaconda Installation

4. Accept the license.

```
Do you accept the license terms? [yes|no]
[no] >>> yes
```

Figure 2.9.Accept License Terms

5. Confirm the installation path, follow the instruction on screen if you want to change the default path.

```
[no] >>> yes


Anaconda3 will now be installed into this location:
/home/user/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/user/anaconda3] >>> /home/user/anaconda3
```

Figure 2.10.Confirm/Edit Installation Location

6. After installation, enter No as shown in Figure 2.11.

```
Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>>  no
```

Figure 2.11.Launch/Initialize Anaconda Environment on Installation Completion

2.2.3. Creating new environment with python 3.7

1. Activate base conda environment using below command

```
$ source <conda directory>/bin/activate
```

2. To create new environment run below command

```
$ conda create -n <Name of New environment> python=3.7
```

3. Activate newly created environment

$ conda activate <Name of New environment>

2.2.4. Installing the TensorFlow v1.15

1. Install the TensorFlow by running the command below:

```
$ conda install tensorflow-gpu==1.15
```

```
$ conda install tensorflow-gpu=1.15
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/anaconda3.1/envs/tf1.15-enet

  added / updated specs:
    - tensorflow-gpu=1.15
```

Figure 2.12.TensorFlow Installation

2. After installation, enter **Y** as shown in Figure 2.13.

```
Proceed ([y]/n)? y


Downloading and Extracting Packages
tensorflow-1.15.0     | 4 KB      | ################################### | 100%
cudatoolkit-10.0.130  | 261.2 MB  | ################################### | 100%
tensorboard-1.15.0    | 3.2 MB    | ################################### | 100%
gast-0.2.2            | 155 KB    | ################################### | 100%
cudnn-7.6.5           | 165.0 MB  | ################################### | 100%
numpy-1.21.5          | 10 KB     | ################################### | 100%
numpy-base-1.21.5     | 4.8 MB    | ################################### | 100%
tensorflow-base-1.15  | 156.5 MB  | ################################### | 100%
tensorflow-estimator  | 271 KB    | ################################### | 100%
typing_extensions-4.  | 28 KB     | ################################### | 100%
six-1.16.0            | 18 KB     | ################################### | 100%
cupti-10.0.130        | 1.5 MB    | ################################### | 100%
webencodings-0.5.1    | 19 KB     | ################################### | 100%
werkzeug-0.16.1       | 258 KB    | ################################### | 100%
importlib-metadata-4  | 40 KB     | ################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Figure 2.13.TensorFlow Installation Confirmation

2.2.5. Installing the Python Package

To install the Python package:

1. Install OpenCV by running the command below:

```
$ conda install -c menpo opencv
```

```
$ conda install -c menpo opencv
Collecting package metadata (current_repodata.json): done
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/anaconda3.1/envs/tf1.15-enet

  added / updated specs:
    - opencv
```

Figure 2.14.OpenCV Installation

2. Install Pillow by running the command below:

```
$ conda install pillow
```

```
$ conda install pillow
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/anaconda3.1/envs/tf1.15-enet

  added / updated specs:
    - pillow
```

Figure 2.15.Pillow Installation

3. Install tqdm by running the command below:

```
$ conda install tqdm
```

```
$ conda install tqdm
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/anaconda3.1/envs/tf1.15-enet

  added / updated specs:
    - tqdm
```

Figure 2.16.Tqdm Installation

4. Install scipy by running the command below:

```
$ conda install scipy=1.1.0
```

```
$ conda install scipy=1.1.0
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/anaconda3.1/envs/tf1.15-enet

  added / updated specs:
    - scipy=1.1.0
```

Figure 2.17. Scipy Installation

5. Install matplotlib by running the command below:

```
$ conda install matplotlib
```

```
$ conda install matplotlib
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/anaconda3.1/envs/tf1.15-enet

  added / updated specs:
    - matplotlib
```

Figure 2.18. Matplotlib Installation

3. Code Structure

Download the Lattice Background Blurring demo training code. Its directory structure should look like:

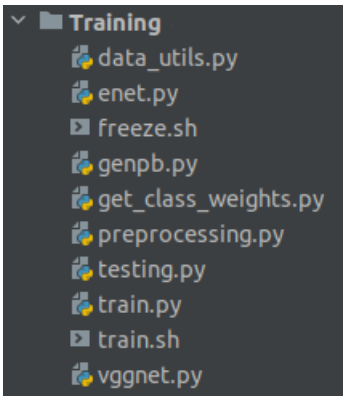


Figure 3.1. Training Code Directory Structure

4. Dataset Preparation

This section describes steps & guidelines used to prepare dataset to train the Background Blurring demo for CNX-VNV. Please note that this section is for the example reference. With following sections, Lattice is just providing the guidelines and/or example which can be used as reference for preparing dataset for given use cases but in no case, Lattice is recommending and/or endorsing any dataset(s). Lattice strongly recommends customers to gather and prepare their own datasets for their end applications.

4.1. Downloading the Dataset

To download the dataset, run the commands below.

In this demo document we are using "https://www.kaggle.com/datasets/laurentmih/aisegmentcom-matting-human-datasets" dataset as reference. Download Dataset from kaggle from link Download Dataset and extract the data.

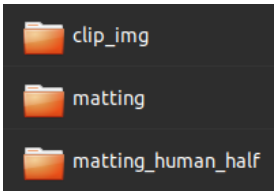


Figure 4.1. Matting dataset Directory Structure

Figure 4.2. Matting dataset sample image and ground truth

4.2. Convert Matting dataset to segmentation dataset

Convert matting-human dataset to segmentation images and ground truth using below command.

```
$ python data_utils.py --mode Generate --input_data <Matting dataset Path> --
output_Data <Output converted dataset>
```

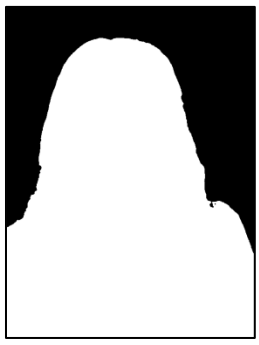


Figure 4.3. Converted matting GT to segmentation GT

Note: The image shown above is for visualization. Actual image will have pixel values of 0 and 1 rather than 0 and 255.

4.3. Dataset Augmentation

python data_utils.py --mode Augment --input_data <converted dataset path> --output_Data <Output augmented dataset> --background_data <background images path> --output_dim 160

Figure 4.4. Augmentation sample images

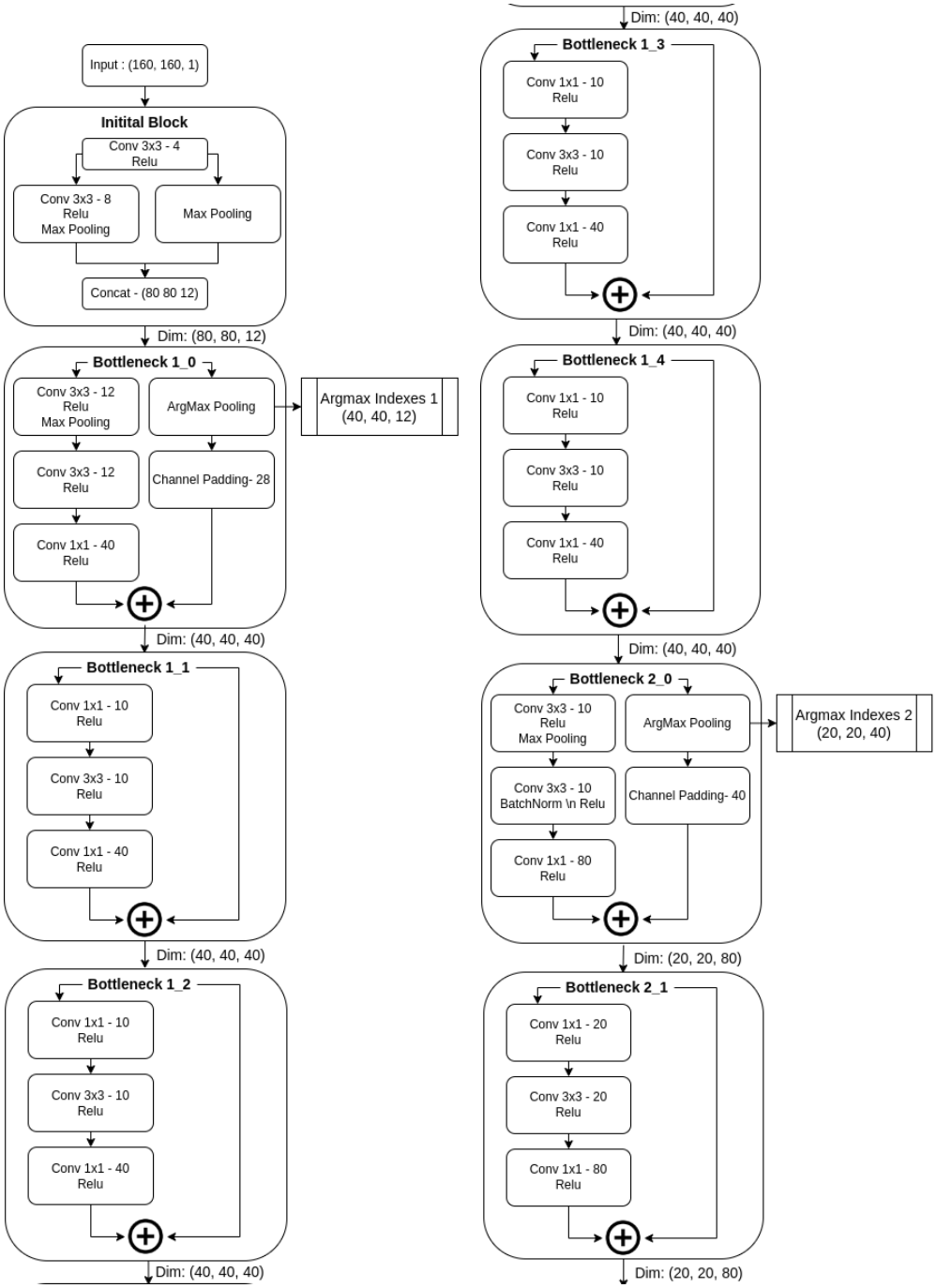
Note: background images path should contain multiple images of sample background as shown in above figure.

5. Training Code Preparation

5.1. Neural Network Architecture

5.1.1. Neural Network Architecture

This section provides information on the Convolution Neural Network Configuration of the Background Blurring design.



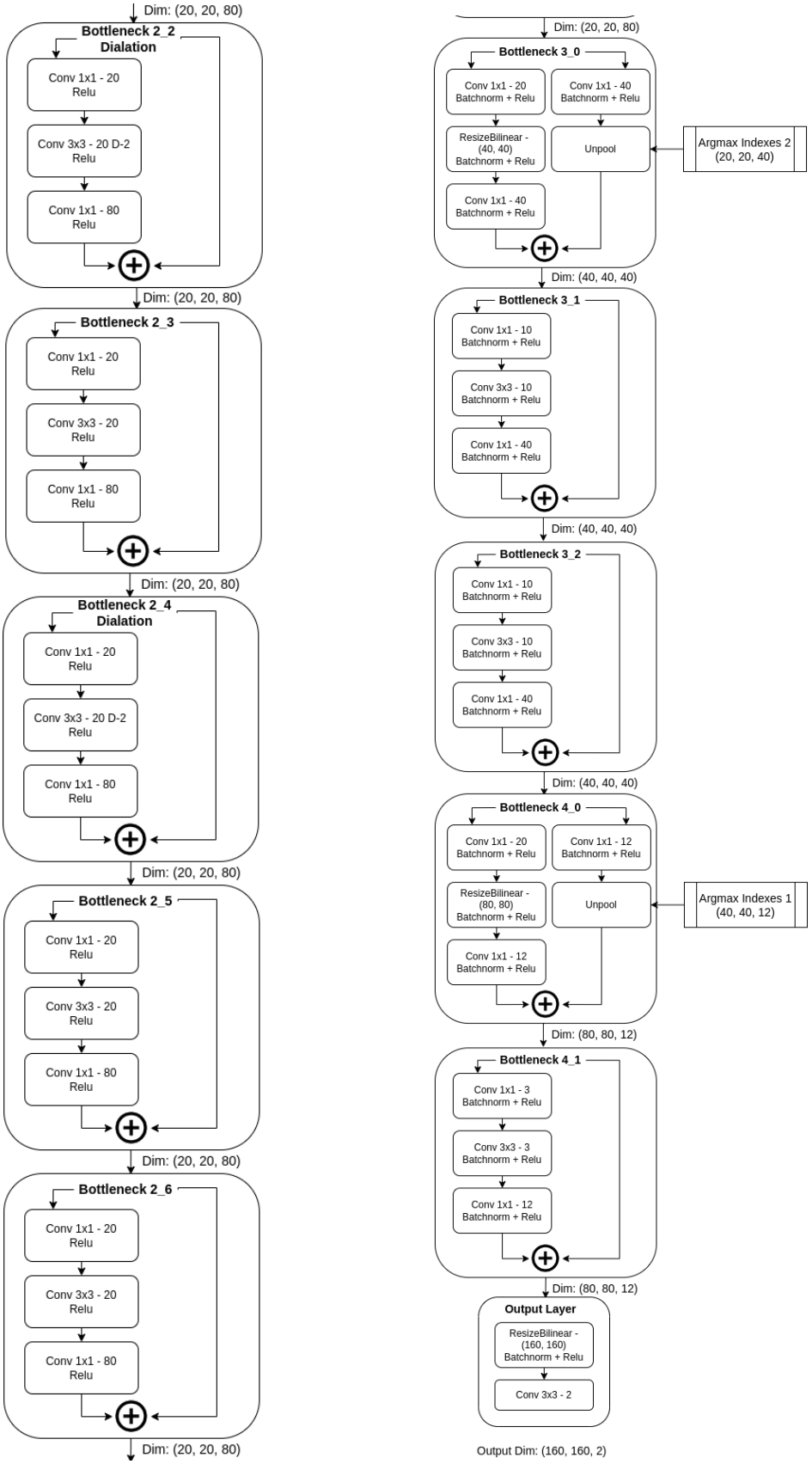


Figure 5.1. Enet Architecture

In above diagram model contains Convolution (conv), batch normalization (bn), Relu, Resize Bilinear, Argmaxpooling, unpooling, maxpooling and channel padding layers.

- Layer information
  - Convolutional Layer

    In general, the first layer in a CNN is always a convolutional layer**.** Each layer consists of number of filters (sometimes referred as kernels) which convolves with input layer/image and generates activation map (I.e. feature map). This filter is an array of numbers (the numbers are called weights or parameters). Each of these filters can be thought of as feature identifiers, like straight edges, simple colors, and curves & other high-level features. For example, the filters on the first layer convolve around the input image and "activate" (or compute high values) when the specific feature (say curve) it is looking for is in the input volume.
  - Relu (Activation layer)

    After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that Relu layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. The Relu layer applies the function f(x) = max (0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.
  - Pooling Layer

    After some Relu layers, programmers may choose to apply a pooling layer. It is also referred to as a down sampling layer. In this category, there are also several layer options, with Maxpooling being the most popular. This basically takes a filter (normally of size 2x2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub region that the filter convolves around.

    The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. As you can imagine, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. The first is that the number of parameters or weights is reduced by 75%, thus lessening the computation cost. The second is that it will control over fitting. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of over fitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.
  - Batchnorm Layer

    Batch normalization layer reduces the internal covariance shift. In order to train a neural network, we do some preprocessing to the input data. For example, we could normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). Reason being preventing the early saturation of non-linear activation functions like the sigmoid function, assuring that all input data is in the same range of values, etc.

    But the problem appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt them to a new distribution in every training step. This problem is known as internal covariate shift.

    Batch normalization layer forces the input of every layer to have approximately the same distribution in every training step by following below process during training time:
    - Calculate the mean and variance of the layers input.
      - Normalize the layer inputs using the previously calculated batch statistics.
      - Scales and shifts in order to obtain the output of the layer.

    This makes the learning of layers in the network more independent of each other and allows you to be care free about weight initialization, works as regularization in place of dropout and other regularization techniques.
  - Drop-out layer

Dropout layers have a very specific function in neural networks. After training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. The idea of dropout is simplistic in nature. This layer "drops out" a random set of activations in that layer by setting them to zero. It forces the network to be redundant. That means the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. It makes sure that the network isn't getting too "fitted" to the training data and thus helps alleviate the over fitting problem. An important note is that this layer is only used during training, and not during test time.

- Fully connected layer

  This layer basically takes an input volume (whatever the output is of the conv or Relu or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program must choose from.
- Argmax Pooling Layer

  Performs max pooling same as pooling layer and outputs both max values and indices. This later can be reused to perform unpooling layer.
- Unpooling Layer

  The unpooling operation is used to revert the effect of the max pooling operation. The idea is just to work as an upsampler.
- Channel Padding Layer

  Channel padding or zero padding is layer that can be used to pad any dimensions with zeros to get desired dimensions as output.
- Resize Bilinear Layer

  Resizing an image (or a feature map) to a desired spatial dimension is a common operation when building computer vision applications based on convolutional neural networks. Bilinear resizing a 2-D array relies on bilinear interpolation, which can be broken down into linear resizing operations in y (height) and x (width) dimension.
- Quantization

  Quantization is a method to bring the neural network to a reasonable size, while also achieving high performance accuracy. This is especially important for on-device applications, where the memory size and number of computations are necessarily limited. Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers. This dramatically reduces both the memory requirement and computational cost of using neural networks.

Above architecture provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting.

5.1.2. Semantic Segmentation Network Output

Semantic Segmentation network gives Output tensor of dimension (BATCH_SIZE, Image Width, Image Height, Num Classes).

- Background Blurring Demo:

  Demo has 2 classes as below:

  1. User
  2. Background

  Input Dimension: 160x160x1

  Output Dimension: 160x160x2

  Output Interpretation: Each channel in output gives confidence of respective class for that pixel. In Background Blurring demo we have two channels each class. By applying argmax over channel we can get class index for given pixel as post processing.

5.1.3. Training Code Overview

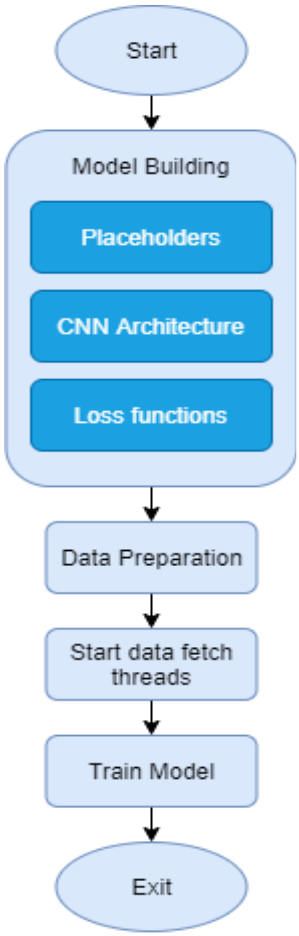


Figure 5.2. Training Code Flow Diagram

Training code can be divided into below parts:

- Model config
- Model building
- Model freezing
- Data preparation
- Training for overall execution flow.

Details of each can be found in subsequent sections.

5.1.3.1. Model Config

Below is summary of configurable parameters:

```
python train.py --train_dataset_dir="/home/dataset/augmented_data" \
                --val_dataset_dir="/home/dataset/generated_data" \
                --logdir="./log/Enet" \
                --is_training=True \
                --save_images=True \
                --combine_dataset=False \
                --num_classes=2 \
                --batch_size=10 \
                --eval_batch_size=25 \
                --image_height=160 \
                --image_width=160 \
                --num_epochs=300 \
                --num_epochs_before_decay=100 \
                --weighting="ENET" \
                --num_initial_blocks=1 \
                --stage_two_repeat=1 \
                --skip_connections=False
```

Figure 5.3. Code Snippet: train.sh configurations

- --train_dataset_dir: Training dataset path
- --val_dataset_dir: Validation dataset path
- --logdir: Checkpoints and tensorboard directory path
- --is_training: Is model training or not. This flag helps to generate freezing graph when user wants to generate frozen model
- --save_images: To save validation output images
- --combine_dataset: True if user wants to include validation dataset in training dataset
- --num_classes: Number of classes to train
- --batch_size: Training batch size
- --eval_batch_size: validation batch size
- --image_height: Input image height
- --image_width: Input image width
- --num_epochs: Number of epochs to train the model
- --num_epochs_before_decay: Number of epochs after user wants to reduce learning rate
- --weighting: “ENET or MFB” to choose between median frequency class weight balancing and Enet class balancing introduced in Enet paper.
- --num_initial_blocks: Number of initial CNN blocks to have in network in order to make network deeper.
- --stage_two_repeat: Number of times to repeat stage two in order to make network deeper.
- --skip_connections: If True, add the corresponding encoder feature maps to the decider. They are of exact same shapes.
- --weight_decay: The weight decay for Enet convolution layers
- --learning_rate_decay_factor: The learning rate decay factor
- --initial_learning_rate: The initial learning rate for training

5.1.3.2. Model Building

Enet class constructor builds model which can be divided in below sections:

- Forward graph
- Loss Function
- Train model

Forward graph

- CNN architecture consists of Convolution, Dilated Convolution, Batch normalization, Relu, Channel padding, Argmax pooling, Resize Bilinear and Maxpool layers.
- Script *“train.py”* handles data preparation, graph creation, training the model as well as test graph creation.

```
images, annotations = tf.train.batch([preprocessed_image, preprocessed_annotation], batch_size=batch_
                                     allow_smaller_final_batch=True)

# Create the model inference
with slim.arg_scope(ENet_arg_scope(weight_decay=weight_decay)):
    logits, probabilities = ENet(images,
                                 num_classes,
                                 batch_size=batch_size,
                                 is_training=is_training,
                                 reuse=None,
                                 num_initial_blocks=num_initial_blocks,
                                 stage_two_repeat=stage_two_repeat,
                                 skip_connections=skip_connections)
```

Figure 5.4. Code Snippet: Create Graph

- Forward graph consists of one initial block, 17 Bottleneck layers and one output block as shown in Figure 5.1.

Loss Function and training graph

This block calculates loss which needs to be minimized. In order to learn segmentation weighed cross entropy loss function will be used to get a mask with each pixel's value representing the weights.

```
def weighted_cross_entropy(onehot_labels, logits, class_weights):
    """A quick wrapper to compute weighted cross entropy...."""
    # weights = onehot_labels * class_weights
    weights = onehot_labels
    weights = tf.reduce_sum(weights, 3)
    loss = tf.losses.softmax_cross_entropy(onehot_labels=onehot_labels, logits=logits, weights=weights)

    return loss
```

Figure 5.5. Code Snippet: Loss Function

```
# Now we can define the optimizer that takes on the learning rate
optimizer = tf.train.AdamOptimizer(learning_rate=lr, epsilon=epsilon)

# Create the train_op.
train_op = slim.learning.create_train_op(total_loss, optimizer)

# State the metrics that you want to predict. We get a predictions that is not one_hot_encoded.
predictions = tf.argmax(probabilities, -1)
accuracy, accuracy_update = tf.contrib.metrics.streaming_accuracy(predictions, annotations)
mean_IOU, mean_IOU_update = tf.contrib.metrics.streaming_mean_iou(predictions=predictions, labels=annotations,
                                                                  num_classes=num_classes)
metrics_op = tf.group(accuracy_update, mean_IOU_update)
```

Figure 5.6. Code Snippet: Training Graph

Adam optimizer will be used to panelize weights. Also, above spinet show the accuracy and IOU calculation while training.

5.2. Training

To train the machine:

1. Modify training script

   Training script at @train.sh is used to trigger training. **Figure 5.3** shows the input parameters which can be configured.

2. Execute the train.sh script which starts training.

```
$ bash train.sh
100%|                                                                                | 172125/172125 [14:07<00:00, 203.07it/s]
========= ENet Class Weights =========
 [1.6701622214015108, 5.025416365192917]
2022-05-16 11:44:07.422311: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1
2022-05-16 11:44:07.481669: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: NVIDIA GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.65
pciBusID: 0000:01:00.0
2022-05-16 11:44:08.003479: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 3491835000 Hz
2022-05-16 11:44:08.004207: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
2022-05-16 11:44:08.004715: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: NVIDIA GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.65
2022-05-16 11:44:08.409419: I tensorflow/compiler/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA GeForce RTX 2080 Ti, Compute Capability 7.5
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting standard services.
INFO:tensorflow:Saving checkpoint to path ./log/Enet/model.ckpt
INFO:tensorflow:Starting queue runners.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Epoch 1.0/300
2022-05-16 11:44:45.263388: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10.0
INFO:tensorflow:Current Learning Rate: [0.0005]
2022-05-16 11:44:59.804109: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
2022-05-16 11:45:05.888458: W tensorflow/stream_executor/cuda/redzone_allocator.cc:312] Not found: ./bin/ptxas not found
Relying on driver to perform ptx compilation. This message will be only logged once.
INFO:tensorflow:global step 0: loss: 0.6934 (15.45 sec/step)  Current Streaming Accuracy: 0.0000  Current Mean IOU: 0.0000
INFO:tensorflow:---VALIDATION--- Validation Accuracy: 0.0000    Validation Mean IOU: 0.0000    (3.44 sec/step)
INFO:tensorflow:---VALIDATION--- Validation Accuracy: 0.7666    Validation Mean IOU: 0.3833    (0.04 sec/step)
```

Figure 5.7. Execute Run Script

3. Start TensorBoard.

```
$ tensorboard –logdir=<log directory of training>
For example: tensorboard –logdir='./logs/'
```

4. Open the local host port on your web browser.

```
$ tensorboard --logdir log/Enet/
TensorBoard 1.15.0 at http://desktop:6006/ (Press CTRL+C to quit)
```

Figure 5.8. TensorBoard – Generated Link

5. Check the training status on TensorBoard

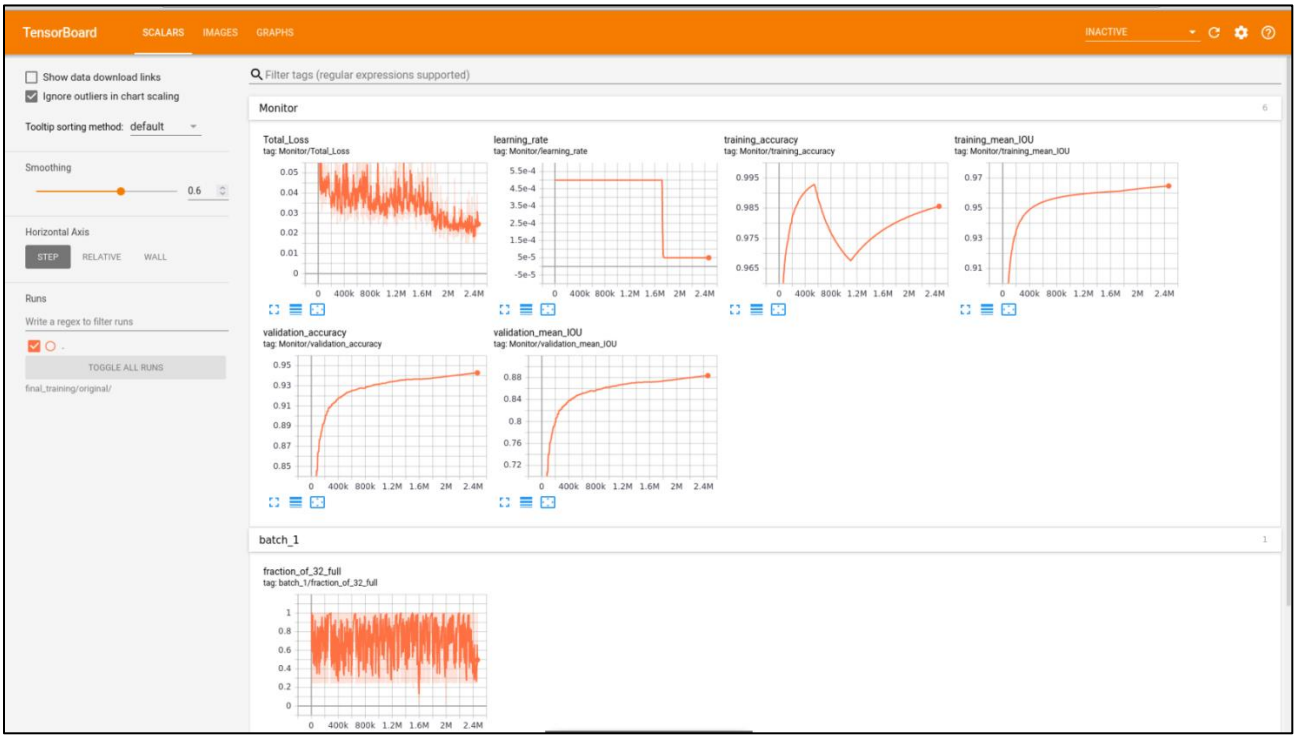


Figure 5.9. TensorBoard

Figure 5.9 shows the image menu of TensorBoard.

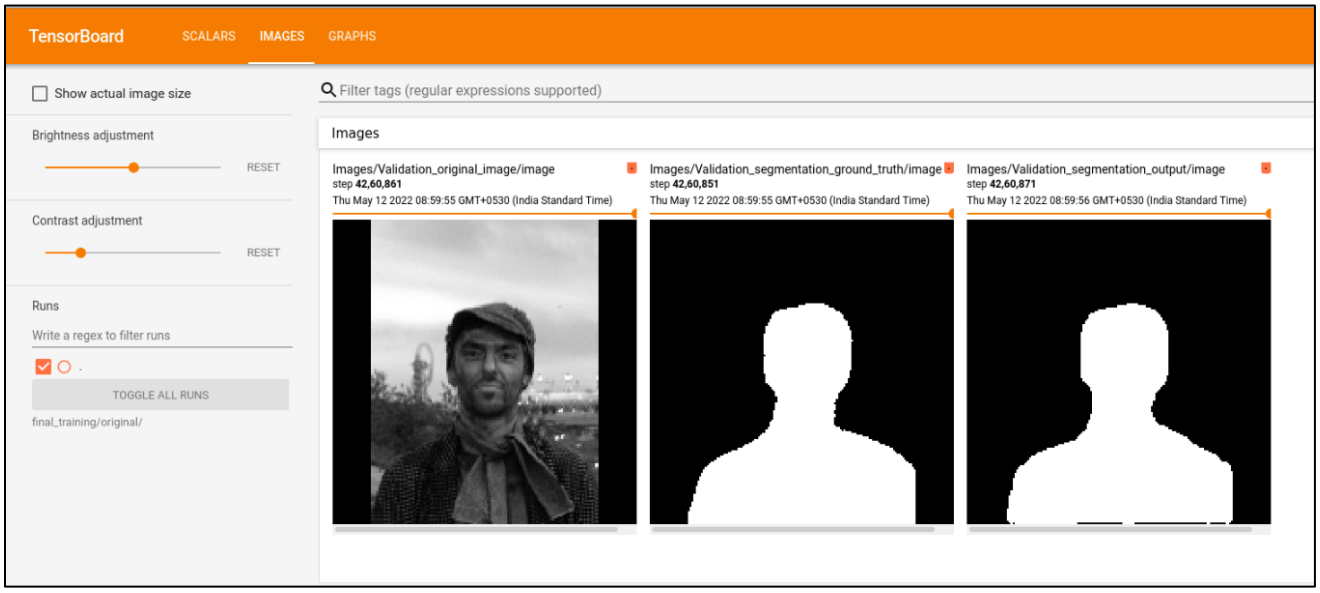


Figure 5.10. Image Menu of TensorBoard

6. Check if the checkpoint, data, meta, index, and events (if using TensorBoard) files are created at the log directory. These files are used for creating the frozen file (*.pb).

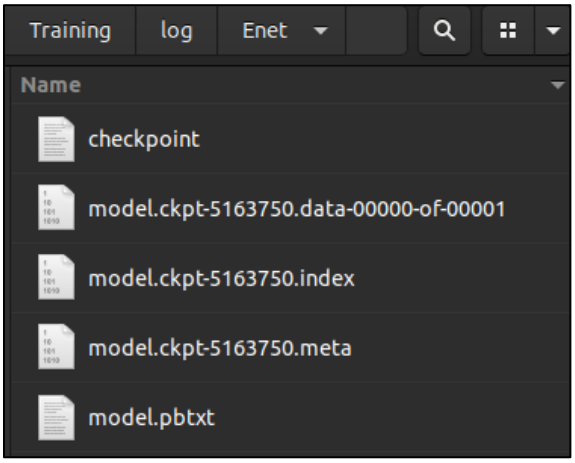


Figure 5.11. Example of Checkpoint Data Files at Log Folder

6. Creating Frozen File

This section describes the procedure for freezing the model, which is aligned with the Lattice SensAI tool. Perform the steps below to generate the frozen protobuf file:

6.1. Generating the Frozen (.pb) File

```
python train.py --train_dataset_dir="/home/dataset/augmented_data" \
                --val_dataset_dir="/home/dataset/generated_data" \
                --logdir="./log/Enet" \
                --is_training=False \
                --num_classes=2 \
                --batch_size=1 \
                --eval_batch_size=25 \
                --image_height=160 \
                --image_width=160 \
                --weighting="ENET" \
                --num_initial_blocks=1 \
                --stage_two_repeat=1 \
                --skip_connections=False

python genpb.py --ckpt_dir="./log/Enet"
```

Figure 6.1. freeze.sh configuration

Set all parameters in freeze.sh same as train.sh which is used while training the model, also set “ckpt_dir” path same as “logdir”.

```
$ bash freeze.sh
```

```
$ bash freeze.sh
100%|| 172125/172125 [00:48<00:00, 3534.92it/s]
========= ENet Class Weights =========
 [1.6701622214015108, 5.025416365192917]
name: NVIDIA GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.65
pciBusID: 0000:01:00.0
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Saving checkpoint to path ./log/Enet/model.ckpt
inputShape shapes [None, None, None, None]
output_shapes of input Node [None, None, None, None]
 **TensorFlow**: can not locate input shape information at: batch
node to modify name: "batch"
op: "Placeholder"
attr {
  key: "dtype"
  value {
    type: DT_FLOAT
  }
}

--Name of the node - batch shape set to  batch [1, 160, 160, 1]
node after modify name: "batch"
op: "Placeholder"
attr {
  key: "dtype"
  value {
    type: DT_FLOAT
  }
}
```

Figure 6.2. Run freeze.sh To Generate Inference .pb

- *“freeze.sh”* will use latest checkpoint in train directory to generate frozen ‘.pb’ file.
- Once the *“freeze.sh”* is executed successfully the log directory will now have <ckpt-prefix>_frozenforInference.pb file as shown in screenshot above.

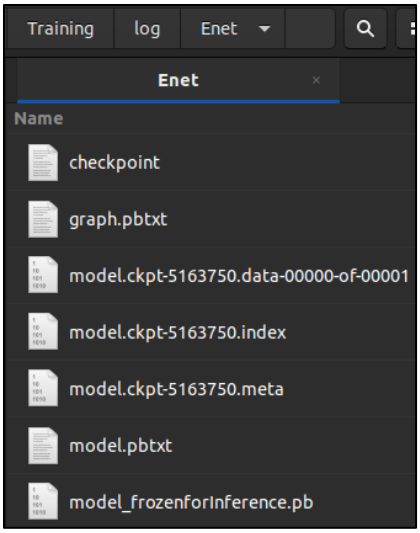


Figure 6.3. Frozen Inference. pb Output

7. Model Evaluation

This section contains guide to calculate model performance in terms of MAP.

7.1. Run Inference on test set and calculate IOU/DICE

Semantic Segmentation code contains ‘testing.py’.

Note: If user did any change in training code regarding image size, number of classes than user should replicate those changes in testing script.

Run below command to run inference on test set.

```
$ python testing.py –pb <converted pb path> --input_images <test set images path>
```

```
$ python testing.py --pb log/Enet/model_frozenforInference.pb --input_images /dataset/Enet/generated_data/

100%|                                                                                | 1000/1000 [00:20<00:00, 47.79it/s]
MEAN IOU 0.5407852523130762
MEAN DICE 0.6953797921528094
```

Figure 7.1. Run Testing

8. Creating Binary File with SensAI

This chapter describes how to generate binary file using the Lattice SensAI version 5.0 program.

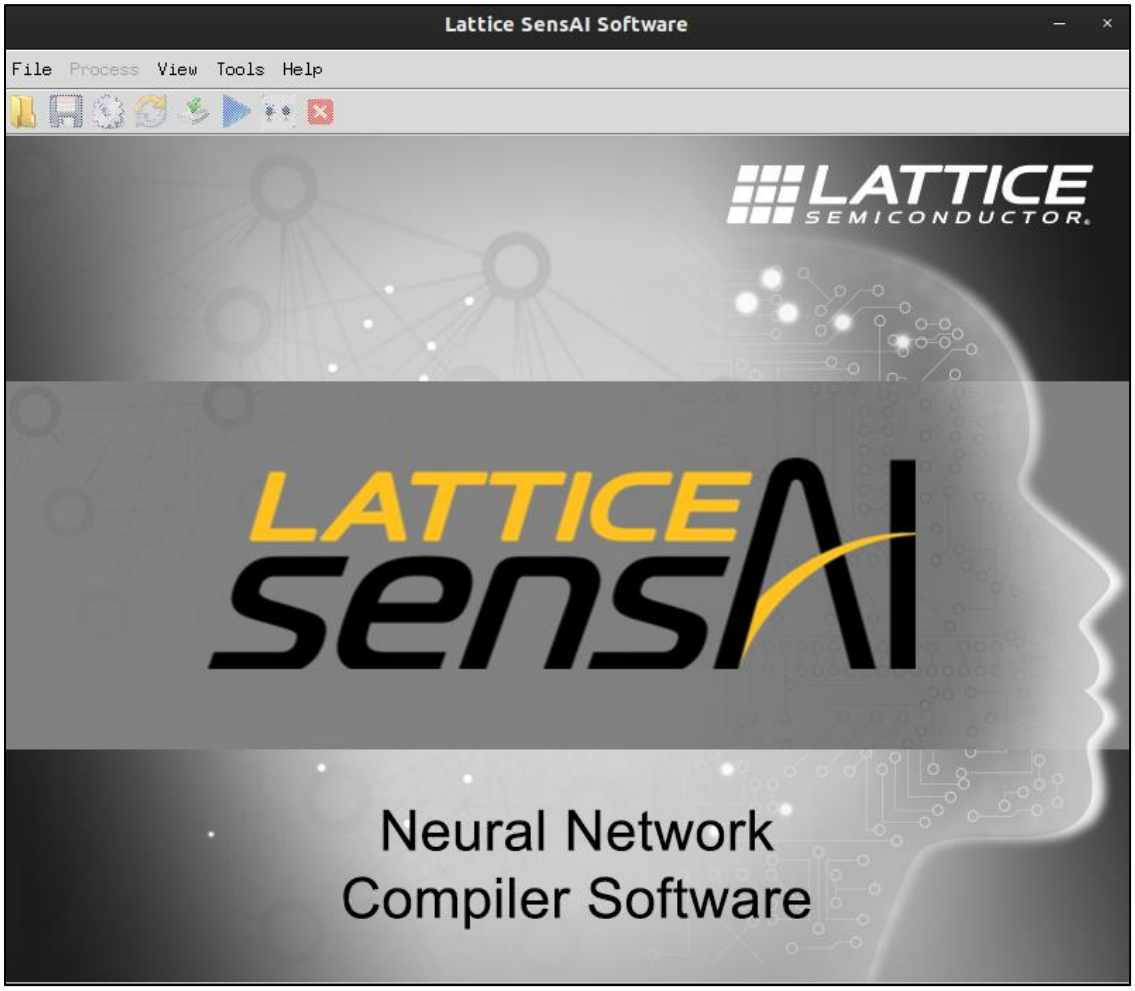


Figure 8.1. SensAI – Home Screen

To create the project in SensAI tool:

1. Click File > New.
2. Enter the following settings:

- Project name
- Framework – TensorFlow
- Class – CNN
- Device – Crosslink-NX
- IP – Extended_CNN

3. Click Network File and select the network (PB) file.

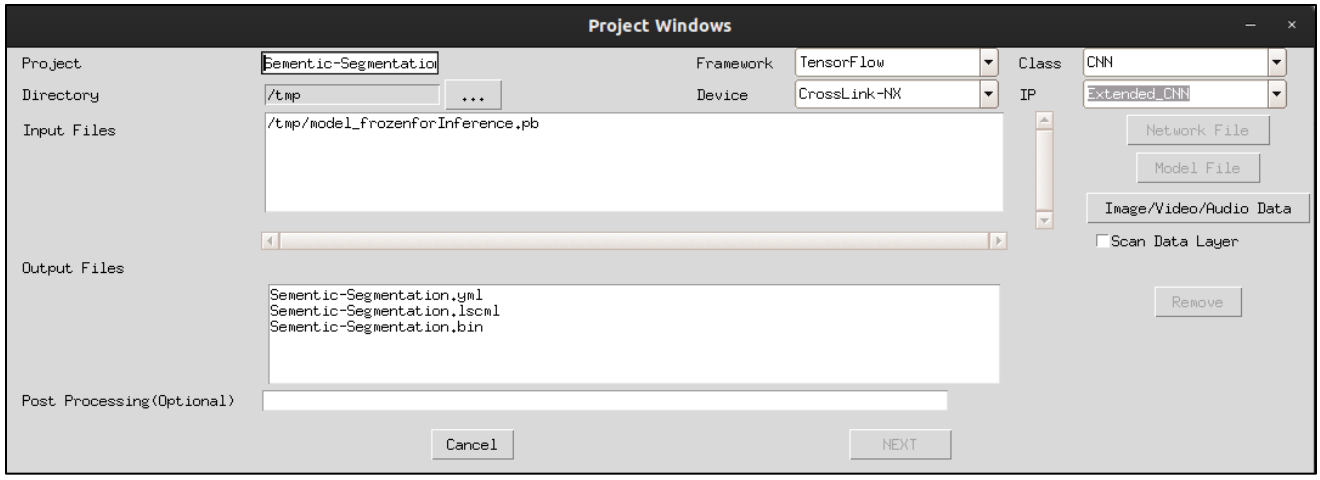


Figure 8.2. SensAI – Select Framework, Device and Network File

4. Click **Image/Video/Audio Data** button and select the audio input file.

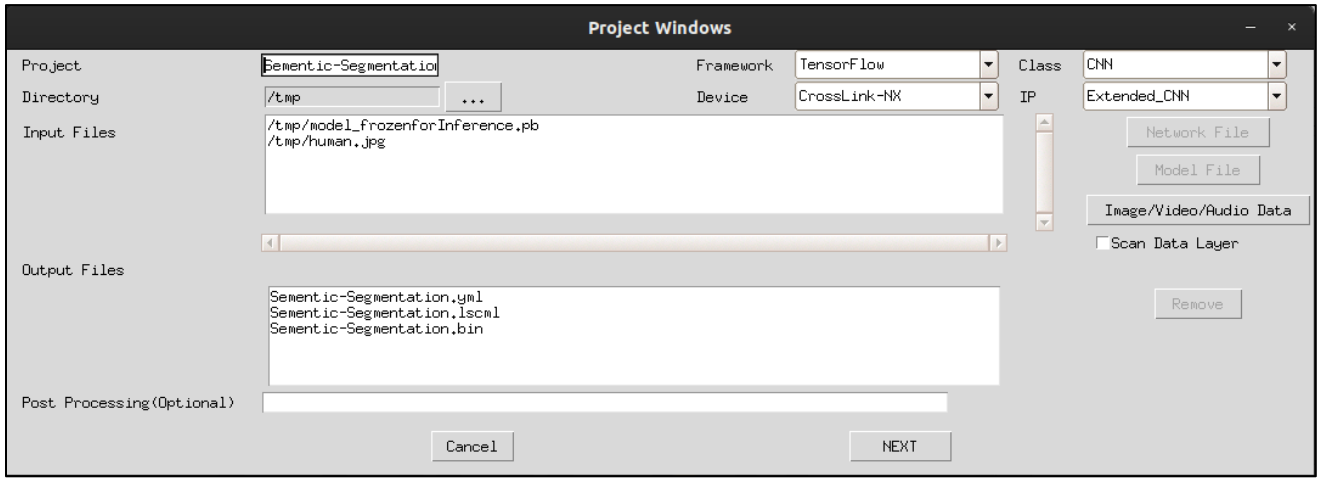


Figure 8.3. SensAI – Select image Data File

5. Click Next.
6. Set following attributes:
    a. Mean Value for Data Pre-Processing: 0
    b. Scratch Pad Memory Block Size: 8191
    c. ARGS MAX Size: 8192
    d. On-Chip Memory Block Size: 131072
    e. Scale Value for Data Pre-Processing: 0.0078125

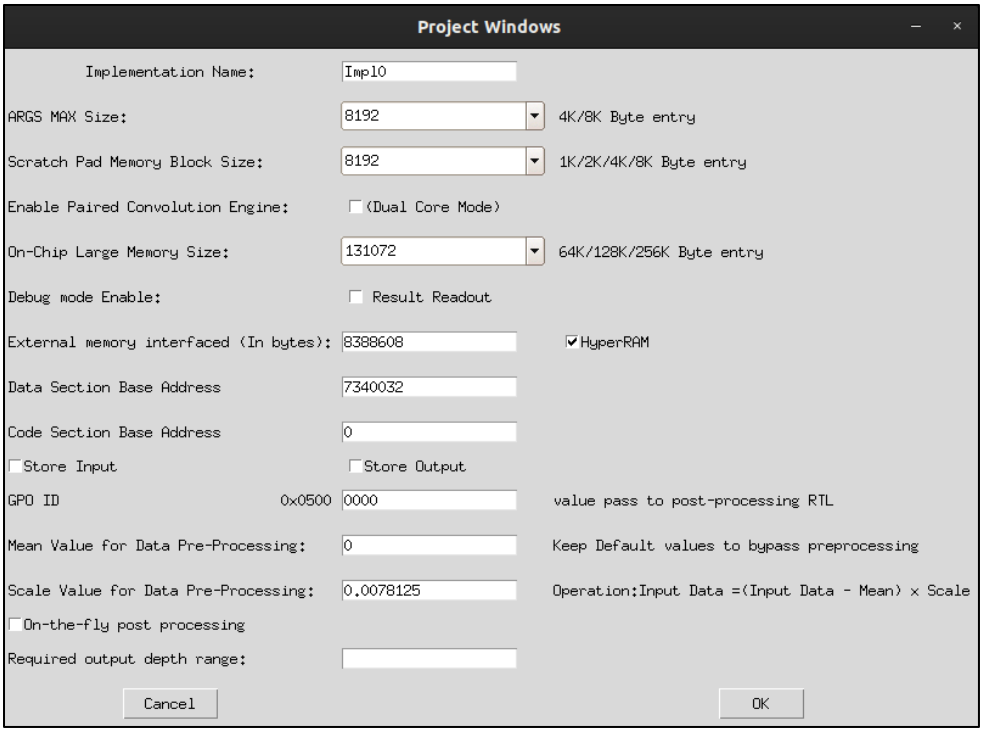


Figure 8.4. SensAI – Update Project Settings

7. Clock Ok to create project.
8. Double click on Analyze.

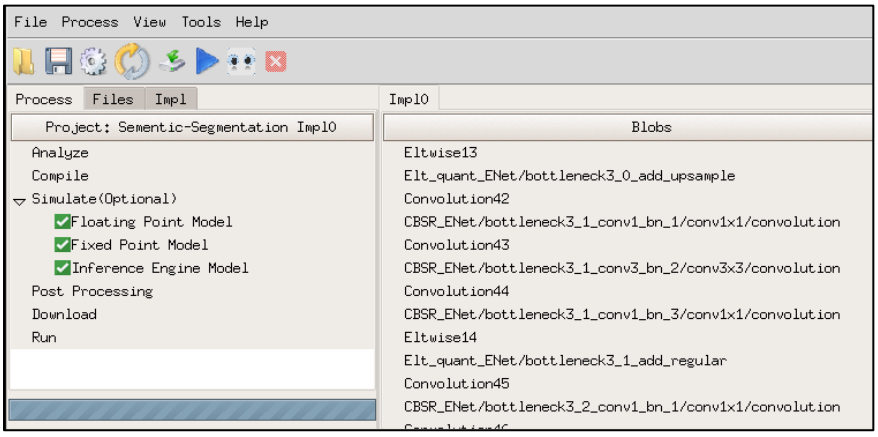


Figure 8.5. Analyze Project

9. Confirm the Q format of each layer as shown in Figure 8.6.

Impl0

Blobs	Data Format (Analyzed)	Stored Data Format	Internal(External) Mem Byte
data	8.7	1.7	25600 (N/A)
Convolution1	5.10	5.10	None
CBSR_ENet/initial_block_1_conv_bn_0/conv3x3/convolution	8.7	1.7	13440 (102400)
Convolution2	5.10	5.10	None
CBSR_ENet/initial_block_1_conv_bn/conv3x3/convolution	8.7	1.7	39360 (51200)
CBSR_ENet/initial_block_1_conv_bn_0/conv3x3/convolution_dummy	5.10	5.10	None
_singlePooling_ENet/initial_block_1_pool/MaxPool	8.7	1.7	39360 (25600)
Concat_PlaceHolder	5.10	5.10	None
Concat_ENet/initial_block_1_concat	8.7	1.7	None
Concat_ENet/initial_block_1_concat_dummy	5.10	5.10	None
Arg_Pooling3	8.7	1.7	12800 (19200)
ChannelPad_PlaceHolder	5.10	5.10	None
ChannelPadding_ENet/bottleneck1_0_main_padding	8.7	1.7	25600 (19200)
Convolution3	5.10	5.10	None
CBSR_ENet/bottleneck1_0_conv3_bn_1/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution4	5.10	5.10	None
CBSR_ENet/bottleneck1_0_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution5	5.10	5.10	None
CBSR_ENet/bottleneck1_0_conv1_bn_3/conv1x1/convolution	8.7	1.7	25600 (64000)
Eltwise1	5.10	5.10	None
Elt_quant_ENet/bottleneck1_0_add	8.7	1.7	32000 (64000)
Convolution6	5.10	5.10	None
CBSR_ENet/bottleneck1_1_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Convolution7	5.10	5.10	None
CBSR_ENet/bottleneck1_1_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution8	5.10	5.10	None
CBSR_ENet/bottleneck1_1_conv1_bn_3/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise2	5.10	5.10	None
Elt_quant_ENet/bottleneck1_1_add_regular	8.7	1.7	32000 (64000)
Convolution9	5.10	5.10	None
CBSR_ENet/bottleneck1_2_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Convolution10	5.10	5.10	None
CBSR_ENet/bottleneck1_2_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution11	5.10	5.10	None
CBSR_ENet/bottleneck1_2_conv1_bn_3/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise3	5.10	5.10	None
Elt_quant_ENet/bottleneck1_2_add_regular	8.7	1.7	32000 (64000)
Convolution12	5.10	5.10	None
CBSR_ENet/bottleneck1_3_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Convolution13	5.10	5.10	None
CBSR_ENet/bottleneck1_3_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution14	5.10	5.10	None
CBSR_ENet/bottleneck1_3_conv1_bn_3/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise4	5.10	5.10	None
Elt_quant_ENet/bottleneck1_3_add_regular	8.7	1.7	32000 (64000)
Convolution15	5.10	5.10	None
CBSR_ENet/bottleneck1_4_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Convolution16	5.10	5.10	None
CBSR_ENet/bottleneck1_4_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution17	5.10	5.10	None
CBSR_ENet/bottleneck1_4_conv1_bn_3/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise5	5.10	5.10	None
Elt_quant_ENet/bottleneck1_4_add_regular	8.7	1.7	33600 (64000)
Elt_quant_ENet/bottleneck1_4_add_regular_dummy	5.10	5.10	None
Arg_Pooling6	8.7	1.7	8000 (16000)
ChannelPadding_ENet/bottleneck2_0_main_padding	8.7	1.7	17600 (16000)
Convolution18	5.10	5.10	None
CBSR_ENet/bottleneck2_0_conv3_bn_1/conv3x3/convolution	8.7	1.7	3200 (N/A)
Convolution19	5.10	5.10	None
CBSR_ENet/bottleneck2_0_conv3_bn_2/conv3x3/convolution	8.7	1.7	3200 (N/A)
Convolution20	5.10	5.10	None
CBSR_ENet/bottleneck2_0_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise6	5.10	5.10	None
Elt_quant_ENet/bottleneck2_0_add	8.7	1.7	17600 (32000)
Convolution21	5.10	5.10	None
CBSR_ENet/bottleneck2_1_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
Convolution22	5.10	5.10	None
CBSR_ENet/bottleneck2_1_conv3_bn_2/conv3x3/convolution	8.7	1.7	4800 (N/A)
Convolution23	5.10	5.10	None
CBSR_ENet/bottleneck2_1_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise7	5.10	5.10	None
Elt_quant_ENet/bottleneck2_1_add_regular	8.7	1.7	17600 (32000)
Convolution24	5.10	5.10	None
CBSR_ENet/bottleneck2_2_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
Convolution25	5.10	5.10	None
CBSR_ENet/bottleneck2_2_dilated_conv3_bn_2/conv3x3/convolution	8.7	1.7	4800 (N/A)
Convolution26	5.10	5.10	None
CBSR_ENet/bottleneck2_2_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise8	5.10	5.10	None
Elt_quant_ENet/bottleneck2_2_add_dilated	8.7	1.7	17600 (32000)
Convolution27	5.10	5.10	None

CBSR_ENet/bottleneck2_3_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
Convolution28	5.10	5.10	None
CBSR_ENet/bottleneck2_3_conv3_bn_2/conv3x3/convolution	8.7	1.7	4800 (N/A)
Convolution29	5.10	5.10	None
CBSR_ENet/bottleneck2_3_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise9	5.10	5.10	None
Elt_quant_ENet/bottleneck2_3_add_regular	8.7	1.7	17600 (32000)
Convolution30	5.10	5.10	None
CBSR_ENet/bottleneck2_4_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
Convolution31	5.10	5.10	None
CBSR_ENet/bottleneck2_4_dilated_conv3_bn_2/conv3x3/convolution	8.7	1.7	4800 (N/A)
Convolution32	5.10	5.10	None
CBSR_ENet/bottleneck2_4_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise10	5.10	5.10	None
Elt_quant_ENet/bottleneck2_4_add_dilated	8.7	1.7	17600 (32000)
Convolution33	5.10	5.10	None
CBSR_ENet/bottleneck2_5_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
Convolution34	5.10	5.10	None
CBSR_ENet/bottleneck2_5_conv3_bn_2/conv3x3/convolution	8.7	1.7	4800 (N/A)
Convolution35	5.10	5.10	None
CBSR_ENet/bottleneck2_5_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise11	5.10	5.10	None
Elt_quant_ENet/bottleneck2_5_add_regular	8.7	1.7	17600 (32000)
Convolution36	5.10	5.10	None
CBSR_ENet/bottleneck2_6_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
Convolution37	5.10	5.10	None
CBSR_ENet/bottleneck2_6_dilated_conv3_bn_2/conv3x3/convolution	8.7	1.7	4800 (N/A)
Convolution38	5.10	5.10	None
CBSR_ENet/bottleneck2_6_conv1_bn_3/conv1x1/convolution	8.7	1.7	17600 (32000)
Eltwise12	5.10	5.10	None
Elt_quant_ENet/bottleneck2_6_add_dilated	8.7	1.7	17920 (32000)
Convolution39	5.10	5.10	None
CBSR_ENet/bottleneck3_0_main_conv1/conv1x1/convolution	8.7	1.7	8000 (N/A)
Upsample_ENet/unpool/ScatterNd	8.7	1.7	32000 (64000)
Convolution40	5.10	5.10	None
CBSR_ENet/bottleneck3_0_conv1_bn_1/conv1x1/convolution	8.7	1.7	4800 (N/A)
ResizeBilinear1	5.10	5.10	None
CBSR_ENet/bottleneck3_0_resize	8.7	1.7	19200 (N/A)
Convolution41	5.10	5.10	None
CBSR_ENet/bottleneck3_0_conv1_bn_4/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise13	5.10	5.10	None
Elt_quant_ENet/bottleneck3_0_add_upsample	8.7	1.7	32000 (64000)
Convolution42	5.10	5.10	None
CBSR_ENet/bottleneck3_1_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Convolution43	5.10	5.10	None
CBSR_ENet/bottleneck3_1_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution44	5.10	5.10	None
CBSR_ENet/bottleneck3_1_conv1_bn_3/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise14	5.10	5.10	None
Elt_quant_ENet/bottleneck3_1_add_regular	8.7	1.7	32000 (64000)
Convolution45	5.10	5.10	None
CBSR_ENet/bottleneck3_2_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Convolution46	5.10	5.10	None
CBSR_ENet/bottleneck3_2_conv3_bn_2/conv3x3/convolution	8.7	1.7	12800 (N/A)
Convolution47	5.10	5.10	None
CBSR_ENet/bottleneck3_2_conv1_bn_3/conv1x1/convolution	8.7	1.7	32000 (64000)
Eltwise15	5.10	5.10	None
Elt_quant_ENet/bottleneck3_2_add_regular	8.7	1.7	33600 (64000)
Convolution48	5.10	5.10	None
CBSR_ENet/bottleneck4_0_main_conv1/conv1x1/convolution	8.7	1.7	12800 (N/A)
Upsample_ENet/unpool_1/ScatterNd	8.7	1.7	25600 (76800)
Convolution49	5.10	5.10	None
CBSR_ENet/bottleneck4_0_conv1_bn_1/conv1x1/convolution	8.7	1.7	12800 (N/A)
ResizeBilinear2	5.10	5.10	None
CBSR_ENet/bottleneck4_0_resize	8.7	1.7	51200 (N/A)
Convolution50	5.10	5.10	None
CBSR_ENet/bottleneck4_0_conv1_bn_4/conv1x1/convolution	8.7	1.7	25600 (84480)
Eltwise16	5.10	5.10	None
Elt_quant_ENet/bottleneck4_0_add_upsample	8.7	1.7	25600 (76800)
Convolution51	5.10	5.10	None
CBSR_ENet/bottleneck4_1_conv1_bn_1/conv1x1/convolution	8.7	1.7	25600 (N/A)
Convolution52	5.10	5.10	None
CBSR_ENet/bottleneck4_1_conv3_bn_2/conv3x3/convolution	8.7	1.7	25600 (N/A)
Convolution53	5.10	5.10	None
CBSR_ENet/bottleneck4_1_conv1_bn_3/conv1x1/convolution	8.7	1.7	25600 (76800)
Eltwise17	5.10	5.10	None
Elt_quant_ENet/bottleneck4_1_add_regular	8.7	1.7	10560 (76800)
ResizeBilinear3	5.10	5.10	None
CBSR_ENet/resize	8.7	1.7	43520 (307200)
ENet/ENet_fullconv/convolution	5.10	5.10	None
CBSR_ENet/ENet_fullconv/convolution	8.7	5.10	10240 (10240)

Figure 8.6. Q Format Settings for Each Layer

10. Double click on **compile** to generate the Firmware and filter binary file.

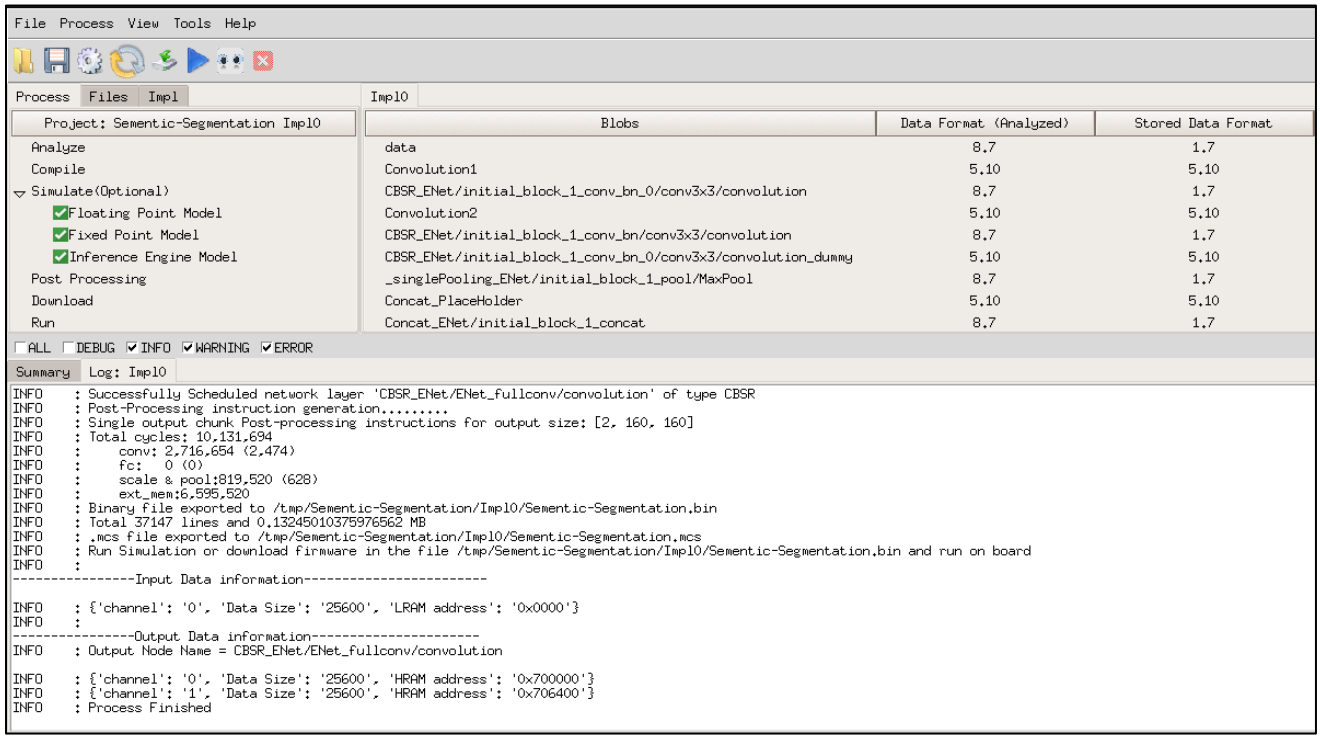


Figure 8.7. Compile Project

Firmware bin file location will be displayed in the compilation log. Use generated firmware bin on hardware for testing.

9. Hardware (RTL) Implementation

9.1. Top Level Information

9.1.1. Block Diagram

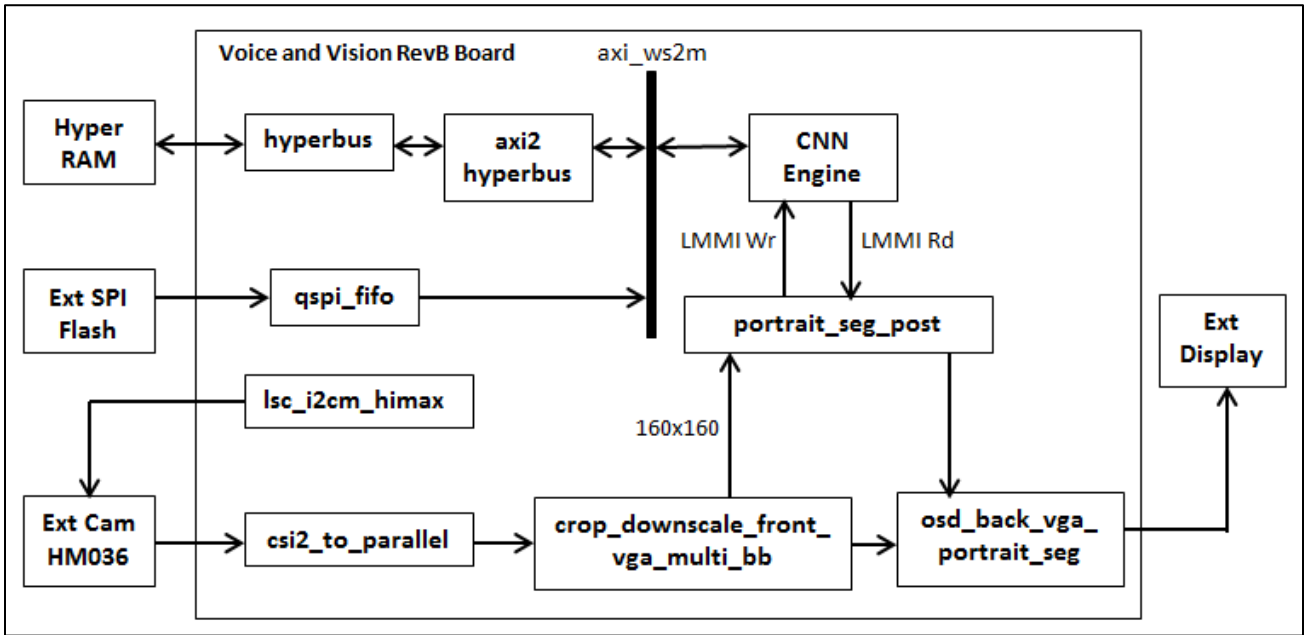


Figure 9.1. Top Block Diagram of Portrait segmentation with Crosslink-NX Voice & Vision ML (RevB) Board

9.1.2. Operational Flow

- The external camera Himax HM0360 is configured using I2C Master Block (*lsc_i2cm_himax.v)*.
- The real time input image data is received by Video path. The RAW8 data from (csi2_to_parallel.v) is sent to pre-processing (*crop_downscale_front_vga_multi_bb.v)* which performs Crop & Downscale operation to provide compatible input image resolution of 160x160 to Extended CNN engine.
- The 1MB firmware BIN file (.mcs) is loaded to the SPI Flash module (q*spi_fifo.v*) configured with starting address 24'h300000 to end address 24'h400000.
- CNN Engine receives the downscaled image data from (*portrait_seg_post.v*) through LMMI Write interface and the firmware file through AXI hyperbus interface to provide inference result output.
- CNN inference output is read again by (*portrait_seg_post.v*) through LMMI Read interface and passed to OSD block for output display.
- For final output display, the (*osd_back_vga_portrait_seg.v*) module performs segmentation on the received downscaled image from Crop & Downscale module using the CNN object detection pixel information obtained from (*portrait_seg_post.v*) block.

9.1.3. Core Customization

Table 9.1. Core Parameters

Parameter	Value	Description
USE_ML	1'b1	Indicates 1:Enable/0:Disable to use Extended CNN Engine
ML_TYPE	EXTENDED_CNN	Can be configured as COMPACT_CNN , OPTIMIZED_CNN or EXTENDED_CNN
SCRATCH_SIZE	8K	Indicates scratch pad memory size of Extended CNN
LB_SIZE	512	Indicates line buffer size of Extended CNN

Parameter	Value	Description
BYTE_MODE	UNSIGNED	Indicates byte mode type of Extended CNN
BYTE_SHIFT	3'b101	Indicates byte shift value of Extended CNN
MEM_TYPE	DUAL_LRAM	Indicates memory type of Extended CNN
DUAL_CONV	1'b1	Indicates convolution engine type of Extended CNN
CONV1X1	OCTA	Indicates Machine learning convolution type 1x1 of Extended CNN
ARGMAX_SIZE	8192	Indicates maximum argument size of Extended CNN
EN_UART	1'b0	Indicates 1:Enable/0:Disable UART for video output
FIRMWARE_ADDR	24'h300000	Indicates starting address to load Firmware in external SPI Flash
FIRMWARE_SIZE	24'h100004	Indicates size of firmware to be loaded in external SPI Flash. Here is 1 MB size.

9.2. Architectural Details

9.2.1. Pre-Processing operation

- The (*crop_downscale_front_vga_multi_bb.v)* video processing block is used to crop and downscale the image data to make it compatible for CNN engine.
- Masking values for incoming image data from camera are set to capture the image data of resolution 640x480.
- As shown in below image initially this 640x480 image is downscaled into 160x120 image resolution using block size 4, and to obtain 160x160 image for CNN input, 40 lines are then padded vertically.

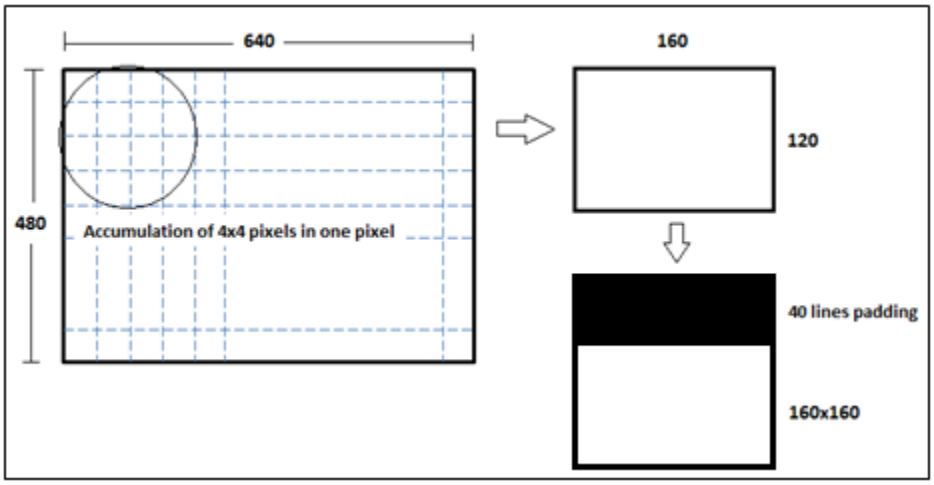


Figure 9.2. Downscaling image

- The accumulated pixel values are written into accumulation buffer. While reading the data from Buffer is sent to the CNN engine for inference through line buffer.

9.2.2. Post-Processing operation

The post processing operation is explained as below.

- The (*portrait_seg_post.v*) block mainly handles the task of providing downscaled input image to CNN for inference and receiving the detection results back.
- The writing and reading back data to and from Extended CNN engine takes place over LMMI Interface when it is idle and not running.
- When CNN Engine is not running, this block initially receives the 160x160 downscaled image from pre-processing module and provides it to CNN over LMMI WRITE Interface (lmmi_write_i).

- When CNN has done inference over this frame and again when it is idle, this module receives the CNN results over LMMI READ Interface (lmmi_rdata_o) and passes it on to the OSD module for display.
- The (*osd_back_vga_portrait_seg.v*) module received mainly the 160x160 downscaled image from pre-processing module and the CNN result data from post processing block.
- Using the pixel information of human detection received from post –processing block, the OSD module performs segmentation over the 160x160 downscaled image.
- In the final segmentation output display, the human presence can be observed in gray-scale and the background is blurred with Green pixels.

This entire Pre and Post processing cycle goes on till the board is powered-on and the real time image is being captured by camera.

10. Creating FPGA Bit stream file

This section provides the procedure for creating your FPGA bit stream file using Lattice Radiant Software.

To create the FPGA bit stream file, follow the below steps.

1. STEP 1: Open Lattice Radiant Software as shown in Figure 10.1.

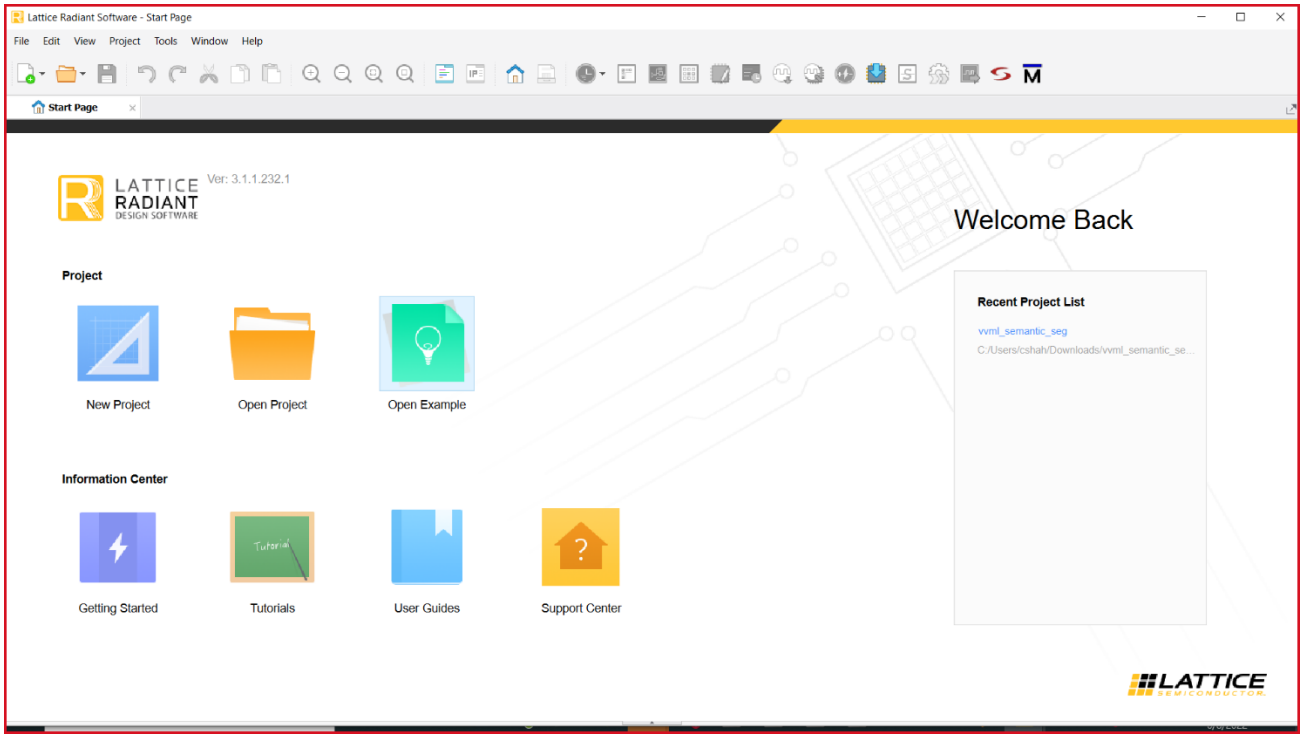


Figure 10.1 Radiant Software

2. STEP 2: Click File > Open Project and from project database open the Radiant project file (.rdf) from *vvml_semantic_seg* folder as shown in Figure 10.2.

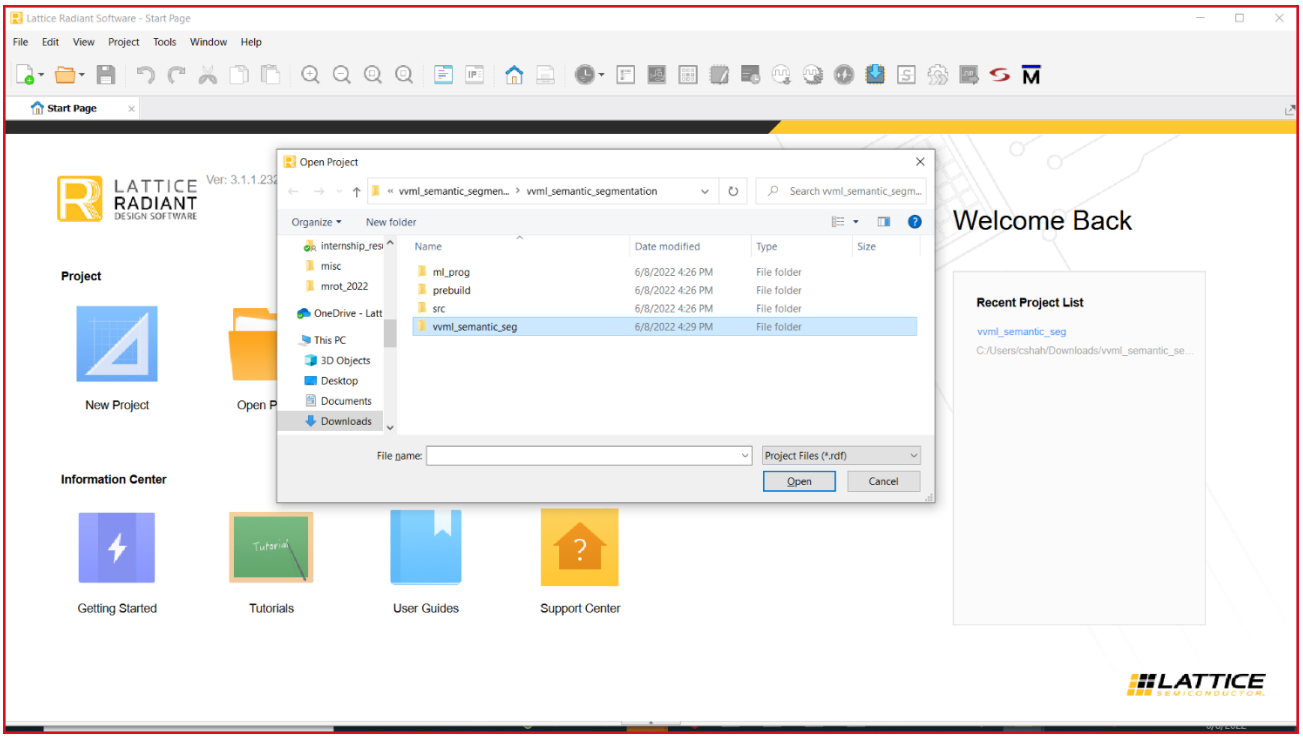


Figure 10.2. Radiant Software – Open Project

3. STEP 3: Click Export Files to generate the bit file. View the log message in Export Reports that indicates the generated bit stream. Find this bit file at location */vvml_semantic_seg/impl_1* as shown in Figure 10.3.

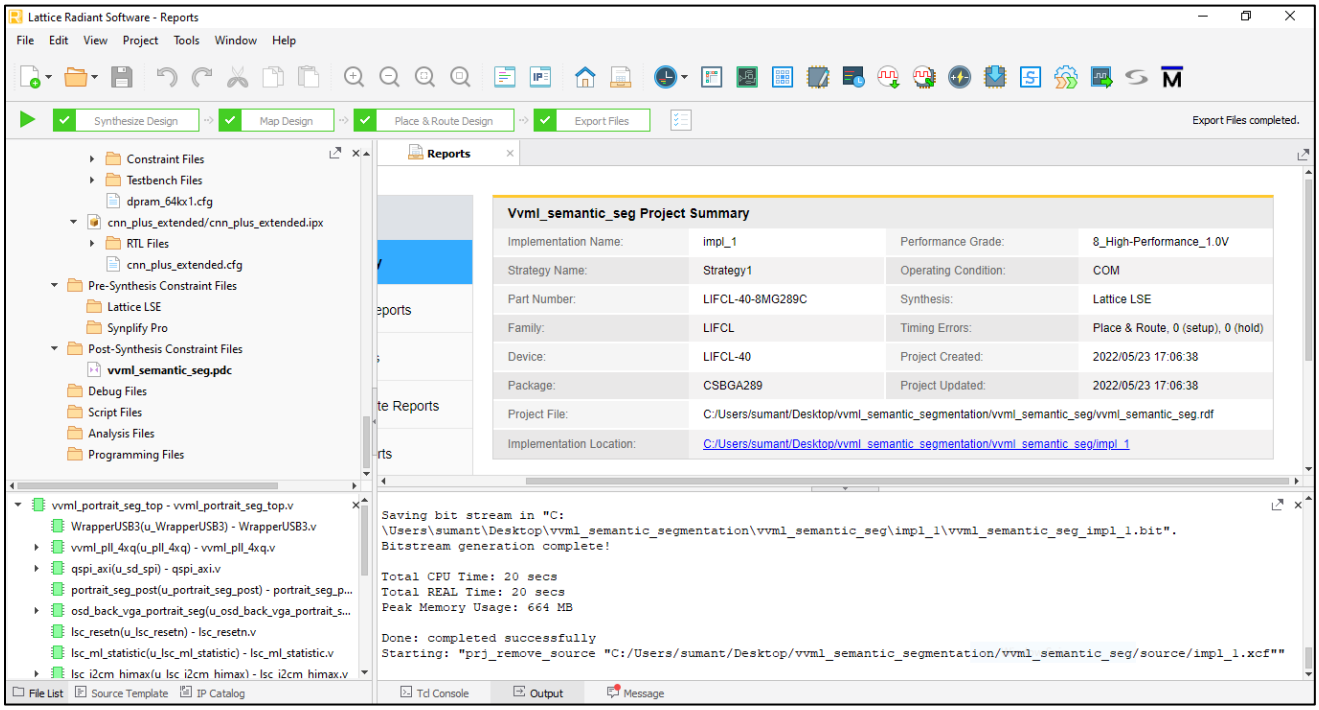


Figure 10.3. Radiant Software – Bit stream Generation Export Report

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.0, June 2022

Section	Change Summary
All	Initial release.

www.latticesemi.com