CrossLink-NX Human Counting Using VGG

Reference Design

FPGA-RD-02208-1.0

December 2020

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 and with all faults, and all risk associated with such information is entirely with 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
AXI	Advanced Extensible Interface
CNN	Convolutional Neural Network
FPGA	Field-Programmable Gate Array
NN	Neural Network
SD	Secure Digital
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory

1. Introduction

This document describes the Human Counting Design using VGG process, which is targeted for the CrossLink™-NX Voice and Vision Platform.

1.1. Design Process Overview

The design process involves the following steps:

1. Training the model
   - Setting up the basic environment
   - Preparing the dataset
     - Preparing 224 × 224 image
     - Labeling dataset of human bounding box
   - Training the machine
     - Training the machine and creating the checkpoint data
   - Creating the frozen file (*.pb)
2. Compiling Neural Network
   - Creating the binary file with Lattice SensAI™ 3.1 program
3. FPGA Design
   - Creating the FPGA bitstream file
4. FPGA Bitstream and Quantized Weights and Instructions
   - Flashing the binary and bitstream files
     - Binary File to Flash Memory on Crosslink-NX Voice and Vision board
     - Bitstream to Flash Memory on Crosslink-NX Voice and Vision Board

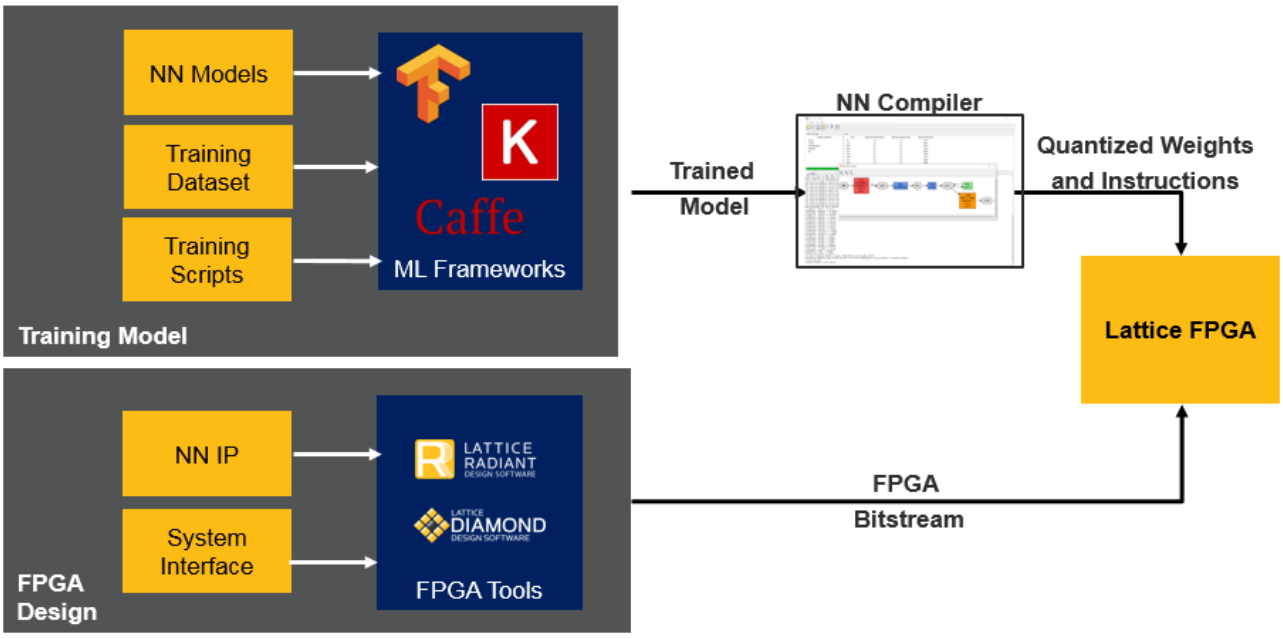

Figure 1.1. Lattice Machine Learning Design Flow

2. Setting Up the Basic Environment

2.1. Software and Hardware Requirements

This section describes the required tools and environment setup for the FPGA bitstream and flashing.

2.1.1. Lattice Software

- Lattice Radiant™ Tool – Refer to http://www.latticesemi.com/LatticeRadiant.
- Lattice Radiant Programmer – Refer to http://www.latticesemi.com/programmer.
- Lattice SensAI Compiler v3.1– 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.

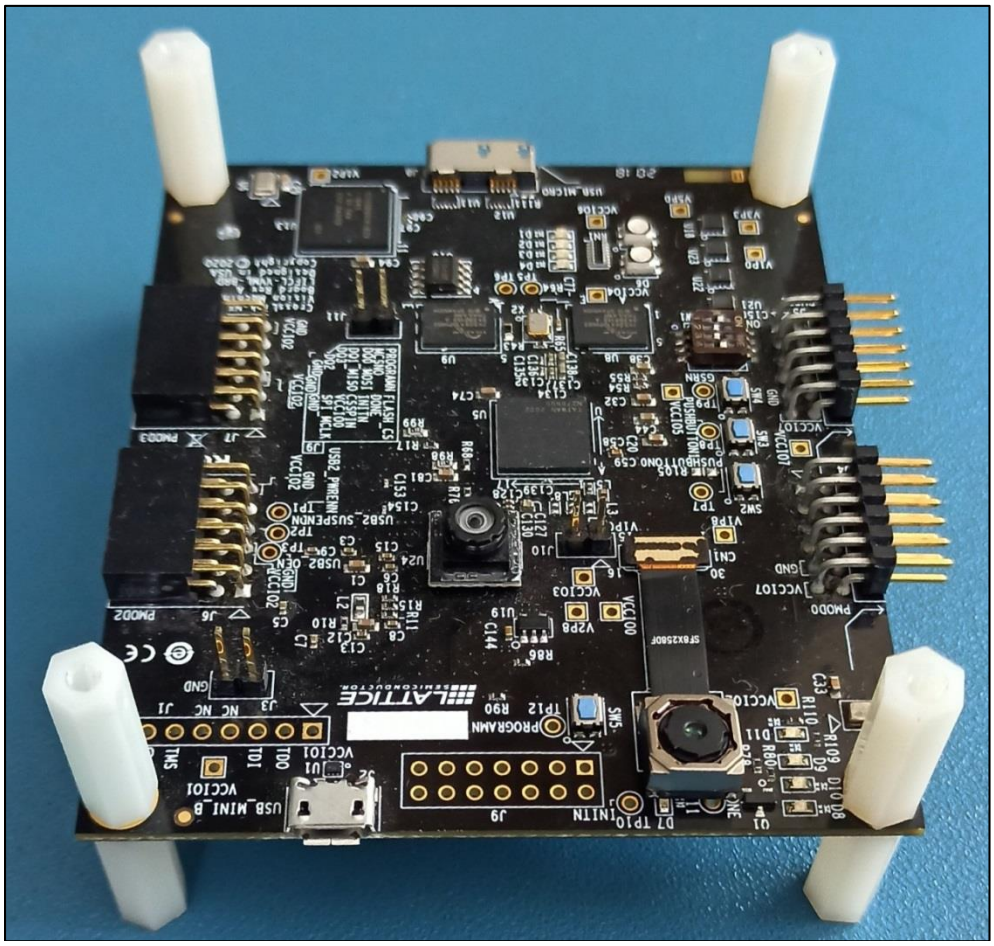

Figure 2.1. Lattice CrossLink-NX Voice and Vision Board

2.2. Setting Up the Linux Environment for Machine Training

2.2.1. 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. CUDA Repo Download

```
$ 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. CUDA Repo Installation

```
$ 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
```

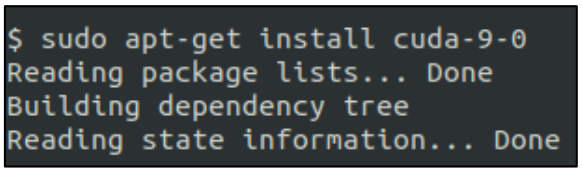

Figure 2.6. CUDA Installation Completed

2.2.2. Installing the cuDNN

To install the cuDNN:

1. Create an NVIDIA developer account in https://developer.nvidia.com.
2. Download cuDNN lib in 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 the commands below 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
$ sudo chmod a+r/usr/local/cuda/include/cudnn.h/usr/local/cuda/lib64/libcudnn*
```

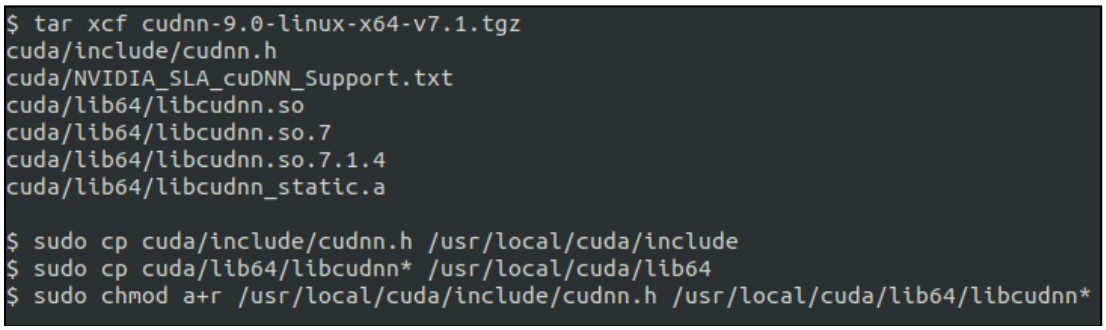

Figure 2.7. cuDNN Library Installation

2.2.3. Installing Anaconda and Python 3

To install Anaconda and Python 3:

1. Go to https://www.anaconda.com/distribution/#download-section.
2. Download Python 3 version of Anaconda for Linux.
3. Install the Anaconda environment by running the command below:

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

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

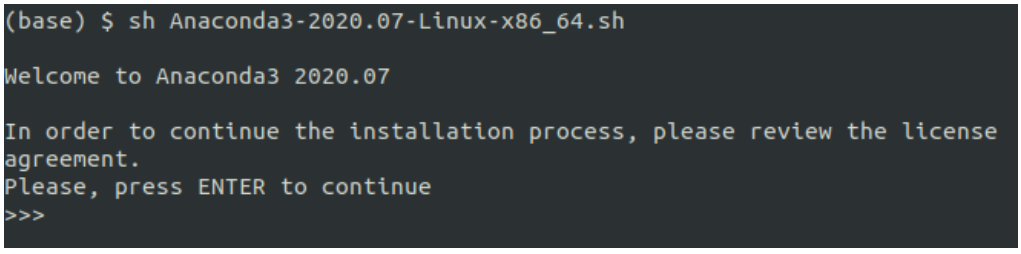

Figure 2.8. Anaconda Installation

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 onscreen 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

2.2.4. Installing TensorFlow v1.14

To install TensorFlow v1.14:

1. Activate the conda environment by running the command below:

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

```
$ source anaconda3/bin/activate
(base) ~$
```

Figure 2.12. Anaconda Environment Activation

2. Install the TensorFlow by running the command example below:

```
$ conda install tensorflow-gpu==1.14.0
```

```
(base) $ conda install tensorflow-gpu==1.14.0
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - tensorflow-gpu==1.14.0
```

Figure 2.13. TensorFlow Installation

3. After installation, enter **Y**, as shown in Figure 2.14.

```
The following NEW packages will be INSTALLED:

  _tflow_select      pkgs/main/linux-64::_tflow_select-2.1.0-gpu
  tensorboard        pkgs/main/linux-64::tensorboard-1.14.0-py36hf484d3e_0
  tensorflow         pkgs/main/linux-64::tensorflow-1.14.0-gpu_py36h3fb9ad6_0
  tensorflow-base    pkgs/main/linux-64::tensorflow-base-1.14.0-gpu_py36he45bfe2_0
  tensorflow-estima~ pkgs/main/noarch::tensorflow-estimator-1.14.0-py_0
  tensorflow-gpu     pkgs/main/linux-64::tensorflow-gpu-1.14.0-h0d30ee6_0


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

Figure 2.14. TensorFlow Installation Confirmation

Figure 2.15 shows that the TensorFlow installation is complete.

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Figure 2.15. TensorFlow Installation Completion

2.2.5. Installing the Python Package

To install the Python package:

1. Install Easydict by running the command below:

```
$ conda install -c conda-forge easydict
```

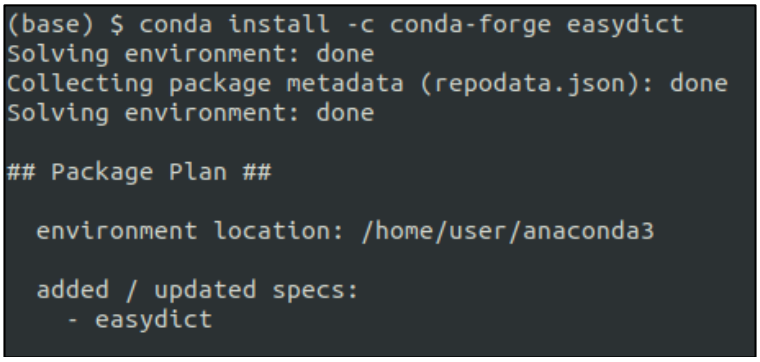

Figure 2.16. Easydict Installation

2. Install Joblib by running the command below:

```
$ conda install joblib
```

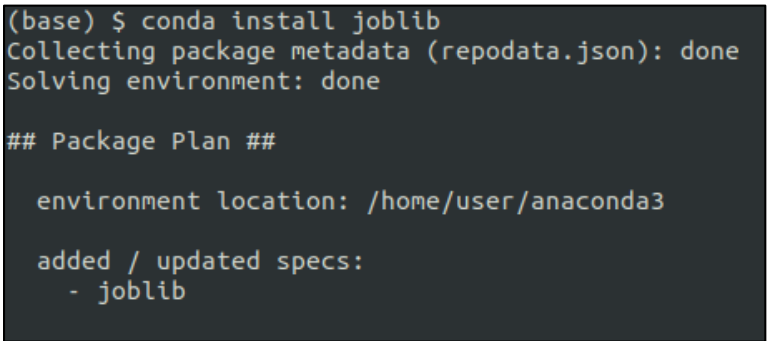

Figure 2.17. Joblib Installation

3. Install Keras by running the command below:

```
$ conda install keras
```

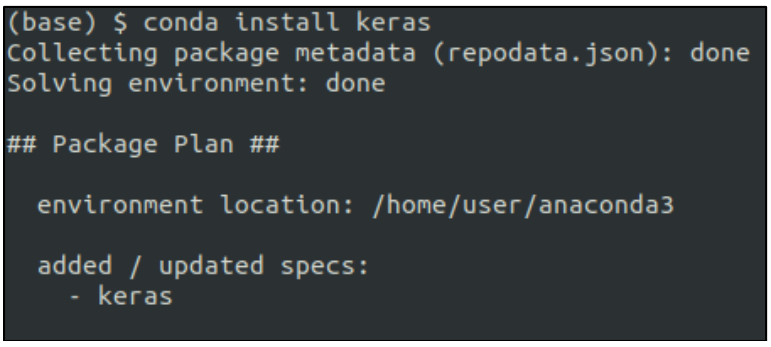

Figure 2.18. Keras Installation

4. Install OpenCV by running the command below:

```
$ conda install opencv
```

```
(base) $ conda install opencv
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - opencv
```

Figure 2.19. OpenCV Installation

5. Install Pillow by running the command below:

```
$ conda install pillow
```

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

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - pillow
```

Figure 2.20. Pillow Installation

3. Preparing the Dataset

This section describes how to create a dataset using Google Open Image Dataset as an example.

The Google Open Image Dataset version 4 (https://storage.googleapis.com/openimages/web/index.html) features more than 600 classes of images. The Person class of images includes human annotated and machine annotated labels and bounding box. Annotations are licensed by Google Inc. under CC BY 4.0 and images are licensed under CC BY 2.0.

3.1. Downloading the Dataset

To download the dataset, run the commands below:

1. Clone the OIDv4_Toolkit repository:

```
$ git clone https://github.com/EscVM/OIDv4_ToolKit.git
$ cd OIDv4_ToolKit
```

```
(base) k$ git clone https://github.com/EscVM/OIDv4_ToolKit.git
Cloning into 'OIDv4_ToolKit'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 382 (delta 3), reused 14 (delta 1), pack-reused 357
Receiving objects: 100% (382/382), 34.06 MiB | 752.00 KiB/s, done.
Resolving deltas: 100% (111/111), done.
(base) k$
```

Figure 3.1. Open Source Dataset Repository Cloning

Figure 3.2 shows the OIDv4 code directory structure.

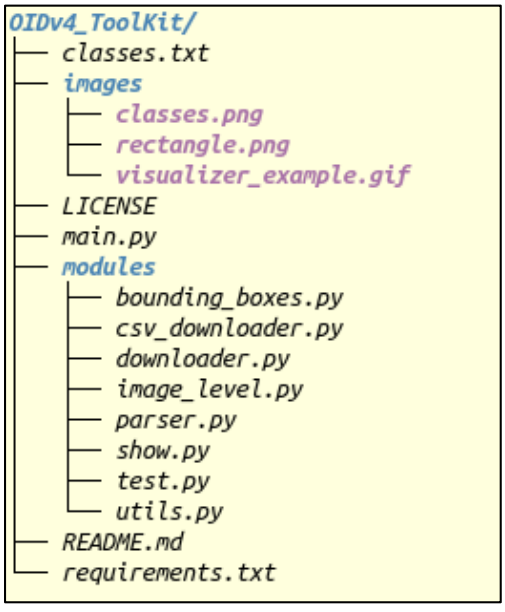

Figure 3.2. OIDv4_Toolkit Directory Structure

View the OIDv4 Toolkit Help menu:

```
$ python3 main.py -h
```

```
(base) k$ python3 main.py -h
usage: main.py [-h] [--Dataset /path/to/OID/csv/]
               [--classes list of classes [list of classes ...]]
               [--type_csv 'train' or 'validation' or 'test' or 'all']
               [--sub Subset of human verified images or machine generated h or m)]
               [--image_IsOccluded 1 or 0] [--image_IsTruncated 1 or 0]
               [--image_IsGroupOf 1 or 0] [--image_IsDepiction 1 or 0]
               [--image_IsInside 1 or 0] [--multiclasses 0 (default or 1]
               [--n_threads [default 20]] [--noLabels]
               [--limit integer number]
               <command> 'downloader', 'visualizer' or 'ill_downloader'.

Open Image Dataset Downloader

positional arguments:
  <command> 'downloader', 'visualizer' or 'ill_downloader'.
                        'downloader', 'visualizer' or 'ill_downloader'.
```

Figure 3.3. Dataset Script Option/Help

2. Use the OIDv4 Toolkit to download dataset. Download the Person class images:

```
$ python3 main.py downloader --classes Person --type_csv validation
```

```
(base) k$ python3 main.py downloader --classes Person --type_csv validation --limit 200
```

Figure 3.4. Dataset Downloading Logs

Figure 3.5 shows the downloaded dataset directory structure.

```
OID
├── csv_folder
│   ├── class-descriptions-boxable.csv
│   └── validation-annotations-bbox.csv
└── Dataset
    └── validation
        └── Person
            ├── ff7b4cc8ca9b6592.jpg
            └── Label
                └── ff7b4cc8ca9b6592.txt
```

Figure 3.5. Downloaded Dataset Directory Structure

3. Lattice training code uses KITTI (.txt) format. Since the downloaded dataset is not in exact KITTI format, convert the annotation using the code below.

```
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/validation/Person/Label/*
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/train/Person/Label/*
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/test/Person/Label/*
```

```
(base) k$ cat OID/Dataset/validation/Person/Label/ff7b4cc8ca9b6592.txt
Person 324.614144 69.905733 814.569472 681.9072
(base) k$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/validation/Person/Label/*
(base) k$ cat OID/Dataset/validation/Person/Label/ff7b4cc8ca9b6592.txt
Person 0 0 0 324.614144 69.905733 814.569472 681.9072
(base) k$
```

Figure 3.6. OIDv4 Label to KITTI Format Conversion

Note:

KITTI Format: Person 0 0 0 324.61 69.90 814.56 681.90

The format includes class ID followed by truncated, occluded, alpha, Xmin, Ymin, Xmax, Ymax.

The code converts Xmin, Ymin, Xmax, Ymax into x, y, w, h while training as *bounding box rectangle coordinates*.

3.2. Visualizing and Tuning/Cleaning Up the Dataset

To visualize and annotate the dataset, run the commands below:

1. Visualize the labeled images.

```
$ python3 main.py visualizer
```

Figure 3.7. Toolkit Visualizer

2. Clone the manual annotation tool from the GitHub repository.

```
$ git clone https://github.com/SaiPrajwal95/annotate-to-KITTI.git
```

```
(base) k$ git clone https://github.com/SaiPrajwal95/annotate-to-KITTI.git
Cloning into 'annotate-to-KITTI'...
remote: Enumerating objects: 27, done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 27
Unpacking objects: 100% (27/27), done.
(base) k$ _
```

Figure 3.8. Manual Annotation Tool – Cloning

3. Go to annotate to KITTI.

```
$ cd annotate-to-KITTI
$ ls
```

```
annotate-to-KITTI/
├── annotate-folder.py
└── README.md
```

Figure 3.9. Manual Annotation Tool – Directory Structure

4. Install the dependencies (OpenCV 2.4).

```
$ sudo apt-get install python-opencv
```

5. Launch the utility.

```
$ python3 annotate-folder.py
```

6. Set the dataset path and default object label.

```
(base) k$ python3 annotate-folder.py
Enter the path to dataset: /tmp/images
Enter default object label: Person
[{'label': 'Person', 'bbox': {'xmin': 443, 'ymin': 48, 'xmax': 811, 'ymax': 683}}]
(base) k$ _
```

Figure 3.10. Manual Annotation Tool – Launch

7. For annotation, run the script provided in the website below.

   https://github.com/SaiPrajwal95/annotate-to-KITTI

   For information on other labeling tools, see Table A.1.

3.3. Data Augmentation

Data Augmentation needs a large amount of training data to achieve good performance. Image Augmentation creates training images through different ways of processing or a combination of multiple processing such as random rotation, shifts, shear and flips, and others.

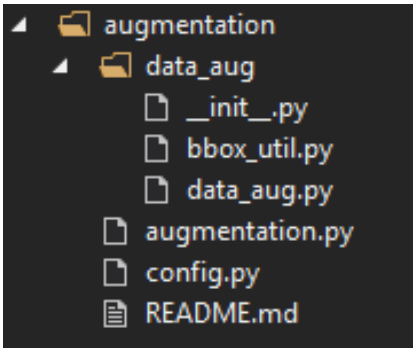

Figure 3.11. Augmentation Directory Stucture

- data_aug – This folder contains basic methods and augmentation classes.
- augmentation.py – This file reads the input images (input labels) and performs preferred augmentation on it.
- config.py – Contains parameters that are used in augmentation operations.

3.3.1. Configuring the Augmentation

To configure the augmentation:

1. Configure the *config.py* file, which contains the parameters shown in Figure 3.12.

```
Input_dict = {
    'AngleForRotation': '90,190,270',
    'GammaForRandomBrightness1': 0.6,
    'GammaForRandomBrightness2': 1.5,
    'FilterSizeForGaussianFiltering': 11,
    'SnowCoeffForAddSnow': 0.5,
    'resizeheight': 224,
    'resizewidth': 224,
    }
```

Figure 3.12. config.py Configuration File Parameters

Choose the operations to perform on the dataset. The operations can be selected in *augmentation.py* by editing the list *all_op*.

```
all_op = [
    'RandomHorizontalFlip',
    #'RandomScale',
    #'RandomRotate',
    #'RandomTranslate',
    #'RandomShear',
    #'Rotate',
    'Translate',
    #'Shear',
    #'GaussianFiltering',
    'RandomBrightness2_0',
    'RandomBrightness0_5',
    #'Resize'
]
```

Figure 3.13. Selecting the Augmentation Operations

2. Add or Remove the operation by commenting/uncommenting the operation in the *all_op* list as shown in Figure 3.13.

3.3.2. Running the Augmentation

Run the augmentation by running the following command:

```
python augmentation.py --image_dir <Path_To_InputImage_Dir> --label_dir
<Path_To_InputLabel_Dir> --out_image_dir <Path_To_OutputImage_Dir> --out_label_dir
<Path_To_OutputLable_Dir>
```

```
earth:~$ python augmentation.py --image_dir /data/images/ --label_dir /data/labels/ --out_image_dir /tmp/labels --out_label_dir /tmp/images/

| ---------------------------------------- |
| Initial Configuration
| ---------------------------------------- |
| AngleForRotation                    | 90,190,270 |
| GammaForRandomBrightness1           | 0.6 |
| GammaForRandomBrightness2           | 1.5 |
| FilterSizeForGaussianFiltering      | 11 |
| SnowCoeffForAddSnow                 | 0.5 |
| resizeheight                        | 224 |
| resizewidth                         | 224 |
| ---------------------------------------- |

Copying original images and labels ...
Done Coping

Running Augmentation...!!!
operation : RandomHorizontalFlip

Operation number  1 out of 4
ignoring image due to small bbox size for /data/images/a5a886c21a52344f.jpg
```

Figure 3.14. Running the Augmentataion

4. Training the Machine

4.1. Training Code Structure

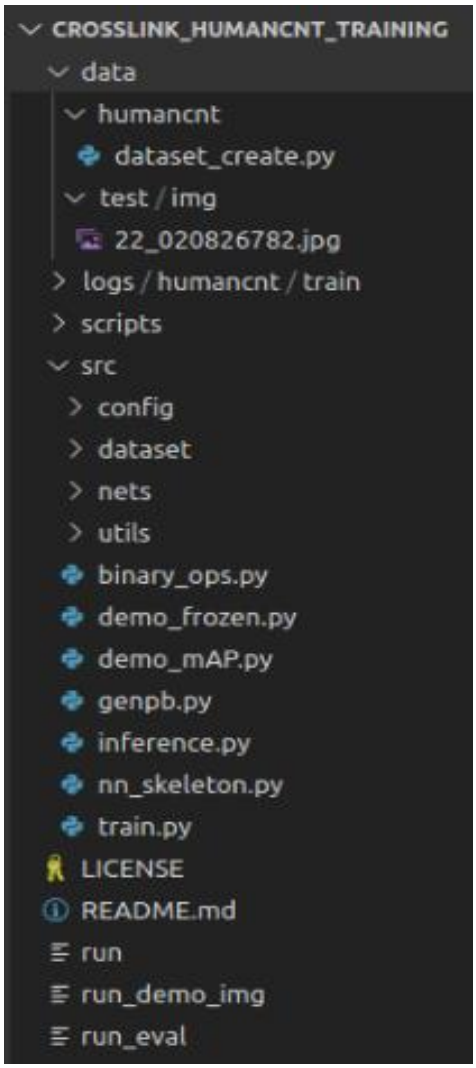

Figure 4.1. Training Code Directory Structure

4.2. Neural Network Architecture

4.2.1. Human Count Training Network Layers

This section provides information on the Convolution Network Configuration of the Human Counting Detection design. The Neural Network model of the Human Count Detection design uses the VGG Neural Network base model and the detection layer of the SqueezeDet model.

Table 4.1. Human Counting Training Network Topology

Image Input (224 × 224 × 1)		
Fire 1	Conv3–32	Conv3 - # where: • *Conv3* = 3 × 3 Convolution filter Kernel size • *#* = The number of filter For example, Conv3 - 16 = 16 3 × 3 convolution filters BN – Batch Normalization
	BN	
	ReLU	
	MaxPool	
Fire 2	Conv3–32	
	BN	
	ReLU	
Fire 3	Conv3–32	
	BN	
	ReLU	
	MaxPool	
Fire 4	Conv3–64	
	BN	
	ReLU	
Fire 5	Conv3–64	
	BN	
	ReLU	
	MaxPool	
Fire 6	Conv3–128	
	BN	
	ReLU	
Fire 7	Conv3–128	
	BN	
	ReLU	
	MaxPool	
Conv12	Conv3–42	

- The Human Count Network structure consists of seven fire layers followed by one convolution layer. A fire layer contains Convolutional, Batch Normalization, and ReLU (Rectified Linear Unit). Pooling layers are only in Fire 1, Fire 3, Fire 5, and Fire 7. Fire 2, Fire 4, and Fire 6 do not contain pooling layers.
- In Table 4.1, the layer contains Convolution (Conv), Batch Normalization (BN), and ReLU layers.
- Layer information:
  - Convolutional Layer
    In general, the first layer in a CNN is always a convolutional layer**.** Each layer consists of a number of filters (sometimes referred as kernels), which convolves with the input layer/image and generates an activation map (that is, feature map). This filter is an array of numbers (called weights or parameters). Each of these filters can be thought of as feature identifiers, such as straight edges, simple colors, curves, and 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 it is looking for (such as curve) is in the input volume.

- ReLU (Activation Layer)
  It is the convention to apply a nonlinear layer (or activation layer) immediately after each conv layer. The purpose of this layer is to introduce nonlinearity to a system that is basically computing linear operations during the conv layers (element wise multiplications and summations). In the past, nonlinear functions such as 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 in 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 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, you 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 2 × 2) and a stride of the same length. It then applies a filter to the input volume and outputs the maximum number in every sub region that the filter convolves around.
  The intuitive reason behind this layer is that once it is known that a specific feature is in the original input volume (there is a high activation value), its exact location is not as important as its relative location to 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 controls over fitting. This term is used 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.

- Batch Normalization Layer
  Batch normalization layer reduces the internal covariance shift. To train a neural network, some preprocessing to the input data are performed. For example, you can normalize all data so that it resembles a normal distribution (which means zero mean and a unitary variance). This prevents the early saturation of non-linear activation functions, such as sigmoid, and assures that all input data are in the same range of values.
  An issue, however, 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 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 the process below during training:
  a. Calculate the mean and variance of the layers input.
  b. Normalize the layer inputs using the previously calculated batch statistics.
  c. Scale and shift 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.

The architecture above provides nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting.

4.2.2. Human Count Detection Network Output

From the input image model, it extracts the feature maps first and overlays them with a W × H grid. Each cell then computes K pre-computed bounding boxes called anchors. Each bounding box has the following:

- Four scalars (x, y, w, h)
  - A confidence score (Pr(Object)*IOU)
  - C° conditional class probability
- The current model architecture has a fixed output of *WxHxK(4+1+C)*. where:
  - W, H = Grid Size
  - K = Number of Anchor boxes
  - C = Number of classes for which you want detection
- The model has a total of 8232 output values, which are derived from the following:
  - 14 × 14 grid
    - Seven anchor boxes per grid
      - Six values per anchor box. It consists of:
        - Four bounding box coordinates (x, y, w, h)
        - One class probability
        - One confidence score

As a result, there is a total of 14 × 14 × 7 × 6 = 8232 output values.

If your images are smaller, it is recommended to stretch the image to default size. You can also up-sample them beforehand.

If your images are bigger and you are not satisfied with the results of the default image size, you can try using a denser grid, as details might get lost during the downscaling.

4.2.3. Training Code Overview

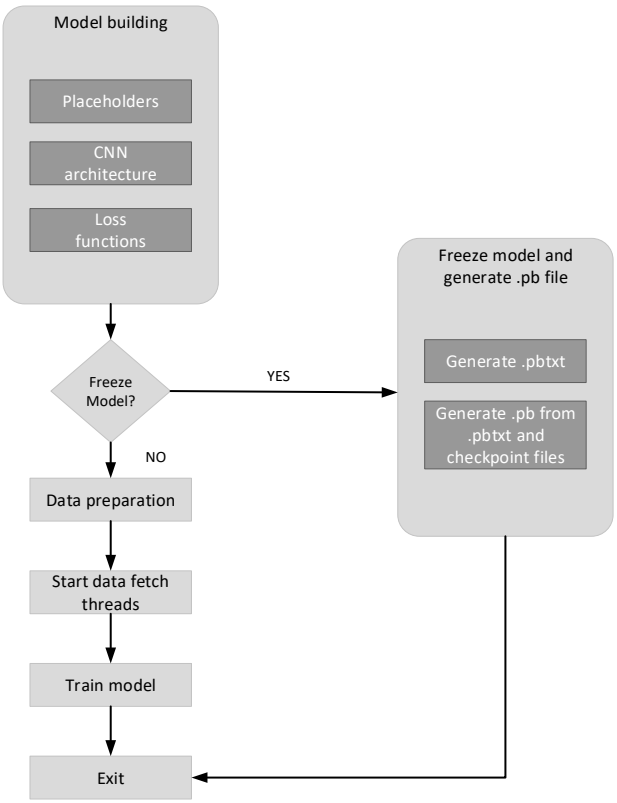

Figure 4.2. Training Code Flow Diagram

Training Code is divided into the following parts:

- Model Configuration
- Model Building
- Model Freezing
- Data Preparation
- Training for Overall Execution Flow

The details of each part can be found in subsequent sections.

4.2.3.1. Model Configuration

The design uses Kitti dataset and SqueezeDet model. *kitti_squeezeDet_config()* maintains all the configurable parameters for the model. Below is the summary of configurable parameters.

- Image size
  - Change mc.IMAGE_WIDTH and mc.IMAGE_HEIGHT to configure image size (width and height) in *src/config/kitti_squeezeDet_config.py.*

```
mc.IMAGE_WIDTH              = 224
mc.IMAGE_HEIGHT             = 224
```

Figure 4.3. Code Snippet – Input Image Size Config

  - Since there are four pooling layers, grid dimension is H = 14 and W = 14. *anchor_shapes* variable of *set_anchors()* in *src/config/kitti_squeezeDet_config.py* indicates anchors width and heights. Update it based on anchors per gird size changes.

```
def set_anchors(mc):
  H, W, B = 14, 14, 7
  div_scale = 2.0 * 1
```

Figure 4.4. Code Snippet – Anchors Per Grid Config #1 (Grid Sizes)

- Batch size
  - Change *mc.BATCH_SIZE* in *src/config/kitti_squeezeDet_config.py* to configure batch size.
- Anchors per grid
  - Change *mc.ANCHOR_PER_GRID* in *src/config/kitti_squeezeDet_config.py* to configure anchors per grid.

```
mc.ANCHOR_BOX               = set_anchors(mc)
mc.ANCHORS                  = len(mc.ANCHOR_BOX)
mc.ANCHOR_PER_GRID          = 7
```

Figure 4.5. Code Snippet – Anchors Per Grid Config #2

  - Change hard coded anchors per grid in *set_anchors()* in *src/config/kitti_squeezeDet_config.py*. Here, B (value 7) indicates anchors per grid.
  - To run the network on your own dataset, adjust the anchor sizes. Anchors are prior distribution over what shapes your boxes should have. The better this fits to the true distribution of boxes, the faster and easier your training is going to be.
  - To determine anchor shapes, first load all ground truth boxes and pictures, and if your images are not of the same size, normalize their height and width by the images' height and width. All images are normalized before being fed to the network, so you need to do the same to the bounding boxes and consequently, the anchors.
  - Second, perform a clustering on these normalized boxes (that is, you can use k-means without feature whitening and determine the number of clusters either by eyeballing or by using the elbow method.)

- Check for boxes that extend beyond the image or have a zero to negative width or height.

```
anchor_shapes = np.reshape(
    [np.array(
        [
     [int(368./div_scale), int(368./div_scale)],
     [int(276./div_scale), int(276./div_scale)],
     [int(184./div_scale), int(184./div_scale)],
     [int(138./div_scale), int(138./div_scale)],
        [int( 92./div_scale), int( 92./div_scale)],
        [int( 69./div_scale), int( 69./div_scale)],
     [int( 46./div_scale), int( 46./div_scale)]])] * H * W,
    (H, W, B, 2)
)
```

Figure 4.6. Code Snippet – Anchors Per Grid Config #3

- Training Parameters
  - Other training related parameters such as learning rate, loss parameters, and different thresholds can be configured from *src/config/kitti_squeezeDet_config.py*.

```
mc.WEIGHT_DECAY           = 0.0001
mc.LEARNING_RATE          = 0.01
mc.DECAY_STEPS            = 10000
mc.MAX_GRAD_NORM          = 1.0
mc.MOMENTUM               = 0.9
mc.LR_DECAY_FACTOR        = 0.5

mc.LOSS_COEF_BBOX         = 5.0
mc.LOSS_COEF_CONF_POS     = 75.0
mc.LOSS_COEF_CONF_NEG     = 100.0
mc.LOSS_COEF_CLASS        = 1.0

mc.PLOT_PROB_THRESH       = 0.4
mc.NMS_THRESH             = 0.4
mc.PROB_THRESH            = 0.005
mc.TOP_N_DETECTION        = 10

mc.DATA_AUGMENTATION      = True
mc.DRIFT_X                = 150
mc.DRIFT_Y                = 100
mc.EXCLUDE_HARD_EXAMPLES  = False
```

Figure 4.7. Code Snippet – Training Parameters

4.2.3.2. Model Building

SqueezeDet class can be configured from *src/nets/squeezeDet.py*. SqueezeDet class constructor builds the model, which is divided into the following sections:

- Forward Graph
- Interpretation Graph
- Loss Graph
- Train Graph
- Visualization Graph
- Forward Graph

Forward Graph

- The CNN architecture consists of Convolution, Batch Normalization, ReLU, and MaxPool.
- Forward graph consists of seven fire layers as described in Table 4.1.

```
depth = [32, 32, 32, 64, 64, 128, 128]

mul_f = 1
#depth = [ 8*mul_f,  8*mul_f, 16*mul_f, 16*mul_f, 24*mul_f, 24*mul_f, 32*mul_f]

##########################################################################
# Quantization layers
##########################################################################
if True: # 16b weight (no quant); 8b activation
    fl_w_bin = 8
    fl_a_bin = 8
    ml_w_bin = 8
    ml_a_bin = 8
    sl_w_bin = 8
    # The last layer's activation (sl_a_bin) is always 16b

    min_rng =  0.0 # range of quanized activation
    max_rng =  2.0
```

Figure 4.8. Code Snippet – Filter Values

- Filter sizes of each convolutional block are mentioned in Table 4.1, which can be configured by changing the values of depth, as shown in Figure 4.9.

```
fire1 = self._fire_layer('fire1', self.image_input, oc=depth[0], freeze=False, w_bin=fl_w_bin, a_bin=fl_a_bin,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire2 = self._fire_layer('fire2', fire1,            oc=depth[1], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire3 = self._fire_layer('fire3', fire2,            oc=depth[2], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire4 = self._fire_layer('fire4', fire3,            oc=depth[3], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire5 = self._fire_layer('fire5', fire4,            oc=depth[4], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire6 = self._fire_layer('fire6', fire5,            oc=depth[5], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire7 = self._fire_layer('fire7', fire6,            oc=depth[6], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, mul_f=mul_f)
fire_o = fire7
```

Figure 4.9. Code Snippet – Forward Graph Fire Layers

```
num_output = mc.ANCHOR_PER_GRID * (mc.CLASSES + 1 + 4)
self.preds = self._conv_layer('conv12', fire_o, filters=num_output, size=3, stride=1,
    padding='SAME', xavier=False, relu=False, stddev=0.0001, w_bin=sl_w_bin, bias_on=bias_on)
print('self.preds:', self.preds)
```

Figure 4.10. Code Snippet – Forward Graph Last Convolution Layer

Interpretation Graph

- The Interpretation Graph consists of the following sub-blocks:
  - interpret_output
    This block interprets output from network and extracts predicted class probability, predicated confidence scores, and bounding box values.
    Output of the convnet is a 14 × 14 × 42 tensor – there are 42 channels of data for each of the cells in the grid that is overlaid on the image and contains the bounding boxes and class predictions. This means the 42 channels are not stored consecutively but are scattered all over and need to be sorted. Figure 4.11 and Figure 4.12 show the details.

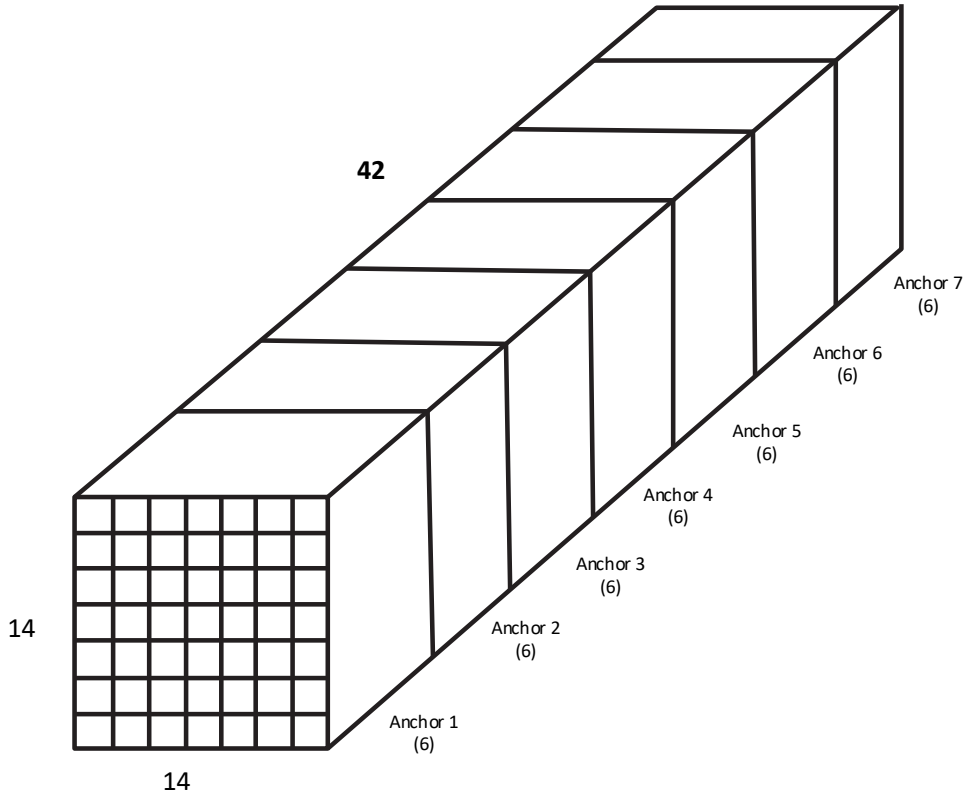

Figure 4.11. Grid Output Visualization #1

For each grid, cell values are aligned as shown in Figure 4.12.

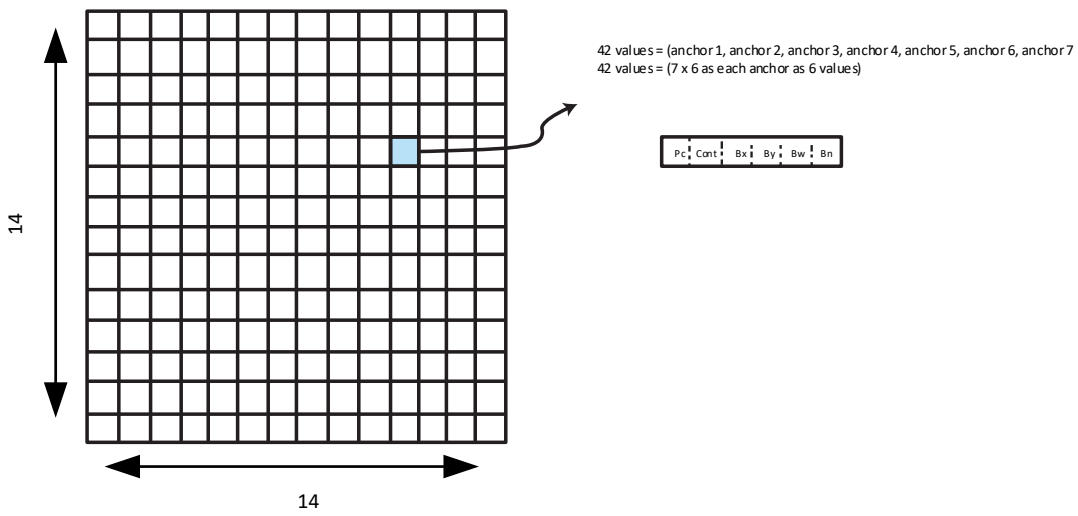

Figure 4.12. Grid Output Visualization #2

Figure 4.13 shows the output from the conv12 layer (4D array of batch size × 14 × 14 × 42) that needs to be sliced with the proper index to get all values of probability, confidence, and coordinates.

```
# confidence
num_confidence_scores = mc.ANCHOR_PER_GRID
self.pred_conf = tf.sigmoid(
    tf.reshape(
        preds[:, :, :, :num_confidence_scores],
        [mc.BATCH_SIZE, mc.ANCHORS]
    ),
    name='pred_confidence_score'
)

# probability
num_class_probs = mc.ANCHOR_PER_GRID*mc.CLASSES+num_confidence_scores

self.pred_class_probs = tf.reshape(
    tf.nn.softmax(
        tf.reshape(
            preds[:, :, :, num_confidence_scores:num_class_probs],
            [-1, mc.CLASSES]
        )
    ),
    [mc.BATCH_SIZE, mc.ANCHORS, mc.CLASSES],
    name='pred_class_probs'
)

# bbox_delta
self.pred_box_delta = tf.reshape(
    preds[:, :, :, num_class_probs:],
    [mc.BATCH_SIZE, mc.ANCHORS, 4],
    name='bbox_delta'
)
```

Figure 4.13. Code Snippet – Interpret Output Graph

For confidence score, this must be a number between 0 and 1, as such, sigmoid is used.
For predicting the class probabilities, there is a vector of NUM_CLASS values at each bounding box. Apply a softmax to make it probability distribution.

- bbox
  This block calculates bounding boxes based on the anchor box and the predicated bounding boxes.
- IOU
  This block calculates Intersection over Union for detected bounding boxes and actual bounding boxes.
- Probability
  This block calculates detection probability and object class.

Loss Graph

- This block calculates different types of losses, which needs to be minimized. To learn detection, localization, and classification, model defines a multi-task loss function. There are three types of losses which are considered for calculation:
  - Bounding Box
    This loss is regression of the scalars for the anchors.

```
with tf.variable_scope('bounding_box_regression') as scope:
  self.bbox_loss = tf.truediv(
      tf.reduce_sum(
          mc.LOSS_COEF_BBOX * tf.square(
              self.input_mask*(self.pred_box_delta-self.box_delta_input))),
      self.num_objects,
      name='bbox_loss'
  )
  tf.add_to_collection('losses', self.bbox_loss)
```

Figure 4.14. Code Snippet – Bbox Loss

  - Confidence Score
    - To obtain meaningful confidence score, the predicted value of each box is regressed against the real and predicted box. During training, compare the ground truth bounding boxes with all anchors and assign them to the anchors with the largest overlap (IOU).
    - Select the *closest* anchor to match the ground truth box such that the transformation needed is reduced to minimum. Equation evaluates to 1 if the k-th anchor at position-(i, j) has the largest overlap with a ground truth box, and to 0 if no ground truth is assigned to it. This way, you only include the loss generated by the *responsible* anchors.
    - As there can be multiple objects per image, normalize the loss by dividing it by the number of objects (self.num_objects).

```
with tf.variable_scope('confidence_score_regression') as scope:
    input_mask = tf.reshape(self.input_mask, [mc.BATCH_SIZE, mc.ANCHORS])
    self.conf_loss = tf.reduce_mean(
        tf.reduce_sum(
            tf.square((self.ious - self.pred_conf))
            * (input_mask*mc.LOSS_COEF_CONF_POS/self.num_objects
                +(1-input_mask)*mc.LOSS_COEF_CONF_NEG/(mc.ANCHORS-self.num_objects)),
            reduction_indices=[1]
        ),
        name='confidence_loss'
    )
```

Figure 4.15. Code Snippet – Confidence Loss

- Class
  - The last part of the loss function is cross-entropy loss for each box to do classification, as you would for image classification.

```
with tf.variable_scope('class_regression') as scope:
    # cross-entropy: q * -log(p) + (1-q) * -log(1-p)
    # add a small value into log to prevent blowing up
    self.class_loss = tf.truediv(
        tf.reduce_sum(
            (self.labels*(-tf.log(self.pred_class_probs+mc.EPSILON))
            + (1-self.labels)*(-tf.log(1-self.pred_class_probs+mc.EPSILON)))
            * self.input_mask * mc.LOSS_COEF_CLASS),
        self.num_objects,
        name='class_loss'
    )
    tf.add_to_collection('losses', self.class_loss)
```

Figure 4.16. Code Snippet – Class Loss

In one model architecture, you obtain the bounding box prediction, the classification, as well as the confidence score.

Train Graph

- This block is responsible for training the model with momentum optimizer to reduce all losses.

Visualization Graph

- This block provides visitations of detected results.

4.2.3.3. Training

```
if mc.NUM_THREAD > 0:
  _, loss_value, conf_loss, bbox_loss, class_loss = sess.run(
      [model.train_op, model.loss, model.conf_loss, model.bbox_loss,
       model.class_loss], options=run_options)
else:
  feed_dict, _, _, _, _ = _load_data(load_to_placeholder=False)
  _, loss_value, conf_loss, bbox_loss, class_loss = sess.run(
      [model.train_op, model.loss, model.conf_loss, model.bbox_loss,
       model.class_loss], feed_dict=feed_dict)
```

Figure 4.17. Code Snippet – Training

sess.run feeds the data, labels batches to network, and optimizes the weights and biases. The code above handles the input data method in case of multiple threads preparing batches, or data preparation in the main thread.

4.3. Training from Scratch and/or Transfer Learning

To train the machine:

1. Go to the top/root directory of the Lattice training code from the command prompt.

   The model works on 224 × 224 images.

   Current human count training code uses mean = 0 and scale = 1/128 (0.0078125) in pre-processing step. Mean and scale can be changed in training code *@src/dataset/imdb.py* as shown in Figure 4.18.

```
v = np.where(v <= 255 - add_v, v + add_v, 255)
final_hsv = cv2.merge((h, s, v))
im = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)

im -= mc.BGR_MEANS #  ------------------------------
im /= 128.0 # to make input in the range of [0, 2)
orig_h, orig_w, _ = [float(v) for v in im.shape]

# load annotations
label_per_batch.append([b[4] for b in self._rois[idx][:]])
```

Figure 4.18. Training Code Snippet for Mean and Scale

   The dataset path can be set in the training code *@src/dataset/kitti.py* and can be used in combination with the --*data_path* option while triggering training using *train.py* to get the desired path. For example, you can have *<data_path>/training/images* and *<data_path>/training/labels*.

```
def __init__(self, image_set, data_path, mc):
  imdb.__init__(self, 'kitti_'+image_set, mc)
  self._image_set = image_set
  self._data_root_path = data_path
  self._image_path = os.path.join(self._data_root_path, 'training', 'images')
  self._label_path = os.path.join(self._data_root_path, 'training', 'labels')
  self._classes = self.mc.CLASS_NAMES
```

Figure 4.19. Training Code Snippet for Dataset Path

2. Create a train.txt file.

```
$ cd data/humancnt/
$ python dataset_create.py
```

Figure 4.20. Create File for Dataset train.txt

Notes:

- train.txt – file name of dataset images
- image_set – train (ImageSets/train.txt)
- data_path – $ROOT/data/humancnt/
  - Images – $ROOT/data/humancnt/images
  - Annotations – $ROOT/data/humancnt/label

3. Modify the training script. *@scripts/train.sh* is used to trigger training.
   Figure 4.21 shows the input parameters, which can be configured.

```
python3 ./src/train.py \
  --dataset=KITTI \
  --pretrained_model_path=$PRETRAINED_MODEL_PATH \
  --data_path=$TRAIN_DATA_DIR \
  --image_set=$IMAGE_SET \
  --train_dir="$TRAIN_DIR/train" \
  --net=$NET \
  --summary_step=100 \
  --checkpoint_step=500 \
  --max_steps=250000 \
  --gpu=$GPUID
```

Figure 4.21. Training Input Parameter

- $TRAIN_DATA_DIR – dataset directory path. */data/humancnt* is an example.
- $TRAIN_DIR – log directory where checkpoint files are generated while model is training.
- $GPUID – gpu id. If the system has more than one gpu, it indicates the one to use.
- --summary_step – indicates at which interval loss summary should be dumped.
- --checkpoint_step – indicates at which interval checkpoints are created.
- --max_steps – indicates the maximum number of steps for which the model is trained.

4. Execute the run command script which starts training.

```
earth:$ ./run
self.preds: Tensor("conv12/convolution:0", shape=(20, 14, 14, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
preds2: Tensor("interpret_output/strided_slice:0", shape=(20, 14, 14, 7), dtype=float32, device=/device:GPU:0)
ANCHORS: 1372
max person: 22
Model statistics saved to ./logs/humancnt/train/model_metrics.txt.
conf_loss: 27.452621459960938, bbox_loss: 10.86892318725586, class_loss: 0.0
2020-03-27 16:31:58.203003: step 0, loss = 38.32 (6.7 images/sec; 2.978 sec/batch)
2020-03-27 16:32:06.568379: step 10, loss = 11.98 (30.6 images/sec; 0.654 sec/batch)
2020-03-27 16:32:13.121694: step 20, loss = 2.76 (30.8 images/sec; 0.650 sec/batch)
2020-03-27 16:32:19.651650: step 30, loss = 2.04 (30.3 images/sec; 0.660 sec/batch)
2020-03-27 16:32:26.894056: step 40, loss = 2.10 (29.4 images/sec; 0.680 sec/batch)
2020-03-27 16:32:34.271516: step 50, loss = 1.98 (27.5 images/sec; 0.728 sec/batch)
2020-03-27 16:32:41.618091: step 60, loss = 2.07 (27.5 images/sec; 0.728 sec/batch)
2020-03-27 16:32:49.307227: step 70, loss = 2.10 (29.5 images/sec; 0.677 sec/batch)
```

Figure 4.22. Execute Run Script

5. Start TensorBoard.

```
$ tensorboard –logdir=<log directory of training>
```

For example: tensorboard –logdir='./logs/'

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

```
earth:$ tensorboard --logdir logs/humancnt/train
TensorBoard 1.12.0 at http://earth:6006 (Press CTRL+C to quit)
```

Figure 4.23. TensorBoard – Generated Link

7. Check the training status on TensorBoard.

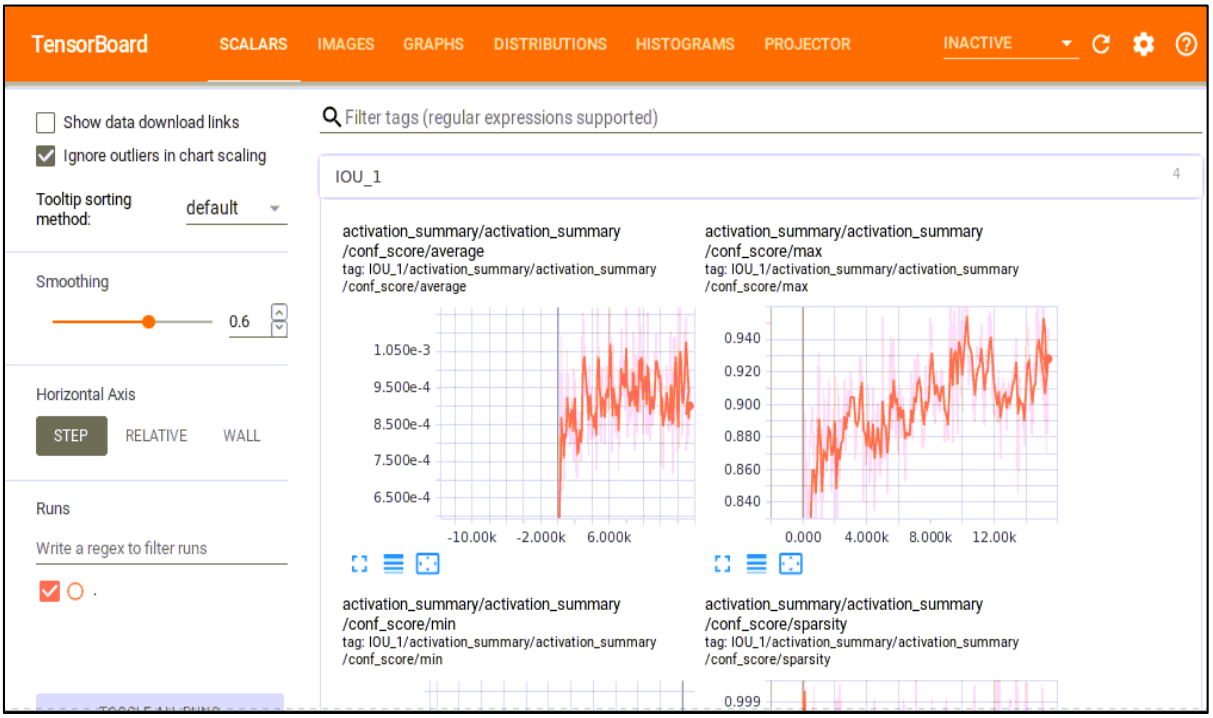

Figure 4.24. TensorBoard

Figure 4.25 shows the image menu of TensorBoard.

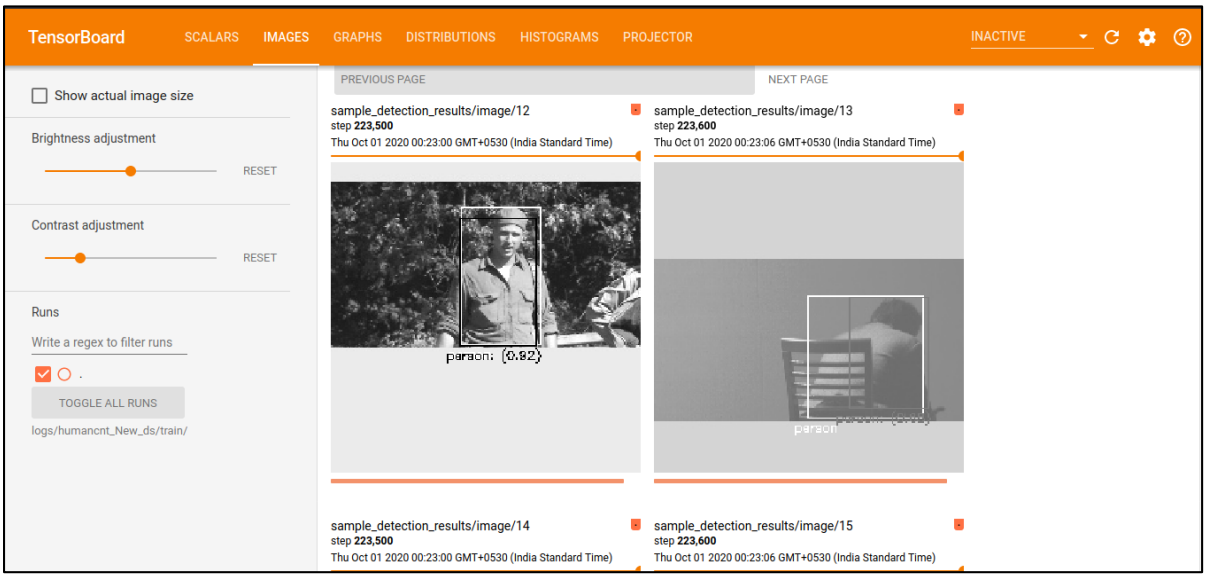

Figure 4.25. Image Menu of TensorBoard

8. 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).

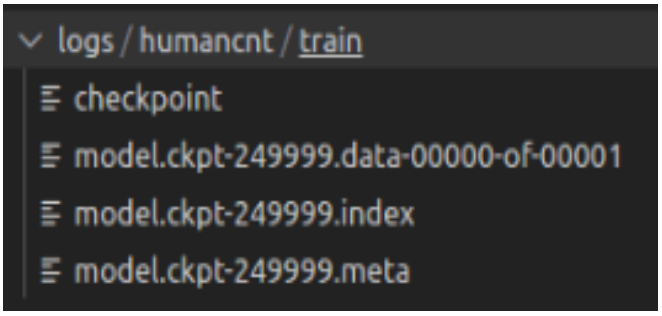

Figure 4.26. Example of Checkpoint Data Files at Log Folder

5. 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.

5.1. Generating the Frozen .pb File

Generate .pb file from latest checkpoint using the command below from the root directory of the training code.

```
$ python src/genpb.py –ckpt_dir <log directory> --freeze
```

For example, *python src/genpb.py –ckpt_dir logs/humancnt/train –freeze*.

```
earth:$ python3 src/genpb.py --ckpt_dir logs/humancnt/train --freeze
genrating pbtxt
self.preds: Tensor("conv12/convolution:0", shape=(20, 14, 14, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
preds2: Tensor("interpret_output/strided_slice:0", shape=(20, 14, 14, 7), dtype=float32, device=/device:GPU:0)
ANCHORS: 1372
Using checkpoint: ./model.ckpt-249999
saved pbtxt at checkpoint direcory Path
inputShape shape [1, 224, 224, 3]
```

Figure 5.1. .pb File Generation from Checkpoint

Figure 5.2 shows the generated .pb file.

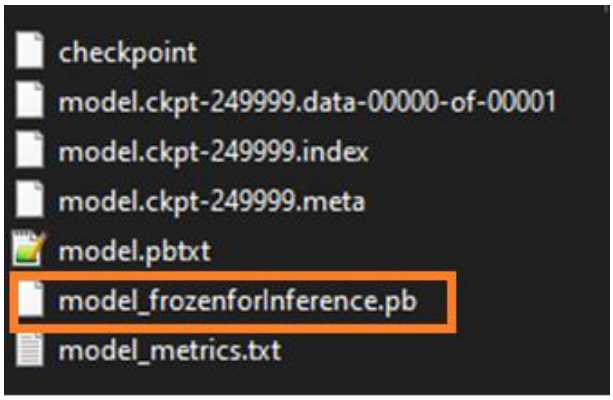

Figure 5.2. Frozen .pb File

6. Creating Binary File with Lattice SensAI

This chapter describes how to generate the binary file using the Lattice SensAI version 3.1program.

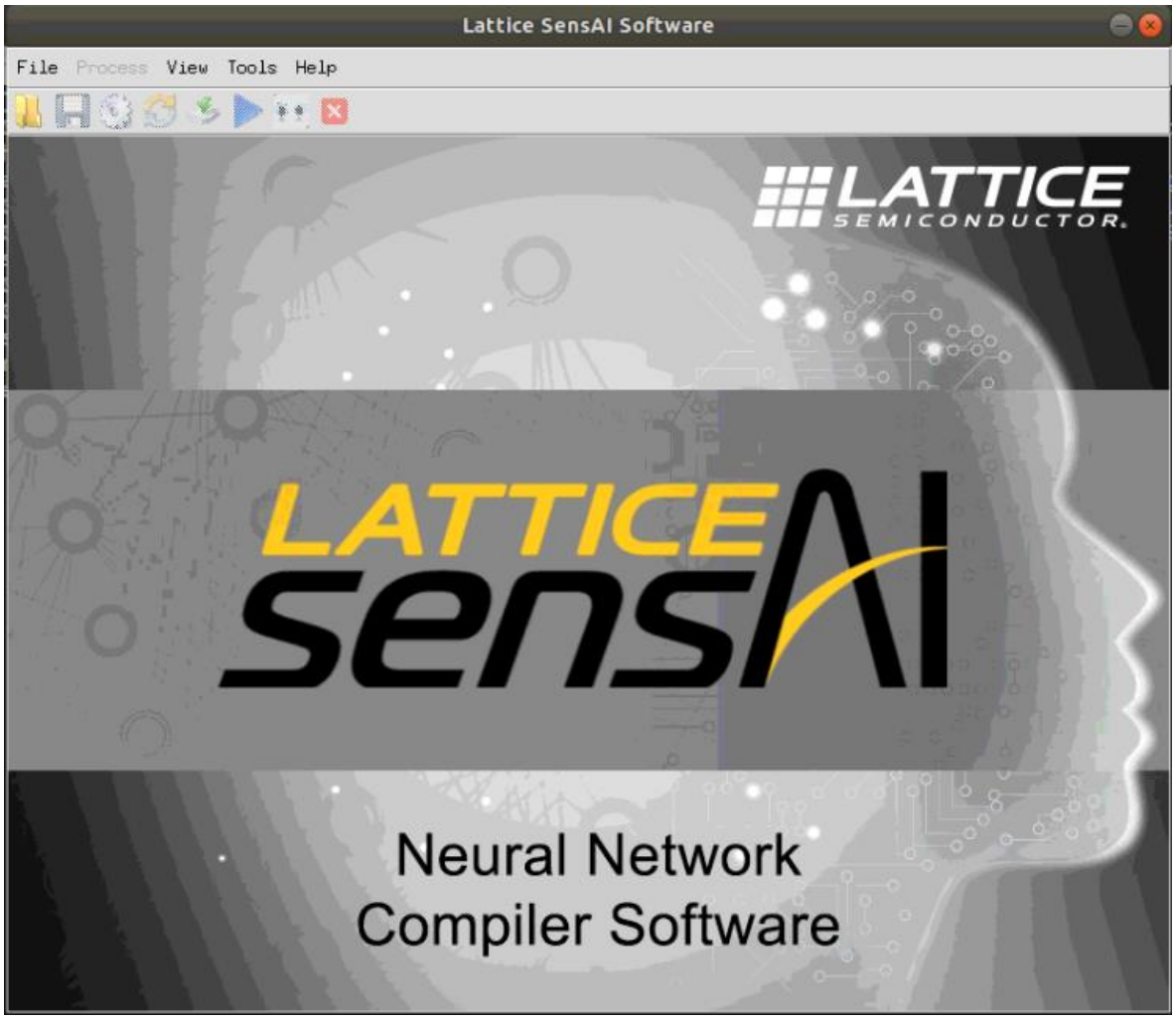

Figure 6.1. SensAI Home Screen

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**
   - **'Compact Mode'** should be unchecked.
3. Click **Network File** and select the network (.pb) file.

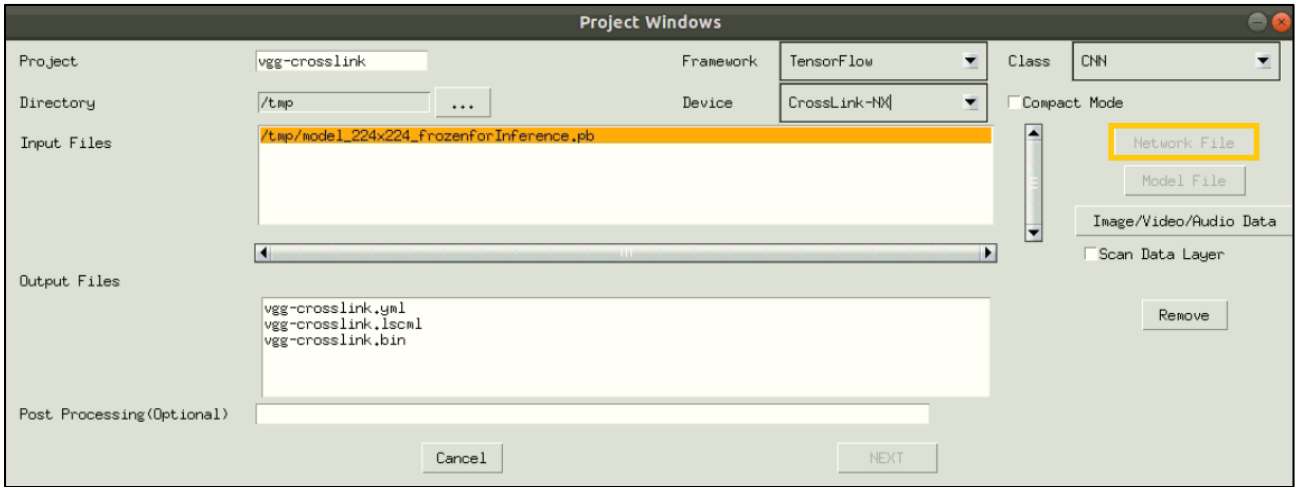

Figure 6.2. SensAI – Network File Selection

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

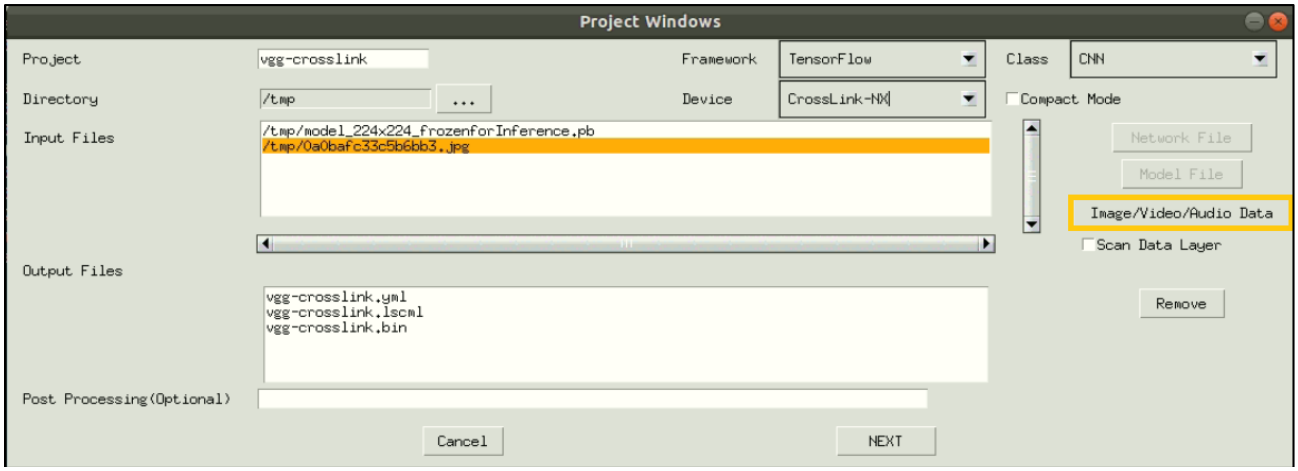

Figure 6.3. SensAI – Image Data File Selection

5. Click **NEXT**.
6. Configure your project settings as shown in Figure 6.4.

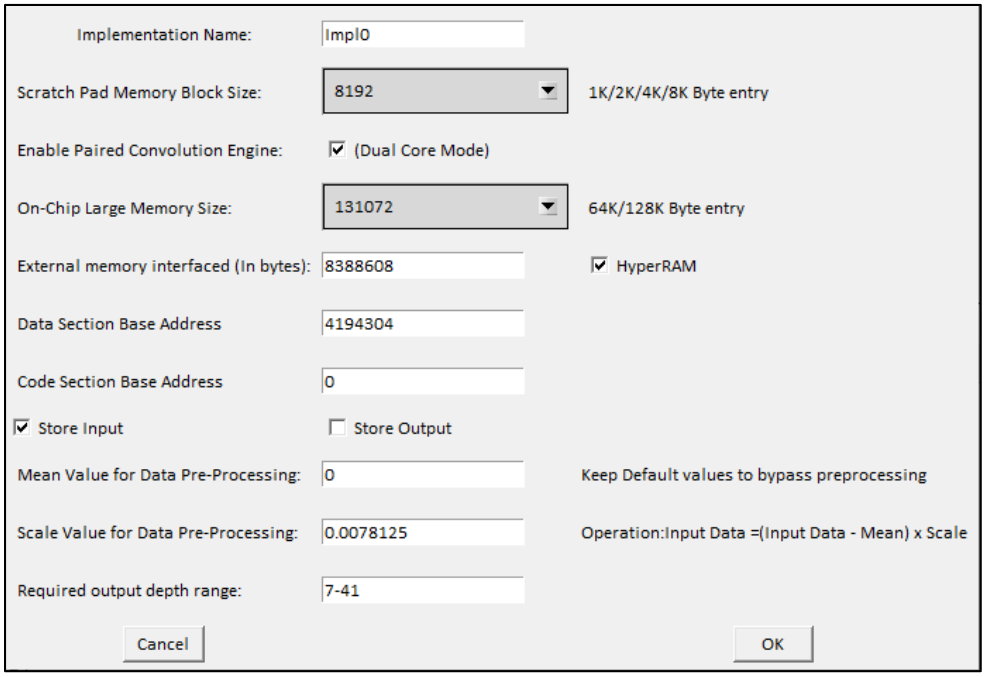

Figure 6.4. SensAI – Project Settings

Scratch Pad Memory Block Size and **Data Section Base Address** should match with FPGA RTL code.

7. Set the output depth range to 7 to 41. This range is set in order to tell the IP to not display the class probability values.
8. Click **OK** to create the project.
9. Double-click **Analyze**.

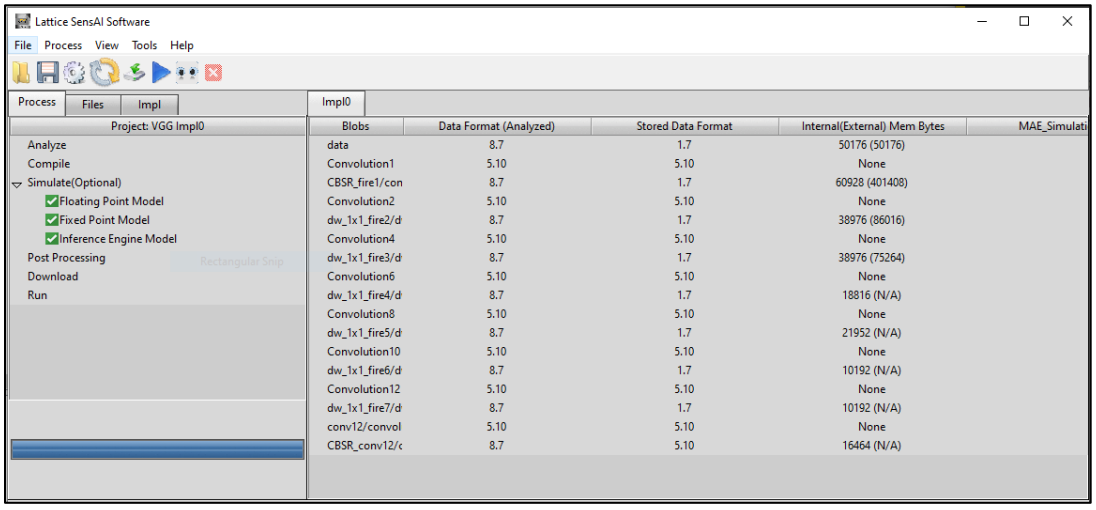

Figure 6.5. SensAI – Analyze Project

10. After analyzing the project, the tool generates the correct format for each layer since you performed the quantization in training.
11. Confirm the correct format of each layer as shown in Figure 6.6. If any of the fractional bits are different, modify the fractional bit for each layer by double-clicking on the values against each layer one by one.

Blobs	Data Format (Analyzed)	Stored Data Format	Internal(External) Mem Bytes
data	8.7	1.7	50176 (50176)
Convolution1	5.10	5.10	None
CBSR_fire1/con	8.7	1.7	60928 (401408)
Convolution2	5.10	5.10	None
dw_1x1_fire2/d	8.7	1.7	38976 (86016)
Convolution4	5.10	5.10	None
dw_1x1_fire3/d	8.7	1.7	38976 (75264)
Convolution6	5.10	5.10	None
dw_1x1_fire4/d	8.7	1.7	18816 (N/A)
Convolution8	5.10	5.10	None
dw_1x1_fire5/d	8.7	1.7	21952 (N/A)
Convolution10	5.10	5.10	None
dw_1x1_fire6/d	8.7	1.7	10192 (N/A)
Convolution12	5.10	5.10	None
dw_1x1_fire7/d	8.7	1.7	10192 (N/A)
conv12/convol	5.10	5.10	None
CBSR_conv12/c	8.7	5.10	16464 (N/A)

Figure 6.6. Q Format Settings for Each Layer

12. After changing the fractional bit, double-click on **Analyze** again.
13. Double-click **Compile** to generate the Firmware file.

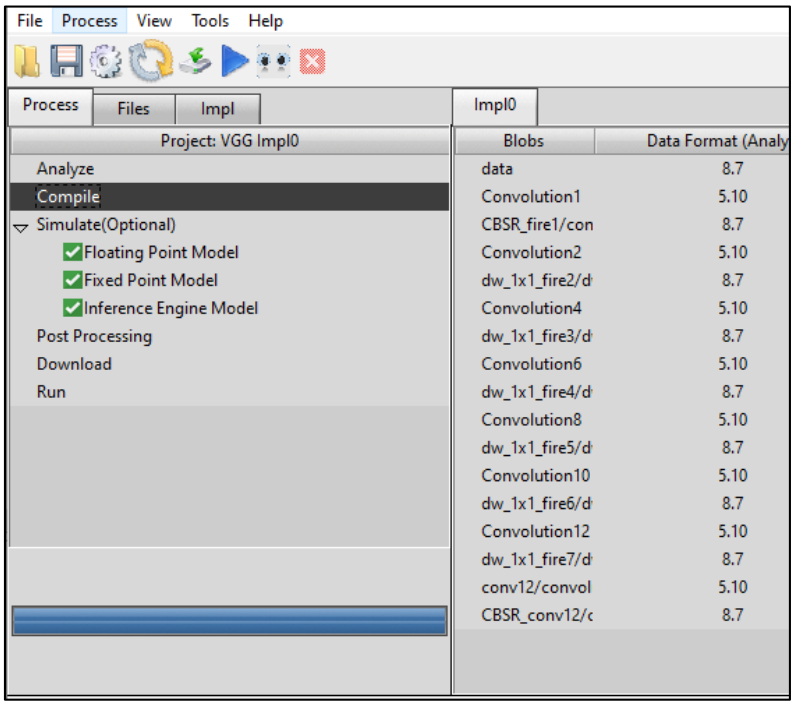

Figure 6.7. Compile Project

7. Hardware Implementation

7.1. Top Level Information

7.1.1. Block Diagram

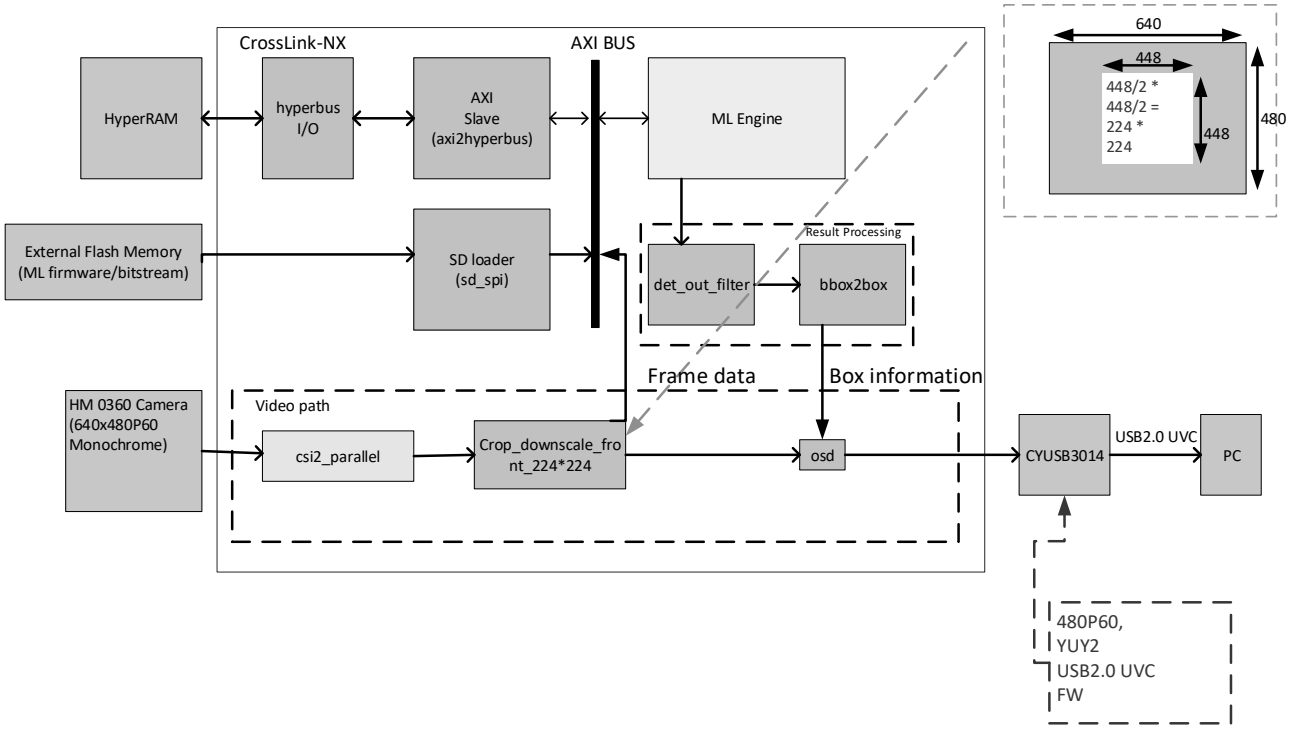

Figure 7.1. RTL Top Level Block Diagram

7.1.2. Operational Flow

This section provides a brief idea about the data flow across the CrossLink-NX board.

- The CNN module is configured with the help of a binary (.bin) file stored in a SD card. The .bin file is a command sequence code, which is generated by the Lattice Machine Learning software tool.
- The command code is written in hyperRAM through AXI before the execution of CNN Accelerator IP Core starts. CNN reads command code from hyperRAM during its execution and performs calculation with it per command code. Intermediate data may be transferred from/to hyperRAM per command code.
- The RAW8 data from the *csi2_to_parallel* module is downscaled to 224 × 224 image resolution by the *crop_downscale_front_224x224* module to match CNN input resolution. This data is written into hyperRAM memory through axi2_hyperbus through the *axi_ws2m AXI* interface module.
- After the command code and input data are available, the CNN Accelerator IP Core starts calculation at the rising edge of start signal.
- The output data of CNN is passed to *det_out_filter* for post processing. *det_out_filter* generates bounding box X, Y, W, and H coordinates associated with top 5 confidence value indexes for 224 × 224 image resolution.
- These coordinates are passed to *osd_back_128x128_human_count* for resizing to fit the actual image resolution on the PC.

7.1.3. Core Customization

Table 7.1. Core Parameter

Constant	Default (Decimal)	Description
OVLP_TH_2X	5	Intersection Over Union Threshold (NMS)
NUM_FRAC	10	Fraction Part Width in Q-Format representation.
EN_INF_TIME	0	Enable Timing measurement logic By default, it is zero and the memory file used is *human_count.mem*. If assigned 1, timing measurement is enabled and the memory file used is *human_count_INF.mem*. In order to configure the respective memory file, follow the steps below: 1. Open *dpram8192x8_human_count.ipx* from the File List in Radiant. 2. Click *Browse Memory File* from Initialization section. 3. Update the mem file path: • For 0 – /src/jedi_common/human_count.mem • For 1 – /src/jedi_common/human_count_INF.mem
INF_MULT_FAC	15907	Inference time multiplying factor calculated as per CNN clock frequency and using Q-Format (Q1.31). CNN Clock Frequency = 135 MHz Hence, CNN clock period = $1/(135 \times 10^6)$ µs = 0.000007407 ms Now, Q1.31 = $0.000007407 \times 2^{31}$ = ~15907
FLASH_START_ADDR	24'h300000	SPI Flash Read Start Address (keep the same address in programmer while loading the firmware file) For example, for the current start address, programmer address should be 0x00300000.
FLASH_END_ADDR	24'h400000	SPI Flash Read End Address (keep the same address in programmer while loading the firmware file) The address must be in multiple of 512 bytes. For example, for the current end address, programmer address should be: 0x00400000.
Constant Parameters (Not to be modified)		
NUM_ANCHOR	1372	Number of reference bounding boxes for all grids
NUM_GRID	196	Total number of Grids (X * Y)
NUM_X_GRID	14	Number of X Grids
NUM_Y_GRID	14	Number of Y Grids
PIC_WIDTH	224	Picture Pixel Width (CNN Input)
PIC_HEIGHT	224	Picture Pixel Height (CNN Input)
TOP_N_DET	10	Number of Top confidence bounding boxes detection
HYPERRAM_BASEADDR	4194304	Indicates hyperRAM starting Base address location value. This should match with the SensAI compiler while generating the firmware.
RAW8_OFFSET	0	Indicates hyperRAM starting address location value to store RAW8.

7.2. Architecture Details

7.2.1. SPI Flash Operation

The RTL module spi_loader_spram provides SPI Flash read operation and writes that data into HyperRAM through the AXI interface. It reads from SPI Flash and as soon as the board gets powered up, the .bit and .bin files are loaded in the expected addresses.

- Expected Address for BIT File (Programmer) – 0x0000000 - 0x00100000
- Expected Address for Firmware File (Programmer) – FLASH_START_ADDR - FLASH_END_ADDR

Typical sequence of the SPI Read commands for SPI Flash MX25L12833F is implemented using FSM in RTL as per the flow of the operation below.

- After FPGA Reset, RELEASE FROM DEEP POWER DOWN command (0xAB) is passed to SPI Flash memory. Then RTL waits for 500 clock cycles for SPI flash to come into Standby mode, if it is in Deep Power Down mode.
- RTL sends FAST READ command code (0x0B) on SPI MOSI signal for indication of Read Operation to SPI Flash.
- RTL sends three bytes of Address on SPI MOSI channel which determines the location in SPI flash from the position the data needs to be read.
- This SPI Flash has eight Dummy cycles as wait duration before read data appears on MISO channel. After waiting for eight dummy cycles, the RTL code starts reading the data.
- This read sequence is shown in Figure 7.2. The SPI Interface Signal Mapping with RTL signals are as follow:
  - CS (Chip Select) => SPI_CSS
  - SCLK (Clock) => SPI_CLK
  - SI (Slave In) => SPI_MOSI
  - SI (Slave Out) => SPI_MISO
- The Read Data on the MISO signal is stored in a FIFO in RTL, which then reads the data in multiples of 512 bytes. After 512 bytes chip select is de-asserted, the AXI FSM state is activated.

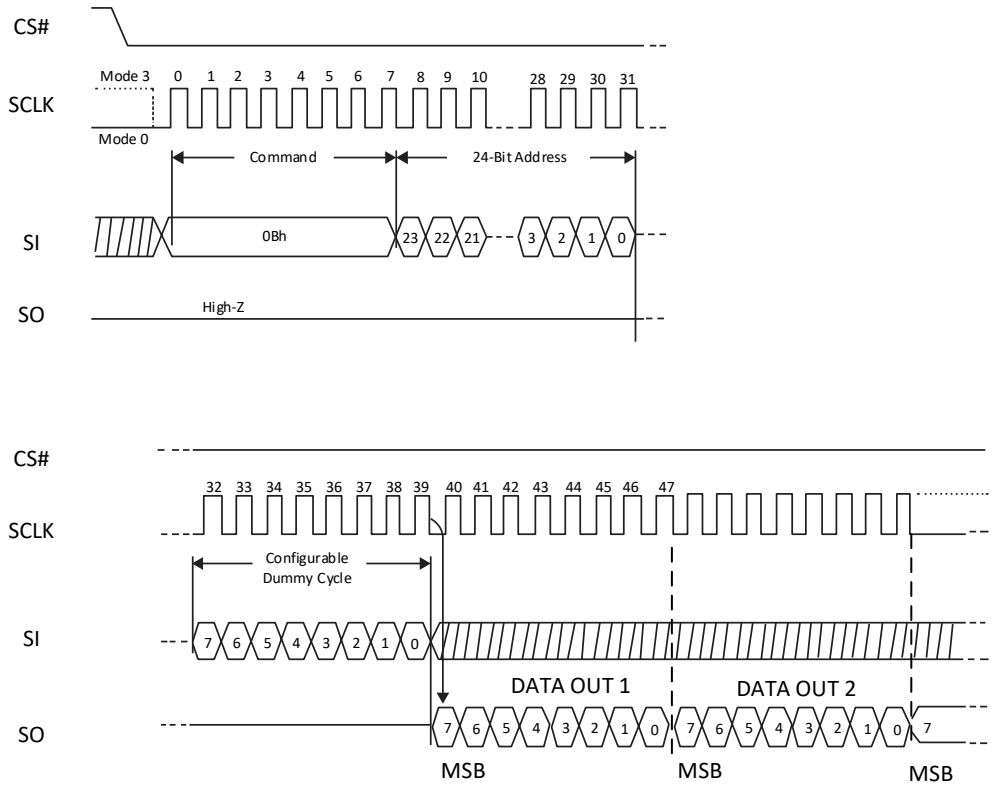

Figure 7.2. SPI Read Command Sequence

- AXI logic reads the data from FIFO in bursts of four on the AXI write channel, with each burst having 128 bytes.
- In accessing the HyperRAM, the *axi_ws2m* module is used as a Muxing module among the multiple input slave AXI interfaces as shown in Figure 7.6. The *spi_loader_spram* module is considered as SLAVE 0 and given priority to write into HyperRAM. The Master Interface connects to the *axi2_hyperbus* module, which provides output interface for accessing HyperRAM.
- After writing to HyperRAM is complete, the 512 bytes are fetched from the SPI Flash using the same command sequence as explained above until the FLASH_END_ADDR is reached.

7.2.2. Pre-processing CNN

The output from the *csi2_to_parallel* module is a stream of RGB data that reflects the camera image, which is given to the *crop_downscale_front_224x224* module.

The *crop_downscale_front_224x224* module processes that image data and generates input of 224 × 224 image data interface for CNN IP.

7.2.2.1. Pre-processing Flow

- RAW8 data values for each pixel are fed serially line by line for an image frame.
- These RAW8 data values are considered as valid only when horizontal and vertical masks are inactive. The mask parameters set to mask out boundary area of resolution (640 × 480) to 448 × 448 are shown below.
  - Left masking = 96
  - Right masking = 544 (Obtained as 96 + 448)
  - Top masking = 16
  - Bottom masking = 464 (Obtained as 16 + 448)
- The image obtained after masking is shown in Figure 7.3.

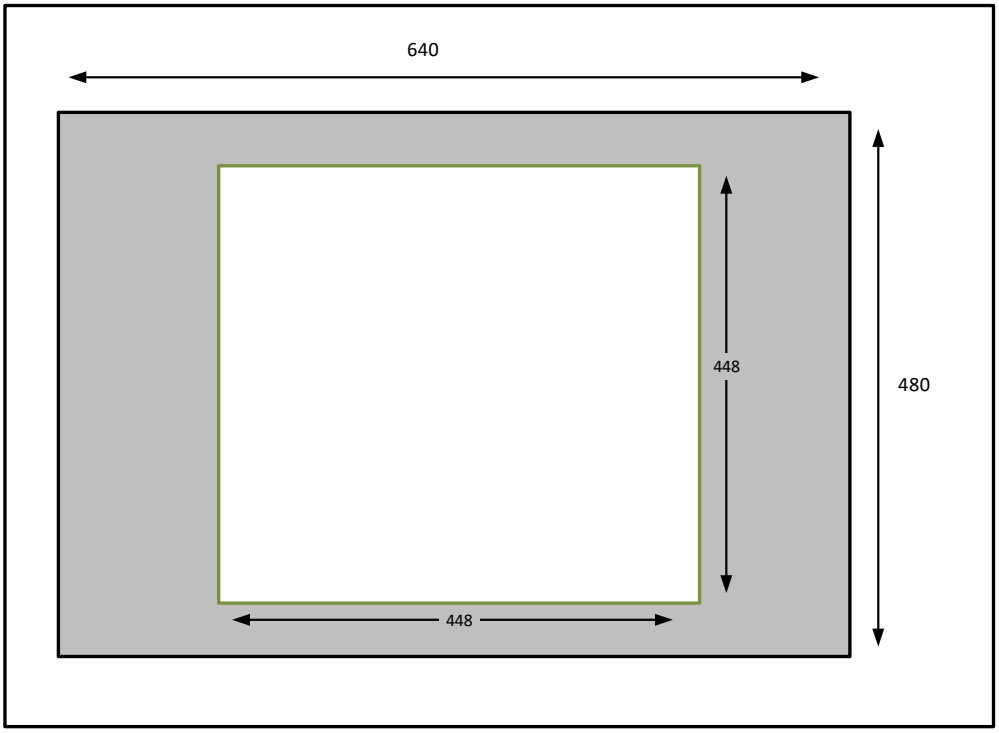

Figure 7.3. Masking

- The 448 × 448 frame block is downscaled into 224 × 224 resolution image as shown in Figure 7.4 by accumulating 2 × 2 pixels into single pixel (that is 448/2 × 448/2 = 224 × 224).

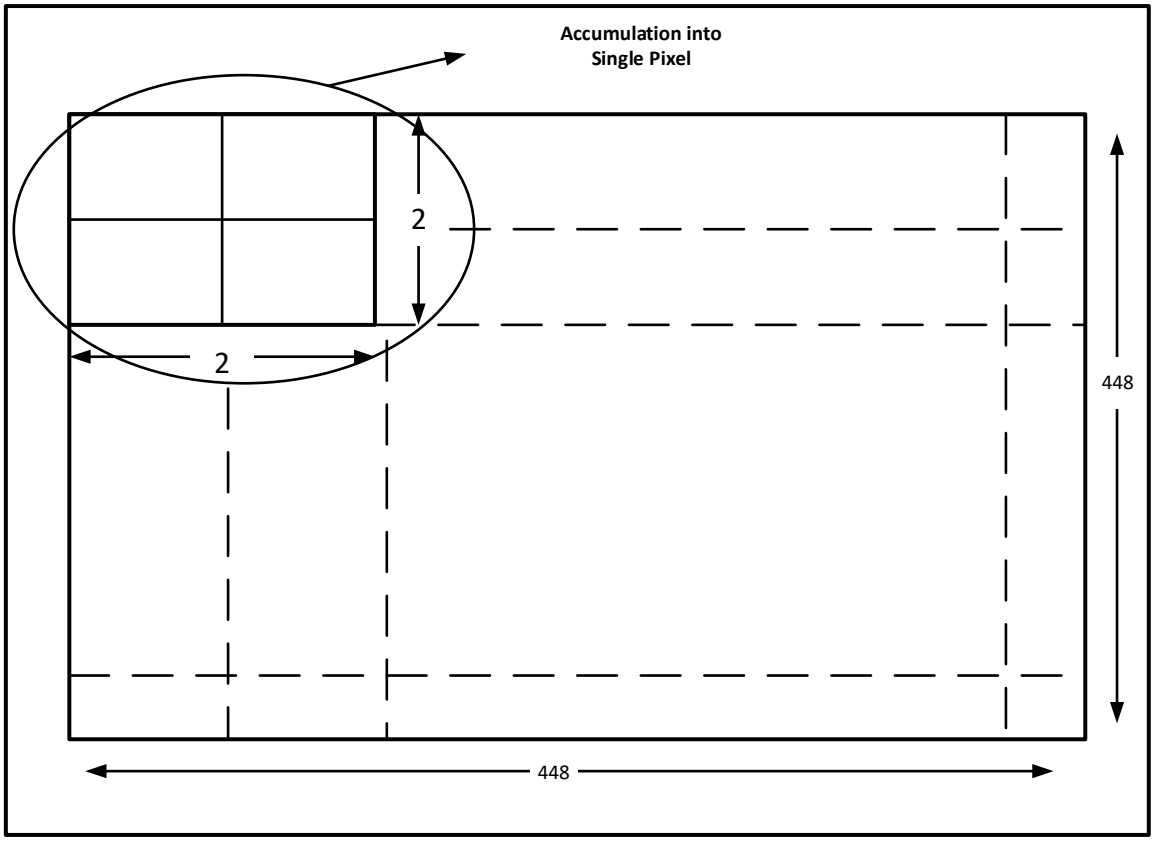

Figure 7.4. Downscaling

- This accumulated value is written into Line Buffer. Line Buffer is a True Dual-Port RAM. Accumulated RAW8 pixel values for 2 × 2 grids are stored in the same memory location.
- When data is read from memory, each RAW8 value is divided by 4 (that is the area of the 2 × 2 grid) to take the average of 2 × 2 grid matrix.
- The data from memory is read and stored in HyperRAM for CNN input through axi2_hyperbus, through the *axi_w2sm* module, which acts as an AXI interface to write data from slave (crop_downscale_front_224x224) to master (axi2_hyperbus). This process is described in the next section.

7.2.3. HyperRAM Operations

The CrossLink-NX board uses external HyperRAM for faster data transfer mechanism among the internal blocks and enhances the system performance. The *crop_downscale_front_224x224* module uses HyperRAM to store the downscaled image data.

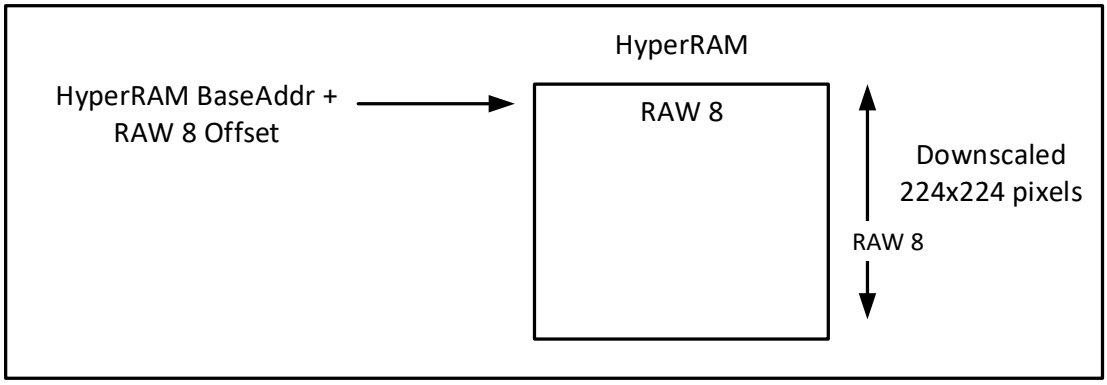

Figure 7.5. HyperRAM Memory Addressing

- The 448 × 448 image is distributed into 224 horizontal and 224 vertical lines, and each block consists of 2 × 2 pixels as shown in Figure 7.4. Thus, there is a total of 224 × 224 pixel values for the downscaled image.
- Primarily, the *crop_downscale_front_224x224* module stores 224 values each of RAW8 into a local FIFO for all 224 horizontal blocks. Later, this stored data is written to HyperRAM through the AXI write data channel.
- As shown in Figure 7.5, when final data is written out, 224 × 224 RAW8 pixels are initially stored into HyperRAM starting from HyperRam Base address location.
- The 224 × 224 pixel values stored in HyperRAM are serially obtained by the CNN engine after getting command sequence through the AXI interface.
- In order for the *crop_downscale_front_224x224* module to access HyperRAM for the operations explained above, the *axi_ws2m* module functions as a Muxing module for multiple input slave AXI interfaces as shown in Figure 7.6.
- For the internal blocks to access HyperRAM, the *axi_ws2m* module considers the sd_spi module as SLAVE 0, the *cnn_opt* module as SLAVE 1, the *crop_downscale_front_224x224* module as SLAVE 2, and the MASTER connects these slaves to the *axi2_hyperbus* module.
- The priority to select write channel is given, respectively, to the pi_loader slave, cnn slave, and crop-downscale slave. Whenever valid address is available from the respective Slave on its write address channel, that slave is given access of master channel if other priority slaves are not accessing it. Thus, when valid write address is obtained from the *crop_downscale_front_224x224* module, access is given to Slave 2 to use HyperRAM.

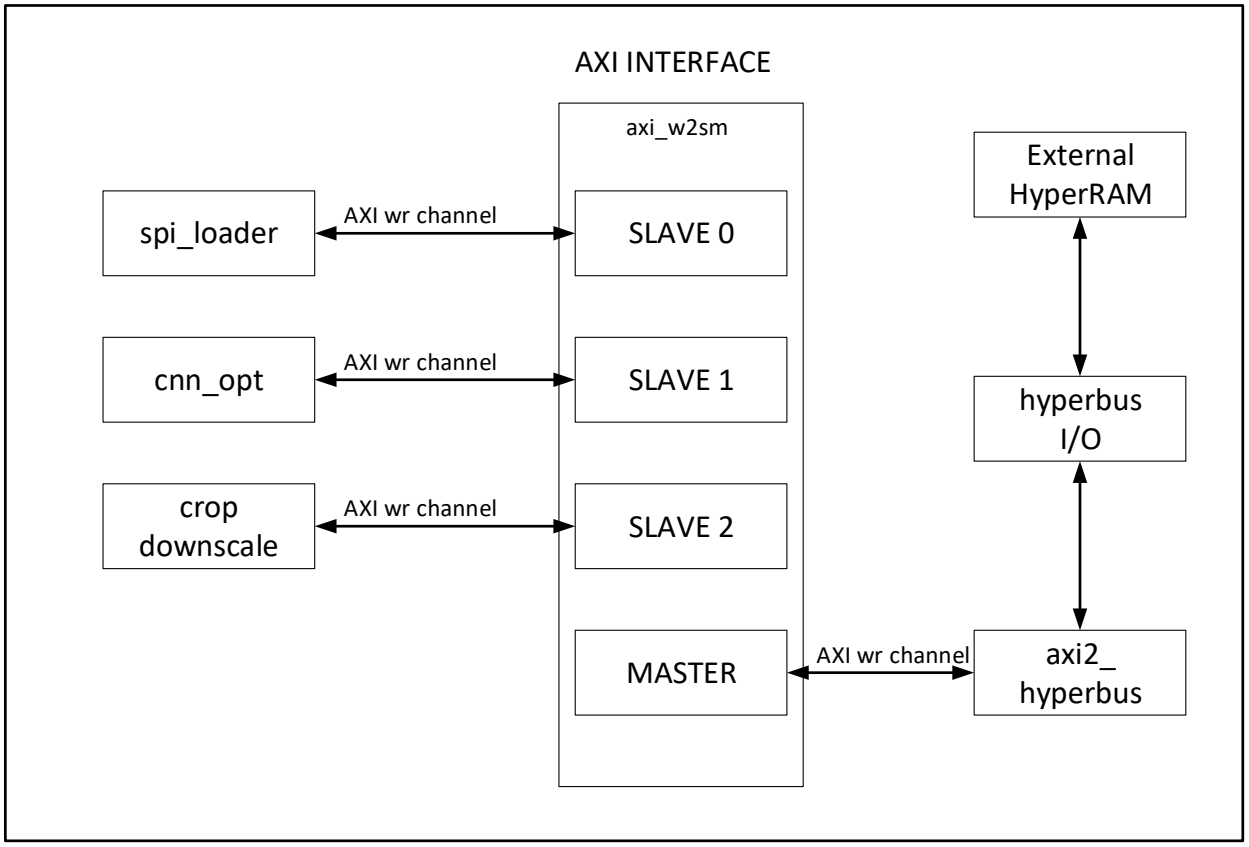

Figure 7.6. HyperRAM Access Block Diagram

7.2.4. Post-processing CNN

CNN provides a total of 6860 [1372 × 5 (C, X, Y, W, H)] values, which are given to the *det_out_filter* module. The CNN output data consists of the following parameters.

Table 7.2. Data Parameters of CNN Output

Parameter	Description
C	This parameter indicates the confidence of detected object class. For each grid cell (14 × 14), one confidence value (16-bit) for each anchor box (7) is provided making total values of confidence 14 * 14 * 7 = 1372 from CNN Output.
X	This parameter indicates the Relative X coordinate to transform the anchor box into a predicted bounding box for detected object. For each grid cell, one Relative X value (16-bit) for each anchor box is provided making total values of 14 * 14 * 7 = 1372 for X from CNN Output.
Y	This parameter indicates the Relative Y coordinate to transform the anchor box into a predicted bounding box for detected object. For each grid cell, one Relative Y value (16-bit) for each anchor box is provided making total values of 14 * 14 * 7 = 1372 for Y from CNN Output.
W	This parameter indicates the Relative W (Width) coordinate to transform the anchor box into a predicted bounding box for detected object. For each grid cell, one Relative W value (16-bit) for each anchor box is provided making total values of 14 * 14 * 7 = 1372 for W from CNN Output.
H	This parameter indicates the Relative H (Height) coordinate to transform the anchor box into a predicted bounding box for detected object. For each grid cell, one Relative H value (16-bit) for each anchor box is provided making total values of 14 * 14 * 7 = 1372 for H from CNN Output.

Figure 7.7 shows the format of CNN output.

OutputData	C				X	Y	W	H	X	Y	W	H
Index No	0-195	196-391	...	1176-1371	1372-1567	1568-1763	1764-1959	1960-2155	2156-2351	2352-2547	2548-2743	2744-2939
Grid No	0-195	0-195	...	0-195	0-195	0-195	0-195	0-195	0-195	0-195	0-195	0-195
Anchor No	1	2	...	7	1				2			

Figure 7.7. CNN Output Data Format

The primary functionality of the *det_out_filter* module is to capture the CNN valid output and modify it to work with the *osd_back_128x128_human_count* module.

The det_out_filter module contains two sub-modules: det_sort_conf and det_st_bbox.

- 1372 values of confidence are passed to the *det_sort_conf* module. It sorts out the top 10 highest confidence values and stores their indexes. Index values are passed to the *det_st_class* and *det_st_bbox* modules.
- 1372 × 4 values of coordinates are passed to the *det_st_bbox* module. It calculates the bounding box coordinates for 448 × 448 image from 224 × 224 coordinates.
- The *osd_back_128x128_human_count* module contains logic for post-processing the *draw_box_simple* module calculates the box coordinates for 448 × 448 image from 224 × 224 coordinates.
- The *lsc_osd_text* module generates character bitmap for text display on PC.

7.2.4.1. Confidence Sorting

- All input confidence values (1372) are compared with threshold parameter CONF_THRESH(500) value. Confidence values that are greater than threshold are considered as valid for sorting.
- The *det_sort_conf* module implements an anchor counter (0-1371), which increments on each confidence value. It provides the index of confidence value given by the CNN output.
- Two memory arrays are generated in this module: (1) sorted top 10 (TOP_N_DET) confidence value array, and (2) sorted top 10 confidence index array.
- For sorting, a standard sorting algorithm is followed. As input confidence values start arriving, each value is compared with stored/initial value at each location of the confidence value array.
- If the input value is greater than stored/initial value on any array location and lesser than stored/initial value of previous array location, the input value is updated on current array location. The previously stored value of current location is shifted into the next array location.
- Refer to Figure 7.8 for sorting of new value of confidence into existing confidence value array. Calculated confidence index (anchor count value) is also updated in the confidence index array along with the confidence value array.

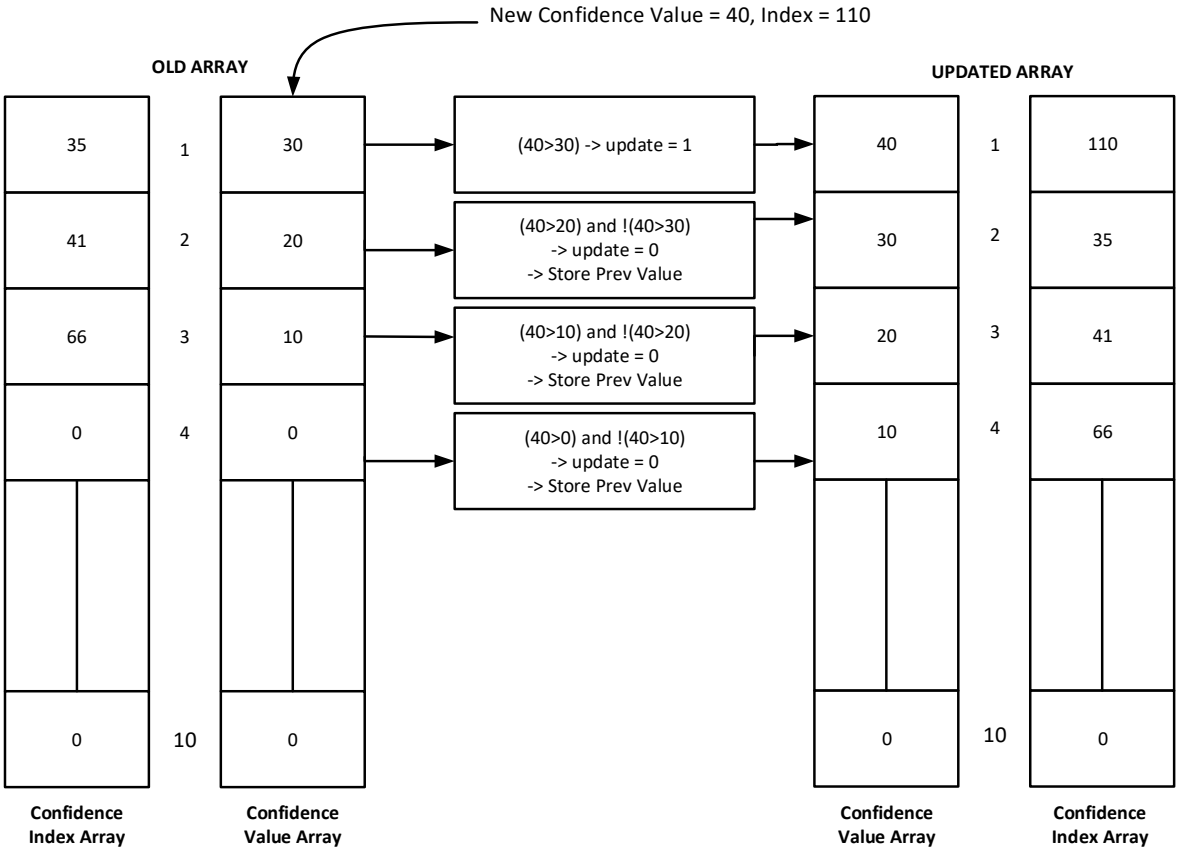

Figure 7.8. Confidence Sorting

- This process is followed for all 1372 confidence values. This module provides 10 indexes (o_idx_00 to o_idx_09) as output along with the count of valid indexes (o_num_conf). o_idx_00 contains the highest confidence value index and o_idx_09 contains the lowest confidence value index.

7.2.4.2. Bounding Box Calculation

The Neural Network for Object Detection is trained with seven reference boxes of pre-selected shapes having constant W (Width) and H (Height). These reference boxes are typically referred as anchors.

Table 7.3. Pre-Selected Width and Height of Anchor Boxes

Anchor No.	1	2	3	4	5	6	7
W × H (pixel)	184 × 184	138 × 138	92 × 92	69 × 69	46 × 46	34 × 34	23 × 23

Anchors are centered around 14 × 14 grid cells of image. So each grid center has above seven anchors with pre-selected shape. 14 × 14 are the number of grid centers along horizontal and vertical directions. The grid center (X, Y) pixel values are shown in Table 7.4.

Table 7.4. Grid Center Values (X, Y) for Anchor Boxes

Grid No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X (pixel)	15	30	45	60	75	90	105	119	134	149	164	179	194	209
Y (pixel)	15	30	45	60	75	90	105	119	134	149	164	179	194	209

CNN provides a total of 1372 (14 × 14 × 7) values of each relative coordinates X, Y, W, and H to transform the fixed size anchor into a predicted bounding box. Input X, Y, W, and H values associated with top 10 sorted confidence indexes are used for box calculation in *det_st_bbox* module.

Each anchor is transformed to its new position and shape using the relative coordinates as shown in logic 1.

```
LOGIC 1
X' = X coordinate of Predicted Box
X = Grid Center X according to Grid number
W = Width of Anchor according to Anchor number
DeltaX = Relative coordinate for X (CNN output)

X' = X + W * DeltaX
Y' = Y + H * DeltaY
W'= W * DeltaW
H' = H * DeltaH
```

The Box coordinates are passed to bbox2box module in *jedi_human_count_top.v* after NMS process.

7.2.4.3. NMS – Non Max Suppression

The NMS is implemented to make sure that in object detection, a particular object is identified only once. It filters out the overlapping boxes using OVLP_TH_2X value.

NMS process is started when the CNN output data is completely received.

- The process starts from the box having highest Confidence coordinates: 0^{th} location in X, Y, W, H array.
- These coordinates are compared against the second highest Confidence coordinates: First location in X, Y, W, H array. From this comparison, Intersection and Union coordinates are found.
- From these coordinates, Intersection and Union area are calculated between the highest confidence box and the second highest confidence box as shown in Figure 7.9.

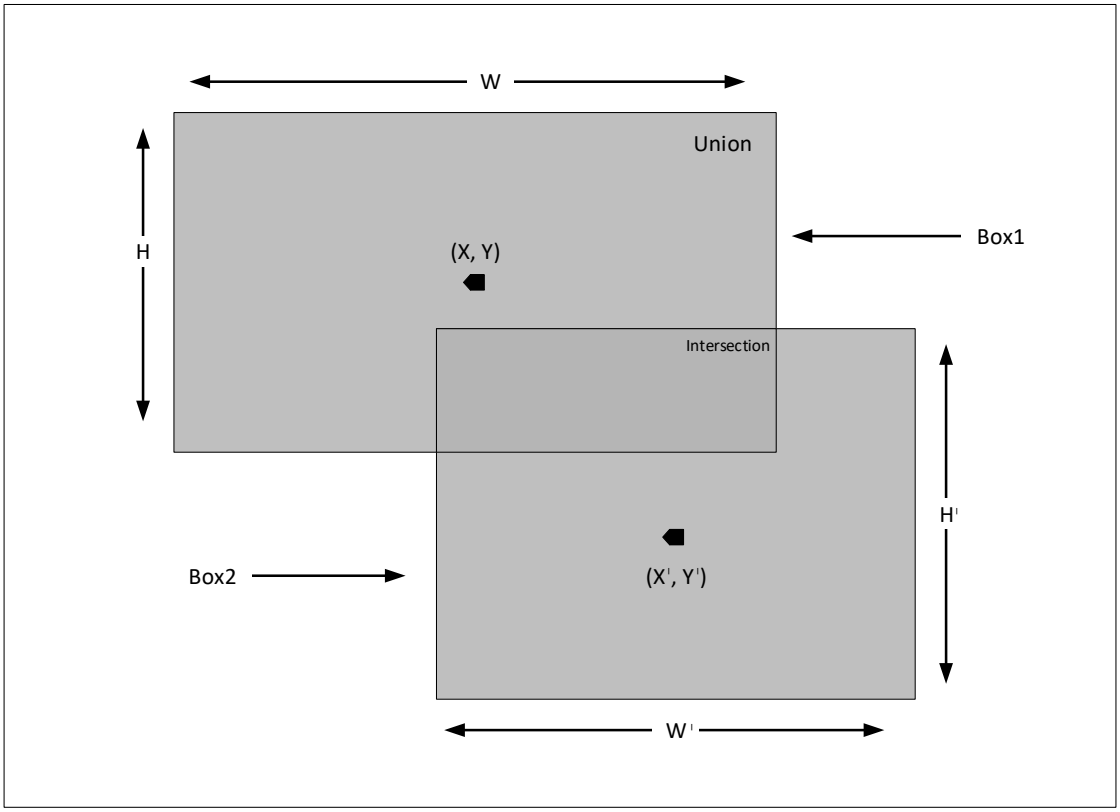

Figure 7.9. Intersection-Union Area NMS

- If *Intersection Area * (OVLP_TH_2X/2) > Union Area*, the box with the lower confidence value is blocked in final output.
- This NMS calculation is performed between all the combinations of two boxes.
- After all combinations are checked, output *array o_bbox_bmap* contains boxes, which are correctly overlapped or non-overlapped. *o_out_en* provides valid pulse for *crop_downscale_human_count* for further processing on these box coordinates.

7.2.4.4. Bounding Box Upscaling

The process of upscaling bounding boxes for 448 × 448 resolution is accomplished by two different modules *bboxbox* and *draw_box_simple*.

Initially, the bbox2box module in *jedi_human_count_top.v* obtains box coordinate outputs from *det_out_filter*.

Considering (X, Y) as center of the box of Width W and Height H, it calculates extreme ends of the Box (X1, X2 and Y1, Y2) for 224 × 224 resolution. It also clamps the coordinate values so that the box remains out of masking area. This is shown in Logic 2.

```
LOGIC 2
X1 = If ((X'- W'/2) < 0)      =>  0      else  (X'- W'/2)
Y1 = If ((Y'- H'/2) < 0)      =>  0      else  (Y'- H'/2)
X2 = If ((X'+ W'/2) > 224)    =>  223    else  (X'+ W'/2)
Y2 = If ((Y'+ H'/2) > 224)    =>  223    else  (Y'+ H'/2)
```

The final calculated X1, X2, Y1, and Y2 values for all the boxes in bbox2box are then sent to *draw_box_simple* module through the *osd_back_128x128_human_count* module. The *draw_box_simple* module converts these input coordinates provided for 224 × 224 resolution into 448 × 448 resolution as shown in Logic 3.

```
LOGIC 3
X1' = (X1) * 2 + Horizontal-Mask (96)
Y1' = (Y1) * 2 + Vertical-Mask (16)
X2' = (X2) * 2 + Horizontal-Mask (96)
Y2' = (Y2) * 2 + Vertical-Mask (16)
```

For converting from 224 to 448, the coordinates are multiplied with 2. Required offset value is added in coordinate calculations to keep the boxes out of mask area. X1, X2 and Y1, Y2 coordinates are calculated for each Box.

Pixel Counter and Line Counter keep track of the pixels of each line, and lines of each frame. The outer boundary of the box and inner boundary of the box are calculated when Pixel and Line counter reaches to coordinates (X1, X2) and (Y1, Y2) respectively. Calculations are done as per Logic 4.

```
LOGIC 4
Outer Box = (Pixel Count >= (X1 - 1)) and (Pixel Count <= (X2 + 1)) and
                        (Line Count >= (Y1 - 1)) and (Line Count <= (Y2 + 1))
Inner Box = (Pixel Count > (X1 + 1)) and (Pixel Count < (X2 - 1)) and
                        (Line Count > (Y1 + 1)) and (Line Count < (Y2 - 1))
```

Each bounding box is calculated by removing the intersecting area of outer and inner box. The box is only displayed if the Box-Bitmap for that box is set to 1 (from the *det_st_bbox via bbox2box* module). Box on calculations are as done as Logic 5.

```
LOGIC 5
Box_on[1] = Outer Box[1] and ~Inner Box[1] and Box-Bitmap[1]
Box_on[2] = Outer Box[2] and ~Inner Box[2] and Box-Bitmap[2]
.
.
Box_on[20] = Outer Box[20] and ~Inner Box[20] and Box-Bitmap[20]
```

The *o_box_obj* signal is asserted when any of the above *Box_on signal* is set which is then connected to *green_on* signal and processed for Bounding Box display through the PC.

7.2.4.5. OSD Text Display

- The *lsc_osd_text* module provides bitmap of each ASCII character to be displayed with specified position on screen. It takes count of detected Humans.
- It sets an output signal (text_on) when text is to be displayed on the PC through the USB. When text_on is set, YCbCr value for that pixel location is assigned FF, 7F, and 7F respectively values (white color) and sent to USB output instead of original pixel value.

7.2.4.6. USB Wrapper

- The *Wrapper_USB3* module is used to transmit 16-bit data to the output 16-bit interface every clock cycle.
- This module takes input data in YCbCr 24-bit format and gives output as 16-bit YCb and YCr format. This module does not change or regenerate input timing parameters.

7.2.4.7. Inference Time Calculation

The time taken by a trained neural network model to infer/predict outputs after obtaining input data is called inference time. The process of this calculation is explained as follows.

- The inference time is calculated by implementing a counter to store the count of CNN engine cycles per frame.
- When the *i_rd_rdy* signal (that is, o_rd_rdy coming from CNN engine) is high, the CNN engine indicates that it is ready to get input; and when it is low, the engine indicates that it is busy.
- When *i_rd_rdy* signal is low, the CNN counter begins and stops when the *i_rd_rdy* signal goes high again indicating that previous execution is over and the CNN is ready for new input.
- As shown in Figure 7.10, when *rdy_h2l* (ready high-to-low) pulse is asserted, the CNN Up-counter starts from 1 and the count value increases until *i_rd_rdy* is not high again. The count value is stored in *count*.
- Similarly, when *rdy_l2h* (ready low-to-high) pulse is asserted, the Up-counter stops and the final CNN count value is obtained *cnn_count*.

```
assign rdy_l2h          =  i_rd_rdy & ~rdy;
assign rdy_h2l          = ~i_rd_rdy & rdy;

// Counter to calculate individual frame inference time
assign count_c          = (o_rd_done & rdy_h2l) ? 27'd1 : (i_rd_rdy) ? count : (count + 27'd1);

// Store individual cnn frame counter
assign cnn_count_c      =  rdy_l2h ? count : cnn_count;
```

Figure 7.10. CNN Counter Design

The methodology used to obtain stable inference time is to calculate inference time per frame and obtain the average inference time value after 16 CNN frames are over as discussed below.

- After completion of every frame, the new count value *cnn_count* obtained, as explained above, is added to the previous value and stored in *cnn_adder*.
- A frame counter monitors the frame count. After 16 frames, when the frame count is done, the *cnn_adder* value is reset as shown in Figure 7.11.

```
// Frame counter to calculate CNN frames upto 16 (0 - 15)
assign frame_counter_c   = (rdy_l2h_rr)? (frame_counter + 4'd1) : frame_counter;

// keep adding indiviual cnn frame counter for 16 frames. Then clear to 0
assign cnn_adder_c       = (count_done)? 31'd0 : (rdy_l2h_r) ? (cnn_adder + {4'd0,cnn_count}) : cnn_adder;

// Counter addition done when 1ll 16 cnn frame counter values are added and averaged
assign count_done        = (frame_counter == 4'd15) & (rdy_l2h_rr);
```

Figure 7.11. Frame Counter Design for 16 CNN Frames Average

- To get the average inference time value *avg_inf_time_hex* after frame count is done, the final *cnn_adder* value is divided by 16 as shown in Figure 7.12.

```
// Average Inference Time calculated by dividing by 16
assign inf_time_c        = (count_done)? cnn_adder[30:4] : inf_time;

// 32 Bit average Inference time in Hex
assign avg_inf_time_hex  = {5'd0,inf_time};
```

Figure 7.12. Average Inference Time Calculation

- Using the Lattice Multiplier library module, the average inference time value is multiplied by *INF_MULT_FAC*. A parameter indicating the inference multiplying factor is explained in Table 7.1.
- The inference time in millisecond *inf_time_ms* is obtained by dividing the output obtained from this multiplier by 2^31 as per the Q-Format shown in Figure 7.13**.**
- All the above obtained values namely the CNN count, the average inference time, and the inference time in millisecond are passed on to the *lsc_osd_text_human_count* module for getting bitmap to display the characters.

```
assign inf_time_ms        = inf_time_mult[46:31];
```

Figure 7.13. Inference Time in Millisecond

7.2.4.8. Inference Time Display Management

The Inference Time Display Management module mainly consists of a DPRAM, which holds the characters at pre-defined address positions indicated by *text_addr* and an 8 × 8 font ROM which provides the bitmap of these characters for PC display.

This module basically functions by using two entities. One is the position of the character where it has to be displayed, and the other is by reading the ASCII value of the character to be displayed.

For this purpose, once the CNN count, individual frame inference time and the inference time in millisecond values are obtained, they are converted from hex into ASCII values, as shown in Table 7.5.

The average inference time input value *i_avg_inf_time_hex* is converted from hex to ASCII values as shown below. To display eight characters of this value on the PC, this input is stored in the respective *r_avginfhex_ch*. The characters obtained by adding 7'h30 and 7'h37 are shown in Figure 7.14.

```
r_avginfhex_ch0 <= (i_avg_inf_time_hex[31:28] > 4'd9)? (i_avg_inf_time_hex[31:28] + 7'h37) : (i_avg_inf_time_hex[31:28] + 7'h30);
r_avginfhex_ch1 <= (i_avg_inf_time_hex[27:24] > 4'd9)? (i_avg_inf_time_hex[27:24] + 7'h37) : (i_avg_inf_time_hex[27:24] + 7'h30);
r_avginfhex_ch2 <= (i_avg_inf_time_hex[23:20] > 4'd9)? (i_avg_inf_time_hex[23:20] + 7'h37) : (i_avg_inf_time_hex[23:20] + 7'h30);
r_avginfhex_ch3 <= (i_avg_inf_time_hex[19:16] > 4'd9)? (i_avg_inf_time_hex[19:16] + 7'h37) : (i_avg_inf_time_hex[19:16] + 7'h30);
r_avginfhex_ch4 <= (i_avg_inf_time_hex[15:12] > 4'd9)? (i_avg_inf_time_hex[15:12] + 7'h37) : (i_avg_inf_time_hex[15:12] + 7'h30);
r_avginfhex_ch5 <= (i_avg_inf_time_hex[11:8]  > 4'd9)? (i_avg_inf_time_hex[11:8]  + 7'h37) : (i_avg_inf_time_hex[11:8]  + 7'h30);
r_avginfhex_ch6 <= (i_avg_inf_time_hex[7:4]   > 4'd9)? (i_avg_inf_time_hex[7:4]   + 7'h37) : (i_avg_inf_time_hex[7:4]   + 7'h30);
r_avginfhex_ch7 <= (i_avg_inf_time_hex[3:0]   > 4'd9)? (i_avg_inf_time_hex[3:0]   + 7'h37) : (i_avg_inf_time_hex[3:0]   + 7'h30);
```

Figure 7.14. Average Inference Time Value to ASCII Conversion

Table 7.5. Signal Values to ASCII Conversion

CHARACTERS FOR DISPLAY	VALUE TO BE ADDED TO SIGNAL	ASCII HEX VALUE	ASCII DECIMAL VALUE
1	7'h30	31	49
2	7'h30	32	50
3	7'h30	33	51
4	7'h30	34	52
5	7'h30	35	53
6	7'h30	36	54
7	7'h30	37	55
A	7'h37	41	65
B	7'h37	42	66
C	7'h37	43	67
D	7'h37	44	68
E	7'h37	45	69
F	7'h37	46	70

To display eight characters of individual frame inference time, the input signal *i_inf_time_hex* is converted from hex to ASCII and stored in the respective *r_infhex_ch* signal, as shown in Figure 7.15.

In the same way to display four characters of inference time in ms, the input signal *i_inf_ms* is converted from hex to ASCII and stored in respective *r_inf_ms* signal, as shown in Figure 7.16.

```
r_infhex_ch0    <= (i_inf_time_hex[31:28] > 4'd9)? (i_inf_time_hex[31:28] + 7'h37) : (i_inf_time_hex[31:28] + 7'h30);
r_infhex_ch1    <= (i_inf_time_hex[27:24] > 4'd9)? (i_inf_time_hex[27:24] + 7'h37) : (i_inf_time_hex[27:24] + 7'h30);
r_infhex_ch2    <= (i_inf_time_hex[23:20] > 4'd9)? (i_inf_time_hex[23:20] + 7'h37) : (i_inf_time_hex[23:20] + 7'h30);
r_infhex_ch3    <= (i_inf_time_hex[19:16] > 4'd9)? (i_inf_time_hex[19:16] + 7'h37) : (i_inf_time_hex[19:16] + 7'h30);
r_infhex_ch4    <= (i_inf_time_hex[15:12] > 4'd9)? (i_inf_time_hex[15:12] + 7'h37) : (i_inf_time_hex[15:12] + 7'h30);
r_infhex_ch5    <= (i_inf_time_hex[11:8]  > 4'd9)? (i_inf_time_hex[11:8]  + 7'h37) : (i_inf_time_hex[11:8]  + 7'h30);
r_infhex_ch6    <= (i_inf_time_hex[7:4]   > 4'd9)? (i_inf_time_hex[7:4]   + 7'h37) : (i_inf_time_hex[7:4]   + 7'h30);
r_infhex_ch7    <= (i_inf_time_hex[3:0]   > 4'd9)? (i_inf_time_hex[3:0]   + 7'h37) : (i_inf_time_hex[3:0]   + 7'h30);
```

Figure 7.15. CNN Count Values to ASCII Conversion

```
r_infms_ch0     <= (i_inf_time_ms[15:12] > 4'd9)? (i_inf_time_ms[15:12] + 7'h37) : (i_inf_time_ms[15:12] + 7'h30);
r_infms_ch1     <= (i_inf_time_ms[11:8] > 4'd9)? (i_inf_time_ms[11:8] + 7'h37) : (i_inf_time_ms[11:8]  + 7'h30);
r_infms_ch2     <= (i_inf_time_ms[7:4] > 4'd9)? (i_inf_time_ms[7:4] + 7'h37) : (i_inf_time_ms[7:4] + 7'h30);
r_infms_ch3     <= (i_inf_time_ms[3:0] > 4'd9)? (i_inf_time_ms[3:0] + 7'h37) : (i_inf_time_ms[3:0] + 7'h30);
```

Figure 7.16. Inference Time in Millisecond Values to ASCII Conversion

The positions where these values have to be displayed are given using *text_addr* signal, as shown in Figure 7.17. The use of these locations is shown in Figure 7.17 and Table 7.5. A memory initialization file *human_count.mem* is used by Lattice Radiant tool to store characters at address locations for display.

```
assign w_avginfhex_ch0_pos  = (text_addr == 13'd1136);
assign w_avginfhex_ch1_pos  = (text_addr == 13'd1137);
assign w_avginfhex_ch2_pos  = (text_addr == 13'd1138);
assign w_avginfhex_ch3_pos  = (text_addr == 13'd1139);
assign w_avginfhex_ch4_pos  = (text_addr == 13'd1140);
assign w_avginfhex_ch5_pos  = (text_addr == 13'd1141);
assign w_avginfhex_ch6_pos  = (text_addr == 13'd1142);
assign w_avginfhex_ch7_pos  = (text_addr == 13'd1143);

assign w_infhex_ch0_pos     = (text_addr == 13'd1172);
assign w_infhex_ch1_pos     = (text_addr == 13'd1173);
assign w_infhex_ch2_pos     = (text_addr == 13'd1174);
assign w_infhex_ch3_pos     = (text_addr == 13'd1175);
assign w_infhex_ch4_pos     = (text_addr == 13'd1176);
assign w_infhex_ch5_pos     = (text_addr == 13'd1177);
assign w_infhex_ch6_pos     = (text_addr == 13'd1178);
assign w_infhex_ch7_pos     = (text_addr == 13'd1179);
// Milisecond display
assign w_infms_ch0_pos      = (text_addr == 13'd1146);
assign w_infms_ch1_pos      = (text_addr == 13'd1147);
assign w_infms_ch2_pos      = (text_addr == 13'd1148);
assign w_infms_ch3_pos      = (text_addr == 13'd1149);
```

Figure 7.17. Text Address Positions to Display Input Values

The address location structure for displaying average inference time (of 16 CNN frames) and inference time in millisecond values along with their strings are stored in *human_count.mem* is shown in Figure 7.18.

	Characters stored as ASCII Values in human_count.mem file for string display											Avg Inf Time	Characters in human_count.mem file except Inf time value				
Character	A	v	g	I	n	f	T	i	m	e	:		(	Inf Time	m	s	)
Address in decimal	1122	1123	1124	1126	1127	1128	1130	1131	1132	1133	1134	1136 to 1143	1145	1146 to 1149	1150	1151	1152

Figure 7.18. Address Locations to Display Individual Frame Time and Inference Time with String in PC

The address location structure for displaying individual frame inference time Values along with the string are stored in *human_count.mem*, as shown in Figure 7.19.

Characters stored as ASCII Values in human_count.mem file for string display *								Individual Frame Inf Time
Character	I	n	f	T	i	m	e	:	
Address in decimal	1162	1163	1164	1166	1167	1168	1169	1170	1172 to 1179

Figure 7.19. Address Locations to Display CNN Count Value and its String in PC Output

To display the input values in address locations shown in Figure 7.18 and Figure 7.19, the ASCII values obtained as shown in Table 7.5, Figure 7.15, and Figure 7.16 are sent to the 8 × 8 font ROM with the help of the *font_char* signal to obtain the bitmap for PC, as shown in Figure 7.20.

```
assign font_char     = (r_face_ch0_pos )? r_face_ch0 :
                       (r_face_ch1_pos )? r_face_ch1 :
                       (r_th_sign_pos  )? r_th_sign :
                       (r_th_ch0_pos   )? r_th_ch0 :
                       (r_th_ch1_pos   )? r_th_ch1 :
                       (r_th_ch2_pos   )? r_th_ch2 :
                       (r_th_ch3_pos   )? r_th_ch3 :
                       //Inference Time Logic
                       (r_avginfhex_ch0_pos )? r_avginfhex_ch0 :
                       (r_avginfhex_ch1_pos )? r_avginfhex_ch1 :
                       (r_avginfhex_ch2_pos )? r_avginfhex_ch2 :
                       (r_avginfhex_ch3_pos )? r_avginfhex_ch3 :
                       (r_avginfhex_ch4_pos )? r_avginfhex_ch4 :
                       (r_avginfhex_ch5_pos )? r_avginfhex_ch5 :
                       (r_avginfhex_ch6_pos )? r_avginfhex_ch6 :
                       (r_avginfhex_ch7_pos )? r_avginfhex_ch7 :
                       (r_infhex_ch0_pos    )? r_infhex_ch0 :
                       (r_infhex_ch1_pos    )? r_infhex_ch1 :
                       (r_infhex_ch2_pos    )? r_infhex_ch2 :
                       (r_infhex_ch3_pos    )? r_infhex_ch3 :
                       (r_infhex_ch4_pos    )? r_infhex_ch4 :
                       (r_infhex_ch5_pos    )? r_infhex_ch5 :
                       (r_infhex_ch6_pos    )? r_infhex_ch6 :
                       (r_infhex_ch7_pos    )? r_infhex_ch7 :
                       (r_infms_ch0_pos    )? r_infms_ch0 :
                       (r_infms_ch1_pos    )? r_infms_ch1 :
                       (r_infms_ch2_pos    )? r_infms_ch2 :
                       (r_infms_ch3_pos    )? r_infms_ch3 :
                                             text_data[6:0];
```

Figure 7.20. Bitmap Extraction from Font ROM

8. Creating FPGA Bitstream File

This section describes the steps to compile RTL bitstream using Lattice Radiant tool.

8.1. Bitstream Generation Using Lattice Radiant Software

To create the FPGA bitstream file:

1. Open the Lattice Radiant software. Default screen is shown in Figure 8.1.

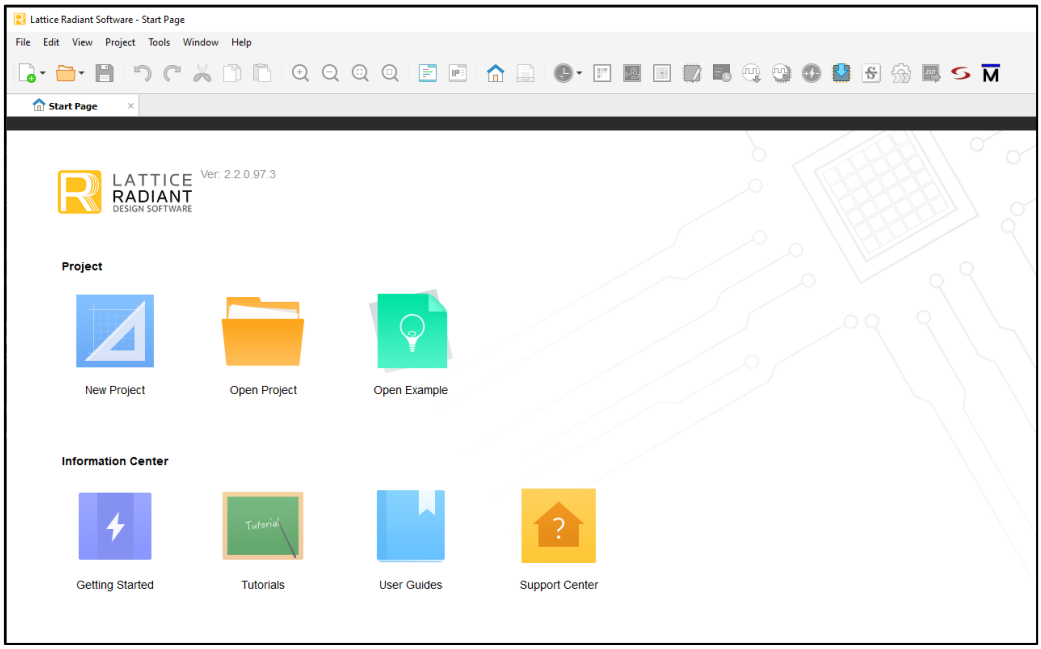

Figure 8.1. Radiant – Default Screen

2. Go to **File > Open > Project**.
3. Open the Radiant project file (.rdf) for CrossLink-NX Voice and Vision Human Count Demo RTL. As shown in Figure 8.2, you can also open the project by selecting the yellow folder shown in the user interface.

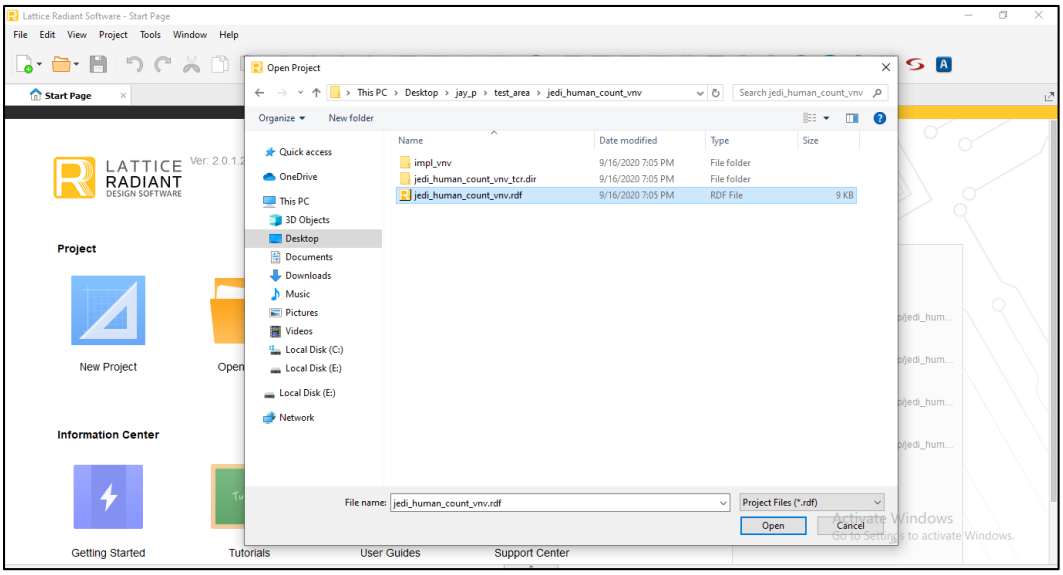

Figure 8.2. Radiant – Open Crosslink-NX Voice and Vision Project File (.rdf)

4. After opening the project file, check the following points shown in Figure 8.3.
   - The design loaded with zero errors message shown in the *Output* window.
   - Check the following information in the *Project Summary* window.
     - **Part Number – LIFCL-40-7MG2891**
     - **Family – LIFCL**
     - **Device – LIFCL-40**
     - **Package – CSBGA289**

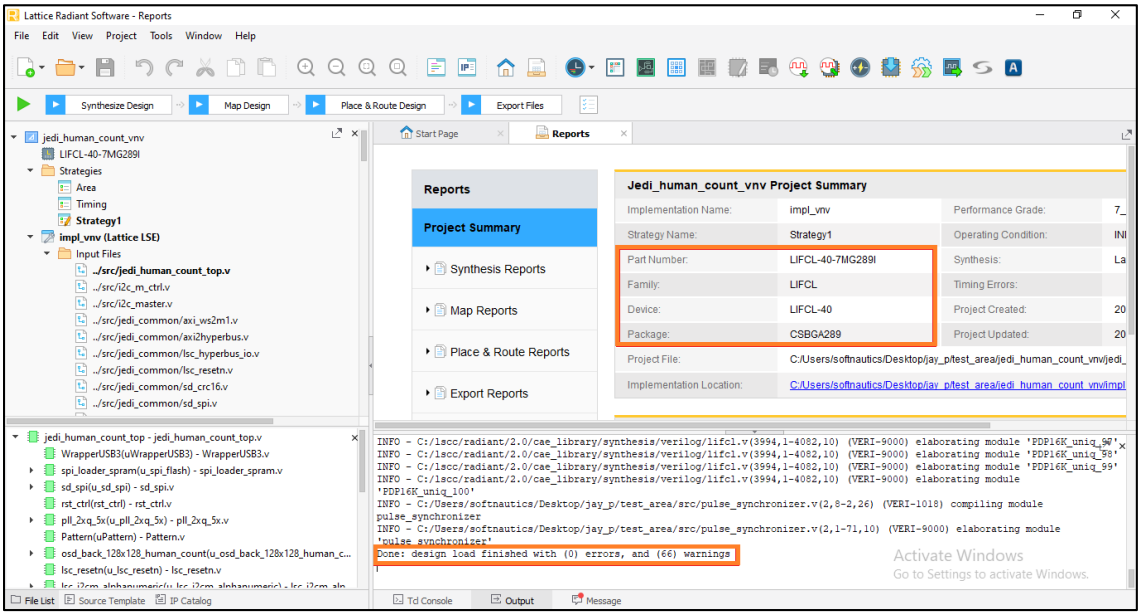

Figure 8.3. Radiant – Design Load Check After Opening the Project File

5. If the design is loaded without errors, click the **Run** button to trigger bitstream generation as shown in Figure 8.4**.**

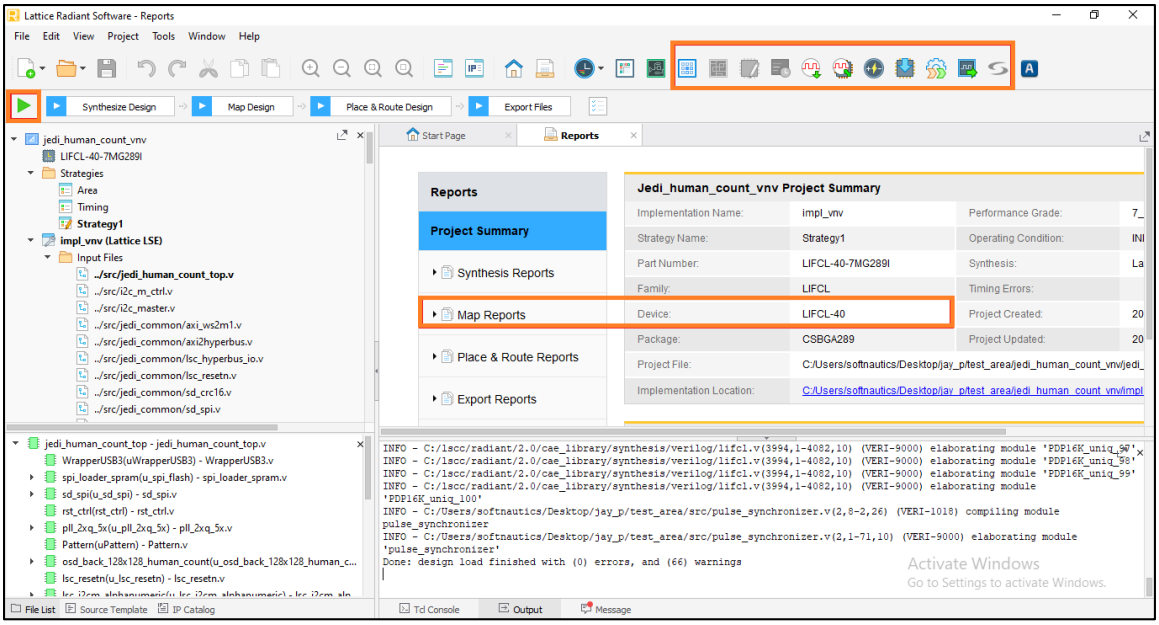

Figure 8.4. Radiant – Trigger Bitstream Generation

6. The Lattice Radiant tool displays *Saving bit stream in …* message in the **Reports** window, as shown in Figure 8.5. The bitstream is generated at *Implementation Location*, as shown in Figure 8.5**.**

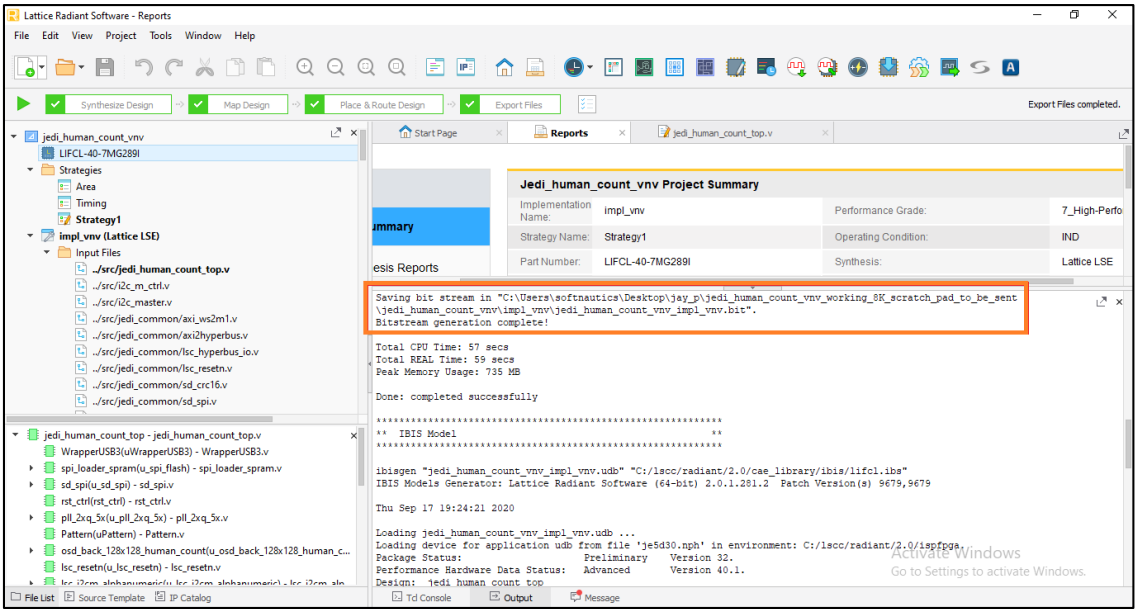

Figure 8.5. Radiant – Bit File Generation Report Window

8.2. IP Configuration in Lattice Radiant Software

If you are going to uninstall an old version of the existing IP or if a latest version of the IP should be installed after loading the design without any errors, follow the procedure below and trigger the Bitstream generation *RUN* option.

To configure the IP:

1. If the existing IP should be uninstalled, go to **IP Catalog**. In the IP tree, go to **IP > DSP** and select your IP. Click **Delete** as shown. Select **Yes** as shown in Figure 8.6 and the IP is successfully removed from the tree.

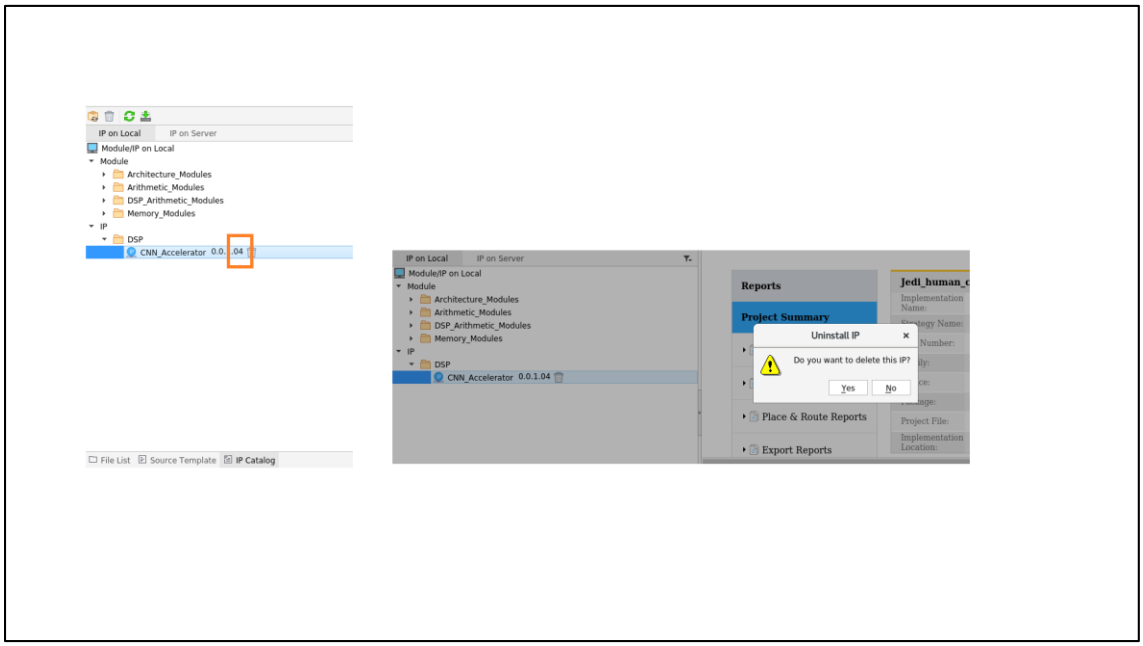

Figure 8.6. Radiant – Uninstall Old IP

2. To install a new IP in the tree, select **Install a User IP**, as shown in Figure 8.7.

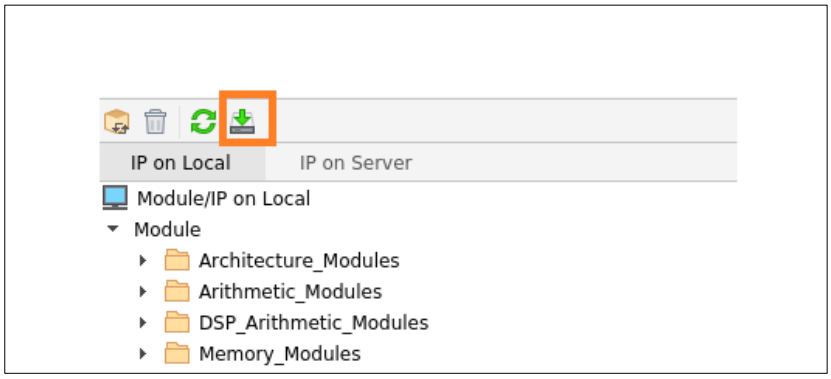

Figure 8.7. Radiant – Install New IP

3. Select your IP package (.ipk) and the **IP License agreement** window pops up. Click **Accept** and the user IP is installed in IP tree.

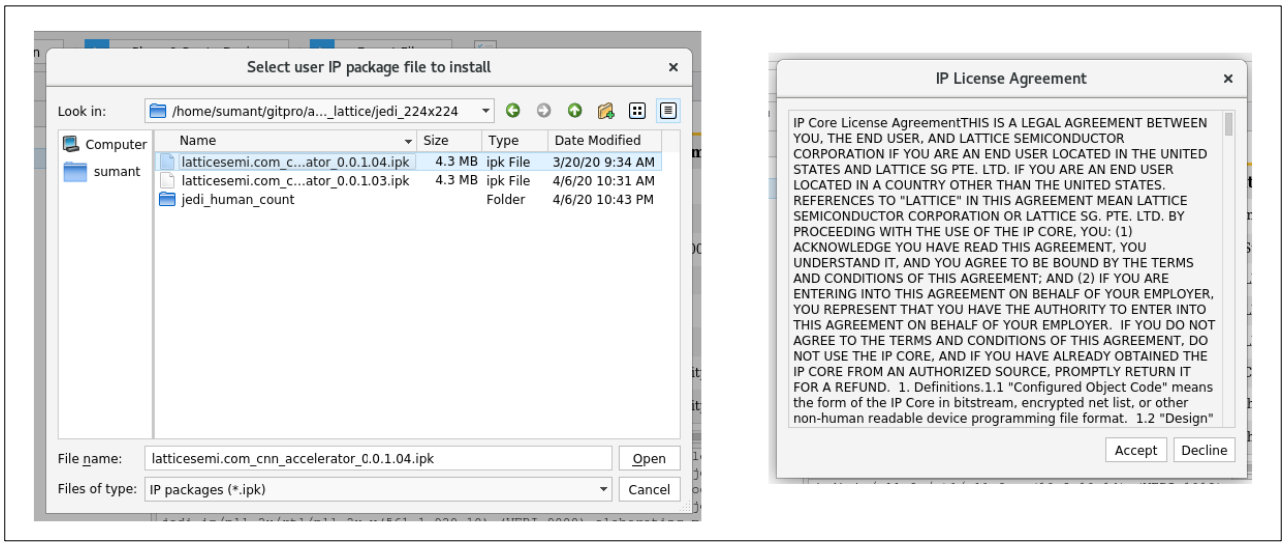

Figure 8.8. Radiant – Select User IP Package to Install

4. After the user IP is installed, the bitstream can be generated by triggering **RUN** button.

9. Programming the Demo

9.1. Programming the CrossLink-NX Voice and Vision SPI Flash

9.1.1. Erasing the CrossLink-NX Voice and Vision SRAM Prior to Reprogramming

If the CrossLink-NX Voice and Vision device is already programmed (either directly, or loaded from SPI Flash), follow this procedure to first erase the CrossLink-NX Voice and Vision SRAM memory before re-programming the CrossLink-NX Voice and Vision's SPI Flash. If you are doing this, keep the board powered when re-programming the SPI Flash (so it does not reload on reboot).

To erase the CrossLink-NX Voice and Vision device:

1. Launch Radiant Programmer. In the Getting Started dialog box, select **Create a new blank project**.

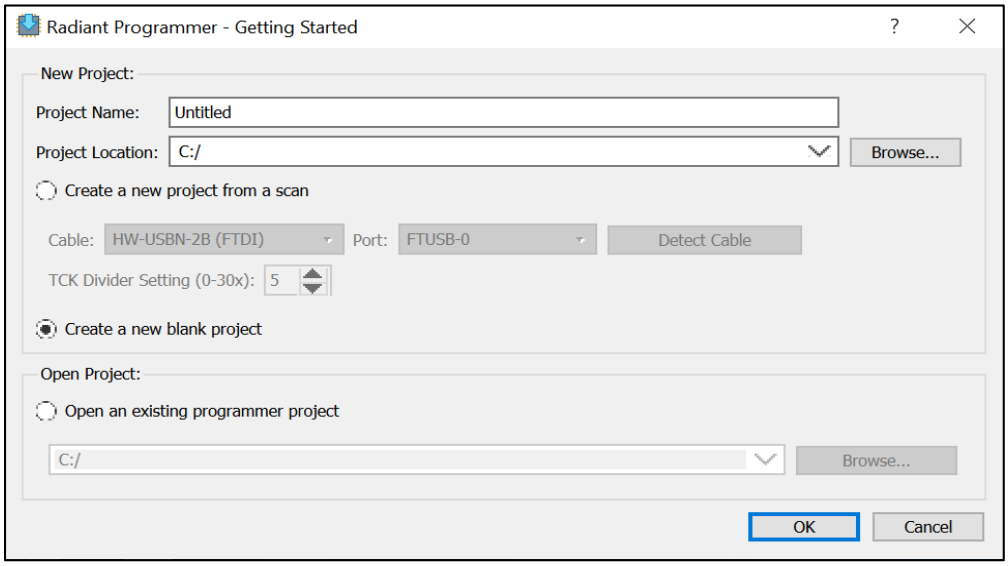

Figure 9.1. Radiant Programmer – Default Screen

2. Click **OK**.
3. In the Radiant Programmer main interface, select **LIFMD** for Device Family, **LIFCL** for Device Vendor, and **LIFCL-40** for Device, as shown in Figure 9.2.

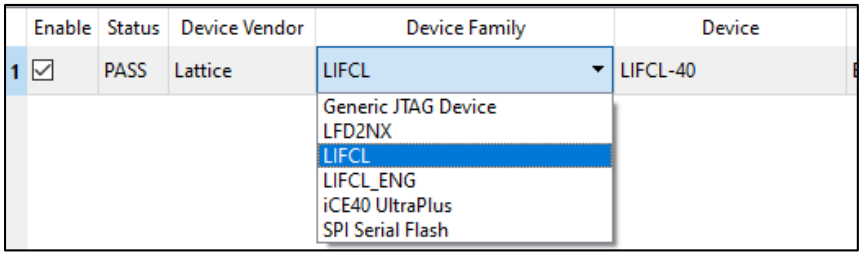

Figure 9.2. Radiant Programmer – Device Selection

4. Right-click and select **Device Properties**.
5. Select **JTAG** for Port Interface, **Direct Programming** for Access Mode, and **Erase Only** for Operation, as shown in Figure 9.3.

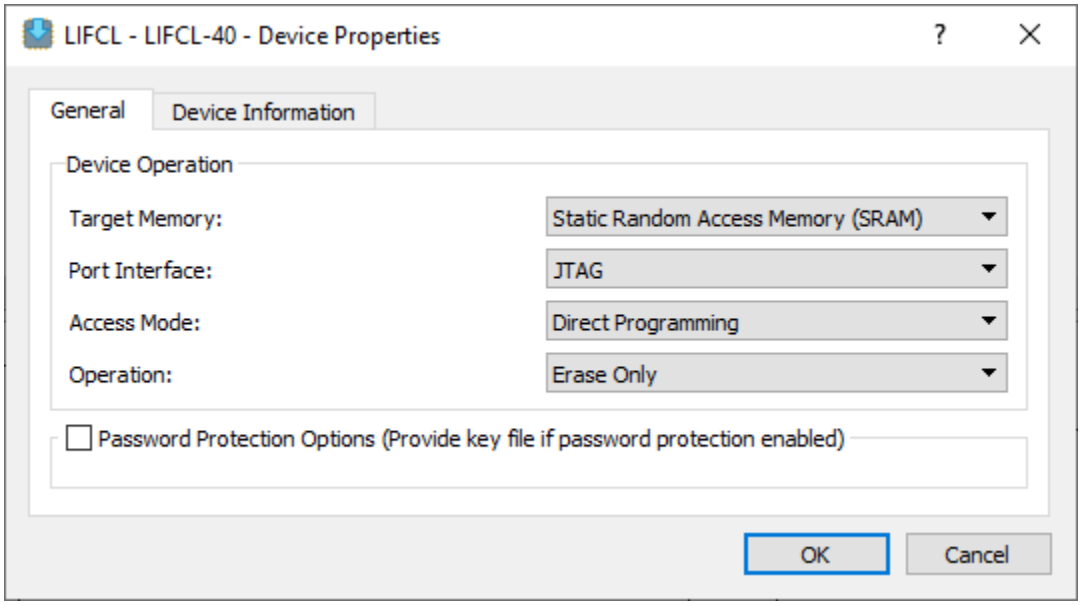

Figure 9.3. Radiant Programmer – Device Operation

6. Click **OK** to close the Device Properties dialog box.
7. Click the **Program** button to start the erase operation.

9.1.2. Programming the CrossLink-NX Voice and Vision Board

To program the CrossLink-NX Voice and Vision SPI Flash:

1. Ensure that the CrossLink-NX Voice and Vision device SRAM is erased by performing the steps in th Erasing the CrossLink-NX Voice and Vision SRAM Prior to Reprogramming section.
2. In the Radiant Programmer main interface, right-click the CrossLink-NX Voice and Vision row and select **Device Properties**.
3. Apply the settings below:
   a. Under Device Operation, select the options below:
      - **Port Interface – JTAG2SPI**
      - **Access Mode – Direct Programming**
      - **Operation – SPI Flash Erase, Program, Verify**
   b. Under Programming Options, select the bitstream file.
   c. For **SPI Flash Options**, select the **Macronix 25L12833F** device, as shown in Figure 9.4.

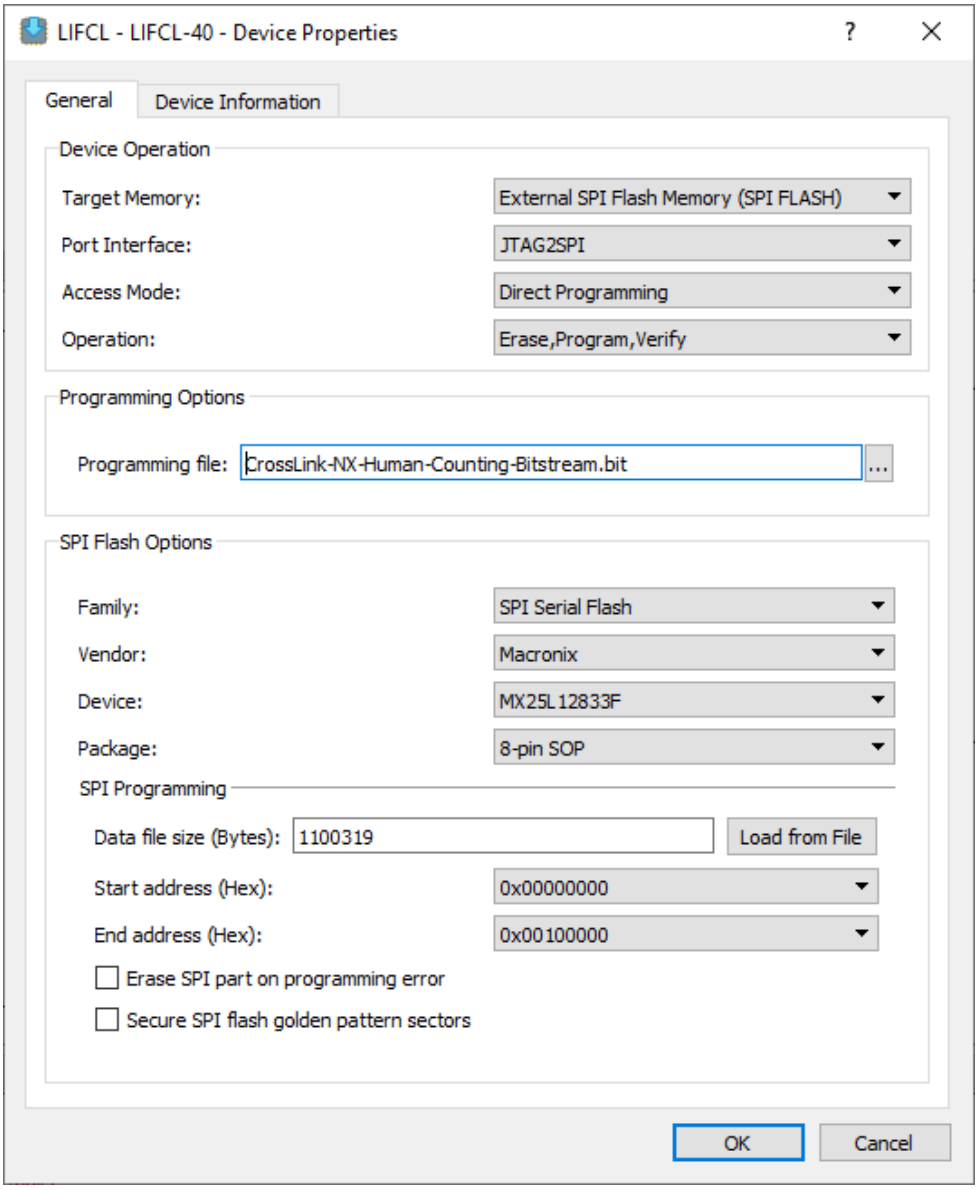

Figure 9.4. Radiant Programmer – Selecting Device Properties Options for CrossLink-NX Flashing

d. Click **Load from File** to update the Data file size (bytes) value.

e. Ensure that the following addresses are correct:
   - **Start Address (Hex) – 0x00000000**
   - **End Address (Hex) – 0x00100000**

4. Click **OK**.

5. Press the **SW4** push button switch before clicking the **Program** button, as shown in Figure 9.5. Hold it until you see the *Successful* message in the Radiant log window.

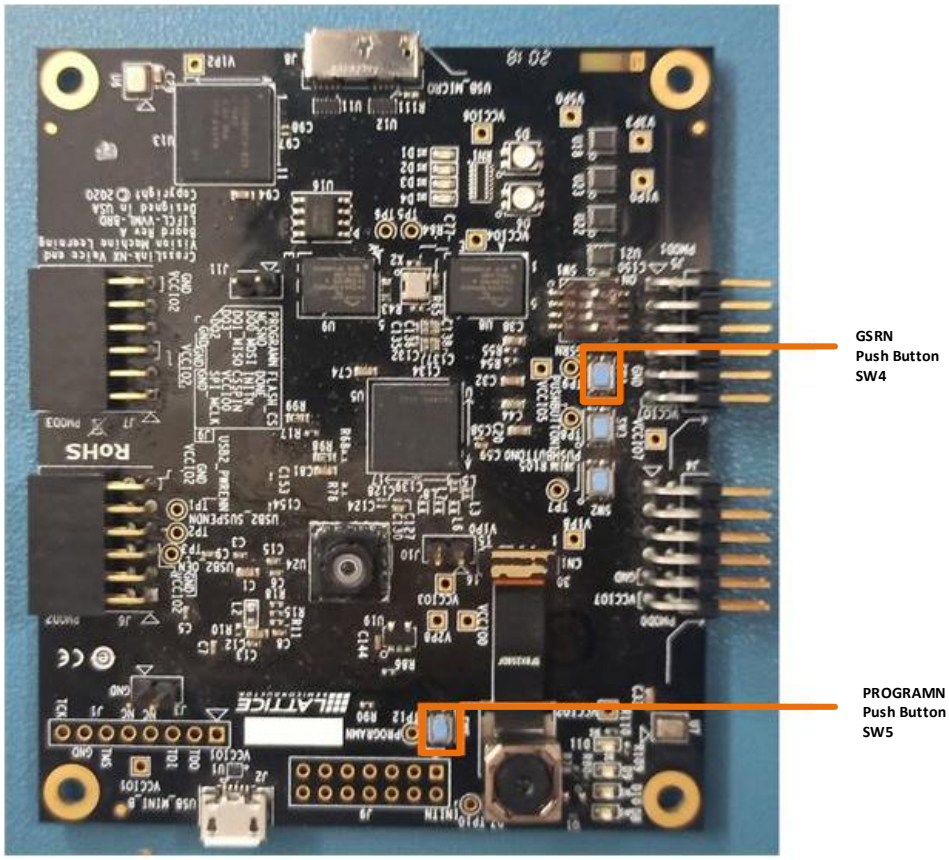

Figure 9.5. CrossLink-NX Voice and Vision Flashing Switch – SW4 Push Button

6. Click the **Program** button to start the programming operation.

7. After successful programming, the **Output** console displays the result, as shown in Figure 9.6.

Figure 9.6. Radiant Programmer – Output Console

9.1.3. Programming SensAI Firmware Binary to the CrossLink-NX Voice and Vision SPI Flash

9.1.3.1. Convert SensAI Firmware Binary to Hex

To program the CrossLink-NX Voice and Vision SPI flash:

1. Use the *bin2hex.exe* to convert the SensAI firmware binary file to hex format using command, as shown in Figure 9.7.
2. Make sure you do not have the target *.mcs* file present in the directory. If the target *.mcs* file is already present at the specified path, utility does not perform anything.

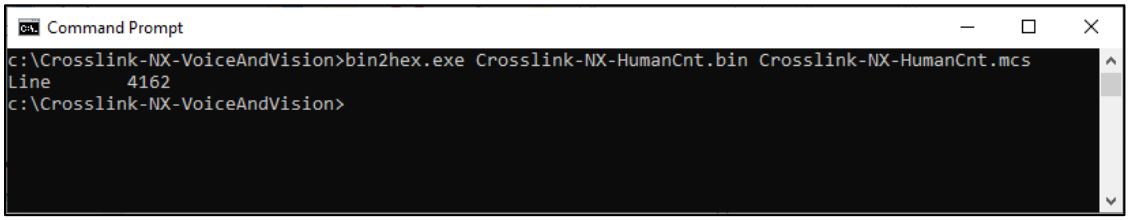

Figure 9.7. SensAI Bin to Hex – Convert SensAI Binary to Hex Format

9.1.3.2. Convert Flash SensAI Firmware Hex to Crosslink-NX Voice and Vision SPI Flash

To program the CrossLink-NX Voice and Vision SPI flash:

1. Ensure that the CrossLink-NX Voice and Vision device SRAM is erased by performing the steps in the Erasing the CrossLink-NX Voice and Vision SRAM Prior to Reprogramming section before flashing bitstream and SensAI firmware binary.
2. In the Radiant Programmer main interface, right-click the CrossLink-NX Voice and Vision row. Select **Device Properties** to open the dialog box, as shown in Figure 9.8.
3. Select **SPI FLASH** for **Target Memory**, **JTAG2SPI** for **Port Interface**, and **Direct Programming** for **Access Mode**.
4. For Programming File, select the **CrossLink-NX SensAI firmware** binary file after converting it to hex (*.mcs).
5. For SPI Flash Options, follow the configurations in Figure 9.8.

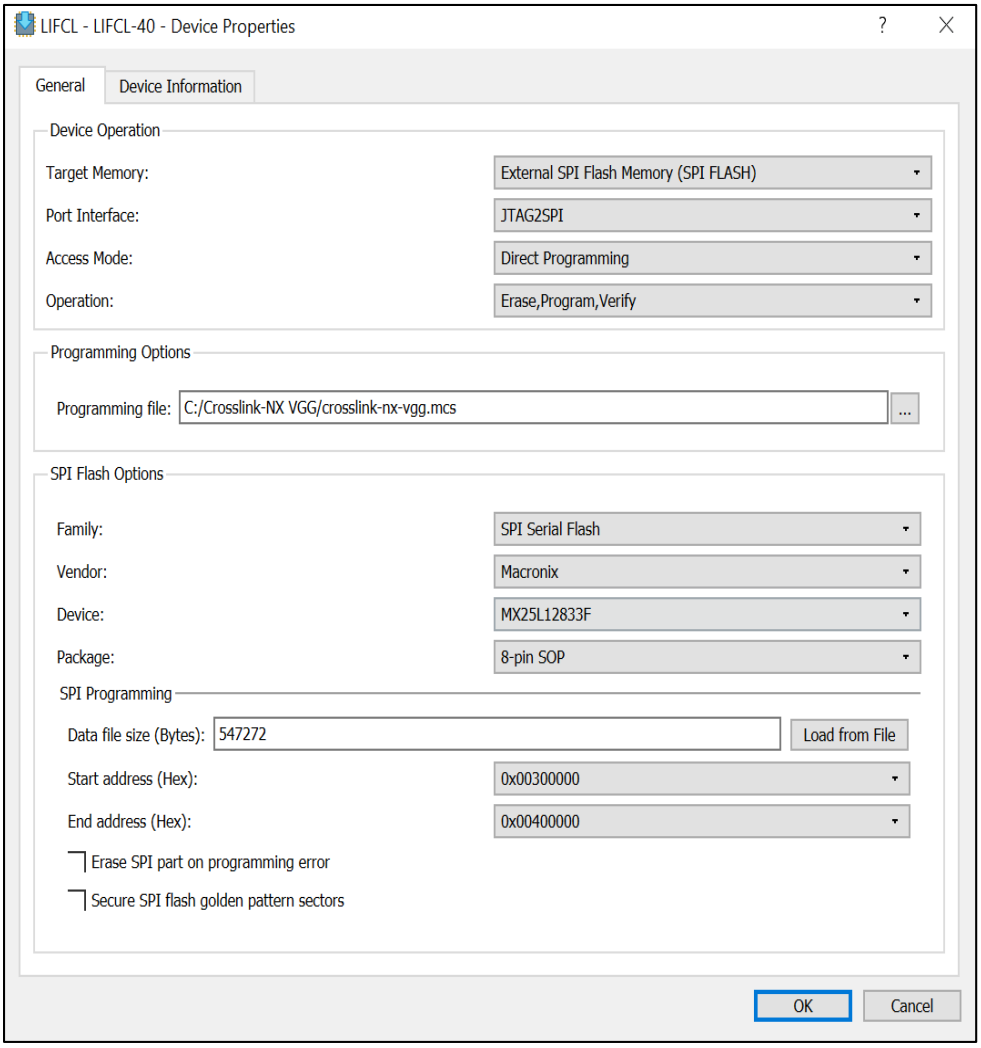

Figure 9.8. Radiant Programmer – Selecting Device Properties Options for CrossLink-NX Voice and Vision Flashing

6. Click **Load from File** to update the data file size (bytes) value.
7. Ensure that the following addresses are correct:
   - **Start Address (Hex) – 0x00300000**
   - **End Address (Hex) – 0x00400000**
8. Click **OK**.
9. Press the **SW4** push button switch. Click the **PROGRAMN** push button and hold it until you see the *Successful* message in the Radiant log window.
10. Click the **Program** button to start the programming operation.
11. After successful programming, the **Output** console displays the result, as shown in Figure 9.9.

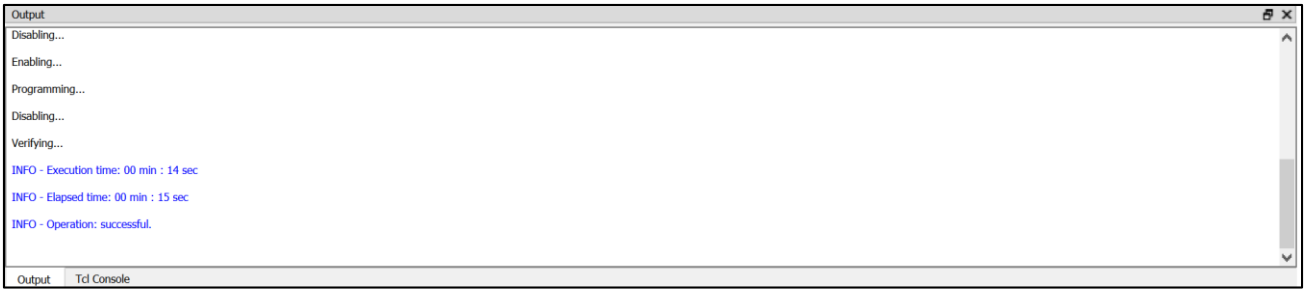

Figure 9.9. Radiant Programmer – Output Console

10. Running the Demo

To run the demo:

1. Cycle the power on the Voice and Visio board.
2. Make sure the position of SWITCH0 is **ON** to set FX3 to boot from I^2C EEPROM.
3. Connect the Voice and Vision board to the PC through the board's USB3 port.
4. Open the AMCap or VLC application and select the **FX3** device as source.
5. The camera image is displayed on monitors, as shown in Figure 10.1.

Figure 10.1. Running the Demo

6. The demo output contains bounding boxes for detected humans in a given frame. It also displays the total number of detected humans in a given frame on the PC.

Appendix A. Other Labelling Tools

Table A.1 provides information on other labelling tools.

Table A.1. Other Labelling Tools

Software	Platform	License	Reference	Converts To	Notes
annotate-to-KITTI	Ubuntu/Windows (Python based utility)	No License (Open source GitHub project)	https://github.com/SaiPrajwal95/annotate-to-KITTI	KITTI	Python based CLI utility that you can clone and launch.
LabelBox	JavaScript, HTML, CSS, Python	Cloud or On-premise, some interfaces are Apache-2.0	https://www.labelbox.com/	json, csv, coco, voc	Web application
LabelMe	Perl, JavaScript, HTML, CSS, On Web	MIT License	http://labelme.csail.mit.edu/Release3.0/	xml	Converts only jpeg images
Dataturks	On web	Apache License 2.0	https://dataturks.com/	json	Converts to json format but creates single json file for all annotated images
LabelImg	ubuntu	OSI Approved:: MIT License	https://mlnotesblog.wordpress.com/2017/12/16/how-to-install-labelimg-in-ubuntu-16-04/	xml	Need to install dependencies given in reference
Dataset_ annotator	Ubuntu	2018 George Mason University Permission is hereby granted, Free of charge	https://github.com/omenyayl/dataset-annotator	json	Need to install app_image and run it by changing permissions

References

- Google TensorFlow Object Detection GitHub
- Pretrained TensorFlow Model for Object Detection
- Python Sample Code for Custom Object Detection
- Train Model Using TensorFlow

Technical Support Assistance

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

Revision History

Revision 1.0, December 2020

Section	Change Summary
All	Initial release.

www.latticesemi.com