Object Counting using Mobilenet CNN Accelerator IP

Reference Design

FPGA-RD-02067-1.0

October 2019

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
CKPT	Checkpoint
CNN	Convolutional Neural Network
EVDK	Embedded Vision Development Kit
FPGA	Field-Programmable Gate Array
ML	Machine Learning
MLE	Machine Learning Engine
SPI	Serial Peripheral Interface
VIP	Video Interface Platform

1 Introduction

This document describes the Human Counting Design process of Mobilnet using the ECP5™ EVDK FPGA platform. Human Counting is a subset of the generic Object Counting base design.

1.1 Design Process Overview

The design process involves the following steps:

1.1.1 Training Model

- Setting up the basic environment
- Preparing the dataset
  - Preparing 224 x 224 Image
  - Labeling dataset of human bounding box
- Training the machine
  - Training the machine and creating the checkpoint data
- Creating Frozen file (*.pb)

1.1.2 Neural Network Compiler

- Creating Binary file with Lattice SensAI™ 2.1 program

1.1.3 FPGA Design

- Creating FPGA Bitstream file

1.1.4 FPGA Bitstream and Quantized Weights and Instructions

- Flashing Binary and Bitstream files
  - Binary File to MicroSD
  - Bitstream to Flash Memory on VIP Board

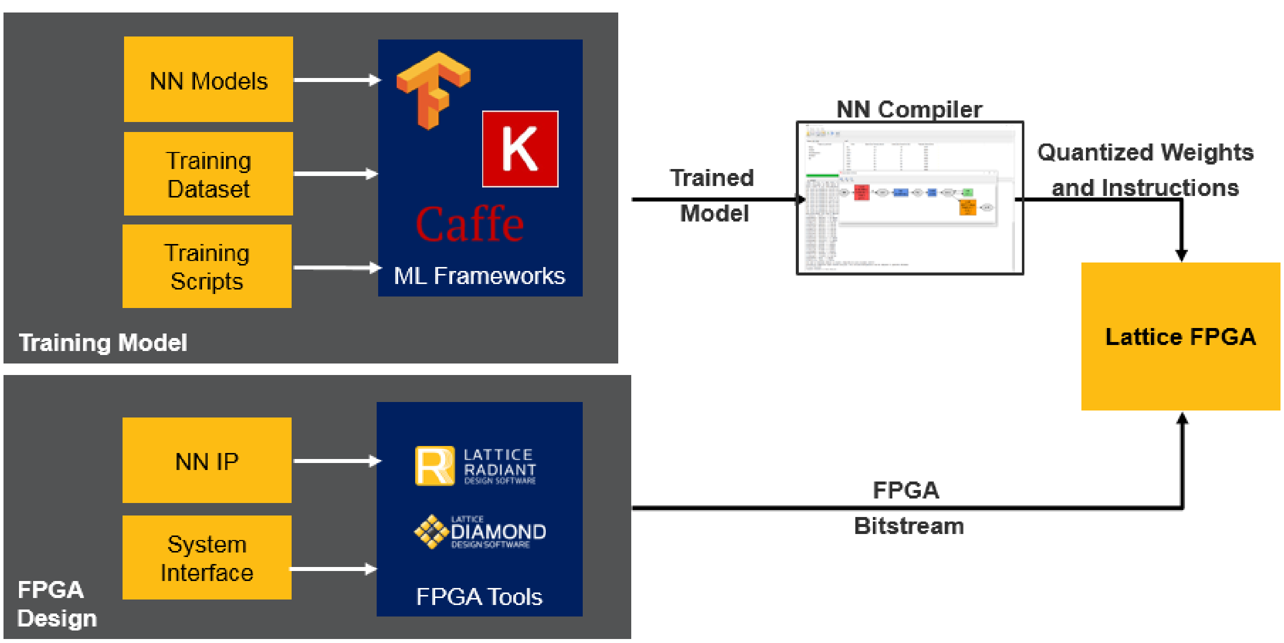

Figure 1.1. Lattice Machine Learning Design Flow

2 Setting Up the Basic Environment

2.1 Tools and Hardware Requirements

This section describes the required tools and environment setup for FPGA Bitstream and Flashing.

2.1.1 Lattice Tools

- Lattice Diamond® Tool – Refer to http://www.latticesemi.com/latticediamond.
- Lattice Diamond Programmer – Refer to http://www.latticesemi.com/programmer.
- Lattice SensAI Compiler v2.1 – Refer to https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler.

2.1.2 Win32 MicroSD Disk Imager

Refer to https://sourceforge.net/projects/win32diskimager/

2.1.3 Hardware

- ECP5 FPGA VIP Board – Refer to http://www.latticesemi.com/Solutions/Solutions/SolutionsDetails02/VIP.

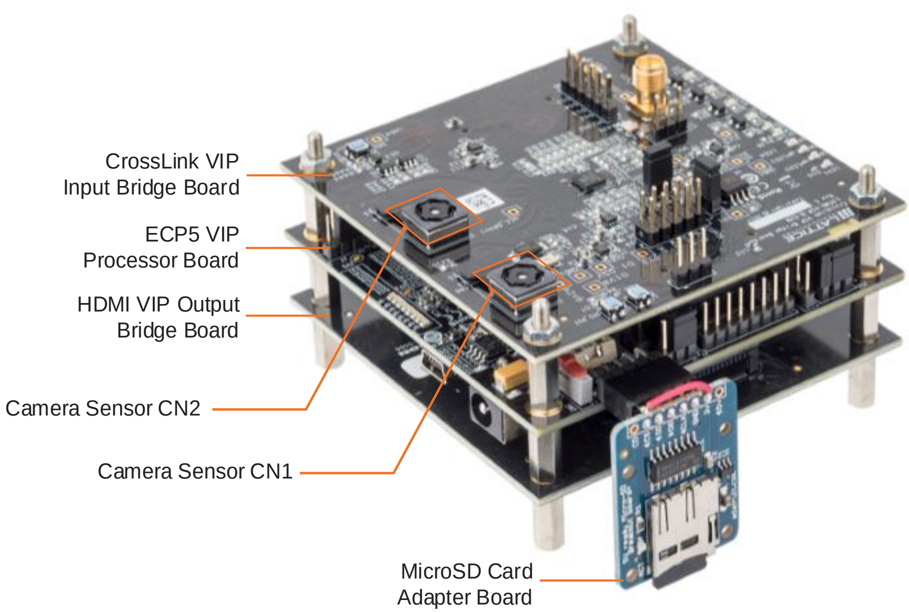

Figure 2.1. Lattice EVDK with MicroSD Card Adapter Board

2.2 Setting Up the Linux Environment for Machine Training

This section describes the steps for NVIDIA GPU drivers and/or libraries for 64-bit **Ubuntu 16.04 OS**.

Note: NVIDIA library and TensorFlow version is dependent on PC and Ubuntu/Windows version.

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

2.2.1.1 Installing the CUDA Toolkit

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

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

```
(base) sib:~/kishan$ 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    514      0  0:00:05  0:00:05 --:--:--   584
(base) sib:~/kishan$ _
```

Figure 2.2. Download CUDA Repo

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

```
(base) sib:~/kishan$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
Selecting previously unselected package cuda-repo-ubuntu1604.
(Reading database ... 288236 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) ...
(base) sib:~/kishan$ _
```

Figure 2.3. Install CUDA Repo

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

```
(base) sib:~/kishan$ sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir /tmp/tmp.oqotmWcGn0 --no-auto-check-trustdb --trust-model
ng /etc/apt/trusted.gpg --keyring /etc/apt/trusted.gpg.d/diesch-testing.gpg --keyring /etc/apt/trusted.gpg.d/george-edison55-cmake-3_x.gpg -
--fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
gpg: key 7FA2AF80: "cudatools <cudatools@nvidia.com>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
```

Figure 2.4. Fetch Keys

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

```
(base) sib:~/kishan$ sudo apt-get update
Ign http://dl.google.com stable InRelease
Ign http://archive.ubuntu.com trusty InRelease
Ign http://extras.ubuntu.com trusty InRelease
Hit https://deb.nodesource.com trusty InRelease
Ign http://archive.canonical.com precise InRelease
Hit http://ppa.launchpad.net trusty InRelease
```

Figure 2.5. Update Ubuntu Packages Repositories

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

```
(base) sib:~/kishan$ sudo apt-get install cuda-9-0
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 2.6. CUDA Installation

2.2.1.2 Installing the cuDNN

To install the cuDNN:

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

```
$ tar xvf cudnn-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*
```

```
k$ tar xvf cudnn-9.0-linux-x64-v7.1.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.1.4
cuda/lib64/libcudnn_static.a
k$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
k$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
k$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
k$ _
```

Figure 2.7. cuDNN Library Installation

2.2.2 Setting Up the Environment for Training and Model Freezing Scripts

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

2.2.2.1 Installing the Anaconda Python

To install the Anaconda and Python 3:

1. Go to https://www.anaconda.com/distribution/#download-section
2. Download Python 3 version of Anaconda for Linux.

```
sib:~/kishan$ wget https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
--2019-04-18 11:34:54--  https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3, 2606:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 685906562 (654M) [application/x-sh]
Saving to: 'Anaconda3-2019.03-Linux-x86_64.sh'

100%[==========================================================================================
```

Figure 2.8. Anaconda Package Download

3. Run the command below to install the Anaconda environment:

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

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

```
sib:~/kishan$ sh Anaconda3-2019.03-Linux-x86_64.sh

Welcome to Anaconda3 2019.03

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

Figure 2.9. Anaconda Installation

4. Accept the license.

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

Figure 2.10. Accept License Terms

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

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

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

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

[/home/sibridge/anaconda3] >>> /home/sibridge/kishan/anaconda3_
```

Figure 2.11. Confirm/Edit Installation Location

6. After installation, enter **No** as shown in Figure 2.12.

```
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> no_
```

Figure 2.12. Launch/Initialize Anaconda Environment on Installation Completion

2.2.3 Installing the TensorFlow v1.12

To install the **TensorFlow v1.12**:

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

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

```
sib:~/kishan$ source anaconda3/bin/activate
(base) sib:~/kishan$ _
```

Figure 2.13. Anaconda Environment Activation

2. Install the TensorFlow by running the command below:

```
$ conda install tensorflow-gpu==1.12.0
```

```
(base) sib:~/kishan$ conda install tensorflow-gpu==1.12.0
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - tensorflow-gpu==1.12.0
```

Figure 2.14. TensorFlow Installation

3. After installation, enter **Y** as shown in Figure 2.15.

```
  wurlitzer                                   1.0.2-py37_0 --> 1.0.2-py36_0
  xlrd                                        1.2.0-py37_0 --> 1.2.0-py36_0
  xlwt                                        1.3.0-py37_0 --> 1.3.0-py36_0
  zict                                        0.1.4-py37_0 --> 0.1.4-py36_0
  zipp                                        0.3.3-py37_1 --> 0.3.3-py36_1


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

Figure 2.15. TensorFlow Installation Confirmation

Figure 2.16 shows TensorFlow installation is complete.

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) sib:~/kishan$ _
```

Figure 2.16. TensorFlow Installation Completion

2.2.4 Installing the Python Package

To install the Python package:

1. Install Easydict by running the command below:

    $ conda install –c conda-forge easydict

```
(base) sib:~/kishan$ conda install -c conda-forge easydict
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - easydict
```

Figure 2.17. Easydict Installation

2. Install Joblib by running the command below:

    $ conda install joblib

```
(base) sib:~/kishan$ conda install joblib
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - joblib
```

Figure 2.18. Joblib Installation

3. Install Keras by running the command below:

    $ conda install keras

```
(base) sib:~/kishan$ conda install keras
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - keras
```

Figure 2.19. Keras Installation

4. Install OpenCV by running the command below:

```
$ conda install opencv
```

```
(base) sib:~/kishan$ conda install opencv
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - opencv
```

Figure 2.20. OpenCV Installation

5. Install Pillow by running the command below:

```
$ conda install pillow
```

```
(base) sib:~/kishan$ conda install pillow
Collecting package metadata: done
Solving environment: done

# All requested packages already installed.

(base) sib:~/kishan$ _
```

Figure 2.21. Pillow Installation

3 Preparing the Dataset

This chapter 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.

```
OIDv4_ToolKit/
├── classes.txt
├── images
│   ├── classes.png
│   ├── rectangle.png
│   └── visualizer_example.gif
├── LICENSE
├── main.py
├── modules
│   ├── bounding_boxes.py
│   ├── csv_downloader.py
│   ├── downloader.py
│   ├── image_level.py
│   ├── parser.py
│   ├── show.py
│   ├── test.py
│   └── utils.py
├── README.md
└── requirements.txt
```

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
```

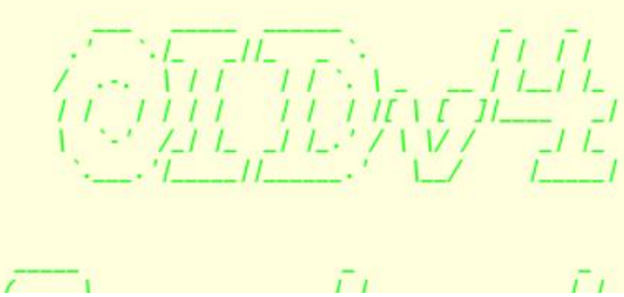

Figure 3.4. Dataset Downloading Logs

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. However, the downloaded dataset is not in exact KITTI format. Convert the annotation to KITTI format.

```
$ 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

It has class ID followed by truncated, occluded, alpha, Xmin, Ymin, Xmax, Ymax.

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

```
augmentation
    data_aug
        __init__.py
        bbox_util.py
        data_aug.py
    augmentation.py
    config.py
    README.md
```

Figure 3.11. Augmentation Directory Stucture

- data_aug – It 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

2. 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

3. 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

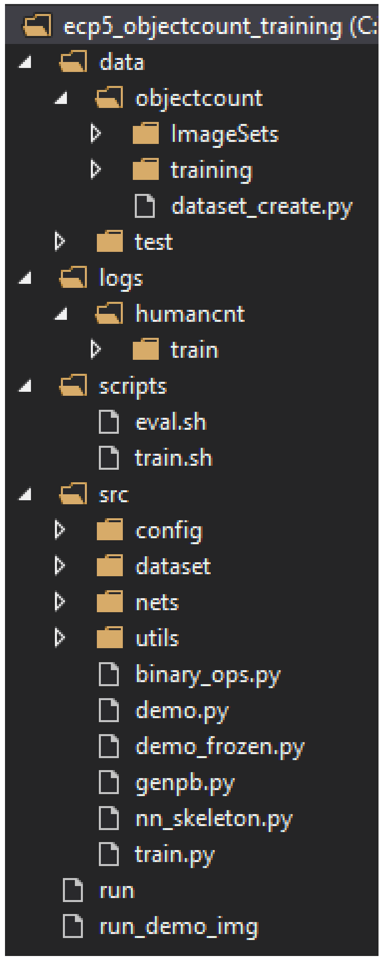

Figure 4.1. Training Code Directory Structure

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 Presence Detection design. The Neural Network model of the Human Presence Detection design uses MobileNet NN base model and the detection layer of SqueezeDet model.

Table 4.1. Human Counting Training Network Topology

Image Input (224 x 224 x 3)		
Fire 1	Conv3 - 32	Conv3 - # where: • *Conv3* = 3 x 3 Convolution filter Kernel size • *#* = The number of filter DWConv3 - 32- # where: • *DWConv3* = Depthwise convolution filter with 3x3 size • *#* = The number of filter Conv1 - 32- # where: • *Conv1* = 1 x 1 Convolution filter Kernel size • *#* = The number of filter For example, Conv3 - 16 = 16 3 x 3 convolution filters BN – Batch Normalization
	BN	
	Relu	
Fire 2	DWConv3 - 32	
	BN	
	Relu	
	Conv1 - 32	
	BN	
	Relu	
Fire 3	DWConv3 - 32	
	BN	
	Relu	
	Maxpool	
	Conv1 - 32	
	BN	
	Relu	
Fire 4	DWConv3 - 32	
	BN	
	Relu	
	Conv1 - 64	
	BN	
	Relu	
Fire 5	DWConv3 - 64	
	BN	
	Relu	
	Maxpool	
	Conv1 - 64	
	BN	
	Relu	
Fire 6	DWConv3 - 64	
	BN	
	Relu	
	Conv1 - 128	
	BN	
	Relu	
Fire 7	DWConv3 - 128	
	BN	
	Relu	
	Maxpool	
	Conv1 - 128	

	BN	
	Relu	
Conv12	Conv3 – 42	

- Human Count Network structure consists of 7 fire layers followed by one convolution layer. Fire layer contains convolution, depth wise convolution, batch normalization and relu layers with pooling layer only in fire 3, fire 5 and fire 7. Layers fire 2, fire 4 and fire 6 do not contain pooling.
- Fire 1 layer has stride=2 in convolution layer while all other conv operation in fire layers has stride = 1
- 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 number of filters (sometimes referred as kernels) which convolves with input layer/image and generates activation map (that is feature map). This filter is an array of numbers (the numbers are called weights or parameters). Each of these filters can be thought of as feature identifiers, like straight edges, simple colors, and curves 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 (for example, curve) it is looking for is in the input volume.
    - Relu (Activation layer)
      After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (element-wise, multiplications and summations). In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. The ReLu layer applies the function f(x) = max (0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.
    - Pooling Layer
      After some ReLu layers, 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 max pooling being the most popular. This basically takes a filter (normally by size 2 x 2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub region that the filter convolves around.
      The intuitive reasoning behind this layer is that once you know 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 the other features. As you can imagine, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. The first is that the number of parameters or weights is reduced by 75%, thus lessening the computation cost. Second is that it controls over fitting. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of over fitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.

    - Batch Normalization
      Batch Normalization layer reduces the internal covariance shift. In order to train a neural network, perform preprocessing to the input data. For example, you can normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). The reason being is to prevent the early saturation of nonlinear activation functions (like the sigmoid function), assuring that all input data is in the same range of values, and so on.
      But the problem appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt them to a new distribution in every training step. This problem is known as internal covariate shift.

The Batch Normalization layer forces the input of every layer to have approximately the same distribution in every training step by following below process during training time:

- Calculate the mean and variance of the layers input.
- Normalize the layer inputs using the previously calculated batch statistics.
- Scales and shifts in order to obtain the output of the layer.

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

- Depthwise Convolution and 1 x 1 Convolution Layer

  Depthwise Convolutions are used to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1 x 1 convolution, is then used to create a linear combination of the output of the depthwise layer.

  Depthwise Convolution is extremely efficient relative to standard convolution. However, it only filters input channels, it does not combine them to create new features. So an additional layer that computes a linear combination of the output of depthwise convolution via 1 x 1 convolution is needed in order to generate these new features.

  A 1 x 1 convolutional layer that compresses an input tensor with large channel size to one with the same batch and spatial dimension, but smaller channel size. Given a 4D input tensor and a filter tensor shape [filter_height, filter_width, in_channels, channel_multiplier] containing in_channels convolutional filters of depth 1, depthwise_conv2d applies a different filter to each input channel, then concatenates the results together. The output has in_channels * channel_multiplier channels.

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 first extracts feature maps, overlays them with a WxH grid and at each cell 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)xIOU)
  - 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 x 14 grid
    - 7 anchor boxes per grid
      - 6 values per anchor box. It consists of:
        - 4 bounding box coordinates (x, y, w, h)
        - 1 class probability
        - 1 confidence score

So in total, 14 x 14 x 7 x 6 = 8232 output values.

4.2.3 Training Code Overview

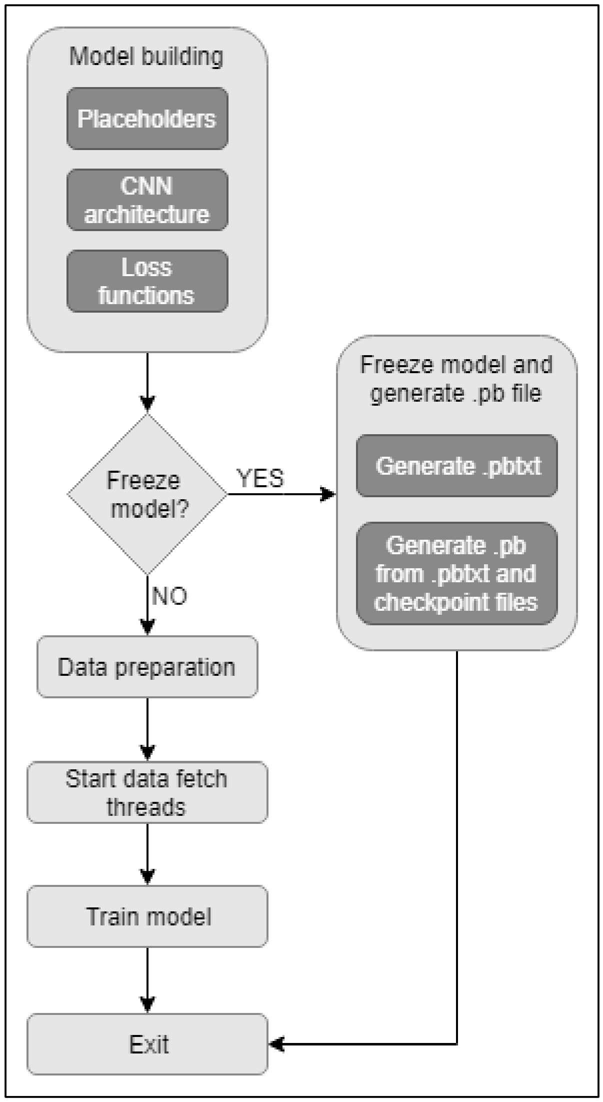

Figure 4.2. Training Code Flow Diagram

Training Code is divided into the following parts:

- Model config
- Model building
- Model freezing
- Data preparation
- Training for Overall Execution Flow

Details of each can be found in subsequent sections.

4.2.3.1 Model Config

The design uses Kitti dataset and SqueezeDet model. *kitti_squeezeDet_config()* maintains all the configurable parameters for the model. Below is 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 three pooling layers and fire 1 layer has stride=2, grid dimension would be 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 grid size changes

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

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.
- If you want to run network on your own dataset, you need to adjust the anchor sizes. Anchors are kind of 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.
- In order to determine anchor shapes, first load all ground truth boxes and pictures, and if your images do not have all 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. (I.e. you can just 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 like 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 constructor builds model which is divided into the following sections:

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

Forward Graph

- CNN architecture consist of Convolution, Batch Normalization, ReLu, Maxpool, and 1 x 1 depthwise convolution layers.
- Forward Graph consists of seven fire layers as described in Table 4.1.

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

######################################################################
# Quantization layers
######################################################################
if True: # 16b weight (no quant); 8b activation
    fl_w_bin = 16
    fl_a_bin = 16
    ml_w_bin = 16
    ml_a_bin = 16
    sl_w_bin = 16
    # 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 – Quantization Setting

```
if True: # Stride 2
    fire1 = self._fire_layer  ('fire1', self.image_input, oc=depth[0], freeze=False, w_bin=fl_w_bin,
                        a_bin=fl_a_bin, pool_en=False, min_rng=min_rng, max_rng=max_rng, stride=2)
else: # Max pool
    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, stride=1)
fire2 = self._mobile_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)
if False: # stride 2
    fire3 = self._mobile_layer('fire3', fire2, oc=depth[2], freeze=False, w_bin=ml_w_bin,
                        a_bin=ml_a_bin, pool_en=False, min_rng=min_rng, max_rng=max_rng, stride=2)
else:    # Max pool
    fire3 = self._mobile_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, stride=1)
fire4 = self._mobile_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)
fire5 = self._mobile_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)
fire6 = self._mobile_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)
fire7 = self._mobile_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)
fire_o = fire7
```

Figure 4.9. Code Snippet – Forward Graph Fire Layers with Mobilenet

```
def _mobile_layer(self, layer_name, inputs, oc, stddev=0.01, freeze=False, w_bin=16, a_bin=16, pool_en=True, min_rng=-0.5, max_rng=0.5, stride=1):
  with tf.variable_scope(layer_name):
    ex3x3 = self._conv_layer('dw_conv3x3', inputs, filters=int(inputs.get_shape()[3]), size=3, stride=stride,
      padding='SAME', stddev=stddev, freeze=freeze, relu=False, w_bin=w_bin, depthwise=True) # <----

    tf.summary.histogram('dw_before_bn', ex3x3)
    ex3x3 = self._batch_norm('dw_bn', ex3x3) # <----
    tf.summary.histogram('dw_before_relu', ex3x3)
    ex3x3 = self.binary_wrapper(ex3x3, a_bin=a_bin, min_rng=min_rng, max_rng=max_rng) # <---- relu
    tf.summary.histogram('dw_after_relu', ex3x3)

    if pool_en:
        pool = self._pooling_layer('pool', ex3x3, size=2, stride=2, padding='SAME')
    else:
        pool = ex3x3
    tf.summary.histogram('dw_pool', pool)

    ex1x1 = self._conv_layer('conv1x1', pool, filters=oc, size=1, stride=1,
        padding='SAME', stddev=stddev, freeze=freeze, relu=False, w_bin=w_bin, depthwise=False) # <----

    tf.summary.histogram('1x1_before_bn', ex1x1)
    ex1x1 = self._batch_norm('1x1_bn', ex1x1) # <----
    tf.summary.histogram('1x1_before_relu', ex1x1)
    ex1x1 = self.binary_wrapper(ex1x1, a_bin=a_bin, min_rng=min_rng, max_rng=max_rng) # <---- relu
    tf.summary.histogram('1x1_after_relu', ex1x1)
    return ex1x1
```

Figure 4.10. Code Snippet – '_mobile_layer'

```
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)
print('self.preds:', self.preds)
```

Figure 4.11. 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 x 14 x 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, which means the 42 channels are not stored consecutively but are scattered all over the place and you need to sort these out somehow. Figure 4.12 and Figure 4.13 explain the details.

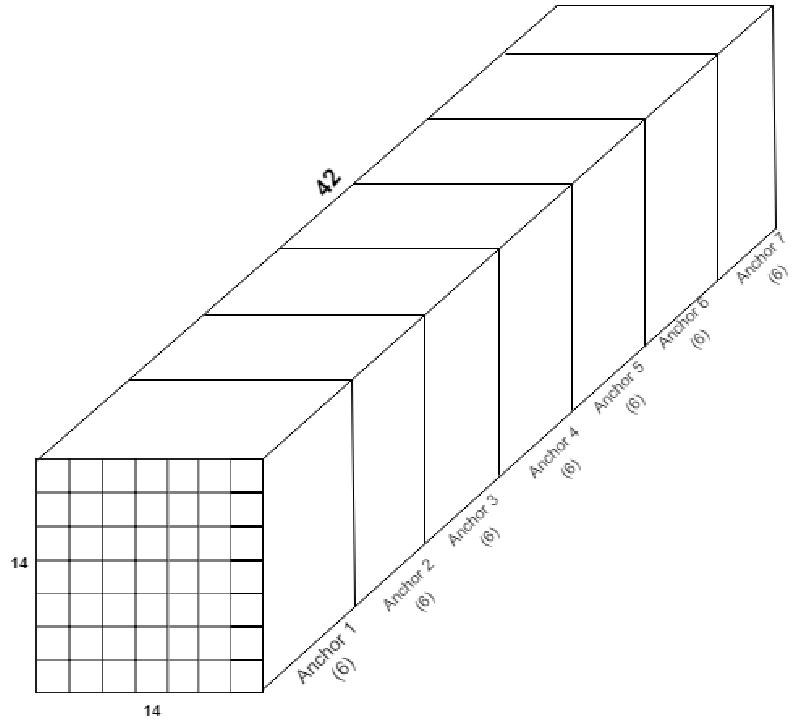

Figure 4.12. Grid Output Visualization #1

For each grid cell values are aligned as shown in Figure 4.13.

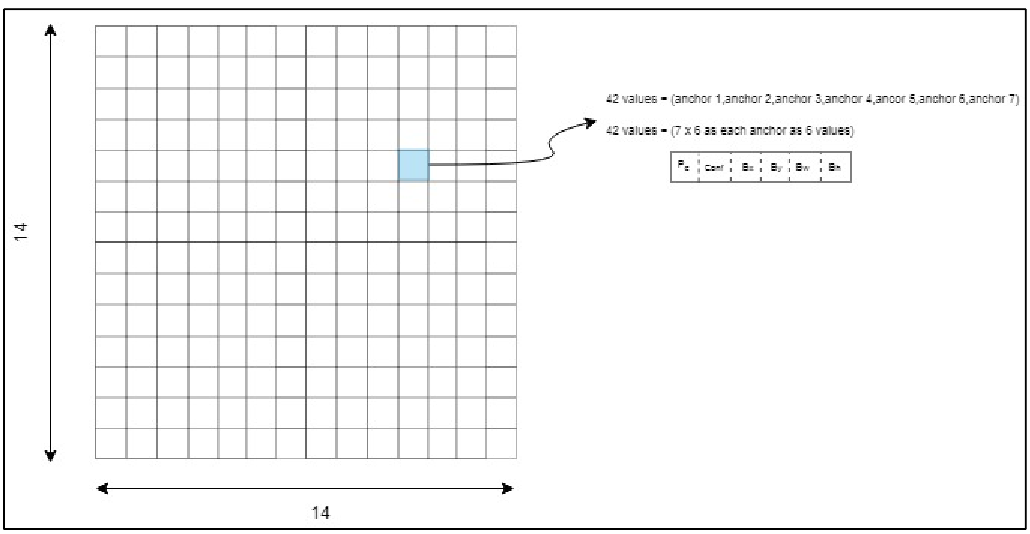

Figure 4.13. Grid Output Visualization #2

As shown in the code below, the output from conv12 layer (4d array of batch size x 14 x 14 x 42) needs to be sliced with 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 4.14. Code Snippet – Interpret Output Graph

For confidence score, this must be a number between 0 and 1, so 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 anchor box and 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 need to be minimized. In order 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 4.15. Code Snippet – Bbox Loss

- Confidence Score
  - To obtain meaningful confidence score, each box's predicted value is regressed against the of the real and the predicted box. During training, compare ground truth bounding boxes with all anchors and assign them to the anchors that have the largest overlap (IOU) with each of them.
  - The reason being, to 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 4.16. Code Snippet – Confidence Loss

- Class
  - The last part of the loss function is just cross-entropy loss for classification 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 4.17. Code Snippet – Class Loss

So, 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 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 4.18. Code Snippet – Training

sess.run feeds the data and 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 main thread only.

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 command prompt.

   The Model works on 224 x 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 4.19.

```
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]
```

Figure 4.19. 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 4.20. Training Code Snippet for Dataset Path

2. Create a train.txt.

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

```
k$ python dataset_create.py
k$ _
```

Figure 4.21. Create File for Dataset train.txt

   Notes:

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

3. Modify the training script.

   Training script at @scripts/train.sh is used to trigger training. Figure 4.22 shows the input parameters which can be configured.

```
python ./src/train.py \
  --dataset=KITTI \
  --pretrained_model_path=$PRETRAINED_MODEL_PATH \
  --data_path=$TRAIN_DATA_DIR \
  --image_set=train \
  --train_dir="$TRAIN_DIR/train" \
  --net=$NET \
  --summary_step=100 \
  --checkpoint_step=500 \
  --max_steps=2000000 \
  --gpu=$GPUID
```

Figure 4.22. Training Input Parameter

- $TRAIN_DATA_DIR – dataset directory path. */data/humandet* 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/bias_add:0", shape=(20, 14, 14, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
ANCHORS: 1372
max person: 22
Model statistics saved to ./logs/humancnt/train/model_metrics.txt.
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.755
pciBusID: 0000:01:00.0
totalMemory: 10.73GiB freeMemory: 10.34GiB
2019-09-16 14:55:32.019017: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512] Adding visible gpu devices: 0
2019-09-16 14:55:35.451832: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
conf_loss: 26.6821231842041, bbox_loss: 12.065683364868164, class_loss: 0.0
2019-09-16 14:55:36.940385: step 0, loss = 38.75 (5.8 images/sec; 3.466 sec/batch)
2019-09-16 14:55:39.423756: step 10, loss = 7.28 (240.5 images/sec; 0.083 sec/batch)
2019-09-16 14:55:40.269647: step 20, loss = 2.74 (241.6 images/sec; 0.083 sec/batch)
2019-09-16 14:55:41.106887: step 30, loss = 2.11 (239.2 images/sec; 0.084 sec/batch)
2019-09-16 14:55:41.949579: step 40, loss = 2.21 (241.7 images/sec; 0.083 sec/batch)
2019-09-16 14:55:42.786778: step 50, loss = 1.87 (241.1 images/sec; 0.083 sec/batch)
2019-09-16 14:55:43.613277: step 60, loss = 2.03 (241.3 images/sec; 0.083 sec/batch)
2019-09-16 14:55:44.448544: step 70, loss = 1.96 (240.3 images/sec; 0.083 sec/batch)
2019-09-16 14:55:45.289586: step 80, loss = 1.93 (240.5 images/sec; 0.083 sec/batch)
2019-09-16 14:55:46.123831: step 90, loss = 2.19 (242.3 images/sec; 0.083 sec/batch)
conf_loss: 1.1395012140274048, bbox_loss: 1.0199577808380127, class_loss: 0.0
```

Figure 4.23. 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 4.24. TensorBoard – Generated Link

7. Check the training status on TensorBoard.

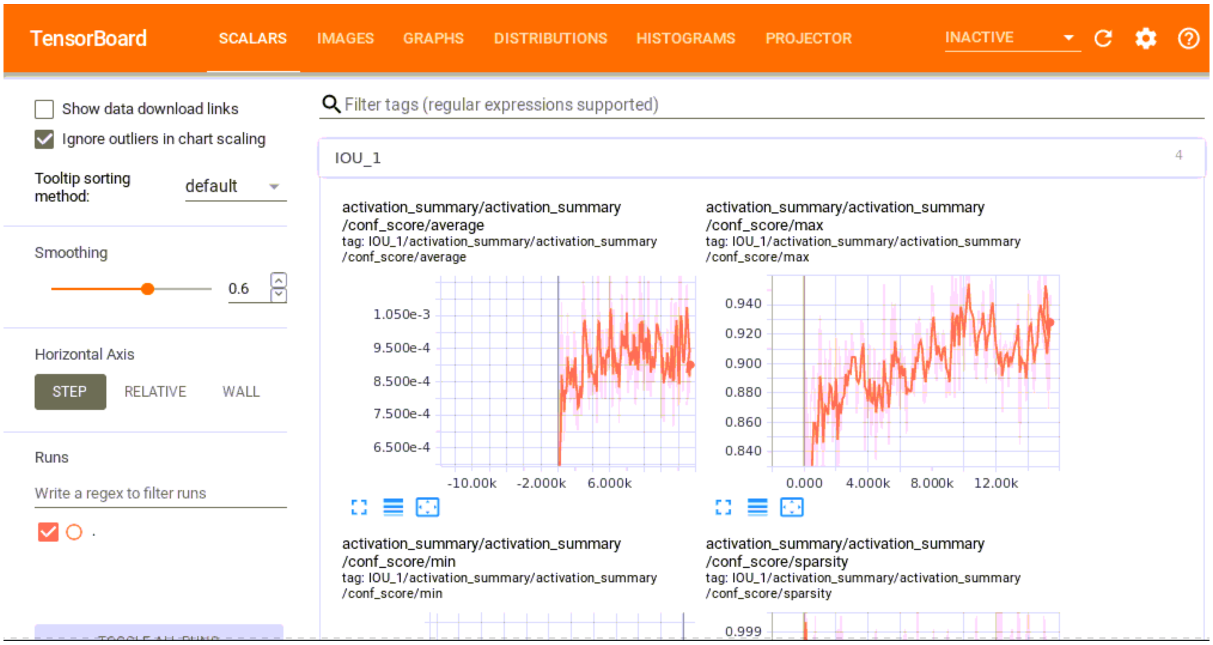

Figure 4.25. TensorBoard

Figure 4.25 shows the image menu of TensorBoard.

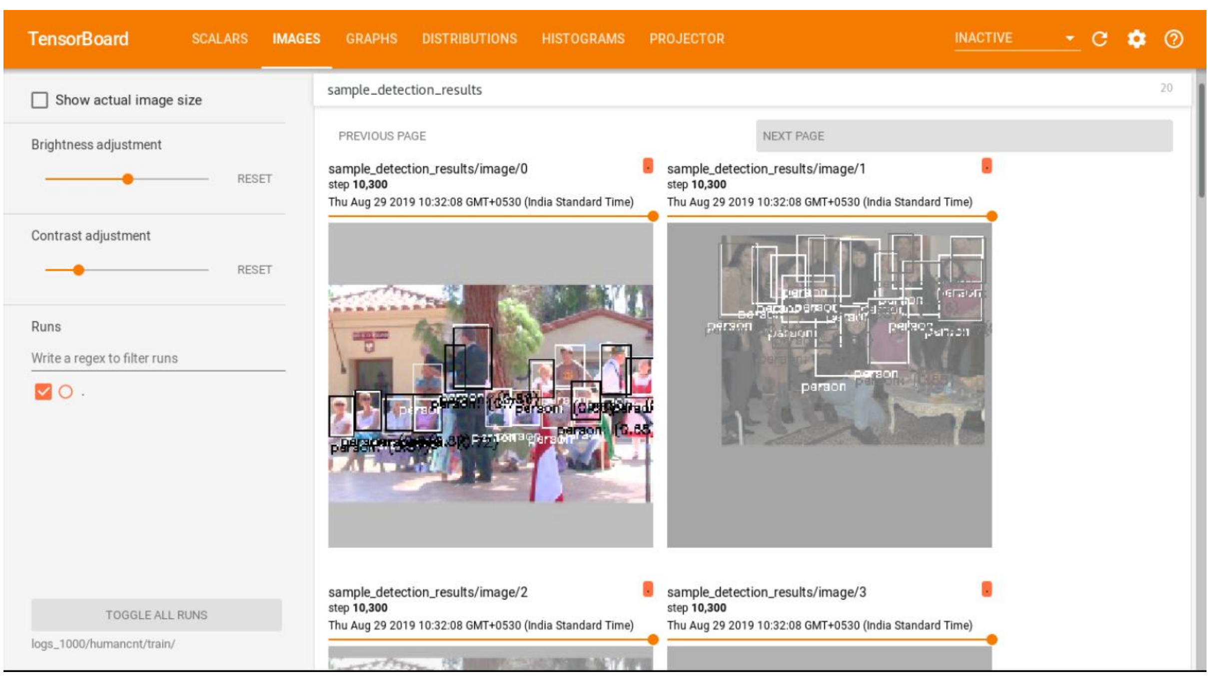

Figure 4.26. Image Menu of TensorBoard

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).

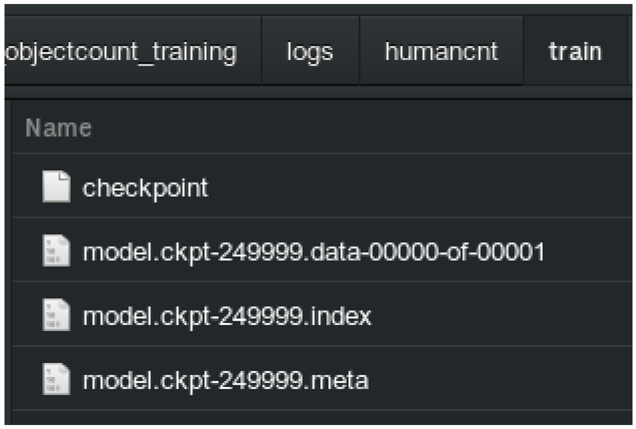

Figure 4.27. Example of Checkpoint Data Files at Log Folder

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 below command from the training code's root directory.

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

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

```
earth:$ python src/genpb.py --ckpt_dir logs/humancnt/train/ --freeze
genrating pbtxt
self.preds: Tensor("conv12/bias_add:0", shape=(20, 14, 14, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
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.

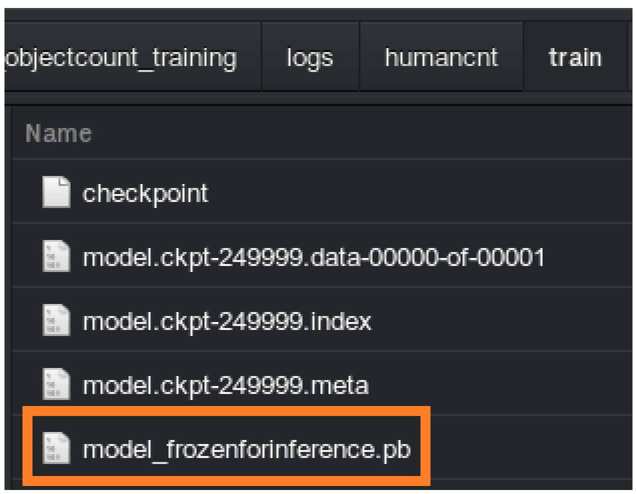

Figure 5.2. Frozen pb File

6 Creating Binary File with Lattice SensAI

This chapter describes how to generate binary file using the Lattice SensAI version 2.1 program.

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

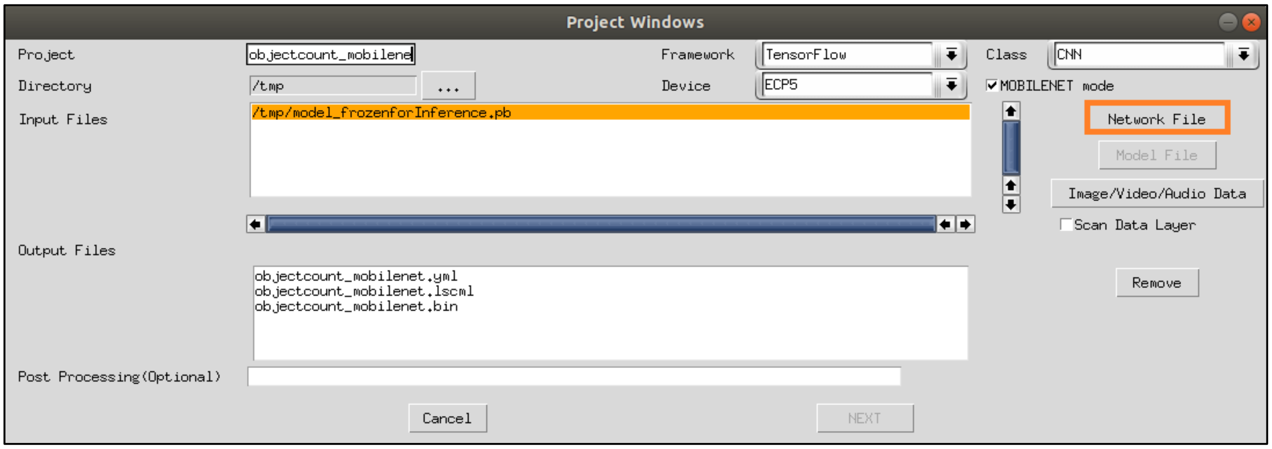

Figure 6.2. SensAI –Network File Selection

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

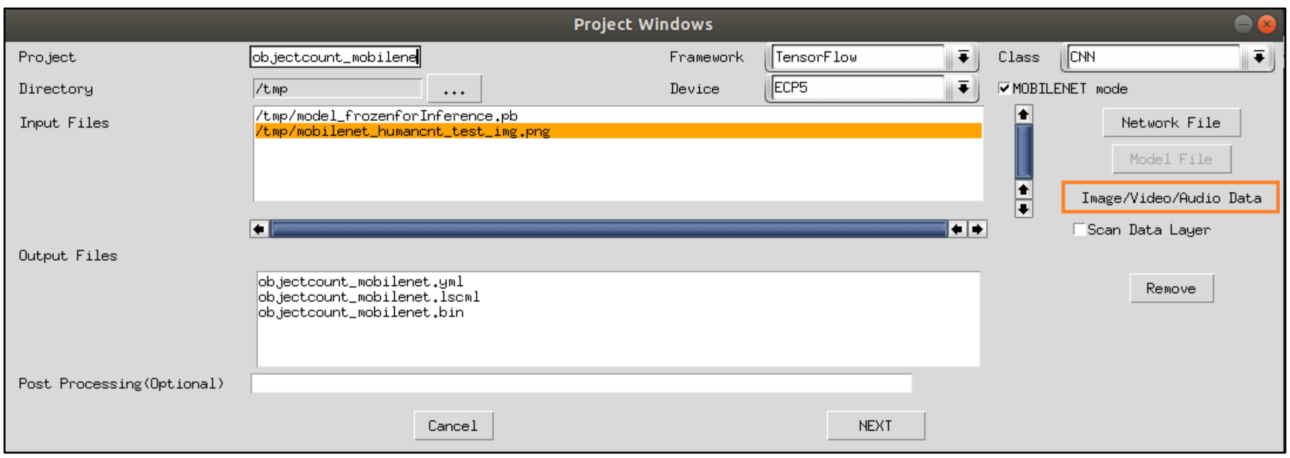

Figure 6.3. SensAI –Image Data File Selection

5. Click **NEXT**.
6. Configure your project settings.

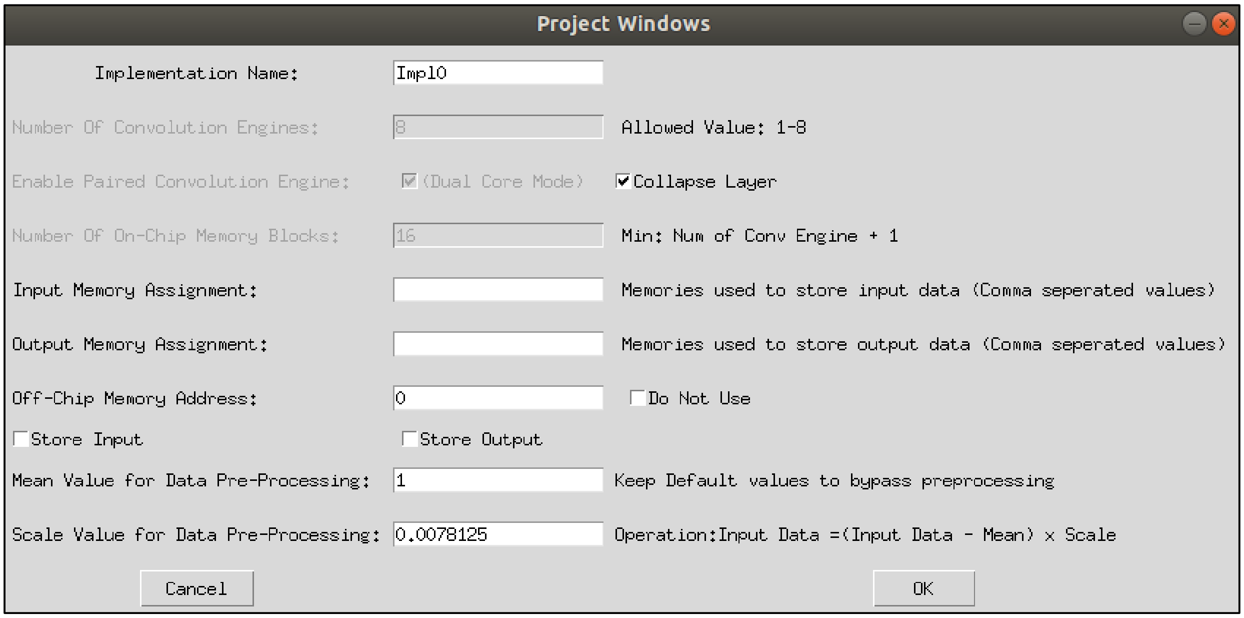

Figure 6.4. SensAI – Project Settings

7. Click **OK** to create project.
8. Double-click **Analyze**.

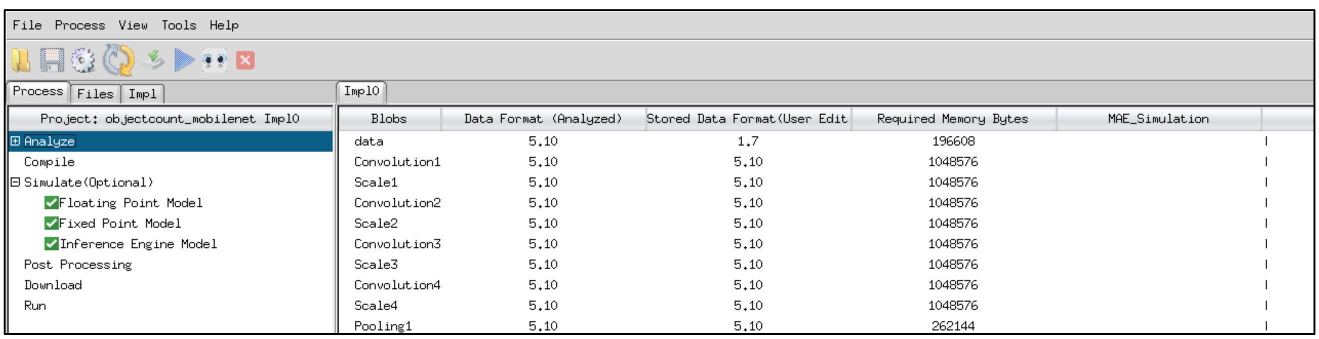

Figure 6.5. SensAI – Analyze Project

9. Double-click **Compile** to generate the Firmware file.

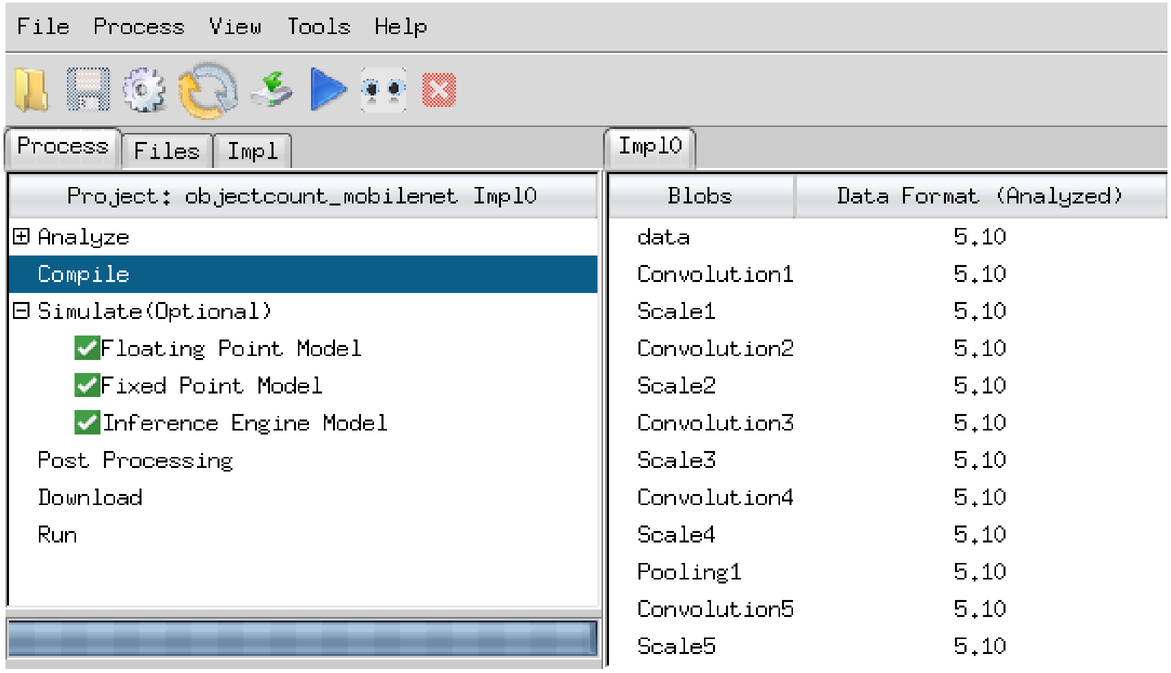

Figure 6.6. Compile Project

7 Hardware Implementation

7.1 Top Level Information

7.1.1 Block Diagram

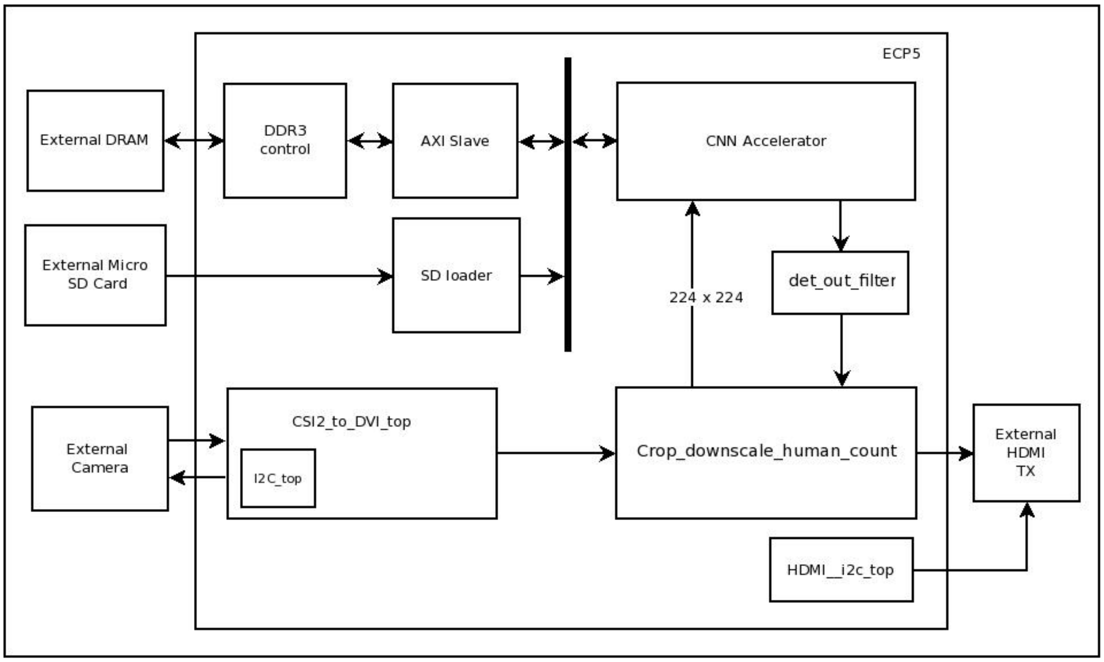

Figure 7.1. RTL Top Level Block Diagram

7.1.2 Operational Flow

This section provides a brief idea about the data flow across ECP5 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.
- Command code is written in DRAM through AXI before the execution of CNN Accelerator IP Core starts. CNN reads command code from DRAM during its execution and does calculation with it per command code. Intermediate data may be transferred from/to DRAM per command code.
- The external camera configured using *I2C_top* logic block captures the raw image and pass it to *CSI2_to_DVI_top* module. *CSI2_to_DVI_top* module separates the R, G, B pixels from raw data and makes separated colors to match the real world using gain and offset controls.
- The RGB data from *CSI2_to_DVI_top* module is downscaled to 224 x 224 image resolution by *crop_downscale_human_count* module to match CNN's input resolution. This data is written into internal memory block of CNN Accelerator IP Core through input data ports.
- After command code and input data are available, CNN Accelerator IP Core starts calculation at the rising edge of start signal.
- Output data of CNN is passed to *det_out_filter* for post processing. *det_out_filter* generates bounding box coordinates X, Y, W, H associated with top 20 confidence value indexes for 224 x 224 image resolution.
- These coordinates are passed to *crop_downscale_human_count* again for resizing them to fit the actual image resolution on HDMI display. HDMI is configured using *hdmi_i2c_top* block.

7.1.3 Core Customization

Table 7.1. Core Parameter

Constant	Default (Decimal)	Description
CONF_THRESH	65472 (that is -64)	It is signed confidence threshold value calculated as per Q-Format of Last Layer of CNN. For example, if threshold is to be kept is (-0.0625) and Q-Format is Q5.10, CONF_THRESH. = 2's complement ((0.0625) * (2^10)) = 2's complement (64) = 65472 Decimal = FFC0 Hex
OVLP_TH_2X	5	Intersection Over Union Threshold
NUM_FRAC	10	Fraction Part Width in Q-Format representation.
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)
NUM_CLASS	1	Number of probability classes
TOP_N_DET	20	Number of Top confidence bounding boxes detection
OBJECT	BODY	Detection of upper human body from input image

7.2 Architecture Details

7.2.1 Pre-processing CNN

Output from *CSI2_to_DVI_top* module is a stream of RGB data that reflects the camera image which is given to crop_downscale_human_count module.

The *crop_downscale_human_count* module processes that image data and generates input of 224x224 image data interface for CNN IP.

7.2.1.1 Pre-processing Flow

- RGB data values for each pixel are fed serially line by line for an image frame.
- These RGB data values are considered as valid only when horizontal and vertical masks are inactive. Mask parameters are set such that it masks out boundary area of full HD resolution (1920 x 1080) to 1792 x 896.

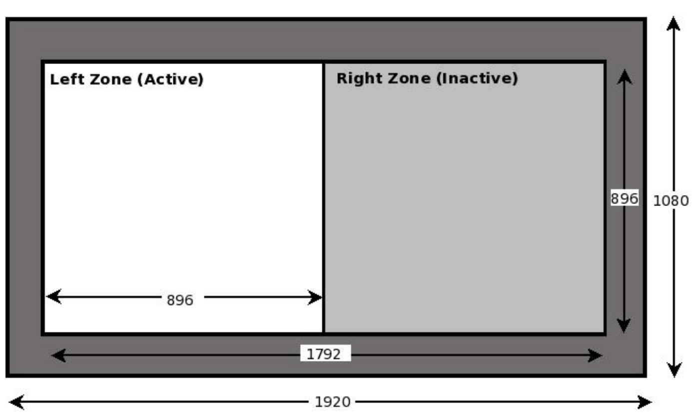

Figure 7.2. Masking and Zoning

- The Frame of 1792 x 896 is further divided in half horizontally by making two blocks of 896 x 896 of the same frame as shown in Figure 7.2. This is done to make the downscaling process easier.
- When the left zone is active pixel values from left zone are used to generate CNN input image data and pixel values from right zone are ignored. After data is sent to CNN, the active zone is changed to right zone. When CNN is ready to accept data, the pixel values from right zone are used to generate CNN input image data.
- Each 896x896 frame block is downscaled into 224 x 224 resolution image as shown in the Figure 7.3.

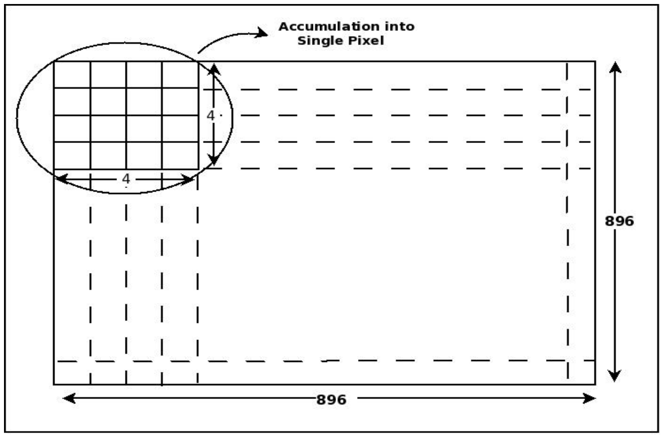

Figure 7.3. Downscaling

- It generates a single accumulated pixel value for each 4 x 4 grid of pixels. This leads to generate 224 x 224 values (896/4 x 896/4) from 896 x 896 values.
- This accumulated value is written into Frame Buffer. Frame Buffer is a True Dual-Port RAM. Accumulated R, G, B pixel values for 4 x 4 grids are stored in the same memory location.
- When Data is read from memory each RGB value is divided by 16 (that is the area of 4 x 4 grid) to take the average of 4 x 4 grid matrix.

Data from Memory is read and formatted for compatibility with the trained network according to CNN input Data layer configuration. According to CNN Input Data layer width configuration, RTL is implemented with half word write with byte mode. It sends 2 downscaled pixel Byte values concatenated in single clock cycle.

7.2.2 Post Processing CNN

CNN provides total of 8232 [1372 x 6 (C, P, X, Y, W, H)] values which are given to det_out_filter module. 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 (14x14), one confidence value (16 Bit) for each anchor box (7) is provided making total values of confidence 14*14*7 = 1372 from CNN Output.
P	This parameter indicates the probability of detected object class. For each grid cell (14x14), one probability value (16 Bit) for each anchor box (7) is provided making total values of probability 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.

Parameter	Description
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 7.4 shows the format of CNN output.

Output Data	C				P				X	Y	W	H	X	Y	W	H
Index No.	0 - 195	196 - 391	....	1176 - 1371	1372 - 1567	1568 - 1763	....	2548 - 2743	2744 - 2939	2940 - 3135	3136 - 3331	3332 - 3527	3528 - 3723	3724 - 3919	3920 - 4115	4116 - 4311
Grid No.	0 - 195	0 - 195	....	0 - 195	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	....	7	1				2			

Figure 7.4. CNN Output Data Format

The primary functionality of det_out_filter module is to capture the CNN valid output and modifying it to make it work with the crop_downscale_human_count module.

The det_out_filter module contains 3 sub-modules: det_sort_conf, det_st_class and det_st_bbox.

- 1372 values of confidence are passed to det_sort_conf module. It sorts out top 20 highest confidence values and stores their indexes. Index values are passed to det_st_class and det_st_bbox modules.
- 1372 values of probability are passed to det_st_class module. It provides the valid class probability bitmap which is passed to det_st_bbox module.
- 1372 x 4 values of coordinates are passed to det_st_bbox module. It calculates the bounding box coordinates, performs NMS and provides valid box bitmap.

The crop_downscale module contains logic for post processing

- draw_box module calculates the box coordinates for 89 x 896 image from 224 x 224 coordinates.
- lsc_osd_text module generates character bitmap for text display on HDMI.
- HDMI display logic implements Muxing logic to provide final serial HDMI output interface.

This module implements logic for providing box coordinates, text and masking information to HDMI interface serially.

7.2.2.1 Confidence Sorting

- All input confidence values (1372) are compared with threshold parameter CONF_THRESH value. Confidence values which are greater than threshold are considered as valid for sorting.
- 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 CNN output.
- Two memory arrays are generated in this module (1) Sorted top 20 (TOP_N_DET) Confidence Value array (2) Sorted top 20 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 then, the input value is updated on current array location. Previously stored value of current location is shifted into the next array location.
- Refer to Figure 7.5 for sorting of new value of confidence into existing Confidence Value array. Calculated confidence index (anchor count value) is also updated in confidence index array along with confidence value array.

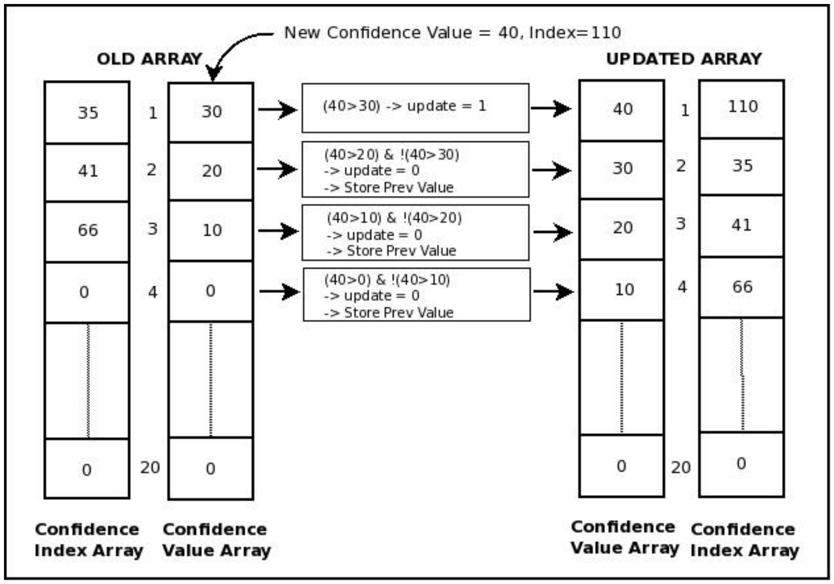

Figure 7.5. Confidence Sorting

- This process is followed for all 1372 Confidence values. This module provides 20 indexes (o_idx_00 to o_idx_19) as output along with the count of valid indexes (o_num_conf). o_idx_00 contains highest confidence value index and o_idx_29 contains lowest confidence value index.

7.2.2.2 Class Probability Detection

- det_st_class module captures total NUM_CLASS * 1372 Probability Class values from CNN output. Currently NUM_CLASS is set to 1 for a single class of Human Upper Body detection.
- This module checks the class probability value for the sorted index numbers obtained from det_sort_conf module.
- If multiple Probability Class exist (NUM_CLASS>1), this module compares values of multiple probability classes for each sorted confidence index value. It marks the maximum valued probability class as valid (1) and other classes as invalid (0) for each sorted confidence index and store this information in a bitmap memory array.
- This array is provided as output to det_out_filter module for differentiating bounding boxes of different probability class by different color. Green box is used for probability class 1. Similarly, red and blue boxes can be used for probability class 2 and 3 respectively.
- For Single Probability Class, this module provides hardcoded value of 1 set as probability for each sorted confidence index value in bitmap array. This only infers green boxes in final output.

7.2.2.3 Bounding Box Calculation

SqueezeDet Neural Network for Object Detection is trained with 7 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 x H (pixel)	184x184	138x138	92x92	69x69	46x46	34x34	23x23

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

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 total 1372 (14 x 14 x 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 20 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 predicted X', Y', W' and H' values are clamped so that the box remains out of masking area. This is shown in logic 2.

```
LOGIC 2
If (X' < 0) => X'' = 0   | Else if (X' > 223) => X'' = 223 | Else X'' = X'
If (Y' < 0) => Y'' = 0   | Else if (Y' > 223) => Y'' = 223 | Else Y'' = Y'
If (W' < 0) => W'' = 0   | Else if (W' > 223) => W'' = 223 | Else W'' = W'
```

The final calculated X'', Y'', W'' and H'' values for all the boxes are stored in separate memory array each having highest confidence coordinate at 1st index and lowest confidence coordinate 20th index.

The Box coordinates are passed to crop_downscale_human_count module after NMS process.

7.2.2.4 NMS – Non Max Suppression

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 CNN output data is completely received.

- It starts from box having highest Confidence coordinates: 0th location in X, Y, W, H array.
  These coordinates are compared against 2nd highest Confidence coordinates: 1st 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 highest confidence box and 2nd highest confidence box as shown is Figure 7.6.

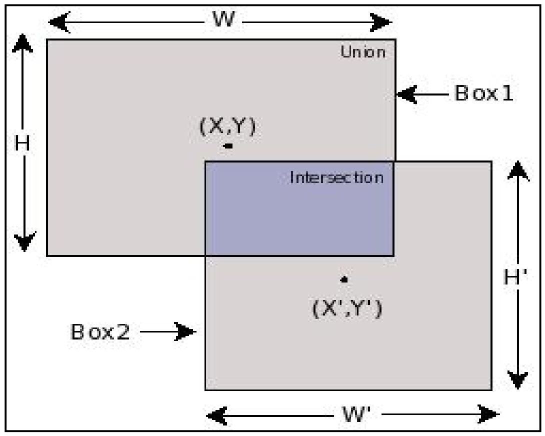

Figure 7.6. Intersection-Union Area NMS

- If *Intersection Area * (OVLP_TH_2X/2) > Union Area*, the box with 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.

7.2.2.5 Bounding Box Upscaling

- draw_box module converts X, Y, W, H input coordinates provided for 224 x 224 resolution into 896 x 896 resolution as shown in logic 3.

```
LOGIC 3
X1 = (X’’ - W’’/2) * 4 + Horizontal-Mask (64/960)
Y1 = (Y’’ – H’’/2) * 4 + Vertical-Mask (92)
X2 = (X’’ + W’’/2) * 4 + Horizontal-Mask (64/960)
Y2 = (Y’’ + H’’/2) * 4 + Vertical-Mask (92)
```

- (X, Y) are considered as center of the Box of Width W and Height H for calculating extreme ends of the Box (X1, X2, and Y1, Y2). For converting from 224 to 896, the coordinates are multiplied with 4. 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 keeps track of pixels of each line and Lines of each frame. 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. Box is only displayed if Box-Bitmap for that box is set to 1(From det_st_bbox 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]
```

- 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 via HDMI.

7.2.2.6 OSD Text Display

- lsc_osd_text module provides bitmap of each ASCII character to be displayed with specified position on screen. It takes count of detected Humans and Threshold value as input.
- It sets an output signal (text_on) when text is to be displayed on HDMI. When text_on is set, RGB value for that pixel location is assigned FFF value (While color) and sent to HDMI output instead of original pixel value.

7.2.2.7 HDMI Display Management

RGB data is passed serially to HDMI and it is multiplexed by following values.

- If Signal Text is on (text_on) – Pass all RGB value as FFF for While color display.
- If Signal Green is on (green_on) – Pass only Green pixel value as FFF. Keep Red and Blue values as 0.
- If Signal Mask is on (fmask_on) – Pass darker RGB pixel values.
- Else – Pass Input RGB Data as it is.

8 Creating FPGA Bitstream File

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

To create the FPGA bitstream file:

1. Open Lattice Diamond Software.

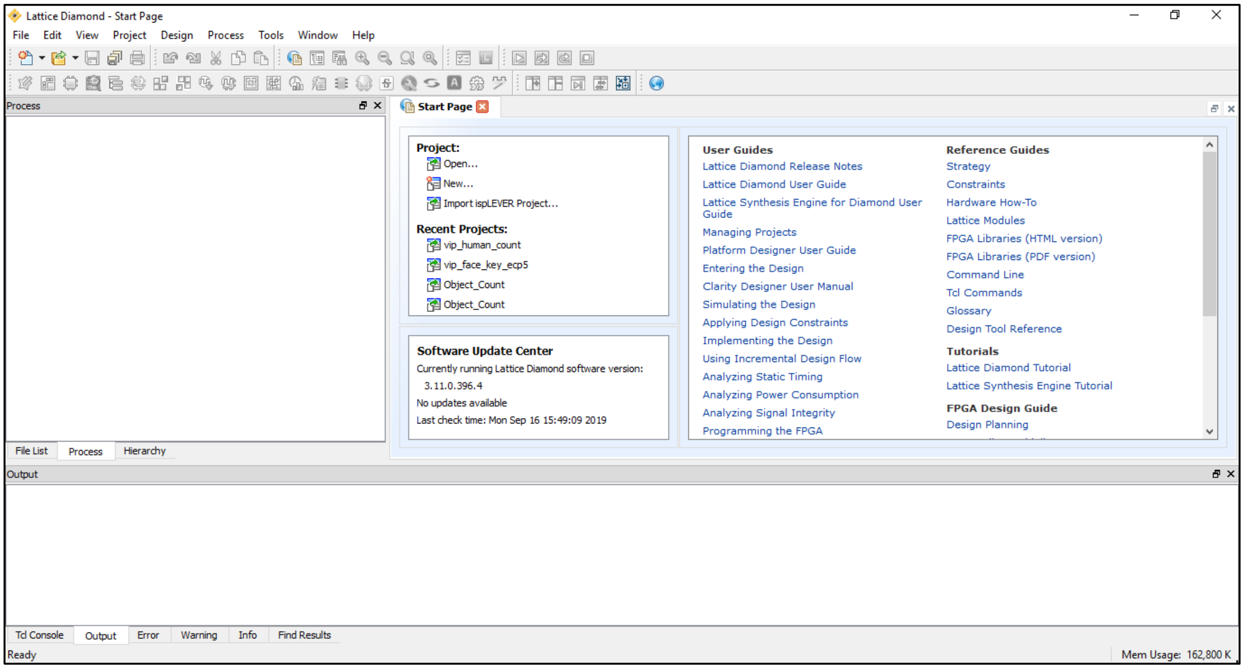

Figure 8.1. Lattice Diamond – Default Screen

2. Go to **File > Open > Project**.
3. Open Diamond project file for ECP5 Face Identification Demo RTL.

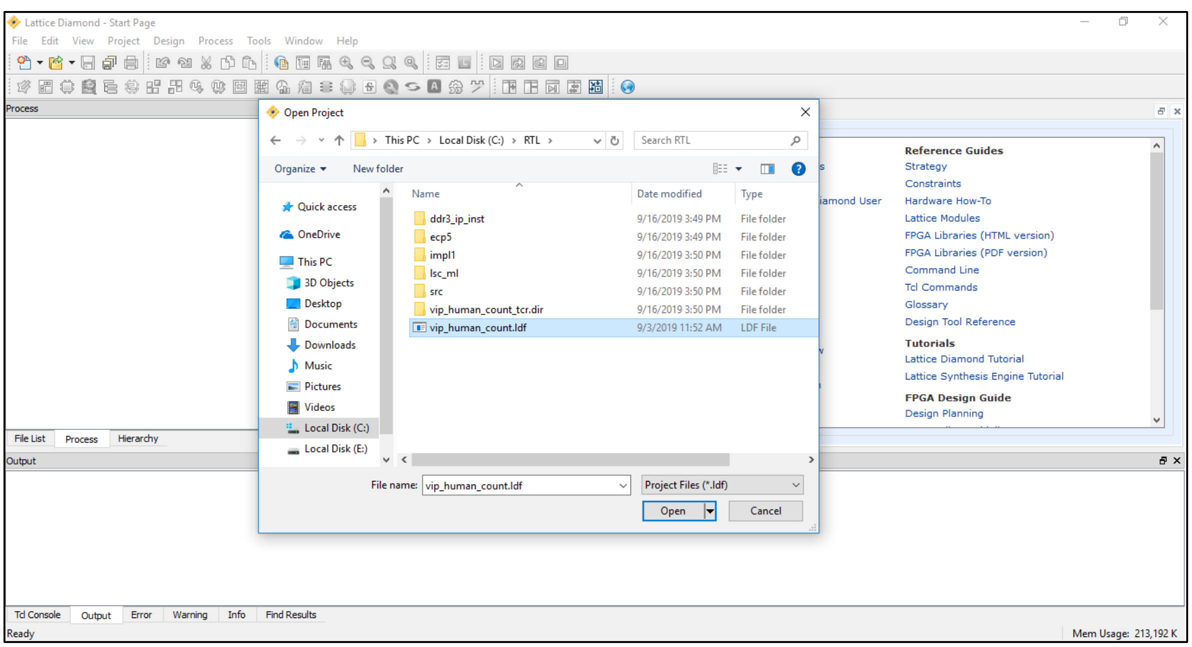

Figure 8.2. Lattice Diamond – Open ECP5 Face Identification Diamond Project File

4. Double-click **Bitstream File** to trigger bitstream generation.

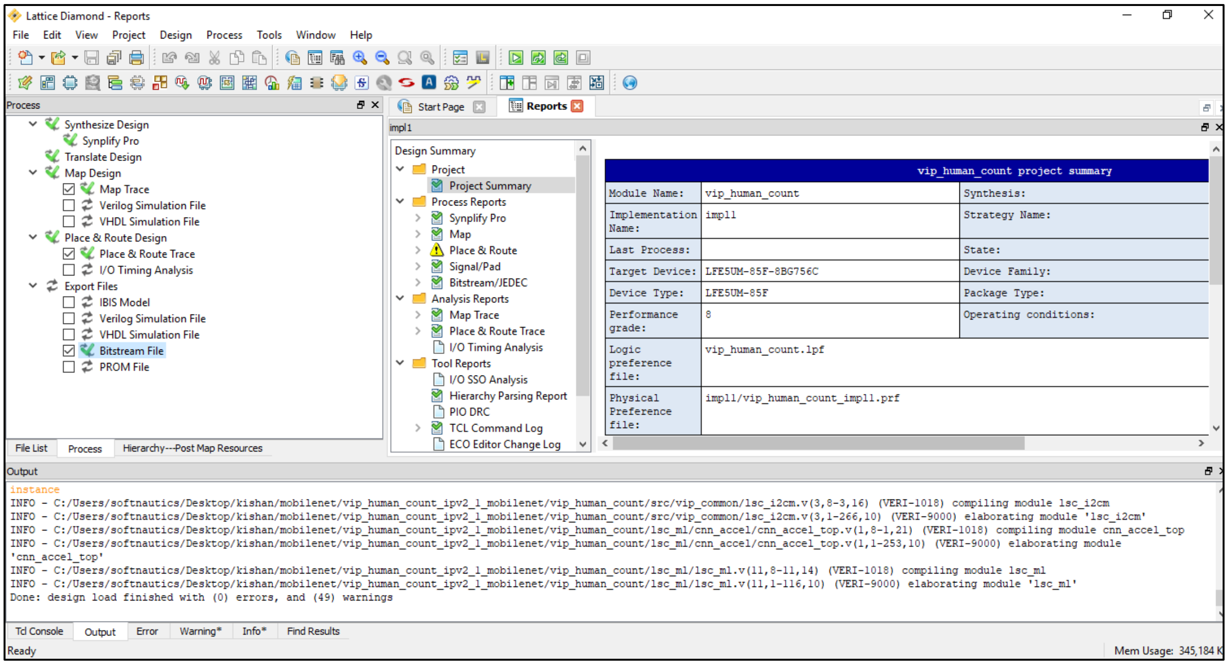

Figure 8.3. Lattice Diamond – Trigger Bitstream Generation

5. The Lattice Diamond tool displays *Saving bit stream in …* message in Reports window as shown in Figure 8.4. Bitstream is generated at *Implementation Location* in Figure 8.3.

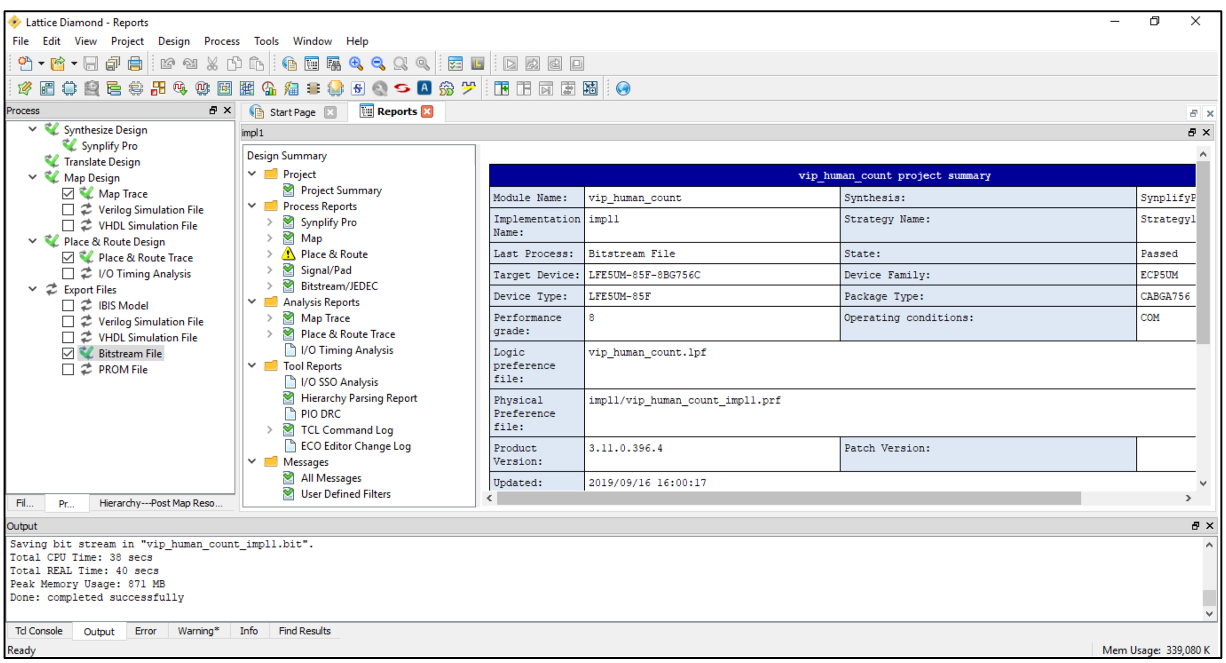

Figure 8.4. Lattice Diamond – Bit File Generation Report Window

9 Programming the Demo

9.1 Programming the CrossLink SPI Flash

9.1.1 Erasing the CrossLink SRAM Prior to Reprogramming

If the CrossLink is already programmed (either directly or loaded from SPI Flash), erase the CrossLink SRAM before reprogramming the CrossLink SPI Flash. Keep the board powered on to prevent reloading on reboot.

To erase the CrossLink device SRAM:

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

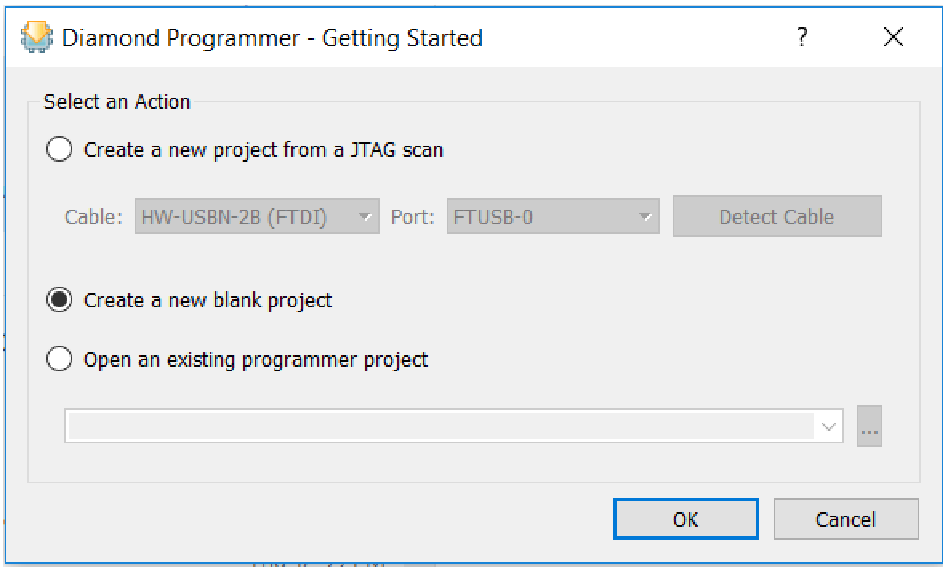

Figure 9.1. Diamond Programmer – Default Screen

2. Click **OK**.
3. In the Diamond Programmer main interface, select **LIFMD** in Device Family and **LIF-MD6000** in Device as shown in Figure 9.2.

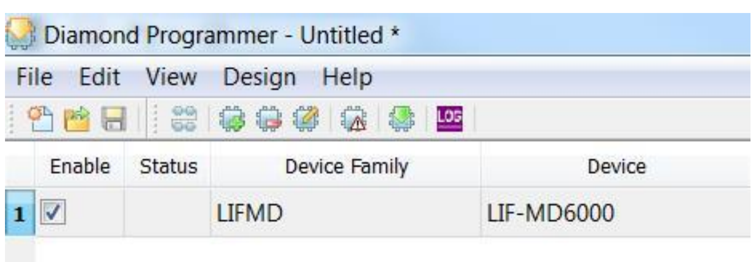

Figure 9.2. Diamond Programmer – Device Selection

4. Click the CrossLink row and select **Edit > Device Properties**.
5. In the Device Properties dialog box, select **SSPI SRAM Programming** in Access mode and **Erase Only** in Operation as shown in Figure 9.3.

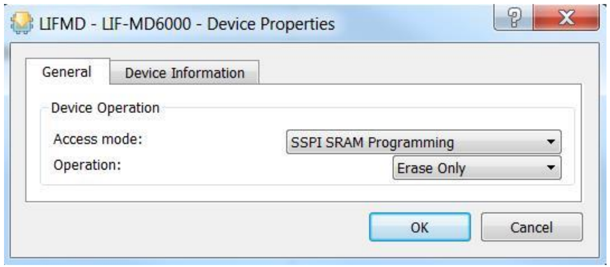

Figure 9.3. Diamond Programmer – Device Operation

6. Click **OK** to close the Device Properties dialog box.
7. In the Diamond Programmer main interface, click the **Program** button to start the erase operation.

Note: If you power OFF/ON the board, the SPI Flash reprograms the CrossLink device. In this case, you must repeat steps 1 to 7.

9.1.2 Programming the CrossLink VIP Input Bridge Board

To program the CrossLink VIP Input Bridge Board:

1. Ensure that the CrossLink device SRAM is erased by performing the steps in **Erasing the CrossLink SRAM Prior to Reprogramming**.
2. In the Diamond Programmer main interface, click the CrossLink row and select **Edit > Device Properties** to open the Device Properties dialog boxes shown in Figure 9.4.

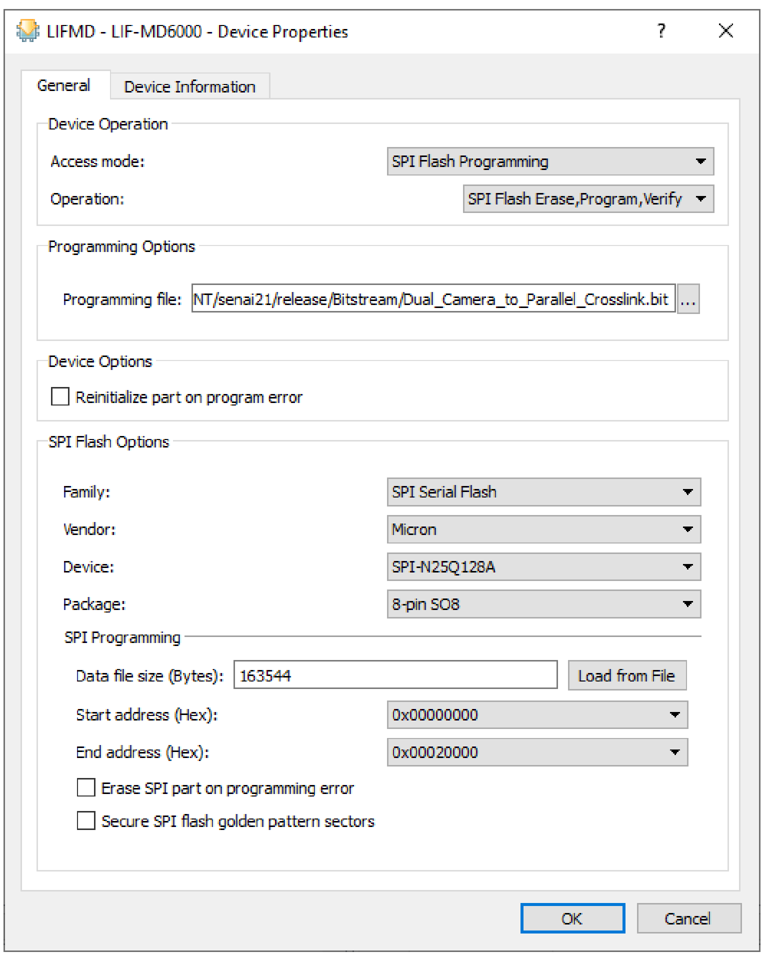

Figure 9.4. Diamond Programmer – Selecting Device Properties Options for Crosslink Flashing

3. Apply the settings below.
    - Under Device Operation, select the options below:
        - **Access Mode – SPI Flash Programming**
        - **Operation – SPI Flash Erase, Program, Verify**
    - Under Programming Options, select the bitstream file *~/Demonstration/Dual_Camera_to_parallel_Crosslink.bit* available in downloaded demo directory in Programming file.
    - For **SPI Flash Options**, refer to Table 9.1:

Table 9.1. Diamond Programmer – SPI Flash Options

Item	Rev B	Rev C - Option 1	Rev C – Option 2
Family	SPI Serial Flash	SPI Serial Flash (SPI Serial Flash Beta for Diamond 3.10 SP1 or earlier)	SPI Serial Flash (SPI Serial Flash Beta for Diamond 3.10 SP1 or earlier)
Vendor	Micron	Micron	Macronix
Device	SPI-M25PX16	SPI-N25Q128A	MX25L12835F
Package	8-pin S08W	8-pin SOP2	8-Land WSON

    - Click **Load from File** to update the **Data file size (Bytes)** value.
    - Ensure that the following addresses are correct:
        - **Start Address (Hex) – 0x00000000**
        - **End Address (Hex) – 0x00020000**
4. Click **OK**.
5. In the Diamond Programmer main interface, click the **Program** button to start the programming operation.
6. After successful programming, the **Output** console displays the result as shown in Figure 9.5.

Figure 9.5. Diamond Programmer – Output Console

9.2 ProgrammingECP5 VIP Processor Board

Both the CrossLink VIP Input Bridge Board and the ECP5 VIP Processor Board must be configured and programmed. Also, the demo design firmware must be programmed onto the MicroSD Card which is plugged into the MicroSD Card Adaptor Board.

9.2.1 Erasing the ECP5 Prior to Reprogramming

If the ECP5 device is already programmed (either directly or loaded from SPI Flash), erase the ECP5 SRAM before reprogramming the ECP5 SPI Flash. Keep the board powered on to prevent reloading on reboot.

To erase the ECP5 SRAM:

1. Launch Diamond Programmer with **Create a new blank project**.

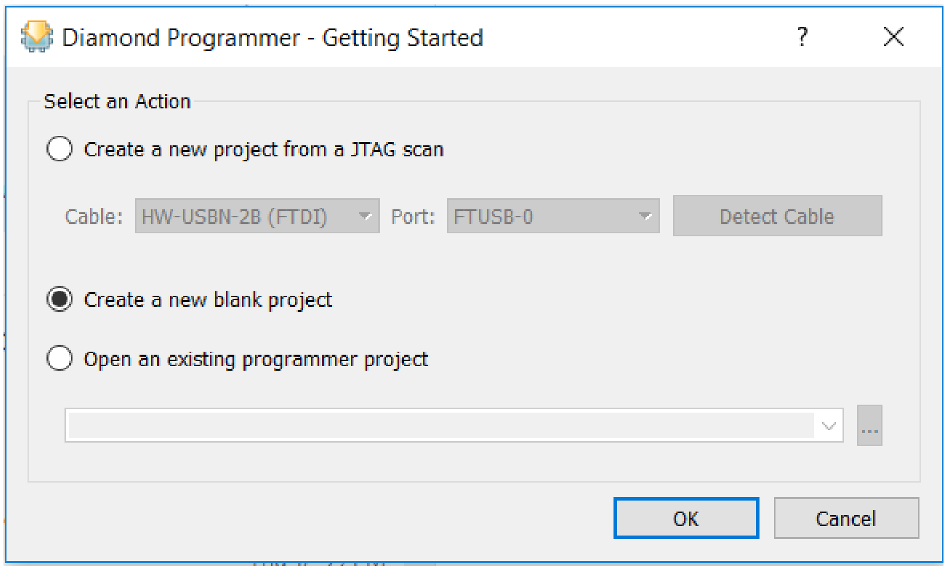

Figure 9.6. Diamond Programmer – Default Screen

2. Click **OK**.
3. In the Diamond Programmer main interface, select **ECP5UM** in Device Family and **LFE5UM-85F** in Device as shown in Figure 9.8.

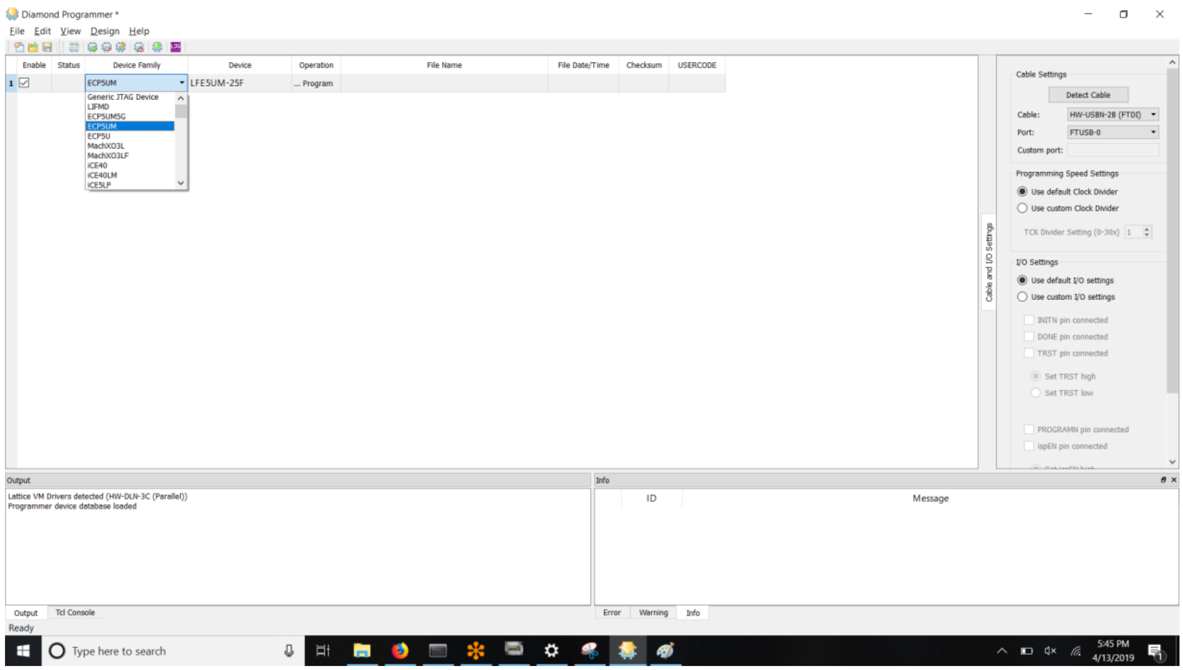

Figure 9.7. Diamond Programmer – Device Family Selection

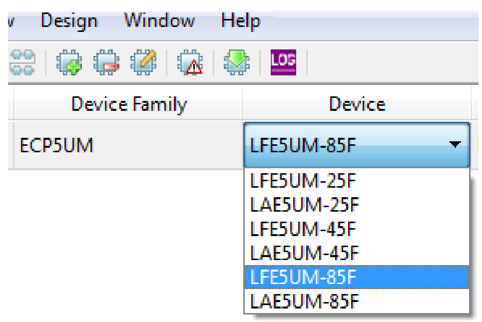

Figure 9.8. Diamond Programmer – Device Selection

4. Click the ECP5 row and select **Edit > Device Properties**.
5. In the Device Properties dialog box, select **JTAG 1532 Mode** in Access mode and Erase Only in Operation (shown in Figure 9.9).

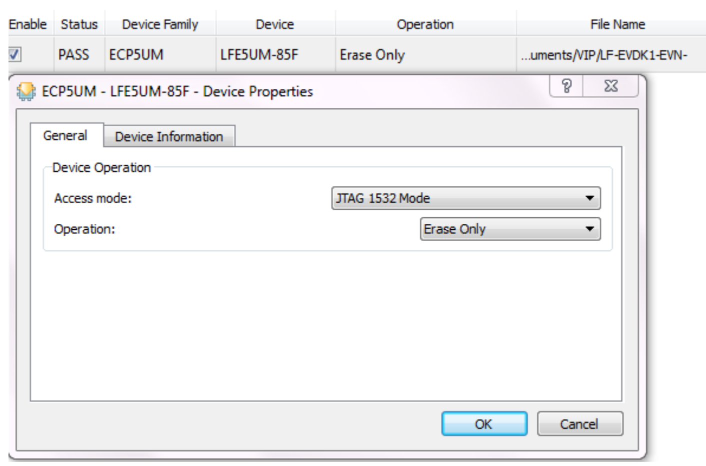

Figure 9.9. Diamond Programmer – Device Operation

6. Click **OK** to close the Device Properties dialog box.
7. In the Diamond Programmer main interface, click the Program button to start the Erase operation.

   Note: If you power OFF/ON the board, the SPI Flash reprograms the ECP5 device. In this case, you must repeat steps 1 to 7.

9.2.2 Programming the ECP5 VIP Processor Board

To program the ECP5 VIP Processor Board:

1. Ensure that the ECP5 device is erased by performing the steps in Erasing the ECP5 Prior to Reprogramming.
2. In the Diamond Programmer main interface, click the ECP5 row and select **Edit > Device Properties**
3. The **Device Properties** dialog box opens. Select human count demo bit file in *Programming file:* section as shown in Figure 9.10 (Rev B).

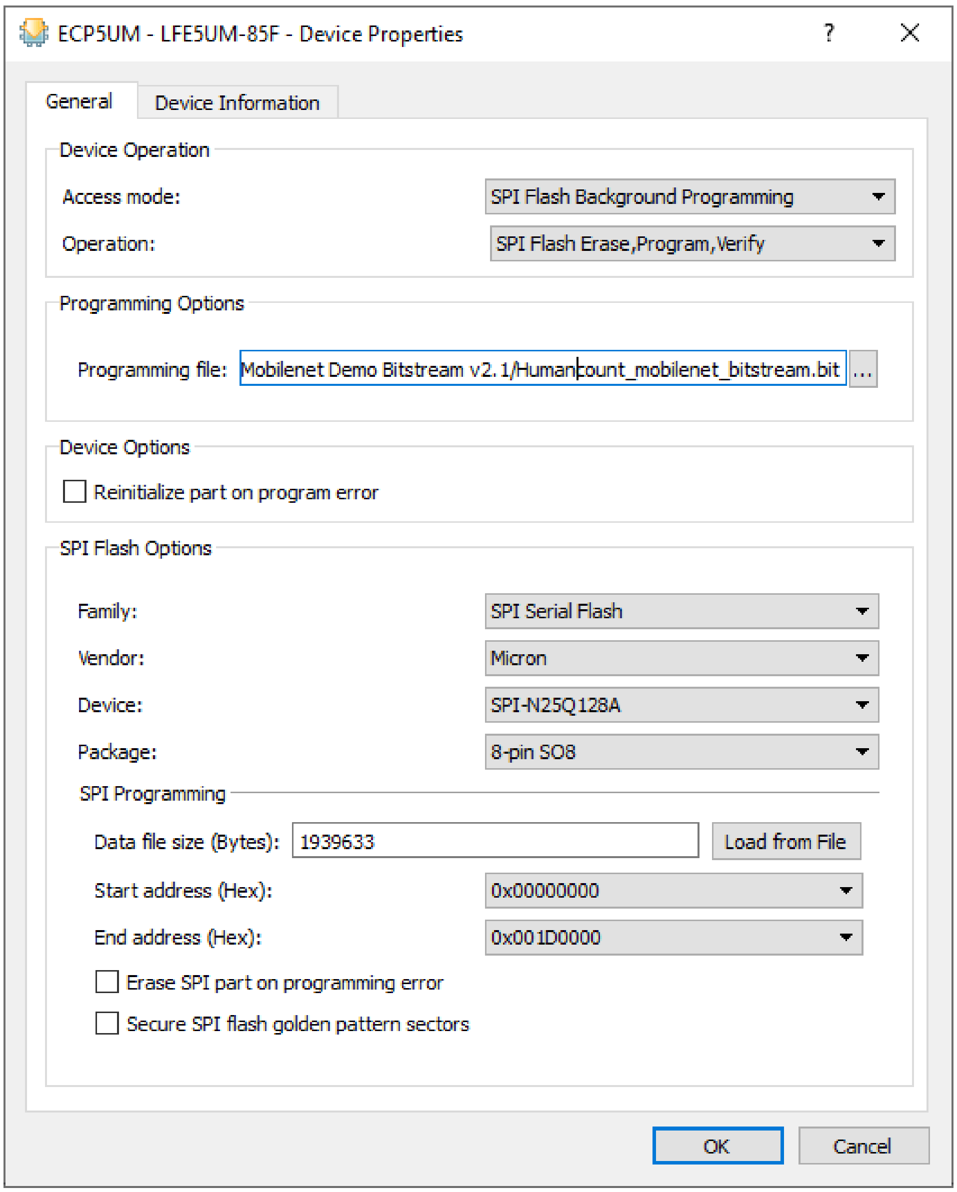

Figure 9.10. Diamond Programmer – Selecting Device Properties Options for ECP5 Flashing

4. Apply the settings below:
   - Under Device Operation, select the options below:
     - **Access Mode – SPI Flash Background Programming**
     - **Operation – Erase, Program, Verify**
   - Under Programming Options, select the appropriate bitstream file for respective demo in Programming file.
   - For **SPI Flash Options**, refer below table:

Table 9.2. Diamond Programmer – SPI Flash Options

Item	Rev B	Rev C - Option 1
Family	SPI Serial Flash	SPI Serial Flash
Vendor	Micron	Macronix
Device	SPI-N25Q128A	MX25L12835F
Package	8-pin SO8	8-Land WSON

   - Click **Load from File** to update the Data file size (Bytes) value.
   - Ensure that the following addresses are correct:
     - **Start Address (Hex) – 0x00000000**
     - **End Address (Hex) – 0x001D0000**
5. Click **OK**.
6. In the Diamond Programmer main interface, click the **Program** button to start the programming operation.
7. After successful programming, the **Output** console displays the result as shown in Figure 9.11.

Figure 9.11. Diamond Programmer – Output Console

9.3 Programming the MicroSD Card Firmware

To write the image to the MicroSD Card:

1. Download and install the Win32diskimager Image Writer software from the following link: https://sourceforge.net/projects/win32diskimager/.
2. Use Win32diskimager to write the appropriate Flash image (binary firmware) file to the SD memory card. You may need SD Card reader and adapter to connect the MicroSD card to PC for firmware flashing.
3. In Win32 Disk Imager, select the image file for respective demo firmware bin file as shown in Figure 9.12**.**
4. Select the Card Reader in **Device** as shown in Figure 9.12**.**
5. Click **Write.**

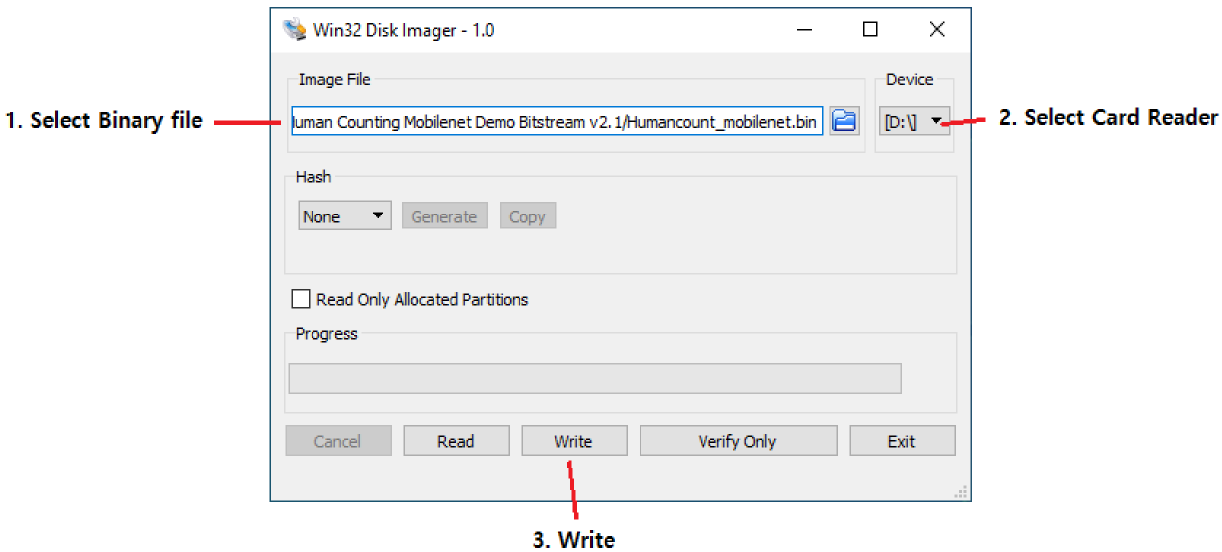

Figure 9.12. Win32 Disk Imager

Optionally, you can click **Verify Only** to confirm whether firmware write is correct.

10 Running the Demo

To run the demo:

1. Insert the configured MicroSD Card into the MicroSD Card Adapter, and connect it to the Embedded Vision Development Kit.

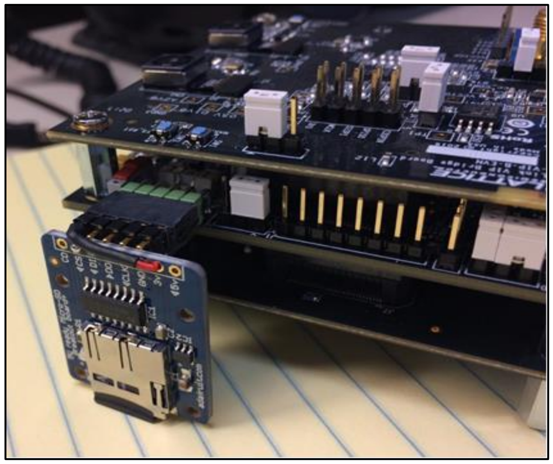

Figure 10.1. Connecting the MicroSD Card

2. Cycle the power on the Embedded Vision Development Kit to allow ECP5 and CrossLink to be reconfigured from Flash.
3. Connect the Embedded Vision Development Kit to the HDMI monitor. The camera image should be displayed on monitors shown in below figure.

Figure 10.2. Running the Demo

4. Demo output contains bounding boxes for detected humans in a given frame and it displays the total number of detected humans in a given frame on HDMI output.

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. Just clone it and launch. Simple and Powerful.
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, October 2019

Section	Change Summary
All	Initial release

www.latticesemi.com