Machine Vision: Barcode Detection

Reference Design

FPGA-RD-02266-1.0

March 2023

Disclaimers

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

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CKPT	Checkpoint
CNN	Convolutional Neural Network
CSP	Cross Stage Partial network
EVDK	Embedded Vision Development Kit
FPGA	Field Programmable Gate Array
LED	Light-Emitting Diode
MLE	Machine Learning Engine
NN	Neural Network
NNC	Neural Network Compiler
SD	Secure Digital
SDHC	Secure Digital High Capacity
SDXC	Secure Digital extended Capacity
SPI	Serial Peripheral Interface
VIP	Video Interface Platform
USB	Universal Serial Bus
VVML	Voice and Vision Machine Learning Board
CPNX	CertusPro-NX
VnV	Voice and Vision

1. Introduction

The Barcode Detection design process uses CertusPro-NX Voice and Vision Machine Learning Board. As the application, this document demonstrates the barcode detection reference design.

1.1. Design Process Overview

The design process involves the following steps:

1. Training the model.
   - Setting up the basic environment.
   - Preparing the dataset.
   - Training the machine.
     - Training the machine and creating the checkpoint data.
   - Creating the frozen file (*.pb).
2. Compiling Neural Network: creating the filter and firmware binary files with Lattice sensAI 6.0 program.
3. FPGA design: creating the FPGA Bit stream file.
4. FPGA bitstream and Quantized Weights and Instructions: flashing the binary and bitstream files to CertusPro-NX VVML hardware.

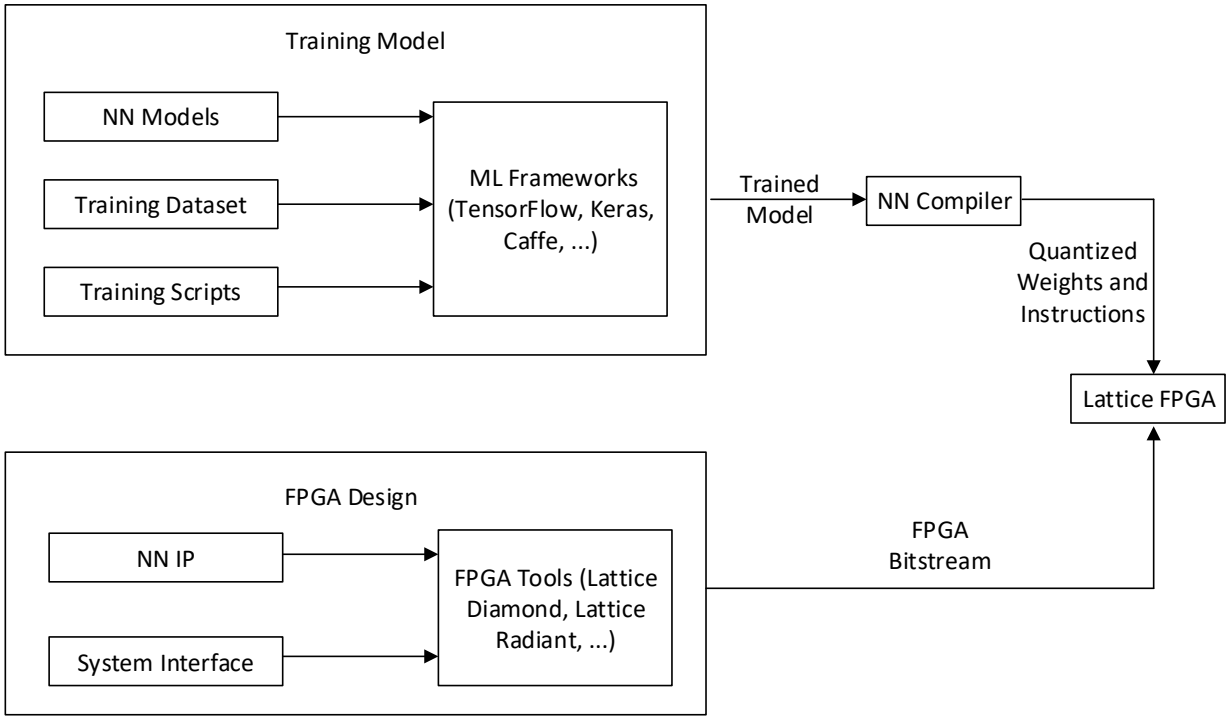

Figure 1.1. Lattice Machine Learning Design Flow

2. Setting Up the Basic Environment

2.1. Tools and Hardware Requirements

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

2.1.1. Lattice Tools

- Lattice Radiant Tool v2022.1: refer to http://www.latticesemi.com/latticeradiant
- Lattice Radiant Programmer v2022.1: refer to http://www.latticesemi.com/latticeradiant
- Lattice Neural Network Compiler v 6.0: refer to https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler

2.1.2. Hardware

This design uses the CertusPro-NX Voice and Vision Machine Learning (VVML) Board, Rev A Board, as shown in Figure 2.1.

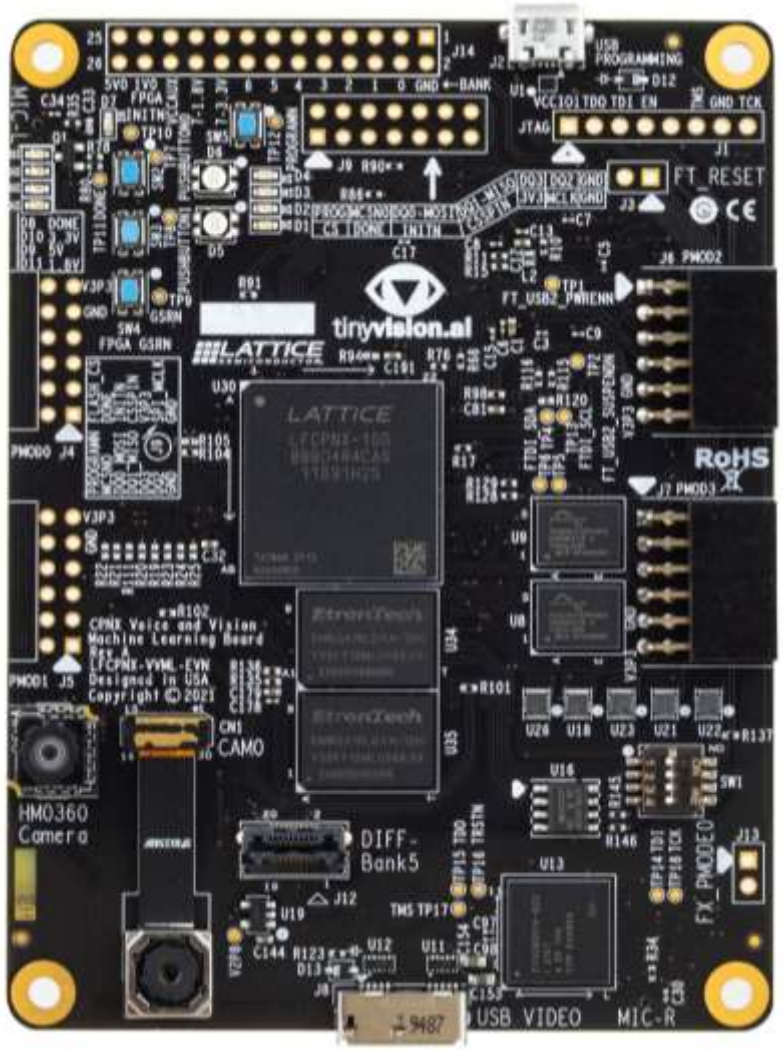

Figure 2.1. Lattice CertusPro-NX Voice and Vision Machine Learning (VVML) Board, Rev A

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 18.04 Operation System. 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 CUDA toolkit-10.1, go to https://developer.nvidia.com/cuda-toolkit-archive and click on specific required version (CUDA 10.1) from the list. See Figure 2.2.

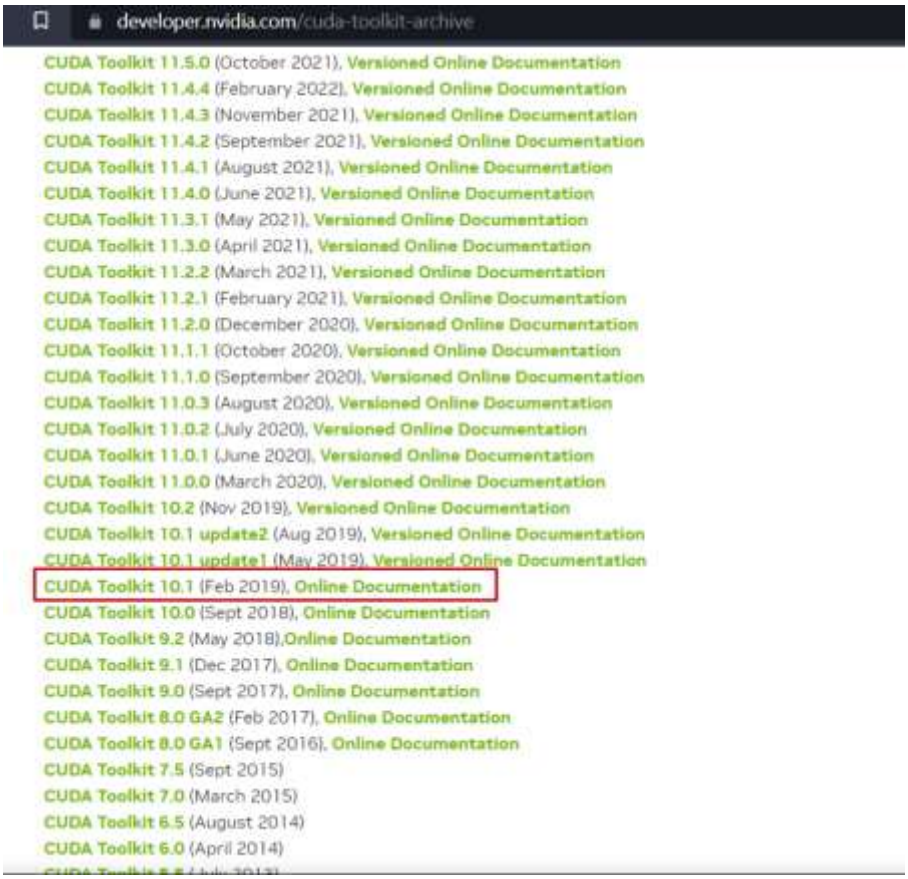

Figure 2.2. CUDA Archive Page

Then select the appropriate combination of as highlighted in Figure 2.3 to get the cuda toolkit download option.

Figure 2.3. Download CUDA Configuration

Once downloaded, follow the Installation Instructions given in the download block of the webpage.

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 the following commands to install cuDNN:

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

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

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

Figure 2.4. 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 18.04. Anaconda provides one of the easiest ways to perform machine learning development and training on Linux.

Installing the Anaconda Python

To install the Anaconda and Python 3:

1. Go to https://www.anaconda.com/products/distribution
2. Scroll down for Anaconda Additional Installers.
3. Download Python3 version of Anaconda for Linux.
4. Run the command below to install the Anaconda environment:

```
$  sh  Anaconda3-2022.10-Linux-x86_64.sh
```

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

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

Welcome to Anaconda3 2020.07

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

Figure 2.5. Anaconda Installation

5. Accept the license.

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

Figure 2.6. Accept License Terms

6. Confirm the installation path. Follow the instruction on screen if you want to change the default path.

```
[no] >>> yes


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

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

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

Figure 2.7. Confirm/Edit Installation Location

7. After installation, enter No, as shown in Figure 2.8.

```
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 2.8. Launch/Initialize Anaconda Environment on Installation Completion

2.2.3. Creating New Environment with the yml File Provided

The project structure is provided with the corresponding yml file *env_barcode.yml* which directly creates the required virtual environment.

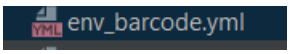

Figure 2.9. env_barcode.yml (1)

Prior to executing this file, you need to edit certain information in yml file. Follow the steps below:

1. After successful installation of Ananconda, navigate to its directory */home/user/anaconda3/* to find env folder. Copy path to this env folder.
2. Open *env_barcode.yml* and make changes below:

   Set your environment name in first line of the file as shown below.

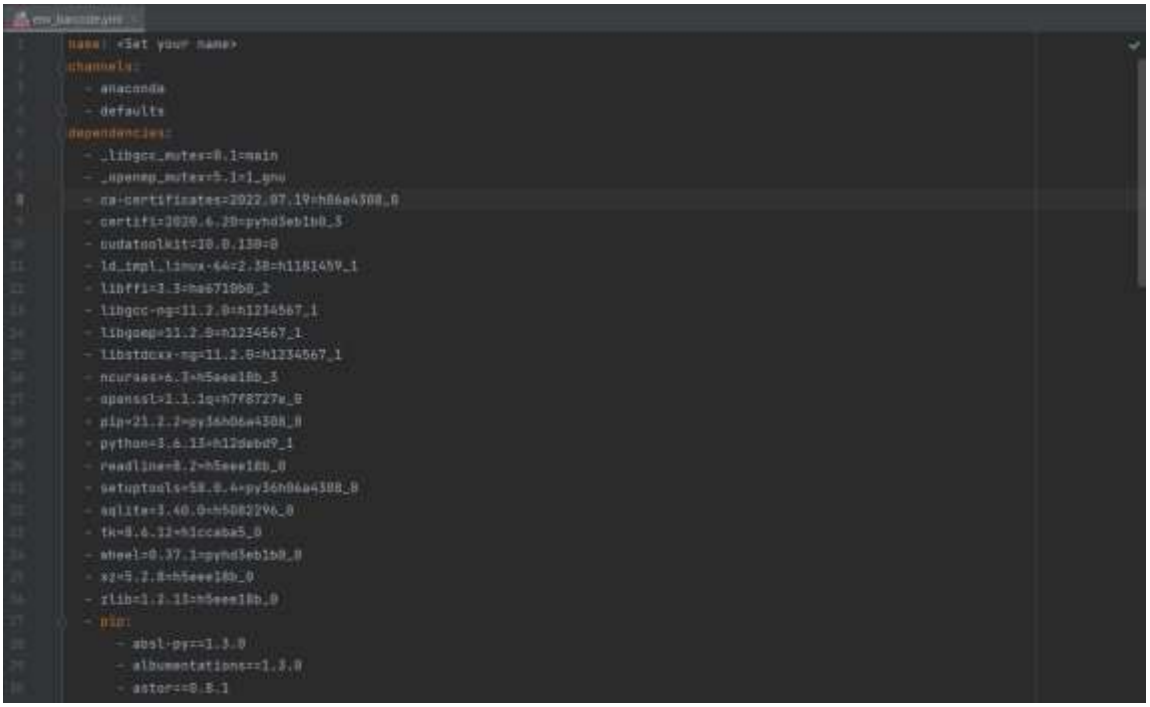

Figure 2.10. env_barcode.yml (2)

3. Modify last line in yml file with new path and env name.

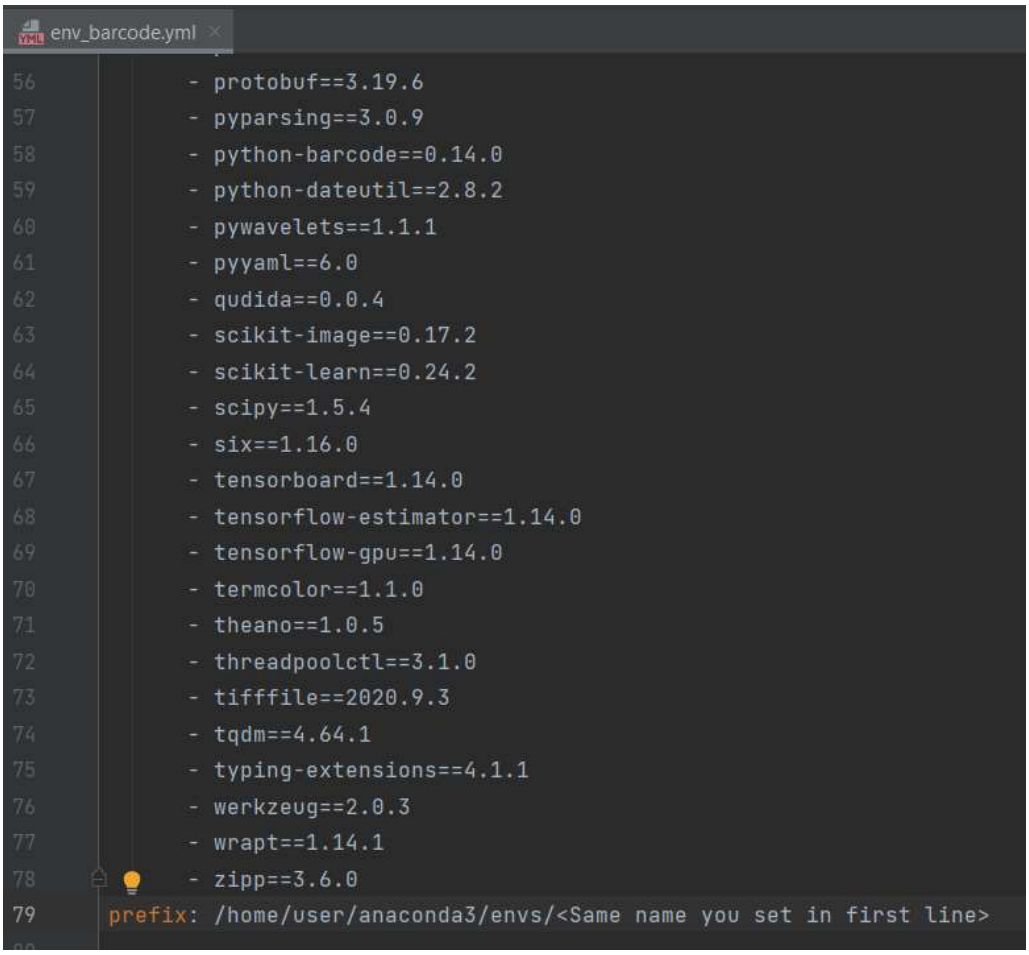

Figure 2.11.env_barcode.yml Prefix Line Edit

4. Activate your conda environment using the command below:

```
$ source anaconda3/bin/activate
```

Figure 2.12. Anaconda Activate

5. Create the environment from the env_barcode.yml file using the command below:

```
$ conda env create -f env_barcode.yml
```

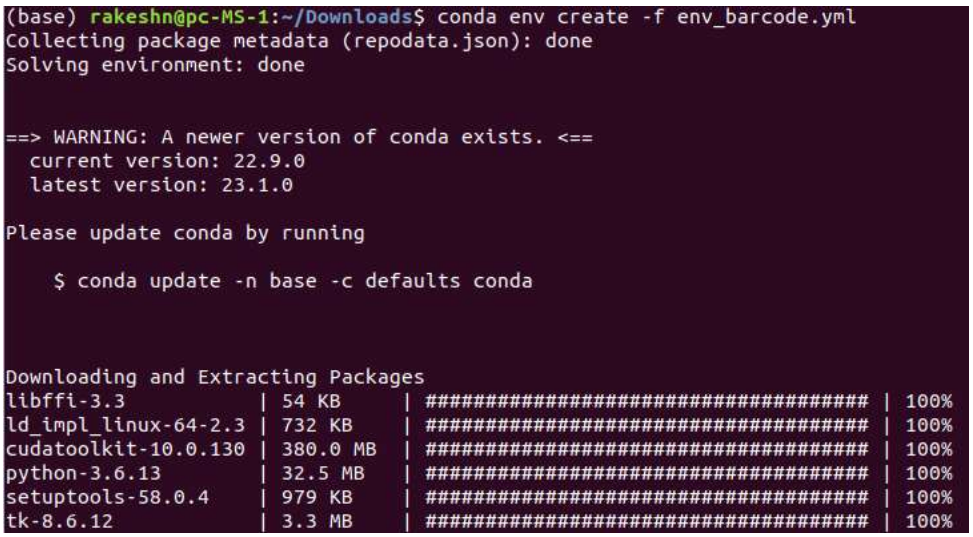

Figure 2.13.env_barcode.yml Execution

Figure 2.14. env_barcode.yml Execution Completion

6. Hit the following command to activate your newly created environment:

```
$ conda activate <your env name>
```

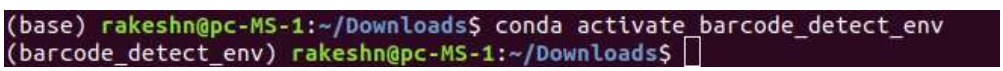

Figure 2.15. Activating the Environment

3. Code Structure

Download the Lattice Bar Code Detection demo training code. Its directory structure should look like the following Figure 3.1.

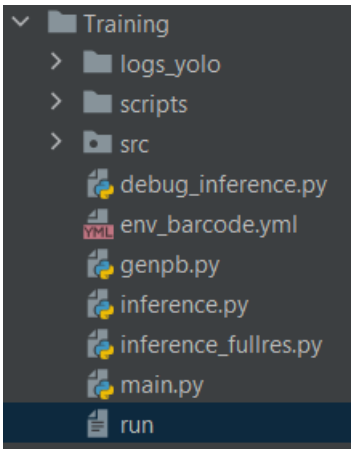

Figure 3.1. Training Code Directory Structure

4. Dataset Preparation

This section describes steps and guidelines used to prepare dataset to train the barcode detection demo for CPNX-VNV.

Note:

This section is for the example reference. With following sections, Lattice Semiconductor just provides the guidelines and/or example which can be used as reference for preparing dataset for given use cases, but in no case, Lattice Semiconductor recommends and/or endorses any dataset(s). Lattice Semiconductor strongly recommends you to gather and prepare your own datasets for your end applications.

4.1. Downloading the Dataset

In this document, we use Artelabs 1-D barcode Dataset (http://artelab.dista.uninsubria.it/downloads/datasets/barcode/medium_barcode_1d/medium_barcode_1d.html) as reference. This dataset is under Creative Commons Attribution 3.0. License. Download Dataset and extract the data.

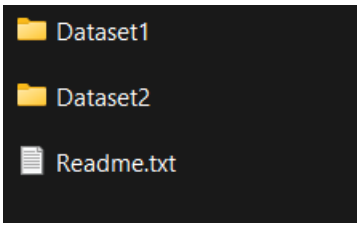

Figure 4.1. Arte-lab Dataset Directory Structure

Figure 4.2. Arte-lab Sample Image

4.2. Labelling Artelab Dataset Using LabelImg Tool

Activate your environment and install LabelImg tool using command below.

```
$ python pip install labelImg
```

```
(base) C:\Users\Rakesh Nakod>activate tf1_14_py_36

(tf1_14_py_36) C:\Users\Rakesh Nakod>pip install labelImg
Collecting labelImg
  Using cached labelImg-1.8.6.tar.gz (247 kB)
Collecting pyqt5
  Downloading PyQt5-5.15.6-cp36-abi3-win_amd64.whl (6.7 MB)
     |████████████████████████████████| 6.7 MB 2.2 MB/s
Collecting lxml
  Downloading lxml-4.9.2-cp36-cp36m-win_amd64.whl (3.8 MB)
     |████████████████████████████████| 3.8 MB 2.2 MB/s
Collecting PyQt5-Qt5>=5.15.2
  Using cached PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl (50.1 MB)
Collecting PyQt5-sip<13,>=12.8
  Downloading PyQt5_sip-12.9.1-cp36-cp36m-win_amd64.whl (83 kB)
     |████████████████████████████████| 83 kB 713 kB/s
Building wheels for collected packages: labelImg
  Building wheel for labelImg (setup.py) ... done
  Created wheel for labelImg: filename=labelImg-1.8.6-py2.py3-none-any.whl size=261544 sha256=90ee214b84cdcbd2e4f7f1708f
1f1e20d2b4dc99803c098bbe011bfde93c9909
  Stored in directory: c:\users\rakesh nakod\appdata\local\pip\cache\wheels\35\6f\d6\f96de77faaf1eab00bfe865ebe8806d3f79
0d609e4d7a15f50
Successfully built labelImg
Installing collected packages: PyQt5-sip, PyQt5-Qt5, pyqt5, lxml, labelImg
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.9.1 labelImg-1.8.6 lxml-4.9.2 pyqt5-5.15.6
```

Figure 4.3. Label Img Installation

4.3. Annotate Images

4.3.1. Open LabelImg Tool

After installation of LabelImg tool, use the command below to launch the tool.

```
$ labelImg
```

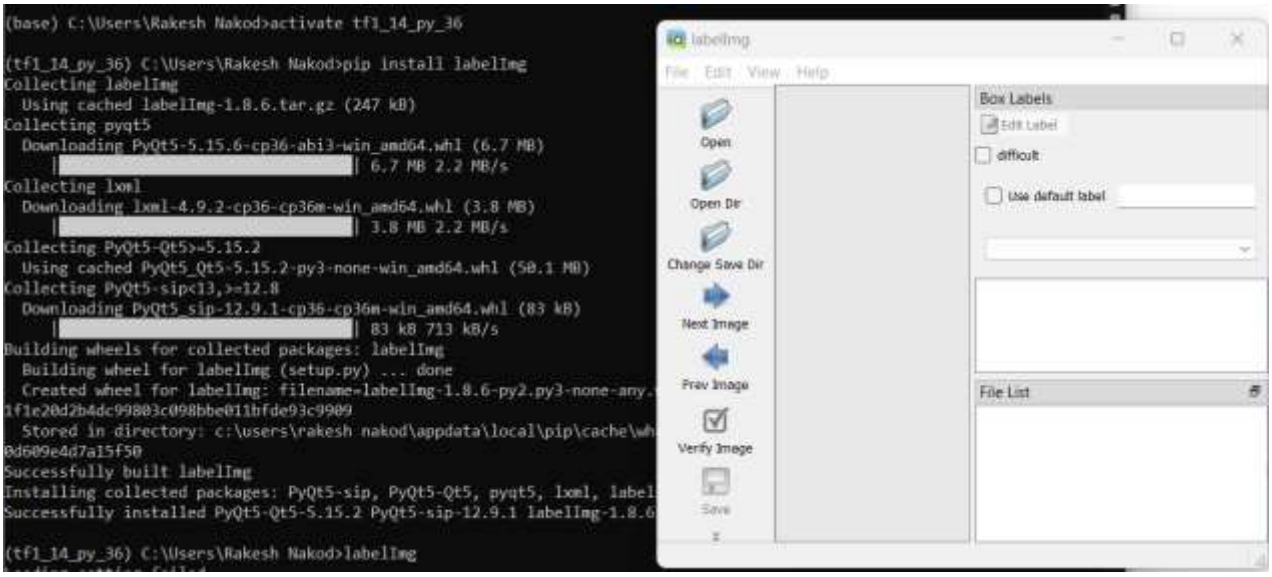

Figure 4.4. Label Img Tool

1. Click on “Open Dir” menu option on left of LabelImg tool, and select your Dataset directory that contains the images.

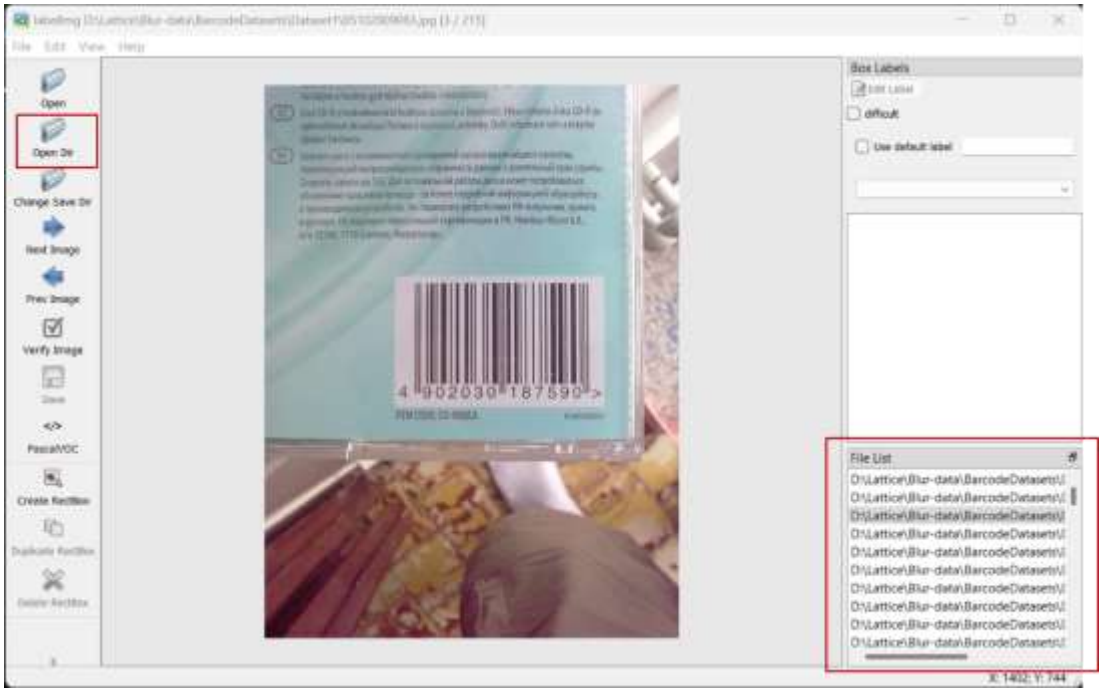

Figure 4.5. Open Dataset Directory

2. Click on “Change Save Dir” menu option to select a folder where you want to save your annotation file.

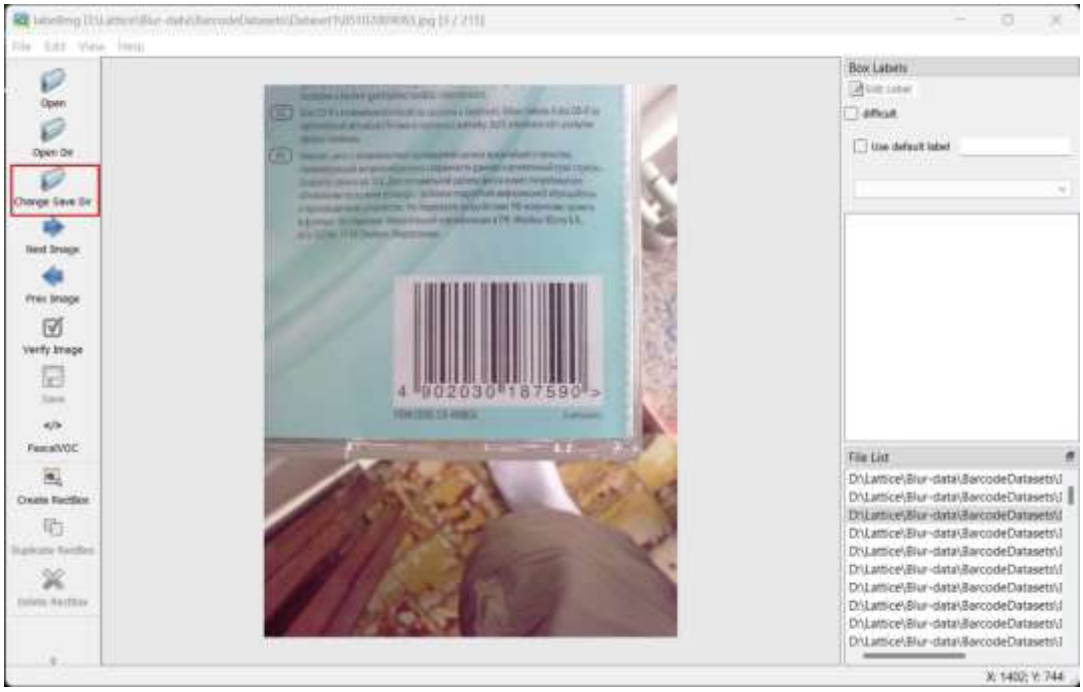

Figure 4.6. Saving Output Directory

3. Click on the “Create RectBox” menu option to draw bounding box around the barcode.

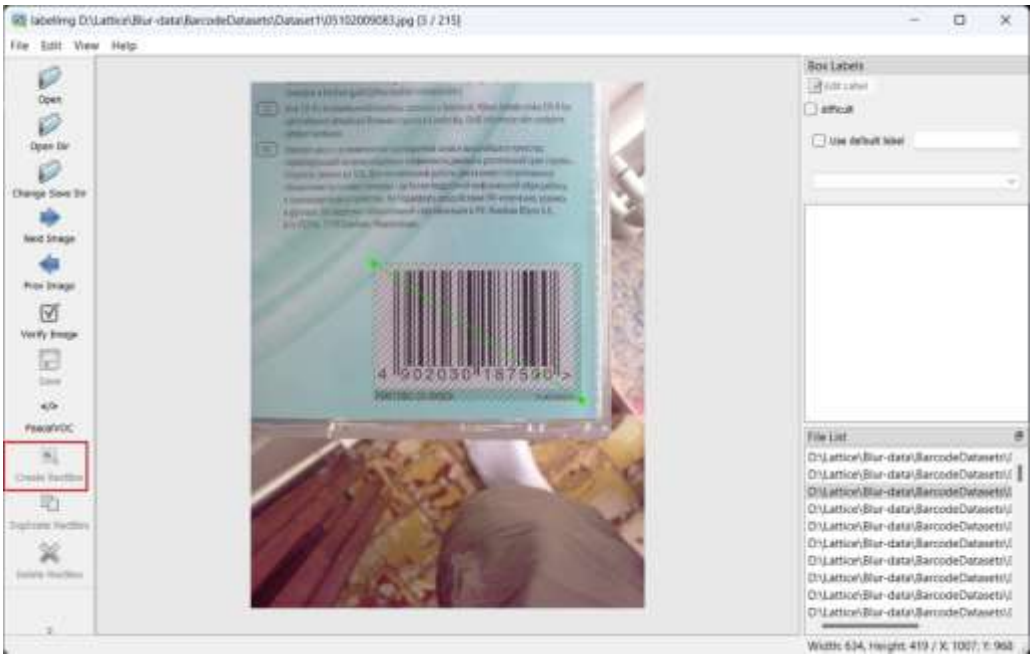

Figure 4.7. Drawing Bounding Box

4. Label them as “barcode” and click on **OK**. After that, the “Save” menu option becomes active. You can now save the label for the image.

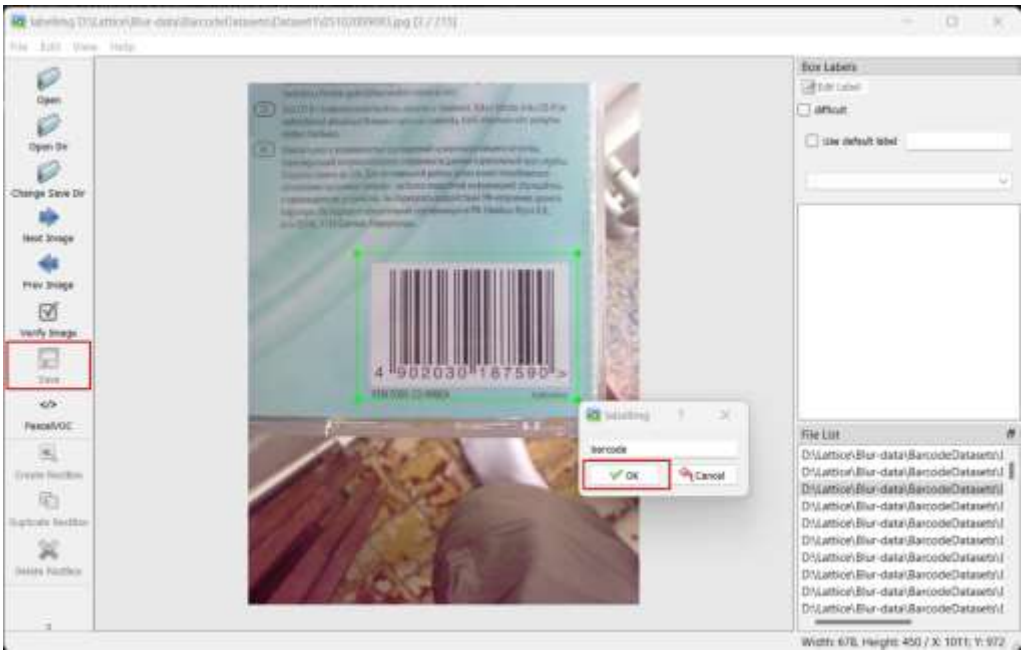

Figure 4.8. Saving Label and Image

5. All saved images can be reviewed. If required, the “Delete Box” menu option enables you to remove or edit the existing bounding box.

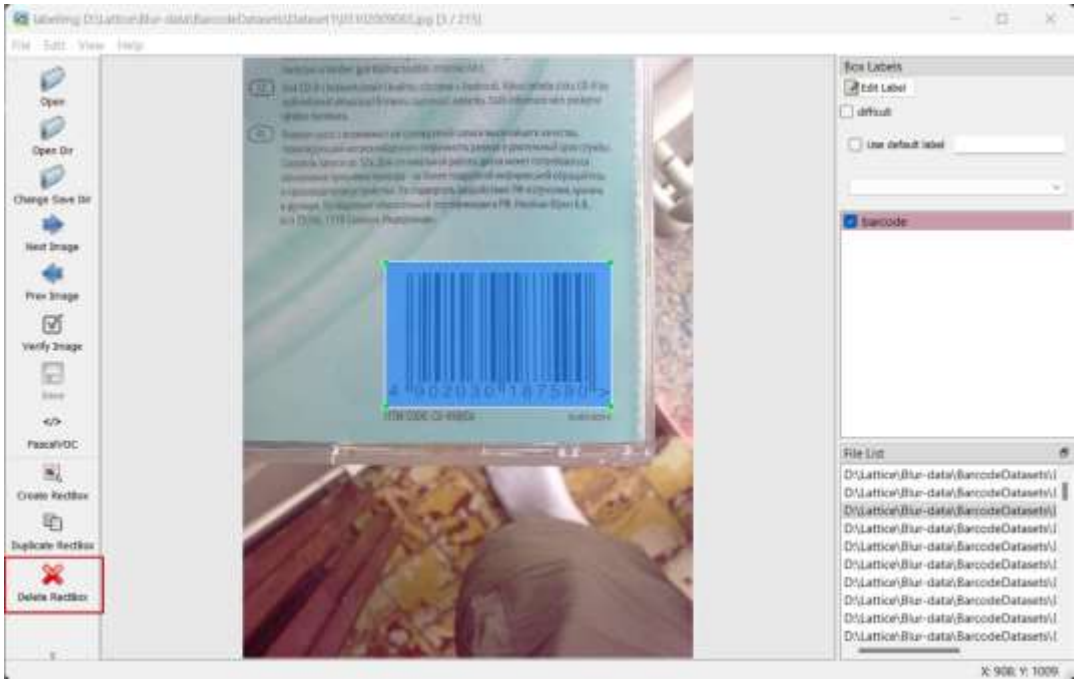

Figure 4.9. Label Img Tool

6. All the labelled images have the corresponding voc format .xml files created in the Change save directory.

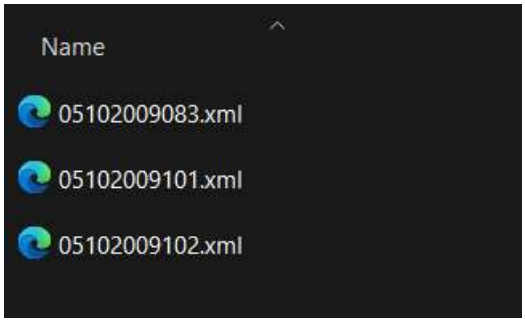

Figure 4.10. Output Label Folder

4.4. Convert LabelImg Tools VOC XML Label Format to kitti Format

- Download the Lattice Barcode Detection and Reading demo training code.
- Go to “data_utils” directory. This directory contains *“voc_xml_to_kitty.py”* file.
- Run command below to convert “VOC XML for images” format to “Kitti” format.

```
$ python voc_xml_to_kitty.py --input _dir < path to output folder of labelImg tool containing xml files> --output <Output folder for saving kitti format files>
```

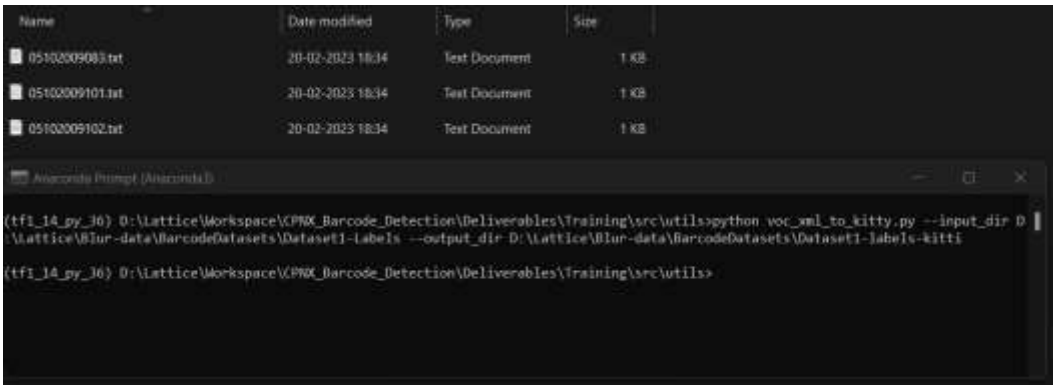

Figure 4.11. Output kitti Label Folder

```
barcode 0.0 0.0 0.0 380 544 982 931
```

Figure 4.12. Kitti Label Format

5. Training Code Preparation

5.1. Neural Network Architecture

5.1.1. Neural Network Architecture

This section provides information on the Convolution Neural Network configuration of the Barcode Detection design.

<table>
<tr><td>1x160x160x1</td><td></td></tr>
<tr><td>ConvBNReLU-16</td><td rowspan="12">ConvBNRelU - # where:
• Conv3x3-BatchNorm-ReLU
• # - The number of filters
For example, ConvBNReLU - 8 = 8 3 x 3 convolution filter followed by BatchNorm and ReLU

BottleNeckCSP - # where:
- The number of outputs</td></tr>
<tr><td>MaxPool</td></tr>
<tr><td>ConvBNReLU-32</td></tr>
<tr><td>MaxPool</td></tr>
<tr><td>BottleneckCSP-32</td></tr>
<tr><td>ConvBNReLU-64</td></tr>
<tr><td>BottleneckCSP-64</td></tr>
<tr><td>ConvBNReLU-128</td></tr>
<tr><td>BottleneckCSP-128</td></tr>
<tr><td>ConvBNReLU-256</td></tr>
<tr><td>BottleneckCSP-256</td></tr>
<tr><td>ConvBNReLU-256</td></tr>
<tr><td>Conv1x1-20</td></tr>
</table>

Figure 5.1. Yolov5 Architecture

As shown in the diagram above (Figure 5.1), this model contains Convolution (Conv), batch normalization (BN), Relu and Bottleneck CSP.

- 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 the 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, such as 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 (say curve) it is looking for is in the input volume.
  - Relu (Activation layer)

    After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations). In the past, nonlinear functions like tanh and sigmoid were used. Researchers found out that Relu layers work far better. The network is able to train a lot faster due to 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.

- Bottleneck CSP

  CSP stands for cross stage partial network. YOLO is a deep network. YOLO uses residual and dense blocks to enable the flow of information to the deepest layers and to overcome the vanishing gradient problem. However, one of the perks of using dense and residual blocks is the problem of redundant gradients. CSP helps tackling this problem by truncating the gradient flow.
  YOLOv5 employs CSP strategy to partition the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy as shown in the figure below (Figure 5.2).

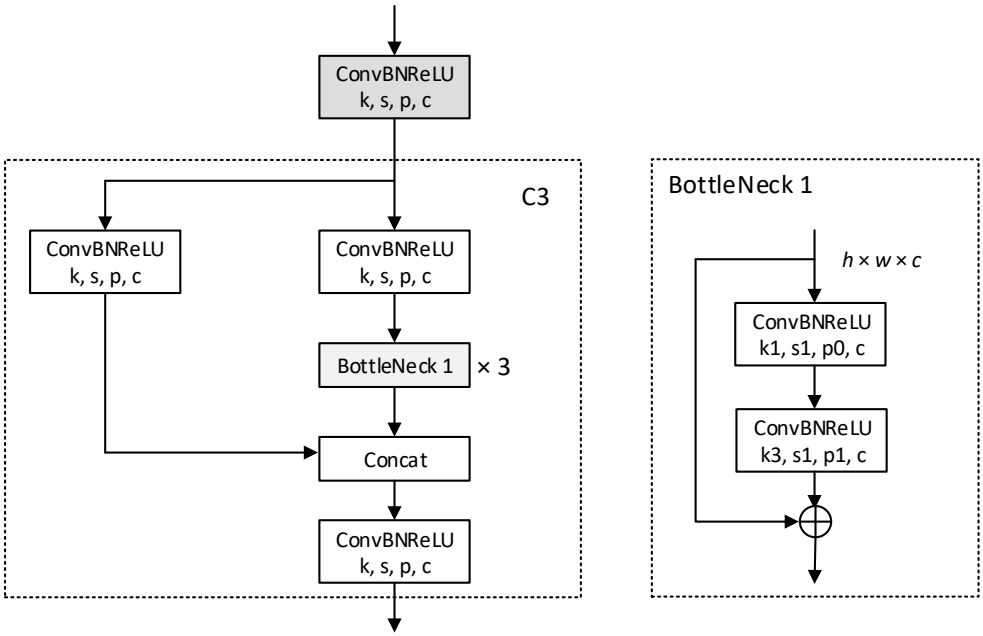

Figure 5.2. Bottleneck CSP Architecture

  Applying this strategy comes with big advantages to YOLOv5, since it helps reducing the number of parameters and helps reducing an important amount of computation (less FLOPS) which lead to **increasing the inference speed** that is crucial parameter in real-time object detection models.

- Pooling Layer

  After some Relu layers, programmers may choose to apply a pooling layer. It is also referred to as a down sampling layer. In this category, there are also several layer options, with Maxpooling being the most popular. This basically takes a filter (normally of size 2 × 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 we know that a specific feature is in the original input volume (there is to be a high activation value), its exact location is not as important as its relative location to the other features. As you can imagine, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. The first is that the number of parameters or weights is reduced by 75%, thus lessening the computation cost. The second is that it can control over fitting. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of over fitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.

- Batchnorm Layer

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

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

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

Calculate the mean and variance of the layers input:

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

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

- Drop-out Layer

  Drop-out layers have a very specific function in neural networks. After training, the weights of the network are so tuned to the training examples they are given that the network does not perform well when given new examples. The idea of dropout is simplistic in nature. This layer “drops out” a random set of activations in that layer by setting them to zero. It forces the network to be redundant. That means the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. It makes sure that the network is not getting too “fitted” to the training data and thus helps alleviate the over fitting problem.

  Note: This layer is only used during training, but not during test time.

- Fully connected Layer

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

- Quantization

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

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

5.1.2. Barcode Detection Network Output

Barcode Detection network gives Output tensor of dimension (BATCH_SIZE, Anchor Width, Anchor Height, 60). This can be interpreted by Yolov5 detection.

Barcode Detection Demo has one class as below:

- barcode

From the input image model, first extracts feature maps, overlays them with a W × H 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(Obj)xIOU)
- C conditional classes

Hence the current model architecture has a fixed output of W × H × K (4+1+C). Where

- W, H = Grid Size
- K = Number of Anchor boxes
- C = Number of classes for which we want detection

As per above description, model has total 12000 output values. It is derived from following:

```
10 × 10 grid
     20 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, total 10 × 10 × 20 × 6 = 12000 output values.

If your images are smaller, we recommend stretching to default size. You can also up-sample them beforehand. We tried a smaller image size with the resulting sparser grid, and it did not seem to work out.

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.

5.1.3. Training Code Overview

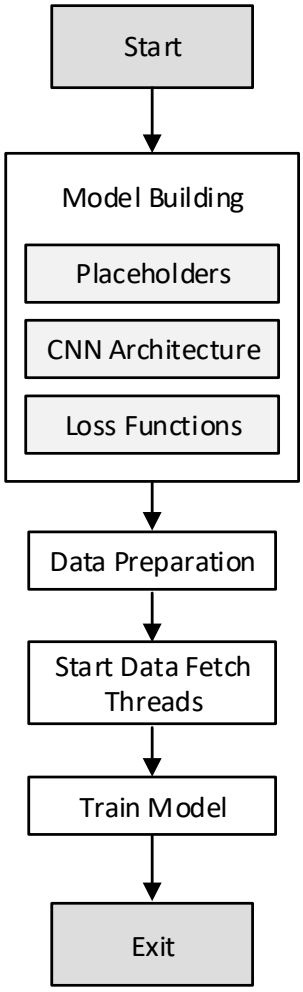

Figure 5.3. Training Code Flow Diagram

Training code can be 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 the subsequent sections.

Model Config

Demo uses Kitti dataset and Yolov5 model. *kitti_yolov5_config()* maintains all the configurable parameters for the model. Summary of configurable parameters is shown below:

- 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 = 160
mc.IMAGE_HEIGHT = 160
```

Figure 5.4. Code Snippet: Snippet: Input Image Size Config

- Grid dimension would be H = 10 and W = 10. *anchor_shapes* variable of *set_anchors()* in *src/config/kitti_yolov5_config.py* indicates anchors width and heights. Update it based on anchors per grid size changes.

```
def set_anchors(mc):
    H, W, B = 10, 10, 20
```

Figure 5.5. Code Snippet: Anchors Per Grid Config #1 (grid sizes)

- Batch size

  Change *mc.BATCH_SIZE* in *src/config/kitti_yolov5_config.py* to configure batch size.
- Output classes

  Edit classes in src/config/config.py as shown below.

```
cfg.CLASS_NAMES = ("barcode",)
```

Figure 5.6. Code Snippet: Classes

- Anchors per grid
  - Change mc.ANCHOR_PER_GRID in src/config/kitti_yolov5_config.py to configure anchors per grid.

```
mc.ANCHOR_BOX = set_anchors(mc)
mc.ANCHORS = len(mc.ANCHOR_BOX)
mc.ANCHOR_PER_GRID = 20
```

Figure 5.7. Code Snippet: Anchors Per Grid Config #2

  - Change hard coded anchors per grid in *set_anchors()* in *src/config/kitti_yolov5_config.py*. Here B (value 20) indicates anchors per grid.
  - If you want to run network on your own dataset, one needs to adjust the anchor sizes. Anchors are kind of prior distribution over what shapes your boxes should have. The better this fit to the true distribution of boxes, the faster and easier your training can be.
  - In order to determine anchor shapes, first load all ground truth boxes and pictures. If your images do not have all the same size, normalize their height and width by the images height and width. All images need be normalized before being fed to the network. We need to do the same to the bounding boxes and consequently, the anchors.
  - Second, perform a clustering on these normalized boxes. 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(
        [[ 22,  52],
        [ 19,  23],
        [ 52,  77],
        [ 69, 113],
        [ 35,  54],
        [ 62,  98],
        [ 23,  62],
        [ 56,  87],
        [ 38,  61],
        [ 40,  93],
        [ 14,  37],
        [ 46,  70],
        [ 16,  79],
        [ 43,  63],
        [ 15,  54],
        [ 81, 131],
        [ 20,  45],
        [ 12,  15],
        [ 28,  47],
        [ 32,  72]]
                    )] * H * W,(H, W, B, 2))
```

Figure 5.8. 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_yolov5_config.py*.

```
mc.WEIGHT_DECAY = 0.0001
mc.LEARNING_RATE = 0.00001
mc.DECAY_STEPS = 20000
mc.MAX_GRAD_NORM = 1.0
mc.MOMENTUM = 0.9
mc.LR_DECAY_FACTOR = 0.9

mc.LOSS_COEF_BBOX = 10.0
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 = 8

mc.DATA_AUGMENTATION = True
mc.DRIFT_X = 90
mc.DRIFT_Y = 60
mc.EXCLUDE_HARD_EXAMPLES = False
```

Figure 5.9. Code Snippet: Training Parameters

Model Building

Yolov5 class constructor builds model which can be divided into the following sections:

- Forward graph
- Interpretation graph
- Loss graph
- Training graph
- Visualization graph

Details of each graph are explained as follows.

- **Forward graph**
  - CNN architecture consists of Convolution, Batch normalization, Relu and Maxpool layers.
  - Forward graph consists of 1 fire layer and 17 bsconv layers.

```
fl_w_bin = 8
fl_a_bin = 8
ml_w_bin = 8
ml_a_bin = 8
ll_w_bin = 8
ll_a_bin = 16

min_rng = 0.0
max_rng = 2.0

bias_on = False

depth = [16, 32, 32, 64, 64, 128, 128, 256, 256]  # another thick version #12
```

Figure 5.10. Code Snippet: Filter Values

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

Figure 5.11. Code Snippet: Forward Graph Last Convolution Layer

- **Interpretation graph**

  This 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 10 × 10 × 120 tensor. There are 120 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 120 channels are not stored consecutively but are scattered all over the place. We need to sort these out somehow. Diagram below (Figure 5.12) explains the details.

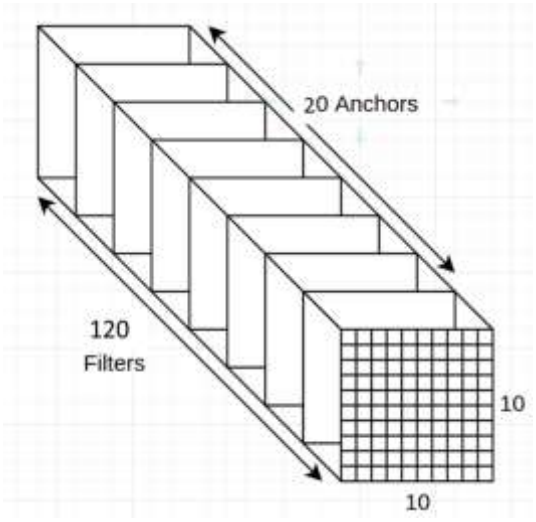

Figure 5.12. Grid Output Visualization #1

For each grid, cell values are aligned as the diagram shown below (Figure 5.13):

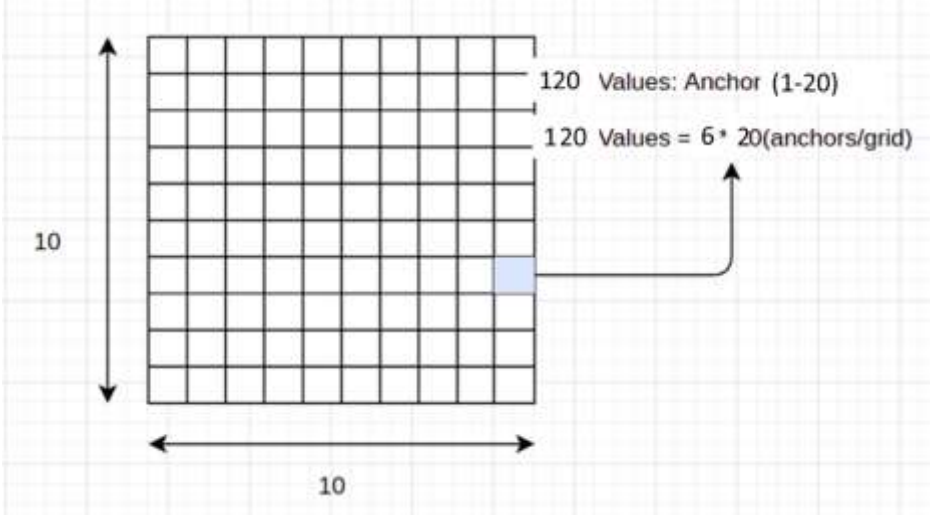

Figure 5.13. Grid Output Visualization #2

As we can see in the code below (Figure 5.14), output from fire_o layer (4d array of batch size × 10 × 10 × 120) needs to be sliced with proper index to get all values of probability, confidence & coordinates.

```
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'
)
```

```
# 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'
)
```

Figure 5.14. Code Snippet: Interpret Output Graph

For confidence score, we want this to 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. We apply a softmax make it a nice 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 5.15. Code Snippet: Bbox Loss

  - Confidence score
    - To obtain meaningful confidence score, the predicted value of each box is regressed against the Intersection over Union of the real and the predicted box. During training, we 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. In this way, we only include the loss generated by the “responsible” anchors.
    - As there can be multiple objects per image, we 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 5.16. 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 we would for image classification.

```
with tf.variable_scope('class_regression') as scope:
    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'
    )
```

Figure 5.17. Code Snippet: Class Loss

So, in one model architecture, we obtain the bounding box prediction, the classification, as well as the confidence score.

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

```
opt = tf.train.MomentumOptimizer(learning_rate=lr, momentum=mc.MOMENTUM)
grads_vars = opt.compute_gradients(self.loss, tf.trainable_variables())

with tf.variable_scope('clip_gradient') as scope:
    for i, (grad, var) in enumerate(grads_vars):
        grads_vars[i] = (tf.clip_by_norm(grad, mc.MAX_GRAD_NORM), var)

apply_gradient_op = opt.apply_gradients(grads_vars, global_step=self.global_step)
```

Figure 5.18. Code Snippet: Optimizer

- Training

```
feed_dict, org_img_per_batch, image_per_batch, label_per_batch, bbox_per_batch = \
    _load_data(load_to_placeholder=False)
op_list = [
    model.train_op, model.loss, summary_op, model.det_boxes,
    model.det_probs, model.det_class, model.conf_loss,
    model.bbox_loss, model.class_loss
]
_, loss_value, summary_str, det_boxes, det_probs, det_class, conf_loss, bbox_loss, class_loss = sess.run(
    op_list, feed_dict=feed_dict)
```

Figure 5.19. Code Snippet: Training

sess.run feeds the data and labels batches to network and optimizes the weights and biases.

5.2. Training

To train the machine:

1. Modify training script

   Training script at @train.sh is used to trigger training. Figure 5.20 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="merged_dataset/train" \
  --net="yolov5" \
  --summary_step=100 \
  --checkpoint_step=10000 \
  --max_steps=300000 \
  --gpu=$GPUID
  --checkpoint_step=1000
```

Figure 5.20. Code Snippet: Training

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

```
$ bash train.sh
100%|                                                                                | 172125/172125 [14:07<00:00, 203.07it/s]
========== ENet Class Weights ==========
 [1.6781622214015108, 5.825416365192917]
2022-05-16 11:44:07.422311: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1
2022-05-16 11:44:07.481669: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: NVIDIA GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.65
pciBusID: 0000:01:00.0
2022-05-16 11:44:08.003479: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 3491835000 Hz
2022-05-16 11:44:08.004207: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
2022-05-16 11:44:08.004715: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: NVIDIA GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.65
2022-05-16 11:44:08.409419: I tensorflow/compiler/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA GeForce RTX 2080 Ti, Compute Capability 7.5
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting standard services.
INFO:tensorflow:Saving checkpoint to path ./log/Enet/model.ckpt
INFO:tensorflow:Starting queue runners.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Epoch 1.0/300
2022-05-16 11:44:45.263380: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10.0
INFO:tensorflow:Current Learning Rate: [0.0005]
2022-05-16 11:44:59.804109: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
2022-05-16 11:45:05.888458: W tensorflow/stream_executor/cuda/redzone_allocator.cc:312] Not found: ./bin/ptxas not found
Relying on driver to perform ptx compilation. This message will be only logged once.
INFO:tensorflow:global step 0: loss: 0.6934 (15.45 sec/step)  Current Streaming Accuracy: 0.0000  Current Mean IOU: 0.0000
INFO:tensorflow:---VALIDATION--- Validation Accuracy: 0.8000    Validation Mean IOU: 0.8000    (3.44 sec/step)
INFO:tensorflow:---VALIDATION--- Validation Accuracy: 0.7666    Validation Mean IOU: 0.3833    (0.04 sec/step)
```

Figure 5.21. Execute Run Script

3. Start TensorBoard.

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

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

```
$ tensorboard --logdir logs_yolo/
TensorBoard 1.15.0 at http://pc-MS-1:6066/ (Press CTRL+C to quit)
```

Figure 5.22. TensorBoard – Generated Link

5. Check the training status on TensorBoard.

Figure 5.23. TensorBoard

Figure 5.24 shows the image menu of TensorBoard.

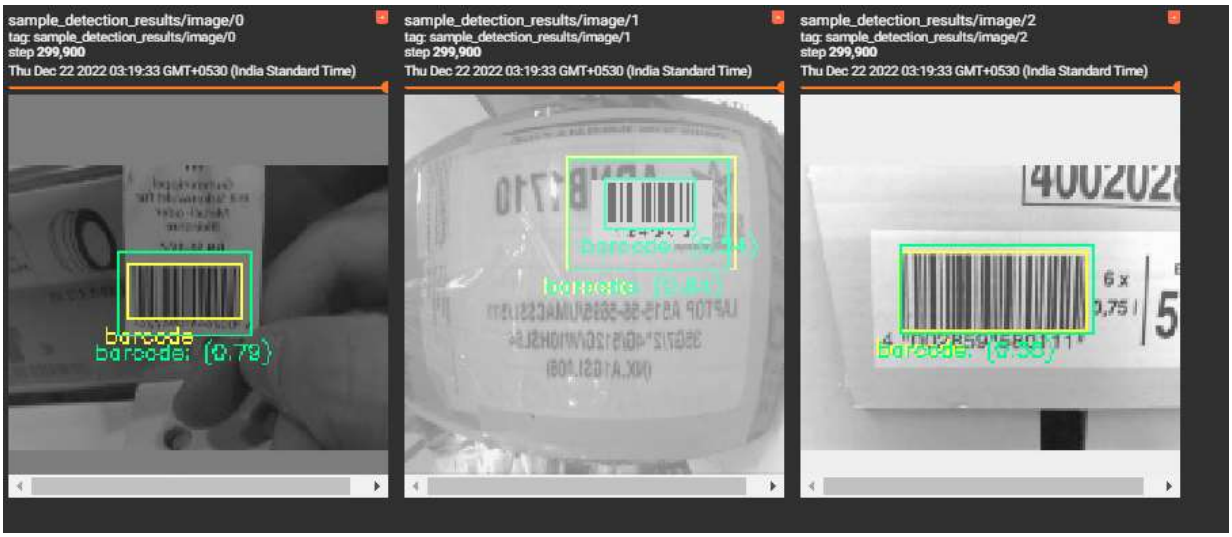

Figure 5.24. Image Menu of TensorBoard

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

```
model.ckpt-290000.data-00000-of-00001
model.ckpt-290000.index
model.ckpt-290000.meta
model.ckpt-299999.data-00000-of-00001
model.ckpt-299999.index
model.ckpt-299999.meta
model_metrics.txt
```

Figure 5.25. Example of Checkpoint Data Files at Log Folder

6. Creating Frozen File

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

6.1. Generating the Frozen (.pb) File

```
$ python genpb.py --ckpt_path <COMPLETE_PATH_TO_LOG_DIRECTORY>/model.ckpt-<ckpt
number> --input_node_name image_input -output_node_name fire_o/convolution
Example: python genpb.py -ckpt_path logs/yolov5/train/model.ckpt-499999. --
input_node_name batch -output_node_name fire_o/convolution
```

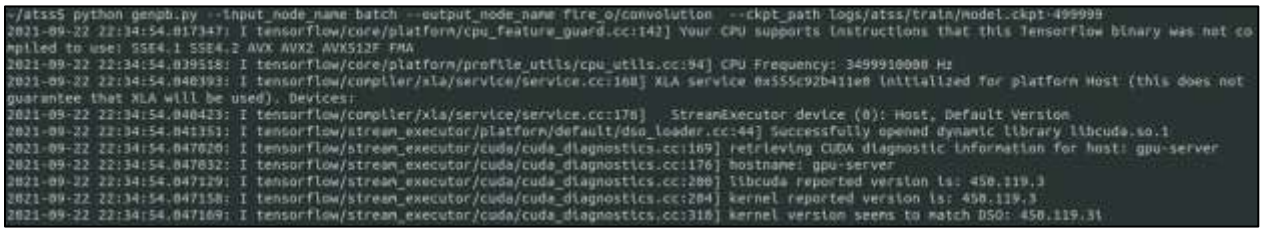

Figure 6.1. Run genpb.py to Generate Inference .pb

- *genpb.py* uses the latest checkpoint in train directory to generate frozen '.pb' file.
- Once the genpb.py is executed successfully <ckpt-prefix>_frozenforInference.pb file can be found in the log directory, as shown in screenshot above (Figure 6.1).

```
model.ckpt-299999.data-00000-of-00001
model.ckpt-299999.index
model.ckpt-299999.meta
model.pbtxt
model_frozenforInference.pb
model_metrics.txt
```

Figure 6.2. Frozen Inference. pb Output

7. Model Evaluation

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

7.1. Run Inference on test set

Barcode code contains 'inference.py' under the inference directory as shown below (Figure 7.1).

Note: If you make any change in the training code regarding image size, number of anchors or grid size, you must replicate those changes in the inference script.

Run the command below to run inference on test set.

```
$ python inference.py –pb <converted pb path> --input_image <test set images path>
```

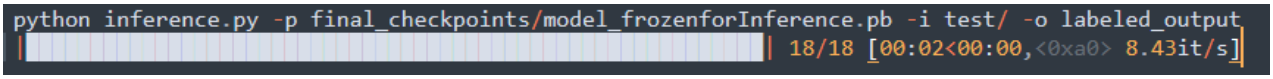

Figure 7.1. Run Inference

The command above can save images with bbox drawn in *inference_output/image_output* and resultant kitti output in *inference_output/predictions*, as shown in Figure 7.2 below.

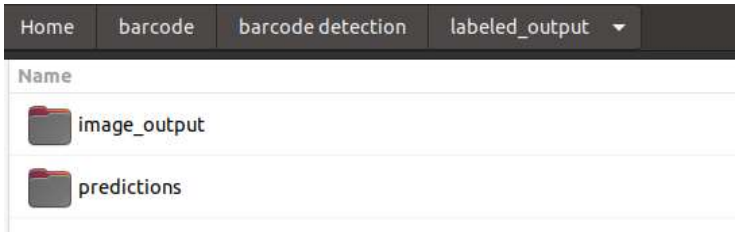

Figure 7.2. Inference Output

7.2. Calculate MAP

Barcode Detection code contains main.py under the Training directory as shown in Figure 7.3.

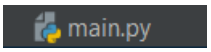

Figure 7.3. mAP File

Run the command below to calculate mAP using predictions generated from inference and groundtruth from test set.

```
$ python .\main.py --input_images ..\training\datasets\test\images\ --
ground_truth ..\training\datasets\t
est\labels\ --predictions ..\training\datasets\test\images_out\predictions\ --
no-animation --no-plot
```

```
mAP$ python main.py --input_images <input test images path> \
                    --ground_truth <input testset groundtruth path> \
                    --predictions <pridictions path> \
                    --no-animation  --no-plot
XX.XX% = person AP
mAP = XX.XX%
```

Figure 7.4. mAP Calculation

After the successful run of the script, mAP for each class and the total mAP are shown.

8. Creating Binary File with Lattice sensAI

This section describes how to generate binary file using the Lattice sensAI version 6.0 program.

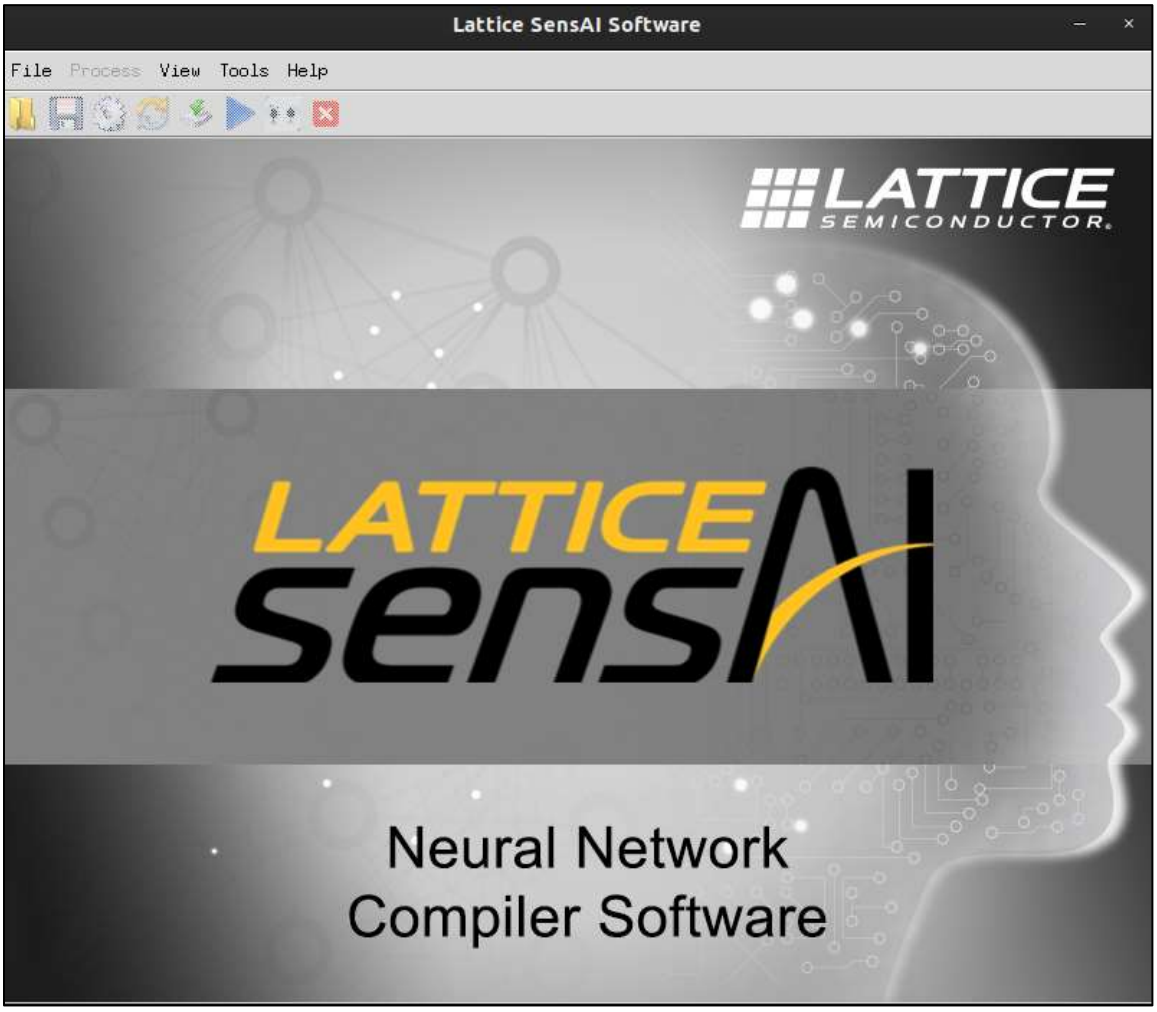

Figure 8.1. sensAI – Home Screen

To create the project in sensAI tool:

1. Click **File > New**.
2. Enter the following settings:
   - Project name
   - Framework – TensorFlow
   - Class – CNN
   - Device – CertusPro-NX
   - IP – Advanced_CNN
3. Click **Network File** and select the network (PB) file (Figure 8.2).

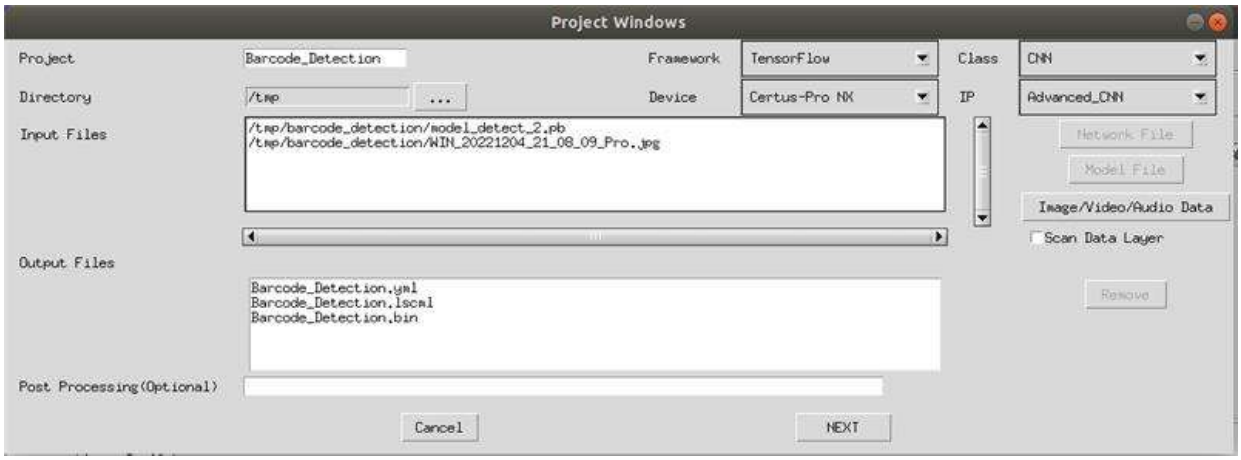

Figure 8.2. sensAI – Select Framework, Device, and Network File

4. Click the **Image/Video/Audio Data** button and select the input image file (Figure 8.3).

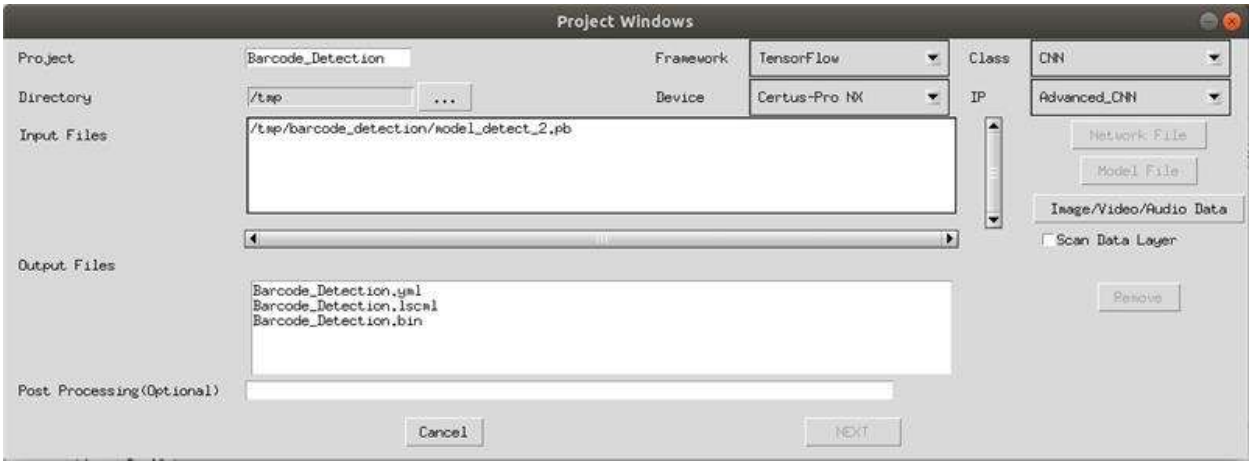

Figure 8.3. sensAI – Select image Data File

5. Click **Next**.
6. Set the following attributes:
   - Mean Value for Data Pre-Processing: 0
   - Scratch Pad Memory Block Size: 8192
   - ARGS MAX Size: 4096
   - External Memory Interfaced: 8388608
   - Scale Value for Data Pre-Processing: 0.0078125

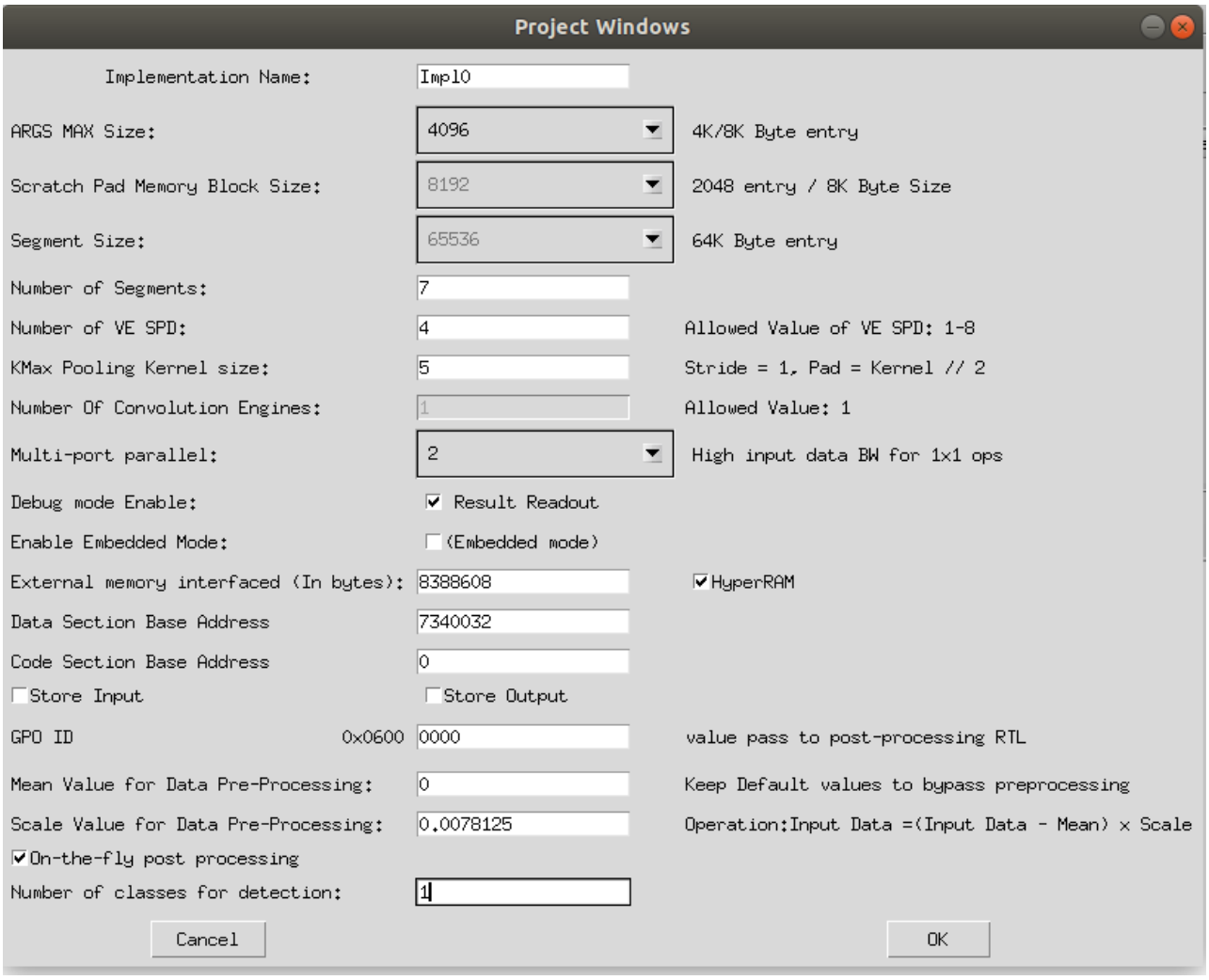

Figure 8.4. sensAI – Update Project Settings

7. Clock **Ok** to create project.
8. Double click on **Analyze** (Figure 8.5).

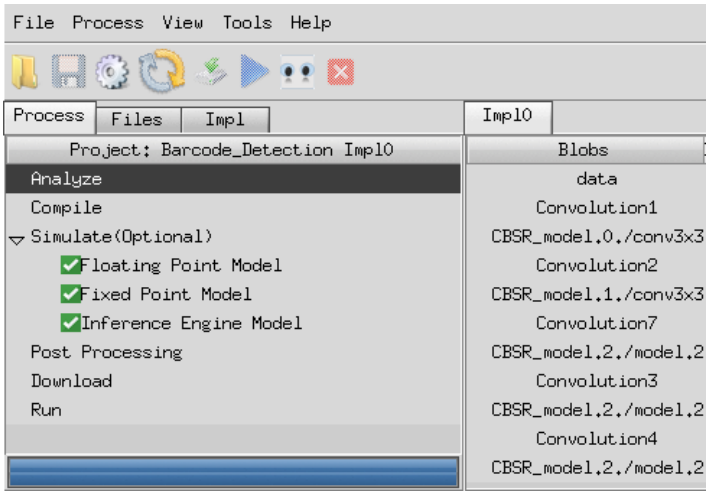

Figure 8.5. Analyze Project

9. Confirm the Q format of each layer as shown in Figure 8.6, Figure 8.7, and Figure 8.8.

Impl0

Blobs	Data Format (Analyzed)	Stored Data Format	Internal(External) Mem Bytes
data	8,7	1,7	None
Convolution1	5,10	5,10	None
CBSR_model.0./conv3x3/convolution	8,7	1,7	None
Convolution2	5,10	5,10	None
CBSR_model.1./conv3x3/convolution	8,7	1,7	None
Convolution7	5,10	5,10	None
CBSR_model.2./model.2.bottleneck_csp_y2/conv3:	8,7	1,7	None
Convolution3	5,10	5,10	None
CBSR_model.2./model.2.bottleneck_csp_x1/conv3:	8,7	1,7	None
Convolution4	5,10	5,10	None
CBSR_model.2./model.2.bottleneck_csp_x10/mode.	8,7	1,7	None
Convolution5	5,10	5,10	None
CBSR_model.2./model.2.bottleneck_csp_x10/mode.	8,7	1,7	None
Eltwise1	5,10	5,10	None
Elt_quant_model.2./model.2.bottleneck_csp_x10.	8,7	1,7	None
Convolution6	5,10	5,10	None
CBSR_model.2./model.2.bottleneck_csp_y1/conv3:	8,7	1,7	None
Concat_model.2./concat_Placeholder	5,10	5,10	None
Concat_model.2./concat	8,7	1,7	None
Convolution8	5,10	5,10	None
CBSR_model.2./model.2.bottleneck_csp_out/conv:	8,7	1,7	None
Convolution9	5,10	5,10	None
CBSR_model.3./conv3x3/convolution	8,7	1,7	None
Convolution18	5,10	5,10	None
CBSR_model.4./model.4.bottleneck_csp_y2/conv3:	8,7	1,7	None
Convolution10	5,10	5,10	None
CBSR_model.4./model.4.bottleneck_csp_x1/conv3:	8,7	1,7	None
Convolution11	5,10	5,10	None
CBSR_model.4./model.4.bottleneck_csp_x10/mode.	8,7	1,7	None
Convolution12	5,10	5,10	None
CBSR_model.4./model.4.bottleneck_csp_x10/mode.	8,7	1,7	None
Eltwise2	5,10	5,10	None

Figure 8.6. Q Format Settings for Each Layer (1)

Impl0

Blobs	Data Format (Analyzed)	Stored Data Format	Internal(External) Mem Bytes
CBSR_model.4./model.4.bottleneck_csp_x10/mode.	8.7	1.7	None
Eltwise2	5.10	5.10	None
Elt_quant_model.4./model.4.bottleneck_csp_x10.	8.7	1.7	None
Convolution13	5.10	5.10	None
CBSR_model.4./model.4.bottleneck_csp_x11/mode.	8.7	1.7	None
Convolution14	5.10	5.10	None
CBSR_model.4./model.4.bottleneck_csp_x11/mode.	8.7	1.7	None
Eltwise3	5.10	5.10	None
Elt_quant_model.4./model.4.bottleneck_csp_x11.	8.7	1.7	None
Convolution15	5.10	5.10	None
CBSR_model.4./model.4.bottleneck_csp_x12/mode.	8.7	1.7	None
Convolution16	5.10	5.10	None
CBSR_model.4./model.4.bottleneck_csp_x12/mode.	8.7	1.7	None
Eltwise4	5.10	5.10	None
Elt_quant_model.4./model.4.bottleneck_csp_x12.	8.7	1.7	None
Convolution17	5.10	5.10	None
CBSR_model.4./model.4.bottleneck_csp_y1/conv3:	8.7	1.7	None
Concat_model.4./concat_Placeholder	5.10	5.10	None
Concat_model.4./concat	8.7	1.7	None
Convolution19	5.10	5.10	None
CBSR_model.4./model.4.bottleneck_csp_out/conv:	8.7	1.7	None
Convolution20	5.10	5.10	None
CBSR_model.5./conv3x3/convolution	8.7	1.7	None
Convolution29	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_y2/conv3:	8.7	1.7	None
Convolution21	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x1/conv3:	8.7	1.7	None
Convolution22	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x10/mode.	8.7	1.7	None
Convolution23	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x10/mode.	8.7	1.7	None
Eltwise5	5.10	5.10	None
Elt_quant_model.6./model.6.bottleneck_csp_x10.	8.7	1.7	None
Convolution24	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x11/mode.	8.7	1.7	None
Convolution25	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x11/mode.	8.7	1.7	None
Eltwise6	5.10	5.10	None
Elt_quant_model.6./model.6.bottleneck_csp_x11.	8.7	1.7	None
Convolution26	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x12/mode.	8.7	1.7	None
Convolution27	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x12/mode.	8.7	1.7	None

Figure 8.7. Q Format Settings for Each Layer (2)

Impl0

Blobs	Data Format (Analyzed)	Stored Data Format	Internal(External) Mem Bytes
CBSR_model.6./model.6.bottleneck_csp_x12/mode.	8.7	1.7	None
Convolution27	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_x12/mode.	8.7	1.7	None
Eltwise7	5.10	5.10	None
Elt_quant_model.6./model.6.bottleneck_csp_x12.	8.7	1.7	None
Convolution28	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_y1/conv3:	8.7	1.7	None
Concat_model.6./concat_Placeholder	5.10	5.10	None
Concat_model.6./concat	8.7	1.7	None
Convolution30	5.10	5.10	None
CBSR_model.6./model.6.bottleneck_csp_out/conv:	8.7	1.7	None
Convolution31	5.10	5.10	None
CBSR_model.7./conv3x3/convolution	8.7	1.7	None
Convolution40	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_y2/conv3:	8.7	1.7	None
Convolution32	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x1/conv3:	8.7	1.7	None
Convolution33	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x10/mode.	8.7	1.7	None
Convolution34	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x10/mode.	8.7	1.7	None
Eltwise8	5.10	5.10	None
Elt_quant_model.8./model.8.bottleneck_csp_x10.	8.7	1.7	None
Convolution35	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x11/mode.	8.7	1.7	None
Convolution36	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x11/mode.	8.7	1.7	None
Eltwise9	5.10	5.10	None
Elt_quant_model.8./model.8.bottleneck_csp_x11.	8.7	1.7	None
Convolution37	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x12/mode.	8.7	1.7	None
Convolution38	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_x12/mode.	8.7	1.7	None
Eltwise10	5.10	5.10	None
Elt_quant_model.8./model.8.bottleneck_csp_x12.	8.7	1.7	None
Convolution39	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_y1/conv3:	8.7	1.7	None
Concat_model.8./concat_Placeholder	5.10	5.10	None
Concat_model.8./concat	8.7	1.7	None
Convolution41	5.10	5.10	None
CBSR_model.8./model.8.bottleneck_csp_out/conv:	8.7	1.7	None
fire_o/convolution	5.10	5.10	None
CBSR_fire_o/convolution	8.7	5.10	None

Figure 8.8. Q Format Settings for Each Layer (3)

10. Double click on **Compile** to generate the Firmware and filter binary file (Figure 8.9).

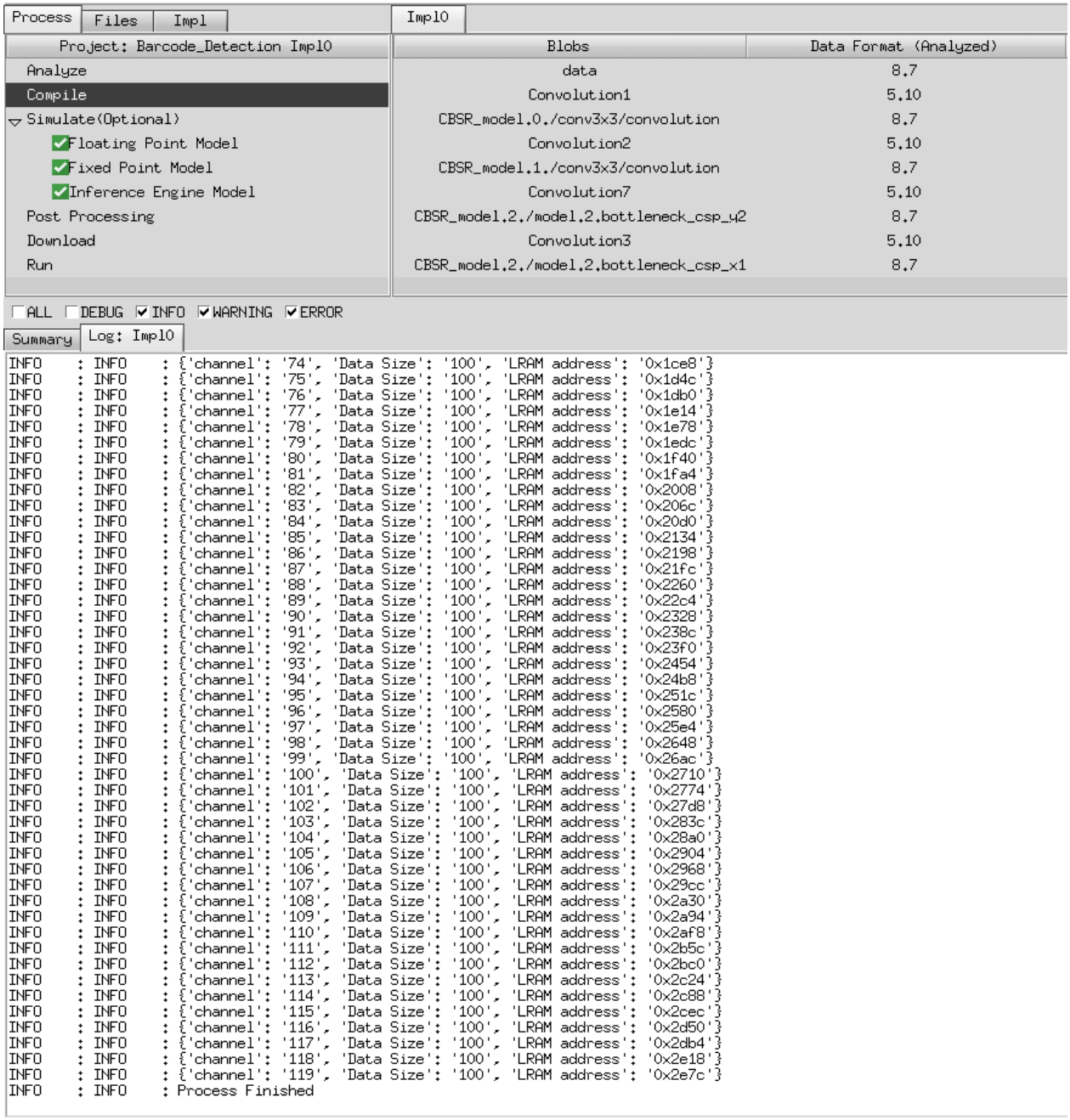

Figure 8.9. Compile Project

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

9. Hardware (RTL) Implementation

9.1. Top Level Information

9.1.1. Block Diagram

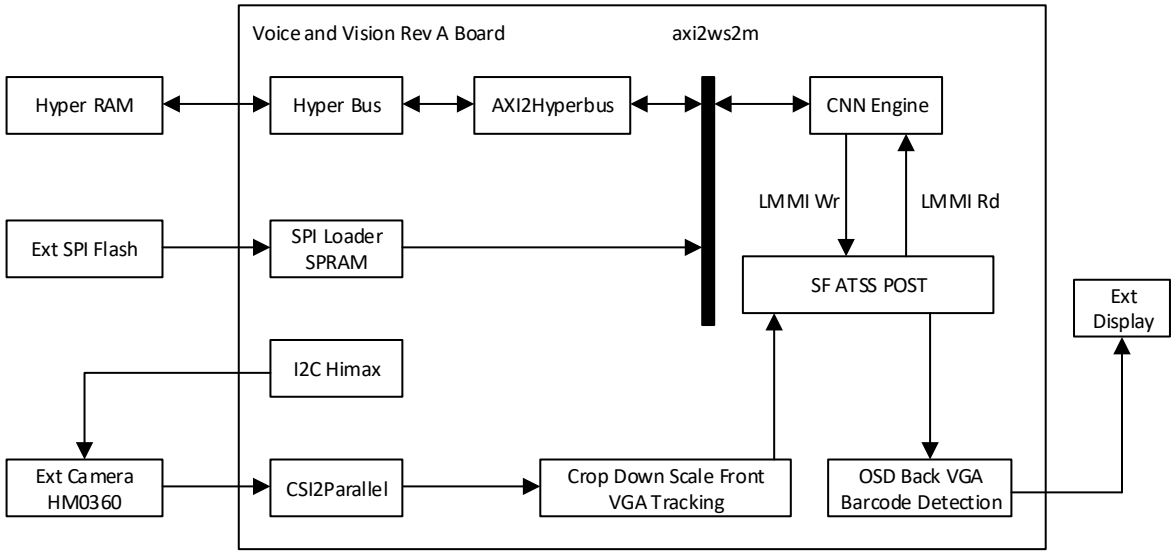

Figure 9.1. Top Block Diagram of Barcode Detection with CertusPro-NX Voice and Vision ML (Rev A) Board

9.1.2. Operational Flow

- The external camera Himax HM0360 is configured using I^2C Master Block *lsc_i2cm_himax.v*.
- The real time input image data is received by Video path. The RAW8 data from (csi2_to_parallel.v) is sent to pre-processing *crop_downscale_front_vga_tracking.v* which performs Crop and Downscale operation to provide compatible input image resolution of 160 × 160 to Extended CNN engine.
- The 6 MB firmware BIN file (.mcs) is loaded to the SPI Flash module *spi_loader_spram.v* configured with starting address 24'h300000 to end address 24'h600000.
- CNN Engine receives the downscaled image data from *sf_atss_post.v* through LMMI Write interface and the firmware file through AXI hyperbus interface to provide inference result output.
- CNN inference output is read again by sf_atss_*post.v* through LMMI Read interface and passed to OSD block for output display.
- For final output display, the *osd_back_vga_barcode_detection.v* module performs barcode detection on image received from Video path and downscaled image obtained from Crop and Downscale module using the CNN object detection pixel information obtained from sf_atss_*post.v* block.

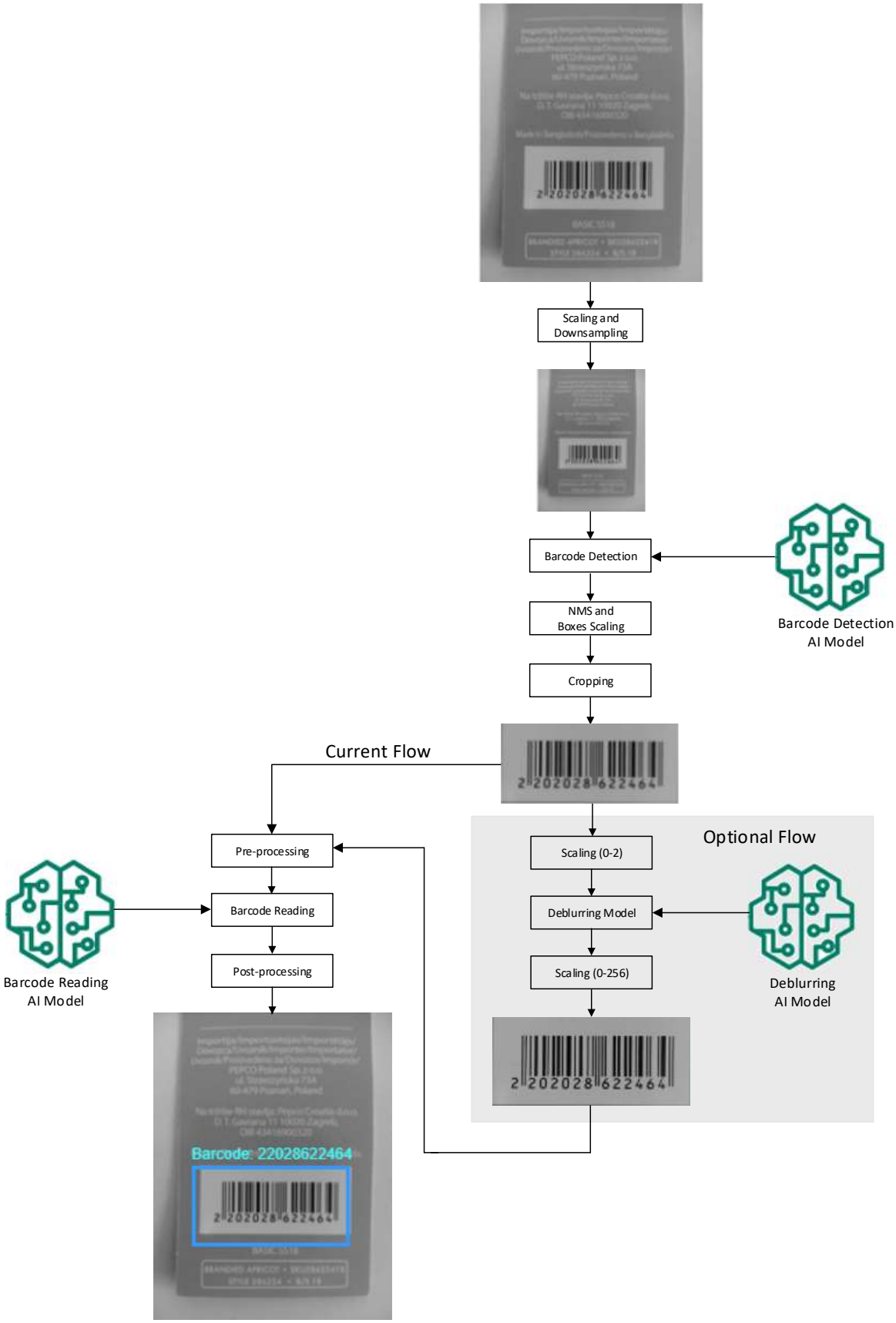

Figure 9.2. Top-level Process Flow Diagram

9.1.3. Core Customization

Table 9.1. Core Parameters

Parameter	Value	Description
USE_ML	1'b1	Indicates 1: Enable/0: Disable to use CNN engine.
EN_UART	1'b0	Indicates 1:Enable/0:Disable UART for video output.
FLASH_START_ADDR	24'h300000	Indicates starting address to load Firmware in external SPI flash.
FLASH_END_ADDR	24'h600000	Indicates ending address to load Firmware in external SPI flash.
CNN IP Attributes	**Default Value**	**Description**
Number of LRAMs	7	Number of LRAMs in the ML SPD.
Number of VE SPD Packs	8	1 VE SPD Pack = 4 VE SPDs. Number of EBRs in a VE SPD is the attribute below.
Number of EBRs in VE SPD	4	1 VE SPD Pack = 4 VE SPDs. This attribute specifies the number of EBRs in a VE SPD.
Max AXI4 external memory DMA Burst	31	Max burst limit on AXI4 bus during DMA transactions with external memory.
LMMI read mode	BYTE	Access width mode for LMMI read interface (reading data out of the IP) BYTE: byte mode; HWORD: 16 bit half word mode; WORD: 32 bit word mode.
LMMI Write mode	BYTE	Access width mode for LMMI write interface (writing data into the IP) BYTE: byte mode; HWORD: 16 bit half word mode; WORD: 32 bit word mode.
No. of Convolution engines	1	Number of parallel convolution engines. Higher compute throughput can be achieved with more number of engines, at the expense of higher utilization of DSP resources.
Enable 4 parallel ports for 1x1 convolution	Unchecked	Enable four parallel ports internally for high bandwidth data accesses for 1 × 1 convolutions.
Enable Vector ALU	Checked	Enables the Vector Engine for pixelwise ALU operations.
Enable 5x5 convolution	Checked	Enable 5 × 5 convolutions inside the Conv EU.
Enable 7x7 convolution	Checked	Enable 7 × 7 convolutions inside the Conv EU.
Enable Argmax pool	Checked	Enable the Argmax pool compute module.
Enable Maxpool stride=1 module	Checked	Enable the 1-D stride=1 maxpool compute engine.
Avant Mode	Unchecked	IMPORTANT: This must be enabled for the Avant devices, and disabled for the CertusPro-NX devices.

9.2. Architectural Details

9.2.1. Pre-processing Operation

- The *crop_downscale_front_vga_tracking.v* video processing block is used to crop and downscale the image data to make it compatible for CNN engine.
- Masking values for incoming image data from camera are set to capture the image data of resolution 640 × 480.
- As shown in image below, initially this 640 × 480 image is downscaled into 160 × 120 image resolution using block size 4, and to obtain 160 × 160 image for CNN input, 40 lines are then padded vertically.

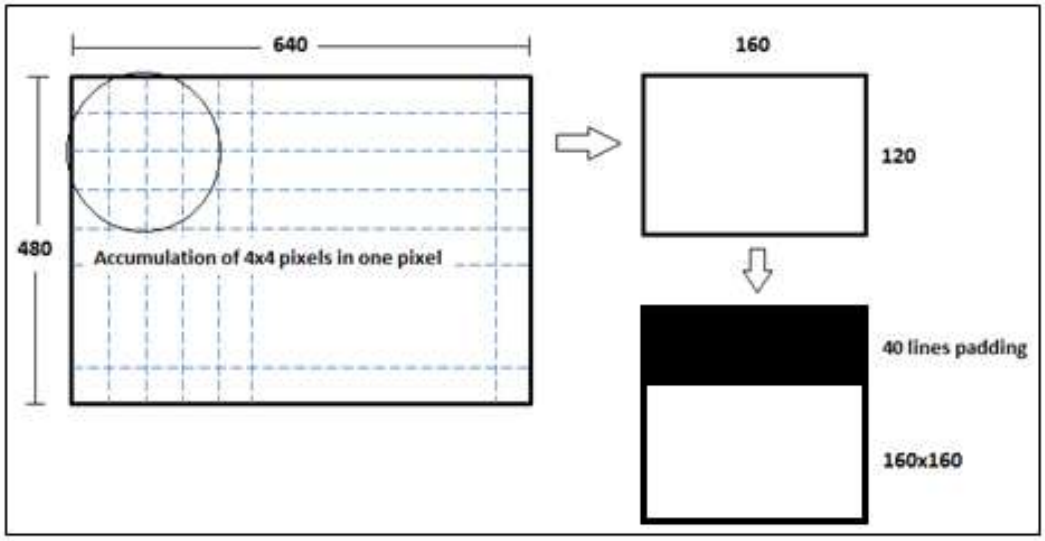

Figure 9.3. Downscaling Image

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

9.2.2. Post-processing operation

The post-processing operation is explained as follows.

- The *sf_atss_post.v* block mainly handles the task of providing downscaled input image to CNN for inference and receiving the detection results back.
- The writing and reading back data to and from Advanced CNN engine takes place over LMMI Interface when it is idle and not running.
- When CNN Engine is not running, this block initially receives the 160 × 160 downscaled image from pre-processing module and provides it to CNN over LMMI WRITE Interface *lmmi_write_i*.
- When CNN has done inference over this frame and again when it is idle, this module receives the CNN results over LMMI READ Interface *lmmi_rdata_o* and passes it on to the OSD module for display.
- The *osd_back_vga_barcode_detection.v* module received mainly the 160 × 160 downscaled image from pre-processing module and the CNN result data from post processing block.
- Using the pixel information of barcode detection received from post-processing block, the OSD module performs text and graphics addition over the 160 × 160 downscaled image.
- In the final segmentation output display, the barcode presence can be observed in gray-scale with bounding box.

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

10. Creating FPGA Bitstream File

This section provides the procedure of creating your FPGA bitstream file using Lattice Radiant Software.

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

1. Open Lattice Radiant Software, as shown in Figure 10.1.

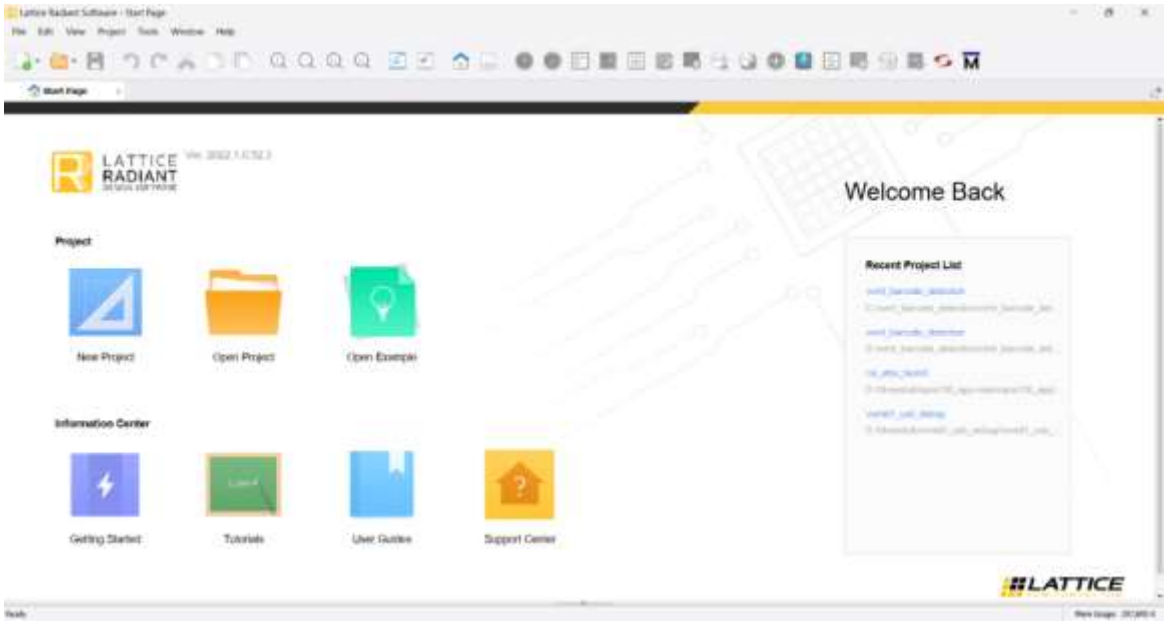

Figure 10.1. Radiant Software

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

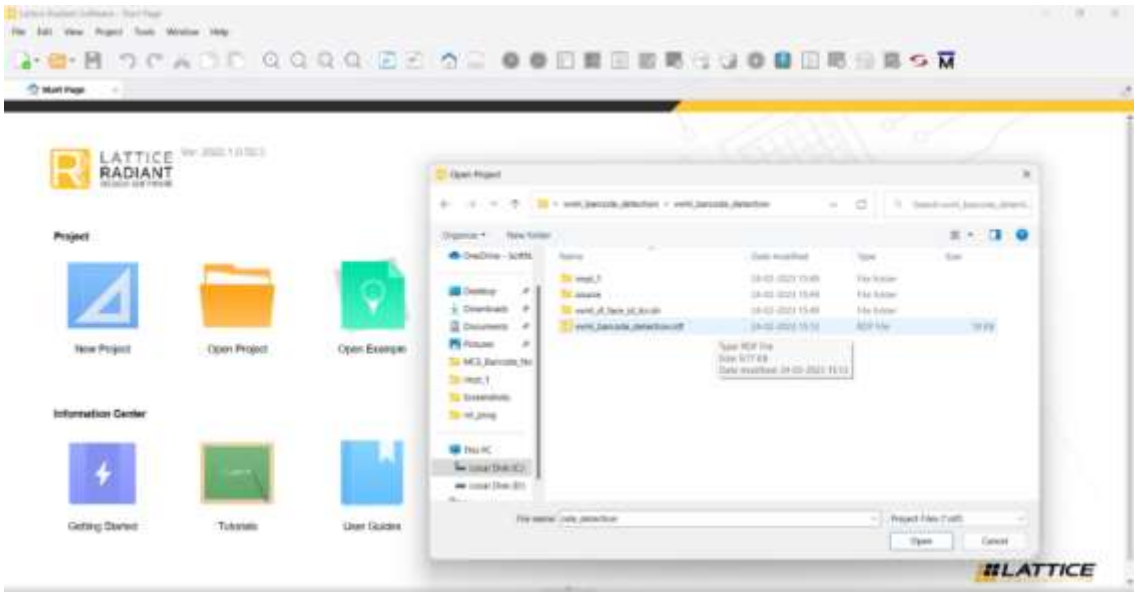

Figure 10.2. Radiant Software – Open Project

3. Click **Export Files** to generate the bit file. View the log message in Export Reports that indicates the generated bitstream. Find this bit file under */vvml_barcode_detection/impl_1,* as shown in Figure 10.3.

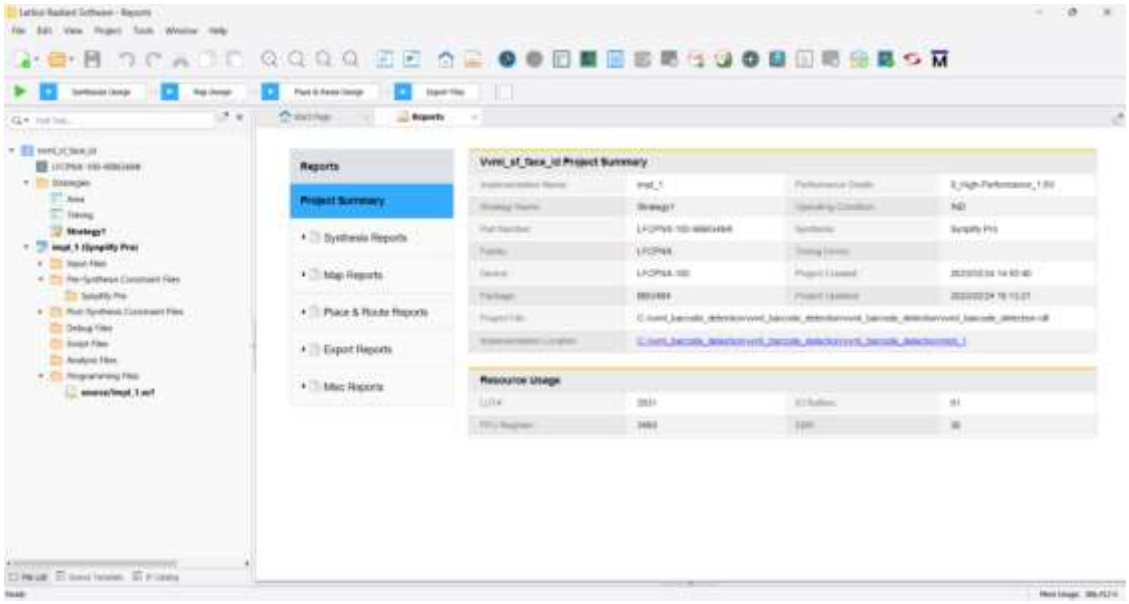

Figure 10.3. Radiant Software – Bitstream Generation Export Report

Technical Support Assistance

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

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/en/Support/AnswerDatabase.

Revision History

Revision 1.0, March 2023

Section	Change Summary
All	Initial release.

www.latticesemi.com