CLASSIFICATION OF ROAD SIDE MATERIAL USING CONVOLUTIONAL NEURAL NETWORK AND A PROPOSED IMPLEMENTATION OF THE NETWORK THROUGH ZEDBOARD ZYNQ 7000 FPGA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Tanvir Rahman

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Lauren Christopher, Chair

Department of Electrical and Computer Engineering

Dr. Maher Rizkalla

Department of Electrical and Computer Engineering

Dr. Paul Salama

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of the Graduate Program

This thesis is dedicated to my Parents, Shahnaj Pervin and MD Ataur Rahman.

ACKNOWLEDGMENTS

I would like thank Dr. Lauren Christopher, my major professor, for all of the guidance and support she has provided me with my research and thesis, along with the rest of my thesis committee, Dr. Maher Rizkalla and Dr. Paul Salama. I would also like to thank Toyota's collaborative safety research with Transport Active Safety Institute (TASI) at IUPUI for providing me an opportunity to participate in this emerging field of research.

I would also want to thank Dr. Ali Jafari to provide me the Support through the Jafaris Graduate Fellowship program, all of my colleagues from Dr. Christopher's research group especially Rifat M. Muied and William M. Boler for all of the feedback, suggestions and support they have provided me.

Finally, I would like to express my sincere gratitude to my parents for all of the encouragement and support throughout this process. I would like to thank my friends and family members for their wishes and hopes.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABBREVIATIONS

CNN	Convolutional Neural Net
NN	Neural Net
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HLS	High Level Synthesis
GPU	Graphics Processing Unit
CPU	Central Processing Unit
SOM	Self Organizing Maps
ReLU	Rectified Linear Unit
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuits
HDL	Hardware Description Language
DSP	Digital Signal Processing
BRAM	Block Random Access Memory

ABSTRACT

Rahman, Tanvir. M.S.E.C.E., Purdue University, December 2017. Classification of Road Side Material using Convolutional Neural Network and a Proposed Implementation of the Network through ZedBoard Zynq 7000 FPGA. Major Professor: Lauren Christopher.

In recent years, Convolutional Neural Networks (CNNs) have become the state-of-the-art method for object detection and classification in the field of machine learning and artificial intelligence. In contrast to a fully connected network, each neuron of a convolutional layer of a CNN is connected to fewer selected neurons from the previous layers and kernels of a CNN share same weights and biases across the same input layer dimension. These features allow CNN architectures to have fewer parameters which in turn reduces calculation complexity and allows the network to be implemented in low power hardware. The accuracy of a CNN depends mostly on the number of images used to train the network, which requires a hundred thousand to a million images. Therefore, a reduced training alternative called transfer learning is used, which takes advantage of features from a pre-trained network and applies these features to the new problem of interest. This research has successfully developed a new CNN based on the pre-trained CIFAR-10 network and has used transfer learning on a new problem to classify road edges. Two network sizes were tested: 32 and 16 Neuron inputs with 239 labeled Google street view images on a single CPU. The result of the training gives 52.8% and 35.2% accuracy respectively for 250 test images. In the second part of the research, High Level Synthesis (HLS) hardware model of the network with 16 Neuron inputs is created for the Zynq 7000 FPGA. The resulting circuit has 34% average FPGA utilization and 2.47 Watt power consumption. Recommendations to improve the classification accuracy with deeper network and ways to fit the improved network on the FPGA are also mentioned at the end of the work.

1. INTRODUCTION

Deep Neural networks have recently emerged as a cutting edge research approach for object recognition in the field of computer vision and machine learning. Deep neural networks bear a resemblance to the mammal's visual cortex [1] which is by far the most powerful biological visual processing system. This visual system first captures small size information through small sub-regions called the receptive fields. In the next stage, complex analysis of this locally captured information is performed by the larger cells, called Neurons, to learn and identify objects. Artificial neural networks work in a similar way, first analyzing simple patterns (lines, curves etc.) from an image in the first layer and then combining simple patterns to identify complex shapes or features in subsequent layers to identify objects. Convolutional Neural Networks (CNNs) are now common in image recognition. CNNs are hierarchical neural nets where parameters of convolution layers alternate with subsampling layers [2] providing flexibility and better accuracy for classification problems.

1.1 OverView and Problem Statement

In recent years, lot of research and development had been done in the field of autonomous driving and vehicle safety. Safety has always been a primary focus point for the automotive industry. In United States crashes due to road departure cover a major portion of total yearly crashes. Lots of safety systems have been developed through advanced research. Most of the systems are based on standalone lane departure system which rely heavily on lane markings, weather and lighting conditions [3] [4] [5]. As a result, these systems are not always helpful in case the road edge markings are faded, or no markings are present at all, or if it is difficult to see lane/road edge markings due to lighting and weather conditions. This is a real concern in the U.S.

because people live in more distributed areas when compared to Asian and European countries. Many people live in suburban areas and commute daily between their home and work places. Such unmarked road conditions are present in these areas. Consequently, current lane departure detection systems would fail to provide timely warning for drivers. An advanced and more efficient road departure system needs to be developed for this kind of situation. Advancement in machine learning promises a lot in this field of interest. A classification based on different types of road edges can be incorporated in next generation road departure prevention or warning system. Such type of classification would assist any next generation smart vehicle to differentiate between on road and off road position. It will also provide a part of a strong foundation for developing automated driving systems. This research work focuses on creating a CNN based classifier to classify road edgers with improved accuracy. CNNs are complex system and can be computationally intense. For a real time operation of such classification system, a hardware model needs to be developed. Since finding the best classification result needs high speeds and low power, Field Programmable Gate Arrays (FPGAs) provides not only the flexibility but also lower power consumption compared to the Graphical Processing Units (GPUs). The second part of this research shows the design of the Neural Net, then high level synthesis (HLS) and then to the bitstream generation with the utilization of the resources available in Xilinx Zynq 7000 FPGA. Because of this accomplished work, the hardware model of the Neural Network can be downloaded to the FPGA and can be accessed from an embedded Advanced RISC Machines (ARM). This shows successful hardware acceleration of the CNN operation. Ultimately, this research can contribute to the future advanced road departure systems which can significantly reduce vehicle road departure incidents and personal injuries and fatalities.

1.2 Literature Review

1.2.1 Neural Network

A Neural Network is a special type of computer system [6], which is inspired by the structure of mammals brain. A cell in the human brain known as the neuron has two major components: Axons and Dendrites. These neurons are electrically excitable. Based on the signal received through the Dendrites, Axons perform the Neural coding [7] and excite the stimuli which is then distributed through the Neurons and can be considered as output of the Nervous system. Axon from one Neuron is connected to Dendrites from another Neurons and this way forms the Neural Network in the human brain (Fig. 1.1).

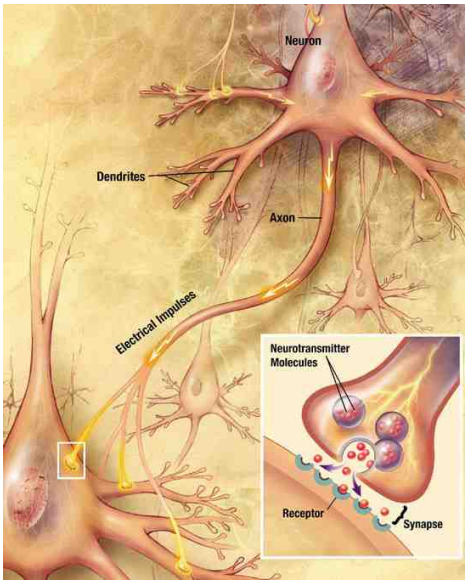

Fig. 1.1. Signal Propagating through the connector from Axon to the Dendrites of next Neuron [8].

Advancement in computer science has made it possible to develop a Connectionist system, which models the biological neurons as nodes [9] and acts through the interaction in between network components [10] using weights, biases and activation functions. The specialty of Neural networks is that they are able to perform tasks

without being programmed specifically, and are able to improve the performance through a learning process. In terms of machine learning, the nodes resemble neurons in the brain. A neural network is composed of several layers of nodes and these nodes are connected in layers. Layers are divided into input layers, output layers and intermediate layers (known as hidden layers). The output of each node is called the activation, which is triggered through an activation function. Each of the nodes is associated with specific values for weights and biases. The weights and biases determine the output of each node. Training a network updates the weight and bias values in each iteration.

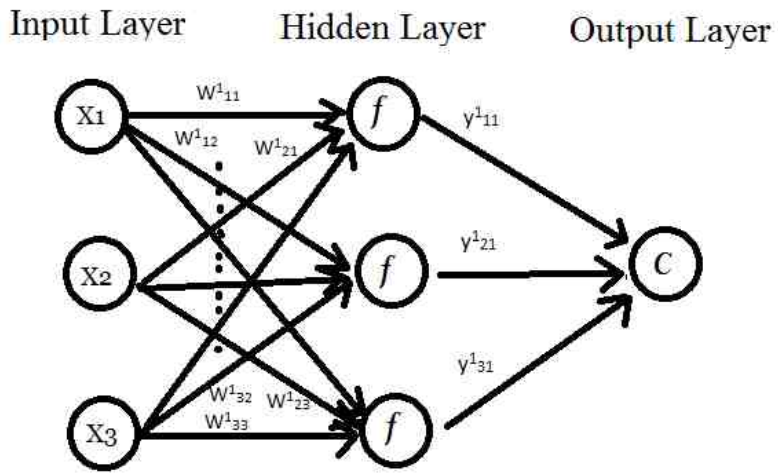

Fig. 1.2. Feedforward Propagation of a Single Layer Perceptron Neural Network.

A graphical description of a simple feedforward neural network is shown as the Single-Layer Perceptron Neural Network in Fig. 1.2. The term Feedforward is used to indicate the dataflow, and is in one direction. The first layer which takes the information from the input is known as input layer. Nodes associated with the input layer have weights. In this simplest form of network, there is one multiplication and one addition per node. Once the input is passed through the input layers, through weights and biases, each input node then calculates the values which are then passed to the hidden layers. There are functions associated with the hidden layers which

then generate the output and pass the value to the final layer. The final layer is the output layer which calculates every score from the previous layer. For a classification problem, there is class score, and the maximum value of the class score is passed through the output layer to result in the final classification. This entire process of data-flow from input node to output node is known as forward propagation. Each output node can be described with the general form of equation 1.1 where i denotes the i^{th} node on the corresponding layer. Weights are expressed through w^i_{mn} and biases are expressed through b^i_m where, m corresponds to the m^{th} neuron from the previous layer and n corresponds to the n^{th} neuron from current layer.

$$y^i_{mn} = f(\sum_{i=1}^{n} w^i_{mn} x^i + b^i_m) \tag{1.1}$$

In this equation, the output node C gets the value for y^1_{11}, y^1_{21} and y^1_{31} and choses the maximum score. Equation 1.2, 1.3 and 1.4 shows the forms for y^1_{11}, y^1_{21} and y^1_{31} respectively.

$$y^1_{11} = f(w^1_{11}x_1 + w^1_{21}x_2 + w^1_{31}x_3 + b^1_1) \tag{1.2}$$

$$y^1_{21} = f(w^1_{12}x_1 + w^1_{22}x_2 + w^1_{32}x_3 + b^1_2) \tag{1.3}$$

$$y^1_{31} = f(w^1_{13}x_1 + w^1_{23}x_2 + w^1_{33}x_3 + b^1_3) \tag{1.4}$$

A system with limited learning capabilities was developed in the work of Lugar and Stubblefield [10], which was known as perceptrons. These can perceive a value (0 or 1) based on specific threshold set to each output node. Perceptrons are trained iteratively through a set of training data. The training data set contains list of input values and their associated true output values. At the beginning, input values are passed through the input layers and corresponding output values are compared to the true output. If the value computed is '0' where it should be '1' then weights and threshold are adjusted to better match the true result. Similar, but opposite, work is done when the output comes as '1' where it should be'0'. Fig 1.3 shows the approach.

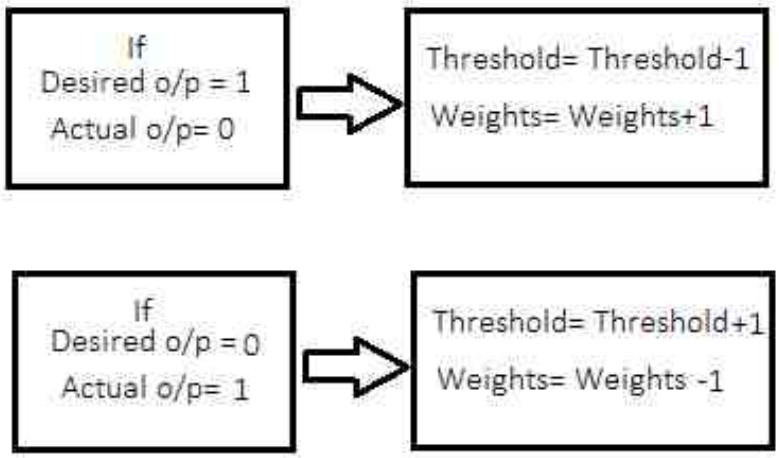

Fig. 1.3. Earlier simple approach to train the perceptron weights to improve the network performance.

After the perceptron the research evolved to use the multilayer network. This is structured and the training was done using the backpropagation algorithm [11]. The backpropagation algorithm follows a delta rule [12] to calculate the error at the output node. The error calculated at output node is propagated backwards through the network after each training phase and updates the weights based on this error function. This type of learning is known as gradient descent learning. As mentioned earlier, the output of a node depends on activation function, and the most commonly used activation function in backpropagation networks is the sigmoid function (equation 1.5) where, $f_x = \sum_{i=1}^{n} w_{mn}^{i} x^i$.

$$\sigma(f_x) = \frac{1}{1 + e^{-f_x}} \tag{1.5}$$

In gradient descent, the derivative of sigmoid function needs to be minimized in order to reduce the error. The sigmoid function applies to the all nodes except the input node and the values are limited in [0,1].

1.2.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a special type of neural networks. The basic working principle for CNNs and the ordinary neural networks described in the previous section remains the same: taking the input at the input layer, passing the input value through some hidden layers, performing dot operation (multiplication and addition) through the associated weights and biases for each node, and generating output values through activation function. Scores at the output nodes are computed and through the back-propagation principle, network parameters are updated after each iteration of the training data. In reality, CNNs are harder to train than other shallow neural networks due to increased complexity. What makes the CNNs so special to the computer vision research? We will look into the broader picture now.

Regular neural nets do not scale well for image inputs. For example if we have a fully connected network which takes 32x32 RGB image as input, then that needs to have 32*32*3= 3072 weights for the first input layer (as for a fully connected network every neurons in the previous layer must be connected to every neuron in the next layer). This number seems reasonable for the small image, but in reality we will have much larger images. So for an image input of size 600x300, we would need to have 600*300*3 = 540000 weights for the first layer of neurons only! CNNs explicitly assume that the input is an image. In 2012, CNNs started to create a striking influence in the field of image classification when Alex Krizhevsky won the Imagenet competition, in which his team created a CNN that reduced the error in classification from 26.2% to 15.3% [13]. CNNs take advantage of any input structure that exhibits spatial correlation, such as images, so the CNN arranges neurons into spatial dimensions. This spatial arrangement allows the efficient flow of information through the network and extensively reduces the number of network parameters. In other words: CNN structure makes use of spatially related information. By following this principle, CNNs can identify patterns and classify them with greater accuracy and efficiency. The basic architecture of a CNN consists of few different parts, mainly

Convolutional Layer, Pooling Layer, Rectified Linear Unit or ReLU Layer and Fully Connected Layer. A very basic convolutional network structure known as LeNet-5 [14] for document recognition is shown in Fig. 1.4.

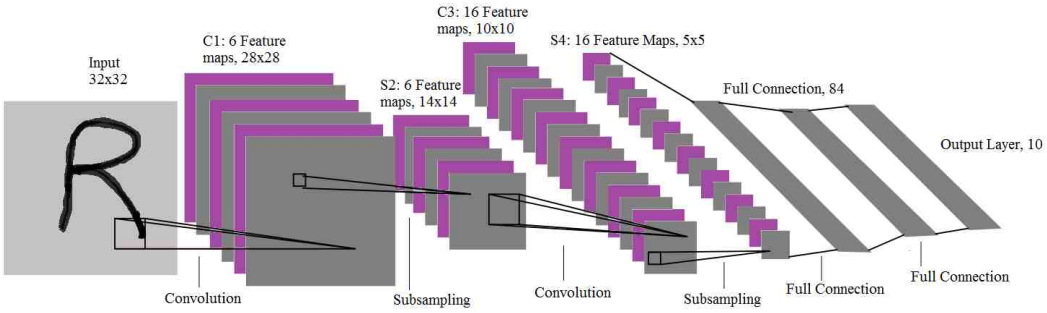

Fig. 1.4. Architecture of LeNet-5, a convolutional neural network for digit recognition [14].

Convolutional layer

The convolutional layer is at the heart of a convolutional network. It is the mathematical part that performs the maximum portion of the computation. Each convolutional layer consists of a set of learnable parameters known as filters or kernels. Each kernel has specific dimension, with associated weights and bias values. An RGB image basically consists of blocks of pixel values. During forward propagation (performing a convolution), each kernel slides through every block of input image pixels, gets activated, and performs 2D dot product operation on the pixel blocks covered by the kernel dimension. Each of the kernels tries to extract different features from the input image. Once the sliding is done for every region of image, the result is known as an activation map. Each kernel corresponds to one activation map. Each activation map has a smaller dimension than the input image. This activation map will be input for the next layer after some other operations performed. Every operation and convolution reduces the dimension of the input. But the network grows

in depth as evident from Fig. 1.4. Three hyper-parameters define the output volume of one convolutional layer. **Depth** is the first hyper-parameter which defines the output volume by the number of kernels or filters present in the convolution layer. Depth of output volume is proportional to the number of kernels. The next hyper-parameter is **stride** which controls the kernel step size. If the stride is set to 1 then the filter will slide one pixel value for every operation. In this way stride controls the output spatial dimension. The third hyper-parameter is **zero-padding**. Zero padding also controls the spatial dimension by adding zeros to the border of input.

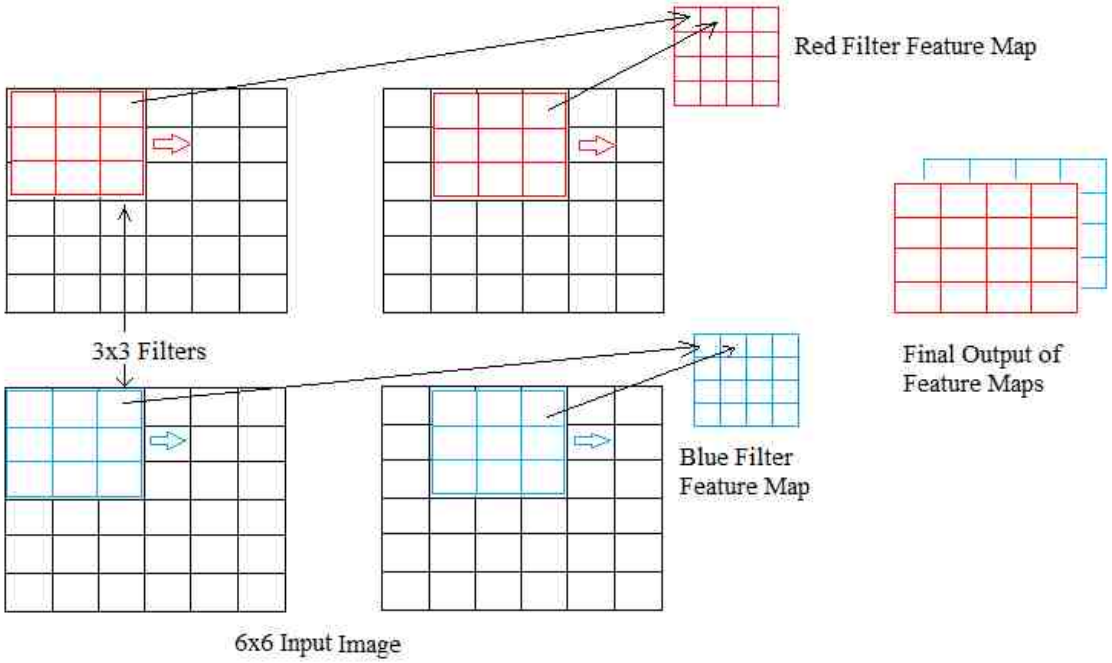

Fig. 1.5. Sliding of filters over input image to generate 2D feature map

To understand the operation of convolutional layer and its output, let us first consider a simple case illustrated in Fig 1.5 where we have an input image of 6x6 and two kernels of size 3x3. Two kernels are marked as red and blue. Stride is selected 1 for this example with no zero padding. So each kernel will slide horizontally through every 3x3 pixel blocks till it reaches the rightmost border and then repeats this step

from the second horizontal pixel line. When the operation is completed, a 4x4 feature map is generated for every kernel. So the final output from the convolutional layer will be two 2D feature maps. So the operation here reduces the input spatial dimension from 6x6 to 4x4, however, it increases the volume or depth to 2. So the final output will be of the form 4x4x2. So for a $N \times N$ image with Y number of $M \times M$ kernel with 1 stride and no zero padding the output dimension will be $(N-M+1) \times (N-M+1) \times Y$.

Pooling Layer

Another important layer known as Pooling Layer often used in between two convolutional layers. Pooling layer is used to reduce the spatial dimension of the input layer. Most common type of pooling layer is Max pooling layer. As the layer is reducing the dimension of the input layer, this layer is also known as downsampling layer. Most common Max pooling filter is of 2x2 dimension with stride of 2 pixels. Once

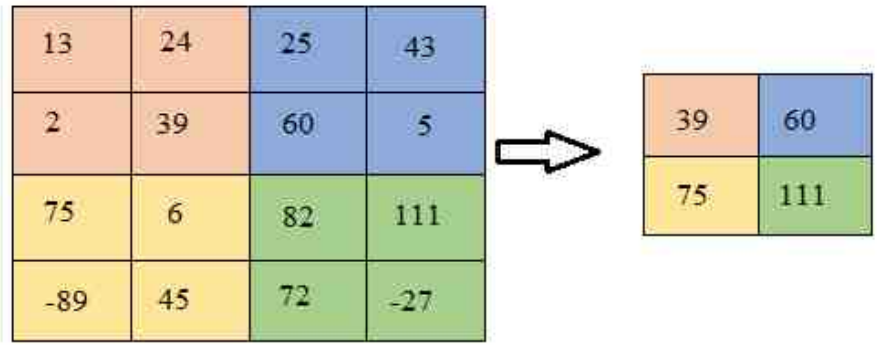

Fig. 1.6. Example of a Max Pooling Layer with 2x2 filter with stride of 2 pixels.

gliding through the input, it takes the maximum value in the 2x2 region and moves to the next 2x2 region after the 2 pixel stride. Continuing this process throughout the input dimension Max Pooling filter generates the reduced spatial dimension output. The example shown in Fig. 1.6 reduces the dimension and number of parameters to 75% (from a 4x4 dimension down to 2x2).

A new practice is prevalent recently that does not include a pooling layer in between convolution layers. The idea behind not using the pooling layer is backed by using higher number of strides in the convolution layer to reduce the spatial dimension and increase the simplicity [15].

ReLU (Rectified Linear Unit) Layer

Rectified Linear Unit or ReLU layer is normally used after the convolution layer and before the max pooling layer. This layer is used to introduce non linearity in the network. As described in previous section, nonlinear activation functions were used in earlier days. Later it was proved that ReLU layers provide a much better network performance and improves the computational efficiency by reducing the vanishing gradient problem [16]. The ReLu layer doesn't change the spacial dimension of the input. In the example shown in Fig 1.7, the layer uses the function $f(x) = max(0, x)$ to change all its negative values to 0.

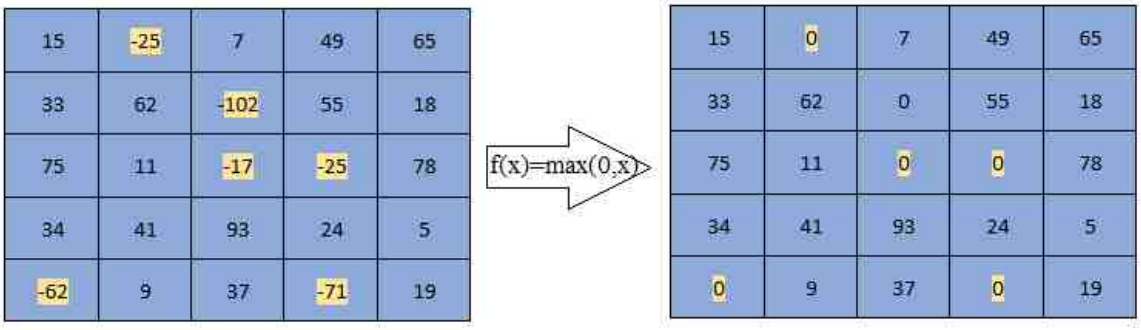

Fig. 1.7. Introducing nonlinearity in the network by changing all negative values to 0.

Fully Connected Layer

A fully connected layer is the final output layer of a CNN. In a fully connected layer every neuron in previous layer is connected to the every neuron in the current layer which is the same as described in the previous section in a regular fully connected network. A fully connected layer converts the feature maps from previous layer into the classification scores.

1.2.3 Transfer Learning

As the main goal of any classification problem is to design a network from a chosen model that can classify with the best possible accuracy, this depends on the type of training data used. In a real life scenario, variations in types of training data are huge. It is very difficult to build a neural network from scratch, moreover, as complex networks provide better accuracy, a robust hardware configuration is required to train such complex networks. Alexnet [13] used 1.2 million labeled images to train on two GTX 580 3GB GPUs and took six days to train the network and classify 1000 categories. A simpler NN alternative is to use transfer learning. Existing trained CNNs can be re-used. For transfer learning, We do not need thousands of classifications in our problem. Transfer learning takes an already existing network, modifies it to fit into the problem of interest, and partially retrains the network with images specific to the new problem. The retraining part is not as intense as the first training. This approach has been producing excellent results in new classification problems. Fig. 1.8 shows the work flow for the transfer learning process.

The definition of transfer learning closely follows the work by Pan and Yang [17] applied to document classification. We need to introduce here the domain and task concepts. Let us consider a domain D with χ feature space. The domain has marginal probability distribution $P(X)$, where, $X = x_1, x_2, ..., x_n \in \chi$. If the task is document classification, then χ is the space for all document representations, x_i is the i -th term in the vector representing some documents and X is the document sample used for

training. This task classifies a document with bag-of-words representation as a type of binary feature that is either $true$ or $false$. Given a domain, $D = \{\chi, P(X)\}$, a task T consists of label space Y and conditional probability distribution $P(Y|X)$ learned from the training data set containing pairs (x_i, y_i) where, $x_i \in X$ and $y_i \in Y$ i.e. $T = \{Y, P(Y|X)\}$ For this binary classification task Y is the set of all labels consists of $true, false$ and y_i is either $true$ or $false$. So, the source domain is denoted as D_s with source task T_s. So for a target domain D_t with target task T_t, the transfer learning objective is to learn conditional probability distribution $P(Y_t|X_t)$ in D_t with knowledge learned from D_s and T_s given that $D_s \neq D_t$ and $T_s \neq T_t$.

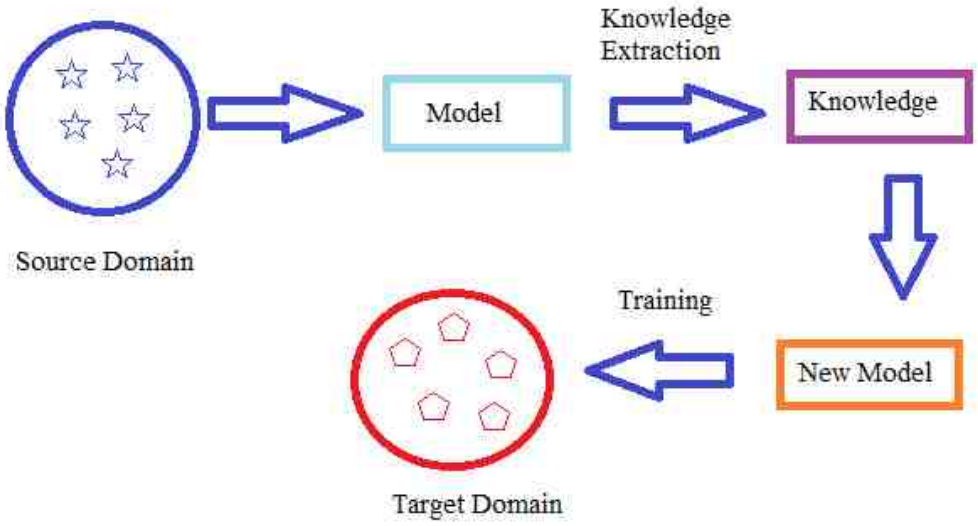

Fig. 1.8. Transfer Learning work-flow.

Four conditions can arise.

1. $\chi_s \neq \chi_t$, source and target domain feature spaces are different. In case of document classification problem, they can be of two different languages.

2. $P(X_s) \neq P(X_t)$, source and target domain marginal probability distributions are different. For example two documents focus on different topics.

3. $Y_s \neq Y_t$, label spaces are different between two tasks i.e. source and target documents have different labels.

4. $P(Y_s|X_s) \neq P(Y_t|X_t)$, source and target tasks conditional probability distributions are different, i.e. source and target documents are unbalanced given user-defined classes.

Knowing these conditions and relating them with the given target domain with the specific target task, it is now possible to define what type of knowledge transfer needs to be selected from which source domain. For our road-edge classification, this will be discussed in later sections.

This completes the introduction of the Convolutional Neural networks used in this project. Next is an introduction to the Field Programmable Gate array that will be used to implement the new CNN for road-edge classification.

1.2.4 FPGA Overview

Field Programmable Gate Arrays are integrated circuits designed in such a way that they can be electrically programmed by the customer or user after being manufactured. FPGA consists of arrays of programmable logic blocks. These blocks are connected through programmable hierarchy of interconnects. These interconnects can be programmed to define which way the logic blocks will be connected and thus data flow can be reconfigured. Logic blocks can be programmed to perform various logic operations. This reconfigurability property makes FPGAs different than Application Specific Integrated Circuits (ASICs) which are custom designed for specific functionality. Hardware Description Language (HDL) (i.e. VHDL or Verilog) is used to program and model FPGAs. Once programmed, the functionality can be verified using program compiler and then a file to configure the function into the hardware is created known as bitstream file. The bitstream file contains the wiring information for the FPGA components. This file is then downloaded into the FPGA board and when powered on, the board is configured to run according to the designed function-

ality. If the performance verification failed to meet desired criteria, using the HDL compiler existing functionality can be modified and by regenerating the bistream file the program can be tested again. This reconfigurable property makes FPGAs very popular choice among application specific researchers to build and test circuitry prototypes. As FPGAs can be reconfigured, it allows researcher to test and run custom functionality without developing physical program specific devices. This saves a lot of development time. In the field of Machine learning, FPGAs contribute by accelerating the computational part of the network [18] [19]. Parallel processing allows FPGAs to execute multiple logic programs at the shortest possible time which as a result reduces the overall throughput. Unlike GPUs, FPGAs consume a smaller amount of power, and for this reason, FPGAs have become a popular choice for the real time operation of classification process.

This research work employs a Xilinx Xynq 7000 series FPGA, which combines the embedded software programmability of an ARM processor with hardware programmability of FPGA. The heart of the processing system is a dual core ARM cortex A9 processor and the FPGA is based on 28 nm fabrication technology. Processing system includes on chip memory, DMA controller, external memory access interface and I/O peripheral interfaces. The programmable logic part consists of several Block Random Access Memories (BRAMs), Digital Signal Processing (DSP) clocks, programmable I/O blocks, Configurable logic blocks, high speed transceivers and Analog to Digital converters. There are AXI interconnects that for connecting processing system, on chip memory, I/O peripherals and programmable logic. Finally the system has 8 clock management tiles that generates wide range of frequencies for different programming modes. This architecture will be used to create a lower power and faster implementation of the chosen CNN for the new road-edge classification.

This introduction has provided the motivation and background for the next sections. The following sections provide the specific CNN algorithmic architecture and FPGA implementation of this architecture, which is the new contribution described in this research.

2. CLUSTERING WORK ON FLYOVER IMAGES

As the goal of this research work is to create an improved classifier for road edges. We discuss first a previously developed standard neural net clustering network for Toyota's collaborative safety research program to classify road edges. The clustering research explored which feature could be extracted from an image input and later could be used to facilitate road edge classification. As an image source, satellite view road-edge images and street view road-edge images from Google were obtained. These images were from specific random GPS locations, from 44000 GPS locations. Although it was expected that all these 44000 GPS locations would have both street view and satellite images, in reality, among those 44000 GPS locations, only around 25000 of them actually had the street view images. So, top view images were used to develop the first classifier due to the completeness of the source data.

Once the input source was selected as the satellite images (top view) images, then next a road-edge patch was selected from the image, because the whole image contains a lot of information and lot of features that would not apply to the road edges. Therefore, patch that contains most of the information of the road edges is extracted from the complete image. From the patch we calculate a similarity score which will is used to cluster the patches with similar score. Finally, the clusters will be mapped back to the human labeled names to check the accuracy of the work.

For two image patches, the similarity is measured by a moving window block of selected dimension and comparing the result score for the images. When the patches are similar, the score is higher, and very different image patches will score near zero. We selected color patch feature and need with a measure of color that is less coupled to the Luminance (brightness) value. Fortunately there are color spaces that can partially decouple the luminance. Therefore we measured the color of the road edge patch using the Cr and Cb color space (commonly used in image compression and

transmission). To test this, we hand-selected 4 grassy Google Earth images, and their respective 4 Google Street view images, and cut a 50x50 pixel patch of road edge grass from each of these images.

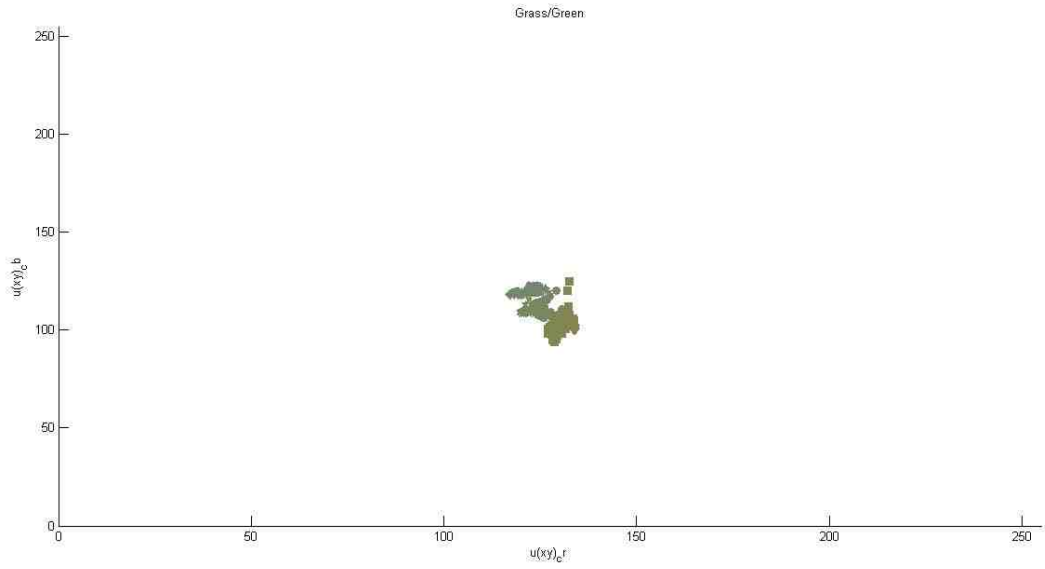

Fig. 2.1. 800 grass color values plotted by Cr and Cb value.

We took the mean of each 5x5 pixel and plotted these on the scatter plot shown in fig. 2.1, where Cr is in the horizontal axis, and Cb is in the vertical axis. These 800 points are plotted in Fig 2.1. The color of the marker represents the average color for that 5x5 patch, with all luminance set at mid-value. The hypothesis behind that is, a color vector or color patch can be a good discriminator for road edge material. As we are not using complete image for the process and we have a fairly large amount of image data, we need preprocessing of images to separate the patches of interest with maximum possible automation.

The extracted photographic data from Top View goes through the automation diagrammed below (Fig. 2.2). Some preprocessing is used to prepare the data for automated clustering, with some of the data needing a road lane edge marked by

human, if automation for this task was not possible. The automation provided the clusters of the image patches based on a sample data set, and then similarity was used to assign the remaining data to the clusters. Finally, the automation clusters are mapped back to the human classifications

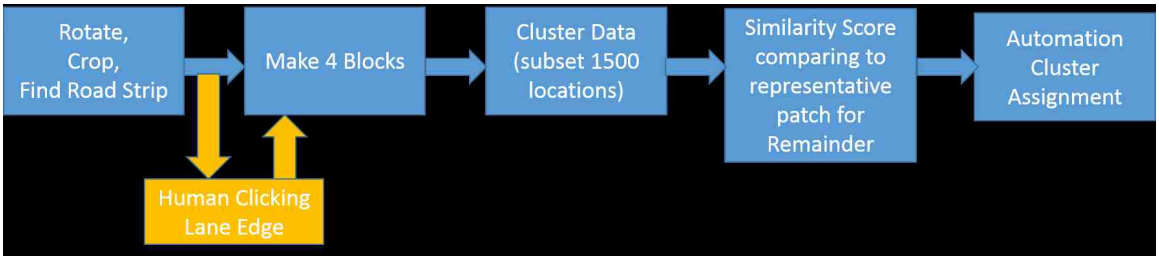

Fig. 2.2. Process of automated clustering and similarity assignment.

Automated image rotation and centering are done on the top view images. The road and road edge patch selection of data is automated. When there are no strong parallel lines found in the automation, the image is separated for manual human marking of the road edge (yellow path in Fig. 2.2)

The automation goal is to extract image snips from the Top View data that represents the roads edges. This first means finding some lines in the image. Hough lines are first used in Top View to rotate the image. Fig 2.3 shows the top view typical input and region of interest (red box, manually drawn) and resulting strip of road, 50 pixels high. The automation uses Hough lines (dominant high frequency straight lines at road edges) in the original color image to rotate, then to find the strip location and width. Finally, in Fig. 2.4, 4 images of 5050 pixel blocks are produced (for 2 lane roads or larger) as, Side Left (SL), Road Left (RL), Road Right (RR), and Side Right (SR). In these data, 25 pixels corresponds to one standard lane width (one benefit of this Top View data is a consistent 15cm per pixel resolution).

For this research, out of the 4 separated parts/blocks, two blocks SL and SR were taken into consideration. The next step is to use 3000 random samples of these blocks in a clustering algorithm to automatically generate self-similar groupings of

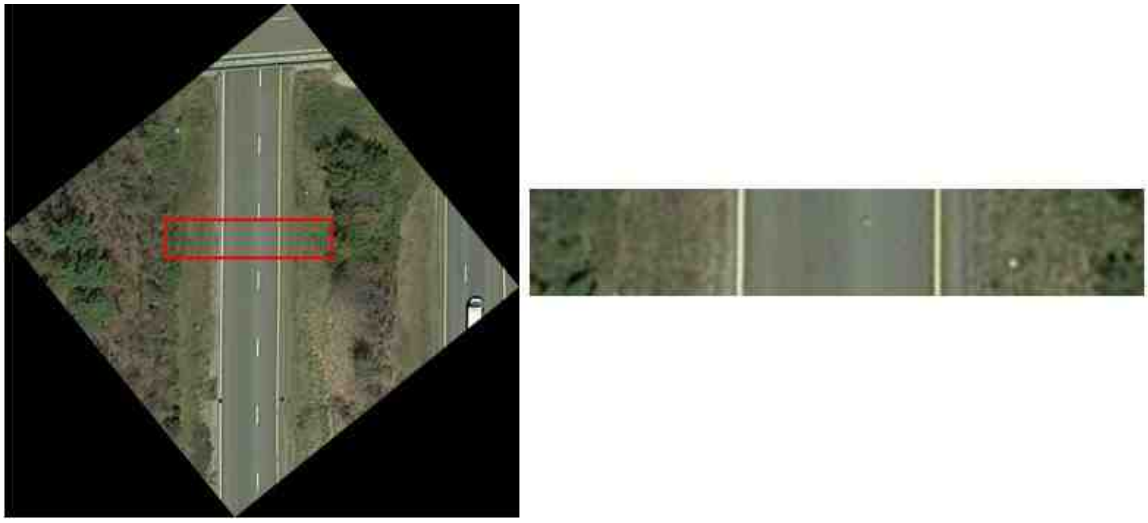

Fig. 2.3. View of GPS location after rotation and centering, and strip of the road taken from center.

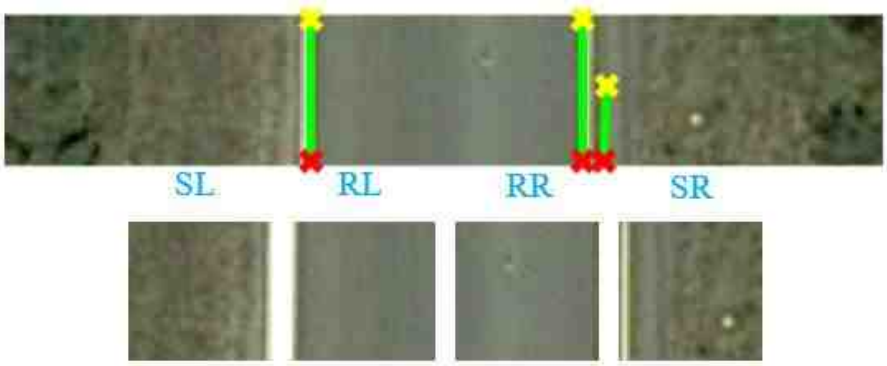

Fig. 2.4. Hough Lines on Strip (top), 4 images/blocks broken out (bottom).

this data. To do this we use the self-organizing map (SOM) tool, based on Neural Net Machine Learning, in MATLAB 2016a. Several experiments were run on this clustering, varying the number of clusters and varying the features that were input to the clustering. A visualization of the clustered data is shown in Fig. 2.5. The map tries to optimize large distance between the feature data, thereby automatically grouping the results. In Fig. 2.5, the blue hexagons represent each of the 25 neurons (25 clusters) with the number of test data associated with each cluster.

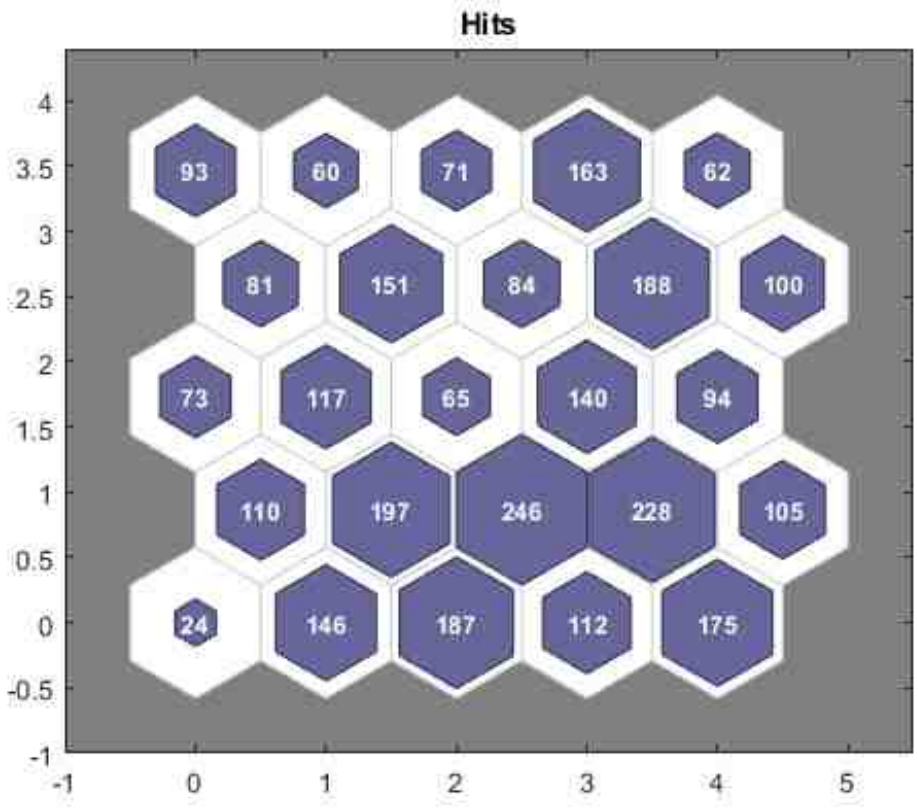

Fig. 2.5. SOM visualization of 3000 blocks of image data separated into 25 clusters.

In addition, a distance connectivity plot is shown in Fig. 2.6. This allows analysis of the uniqueness of the clusters, where a dark connection between hexagons implies that the two connected clusters are substantially different. Dark is better, indicating clusters are substantially different, and lighter yellow implies these clusters might be similar (and could be merged later).

The result was judged by the smoothness and similarity of the data in each cluster. Our experiments showed that 25 clusters with the color side image patch data ($50 pixels \times 50 pixels \times 3$ colors integers) provided the best result. To improve some of the final assignment accuracy (for mapping to human labels) results, five more

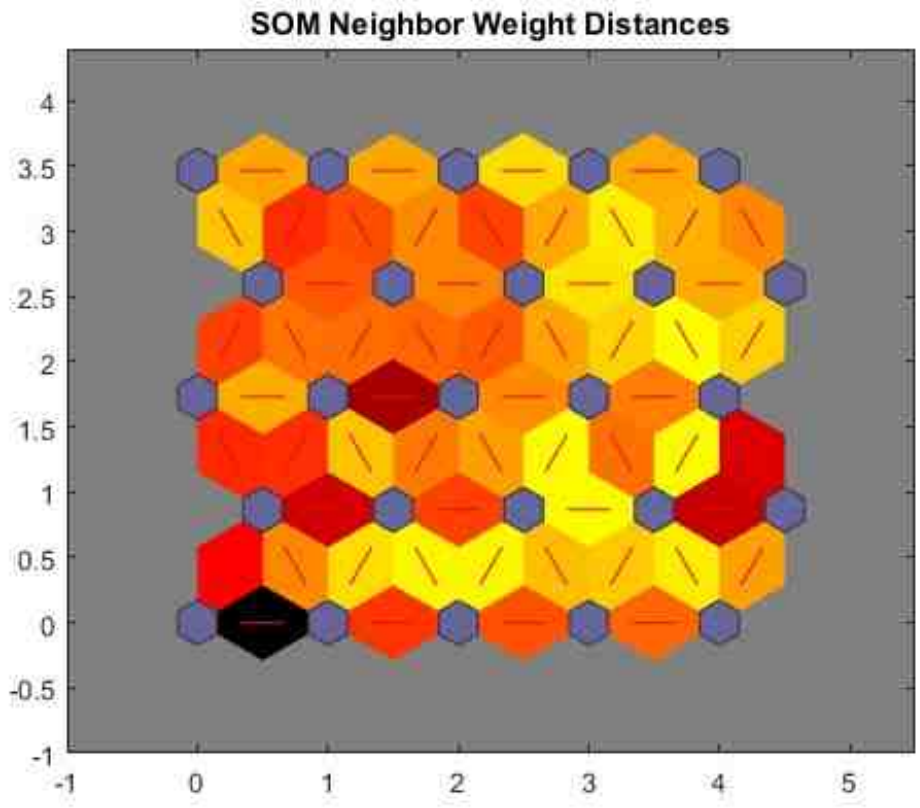

Fig. 2.6. Weights between the clusters in Fig 2.5.

patches were added. These additional patches eliminated some errors. Fig. 2.7 shows the different clusters, one patch each for side clusters.

A representative patch from each of the clusters from the training data was used to compare to each of the remaining data. As we have two patches for the two road side data (SR & SL), the patches were mirrored to make the comparisons. These comparisons are done using the retrieveImages MATLAB function. The queryImage used is each representative patch from the cluster data, and a threshold is set to gather the remaining similar images. There is a score based on color similarity and a score based on edge similarity. The edge data uses Bag of Features – SURF based, so this is scale and translation invariant. The highest scores are used to select the cluster to

Fig. 2.7. The 30 reference patches for side.

which a particular patch belongs. If the score fell below 20% of the maximum against all of the given cluster patches, the result is labelled other, so the final number of clusters is 31 for road-side at the end of this step. An example of some data is shown in Fig. 2.8. The automation is not perfect, but a majority of the patches are self-similar. For each location, the automated clusters are then grouped into the human labels. Therefore, a manual mapping of automated clusters was made to connect to the closest human label from the GUI result (Table 2.1).

Additional patches were added for some error improvement (patches 26-31). After this adjustments, initially we found about 33% of the data automatically labeled for road-side matched the human label in a random sample of 255 locations. Confusion matrix for the initial result is shown in Fig. 2.9.

Fig. 2.8. Cluster grouping example: Cluster 1 (other), 4 (tree), and 8(dirt); 20 similar patches.

Table 2.1.
The mapping of automated clusters to the human labels.

Side Material Types	Human Labels
Grass Green	2,4,6,7,8,9,10,11,13,14,15,16,17,19,21,26,27,29
Grass yellow/gray	18 (combined with other grass)
Asphalt	5, 22, 23, 30 (typically in parking lots)
Concrete	1
Dirt	24, 28
Barrier	12
Tree	25
Other	3, 20, 31

With this initial result, merging of materials (i.e. combining different types of grases) and human revision was done to improve the accuracy (59%) of the automatically labeled data mapped to human labeling.

	Grass green	Grass yellow	Dirt	Barrier	Tree	Concrete	Other	Asphalt	
Grass green	63	4	2	1	0	0	13	1	
Grass yellow	39	0	2	4	0	3	1	5	
Dirt	19	0	4	2	0	0	0	1	
Barrier	2	0	1	6	0	0	0	1	
Tree	20	0	3	0	3	0	8	0	
Concrete	1	0	3	1	0	1	0	0	
Other	2	0	1	0	0	0	1	3	
Asphalt	14	0	4	2	0	2	3	6	
									33%

Fig. 2.9. Confusion matrix for the Automatic Clustering vs Human cluster labeling of the road side materials.

3. WORK ON STREET VIEW IMAGES

Unsupervised clustering work done on the top view images gives us 33% initial accuracy. Although the accuracy was improved through human revision of patch labels, this result was still not accurate enough. The major problems with this previous method are discussed in the next paragraphs.

Size restriction exists for the input image for this model. In the work, we selected $50 \times 50 \times 3$ image patches for input. This image input was then converted to a vector of R, G, B values for SOM operation. As we expect, increasing the input image patch size would be able to feed more color feature information to train the network, but this increases the vector dimension and also increases the number of fully connected nodes in the SOM process. This in turn would increase the computation time and complexity.

Low resolution of patches is the second concern for the work. As we used top view images for the source of patches which limited the highest possible resolution available from Google. There is a 15cm per pixel limit to the image data. Because the data was taken from the air, the color gamut and resolution is not as good as the photographic street view data. This definitely affects the network performance. Fig. 3.1 illustrates the situation.

The lower resolution of patches forms another problem: **erroneous differentiation** between types of classes that has visibly similar color patterns. This creates problem in identifying the different types even for the human eyes. It was found that two different humans identify the same image patch as different material. This as a result lowers the accuracy of the network when patches clustered through automation mapped back to human mapping. An example of similar looking patches is shown in Fig. 3.2.

Fig. 3.1. High resolution satellite image (left) is cropped (red box) to get area of interest surrounding the center GPS location (right).

Fig. 3.2. Four different types of road side material in four different patches. They have visually similar color appearance.

These issues suggested we should select a new approach to classify road edges for this research work. We selected Convolutional Neural Nets (CNN), and our methods are described in the next section.

3.0.1 Conv Net & CIFAR-10

Convolution Neural Network (CNN) algorithms are most used in computer vision related problems. CNNs produce excellent results [13] and exhibit lot of flexibility and reusability. CNNs have been used to identify number (digit) handwriting in a document image [14]. As mentioned earlier, CNNs are excellent choice for object recognition and classification. CNNs take the input image has inherent flexibility in image dimensions.

As mentioned in the previous chapter, for the clustering work done on top view images, the input size was restricted and patches do not contain much color feature information due to the lower resolution. Street view images have much better resolution and contain a lot of desired color feature information. So to improve the classification accuracy with better flexibility, we developed a new CNN based classification for the road edge problem using street view images. 25000 GPS locations among the 44000 total set have street view images. These 25000 locations contain enough of each selected road side material types to provide sufficient variability and information to train the CNN. Moreover, the rigorous steps involved in preprocessing the top view images do not apply to street view images because of the approach selected to be explained later. However, a small sorting process is first applied to the images, because each available GPS location has 8 images provided for different angular camera positions $(0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°)$. Human guided sorting process was used to select the best camera angle for viewing road edges.

Once we have selected the type of input to be used for this step of research, the next step is to select the CNN structure to use. Another advantage of CNNs over regular neural nets is the transfer learning feature which uses an already existing network that is close in features to our problem of interest. In this way, we don't have to build a network from scratch, which would require few hundred thousand images to train the network. Although 25000 images sounds sufficient enough to train the network, still labeling these images would require enormous amount of human work.

There are a number of choices available for the transfer learning network selection. First is a successful CNN: LeNet-5 [14] used to identify digits in handwriting. Another is Alexnet [13] which was the first CNN used for large data image classification. Compared to LeNet, Alexnet has a much deeper network and more parameters. The recent ZF net [20] won the 2013 imagenet contest with an improvement through the architectural modification of Alexnet. GoogleNet [21] won 2014 contest with the structure introducing the concept of inception layers, which reduced the number of parameters significantly compared to Alexnet; from 60 millions to 4 million. Another recent one is VGGNet [22] which was the runner up in 2014 imagenet contest, it showed that network depth is a critical component for overall performance of the network. Although VGGNet uses a significantly higher number of parameters; 140 million, it has proved that the reducing parameters in first fully connected layer does not affect the network performance and hence can reduce the number of parameters significantly.

All of these CNN structures showed excellent accuracy and performance for these applications. However, as the final goal of this research work is to implement the network into the FPGA, we need to select a smaller network that has acceptably better performance due to the limited FPGA resources. We chose the CIFAR-10 [23]. This is a CNN network developed by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. This is an established dataset used for object recognition in images. The training dataset consists of 60000 $32 \times 32 \times 3$ tiny images and the network classifies 10 objects. There are 6000 training images for each category. In their work, 50000 images are used to train and 10000 images are used to test the network. Fig. 3.3 shows few sample images in the dataset used for training the network and table 3.1 includes the 10 classes.

These classes are mutually exclusive and there is no overlap in between the classes. For example, a class 'Automobile' and 'Trucks' does not include any overlapping features. The 'Automobile' classifies things that are like sedans or SUVs whereas 'Truck' classifies the vehicles that are larger in size.

Fig. 3.3. Random training sample images from CIFAR-10 dataset.

Table 3.1.
10 Classes of CIFAR-10 Dataset.

Airplane	Automobile	Bird	Cat	Deer
Dog	Frog	Horse	Ship	Truck

For this research work, we have selected a pretrained CIFAR-10 [23] network for transfer learning. We used MATLAB 2017a [24] as software platform and necessary tools in the Image processing and Computer vision toolbox. Generally, a CNN structure is composed of following layers:

- *imageInputLayer* - Image input layer
- *convolutional2dLayer*- 2D convolution layer
- *reluLayer* - Rectified linear unit (ReLU) layer
- *maxPooling2dLayer*- Max pooling layer
- *fullyConnectedLayer* - Fully connected layer
- *softmaxLayer* - Softmax layer
- *classificationLayer* - Final output layer (Classification Layer) for the neural network

For our application, first a CNN is configured with 3 convolutional layers. The first convolutional layer has 32 filters of size 5×5. It produces 32 feature maps of size 28×28 at the output followed by a ReLU layer and 3×3 Max pooling layer. Stride and zero padding for the convolutional and Max pooling layer are [1 1] and [2 2] and [2 2] and [0 0] respectively. The second convolutional layer has exactly same configuration as the first convolutional layer followed by similar ReLU and Max Pooling layer. The third convolutional layer has 64 filters of size 5×5 with similar stride and zero padding followed by similar ReLU and Max Pooling layer. There are 2 Fully Connected layers with 64 and 10 fully connected neurons respectively with a ReLU layer in between. Finally, there are the Softmax and Classification layers that output the final classification scores of the network. Table 3.2 summarizes the whole network.

Table 3.2.
15 Layer Convolutional Neural Network.

Layer	Layer Type
Image Input	32x32x3 images with 'zerocenter' normalization
Convolution	32 5x5 convolutions with stride [1 1] and padding [2 2]
ReLU	ReLU
Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
Convolution	32 5x5 convolutions with stride [1 1] and padding [2 2]
ReLU	ReLU
Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
Convolution	64 5x5 convolutions with stride [1 1] and padding [2 2]
ReLU	ReLU
Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
Fully Connected	64 fully connected layer
ReLU	ReLU
Fully Connected	10 fully connected layer
Softmax	softmax
Classification Output	crossentropyex

3.0.2 Image Labelling

Once we configured the network, weights and bias data from a pretrained CIFAR-10 network has been loaded and tested. Next the network can classify 10 objects from

an input image from table 3.1. However, the goal for this research is to classify road edges. So, at this point we need the transfer learning feature of the CNN. In order to apply transfer learning, we extract the learned parameters from the pretrained network and then retrain the final output layer of the network with new labeled truth data that corresponds to our new problem of interest. Street view images are now used for the new training, including a Region of interest (ROI). This ROI is selected over the image in such way that the region corresponds to road edge and contains information to train the network for our classification problem.

'Training Image Labeler' app in the 'Image Processing and Computer Vision' toolbox in MATLAB 2017a is used to label the images and create the ground truth data. 239 Images were labeled with 5 different categories: Concrete, Grass, Dirt, Gravel and Tree. Fig. 3.4 shows two examples of image labeling for ground truth data. The ROIs are bounded by rectangular regions. These regions contain feature information required for the training.

3.0.3 Training of CNN for Road Edge

The ground truth data for training consists of 239 images as discussed above. The training has been done using MATLAB 2017a on a single CPU (3.5 Giga Hertz) with 16 Giga Bytes of RAM. Network parameters are summarized in table 3.3. 250 randomly selected images are used to test the network accuracy. Detailed discussions are in results section.

3.0.4 Re-configuration of the Network

The network trained for the road edge classification with 3 convolutional layers has in total 116,646 weights and biases. Web framework [25] developed by NECST Lab at Politecnico di Milano (details mentioned in chapter 4) has a feature to check utilization estimation of the resources available in Zedboard by providing the network parameters (weights and biases). The estimation with the three convolutional layer

Fig. 3.4. Image Labeling through 'Training Image Labeler' app of MATLAB 2017a in Google street view images. Example shows Concrete and Dirt road edge type labeling

network showed 100% BRAM utilization. As the next part of this research work deals with downloading the network in FPGA, with limited resources available, the re-configuration of this current network was needed in order to fit the network into the Zynq 7000 FPGA. Several different configurations were made and estimated resource utilizations for each of these reconfigured networks were evaluated. Finally,

Table 3.3.
Parameter Information for the Trained CNN.

Layer	Number of Parameters
Conv Layer 1	Weights: 5x5x3x32 = 2400 Biases= 32
Conv Layer 2	Weights: 5x5x32x32 = 25600 Biases= 32
Conv Layer 3	Weights: 5x5x32x64=51200 Biases=64
Fully Connected Layer 1	Weights: 64x576=36864 Biases=64
Fully Connected Layer 2	Weights: 6x64=384 Biases=6
	Total = 116,646

a reconfiguration of the network by removing one convolutional layer, reducing the number of filters in first convolutional layer to 16 and reducing the number of first fully connected layer neurons to 32 gives no 100% utilization estimation for any available resources in Zedboard. Table 3.4 summarizes the re-configured network structure.

With the network reconfigured, using same ground truth data the network is trained again and tested for the same 250 random image samples which were used to test 3 layer CNN. Table 3.5 summarizes the network parameter information for the reconfigured network.

3.0.5 Results

We have configured and trained two different CNNs based on CIFAR-10 dataset. For testing, a randomly selected 250 Google street view images are used for both of the

Table 3.4.
12 Layer Convolutional Neural Network.

Layer	Layer Type
Image Input	32x32x3 images with 'zerocenter' normalization
Convolution	16 5x5 convolutions with stride [1 1] and padding [2 2]
ReLU	ReLU
Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
Convolution	32 5x5 convolutions with stride [1 1] and padding [2 2]
ReLU	ReLU
Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
Fully Connected	32 fully connected layer
ReLU	ReLU
Fully Connected	10 fully connected layer
Softmax	softmax
Classification Output	crossentropyex

networks. The test image size is 600x300 and the complete image is fed through the network. The network computes the scores for the detected regions over the image that matches with the trained data. However, the score for the detected regions can be at the different position of the image that may not match with the problem specification. To determine success/failure, an ROI is set over the image. Detections along/near left or right margin of the image and along the upper part of the image (mainly due to presence of horizon/sky) are ignored. Once the image is processed, provided that it computes score for certain material types, positions of the detected

Table 3.5.
Parameter Information for the Reconfigured CNN.

Layer	Number of Parameters
Conv Layer 1	Weights: 5x5x3x16 = 1200 Biases= 16
Conv Layer 2	Weights: 5x5x16x32 = 12800 Biases= 32
Fully Connected Layer 1	Weights: 32x1568=50176 Biases=32
Fully Connected Layer 2	Weights: 6x32=192 Biases=6
	Total = 64,454

regions are analyzed and if these positions fall inside the specified region, the network will show the classified material at the output that gets the maximum confidence score. Otherwise, the network will display the message of no detection.

Fig. 3.5 and Fig. 3.6 shows the extracted features by the first layer of CNN from the input image for 3 layers and 2 layers CNNs respectively. A few examples of the output are shown in Fig. 3.7, 3.8 and 3.9. In Fig. 3.7 and 3.8 the CNN classifies the road side materials correctly. In Fig 3.9 two cases are shown where the network misclassifies (top picture) or failed to classify any road edge materials (bottom picture, no detection message displayed).

The confusion matrices for both of these CNN models are shown in Fig 3.10. For 3 Layer CNN model the accuracy comes as 52.8% and for 2 Layer CNN model the accuracy comes as 35.2%. Both of these models show improvements over clustering work initial accuracy of 33% without any further modification. Further improvement in the work still possible and is mentioned later in the recommendation chapter. This work shows, for the classifying road edges problem, the use of CNNs provide better

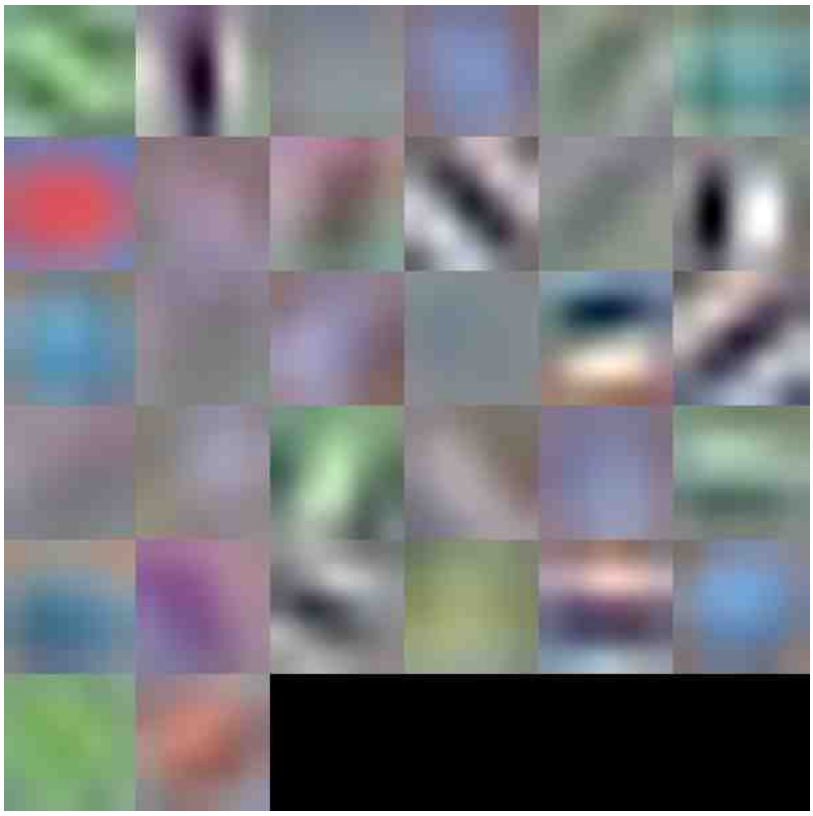

Fig. 3.5. Feature extracted from input image by 32 Filters from first layer conv net of the 3 layers CNN

results compared to self-organizing map (SOM) generalized neural networks. CNNs can use higher resolution and higher dimension input images, so that the network can get more color features and in the end provide better accuracy. With a higher number of filters and convolutional layer, the network extracts more features which results in better accuracy and vice versa when using a 16 filter network with one less convolutional layer. Next we discuss our research and results that led to an efficient FPGA hardware implementation.

Fig. 3.6. Feature extracted from input image by 16 Filters from first layer conv net of the 2 layers CNN

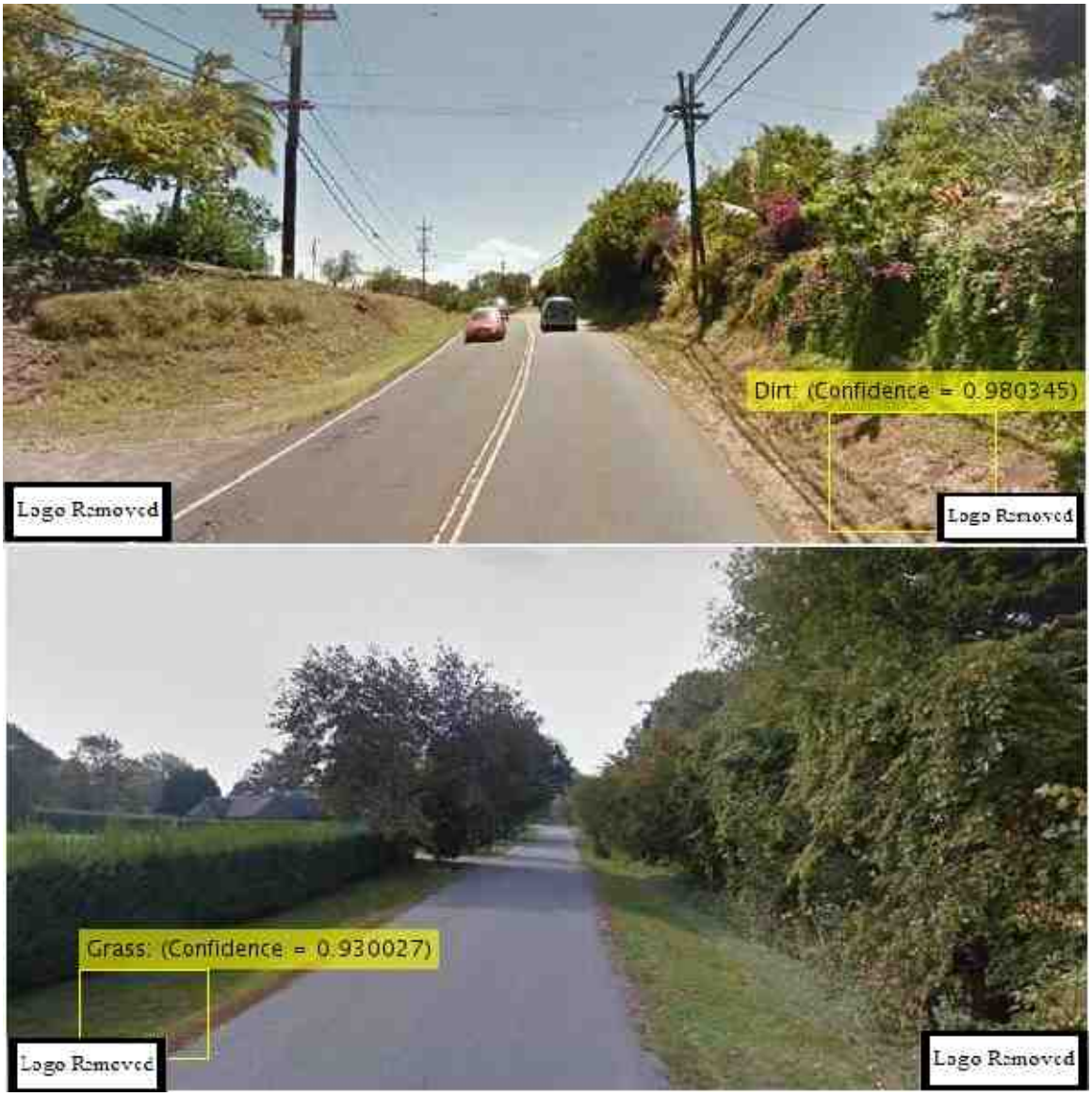

Fig. 3.7. Network classifies Concrete (top) and Gravel (bottom).

Fig. 3.8. Network classifies Dirt (top) and Grass (bottom).

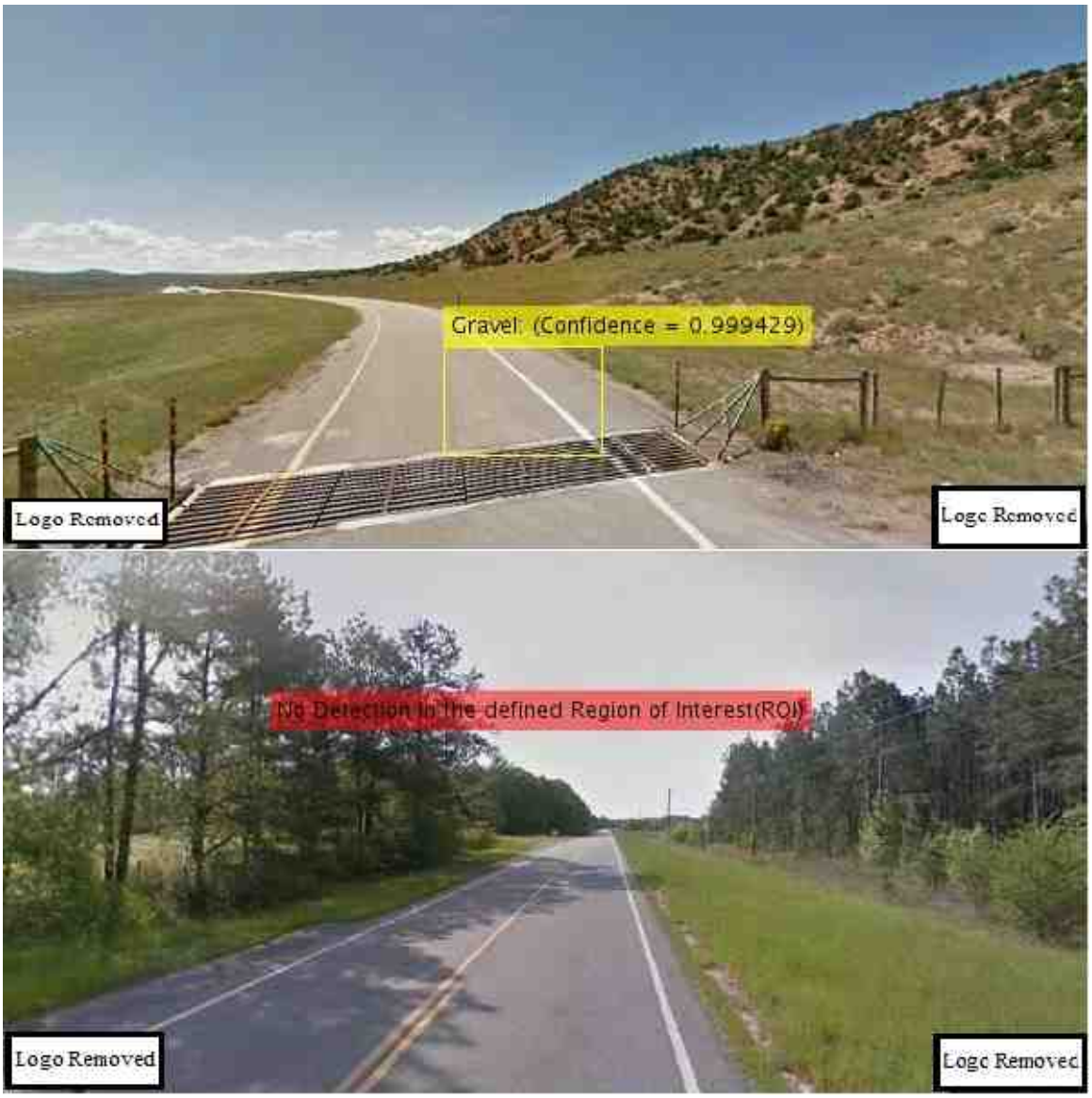

Fig. 3.9. Network mis-classifies Gravel (top) and fails to classify any material (bottom)

	Grass	Tree	Dirt	Concrete	Gravel	
Grass	51	2	1	3	0	
Tree	0	11	1	0	0	
Dirt	2	1	24	1	0	
Concrete	1	2	2	38	1	
Gravel	0	0	5	3	8	
None	32	12	26	23	0	
Total	86	28	59	68	9	
						52.80%

	Grass	Tree	Dirt	Concrete	Gravel	
Grass	28	3	0	5	0	
Treee	1	17	3	1	0	
Dirt	0	1	20	3	1	
Concrete	2	1	1	20	2	
Gravel	1	0	5	0	3	
None	53	10	32	35	2	
Total	85	32	61	64	8	
						35.20%

Fig. 3.10. Confusion Matrix for Human Labeled vs Network Classified. Top one for 3 Layer CNN, bottom one for 2 Layer CNN

4. HARDWARE IMPLEMENTATION

As mentioned in the previous chapters, using CNNs in computer vision related problem is of great research interest in recent days. Once one obtains a CNN architecture, it is important to find an efficient way to incorporate the model into application specific hardware. CNNs have become state of the art in big data analysis, remote surveillance, and machine vision. As this research goal is to create a model to classify road edges, the future vision is to contribute to the next generation automotive safety feature technologies and autonomous driving. In this part of the research, a successful implementation of the network in Zynq 7000 FPGA is shown.

CNNs are computationally complex networks and typically require GPU support to perform faster. GPUs consume a lot of power and do may not be an option in application specific (ASIC) neural net applications. Since the FPGA consumes less power, includes reprogramming capability, and also is able to perform multiple executions at the same time (parallel processing); therefore FPGAs offer lower throughput and turnaround time in order to accelerate complex CNNs.

4.0.1 Framework description and Validation

NECST Lab at Politecnico di Milano, Italy created a web framework [25] that uses a Graphical User Interface (GUI) to take network parameters as input and provides files necessary to Synthesize the network and command line (tcl) scripts required to use Xilinx tools for hardware implementation. The framework also allows changes to network configuration and generates synthesizable files, required (tcl) scripts and C++ source code for the configured network. The complete workflow mentioned in [26] and has been successfully validated using software tools: Vivado 2015.3 and Xilinx SDK [27] and the hardware: ZedBoard Zynq 7000 FPGA [28]. The CNN

model used for the validation work was LeNet-5 [14] (digit classification). This initial validation network has 2 convolutional layers, takes 16x16 grayscale image input and classifies 10 categories. The network has only a small number of network parameters (2742 weights and biases), compared to our network. For this network the maximum FPGA utilization is 59% and on chip power consumption is 2.15 Watt. Fig 4.1 shows the post implementation utilization and power consumption for the simple test.

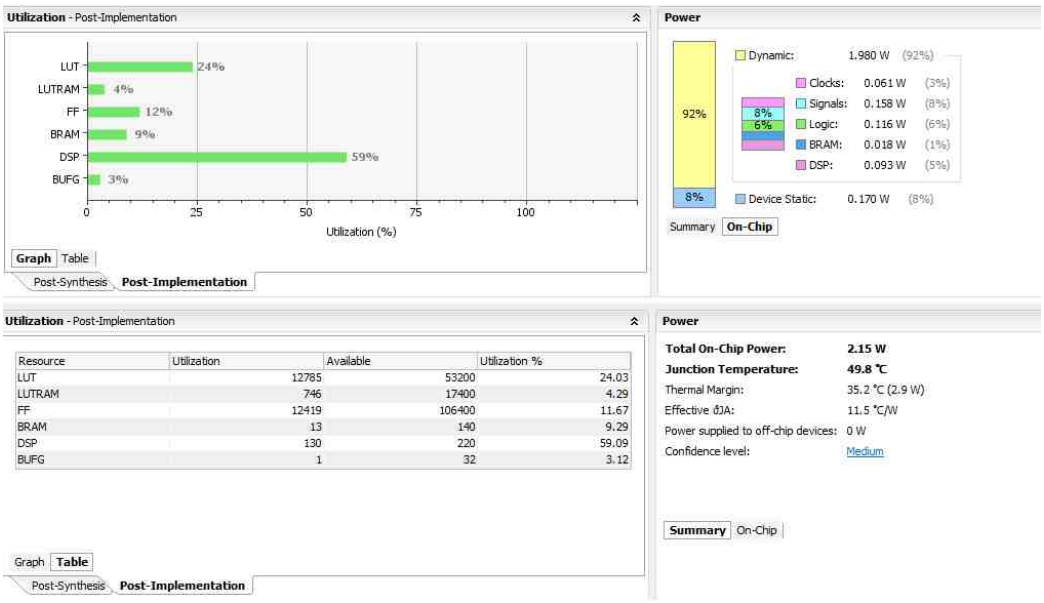

Fig. 4.1. Post Implementation Utilization and Power consumption for the Test Network.

4.0.2 Download of trained network on hardware

As the workflow to implement the network on the hardware has been validated for a smaller network, now we introduce the new developed network from our research. The web framework initially failed to generate C++ source codes, tcl scripts and synthesizable files because of the comparatively large number of network parameters.

So, our next step was to create the network for the road edge classifier from the imported weights and biases from MATLAB. In this case we used the smaller of our two CNNs (2-layer network).

In the process, three scripts are required, two by Vivado HLS tools to generate IP core and the one required by Vivado Design Suite to generate bitstream of the design. Using the C++ source code for the network and directives information from the TCL script, the Vivado HLS tool accelerates IP core generation. The information from the image to be classified is passed through the IP core followed by a data flow pattern where intermediate buffers are used in between input and output layers. The first layer takes the data through the *AXI4* stream from on-board memory using the ARM processor to move data to the hardware CNN. Finally the resulting classification value is returned to the ARM processor.

Once the design part is done, in the next step implements the block design including embedded ARM in Vivado Design Suite and it utilizes the third script. Fig. 4.2 shows the generated block design of the network.

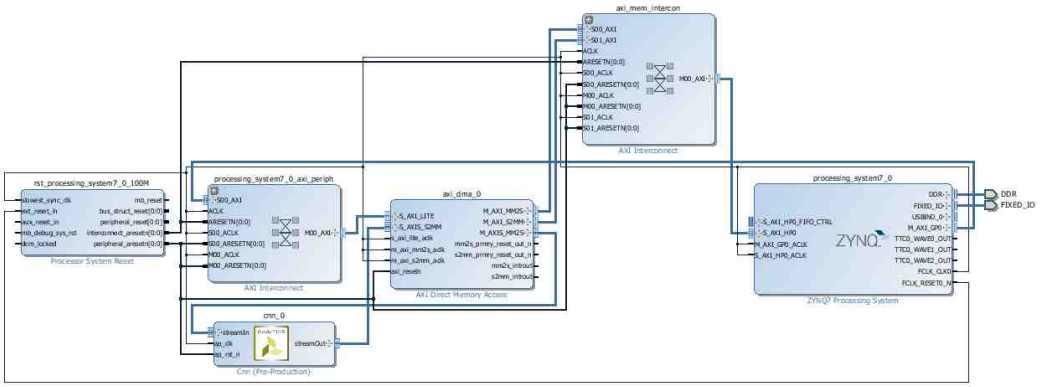

Fig. 4.2. Generated Hardware Block design for the Network.

The block design composed of following components:

- Zynq7 Processing System,
- AXI Interconnects,

- AXI Direct Memory Access (DMA),
- Processing System Reset and
- CNN IP core.

The processing system transfers data to AXI- DMA through Axi Interconnects. At next stage CNN IP core receives the stream data from AXI-DMA. Once the CNN core calculates the classification value, it sends back the information to AXI-DMA and then finally to Processing system.

Once the design validation is completed, a HDL wrapper is created and the synthesis flow towards the bitstream generation is started. After successful bitstream generation, then network is available is to be downloaded into Zynq 7000 FPGA. Fig 4.3 shows the utilization information for our new 2-level network based on CIFAR-10. The maximum utilization is 76% and the total on chip power consumption is 2.47 Watt.

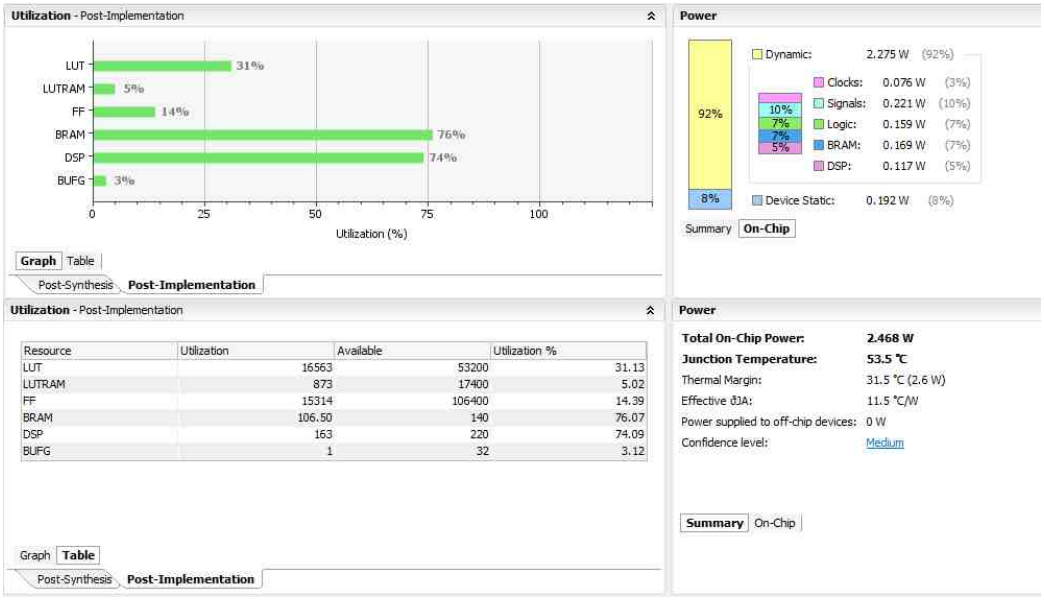

Fig. 4.3. Post Implementation Utilization and Power consumption for road edge classification, 2-level network based on CIFAR-10.

4.0.3 Recommendation

As the bistream generation is now successful for the network, it can be downloaded into any target FPGA. To download and run the network, a boot image needs to be generated using Xilinx SDK tool in Vivado Design suit and the process requires first stage bootloader (fsbl) file and should include the generated bitsream file. With the generated boot image file, necessary files to boot the Linux on Zedboard then need to be copied in the SD card (which will be plugged into the Zedboard) and after that the network can be executed through the Zedboard. An executable file also needs to be created to test the network. This research work have not included the complete hardware acceleration process of the network. Once the source code of the network in C/C++ is compiled through the ARM processor, it will create the executable file from the software version of the network. With that executable file, the network can be executed through the Zedboard which completes the hardware acceleration of the network using FPGA. Further possible improvements of the network performance is mentioned in next chapter.

5. SUMMARY

5.0.1 Conclusion

Since the beginning of this decade, a lot of research has been done on automotive safety and autonomous driving. The applicability of CNNs has been key in next generation driver safety technology and fully autonomous driving. This research work contributed to the state of the art in two ways: First we created a new road edge classifier using a CNN model which showed an improvement in initial accuracy (53% and 35%) over regular neural network model (33%). This can contribute to the next generation road departure warning or emergency pull over assist systems. In the second part of the research, the FPGA hardware implementation of the network was created for the road edge classifier network, and it achieved FPGA maximum utilization of 76% with an on chip power consumption odf 2.47 Watts. A way to implement hardware-software acceleration of these CNNs is shown.

5.0.2 Future Work

The improvement in the current research work can be made in following ways:

- On the existing network, the accuracy can be improved by increasing the number of training images. In this research work only 239 human labeled training images were used. Around thousand training images will provide stronger knowledge base for the network and possibly a further improvement in the accuracy.

- The utilization can be reduced by modification of hardware data flow of the network parameters without changing the network architecture configuration. This can be by using different methods to lower hardware requirements for the network parameters with a slight loss in accuracy. Concepts of binary neural

networks [29] where network weights and activations are constrained to +1 or -1 was introduced to reduce network parameters for a complex network. Jiantao et al. [30] showed different data quantization techniques that reduces the number of network parameters. The work evaluated 8/4 bit quantization with only 0.4% loss of accuracy. With adaptation of any proved methods any state-of-the-art network can be applied to classify road edges with significantly less number of network parameters with a slight loss in accuracy.

- With the successful reduction of the hardware utilization, it is possible to use more complex networks such as VGG16, AlexNet or GoogleNet. These networks are proved to provide better accuracy so there is a very high possibility of improving the accuracy for the road edge classifier.

- With a single CPU, the processing time is slow (around a minute per image). However, with the successful FPGA hardware acceleration, and with the use of more complex network with higher number of filters and convolutional layers, it will be possible to use the system in real time scenarios (1 second per image).

REFERENCES

REFERENCES

[1] W. T. N. Hubel D. H., "Receptive fields and functional architecture of monkey striate cortex." *Journal of Physiology (London)*, vol. 195, no. 1, pp. 215–243, March 1968.

[2] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," *International Joint Conference on Artificial Intelligence*, 2011.

[3] K.-Y. Chiu and S.-F. Lin, "Lane detection using color-based segmentation," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, June 2005, pp. 706–711.

[4] C. R. Jung and C. R. Kelber, "A lane departure warning system based on a linear-parabolic lane model," in *IEEE Intelligent Vehicles Symposium, 2004*, June 2004, pp. 891–895.

[5] P.-Y. Hsiao and C.-W. Yeh, "A portable real-time lane departure warning system based on embedded calculating technique," in *2006 IEEE 63rd Vehicular Technology Conference*, vol. 6, May 2006, pp. 2982–2986.

[6] J. L. McClelland, D. E. Rumelhart, and G. E. Hinton, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1." Cambridge, MA, USA: MIT Press, 1986, pp. 3–44.

[7] S. J. Thorpe, "Spike arrival times: A highly efficient coding scheme for neural networks," *Parallel Processing in Neural Systems*, pp. 91–94, 1990.

[8] National Institute of Aging. Neuron image. Accessed: 12.30.2009, Permission: In United States' Public domain under the terms of Title 17, Chapter 1, Section 105 of the US Code. [Online]. Available: https://www.nia.nih.gov/alzheimers/publication/alzheimers-disease-unraveling-mystery/preface

[9] P. H. Winston, *Artificial Intelligence (3rd Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[10] G. Luger and W. A. Stubblefield, *Artificial Intelligence : Structures and Strategies for Complex Problem Solving*, 2nd ed. Redwood City, California.: Benjamin/Cumming Publishing, 1993.

[11] D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986.

[12] B. Widrow and M. E. Hoff, "Adapting switching circuits," in *1960 IRE WESCON Convention Record*, no. 4. New York, NY: IRE, 1960, pp. 96–104.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[15] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014.

[16] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814.

[17] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.

[18] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. Chung, "Accelerating deep convolutional neural networks using specialized hardware." Microsoft Research, February 2015, Accessed: 11.29.2017. [Online]. Available: https://www.microsoft.com/en-us/research/publication/accelerating-deep-convolutional-neural-networks-using-specialized-hardware/

[19] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161–170.

[20] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[23] A. Krizhevsky, "Learning multiple layers of features from tiny images." Toronto,Canada: University of Toronto, May, 2012, "M.Sc. Thesis".

[24] Mathworks Inc. (2017) Matlab 2017a. Accessed: 05.18.2017. [Online]. Available: https://www.mathworks.com/products/matlab.html

[25] M. I. NECST Labrotory, Milano. (2017) Synthesizable conv net creation. Accessed: 11.29.2017. [Online]. Available: https://cnnecst.hosting.necst.it

[26] E. D. Sozzo, A. Solazzo, A. Miele, and M. D. Santambrogio, "On the automation of high level synthesis of convolutional neural networks," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 217–224.

[27] Xilinx Corporation. (2015) Vivado design suite 2015.3. Accessed: 11.29.2017. [Online]. Available: https://www.xilinx.com/news/press/2015/xilinx-vivado-design-suite-2015-3-takes-design-to-new-heights-with-ip-sub-systems.html

[28] Digilant Corporation. (2014) Zynq 7000. Accessed: 11.29.2017. [Online]. Available: https://www.digikey.com/catalog/en/partgroup/zedboard-zynq-7000-development-board/49272

[29] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[30] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 26–35.