MAEER's

MAHARASHTRA INSTITUTE OF TECHNOLOGY, PUNE.

BE Project Report

ON

APPLICATION SPECIFIC IC FOR CNN

SUBMITTED BY,

NISHA PATIL (B150023087)

SHARAD TEMBHURNE (B150023122)

Project Guide:

Prof. Anuja A. Askhedkar (Internal Guide)

Sponsored by:

College

Year: 2018-2019

Department of Electronics and Telecommunication

Maharashtra Institute of Technology, Pune - 38**.**

MAEER's

MAHARASHTRA INSTITUTE OF TECHNOLOGY, PUNE.

CERTIFICATE

This is to certify that Project Stage-II entitled

APPLICATION SPECIFIC IC FOR CNN

has been carried out successfully by

NISHA PATIL (B150023087)

SHARAD TEMBHURNE (B150023122)

during the Academic Year **2018-2019**
in partial fulfillment of their
course of study for Bachelor's Degree in
Electronics and Telecommunication as per the syllabus prescribed by the
SPPU.

Prof. Anuja A. Askhedkar
Internal Guide

Head of Department
(Electronics & Telecommunication)

DECLARATION

We the undersigned, declare that the work carried out under

Project Stage-II entitled

APPLICATION SPECIFIC IC FOR CNN

has been carried out by us and it is not being implemented by any external agency/company that sells projects**.** We further declare that work submitted in the form of a report is not been copied from any paper/thesis/site as it is**.** However, existing methods/approaches from any paper/thesis/site are being cited and acknowledged in this report's reference section**.**

We are aware that our failure to adhere to the above, the Institute/University/Examiners can take strict action against us**.** In such a case, whatever action is taken, would be binding on us**.**

Roll no	**Name of student**	**Signature with date**
405015	Nisha Patil	
405077	Sharad Tembhurne	

Academic Year 2018-2019

MIT, Pune**.**

ACKNOWLEDGEMENT

A project is an opportunity for the students to practically implement theoretical concepts. It proves to be a learning platform for the students so that they can compete successfully in their professional life. However, in this entire journey of completing the project, we need proper guidance so as to avoid obvious mistakes.

We would like to thank our principal Prof. Dr. L. K. Kshirsagar for his constant encouragement. We would also like to thank our head of the E&TC department Prof. Dr. B. S. Chaudhari.

We would thank our internal guide Prof. Anuja A. Askhedkar for her invaluable guidance and support in making this project a success. We would also like to thank Prof. P. M. Jadhav for her profound help.

Lastly, our sincere thanks to all staff members and friends who helped us in all possible ways to make this project a success.

ABSTRACT

The goal of the project is to design an application-specific integrated circuit that implements a convolutional neural network (CNN) to process and detect images. A convolutional neural network uses multiple layers of convolution, pooling, and rectified linear units to give a result.

CNNs compare images piece by piece. The pieces that it looks for are called features. By finding rough feature matches in roughly the same positions in two images, CNNs get a lot better at seeing similarity than whole-image matching schemes.

When presented with a new image, CNN doesn't know exactly where these features will match so it tries them everywhere, in every possible position. In calculating the match to a feature across the whole image, a filter is made. The math used to do this is called convolution, from which Convolutional Neural Networks take their name.

Another power tool that CNNs use is called pooling. Pooling is a way to take large images and shrink them down while preserving the most important information in them. It consists of stepping a small window across an image and taking the maximum value from the window at each step. In practice, a window of 2 or 3 pixels on a side and steps of 2 pixels work well.

A small but important player in this process is the Rectified Linear Unit or ReLU. Its math is also very simple—wherever a negative number occurs, swap it out for a 0. This helps the CNN stay mathematically healthy by keeping learned values from getting stuck near 0 or blowing up toward infinity.

Once the network has been designed and trained it can then be translated into a hardware description language to be able to design the integrated circuit using the design tool Mentor Graphics.

CONTENTS

LIST OF FIGURES

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses, and many other aspects of visual data.

Convolutional networks perform optical character recognition (OCR) to digitize text and make natural-language processing possible on analog and hand-written documents, where the images are symbols to be transcribed. CNNs can also be applied to sound when it is represented visually as a spectrogram. More recently, convolutional networks have been applied directly to text analytics as well as graph data with graph convolutional networks.

The efficacy of convolutional nets (ConvNets or CNNs) in image recognition is one of the main reasons why the world has woken up to the efficacy of deep learning. They are powering major advances in computer vision (CV), which has obvious applications for self-driving cars, robotics, drones, security, medical diagnoses, and treatments for the visually impaired.

1.1 SCOPE OF PROJECT

The main objective is to identify and classify images containing garbage in them. The proposed system must take care of two main tasks: The first task is to process the image dataset and then train itself. The second task is to classify all images that are being tested into images with or without garbage. The fact that CNNs are trained end-to-end, from raw pixels to final classes, makes them much more advantageous for many tasks than manually designing a suitable feature extractor.

1.2 ORGANIZATION OF THE REPORT

Chapter 1: Introduction and scope
This chapter provides an overview of the basic functionality of the system and describes its scope of expansion.

Chapter 2: Literature Survey and present scenario

This chapter enlightens the literature survey of the work done in this field so far as well as the present scenario.

Chapter 3: System Block Diagram and Flow Chart

Explain in detail the design and development process of the system. Includes system specifications block diagram, and description of each block.

Chapter 4: System Design

It includes all the algorithms used for feature extraction and classification and filtering.

Chapter 5: Result and Conclusion

It includes the efficiency of the network obtained thus far. Also, it includes applications of the system, conclusions, and future scope of the project.

Chapter 6: Appendix

Includes important papers and previous projects referred for algorithms of the system.

2.1 PRESENT SCENARIO

Convolutional neural networks – CNNs or ConvNets for short – are at the heart of deep learning, emerging in recent years as the most prominent strain of neural networks in research**.** They have revolutionized computer vision, achieving state-of-the-art results in many fundamental tasks, as well as making strong progress in natural language processing, computer audition, reinforcement learning, and many other areas**.** ConvNets have been widely deployed by tech companies for many of the new services and features we see today**.**

Although ConvNets have been around since the 1980s and have their roots in earlier neuroscience research, they've only recently achieved fame in the wider scientific community for a series of remarkable successes in important scientific problems across multiple domains**.** They extend neural networks primarily by introducing a new kind of layer, designed to improve the network's ability to cope with variations in position, scale, and viewpoint**.** Additionally, they have become increasingly deep, containing upwards of dozens or even hundreds of layers, forming hierarchically compositional models of images, and sounds, as well as game boards and other spatial data structures**.**

Because of their success at vision-oriented tasks, they have been adopted by creative technologists and interaction designers, allowing their installations not only to detect movement, but to actively identify, describe, and track objects in physical spaces**.**

3.1 PROJECT SPECIFICATIONS

Software requirements:

- Python v 3.5.2
- Python IDLE software environment
- TensorFlow
- Xilinx ISE Design Suite 14.7
- Mentor Graphics design software

3.2 SYSTEM BLOCK DIAGRAM

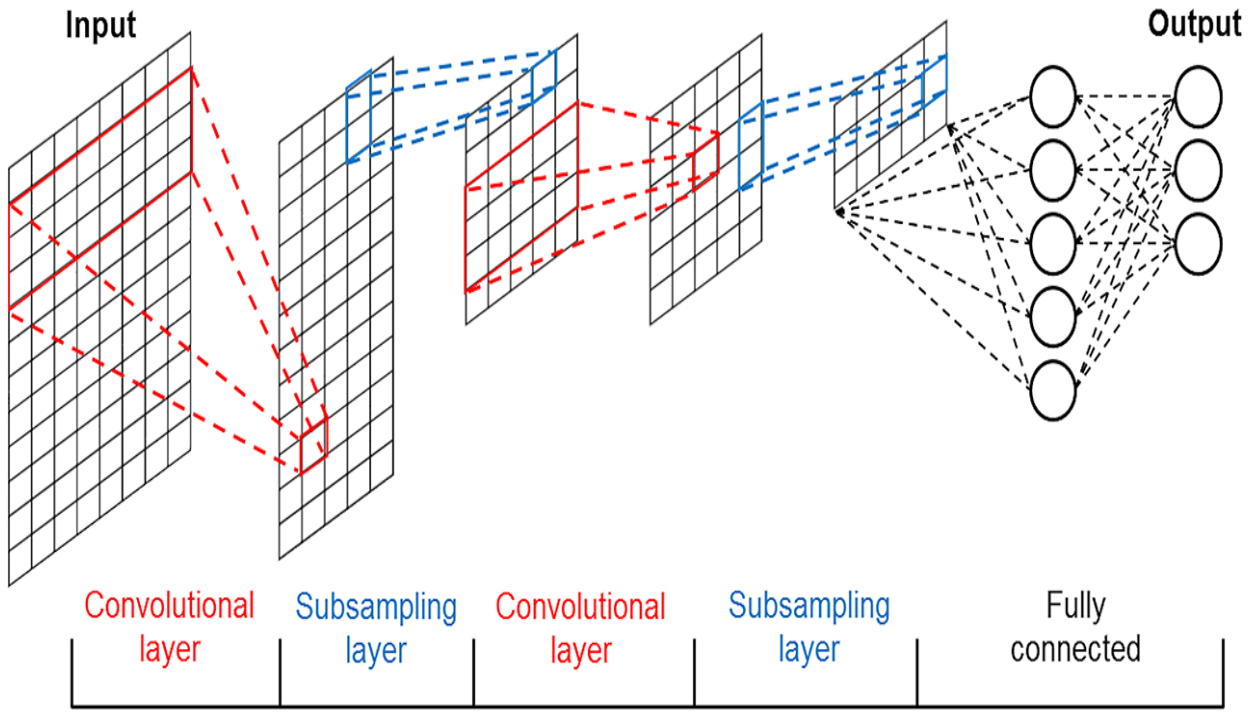

Figure 1: System block diagram.

Given a dataset of grayscale images with a standardized size of 32×32 pixels each, a traditional feedforward neural network would require 1024 input weights (plus one bias).

This is fair enough, but the flattening of the image matrix of pixels to a long vector of pixel values loses all of the spatial structure in the image. Unless all of the images are perfectly resized, the neural network will have great difficulty with the problem.

Convolutional Neural Networks expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data. Features are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network.

It is this reason why the network is so useful for object recognition in photographs, picking out digits, faces, objects, and so on with varying orientations.

There are three types of layers in a Convolutional Neural Network:

1. Convolutional Layers

Convolutional layers are comprised of filters and feature maps.

Filters

The filters are the “neurons” of the layer. They have input weights and output a value. The input size is a fixed square called a patch or a receptive field.

If the convolutional layer is an input layer, then the input patch will be pixel values. If the deeper in the network architecture, then the convolutional layer will take input from a feature map from the previous layer.

Feature Maps

The feature map is the output of one filter applied to the previous layer.

A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in the activation of the neuron and the output is collected in the feature map. You can see that if the receptive field is moved one pixel from activation to activation, then the field will overlap with the previous activation by (field width – 1) input values.

Zero Padding

The distance that the filter is moved across the input from the previous layer each activation is referred to as the stride.

If the size of the previous layer is not cleanly divisible by the size of the filter's receptive field and the size of the stride then it is possible for the receptive field to attempt to read off the edge of the input feature map. In this case, techniques like zero padding can be used to invent mock inputs for the receptive field to read.

2. Pooling Layers

The pooling layers down-sample the previous layer's feature map.

Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layer's feature map. As such, pooling may be considered a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.

They too have a receptive field, often much smaller than the convolutional layer. Also, the stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap.

Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

For more about pooling layers, see the post:

3. Fully Connected Layers

Fully connected layers are the normal flat feed-forward neural network layer.

These layers may have a non-linear activation function (tanh or sigmoidal) in order to output probabilities of class predictions.

Fully connected layers are used at the end of the network after feature extraction and consolidation have been performed by the convolutional and pooling layers. They are used to create final non-linear combinations of features and for making predictions by the network.

Chapter 4

SYSTEM DESIGN

4.1 DATA COLLECTION AND PREPROCESSING

The starting point of the project was the creation of a database with all the character images that would be used for training and testing**.** The images are then needed to be resized to the predefined dimensions of 128 x 128 x 3**.** After being resized the images were placed in two separate folders named training and testing which held the training and validation images respectively**.** The images in the training folder are further divided into positive and negative images, this aids the network in learning the classification process**.**

After studying data preparation performed across top-performing models (such as GoogLeNet), one can summarize a number of best practices to consider when preparing data for own image classification tasks**.**

1. **Data Preparation.** A fixed size must be selected for input images, and all images must be resized to that shape**.** The most common type of pixel scaling involves centering pixel values per channel, perhaps followed by some type of normalization**.**
2. **Train-Time Augmentation.** Train-time augmentation is required, most commonly involving resizing and cropping of input images, as well as modification of images such as shifts, flips, and changes to colors**.**
3. **Test-Time Augmentation.** Test-time augmentation was focused on systematic crops of the input images to ensure features present in the input images were detected**.**

Another good practice while preparing the database is to manually scale the images**.**

Images are comprised of matrices of pixel values**.**

Black and white images are a single matrix of pixels, whereas color images have a separate array of pixel values for each color channel, such as red, green, and blue**.**

Pixel values are often unsigned integers in the range between 0 and 255**.** Although these pixel values can be presented directly to neural network models in their raw format, this can result in challenges during modeling, such as in the slower-than-expected training of the model**.**

Instead, there can be a great benefit in preparing the image pixel values prior to modeling, such as simply scaling pixel values to the range 0-1 to centering and even standardizing the values.

Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. It is valid for images to have pixel values in the range of 0-1 and images can be viewed normally.

This can be achieved by dividing all pixel values by the largest pixel value; that is 255. This is performed across all channels, regardless of the actual range of pixel values that are present in the image.

Another popular data preparation technique for image data is to subtract the mean value from the pixel values. This approach is called centering, as the distribution of the pixel values is centered on the value of zero.

Centering can be performed before or after normalization. Centering the pixels and then normalizing will mean that the pixel values will be centered close to 0.5 and be in the range of 0-1. Centering after normalization will mean that the pixels will have positive and negative values, in which case images will not display correctly (e.g. pixels are expected to have values in the range of 0-255 or 0-1). Centering after normalization might be preferred, although it might be worth testing both approaches.

Centering requires that a mean pixel value be calculated prior to subtract it from the pixel values. The mean can be calculated for all pixels in the image, referred to as global centering, or it can be calculated for each channel in the case of color images, referred to as local centering.

4.2 VISUALIZING FILTERS AND FEATURE MAPS

Deep learning neural networks are generally opaque, meaning that although they can make useful and skillful predictions, it is not clear how or why a given prediction was made.

Convolutional neural networks have internal structures that are designed to operate upon two-dimensional image data, and as such preserve the spatial relationships for what was learned by the model. Specifically, the two-dimensional filters learned by the model can be inspected and visualized to discover the types of features that the model will detect, and the activation maps output by convolutional layers can be inspected to understand exactly what features were detected for a given input image. Each convolutional layer has two sets of weights. One is the block of filters and the other is the block of bias values.

The models are comprised of small linear filters and the result of applying filters called activation maps, or more generally, feature maps. Both filters and feature maps can be visualized.

For example, one can design and understand small filters, such as line detectors. Perhaps visualizing the filters within a learned convolutional neural network can provide insight into how the model works.

The feature maps that result from applying filters to input images and to feature maps output by prior layers could provide insight into the internal representation that the model has of a specific input at a given point in the model.

1. Visualizing filters

The simplest visualization to perform is to plot the learned filters directly. In neural network terminology, the learned filters are simply weights, yet because of the specialized two-dimensional structure of the filters, the weight values have a spatial relationship to each other, and plotting each filter as a two-dimensional image is meaningful (or could be). However, an architectural concern with a convolutional neural network is that the depth of a filter must match the depth of the input for the filter (e.g. the number of channels).

2. Visualizing feature maps

The activation maps, called feature maps, capture the result of applying the filters to input, such as the input image or another feature map. The idea of visualizing a feature map for a specific input image would be to understand what features of the input are detected or preserved in the feature maps. The expectation would be that the feature maps close to the input detect small or fine-grained detail, whereas feature maps close to the output of the model capture more general features. This pattern is to be expected since the model abstracts the features from the image into more general concepts that can be used to make a classification.

4.3 CONVOLUTION

Central to the convolutional neural network is the convolutional layer that gives the network its name. In the context of a convolutional neural network, convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is an element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the scalar product.
Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

This systematic application of the same filter across an image is a powerful idea. If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input image allows the filter an opportunity to discover that feature anywhere in the image. This capability is commonly referred to as translation invariance, e.g. the general interest in whether the feature is present rather than where it was present.

The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a feature map.

Once a feature map is created, each value in the feature map can be passed through a nonlinearity, such as a ReLU.

4.4 ReLU

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

The function must also provide more sensitivity to the activation sum input and avoid easy saturation.

A node or unit that implements this activation function is referred to as a **rectified linear activation unit** or ReLU for short. Often, networks that use the rectifier function for the hidden layers are referred to as rectified networks.
Adoption of ReLU may easily be considered one of the few milestones in the deep learning revolution, e.g. the techniques that now permit the routine development of very deep neural networks.

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero.

Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimize with gradient-based methods. They also preserve many of the properties that make linear models generalize well.

Because the rectified function is linear for half of the input domain and nonlinear for the other half, it is referred to as a piecewise linear function or a hinge function.
However, the function remains very close to linear, in the sense that is a piecewise linear function with two linear pieces.

4.5 POOLING

Convolutional layers in a convolutional neural network systematically apply learned filters to input images in order to create feature maps that summarize the presence of those features in the input.

Convolutional layers prove very effective, and stacking convolutional layers in deep models allows layers close to the input to learn low-level features (e.g. lines) and layers deeper in the model to learn high-order or more abstract features, like shapes or specific objects.

A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image.

A common approach to addressing this problem from signal processing is called downsampling. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task.

Downsampling can be achieved with convolutional layers by changing the stride of the convolution across the image. A more robust and common approach is to use a pooling layer. A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g. ReLU) has been applied to the feature maps output by a convolutional layer; for example, the layers in a model may look as follows:

1. Input Image
2. Convolutional Layer
3. Nonlinearity
4. Pooling Layer

The addition of a pooling layer after the convolutional layer is a common pattern used for ordering layers within a convolutional neural network that may be repeated one or more times in a given model.

The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels.

This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g. each dimension is halved, reducing the number of pixels or values in each feature map to one-quarter the size. For example, a pooling layer applied to a feature map of 6×6 (36 pixels) will result in an output pooled feature map of 3×3 (9 pixels).

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

1. **Average Pooling**: Calculate the average value for each patch on the feature map.
2. **Maximum Pooling (or Max Pooling)**: Calculate the maximum value for each patch of the feature map.

In this instance, the max pooling operation has been employed.
The result of using a pooling layer and creating down-sampled or pooled feature maps is a summarized version of the features detected in the input. They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same location. This capability added by pooling is called the model's invariance to local translation.

4.6 FULLY CONNECTED LAYERS

CNNs have one more arrow in their quiver. Fully connected layers take the high-level filtered images and translate them into votes. Fully connected layers are the primary building block of traditional neural networks. Instead of treating inputs as a two-dimensional array, they are treated as a single list and all treated identically. Every value gets its own vote on whether the current image contains garbage or not. However, the process isn't entirely democratic. Some values are much better than others at knowing when the image has garbage in it. These get larger votes than the others. These votes are expressed as weights, or connection strengths, between each value and each category.

When a new image is presented to the CNN, it percolates through the lower layers until it reaches the fully connected layer at the end. Then an election is held. The answer with the most votes wins, and it is declared the category of the input.

Fully connected layers, like the rest, can be stacked because their outputs (a list of votes) look a whole lot like their inputs (a list of values). In practice, several fully connected layers are often stacked together, with each intermediate layer voting on phantom "hidden" categories. In effect, each additional layer lets the network learn ever more sophisticated combinations of features that help it make better decisions.

5.1 SUB-MODULE ALGORITHMS

1. **Convolution**
   The first layer is the convolution layer which requires input in the form of a feature. A feature is a small part of the image that is selected beforehand to be run through the image and then match with a part of the input image. Each feature is like a mini-image, a small two-dimensional array of values. Features match common aspects of the images. The parts of both images representing features are matched with each other. This step is a better approach as compared to matching the whole image as it makes the network more efficient and helps it to learn to identify the image even if it were modified, for example, if the image was rotated or translated. During the convolution process, the feature is convoluted with every section of the input image to produce a grid image containing the convolution. To calculate the match of a feature to a patch of the image, simply multiply each pixel in the feature by the value of the corresponding pixel in the image. Then add up the answers and divide by the total number of pixels in the feature. If all the pixels in a feature match, then adding them up and dividing by the total number of pixels gives a 1. Similarly, if none of the pixels in a feature match the image patch, then the answer is a -1.

2. **ReLU**
   First, the length of the input array is found and saved in a variable. Zeroes equal to the length of the array are created. This helps later because the VHDL script faces difficulty in dynamic allocation which leads to errors. All the necessary variables are generated after this. Now, begin traversing through the array. Check every element in the array. If the value of the element is less than zero, update the value of the element equal to zero. This is done till the last element in the array. The output of a ReLU layer is the same size as whatever is put into it, just with all the negative values removed.

3. **Pooling**
   Pooling is used mainly to bring the image closer to becoming a fully connected network so that the network can decide upon the result. After pooling, an image has about a quarter as many pixels as it started with. Because it keeps the maximum value from each window, it preserves the best fit of each feature within the window. This means that it doesn't care so much exactly where the feature fits as long as it fits somewhere within the window. The result of this is that CNNs can find whether a feature is in an image without worrying about where it is. Max pooling is done in this step. If the length of the input array is odd, zero padding is done to make it even. Max pooling provides a feature map that contains all the maximum values obtained when the pooling window traverses the input image. Elements that have been compared are not selected during the next comparison.

6.1 EXPERIMENTAL IMPLEMENTATION

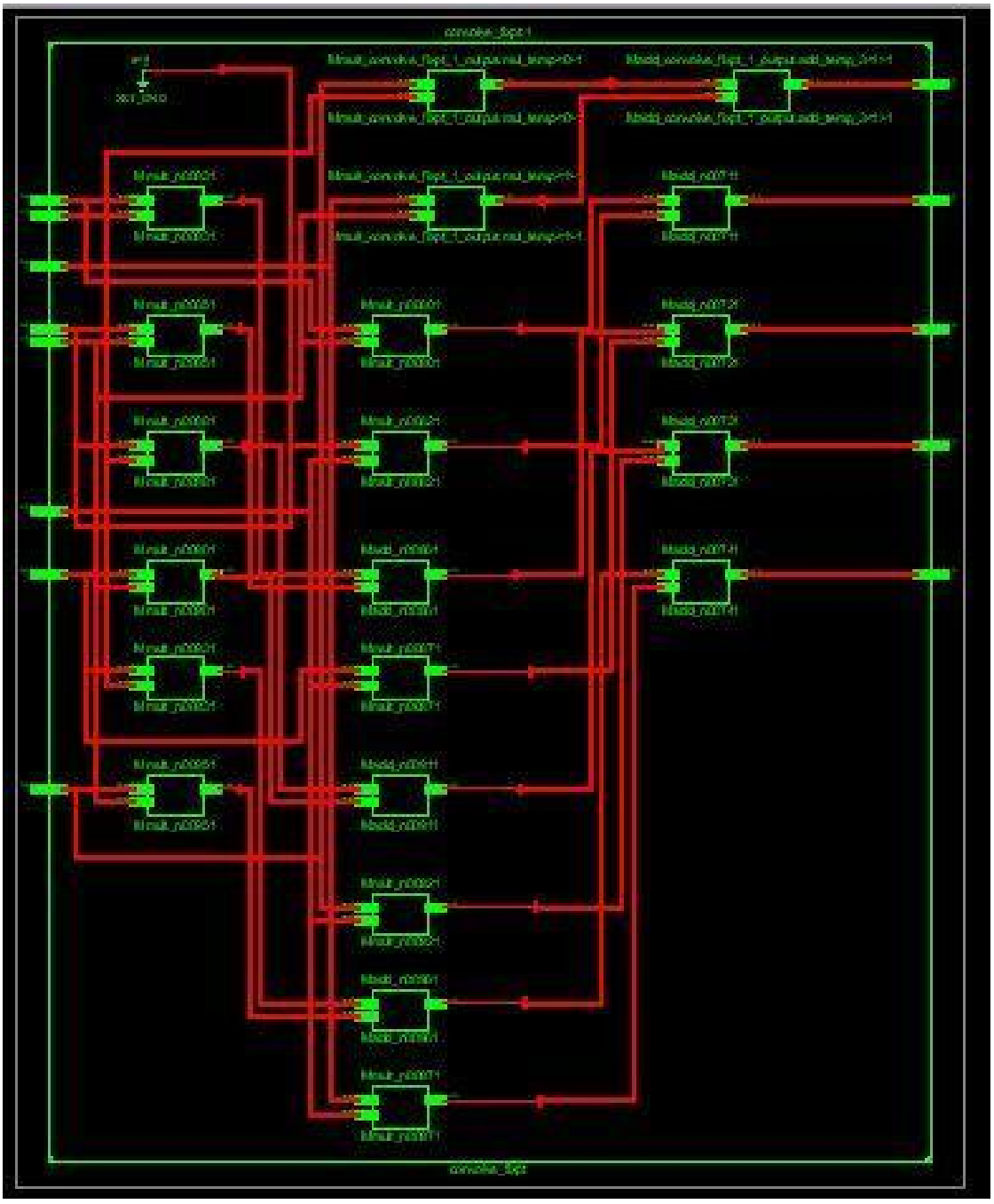

Figure 2: RTL diagram for convolutional layer implementation.

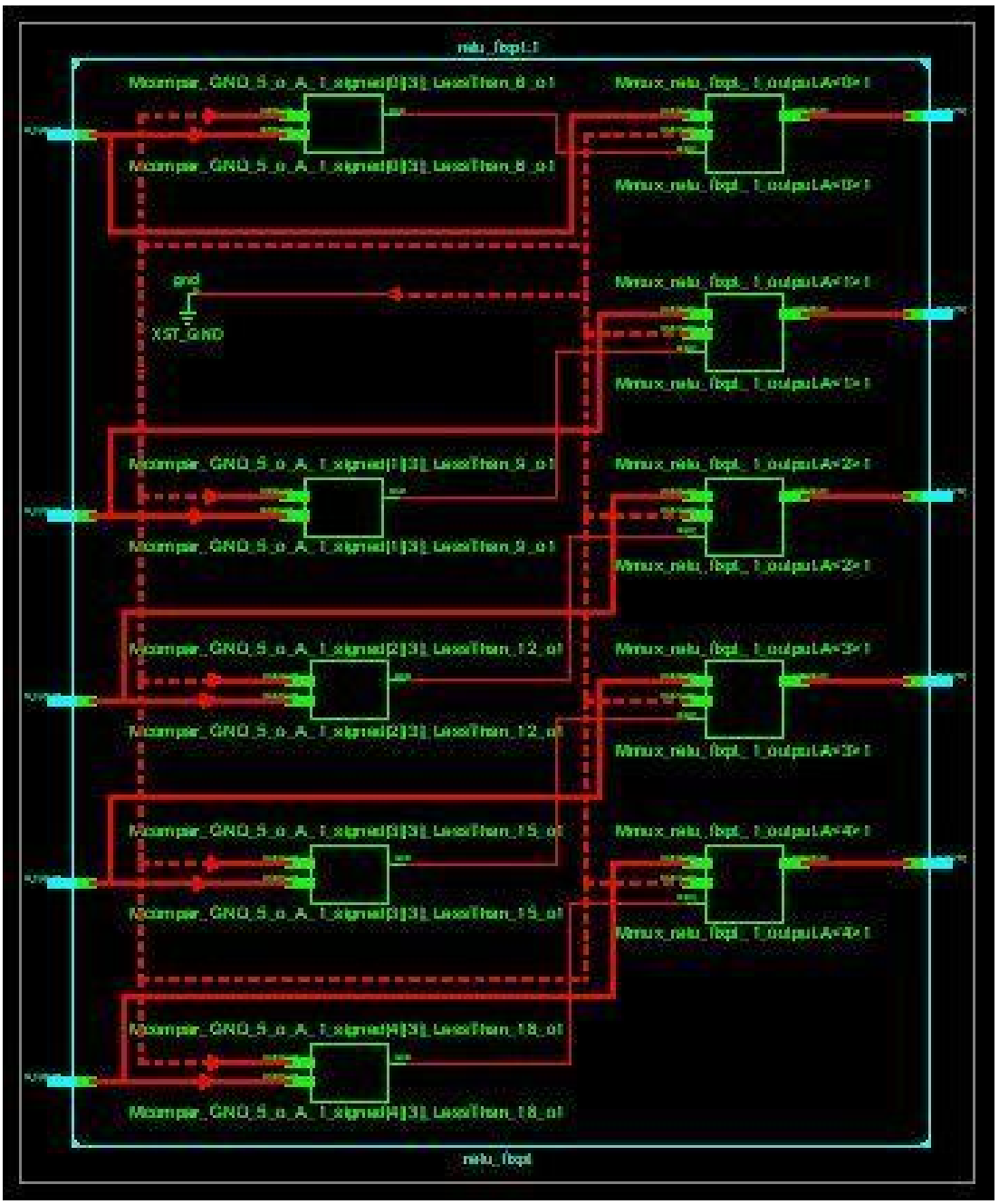

Figure 5: RTL diagram for ReLU implementation.

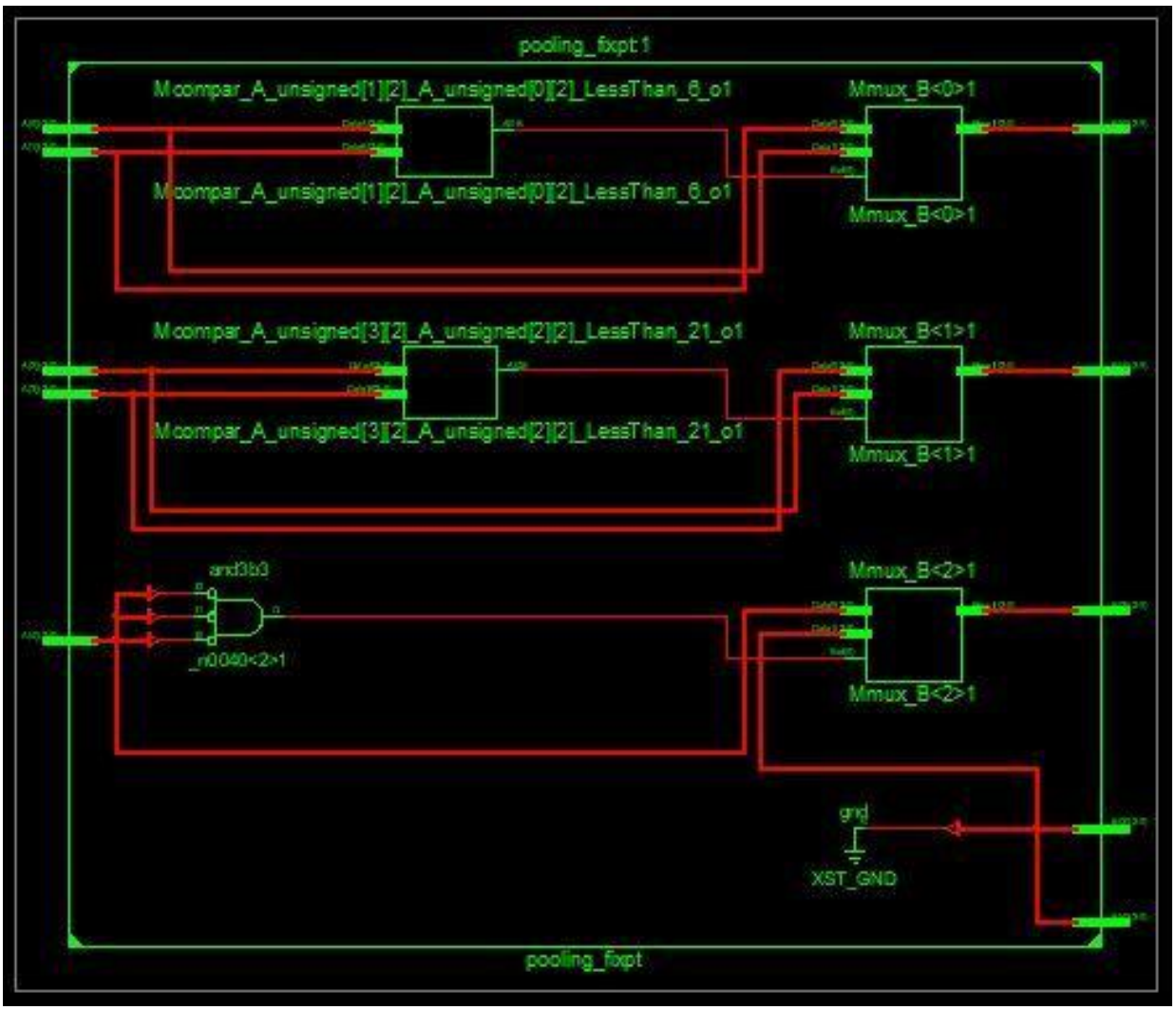

Figure 5: RTL diagram for pooling layer implementation.

The above representations have been obtained through the generation of a test bench of the VHDL code on the Xilinx software platform.

6.2 GOAL OF IMPLEMENTATION

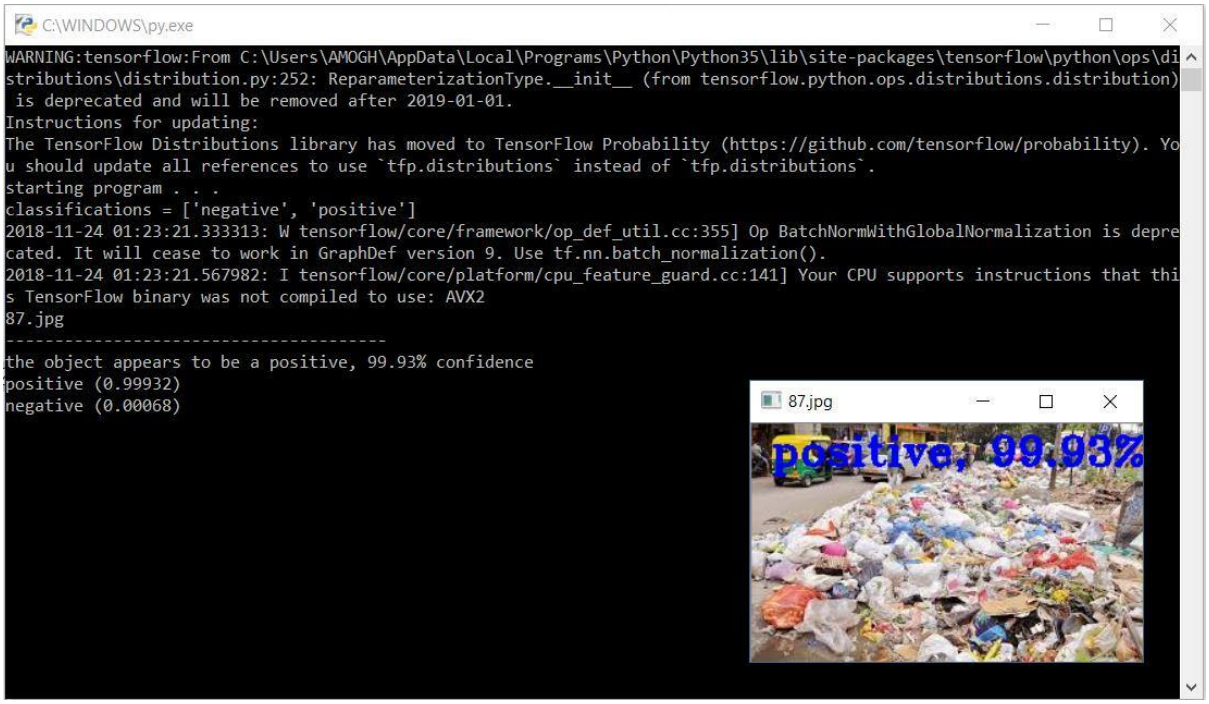

Figure 5: Result image for positive input.

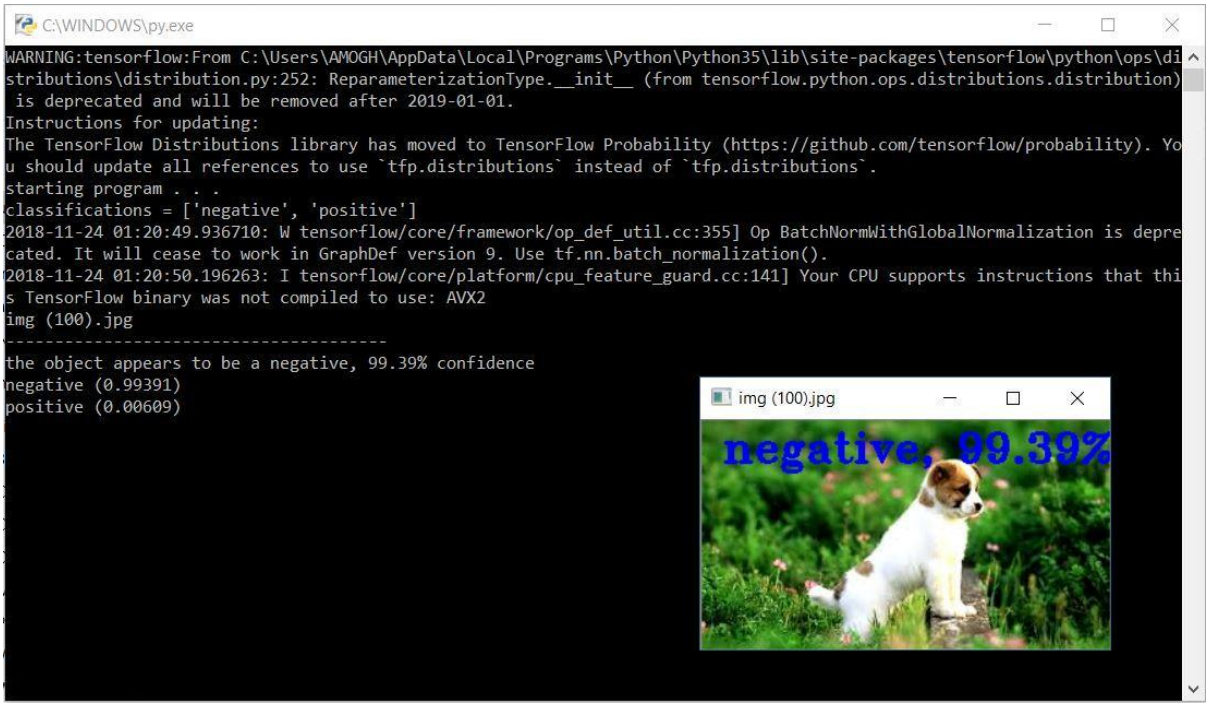

Figure 6: Result image for negative input.

The above images represent the outputs that are expected from the system on input of positive image (one containing garbage) and negative image (one not containing garbage).

The diagrams are the output of a basic CNN developed using Python, OpenCV, and TensorFlow.

6.2 CONCLUSION

The development of VHDL codes for the convolutional, ReLU, and pooling layers of the CNN has been done.

6.3 FUTURE SCOPE

The elements of a convolutional neural network, such as convolutional and pooling layers, are relatively straightforward to understand. Although simple, there are near-infinite ways to arrange these layers for a given computer vision problem.

Fortunately, there are both common patterns for configuring these layers and architectural innovations that one can use in order to develop very deep convolutional neural networks. A useful approach to learning how to design effective convolutional neural network architectures is to study successful applications.

Studying architectural design decisions developed for state-of-the-art image classification tasks (such as GoogLeNet) can provide both a rationale and intuition for how to use these designs when designing convolutional neural network models.

Hence, using the codes developed for convolutional, ReLU, and pooling layers as a base and developing an efficient stacking mechanism, a fully connected layer can be developed to classify images through a hardware-based device.

[1] F. Li, J. Johnson and S. Yung, “CS 231N Course Slides and Notes,” Stanford University. [Online] Available: http://cs231n.stanford.edu/
[2] F. Chollet, “Building powerful image classification models using very little data”. [Online] Available: https://blog.keras.io/building-powerful-imageclassification-models-using-very-little-data.html
[3] M. Ulicny, J. Lundstrom, and S. Byttner, “Robustness of Deep Convolutional Neural Networks for Image Recognition,” Intelligent Systems Department, Halmstad University, Box 823, S301 18 Halmstad, Sweden, Springer 2016.
[4] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.
[5] B. Bosi, G. Bois, and Y. Savaria, “Reconfigurable pipelined 2-D convolvers for fast digital signal processing,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, September 1999. ISSN 1063-8210.
[6] L. M. Russo, E. C. Pedrino, E. Kato, and V. O. Roda,” Image convolution processing: A GPU versus FPGA comparison,” Proc. Southern Conf. Programmable Logic, pages 1–6, March 2012.
[7] R. G. Shoup,” Parameterized convolution filtering in a field programmable gate array interval,” Technical report, 1993.
[8] F. Talbi, F. Alim, S. Seddiki, I. Mezzah, and B. Hachemi,” Separable convolution Gaussian smoothing filters on a Xilinx FPGA platform,” In Proc. Int. Conf. Innovative Computing Technology, 2008.
[9] S. Yu, M. Clement, Q. Snell, and B. Morse,” Parallel algorithms for image convolution,” In Proc. Int. Conf. Parallel and Distributed Techniques and Applications, 1998.