WORCESTER POLYTECHNIC INSTITUTE

DOCTORAL THESIS

NDTNet: An End-to-End Design of an Optical Nondestructive Evaluation System for Metallic Objects on FPGA

Author:
Raunak BORWANKAR

Supervisor:
Dr. Reinhold LUDWIG

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Center for Imaging and Sensing
Electrical and Computer Engineering

May 2, 2022

Abstract

The automated and rapid inspection of high-volume metallic components has been a key area of industrial research for many years. A main challenge is to nondestructively detect faulty components on a consistent basis without human intervention. Although a host of nondestructive evaluation (NDE) methods exhibit high resolution, they are difficult to implement on the factory floor; the approaches encompass acoustic resonance, magnetic particle, laser scanning, eddy current, and ultrasonics. Unfortunately, these inspection techniques are usually tailored to specific industrial components and flaw t ypes. They f urthermore r equire s ophisticated t est arrangements which are not scalable to high-speed inspection.

As computer systems have evolved, research in computer vision based optical NDE methods have become feasible. These optical approaches can be broadly categorized into two parts, computer vision based and neural network based. While computer vision based optical NDE methods are highly efficient, t hey r equire human intervention in identifying critical flaw f eatures. M oreover, c omputer vision algorithms are highly sensitive to the manufacturing environment, particularly fluctuating light conditions and background noise. As an alternative, convolutional neural networks (CNNs) are increasingly employed in nondestructive optical inspection. Even though state-of-the-art CNNs have proven efficient w hen coupled with transfer learning, they are generally not optimized for rapid testing of production samples on low-cost, dedicated hardware platforms.

In this dissertation, we propose a general workflow t o a utomatically construct and optimize a CNN architecture using existing computational frameworks. The proposed approach is tested with different production datasets for surface flaw inspection. Based on a novel sensing arrangement, we achieve precision of nearly 99% for both datasets. Out of other convolution acceleration techniques like fast finite impulse response (FIR) and fast Fourier transform (FFT), we use Winograd based accelerators to speed up the convolutions. State-of-the art Winograd based accelerators are usually designed to perform stride-1 convolutions. In this research, we developed a Winograd accelerator which can perform both stride-1 and stride-2 convolutions on the same hardware platform. The novel hardware implementation is 3.25

times more computationally efficient for stride-1 convolutions when compared to a standard approach, while it is 1.44 times more efficient for stride-2 convolutions. It therefore lends itself as a highly flexible, scalable and rapid inspection methodology suitable for many high-volume production environments.

Acknowledgements

Most importantly, I would like to extend my sincere gratefulness to Dr. Reinhold Ludwig for being such an encouraging advisor. He has always supported me and provided me with great advice. Through his actions, he motivated me to trust my instincts. I am fortunate to have an advisor of such immense knowledge and patience.

I would like to express my sincere gratitude to Dr. Xinming Huang, Dr. Ziming Zhang and Dr. John Sullivan for taking time out of their busy schedule to serve on my PhD dissertation Committee. I am honored to have such a reputable committee.

I thank Dr. Gene Bogdonav for his image acquisition system and extraordinary feedback which has helped shape this work significantly. I would also like to thank Jason Lott and his team at US Synthetic, Provo, UT, for providing the PDC data set as well as helping with labeling of the images.

I express thanks to my parents, Mukund Borwankar and Rashmi Borwankar and my sister Swara Joshi who have been cheering me and stood by me through the good and bad times. Finally, I thank my fiancé Akshata Karekar who has always aspired me to believe in myself and supported me throughout my PhD journey

Contents

List of Figures

List of Tables

I dedicate this PhD thesis to my parents, fiancé and my sister.

Chapter 1

Review of Existing NDT Techniques

Due to increased recall of consumer product particularly the automotive industry, nondestructive inspection of metallic components has become a center of attention in these industries. Many companies have often relied on visual/human perception to decide the quality of metallic components. However, visual inspection alone of complex metallic components can prove to be inconsistent in deciding the quality of the product. Visual inspection requires high concentration as well as consistency from the inspectors. The computer vision or artificial intelligence based systems can achieve consistency which is unmatched by human inspectors. Due to significant improvement in artificial intelligence algorithms and ease of deployment, extensive research varying from image retrival to autonomous driving is being done. Inspection of high volume complex metallic components remain an active area of research.

1.1 Review of Existing NDT Methods

In this section, we will review some of the existing nondestructive testing (NDT) techniques that are already in use in the manufacturing industry. The most common NDT techniques used today in industry is acoustic resonance in [7]. Apart from acoustic resonance, we discuss optical testing, laser testing, eddy current testing, and magnetic particle testing. Nondestructive testing techniques generally follow three steps for the diagnosis of defects in metallic components, i.e., detection, localization, and characterization. Acoustic Resonance Testing (ART) uses the vibrational characteristics of an object to find defects. Acoustic resonance testing is used in a wide range of products and it is a rapidly growing NDT method. The references [7, 8] proposes a ART based method that performs rapid detection of defects like cracks, other manufacturing-related flaws, within less than a second per component. ART is usually done as a whole-part test, which means that measurements taken at one place can show defects everywhere on the part. ART can make quantitative, objective judgments and can be completely automated, which can eliminate human error. ART uses the fact that a part's physical structure makes it to have a unique and distinct set of harmonic and characteristic frequencies. When a part is subjected to an external force, or dropped upon a suitable surface it will resonate at a particular frequency which is specific and unique to that part, which are also called natural modes. These modes are impacted by cracks and other material related defects which causes a change in the resonant frequencies. Every defective part will cause a vibration to change than a flawless parts with same physical properties will have the same vibration properties. After a hammer impact on the component, the characteristic frequencies of the part will be excited, while every other frequency will attenuate. This resonance is captured with a microphone to get the resonance spectrum. By measuring the location and amplitude of many peaks of the part's response and then comparing these peak values to an existing database of acceptable values, defective parts are separated. ART can also be used to find components with cracks, out-of-tolerance dimensions, and voids among many other things. While ART cannot determine the cause of rejection, with the help of appropriate transducer sensitivity and software we can reliably find faults in a component. ART is being used on some automotive production line along with power transformer, piping sections and many other applications to ensure 100% compliance with specifications. Some of these applications are discussed below.

In reference [1] proposes a nondestructive, acoustic emission method of detecting cracks caused by an automotive stamping process. In order to obtain high quality and reliable components, continuous monitoring of crack detection based on nondestructive tests is required in manufacturing process. Additionally, it is important to use an optimized method that will automatically adapts to its surrounding environment and to the facility. In [1], they propose a comparative method which measures the amplitude distribution between cracked and un-cracked components. The cracked components from the automotive stamping release low elastic energy, a filter is used which sets at a particular frequency. An amplitude distribution method is then applied with ratio conversion to get digitization. In another experiment of inspection of the spot weld quality, reference [2] uses a real-time, portable, through-transmission and continuous wave ultrasonic non-destructive evaluation system. They used an ultrasonic transducers which are attached to the top as well as bottom electrode arms of the spot welding machine. An improvement is seen in the results demonstrated when compared to other destructive tests.

Partial Discharge (PD) detection which is based on ART technique, is achieving importance due to its many advantages like being on-line nondestructive, localization of PD sources. Acoustic resonance in transformers and reactors is usually associated with high degradation of the core as well as tank assemblies and the insulation system. Detection along with analysis of such resonance can lead to early detection of defects in the power equipment. In reference [3], the authors identify the response of acoustic resonance signals from utility inductive reactors to develop a monitoring system. A number of reactors are subjected to acoustic resonance measurements. The results obtained were compared with concurrent dissolved gas in oil analysis. Developing a monitoring system with the help of results is discussed in [4] with the help of experiments conducted with testing of power transformers. The ART data in addition to PD signals, may contain acoustic resonance signals. These signals come from different sources like thermal, electrical, mechanical, etc. For a realistic analysis of ART data, it is important to remove noise. Acoustic resonance signals of different characteristics are generated by different defects, which discriminate PD from noise. A precise knowledge of the individual characteristics of the sources is required. Prior knowledge of known defects in power transformers has greatly to gather ART characteristics as well as sperate out noise from PD source.

Next, we look at other areas where acoustic resonance is highly successfully and used for NDT. In [5], the authors propose a higher order statistics (HOS) to classify acoustic resonance events for a ring-type parts such as steel pipes. The diagonal bi-spectrum allows for separation in the original deformation from the reflections produced in the suppressed chord. A cumulate-based independent component analysis (ICA) is used before the bi-spectrum. Such an algorithm will suppress the mutual influence in the sensors. For high-pressure stainless steel pipe defect evaluation, an acoustic resonance along with the hydrostatic testing has been successfully implemented by reference [6]. This technique uses only 100 kHz and 300 kHz frequencies of the acoustic resonance sensors that are mounted on the pipe. Many hydrostatic pressures ranging from zero to 120 bars are applied to the pipe and ART signals are captured on a computer. The processed AR parameters such as crest factor and AR energy,are used to indicate growth in the pipe material or crack initiation. Several micro cracks have been found by micro structure test in both defected pipes. On similar lines, [9] proposes a crack detection system for synthesized metallic pieces which can be done in real time. The signal measured with the help of ART is then passed to a signal-processing algorithm to conduct spectral analysis. The spectrum is obtained by applying an FFT based chirp algorithm and from the resonance frequencies a minimum Euclidean distance algorithm, controlled by the false alarm probability, is applied to parts that are cracked or defective pieces from the production system. The system provides satisfactory results when tested in different environments and on different objects. Hence, sufficient sensitivity is a key advantage of AR technique when applied to detect a micro cracking propagation.

Another NDT technique we will discuss is in the form of Magnetic particle Testing (MT) which uses ferromagnetic materials after being magnetized. The discontinuity in the component generates a leakage magnetic field. When the lines of magnetic induction either leave or enter the surface, magnetic field is created. Under appropriate lighting conditions, the deformities, their location and severity can be seen when the defective object absorbs magnetic particles. Therefore, in ferromagnetic material based components, magnetic particle indications are used to identify defects. Exposed small defects which cannot be seen by naked eye or under microscope or near-surface flaws that are not exposed but are just few millimeters under surface can be detected by magnetic particle testing as shown in [10]. Magnetic particle testing is more suitable to area defects like cracks formed due to welding, rolling, quenching, plating, casting, grinding, fatigue or forging. It is not very effective on volume type flaws such as incomplete fusion, gas pole or slag. Magnetic flaw detection can be shown in several ways. The methods which use magnetic particles are called magnetic particle testing, while those that do not use magnetic particles are usually known as magnetic leakage. In this form of testing, predominantly induction coil, Hall element or magnetic tube is used. This is a cleaner way of testing compared to using magnetic particles but not as intuitive. Another NDT technique we would like to discuss is eddy current testing. In the mid 1990s eddy current testing was predominantly used on steam generator tubes [11]. The structural integrity of a metallic component can be evaluated using eddy current testing by estimating the parameters associated with defects. Most eddy current testing based algorithms propose to use Neural Network (NN) for classification. It is difficult to use just the statistical regression and accurately measure the parameters from two interdependent, complex and simultaneous systems. Hence, use of artificial neural networks (ANNs)is explored to deal with such complex relation between gathered data and defect properties. In reference [11], inspection of an in-service nuclear steam generator and its surrounding support structure is performed. The pulsed eddy current data was obtained using a single driver with an array of eight pick-up coils configured for inspection of Alloy-800 SG tube fretting, accompanied by tube offset within a simulated corroding ferromagnetic support structure. Modified principal component analysis was used to process time-voltage data. This helped to reduce the data dimensionality. The scores were then fed to a NN which would simultaneously target four parameters associated with hole size, fret depth, support structure, and tube off-centering in two dimensions. Similarly, in references [12, 13] they have used a NN along with eddy current testing for defect classification. Particularly in [12] proposes to use discrete Fourier transform, wavelet transform and principal component analysis (PCA) to extract relevant features which will be further passed on to the ANN. The synthetic data set generated with the help of finite-element modeling of eddy current probe is used to extract features which will then be fed into a NN. In [13], they reliably estimate the dimension and shape of a crack in conductive materials using eddy current principle and NN based post processing. A comparison between neural network and support vector machine (SVM) based classification is done after the design and tuning stages of eddy current testing. To solve the inversion problem in eddy current testing, reference [18] proposes a neural network mapping approach. They use a data fusion, a data fragmentation technique and PCA data transformation to solve the inversion problem.

Continuing with the steam generator tubes testing topic, [14] proposes an examination of support structures coupled with maintenance programs that can prevent and ameliorate these effects by implementing pulsed eddy current in combination with principal components analysis (PCA). Conventionally, inspection of eddy current technologies are extended to be used for activities like detection and scaling indications from wall loss, cracks, other degradation modes in the tubes, frets at supports and also for assessing support structure conditions. Nevertheless, there are limitations for these methods when multiple degradation mode are present at the same time, or when added with fouling. A similar example is an eddy current NDT of thick ferromagnetic tubes with average wall thickness calculated with an exciter coil. In [16] studies the consequences of the excitation frequency and the interval between the detected signal sensitivity coil to the tube properties, a foundation for using pulsed eddy current (PEC). It is done inorder to defeat the key disadvantages of these eddy current tools. With continuous frequency spectrum and time gap of direct and remote zone signals, the pulsed eddy current, simultaneously measures the average wall thickness and inner diameter with exciter separated by 1 detector coil by couple of tube diameters. Few other applications which are employed with pulsed eddy current NDT are explained in [15, 17, 19, 20, 21]. In reference [15], an application of pulsed eddy current NDT scheme is testing aging aircraft wing. The current methods of crack detection with bolt hole eddy current requiring fastener release, effects confirmatory damage hence [15] proposes an NDT where pulsed eddy currents induced in the aluminum wing structure are probed in center over a metallic fastener that assist in recording and analyzing response signals with the assistance of modified principal components analysis and showing breakup of groups of PCA scores from fasteners along with and without borehole notches. Inorder to achieve a relative distance measurement between scores with cracks and without cracks, a cluster analysis method Mahalanobis distance (MD) is used. Usage of multiple sensors in pulsed eddy-current detection for three-dimensional (3-D) subsurface flaw imaging is discussed in [17]. To expel the variations among devices based on the Hall Effect, a normalization technique has been proposed in [17]. A feature supported principal component analysis technique for multiple sensor fusion has been extracted by an orthogonal information here. Features of multiple projection coefficients achieved through these sensors are used for 3D design and measurement.

Pulsed eddy current is often preferred over eddy current due to its many features like wide frequency range and large exciting currents. Pulsed eddy current testing is especially effective for reconstruction of stress corrosion cracks (SCCs). In [19], the authors investigates various ways to reconstruct SSC profile with the help of stochastic optimization of neural network or simulated annealing on the pulsed eddy current signals. In [20], the authors introduces a pulsed eddy current thermography to detect surface cracks. Pulsed eddy current thermography uses infrared camera to pickup the heat developed by eddy current distribution. This helps to identify defects over a wide area in a mater of milliseconds. Reference [20] studies the effect of pulsed eddy current thermography on carbon fiber reinforced plastic (CFRP) materials. They observe the heating pattern of CFRP material when subjected to directional electrical conductivity. One more example of pulsed eddy current thermography is shown in [21]. Stainless steel is important in many industries, hence it is key to understand the stainless steel weld which is the weaker section of the component but plays an key role in structure integrity. Reference [21] also introduces a component analysis to remove any influence of weld irregularities which can play a role in improving the defect information for pulsed eddy current thermography testing.

The fourth method we would discuss is laser based NDT. Typically, laser inspection is done in tubular structures like pipes. In reference [22], the authors propose a non-contact laser-based inspection method to inspect the inner surface of mini-diameter pipes. This inspection is based on a position-sensitive detector (PSD). A light is projected onto the inner wall of the pipe with the laser beam by reflecting it using two mirrors. This will create four current signals when the light spot is detected by the 2-D PSD. The inner wall can be scanned by the laser beam with the help of a micro motor. Then a data segmentation and least squares fitting algorithms are used on the co-ordinates obtained to reconstruct the curve section. To inspect flaws in inner surface of a long curved mini-diameter pipe, a mini-robot is used. Based on similar concept, references [23, 24] present a new automated inspection methodology for pipes. A standad CCTV camera is attached to a low-cost laser profiler. Both the references [23, 24] acquire images of pipe wall through light projection. The only difference between these two approaches is they acquire image by laser based profilers but are analyzed differently. In [23] uses artificial neural network to analyzes the image data. While, [24] uses the intensity distribution from the image and pass it on to artificial neural network for fault detection. Similarly, in [25] the authors propose a method using triangulation to locate a spot on surface of a pipe using laser spot arrays in a 3D co-ordinate reference system of the camera data. Reference [26] introduces an thermal excitation based infrared thermography from a CO2 laser of 10.6m wavelength. In their work, the main focus is of finding flaws in bio ceramics through thermal excitation. The thermal excitation is achieved by focusing the laser beam from vertical to horizontal direction. As a basis, they used a temperature of a fault free sample and compared it with other samples to find faulty and fault free sample.

The industry is moving towards optical image based NDT. Over past few years, a trend in optical NDT is seen. In [29], an automated 3D optical measurement system is proposed. Typically, coordinate measurement machines (CMMs) in industrial inspection provide accurate measurement. However, the CMMs are very time consuming. The authors of reference [29] used a pixel- to- pixel sensor calibration. This can acquire a patch-by-patch data of an automotive part. Further the patch is inspected for defects, which saves a lot of time. Optical NDT is predominantly used to detect the surface flaws. These surface flaws can be on wind turbine blades,pipes or automotive parts. Reference [27] studied that it is possible to detect and classify small cracks with hair like thickness. The optical NDT setup is not sensitive to crack orientation or the angle at which the camera is setup. The uneven background is negated with the help of canny edge detectors as it uses threshold values. Using both the sobel and canny edge detectors increases the accuracy of detecting cracks as it reduces the noise in the image. Most of the conventional NDTs are time consuming, [28] proposes a new digital image processing based NDT approach which can potentially decrease the inspection time. The image is analyzed in the transform domain using the Counterlet Transform (CT) and Discrete Cosine Transform (DCT). The counterlet transform uses iterative filter banks and creates a 2D spectrum of fine slices. The directional energy components of the decomposed sub-bands are recorded. These energy values are used to differentiate between the faulty and fault free part. In another approach of discrete cosine transform, the 2D- spectrum is divided into high and low frequency components. The feature vectors are basically first order moment of these components.A correlation based classifier is needed to classify the part under test into a faulty or fault free bins. In the experimental results, it was seen that discrete cosine transform performs better than Counterlet transform. As seen in the past few examples many of the feature extraction techniques used are observed in algorithms like face recognition or image retrieval. Hence, authors in [30] used face recognition algorithms to perform optical NDT. The the proposed methodology primarily focuses on running the algorithm in real time by maintaining its computational efficiency in high volume manufacturing environments. For training, the algorithm uses previously classified images. These images of parts under test are then classified into two bins, faulty and fault-free. In [30] the authors proposes a method to detect surface breaking cracks which combines Discrete Cosine Transform with Fisher's Linear Discriminant Analysis.

As discussed earlier, the optical NDT is typically used for surface flaw inspection. In manufacturing industries, identifying surface flaws is key to produce high quality components. Evolution in cameras, image processing algorithms and computer systems, made use of optical inspection techniques for surface defect inspection commonplace. Few examples of optical NDT based surface law inspection are steel products, rail tracks, screen glasses, etc. [31, 32, 33]. A lot of details are captured by cameras and the image processing algorithms can be used to extract features and classify surface flaws. Li et al [33] proposed an automated optical inspection system fo rail components inspection like anchors, tie plates and ties. The system had four cameras which captured images at 20 frames/sec. A entropy-rate clustering and shape contraint based real-time optical inspection system is shown in [34]. They divided the captured image into multiple sections which is then classified into multiple flaws. Tao et al., [35] proposed a surface inspection system for larger-aperture optical elements. In this approach they combined bright-field and dark-field imaging system to achieve high efficiency as well as high accuracy. The methods discussed above are very specific to the components under test and cannot be scaled or generic use.

The optical NDT methods can be classified into two categories; deep-learning methods and conventional methods. The conventional methods are mostly two-fold. Fist they use computer vision algorithms for feature extraction and second, they use classifier for classification. In [36, 37], the features from the image are extracted using histogram curve and edge detection. In reference [38] use a rotational invariant measure of local variance operator for feature extraction and support vector machine as classifier. The accuracy of the computer vision based optical NDT methods is directly related to the quality of image and features it possess. However, there is still a human reliance relative to the selection of features. Furthermore, the computer vision based NDT methods cannot be generalized as they are customized for a particular inspection task.

In recent years, deep learning based optical inspection methods are being developed for feature extraction and classification. In [39, 40] proposes a deep learning based optical inspection method. A photometric stereo image data set of steel components is used to train the convolutional neural networks for defect detection by [41]. The major obstacle for deep-learning techniques is availability of large dataset for training. This issue is solved by using transfer learning. in tansfer learning, the neural networks are trained on generic datasets with millions of images initially. Then they are re-trained on target dataset. This approach is used by [42, 43] for surface defect detection, where they initially use generic dataset to train the neural network and then retrain it on target dataset. In transfer learning, the general rule of thumb is, the features in generic and target dataset should have some similarities. Otherwise, it is difficult to achieve high classification accuracy with this method. Another such CNN based optical NDT is shown by Natarajan et al. [44]. A optical defect inspection and localization method is proposed by [45], where they use a sliding window-based CNN approach. A faster R-CNN based optical NDT is used for multi-type defect detection by [46]. In reference [47], a deep convolution neural network (DCNN) is used for surface flaw inspection. Here one DCNN is used for localizing key defective areas, while another one is used for defect classifications. Similarly, Xian et al. [48] proposed a twofold methodology in which they cascade an auto encoder for localization of fault and a CNN for classifying faults.

1.2 Motivation

The motivation behind this dissertation is to develop and implement a highly efficient as well as consistent optical NDT algorithm for surface flaw inspection of metallic parts. The algorithm should be highly effective but at the same time easy to implement on the factory floor. Even though many NDT approaches are proposed in the past, like magnetic particle laser scanning, acoustic resonance, eddy curent testing for nondestructive testing, these algorithms are highly specific to the part under test and cannot be generalized easily. These algorithms require highly sophisticated equipment which are often expensive and complex to use. We wanted to concentrate our effots on optical NDT which has shown to be successful for surface flaw inspection in recent years. In this thesis we focus on developing a convolutional neural network based optical NDT. The thesis makes the following contributions:

(1) A novel image acquisition process that can efficiently image a cylindrical object as well as compensate for variable lighting effects on the metallic surface.

(2) A classification scheme suitable for industrial implementation with greater reliability and accuracy using a compact CNN architecture. The compact CNN is trained by employing transfer learning and fine turning.

(3) An Evaluation procedure that can be performed on samples of poly-crystalline diamond cutter (PDCs) and that can be compared with conventional methods.

(4) A compact CNN whose low number of parameters and small weight size can be seamlessly implemented on cost-effective embedded devices like FPGAs and TPUs.

(5) Explore a compact CNN using automated machine learning techniques that achieves high classification efficiency.

(6) Use model compression methods like parameter pruning and weight quantization to compress the compact CNN to fixed point form.

(7) Develop a FPGA based convolution accelerator that will deploy compact CNN for nondestructive testing.

1.3 Organization of Thesis

This thesis is organized into six chapters. After reviewing the state-of-the art of various inspection systems in Chapter 1, Chapter 2 gives a brief introduction of convolutional neural network and algorithmic optimizations iuse to accelerate CNN implementation on hardware. Chapter 3 discusses various CNN optimization algorithms used for CNN model compression. Chapter 4 presents a unique hardware architecture to implement CNN on FPGA. In chapter 5, we review the experimental setup and demonstrate results of the proposed approach. Finally we conclude in Chapter 6 and provide an outlook for future work.

Chapter 2

Convolution Neural Networks

The increasing requirement for extracting useful information from raw data like images, videos, speech, text has dominated recent research. Convolution Neural Networks (CNNs) are at the heart of this research due to their impressive ability to learn from the raw labeled data. Hence CNNs are used as a de-facto in many machine learning applications like image/speech/text classification, image segmentation, etc. CNNs come at a huge computational cost as some recent CNNS require up to 32 GOP/s to classify one image frame. Such high computational cost requires a dedicated acceleration platform. Graphics processing units (GPUs) are widely used to implement CNNs due to their ability to parallelize the workload. Even though GPUs offer high performance throughput, they are known to be power hungry. In recent years, Field Programmable Gate Arrays (FPGAs) based CNNs have been used in embedded applications as well as High Performance Computing (HPC) data centers due to their power efficiency.

Although GPUs can provide high computational performance, the recent research is moving towards FPGA based CNN acceleration techniques for two reasons. First, due to advancement in FPGA technology, FPGAs can deliver high performance throughput of the order 9 TFLOP/s which is comparable to the GPUs while being power efficient. Secondly, current CNNs research is driving towards making CNNs compact and sparse. These factors make development of FPGA based CNN accelerator more viable than GPU based.

TABLE 2.1: Tensor Sizes of Convolution Elements for a given Layer l.

Notation	Name	Tensor Size	Description
X	Input Feature Map	$B \times N \times N \times P$	B: Batch Size; N: Height and Width; P: No. of Input Channels
Y	Output Feature Map	$B \times O \times O \times K$	B: Batch Size; O: Height and Width; K: No. of Output Channels
θ	Learned Filter Kernel	$M \times M \times P \times K$	M: Height and Width; P: No. of Input Channels; K: No. of Output Channels
β	Learned Bias	$1 \times 1 \times 1 \times K$	K: No. of Output Channels

2.1 Layers in Convolution Neural Network

The CNN is build by pipelining different layers to create a deep, feed-forward, sparsely connected network. Every layer has a set of input and output data called Feature Map (FM). Before diving deep into various convolution layers, let us understand some of the prerequisites. First, we will understand what a *tensor* is. Mathematically speaking, a tensor is an object that can establish a relationship between group of objects related to a vector space. A scalar value is a zeroth order tensor, a vector is first order tensor. Similarly, a matrix is second order tensor, which makes a color image a third order tensor. A RGB image is a tensor of dimension $W \times H \times 3$.

With convolution we are extracting features like edges from the image. We typically use odd dimension filters so as to keep the convolutions around the center pixel. But this can throw away any information on the edge of the image. Hence, padding is done along all the edges of the image to preserve the information. We pad the image with zeros and a padding parameter p which represents number of rows/columns of zeros added on each side. A stride is a step taken in convolution. A stride s can reduce the size of the output by a factor of s. Once we have figured out the stride and the padding we can define the convolution product between a image tensor and a filter kernel.

The tensors used in the subsequent sections are summarized in the Table 2.1

2.1.1 Standard Convolution Layer

Standard Convolution (SC) layer involves convolution of input feature maps with a specific filter followed by summation of the results into one channel. As seen in Fig. 2.1, the output feature maps are an inner product of the filter and input feature maps. The feature extraction is carried out by applying filter kernels θ^{conv} to the input layer X^{conv}. The input has a certain depth P. For first layer, if the input image is RGB, the depth P is 3. For layers after first layer, the input is output feature map Y^{conv} of previous layer. When a 3D filter kernel θ is convolved with a 3D input tensor X, a 2D feature map is generated. If K such 3D filter kernels were used, it would give a 3D output feature map Y. Most convolution layers add a bias β to the output feature map Y. The mathematical computations involved in standard convolution layer are given in Equation 2.1.

$$Y^{conv}[b,o,o,k] = \beta^{conv}[k] + \sum_{p=1}^{P}\sum_{m=1}^{M}\sum_{m=1}^{M} X^{conv}[b,p,o+m,o+m].\theta^{conv}[k,p,m,m] \quad (2.1)$$

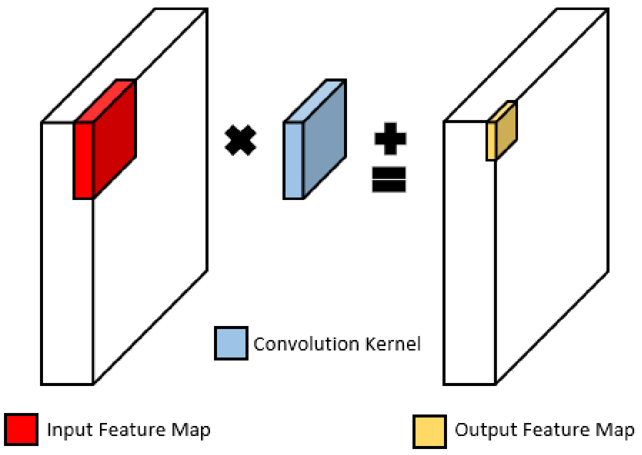


FIGURE 2.1: Standard Convolution Layer (SC).

2.1.2 Depthwise Separable Convolution Layer

As the name suggests, the standard convolution (SC) is factorized into two separate operations. The first operation is depthwise convolution (DC) followed by pointwise convolution (PC). Fig. 2.2 depicts the depthwise, and pointwise convolution schemes. In DC layer, every input feature map convolves with its respective filter and generates its corresponding output feature map. Pointwise separable layer is

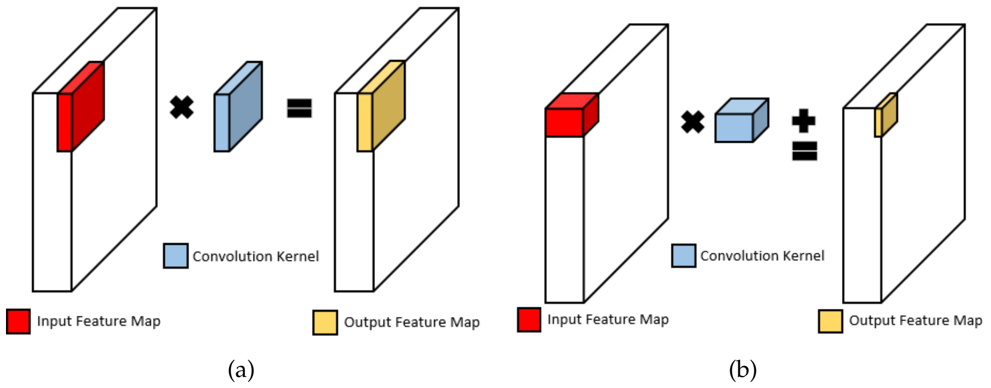


FIGURE 2.2: Depthwise Separable Convolution Layers (DSC). (a) Depthwise Convolution Layer (DC), (b) Pointwise Convolution Layer (PC).

similar to convolution layer but it only uses filters of size 1 x 1. Notice in Fig. 2.2, a convolution layer can be formed by cascading a DC followed by a PC layer. Such cascading of layers is called a Depthwise Separable Convolution (DSC) layer. The purpose of using depthwise separable convolution over the standard convolution scheme is to significantly reduce, both the number of operations and parameters.

We will show mathematically how the depthwise separable convolution layer can reduce the number of parameters and operations as compared to standard convolution layer. As denoted in Fig. 2.2, for an input tensor of N x N x P and a kernel size of M x M x P x K, the number of weights (W_{SC}) and number of operations (O_{SC}) needed for the SC (for stride = 1) is given by:

$$W_{SC} = M \times M \times P \times K \tag{2.2}$$

$$O_{SC} = N \times N \times M \times M \times P \times K \tag{2.3}$$

Similarly, for depthwise separable convolution, the weights and number of operations become

$$W_{DSC} = \underbrace{M \times M \times P}_{\text{DC}} + \underbrace{P \times k}_{\text{PC}} \tag{2.4}$$

$$O_{DSC} = \underbrace{N \times N \times P \times M \times M}_{\text{DC}} + \underbrace{N \times N \times P \times K}_{\text{PC}} \tag{2.5}$$

The depthwise separable convolution effectively reduces the parameters such as weights and number of operations by a factor of:

$$O_{RF} = \frac{O_{SC}}{O_{DSC}} = \frac{M^2 \times K}{M^2 + K} \tag{2.6}$$

Typically, for a kernel size of M = 3, the reduction factor in number of operations and weights is about 8 to 9 times.

2.1.3 Fully Connected Layers

Fully Connected layer (FC) as the name suggests connects each node to all the nodes in adjacent layers. This is used to establish a linear transformation between the input and out vectors. As shown in Fig. 2.3, each connection between two nodes represents a weight. The fully connected layers are deployed at the end of convolution pipeline when a CNN is used for classification task. The FC layer is simply a convolution layer without wweight sharing i,e, $N = M$. Similar to convolution layers, the fully connected layers are accompanied by non-linear activation layers. The mathematical representation of a fully connected layer is given by

$$Y^{FC}[b,o] = \beta^{FC}[k] + \sum_{p=1}^{P}\sum_{n=1}^{N}\sum_{n=1}^{N} X^{conv}[b,p,n,n].\theta^{conv}[k,p,n,n] \tag{2.7}$$

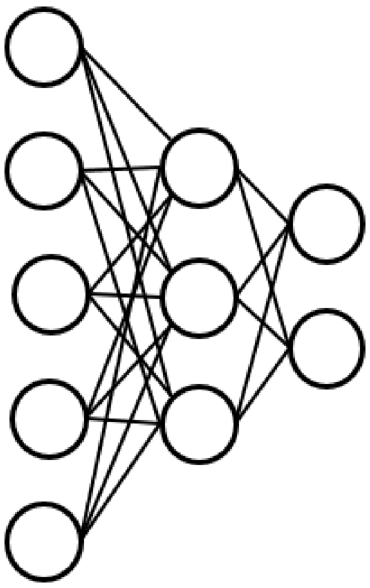


FIGURE 2.3: Fully Connected Layer (FC).

2.1.4 Activation Layer

Non-linearity layer also called as activation layer is used so that the CNN model learns rather than memorize the data. ReLU6 (rectified linear unit) shown in Fig. 2.4 is most commonly used. Other non-linear functions used are hyperbolic tangent and sigmoid. Mathematically, an activation layer is represented by

$$Y^{act}[b,k,o,o] = act(X^{act}[b,p,n,n]) \quad act = TanH, Sigmoid, ReLU \tag{2.8}$$

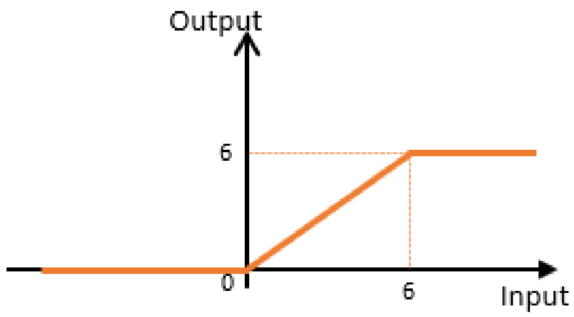


FIGURE 2.4: Activation Layer (ReLU6).

2.1.5 Pooling Layers

Pooling layers are used to reduce the spatial dimension of the layer. Average or max pooling layers are commonly used which outputs the average or maximum value of subarea. Figure 2.5 shows a simple 2 x 2 average and max pooling. The benefit of adding a pooling layer is not only to reduce the dimension of the model and computations in following layers but to also add a translation invariance. The pooling layers are mathematically represented as shown in Equation (2.9).

$$Y^{pool}[b,k,o,o] = pool_{(u,v\in[1:M])}(X^{pool}[b,p,n+v,n+u]) \quad pool = Avg., Max. \tag{2.9}$$

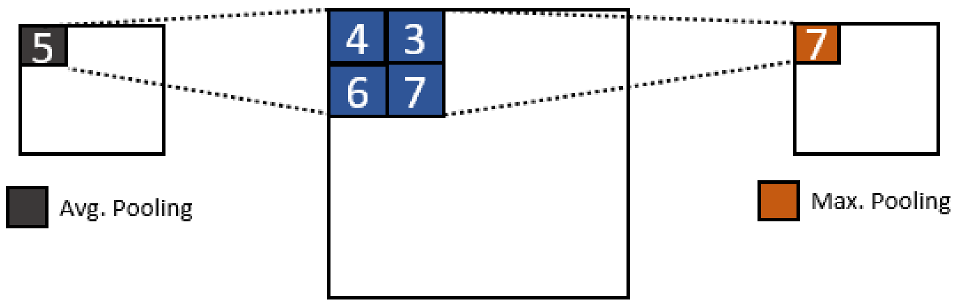


FIGURE 2.5: Pooling Layers (Max. Pooling and Avg. Pooling).

2.1.6 Batch Normalization Layer

Batch Normalization was introduced by to improve the time required for training. This was achieved by linearly shifting and scaling the given batch of Inputs to have no mean or unit variance. Batch normalization is alos used in Binary Neural Network (BNN) to reduce the quantization error as compared to arbitrary distribution. The mathematical equation for a batch normalization layer is given in (2.10). Here the parameters μ and σ are statistics collected during training. The hyper-parameters used in training are denoted by α, ϵ and γ.

$$Y^{BN}[b,k,o,o] = \frac{X^{BN}[b,p,n,n] - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \alpha \tag{2.10}$$

2.2 Parallelism in Convolution Neural Networks

Due to the large number of computations involved in CNN, it is extremely difficult for real-time CNN inference on low-power embedded devices like FPGA. In order to achieve this task, we exploit the extre concurrency displayed by CNNS. The different types of parallelism explored in CNNs are itemized below.

(1) Batch Parallelism: CNN can simultaneously classify different frames grouped together as batches. This will reuse the filter in each layer so that overhead of using DRAM to transfer the filter kernel is reduced. This accelerates the inference process.

(2) Inter-Layer Parallelism: Since CNN inference is a feed-forward network. It has multiple layers that are data dependent on each other. These layers can be pipelined by launching the next layer before the current layer is finished.

(3) Inter Feature Map Parallelism: Each output feature map in the convolution layer can be processed separately. We can implement W_K feature maps of Y^{conv} in parallel ($0 < W_K < K$).

(4) Intra Feature Map Parallelism: Multiple pixels of one output feature map can be processed parallely by calculating $W_o \times W_o$ values of $Y^{Conv}[K]$ ($0 < W_o \times W_o < O \times O$).

(5) Inter Convolution Parallelism: The 3D convolutions can be split as sum of 2D convolutions as shown in (2.11). These convolutions can be calculated parallely W_p elements at a time ($0 < W_p < P$).

(6) Intra Convolution Parallelism: The 2D convolutions can be processed in pipeline fashion. In this method $W_m \times W_m$ multiplications are implemented parallely ($0 < W_m \times W_m < M \times M$).

2.3 Algorithmic Optimizations for Accelerating FPGA-based CNN

Computational transformations are required in order to accelerate different convolution layers as well as fully connected layer. The transforms vectorize the implementations and decrease the MAC operations. These transforms are predominantly developed to be implemented on CPUs and GPUs in the form of software libraries like OpenBlas and cuBLAS for CPU and GPU respectively. However, some concepts can be used to accelerate CNNs on FPGA.

2.3.1 GEMM Transformation

A common method to implement CNN on CPUs and GPUs is to map the convolution as well as fully connected layers into General Matrix Multiplication (GEMM). Reference [49] used GEMM based approach to develop OpenCL based FPGA CNN Accelerator. In GEMM based works, as both the input feature maps and filter kernels are effectively 3D, they are flattened into a 2D matrix. A input convolution layer X^{conv} of size $PMM \times OO$ and filter kernel θ of size $K \times PMM$ to give output feature maps as shown in Figure 2.6

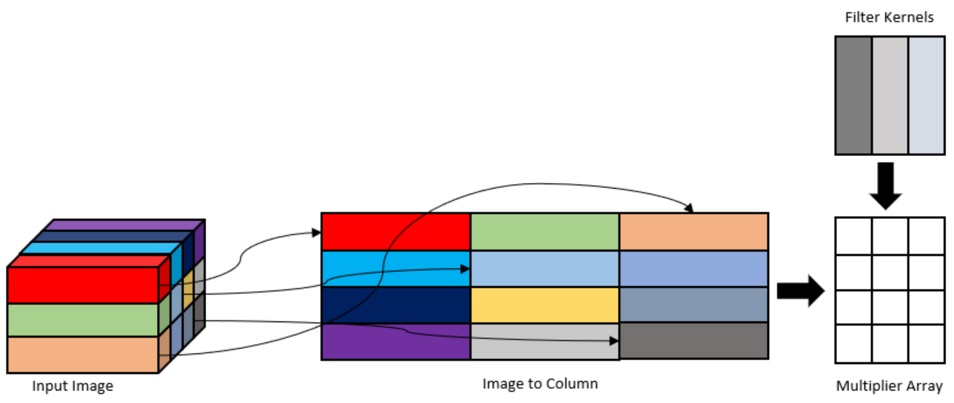


FIGURE 2.6: GEMM operation as a component of GEMM-based convolution algorithm.

Similarly, for fully connected layers a batch of feature maps are flattened into chw x B matrix. The feature maps of FC layer are mapped in such a way that they are loaded once per batch. This approach is used in [50, 51] to increase the computational throughput without increasing memory bandwidth. As the sparsity of the network grows the efficiency of this method increases.

GEMM-based algorithms convert convolution operation into matrix multiplication operation by performing im2col or im2row memory transformations. The memory transformations copy input pixels into a matrix row that are corresponding to output pixels. These input pixels and filter kernels are matrix multiplied to obtain output matrix. The memory overhead for such operation is trivial, for example, in a 3x3 convolution each pixel of the input feature map is repeated 9 times.

2.3.2 FFT Transformation

Fast Fourier Transform (FFT) is a very commonly used algorithm to convert convolutions into Elementwise Matrix Multiplication in the frequency domain. This is because the convolutions in the time-domain are multiplications in the frequency domain. The equation (2.11) shows how a convolution can be achieved using a FFT and a IFFT transform.

$$Y^{conv} = \mathcal{F}^{-1}(\mathcal{F}X^{conv} \times \mathcal{F}\theta^{conv}) \tag{2.11}$$

First, both the input feature map and filter kernel are transformed into the Fourier domain by performing a FFT of length (N + M -1). Then they are elementwise multiplied, which is followed by an inverse Fourier transform to obtain the output feature map in the time-domain.

The use of FFT for convolution reduces the number of arithmetic operations required and the arithmetic complexity is of the order o $(W^2 log W)$. However, using FFT also introduces complex number calculations. Typically, FFT transformation is used when the kernel size is greater than five [52, 53]. Reference [54] implemented overlap and add FFT method to reduce the computational complexity to O(W log k), but this can be applied to convolutions which have signal size larger than filter size which is the case with convolution layers.

The overlap and add FFT (OVA-FFT) based convolution method will sample the data into small portions and treat them independently for convolution with the filter. The data is segmented into small parts of size $X^{conv}[n - mL]$, where L is length of each part. Then convolution is performed by using FFT method and the outputs are aligned and added.

In Figure 2.7, we compare the FFT based convolution method and Overlap and Add FFT based convolution method. In OVA-FFT convolution method, minimal increment in intermediate memory is required compared to FFT base convolution.

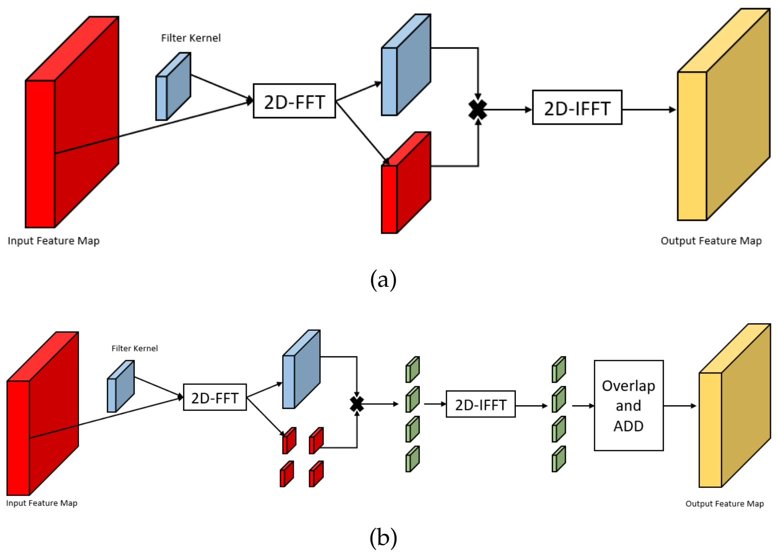


FIGURE 2.7: Different convolution schemes.(A) FFT-Conv with computational complexity of $O(N^2 log(N))$. (B) FFT-OVA-Conv with $O(N^2 log(K))$ where data dimension is N × N and filter dimension is K × K.

2.3.3 Winograd Transformation

Winograd minimal filtering was developed by [55] to exploit the overlapping in stride one convolutions. However, recent works have developed a way to implement Winograd transform on stride two convolutions as well. Winograd minimal filtering algorithm works best when the filter kernel is less than 3. Some studies [56] report acceleration of 8x compared to a GEMM based implementation of VGG16 on TitanX GPU. In the Winograd transformation the input feature map and filter kernel are transformed into the Winograd domain. The output feature map is the result of an elementwise matrix multiplication of the transformed filter kernel with the input feature map. But this is still in the Winograd domain and needs to be transformed back. A Winograd algorithm requires $(u+k-1)^2$ multiplications while standard convolution requires u^2xK^2.

Winograd algorithm has a few other advantages than just reducing the number of multiplications. The multiplications are effectively replaced by additions and shifting in the Winograd algorithm. But the transformation matrices can be generated offline and stored in LUTs [57]. The Winograd transform can utilize loop optimization techniques to improve the performance. Loop unrolling can be used to increase the throughput while loop tiling can be used to optimize the memory bandwidth.

2.4 Data-path Optimizations for Accelerating FPGA-Based CNN

GPUs are predominately used to implementing CNNs as they can exhibit high degrees of parallelism. Due to the resource limitation on FPGAs, it is challenging to exploit all parallelism patterns. The current deep CNN models cannot be fully unrolled, i.e. executed parallelly even for a single layer due to the sheer volume of operations involved. Hence all the state-of-the-art FPGA implementations of CNN are done by temporally iterating data through Processing Elements (PEs). There are three methodologies namely Systolic Arrays, SIMD and Dataflow Model of Computation(MOC) that are broadly used when designing PEs for FPGA implementation.

2.4.1 Systolic Arrays

Systolic arrays were adopted early on to accelerate CNN on FPGA. A systolic array is a static collection of PEs that operate under the control of a CPU. The PEs are arranged in a 2D grid which are controlled by a CPU host. These PEs are agnostic to CNN architecture. These PEs can support CNN upto a certain filter kernel size for which it is designed. For example, in [58], the PEs can support convolution upto filter size of 7, while in [59], the PEs can support convolution upto filter size of 10. Systolic arrays do not support data caching, hence it is required to fetch inputs and write outputs of convolution operations to an off-chip memory. Hence performance efficiency of systolic arrays is memory bandwidth limited.

2.4.2 SIMD Accelerators

The systolic arrays demonstrate inefficiency due to their dependency on memory bandwidth of the device. Recent research [60, 61, 62, 63, 64] focused on flexible single instruction multiple data (SIMD) accelerators for speeding up CNNs. The general flow in this architecture is to fetch feature maps and weights from DRAM and cache them in on-chip buffers. This data is passed to the PEs and the results are again stored on on-chip buffers. Upon completion of a layer the results are moved to DRAM for the next layers. Each PE has its own computational units and on-chip buffers for data caching.

With SIMD Accelerators, the challenge while mapping CNNs comes down to finding optimal number of PEs, number of DSP blocks in each PEs, on-chip buffer sizes for data caching and scheduling of data to maximize the throughput [63, 64, 65]. For convolution layers, we need to consider loop optimization strategy to find out the optimal PE configuration. There are two types of loop optimizations considered, loop unrolling and loop tiling. We are trying to optimize the seven nested loops shown in Eqn. 2.1 that perform convolution by loop unrolling or loop tiling. We determine the unrolling and tiling factors to find the optimal number of PEs, DSP blocks per PE, on-chip buffer sizes and number of DRAM access required.

2.4.3 Dataflow Model of Computation (MOC)

CNN architectures are feed-forward by nature, hence they are contradictory to a Von Neumann execution methodology. If CNN implementations have to access each instruction from memory they would be memory bound. The dataflow MOC approach can simultaneously process multiple streams of data through multiple fragments of instructions. The dataflow MOC architectures are referred as Data-flow Process Networks [66]. In such network [67, 68], each node is a fundamental processing element known as an actor and the edge is a communication FIFO Channel. Abstract tokens are exchanged between the actors through FIFOs. The actor is executed only when there is availability of input operands, in other words, they are data driven. A CNN can be mapped into such an approach as the layers in CNN only process data once they have feature maps available.

Chapter 3

Model Compression

As deep neural networks are explored, it is imperative to reduce the size and computation costs in order to enable real-time applications. There are some fundamental challenges in implementing deep neural networks on portable embedded devices due to memory bandwidth, energy consumption and resources available. There is a growing need to have efficient compact convolution neural network frameworks to be deployed on embedded hardware. For example, a deep neural network like ResNET-50 [69] is 50 layers deep and has about 95 MB of weights along with 3.8 billion floating point multiplications to process a single image. It is almost impossible to implement such a network efficiently on embedded devices like an FPGA. If we can discard some redundant connections, the network will still work the same baring a slight drop in efficiency, but it would save about 75% of parameters and 50% of computational time [70].

There are four categories in which convolution neural network compression algorithms can be classified: parameter pruning and sharing, knowledge distillation, low-rank factorization and compact/transferred convolution filters. The redundancy in the network connections is exploited by parameter pruning and sharing to compress the CNN. A special structural convolution filters are designed to save storage space and parameter space in compact/transferred convolution filters approach. A distilled compact CNN model is trained to produce the output of a larger CNN model in knowledge distillation. Finally in low-rank factorization, the matrix/tensor decomposition is used to predict the CNN parameters. We have summarized all the four categories in Table 3.1.

The transferred/compact filters methods are typically used only for networks

TABLE 3.1: Summary of Different Model Compression Techniques.

Compression Technique	Description	Model	Training Supported
Parameter Pruning & Sharing	Reduces redundant parameters	Both Convolution & Fully-Connected layers	Can support training from scratch or transfer learning
Transferred or Compact Convolutional Filters	Design structural filters to save space	Convolutional layer only	Only supports training from scratch
Low-rank Factorization	Matrix decomposition to estimate useful parameters	Both Convolution & Fully-Connected layers	Can support training from scratch or transfer learning
Knowledge Distillation	Train a smaller network from a larger network	Both Convolution & Fully-Connected layers	Only supports training from scratch

with purely convolution layers. If the CNNs have fully connected layers, then parameter pruning and sharing, low-rank factorization and knowledge distillation methods are used. The parameter sharing and pruning approaches have different methods like quantization, binarization to compress the CNN so they are little bit difficult to integrate in the CNN pipeline, while the low-rank factorization method fits well into the CNN pipeline to be easily integrated in the CPU/GPU environment. With respect to the training protocols, the knowledge distillation and transferred/compact filters method require training from scratch, while parameter pruning and sharing as well as low-rank factorization allows for transfer learning or training from scratch. Since these methods are mutually exclusive in terms of their compression approach, they can be clubbed together to further optimize a CNN. Transferred/compact filters along with parameter pruning and sharing, while quantization with low-rank factorization can be used together to achieve better compression of CNNs.

3.1 Parameter Pruning and Sharing

Initial research [71] showed promised that pruning is effective to reduce the complexity of the CNN. Following this research, more efforts were focused on identifying the redundant parameters in a network that can be pruned in order to improve the generalization of the CNN. The parameter pruning and sharing can be further categorized into 3 parts: pruning and sharing, quantization and binarization and finally structural matrix.

3.1.1 Pruning and Sharing

Parameter pruning and sharing is used to reduce the complexity as well as the overfitting in the CNNs. Early adoptions of pruning [72] reduced the number of connections with the help of hessian loss function and it was found this approach provides better pruning results than magnitude based weight decay method [73]. For both these approaches training was done from scratch.

In recent years, there is a growing consensus of pruning the redundant parameters of a pre-trained CNN. Some of the recent works [74, 75, 76] proposed to reduce the redundant parameters using a data-free method or using a lost-cost hash function to group wights in buckets for sharing. Some works like [77] proposed to remove the redundant parameters, followed by quantization and encode the quantized weight using huffman coding to improve the space requirements. Currently, there active area of research is to introduce the sparsity constraint while training the CNNs. These constraints are introduced in optimization with l_0 and l_1 norm regularizers [78, 79, 80].

Some of the drawbacks for pruning and sharing is that it requires additional iterations in order to converge. Furthermore,a manual setup is required to fine-tune the parameters and can be inefficient for some applications.

3.1.2 Quantization and Binarization

In quantization, the idea is to reduce the number of bits required to represent individual weights. Some of the approaches [81, 82] used k means scalar quantization to reduce the size of the weights. It is seen in [83] that 8-bit quantization of weights have little effect on accuracy. A stochastic rounding based 16-bit fixed point representation is used in [84] which drastically reduces the memory usage while having minimal effect on accuracy. In reference [85], it is shown that Hessian weight can be used to identify the importance of the network parameters and they proposed to minimize the Hessian weighted quantization errors for clustering the network parameters.

Some quantization approaches represent each weight in 1-bit, hence binarizing the weight. A few networks that are developed to train with binary weights are BinaryNet [86], XNORNet [87], BinaryConnect [88].The core idea to learn the binary weights or activation during training. It has been observed that CNNs trained using back-propagation even the binary weights are more resilient to weight-distortion.

One of the interesting approach proposed by [89] quantized the weigths by weight sharing and then encoding then using Huffman coding as well as a codebook. They first establish the network and train normally. Then they prune the network and retrain it on the sparse connections. Once the training is done on the pruned network, the weights are grouped and quantized according to the codebook. Further they encode the quantized weights with Huffman coding and store them. This approach has not only seen great reduction in memory usage but state-of-the-art performance amongst all the quantization strategies.

The quantization and binarization do come with some drawbacks. On large networks like ResNet or GoogleNet, binary or quantized weights have seen reduction in classification accuracy. This is due to both quantization and binarization does not account for its losses in while training the network.

3.1.3 Structural Matrix

In fully-connected networks, the bottleneck is the memory consumption. It is important to determine any redundancy in the full-connected layers. The fully-connected networks employ nonlinear transform in the form of $f(x, M) = \sigma(xM)$, where x is the input vector, M is $m \times n$ matrix of parameters and $\sigma(.)$ is an element-wise nonlinear operator. For a large dense matrix M, the cost of storing and computation times is $\mathcal{O}(mn)$. A naive approach to prune parameters would be to use x as a parameterized structural matrix. A matrix that can be effectively represented by fewer number of parameters is called a structural matrix. Usually, the structure should not just reduce the memory usage but also lower the computation time by accelerating the training and inference with the help of fast matrix-multiplication.

A novel Adaptive FastFood transform was proposed by [90] to re-parameterize the matrix multiplications of fully-connected layers. The adaptive FastFood transform matrix $R \in R^{n \times d}$ is defined as:

$$R = SHG\Pi HB \tag{3.1}$$

where S, G, B are random matrices and H is the Walsh-Hadamard matrix. Here $\Pi \in 0, 1^{d \times d}$ is a random permutation matrix. This approach of re-parameterizing a fully connected layer with d inputs and n outputs resulted in reduction in storage and computation costs from $\mathcal{O}(nd)$ to $\mathcal{O}(n)$ and $\mathcal{O}(nlog(d))$, respectively.

However, the issue with this kind of optimization approach is that it may introduce a bias to the network which will hurt its accuracy. There is not a definitive or theoretical approach to find the structural matrix.

3.2 Transferred or Compact Convolution Filters

A large amount of empirical evidence suggest that the translation invariant property of CNNs combined with parameter sharing can provide good predictive performance. Reference [91] introduced equivariant group theory, which is first of its kind in compressing CNNs using transferred convolution filters. The concept of equivalence is described as follows:

$$\mathcal{T}^1 \phi(x) = \phi(\mathcal{T} x) \tag{3.2}$$

where $\phi(.)$ is the network or layer, x is an input and $\mathcal{T}(.)$ is the transform matrix. The above equation suggest that, for any given input x, first transforming it with $\mathcal{T}(.)$ and then passing it through the network or layer $\phi(.)$ would result in same as first passing the input through then network or layer and then transforming it. It is noted that the transforms $\mathcal{T}(.)$ and $\mathcal{T}^1(.)$ may not be same.

This directive is followed by many recent works [92, 93, 94, 95] to build the convolution layers from a set of base filters. Reference [94] observed, the convolution layers of initial stages created redundant filters to extract positive and negative phase information of input. Hence they developed a transform which simply performs a negation function as given in (3.3).

$$\mathcal{T}(W_x) = W_x^- \tag{3.3}$$

Here, W_x is the basis convolution filter and W_x^- is the filter whose activation is opposite to the original basis filter and performed after max-pooling. This approach resulted in 2x compression in size of the convolution layers. They also observed that the negation transform also acts as a strong regularizer which improves the classification accuracy. Similarly, reference [94] observed wide diversity of pattern representations in the convolution layers. Hence, they figured it is incorrect to discard single threshold weak signals. They proposed a multi-bias non-linearity activation function that will generate more patterns in less feature space at reduced computational cost. The transform was proposed is as shown in (3.4) with δ as a multi-bias factors.

$$\mathcal{T}\phi(x) = W_x + \delta \tag{3.4}$$

There are a few issues with this kind of approach where a transform is applied to convolution filters. Firstly, they can be applied to wide/flat networks like VGG16 but not thin/deep architectures like GoogleNet. Secondly, the transfer function can lead to unstable results due to issues with learning.

3.3 Low-Rank Factorization

The majority of computations in CNNs arise from convolution operations. Hence reducing the number of convolutions will reduce the size of the CNN as well as increase the computation speed. A convolution layer is effectively a 4D tensor, hence there is a large amount of redundancy.

The low-rank approximation is done on each layer individually. The parameters of one layer are fixed before moving onto the next layer. The layers are then fine-tuned to compensate for the reconstruction error. There are two types of low-rank compression methods. The first, reference [96] uses Canonical Polydaic (CP) decomposition, which uses the nonlinear least squares to compute the decomposition. Reference [97] used Batch Normalization (BN) decomposition in order to transform the activation of hidden units. Both the CP and BN decomposition schemes can be employed to train CNNs from scratch. However, there is a slight difference between them. In CP decomposition, the best rank for decomposition may not exist whereas in BN, there is always a valid decomposition. For a fully-connected layer, both CP and BN decomposition schemes can be applied as a fully-connect layer which is effectively a 2D matrix.

Some of the drawbacks of low-rank approximation is finding the decomposition which is sometimes computationally expensive. The low-rank decomposition fails to optimize the CNN globally as it performs decomposition layer-by-layer. Low-rank factorization requires extensive training to converge compared to the original model as it is done layer-by-layer.

3.4 Knowledge Distillation

In knowledge distillation (KD), reference [99] trained a compressed model with strong classifiers on a pseudo-labeled dataset which produced the output of the larger network. In KD, the larger teacher model transfers its knowledge to the smaller student models by providing the class distributions via softmax. In [98] the student network is penalized based on the softened version of the teacher output. The student model was trained to predict the output labels of the teacher model. Reference [99] proposed a framework which would build thin and deep models from wide and shallower models. This framework would create a thinner and deeper student models. Here the student models would mimic the feature maps of the teacher model.

Knowledge distillation make the deeper models thinner, hence reducing the computation time. However, KD can be only applied to networks which uses softmax at the classification layer. Furthermore, the assumptions are very strict to achieve comparable performance with other approaches.

3.5 Summary and Challenges

To summarize the usage for various compression techniques, we provide different scenarios in which certain approaches can be made more efficient. To choose which compression approach to use depends on the application at hand. In some classification like medical image classification, transferred convolution filters may work better as there is a rotational component to the medical images. If one is working with limited or small size datasets, then knowledge distillation can be beneficial. The compact student model can learn from the teacher model. If one is building compact models from pre-trained large models then pruning and sharing or low-rank factorization methods can prove to be beneficial. Usually the pruning and sharing approach gives sizable reduction in network size while not hurting the accuracy by much. In order to maximize the performance one or more compression approaches can be combined together. For example, in object detection which has both convolution and fully-connected layers, one can use pruning and sharing as well as low-rank factorization to improve the performance of the model.

However, the above mentioned techniques still pose some challenges. One of most important challenges is to implement the CNNS on small hardware platforms like FPGAs or Tensor Processing Units. The model compression approaches should efficiently map the large CNNs on such platforms. The pruning approach is still a black-box mechanism. Most of the algorithms are build on well designed models. Hence there is little room to change the configuration like hyper-parameters or architecture.

Chapter 4

Hardware Architecture

In this chapter, the implementation of the proposed NDTNet on hardware is presented. A generic CNN accelerator architecture is shown in Fig. 4.1. Due to the shear size of the data involved, the weights and intermediate data are stored in external DRAM memory. The input and weight data from the DRAM memory is loaded onto the on-chip input buffers and is further passed to the processing elements (PEs). The intermediate result from the PEs is stored in on-chip output buffers which is then written back on DRAM memory and will be used as input data in following layers.

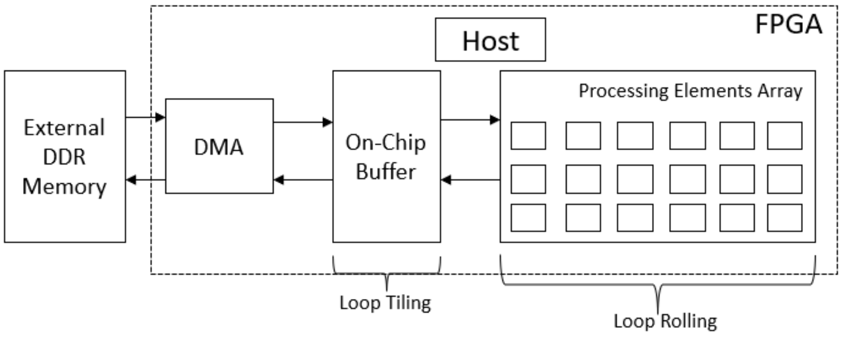


FIGURE 4.1: Generic Architecture of FPGA based CNN Implementation.

4.1 Convolution Operation and Loop Optimizations

A convolution is the accumulation of products of input data like raw pixel values or intermediate feature maps with filter kernel, along different kernel dimensions. A convolution is performed in four nested loops as shown in Fig. 4.2. Various parameters seen in Fig. 4.2 are generated by using the following terminology; i, k and o stand for input, kernel and output, respectively, x and y are width and height, and

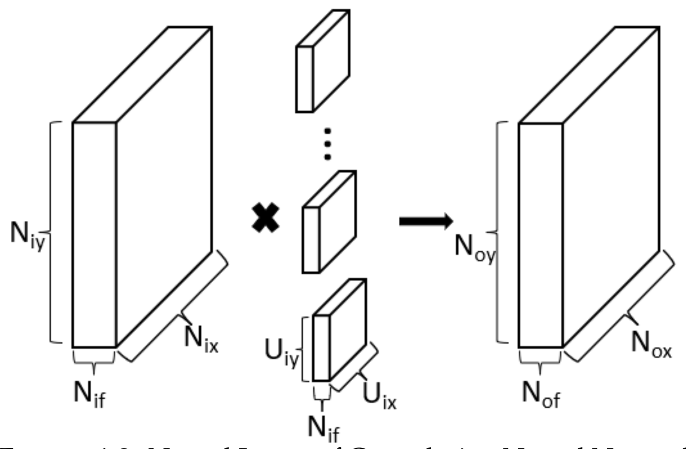


FIGURE 4.2: Nested Loops of Convolution Neural Network.

N stands for number of feature maps as well as filter kernels. Equation (4.1) shows the loop operations seen in Fig. 4.2, where S is stride.

$$Data(o;x,y) = \sum_{ni=1}^{Nif} \sum_{ky=1}^{Nky} \sum_{kx=1}^{Nkx} Data(i;Sx+kx,Sy+ky) \times weight(i,o;kx,ky) + bias(o) \tag{4.1}$$

In this dissertation, we have used loop optimization techniques in the form of loop unrolling for parallel computations and loop tilling for efficient data buffering. Loop unrolling helps us decide how many number of DSPs are required and how many PEs can be implemented in parallel. Loop unrolling also impacts the number of buffer access required to transfer data into and from PEs. Loop tilling exploits the spatial locality of data access by splitting the CNN layer into small tiles. Tiling sizes is determined by the on-chip memory available. This also impacts the number of DRAM memory accesses and consequently affects latency. The loop unrolling and tilling affect key specifications of accelerator like latency, DSP and on-chip memory usage and consequently power consumption.

4.1.1 Convolution Acceleration Strategy

To obtain a precise acceleration strategy, first we need to understand how loop unrolling works and what combinations of loop unrolling can be explored. Unrolling different loops gives different parallelization which can affect PE architecture.

1) In loop 1 unrolling as shown in Fig. 4.3, the inner product Ukx and Uky pixels and filter kernel from different (x,y) location is calculated each cycle and accumulated using an adder tree. This requires $Ukx \times Uky$ multipliers and fan-in adder tree.

2) For loop 2 unrolling, Uif different filter weights and feature maps from the same (x, y) position are used to compute the inner product each cycle. As shown in Fig. 4.4, Nif multipliers are required but fan-in for adder tree is 1.

3) In loop 3 unrolling, every cycle $Uix \times Uiy$ pixels from different (x, y) location are multiplied with same filter kernel. As seen in Fig. 4.5, loop 3 unrolling requires $Uix \times Uiy$ multipliers, but the products are independent, no adder tree is required.

4) In each cycle, for loop 4 unrolling one pixel from feature map is multiplied by Uof filter weights at same (x, y) location but different filter kernels. This requires Uof multipliers and no adder tree is necessary due to independent outputs as shown in Fig. 4.6.

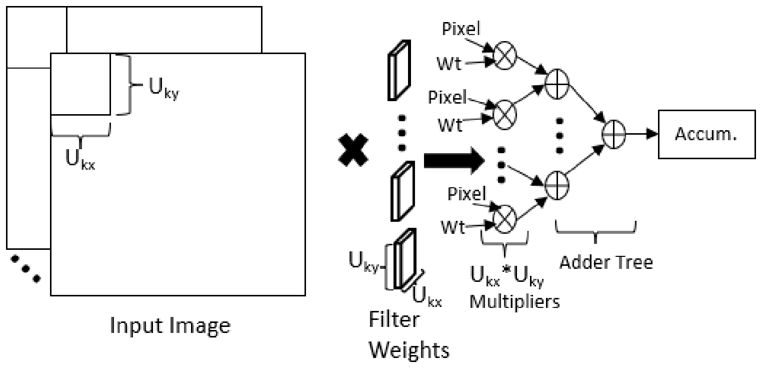


FIGURE 4.3: First Loop unrolling of Convolution Neural Network.

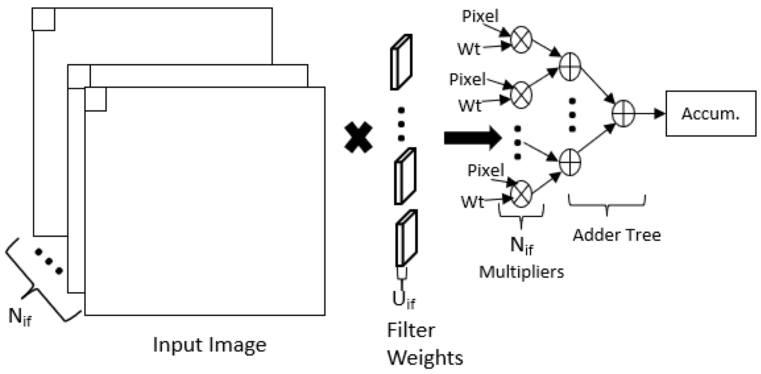


FIGURE 4.4: Second Loop unrolling of Convolution Neural Network.

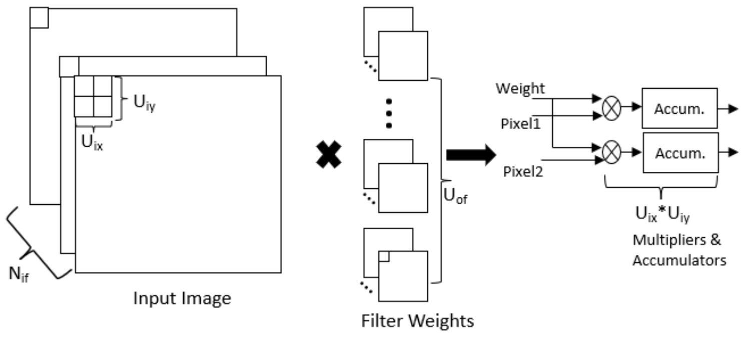


FIGURE 4.5: Third Loop unrolling of Convolution Neural Network.

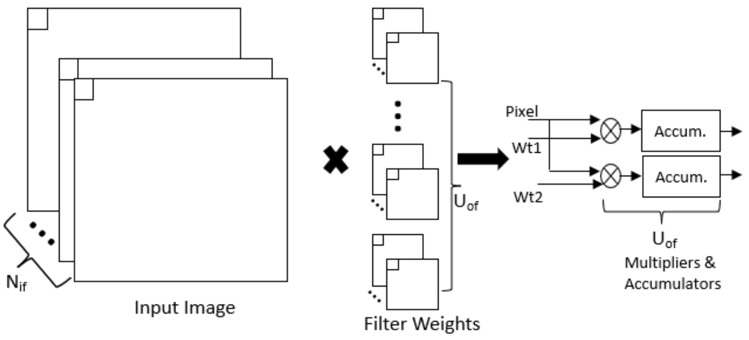


FIGURE 4.6: Fourth Loop unrolling of Convolution Neural Network.

In order reduce the number of DRAM access, we need to reuse both input feature maps and filter kernels. There are two types of data reuse, spatial and temporal. When feature map or filter kernel is read from on-chip buffer and is used by multipliers in same cycle, it is called spatial reuse. Temporal reuse means when same feature map or filter kernel is used in multiple consecutive clock cycles.

Some of the most common loop unrolling optimizations can be categorized into four categories. Loop 1, loop 2 and loop 4 unrolled together (Type-I); Loop 2 and loop 4 unrolled (Type-II); Loop 3 and loop 4 unrolled (Type-III) and finally, loop 1 and loop 3 unrolled (Type-IV). In Type-I, by unrolling loop 1, loop 2 and loop 4, we are achieving parallelism at input, kernel and output. This is most common combination of unrolling seen in many works. If a PE is designed for Type-I unrolling, we can achieve the output feature map without needing intermediate buffering. Type-I uses $Ukx \times Uky \times Uif \times Uof$ multipliers and generate only Uof outputs. As filter kernels are not reused, DRAM access to load filter kernels and store output feature maps is very often, which can increase the latency. Additionally, Type-I is not very efficient to implement depthwise convolution as there is no need for loop 2 unrolling. In Type-II unrolling combination is similar to Type-I without loop 1 unrolled. It has the same drawbacks which Type-I has. Moreover, both Type-I and II unrolling combinations do not unroll loop 3, hence they do not reuse filter kernels which will increase latency due to frequent DRAM access.

Focusing on input feature map as well as filter kernel reuse would help with finding the most efficient loop optimization. In Type-IV, as loop 1 and loop 3 are unrolled, it promotes both input feature map and filter kernel reuse. Although, further data resuse cannot be exploited as loop 4 is not unrolled, so we would need to either load input feature map or filter kernel Nof times. In Type-III, both loop 3 and loop 4 are unrolled hence reusing both input feature map and filter kernel. We would need $Uix \times Uiy \times Uof$ multipliers, depending on DSP slices available on the FPGA, a high degree of parallelism can be achieved. For smaller FPGAs, Type-III cannot exploit filter kernel reuse completely as is less efficient as there are not enough DSPs.Due to this in our work we used a modified Type-III unrolling combination. Instead of using only one filter weight in loop 3 unrolling we propose using a matrix 5×5 filter weights and reduce the number of Uix depending on DSP availability. This 5×5 filter weight matrix can be used to get four partial convolution results when the filter kernel is of size 3×3 or one partial convolution with filter kernel of size 5×5 or for different combination of winograd algorithm.

4.2 Convolution Acceleration Performance Dependency

In this section, we look into factors we need to consider when deciding the unrolling and tiling sizes. The most important factors to look at are DRAM access latency and on-chip buffer size. We also need to consider the computation latency for each convolution.

4.2.1 Computation Latency

The number of multiplication operations required for any given convolution layer is shown in equation (4.2)

$$\# \, Muls = N_{if} \times N_{ix} \times N_{iy} \times N_{of} \times N_{ox} \times N_{oy} \tag{4.2}$$

By unrolling the number of PEs that decide the level of parallelism achieved is given by equation (4.3).

$$\#\ PEs = U_{if} \times U_{ix} \times N_{iy} \times U_{of} \times U_{ox} \times U_{oy} \tag{4.3}$$

Let the the size of a input buffered tile be $T_{if} \times T_{ix} \times T_{iy}$ which produces the output buffered tile of size $T_{of} \times T_{ox} \times T_{oy}$. Hence in theory the clock cycles required to complete the convolution of one buffered tile (# $ClockC_1T$) with the number of PEs possible with any given unrolling strategy is shown in equation (4.4).

$$\#\ ClockC\_1T = \lceil \frac{T_{if}}{U_{if}} \rceil \times \lceil \frac{T_{ix}}{U_{ix}} \rceil \times \lceil \frac{T_{iy}}{U_{iy}} \rceil \times \lceil \frac{T_{of}}{U_{of}} \rceil \times \lceil \frac{T_{ox}}{U_{ox}} \rceil \times \lceil \frac{T_{oy}}{U_{oy}} \rceil \tag{4.4}$$

Similarly, number of tiles (# $Tiles$) in any given convolution layer is shown in equation (4.5).

$$\#\ Tiles = \lceil \frac{N_{if}}{T_{if}} \rceil \times \lceil \frac{N_{ix}}{T_{ix}} \rceil \times \lceil \frac{N_{iy}}{T_{iy}} \rceil \times \lceil \frac{N_{of}}{T_{of}} \rceil \times \lceil \frac{N_{ox}}{T_{ox}} \rceil \times \lceil \frac{N_{oy}}{T_{oy}} \rceil \tag{4.5}$$

Thus from equations (4.4) and (4.5) we can determine the number of clock cycles (# $ClockC\_CL$) required for a convolution layer as shown in equation (4.6) below.

$$\#\ ClockC\_CL = \#\ ClockC\_1T \times \#\ Tiles \tag{4.6}$$

The above equation gives us the latency of one convolution layer. Let us now find computation delay of one convolution tile and subsequently of one convolution layer in terms of milliseconds. We assume only that there is one PE i.e. no parallelism. Hence $U_{if} = U_{ix} = U_{iy} = 1$. And from equation (4.4), we get

$$\#\ ClockC\_1T = T_{if} \times T_{ix} \times T_{iy} \times \lceil \frac{T_{of}}{U_{of}} \rceil \times \lceil \frac{T_{ox}}{U_{ox}} \rceil \times \lceil \frac{T_{oy}}{U_{oy}} \rceil \tag{4.7}$$

If we assume we can buffer in a complete input feature map into one tile, then $T_{if} = N_{if}$ $T_{ix} = N_{ix}$ $T_{iy} = N_{iy}$ and $T_{ox} = N_{ox}$. Therefore, substituting this into equation (4.7) we get,

$$\#\ ClockC\_1T = N_{if} \times N_{ix} \times N_{iy} \times \lceil \frac{T_{of}}{U_{of}} \rceil \times \lceil \frac{N_{ox}}{U_{ox}} \rceil \times \lceil \frac{T_{oy}}{U_{oy}} \rceil \tag{4.8}$$

Thus, the computation delay (*ComputeD*) in (ms) can be found with the help of equation (4.8).

$$ComputeD\ (ms) = \frac{\#\ ClockC\_1T}{Operating\_Freq(MHZ) \times 10^3} \tag{4.9}$$

where $Operating\_Freq(MHZ)$ is the operating frequency of the accelerator in MHz. Using equation (4.5) and our assumption regarding tiles above, the number of tiles in one convolution layer is $\#\ Tiles = \lceil \frac{N_{of}}{T_{of}} \rceil \times \lceil \frac{N_{oy}}{T_{oy}} \rceil$. Therefore, the computation delay for a whole convolution layer is $ComputeD\_CL\ (ms) = \#\ Tiles \times ComputeD\ (ms)$.

4.2.2 On-Chip Buffer Size and Access Latency

The on-chip buffer size completely depends on the tiling size. The size of input buffer ($InBuf\_Size$) required to store one tile is given as follows:

$$InBuf\_Size\ (bits) = T_{ix} \times T_{iy} \times T_{if} \times Bits\_per\_Pixel \tag{4.10}$$

Similarly, the size to store the weights in its corresponding weight buffer ($WtBuf\_Size$) is given as:

$$WtBuf\_Size\ (bits) = T_{ix} \times T_{iy} \times T_{if} \times T_{of} \times Bits\_per\_Weight \tag{4.11}$$

The size required to store one tile of output feature map in output buffer ($OutBuf\_Size$) is as follows:

$$OutBuf\_Size\ (bits) = T_{ox} \times T_{oy} \times T_{of} \times Bits\_per\_Pixel \tag{4.12}$$

Equations (4.10 - 4.12) are the theoretical maximum possible buffer sizes for input, weight and output buffers of all convolution layers. In actual implementation the buffer sizes for all these buffers would be greater than the these theoretical values due to inefficient storage pattern. For example, if the input buffer size requirement is X, the actual buffer size in implementation would be $2^{ceil(log_2 X)}$. Hence the extra space is wasted.

The number of on-chip buffer access (# Buf_access) in terms of bits is computed by multiplying number of clock cycles accesses (# CC_access) with the bit width of a given buffer ($Buffer\_Bw$).

$$\#\, Buf\_access(bits) = \#\, CC\_access \times Buffer\_Bw \tag{4.13}$$

A general assumption is that during every clock cycle, the data is continuously read from input and weight buffers and written into output buffer. A good estimation for on-chip buffer access during computation will be if the number of clock cycles accesses are equal to the number of computation cycles while the buffer width is the width of either of input or weight or output buffer. Depending on the unrolling strategy used, the data stored in the input buffer or weight buffer can be read multiple times. But the data written to the output buffer is only once for one convolution operation. Hence the size of read and write operations of the output buffer remains the same. The size of data read from input buffers can be much greater than data written in input buffers. This is also true for weight buffers as well.

Size and Storage Estimation for On-Chip Buffers

For U_{ox} parallel computations in one feature map row, the width of input buffer would depend on U_{ox}. The total number of input buffers depend on U_{oy} that will perform U_{oy}parallel output computations. In Fig. 4.7, $c(x)$ is the one input feature map in the xth column of a given row. Here $x \in \{1, 2, \ldots, T_{ix} - 2 \times pad\}$. Here the T_{ix} includes the left and right padding as it is taken care by the line buffer. The total number of words in one row (# $1row\_words$)can be given by below equation.

$$\#\, 1row\_words = \lceil \frac{(T_{ix} - 2 \times pad)}{U_{ix}} \rceil \tag{4.14}$$

As seen in Fig. 4.7, $r(i, y)$ corresponds to i$_{th}$ element of the feature map in the y$_{th}$ row. Here, $i \in \{1, 2, \ldots, T_{if}\}$ and $y \in \{1, 2, \ldots, T_{iy}\}$. The T_{iy} includes any top or bottom padding. The adjacent rows are stored continuously in a same input buffer for stride 2. In one input buffer, the number of rows of input feature maps (# $1Fmap\_rows$) can be given by

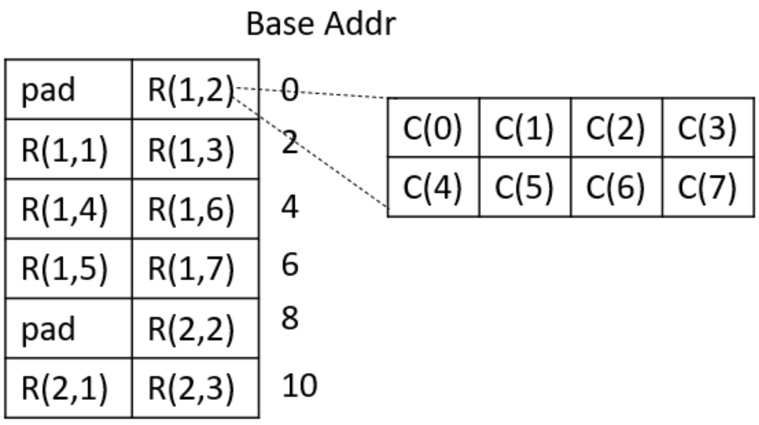


FIGURE 4.7: Address logic for Input Buffer.

$$\#\, 1Fmap\_rows = \lceil \lceil \frac{T_{iy}}{stride} \rceil \times \frac{1}{U_{oy}} \rceil \times stride \tag{4.15}$$

We aligned the storage location of subsequent feature maps with the first feature map for simpler address generation logic. The depth one input buffer ($Depth\_inBuf$) which stores T_{if} input feature maps for a given convolution layer is shown in equation (4.16) by using equations (4.14 and 4.15).

$$Depth\_inBuf = \#\, 1row\_words \times \#\, 1Fmap\_rows \times T_{if} \tag{4.16}$$

The bit width of a single input buffer is $U_{ox} \times Bits\_per\_Pixel$. For efficient buffering we are using a ping-pong buffer structure. Hence each clock cycle $U_{ox} \times U_{oy}$ pixels can be passed on to the MAC units in the pipeline. Therefore, the input buffer size for one convolution layer in terms of bits ($In\_Buf(bits)$) is expressed by

$$In\_Buf(bits) = Depth\_inBuf \times U_{ox} \times U_{oy} \times 2 \times Bits\_per\_Pixel \tag{4.17}$$

The actual input buffer size would be maximum value of equation (4.16) computed for all convolution layers. This value is larger than the estimated value in equation (4.10) due to mismatch in tile and buffer dimension.

We can find the storage requirements for weight buffer as above. In Fig, 4.8, k(x, y) is one weight element in $N_{ix} \times N_{iy}$ kernel. Here, $x \in \{1, 2, \ldots, T_{ix}\}$ and

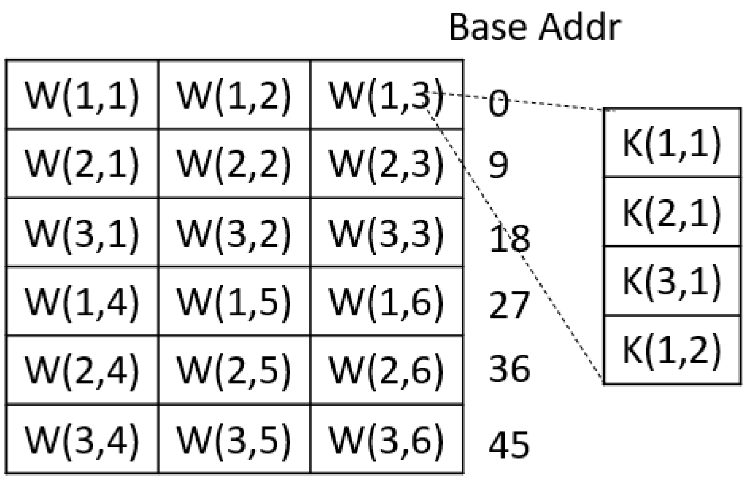


FIGURE 4.8: Address logic for Weight Buffer.

$y \in \{1, 2, \ldots, T_{iy}\}$. We always have one weight kernel always buffered in the design. The weight kernels corresponding to same input channels i.e. $T_{ix} \times T_{iy}$ are stored in continuous addresses, while the weights corresponding to different input channels T_{if} are stored in different addresses as we serially compute the channels. To parallelly compute U_{oy}, the weights are stored in same address of U_{oy} buffers. The depth of weight buffer ($Depth\_wtBuf$) is given by

$$Depth\_wtBuf = T_{ix} \times T_{iy} \times T_{if} \times \lceil \frac{T_{of}}{U_{oy}} \rceil \tag{4.18}$$

Like the input buffer, we use a ping-pong buffer structure for efficient data access. Hence the weight buffer size for one convolution layer in terms of bits ($Wt\_Buf(bits)$) is expressed as

$$Wt\_Buf(bits) = Depth\_wtBuf \times Bits\_per\_Weight \times U_{oy} \times 2 \tag{4.19}$$

The actual weight buffer size in the design would be maximum of equation (4.19) computed across all convolution layers.

We expect $U_{ox} \times U_{oy} \times U_{of}$ outputs from MAC units every $N_{ix} \times N_{iy} \times N_{if}$ clock cycles. The bit width of one output buffer is $U_{ox} \times Bits\_per\_Pixel$. In Fig. 4.9, c(x) is the xth column element of the output feature map row, where $x \in \{1, 2, \ldots, T_{ox}\}$. The r(o, y) corresponds to the yth row in the oth output. Here $y \in \{1, 2, \ldots, T_{oy}\}$ and $o \in \{1, 2, \ldots, T_{of}\}$. In one row, we have T_{ox} elements. These elements are continuously stored as the output buffer uses row-major storage format. The c(x) elements

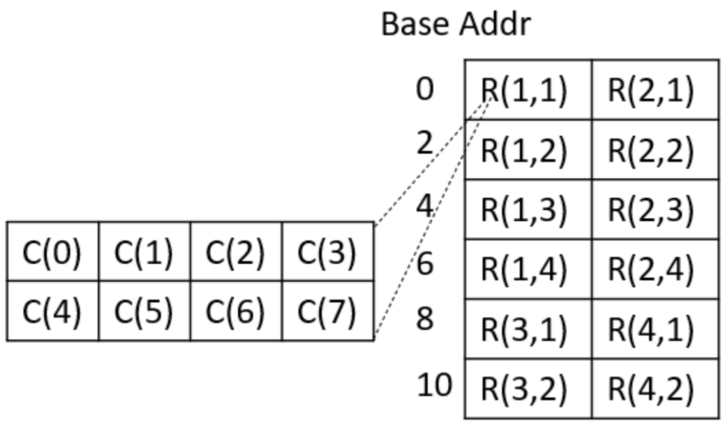


FIGURE 4.9: Address logic for Output Buffer.

would then require $\lceil \frac{T_{ox}}{U_{ox}} \rceil$ continuous addresses. The value T_{ox} is equal to N_{ox} in order to enable the row-major storage format. Each buffer stores a maximum of U_{of} elements. In the design each output buffer can store a maximum$\lceil \frac{T_{of}}{U_{of}} \rceil$ feature maps. Each feature map has T_{oy} number of rows. Hence the depth of the output buffer ($Depth\_outBuf$) is given by

$$Depth\_outBuf = \lceil \frac{T_{ox}}{U_{ox}} \rceil \times T_{oy} \times \lceil \frac{T_{of}}{U_{of}} \rceil \tag{4.20}$$

We use ping-pong buffer structure, hence the output buffer size for one convolution layer in terms of bits ($Out\_Buf(bits)$) is expressed as

$$Out\_Buf(bits) = Depth\_outBuf \times Bits\_per\_pixel \times U_{ox} \times 2 \times U_{of} \tag{4.21}$$

Clock Cycle Requirements for On-Chip Buffer Access

The biggest bottle-neck in a CNN accelerator design is memory bandwidth. Furthermore, the energy consumption is directly proportional to the number of times memory is accessed. Reducing buffer access through loop unrolling strategies, i.e. sharing as much data as possible between multiple PEs, is one of the most efficient way to reduce energy consumption as well as address the increase in latency caused frequent memory access.

The number of clock cycles required for a MAC computation of one tile is precisely equal to the on-chip buffer access shown in equation (4.13). Hence the number of clock cycles required for all convolution layers ($\# \, CC\_Conv$) is. Here $\#CL$ is the number of total convolution layers and $\#T$ is tiles in each convolution layer.

$$\# \, CC\_Conv = \sum_{L=1}^{\#CL} \# \, ClockC\_1T[L] \times \#T[L] \tag{4.22}$$

We will now calculate the read clock cycles for input and weight buffer. For input buffer, $U_{ox} \times U_{oy}$ pixels are reused by U_{of} MAC units. While, for weight buffer U_{of} weights are reused by $U_{ox} \times U_{oy}$ MAC units. Hence, the read access time for both input and weight buffers (Rd_InBuf and Rd_WtBuf respectively) in terms of bits is expressed as

$$Rd\_InBuf(bits) = \# \, CC\_Conv \times (U_{ox} \times U_{oy} \times Bits\_per\_pixel) \tag{4.23}$$

$$Rd\_WtBuf(bits) = \# \, CC\_Conv \times (U_{of} \times Bits\_per\_weight) \tag{4.24}$$

Now we discuss the write part of the input and weight buffer. As we know, before the computation begins the input feature map and weight data is loaded into these buffers from DRAM. But due to unrolling strategies used, we don't need to load new weight or input data every clock cycle as it is reused multiple times. The number of tiles in which new weight and input data is written every convolution layer is

$$\# \, Wt\_T = \lceil \frac{N_{of}}{T_{of}} \rceil \tag{4.25}$$

$$\# \, In\_T = \lceil \frac{N_{oy}}{T_{oy}} \rceil \tag{4.26}$$

The number of bits of one tile of input and weight data written in one write cycle to the corresponding buffers can be expressed as

$$\# \, Wt\_1T(bits) = Depth\_wtBuf \times Bits\_per\_Weight \times U_{oy} \tag{4.27}$$

$$\# \, In\_1T(bits) = Depth\_inBuf \times U_{ox} \times U_{oy} \times Bits\_per\_Pixel \tag{4.28}$$

Using the above equations (4.27-4.28) the total amount of data in terms of bits that is written into the input and weight buffers is found as

$$\# \, Wt\_Conv(bits) = \sum_{L=1}^{\#CL} \# \, Wt\_1T[L] \times \#T[L] \tag{4.29}$$

$$\# \, In\_Conv(bits) = \sum_{L=1}^{\#CL} \# \, In\_1T[L] \times \#T[L] \tag{4.30}$$

For each output buffer, every clock cycle one word is written from its corresponding tile. Hence the number of clock cycles required to write data into output buffer is simply the depth of the output buffer times number of tiles in that layer. So for the whole convolution the clock cycles required to write the data in the output buffer is

$$\# \, Out\_CC\_Conv = \sum_{L=1}^{\#CL} Depth\_outBuf[L] \times \#T[L] \tag{4.31}$$

Therefore, the number of bits written in output buffer is as follows. Since every output is written and read from the output buffer only once, the size of data written into the output buffer (*Out_Conv_Wr*) and read by DMA (*Out_Conv_Rd*) is the same.

$$Out\_Conv\_Wr = Out\_Conv\_Rd = \# \, Out\_CC\_Conv \times \# \, Out\_Buf \times U_{ox} \times Bits\_per\_Pixel \tag{4.32}$$

4.2.3 DRAM Data Access

Theoretically the size of data read from or written into DRAM for one tile should be same as the data that is buffered into the input/weight/output buffer. The number of bytes of input, weight and output data that is read from or written into DRAM for one tile is given by

$$In\_Rd(byte) = InBuf\_Size/8 \tag{4.33}$$

$$Wt\_Rd(byte) = WtBuf\_Size/8 \tag{4.34}$$

$$Out\_Wr(byte) = OutBuf\_Size/8 \tag{4.35}$$

The latency of the DRAM transaction for one tile is determined by the data accessed by DRAM and memory BW (Mem_BW in GB/s) as

$$DRAM\_1T(ms) = \frac{In\_Rd \text{ } or \text{ } Wt\_Rd \text{ } or \text{ } Out\_Wr}{Mem\_BW \times 10^6} \tag{4.36}$$

In order to achieve maximum efficiency the bitwidth for both the DMA (DMA_bit) and DRAM ($DRAm\_bit$) is set to 512 bits. U_{ox} number of pixels are generated in parallel PEs. The group of P_{ox} pixels that will be make up one DMA address is given by

$$\#U_{ox}\_G = \lfloor DMA\_bit/(U_{ox} \times Bits\_per\_Pixel) \rfloor \tag{4.37}$$

The effective DMA bits for the input and output data (DMA_bits_eff) and weight ($DMA\_wt\_bits\_eff$)out of the set DMA bits (512) is given by

$$DMA\_bits\_eff = \lceil \frac{\#U_{ox}\_G \times U_{ox} \times Bits\_per\_Pixel}{DMA\_bit} \rceil \tag{4.38}$$

$$DMA\_wt\_bits\_eff = \lceil \frac{\lfloor \frac{DMA\_bit}{Bits\_per\_weight} \rfloor \times Bits\_per\_weight}{DMA\_bit} \rceil \tag{4.39}$$

The intermediate results from the convolution layers are stored row-by-row, then map-by-map and finally layer-by-layer in the DRAM. Each convolution tile requires $T_{ix} \times T_{iy} \times T_{if}$ input data. The size in terms of bytes that is read from the DRAM for one convolution tile is

$$In\_Rd(byte) = \frac{T_{ix} \times T_{iy} \times T_{if} \times Bits\_per\_Pixel}{DMA\_bits\_eff \times 8} \tag{4.40}$$

Similarly, the data written into DRAM from the output buffer of one convolution tile is given as

$$Out\_Wr(byte) = \frac{T_{ox} \times T_{oy} \times T_{of} \times Bits\_per\_Pixel}{DMA\_bits\_eff \times 8} \tag{4.41}$$

The bytes of weights that are read from DRAM for one convolution tile is

$$Wt\_Rd(byte) = \frac{T_{ix} \times T_{iy} \times T_{if} \times T_{of} \times Bits\_per\_weight}{DMA\_wt\_bits\_eff \times 8} \tag{4.42}$$

4.2.4 Implementation

In this section, the implementation of proposed NDTNet on hardware is presented. We have used Winograd's algorithm to boost convolution acceleration. Most acceleration algorithms are focused on solely improving stride-1 convolutions. In this work, we proposed a single datapath solution for both stride-1 and 2 convolution acceleration. Furthermore, the proposed hardware implementation in addition to standard convolution (SC), depthwise convolution (DC), pointwise convolution (PC), and depthwise separable convolution (DSC) as needed in modern CNNs.

4.3 Architecture Overview

The overall hardware architecture is shown in Fig. 4.10. The proposed hardware architecture can be divided into three parts: 1) Neural Engine array; 2) Memory Organization; 3) Controller. The neural engine (NE) array is responsible for implementing all the different layers in the CNN as describe in Section II. The weights and input data are stored in external DDR memory. On-chip feature map buffers are used to reduce the latency caused by off-chip memory operations.

4.3.1 NE Array

Each NE array consists of a Line Buffer, a Winograd based Convolution Accelerator, an Adder Tree used for standard convolution and adding biases, a pooling and ReLU stage. We have used an input channel level of parallelism in the NE array. Each accelerator in the NE array performs the convolution in parallel on different

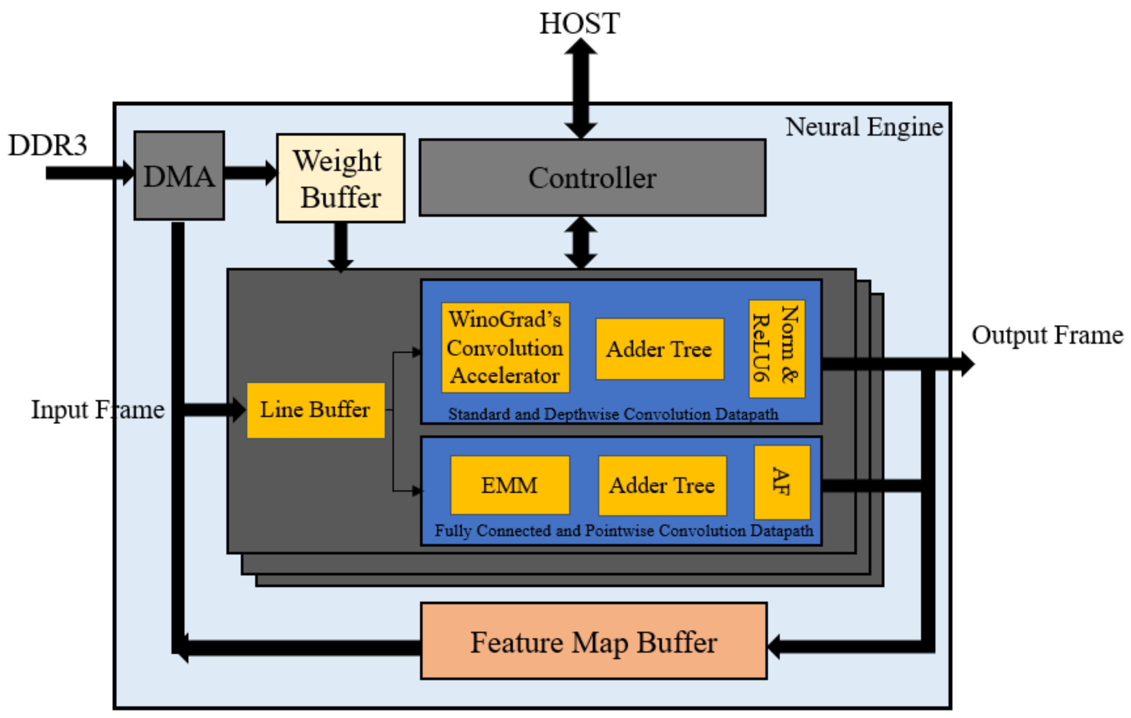


FIGURE 4.10: Block diagram of the proposed Hardware Accelerator.

input channels. There are two datapaths in NE array, first for standard and depthwise convolution which uses Winograd's algorithm to accelearte convolutions. Second datapath consists of simple multiplier array for pointwise and fully connected convolutions. The number of parallel NE arrays depends on the DSP amd on-chip memory available on FPGA. The line buffer serves two purposes, first to automatically pad the input data when required and second to provide the window for matrix multiplication. The implementation of the line buffer is shown in Fig. 4.11. We have used a Winograd's algorithm based convolution accelerator that can perform 1 and 2 stride convolutions. Complete details of the approach is discussed in the following subsection.

4.3.2 Memory Organization

In this accelerator design, we have adopted a hierarchical memory organization. An efficient memory organization requires striking a fine balance between on-chip memory resources and the latency of external DDR memory. Due to the large size of the external DDR memory, it is used to store the input as well as the output feature maps and the parameter data. The feature maps are always stored in a row-major

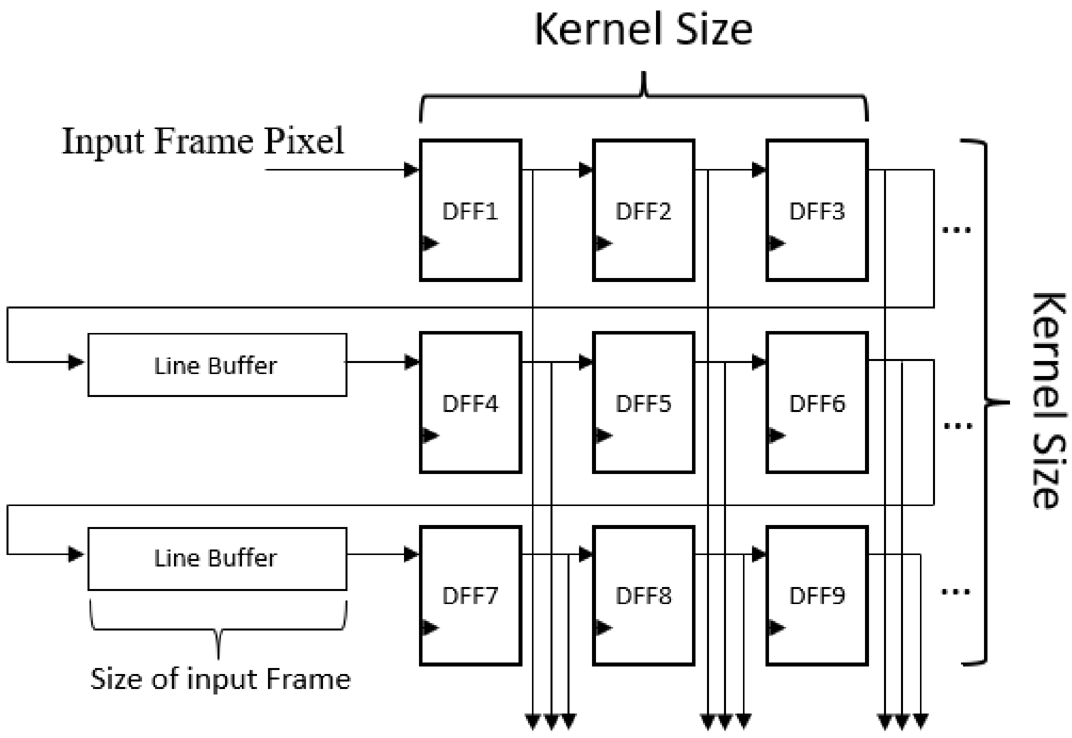


FIGURE 4.11: Line Buffer Architecture.

fashion. The on-chip memory is used as input and intermediate feature map buffer. The amount of on-chip memory required will depends on the number of NE arrays operating in parallel. The parameters, specifically the convolution kernels and biases are initially stored in external DDR memory. Depending on the layer, these parameters are loaded onto the on-chip memory. We have used a ping-pong buffer to reduce the effect of latency caused by the DDR memory. In a ping-pong buffer, when one buffer is loading the data from external memory, the other buffer is sending the data to carry out the convolution.

4.3.3 Controller

The entire CCN computation can be divided into three parts: 1) load data, 2) calculate and 3) save data. These parts correspond to the I/O memory operation with external DDR memory as well as the convolution using the accelerator. A 32-bit instruction set is created to implement these three parts. The instruction set is made up of different fields which provides information on convolution layers, number of channels in the current layer, size of the feature map, stride, address of on-chip memory and instructions (load data, save data and calculate). This instruction set is loaded into a register from which the controller reads the instruction and decodes it. The controller executes its state machine depending on the type of instruction. It will monitor the progress of each instruction and will issue control signals as required. The controller communicates with the host through interrupt. Once interrupted, the host will read the status registers and assign the next instruction to the controller.

4.3.4 Direct Memory Access

In this sub-section we will discuss the Direct Memory Access (DMA) used to load or store data to the external DDR memory. DMAs are used to offload data transaction tasks from CPU to speed up the memory transfer as well as keep the CPU less busy. We use AXI DMA IP available for Xilinx SOC FPGAs. This IP provides a high bandwidth for data transfer between DDR memory and AXI-4 Stream peripherals. In our case the convolution accelerator block as a whole will act as a peripheral to the CPU. There are two modes in DMAs, direct register mode and scatter-gather (SG) mode. DMAs configured in direct register mode use less resources provide lower performance. The DMAs configured in scatter-gather provide higher performance but require more resources than direct register mode.

When the DMA is configured in direct register mode, the CPU will load the length of data transfer, address of memory-mapped streaming peripherals, address of DDR memory, etc. in the specific registers of DMA to perform either memory-mapped to DDR or DDR to memory-mapped data transactions through DMA. Even though the CPU is not actively involved in the memory transactions, there is still some involvement which could potentially affect other tasks handled by the CPU or there will be added latency in memory transactions due to CPU being busy with other tasks.

The SG mode makes use of buffer descriptors (BD) that can be stored in any memory-mapped block like BRAMs. The job of BDs is to provide the fundamental parts of data transfer i.e. address where the data is stored and where the data is to be loaded and size of data. In SG mode, the processor does not directly provide the address of DRAM or memory-mapped peripheral for data transfer. The BDs will provide this address. The BDS will contain the important details like the address and size of data transfer relevant to the data that the DMA will receive of transfer to the DRAM. There are two types of BDs viz., transmission BDs and receiving BDs. The transmission BDs are responsible for data transmission from DRAM to DMA through the MM2S (memory-mapped to stream) channel. While the receiving BDs are responsible for data transfer from DMA to DRAM through the S2MM (stream to memory-mapped) channel. Depending on the type of data transfer or number of data transfers, we can have CPU generate one or more BDs and store it in BRAM. The DMA will sequentially process the BD from the BRAM till there are no BDs left in BRAM unless the DMA is not in cyclic mode. In cyclic mode the DMA will go back to the first BD and keep on cycling through the BDs in the BRAM. Once the DMA processes the final BD which is also called the 'Tail BD', the data transmission is stopped and CPU is notified that the data transmission is done. The CPU can then update the BDs as and when required for the future data transmissions.

4.4 Winograd's Algorithm based Convolution Accelerator

The Winograd's algorithm exploits the overlapping computations in neighboring windows **ref34**. The algorithm replaces multiplications with additions, thereby reducing the number of multiplications required per convolution operation. Since multiplication requires more hardware, the trade-off proposed by the algorithm is quite desirable.

Winograd formulated an efficient method that uses transformation matrices A, B and G on the input data d, kernel k while generating output y as shown in Eqn. 4.43, where (.) is the dot product.

$$y = A^T[(Gk) \odot (B^T d)] \tag{4.43}$$

For a convolution of 1-D input of size 4 and a kernel of size 3, the Winograd's algorithm require 4 multiplications to generate 2 outputs while a conventional method requires 6 multiplications. This algorithm was originally proposed for 1-D convolutions only *F(m,r)*, where m is size of output and r is size of kernel.

Lavin and Gray **ref35**, introduced matrices that are compatible for 2-D convolutions. For 2-D convolutions, the Winograd's algorithm takes the form of *F*(*m* x *m*, *r* x r), where the kernel size is r x r, the output size is m x m while, the input is n x n which is given as $n = m + r - 1$. Equation 4.43 can be modified for 2-D convolutions as follows

$$y = A^T[(GkG^T) \odot (B^T dB)]A \tag{4.44}$$

where, $\odot$ is element-wise matrix multiplication.

In recent years, some researchers have adopted Winograd's algorithm to reduce the computation cost of CNN. As mentioned above, Lavin and Gray modified the Winograd's algorithm for stride-1 2-D CNN and implemented it on a GPU. It reduced the use of multiplications by 2.25x and improved the performance over the CuDNN library. An OpenCL based FPGA implementation of CNNs using Winograd's algorithm is proposed by **ref36** for the 1-D convolution. A 2-D CNN using Winograd's algorithm is implemented on a FPGA platform by Huang *et al* **ref37**. Most of the implementations of Winograd's algorithm are for stride-1 convolutions **ref40**; **ref41**; **ref42**; **ref43**. Yepez and ko **ref44**, proposed modifications on Winograd's algorithm for stride-2 1-D, 2-D and 3-D convolutions.

In a CNN that uses both stride 1 and 2 convolutions, using Winograd's algorithm will require two separate data-paths. A F(2 x 2, 3 x 3) convolution requires 16 and 25 multiplications for stride 1 and 2, respectively **ref44**. As the number of stride-2 convolutions are fewer than stride-1 in any given CNN, the additional resources allocated for stride-2 makes the design less efficient.

In this dissertation, we propose a single datapath that can be used for stride-1 and 2 convolutions. For a stride-2 convolution with a 3-by-3 kernel, the input size should be 5-by-5. In order to have the same datapath, the number of multiplications performed for both stride-1 and 2 convolutions should be the same. We propose to use a F(3 x 3, 3 x 3) Winograd based convolution which will require a 5-by-5 input data and will provide a 3-by-3 output. This approach will require 25 multiplications compared to 81 required by a conventional convolution to get similar number of outputs, which is 3.25x computationally more efficient. As seen in Fig. 4.12, we can use the same 5-by-5 input data with 3-by-3 kernel, but for stride-2 convolutions, it will give 2-by-2 output instead of 3-by-3 output.

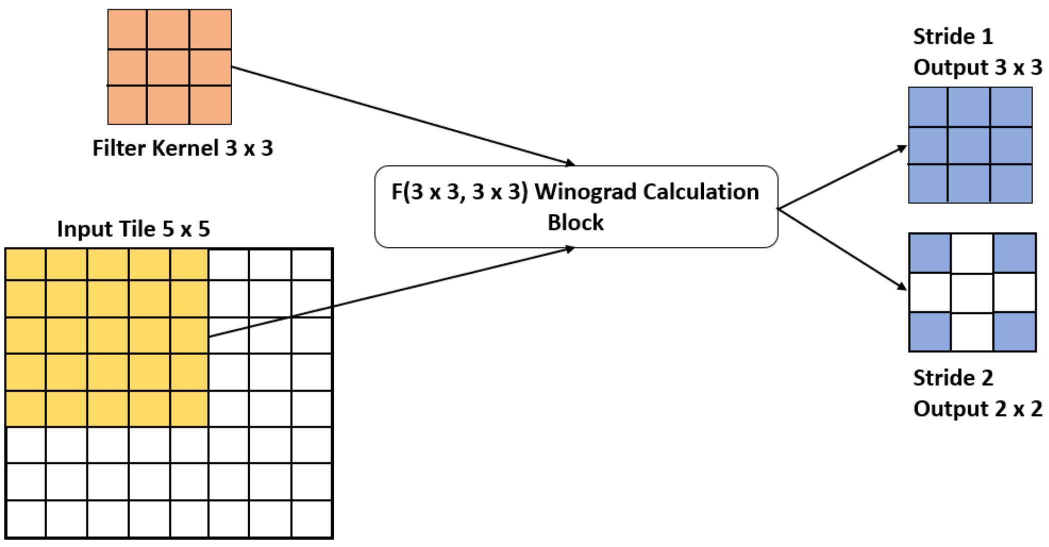


FIGURE 4.12: Proposed Winograd's algorithm for both stride-1 and stride-2 convolutions.

Since there are no available $F(3 \text{ x } 3, 3 \text{ x } 3)$ Winograd matrices, we derive them using the Chinese Remainder Theorem as it was done originally in [17].

Using the Chinese Remainder Theorem, we can solve the above equations represented in matrix form as follows:

The three element filter g and data d can be represented as follows:

$$g(x) = g_2x^2 + g_1x + g_0 \tag{4.45}$$

$$d(x) = d_2x^2 + d_1x + d_0 \tag{4.46}$$

The linear convolution $d * g$ is

$$y(x) = d(x)g(x) \tag{4.47}$$

For the polynomial *m(x)* of degree 5,

$$y(x) = d(x)g(x) \quad mod \quad m(x) \tag{4.48}$$

We select $m(x)$ as

$$\begin{aligned} m(x) &= m^{(0)}(x)m^{(1)}(x)m^{(2)}(x)m^{(3)}(x)m^{(4)}(x) \\ &= x(x-1)(x+1)(x-2)(x-\infty) \end{aligned} \tag{4.49}$$

The residues of $m(x)$ w.r.t. $g(x)$ is

$$\begin{aligned} g^{(0)} &= g(x) \;\; mod\; m^{(0)}(x) = g_0 \\ g^{(1)} &= g(x) \;\; mod\; m^{(1)}(x) = g_0 + g_1 + g_2 \\ g^{(2)} &= g(x) \;\; mod\; m^{(2)}(x) = g_0 - g_1 + g_2 \\ g^{(3)} &= g(x) \;\; mod\; m^{(3)}(x) = g_0 + 2g_1 + 4g_2 \\ g^{(4)} &= g(x) \;\; mod\; m^{(4)}(x) = g_2 \end{aligned} \tag{4.50}$$

and for $d(x)$ is,

$$\begin{aligned} d^{(0)} &= g(x) \;\; mod\; m^{(0)}(x) = d_0 \\ d^{(1)} &= g(x) \;\; mod\; m^{(1)}(x) = d_0 + d_1 + d_2 \\ d^{(2)} &= g(x) \;\; mod\; m^{(2)}(x) = d_0 - d_1 + d_2 \\ d^{(3)} &= g(x) \;\; mod\; m^{(3)}(x) = d_0 + 2d_1 + 4d_2 \\ d^{(4)} &= g(x) \;\; mod\; m^{(4)}(x) = d_2 \end{aligned} \tag{4.51}$$

Define $M^{(i)}(x) = m(x)/m^{(i)}(x)$, which gives us:

$$\begin{aligned} M^{(0)}(x) &= (x^2-1)(x-2) = x^3 - 2x^2 - x + 2 \\ M^{(1)}(x) &= (x^2+x)(x-2) = x^3 - x^2 - 2x \\ M^{(2)}(x) &= (x^2-x)(x-2) = x^3 - 3x^2 + 2x \\ M^{(3)}(x) &= (x^2-1)(x) = x^3 - x \\ m(x) &= (x^2-2x)(x^2-1) = x^4 - 2x^3 - x^2 + 2x \end{aligned} \tag{4.52}$$

To solve with the help of Chinese Remainder Theorem, we need to obtain $n^{(i)}(x)$ and $N^i(x)$ from the equation below,

$$n^{(i)}(x)m^{(i)}(x) + N^{(i)}(x)M^{(i)}(x) = 1 \tag{4.53}$$

TABLE 4.1: Values for $n^{(i)}(x)$ and $N^{i}(x)$

i	$m^{(i)}(x)$	$M^{(i)}(x)$	$n^{(i)}(x)$	$N^{(i)}(x)$
0	x	$x^3 - 2x^2 - x + 2$	$1/2(-x^2 + 2x + 1)$	$1/2$
1	$x - 1$	$x^3 - x^2 - 2x$	$1/2(x^2 - 2)$	$-1/2$
2	$x + 1$	$x^3 - 3x^2 + 2x$	$1/6(x^2 - 4x + 6)$	$-1/6$
3	$x - 2$	$x^3 - x$	$1/6(-x^2 - 2x - 3)$	$1/6$

$n^{(i)}(x)$ and $N^{i}(x)$ values are tabulated above in Table 4.1 The residues of $d^{(i)}(x)$ can be represented in matrix form as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

The G matrix can be constructed by setting the rows G_i equal to the product of coefficients of $g^{(i-1)}$ and and $N^{(i-1)}$

$$G = \begin{bmatrix} 1/2 & 0 & 0 \\ -1/2 & -1/2 & -1/2 \\ -1/6 & 1/6 & -1/6 \\ 1/6 & 1/3 & 2/3 \\ 0 & 0 & 1 \end{bmatrix}$$

Finally, the B matrix is constructed such that the values in B_i are equal to coefficients of $M^{(i-1)}$

$$B = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ -1 & -2 & 2 & -1 & 2 \\ -2 & -1 & -3 & 0 & -1 \\ 1 & 1 & 1 & 1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

With the help of the above transform matrices, we can use Eqn. (2) to compute 2-D convolutions with stride 1 and 2 using 25 multiplications.

The NE array are designed to accommodate the Winograd's algorithm which requires transformation of input, output and filter kernel. In order to optimize resources, the filter kernels for standard and depthwise convolution stored in DDR memory are already transformed in Winograd domain. The input data transformation and inverse transformation are done in the NE array. Fig. 4.13(a) shows the architecture of both input as well as inverse transformation. In order to accomplish the transformations in Winograd's algorithm, which are of the type $X = B^T dB$, we split the operation in two parts, $X' = B^T d$ (performed in the block with subscript 1) followed by $X = X'B$ (performed in the block with subscript 2). All transformations are done using left shifts and adders only. The outputs of each part are registered using flip-flops in order to get timing closure at higher clock frequency. This comes at a cost of two clock cycles in increased latency for each transformation, i.e., input and output.

Each input transformation block consist of five identical $DataT_1$ and $DataT_2$ blocks as seen in Fig. 4.13(a). The inputs to each $DataT_1$ block (e.g. d_{00}, d_{10}, d_{20}, d_{30}, d_{40}) is a column from the tile generated by the line buffer. The $DataT_1$ block performs matrix multiplication of B^T (from Eqn. (3)) and d (input data) to generate intermediate result of input transformation (e.g. Btd_{00}, Btd_{10}, Btd_{20}, Btd_{30}, Btd_{40}). This intermediate result is then passed to each $DataT_2$ block to complete the input data transformation. The output of $DataT_2$ block is forwarded to an 5 x 5 element-wise multiplier array to perform $X \odot Y = (GfG^T) \odot (B^T dB)$. Each $DataT_1$ block requires three 2's complement, five left shift and eleven addition operations, while each $DataT_2$ block requires three 2's complement, three left shift and eleven addition operations.

The output transformation block is shown in Fig. 4.13(b). The result of element-wise matrix multiplication $X \odot Y$ (e.g. XY_{00}, XY_{10}, XY_{20}, XY_{30}, XY_{40}) is passed as the input to each $OutT_1$ block. As mentioned earlier, the transformation blocks are split into two parts, the $OutT_1$ block is responsible to perform $A^T[(GkG^T) \odot (B^T dB)]$ part of the equation 2. This intermediate result (e.g. $AtXY_{00}$, $AtXY_{10}$, $AtXY_{20}$, $AtXY_{30}$, $AtXY_{40}$) is passed as input to $OutT_2$ block which generates the final output of Winograd's algorithm. Five $OutT_1$ and three $OutT_2$ blocks are required to complete the output transformation. Every $OutT_1$ and $OutT_2$ block requires one 2's complement, two left shift and seven addition operations. The final output using Winograd's algorithm accelerator F(3 x 3, 3 x 3) is a 3 x 3 and 2 x 2 matrix for stride-1 and 2

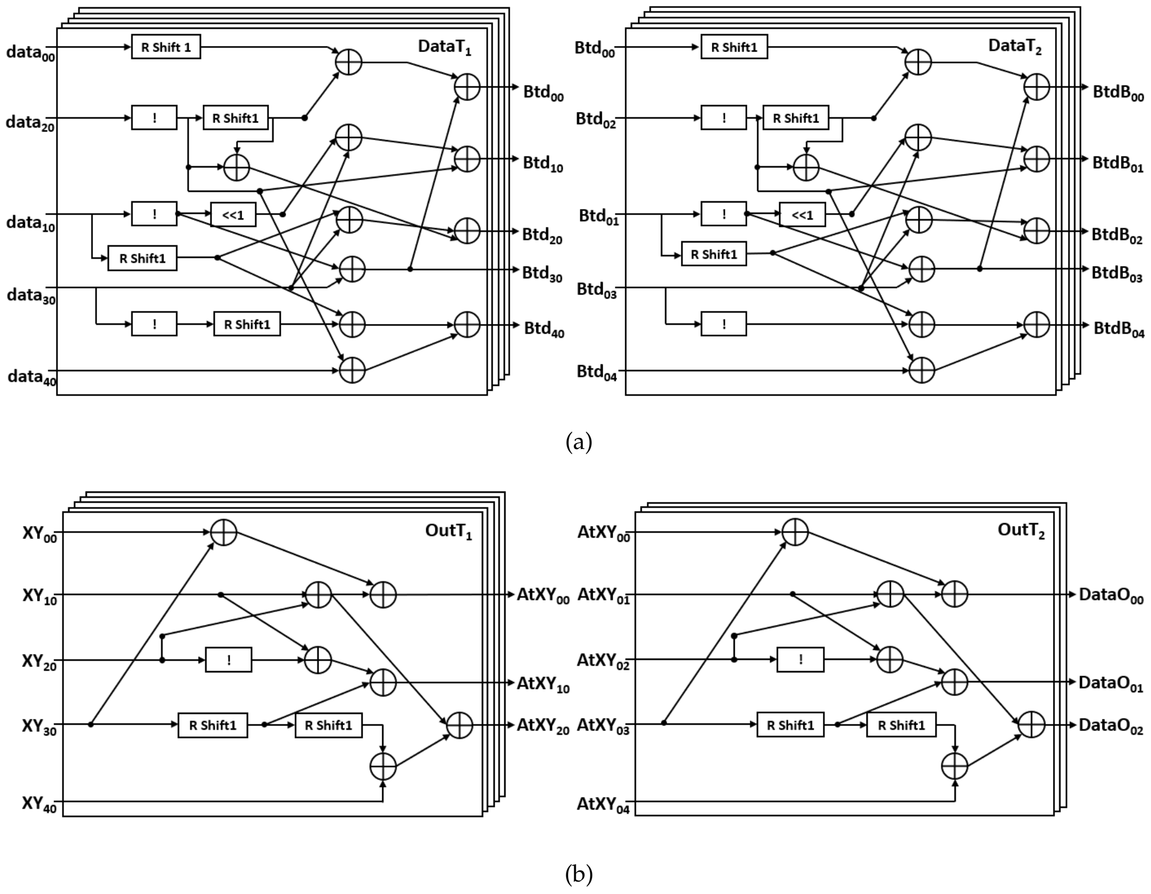


FIGURE 4.13: Transformation block for $F(3 \text{ x } 3, 3 \text{ x } 3)$. (a)Input transformation block, (b) Out transformation block.

convolutions, respectively.

Chapter 5

Results and Discussion

5.1 Dataset Generation

The poly-crystalline diamond (PDC) that we are inspecting is a cylindrical shaped component as shown in Fig. 5.1. It is not possible to image the whole surface of a cylindrical object in one image. Hence an image acquisition system is developed as shown in Fig. 5.1. We used a servo motor which rotates at 10 Hz frequency in a roller fixture to rotate the PDC. We take multiple images of the PDC while it is rotating. A 5 mega-pixel camera is used to image small stripes of the PDC. Once the PDC is completely imaged, the images are passed through a custom code that will collect the stripes of PDC and stich them together. This creates an image of approximately

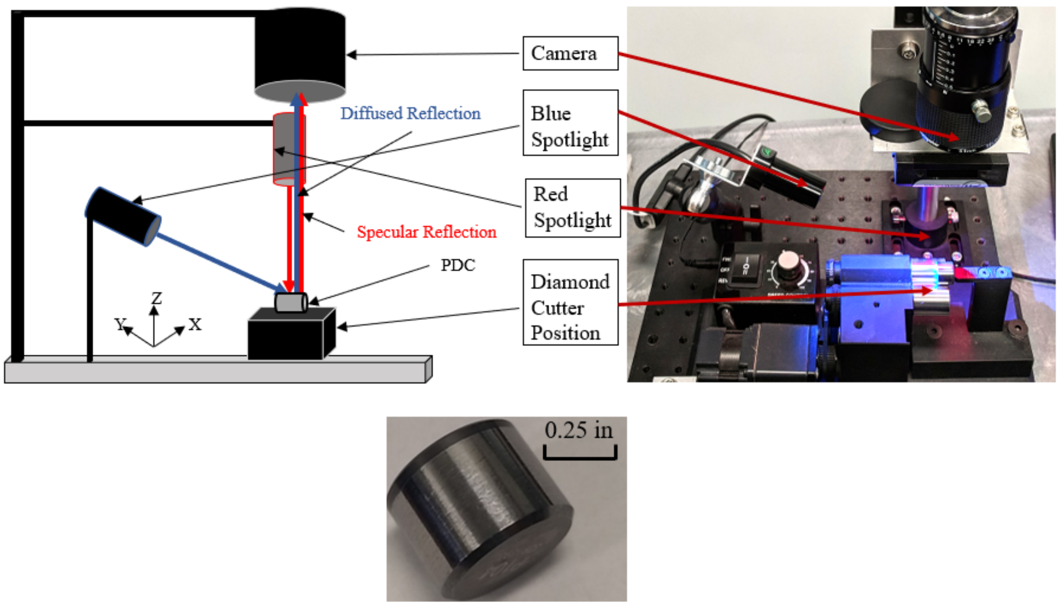
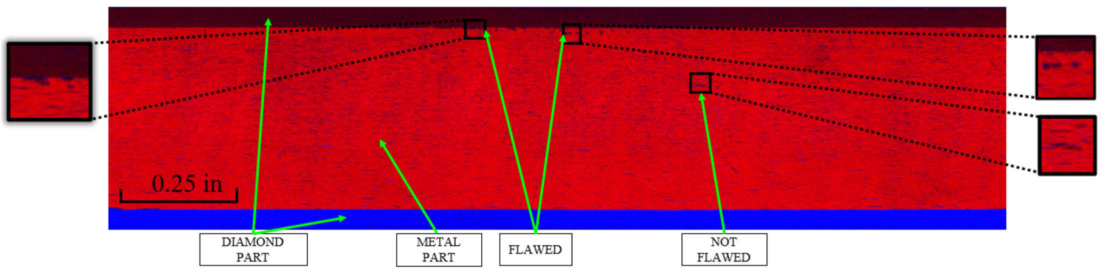
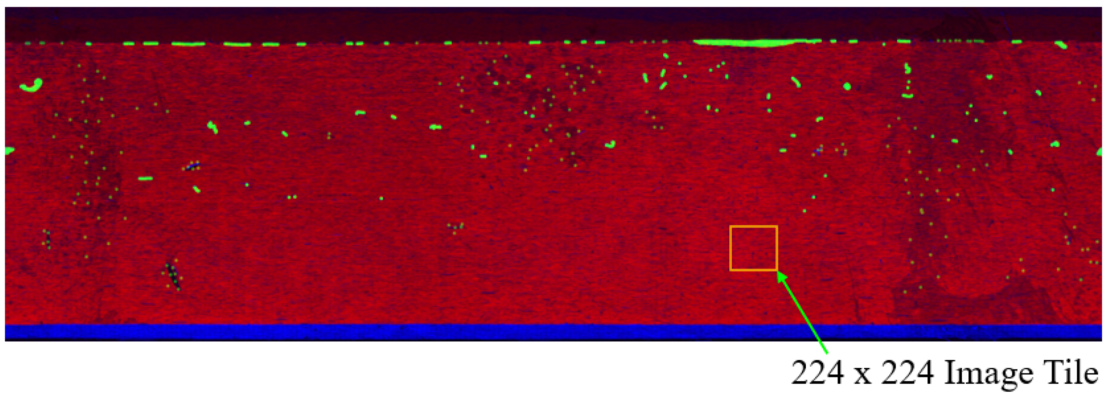


FIGURE 5.1: Image Acquisition Setup and poly-crystalline diamond (PDC) under test.

7000 x 2500 pixels. It is difficult to image the metallic surface in normal lighting due to their high reflectivity. Therefore, we used a special lighting setup in which we used two types spotlights red and blue in a dark room. The red light is positioned in parallel to the camera, while the blue spotlight is positioned at an angle with respect to the camera. The red light reflected from the PDC into camera will have the same angle as the incident light from the spotlight. This setup of red spotlight provides specular reflection. Since the blue spotlight is at a certain angle the reflected light from the PDC into camera will have a certain angle. As the angle of reflection of blue spotlight is not same as incident angle, it provides diffused reflection. This provides us with baseline standard. When looking into the red channel of the image the diamond part of the diamond is medium bright in color while the smooth metallic surface is bright in color. The scratches, dust and stains appear bright while the smooth metal appears dark while the diamond part appears medium dark in the blue channel. When the red and blue color channels are combined we get the image of the PDC as seen in Fig. 5.2(a).



(a)



(b)

FIGURE 5.2: (a) Raw Image of poly-crystalline diamond (PDC) acquired from the imaging setup, (b) Labeled Imaged of poly-crystalline diamond (PDC)

Since the PDC has to be kept at the same location on the roller fixture, it is possible it is not kept consistently at the same location. A realignment algorithm is used which compare the image acquired image with a sample to check for any misalignment and correct it if necessary. The majority of the flaws are on diamond-metal interface and the metallic surface. The image is cropped such that any unnecessary to save valuable processing time. As most neural network takes input image of size 224-by-224 or 227-by-227. IF the image of size 7000-by-2500 is feed directly to the neural network it will be resized to the desired input size. This will downscale the image and remove important details from the raw image. The image is sub-divided into 224-by-224 tiles at a stride length of 32. Each tile of image corresponds to 6-by-6 mil inch of the PDC surface. As only the red and blue spotlights are used, there is no information in the green channel. The green channel is used to label the image as shown in Fig. 5.2(b). This is done by adding the marking the flaw zones with 255 value in the green channel. This is done only on the training and validation dataset by an expert operator. As the neural network architecture expects three channels and there is no spectral information in the green channel. Hence the green channel is filled with the data from the red channel. This can be seen in the image as it is in yellow color as shown in Fig. 5.3(a).

The flawed regions are marked by expert operators in the green channel as shown in Fig. 5.2(b). The training, validation and testing images are generated from 207 PDCs. The neural network is feed with 224-by-224 sized image tiles. The tiles are generated with a stride length of 32 to get a total of 600 tiles from each image. The stride length of 32 is selected in order to center the flawed image tile over the labeled flaw. In any PDC image, the flawed part is smaller than unflawed part. This has to be reflected in training dataset. Burt the number of unflawed tiles should not be too large otherwise we run the risk to create a bias for neural network towards unflawed surface. The training set consists of over 120,314 image tiles with 75 % of unflawed tiles and 25% of flawed tiles. The validation dataset consists of 17,700 image tiles with same ratio of flawed to unflawed tiles. We have further used data augmentation to improve the overall distribution of the under-represented class. We have used only vertical flip in data augmentation due to the limitation of region of interest. We have ensured that the repetition of similar flaws is reduced and a variety of flawed and unflawed features are present in the dataset.

Apart from the PDC dataset, we tested our approach on another dataset. The other dataset used is an open source NEU surface defect database [100] which consists of six types of flaws such as inclusions, crazing, patches, rolled-in scale, scratches and pitted surface. The images in datasets are shown in Fig. 5.3(b) Both the two datasets provide us with wide variety of flaws to examine our proposed approach.

5.2 Model Compressing Techniques Implemented

A fast inference of the part-under-test on embedded platforms can be achieved if the CNN model deployed is compact. The key bottlenecks for fast CNN implementation is memory bandwidth and multiply accumulate (MACs) operations. The above proposed architecture take care of MACs, but the memory issues for embedded platforms still remains. In order to reduce the size and number of weights stored, model compression techniques like parameter pruning and quantization are employed.

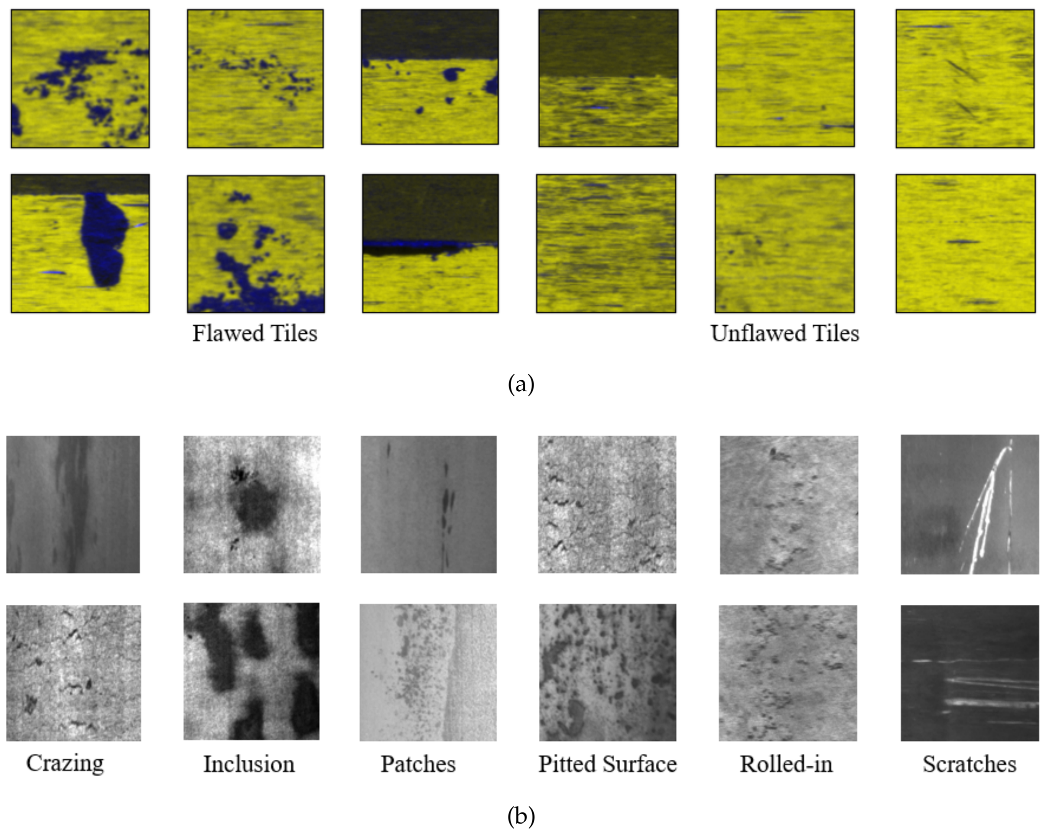


FIGURE 5.3: Datasets used in Experimentation. (a) Poly-Crystalline Diamond Dataset, (b) NEU Surface Defect Dataset.

In parameter pruning, the redundant weights are explored in training and consequently pruned. The quantization technique stores the weights in fixed-point as compared to floating-point representation.

5.2.1 Network Pruning

Algorithm 1: Parameter Pruning

```
1 Create network;
2 Train network to learn features and connections;
3 Set X% sparsity for the network;
4 Gradually increase the sparsity to X′% while fine-tuning the network;
5 Evaluate network and modify final sparsity X′% if required;
```

Network pruning is a efficient way to reduce the complexity as well as size of the network. An elementary approach is to remove unnecessary non-informative weight from the network. Molchanov, et.al, [101] demonstrated that pruning stateof-the-art CNNs does not affect accuracy of the model. A rudimentary method to prune, is to the set a threshold and remove the weights that are below this level. It is a good starting point, but it is very fortuitous. The pruning approach used in our work is presented in Algorithm 1. Once the network is created, we train it for few epochs. This allows the network to build features maps and learn the connections between neurons. Set an initial sparsity level X. This level will tell the algorithm what percent of the weights to be pruned. We iteratively increase the sparsity level to X′. While the level in gradually increased we fine-tune the network and let it recover from loss of parameters to achieve better convergence. We use the API provided by TensorFlow to achieve this. The API takes initial and final sparsity, starting and ending step, frequency of pruning. A starting step is the beginning point and end step is final point of pruning. The frequency is interval after which weights will be pruned.

5.2.2 Weight Quantization

All CNNs employ floating-point variables, as they are deployed on CPUs and GPUs that can handle them. Fixed-point variables are most commonly used on embedded

Algorithm 2: Weight Quantization

```
1 Generate Training graph for your network;
2 Set [Upper bound:$2^N$, Lower bound:0] for N-bit quantization ;
3 Add fake quantization TensorFlow operations in your training graph;
4 Train the network till convergence is achieved;
5 Generate evaluation graph and freeze the variables;
```

platforms as they utilize less hardware resources and are memory efficient. Quantization is representing a floating-point variable as an integer only, i.e. as fixed-point. Since the quantization process is non-linear, the quantization error will propagate and it will result in sub-optimal performance of the network. Hence quantization of a trained network from floating-point to fixedpoint parameters is not an ideal solution. A typical approach to tackle this issue is using simulated quantization [102]. In this process the effects of quantization are taken into account while training a neural network. By quantization, we intend to achieve an inference model with fixed-point parameters that can be easily deployed on embedded platforms. The inference model is nothing but a forward pass in training model. Hence while training with simulated quantization, the parameters in forward pass are quantized to fixed-point. During back-propagation or backward pass, the parameters are kept in their original floating-point representation. Algorithm 2 describes the procedure to achieve simulated quantization.

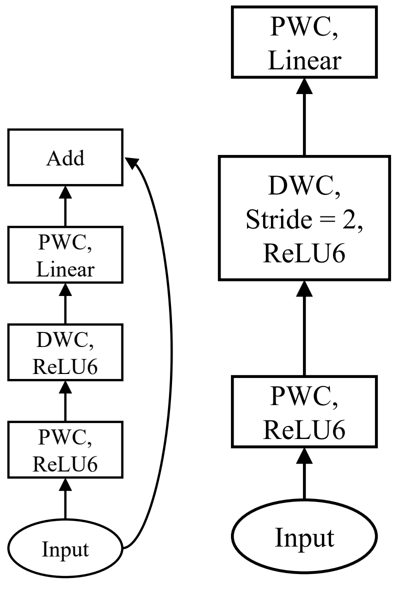
5.3 Neural Network Architecture

5.3.1 Heuristically built Convolution Neural Network

All modern efficient convolution neural network architectures are based upon depthwise separable convolution (DSC) blocks [106, 107]. As the name suggests, the standard convolution (SC) is factorized into two separate operations. The first operation is depthwise convolution (DC) followed by pointwise convolution (PC). The SC involves convolution of input channels with a specific kernel followed by summation of the convolution results into one channel. In DC, each input layer performs the convolution on its respective kernel and generates one output channel. Pointwise convolution is simply a SC performed using a 1 x 1 kernel. The purpose of using depthwise separable convolution over the standard convolution scheme is to significantly reduce, both the number of operations and parameters. For an input tensor of N x N x P and a kernel size of M x M x P x K, the number of weights (WSC) and number of operations (OSC) needed for the SC (for stride = 1) is given by [106]: Similarly, for depthwise separable convolution, the weights and number of operations become

The depthwise separable convolution effectively reduces the parameters such as weights and number of operations by a factor of: $ORF = OSC/ODSC = M2_K/M2 + K$ Typically, for a kernel size of M = 3, the reduction factor in number of operations and weights is about 8 to 9 times. Recent architectures like MobileNetV1/MobileNetV2 [106, 107] use depthwise separable convolution. MobileNetV2 is known to be one of the highly efficient small scale networks in the ImageNet competition [108]. In this competition, the CNNs are trained on millions of images belonging to thousands of classes and are tested for top-1 or top-5 accuracy. As seen in Fig. 5.3, the features in the image of the part-under-test are fewer when compared to features that are found in the ImageNet database [108]. The features are diverse and complex in the ImageNet database, while the deep layers in the CNN are trained to identify them. The CNN trained on ImageNet database will attempt to find complex features that are not present in the target database, thus resulting in poor efficiency. It can be argued that transfer learning should be used, in which the last fully-connected (FC) layer of the CNN pre-trained on a database (typically ImageNet) is removed and a new FC layer consisting of classes in the target database is created. This CNN is then retrained on the target database. However, the sole purpose of building deep CNNs is their ability to identify complex features and to distinguish between thousands of classes, which is not the case in the target database.

We implemented a classification scheme in NDTNet (Nondestructive testing network) which employs depthwise separable convolution. The NDTNet starts with an input layer that accepts an input tensor of size (224-by-224-by-3). It is followed by a zero-padding layer in order to include boundary pixels in the convolution. This is succeeded by a standard convolution layer, a batch-normalization layer and a rectified linear unit (ReLU) layer. The batchnormalization layer helps to control the sensitivity of the initial weights. It keeps a check on the activation function from going too high or too low. Finally, the batch-normalization helps to avoid over-fitting of the data. Over-fitting implies the neural network memorizes the data instead of learning the data. The above layers are followed by a combination of depthwise separable convolutions, batch-normalization and ReLU layers which is termed as bottleneck [107], as seen in Fig. 5.4. The detailed architecture is demonstrated in Table I. The dense layers at the end of the network are used to obtain the class scores. In order to reduce over-fitting, we use dropout layers following the dense layers. Based on the dropout value (0.0-1.0), the neurons in this particular layer are deactivated randomly. This forces the layer to learn the same features with different neurons which in return provides better generalization. The dropout is only active during the training phase. The first few layers of any CNN capture generic features like color, patches, blobs, edges, lines, etc. Thus, the initial layers of the NDTNet are similar to those of MobileNetV2.



(a) Stride = 1 (b) Stride = 2

FIGURE 5.4: Bottleneck architecture for two different strides.

Since both networks use depthwise separable convolution, it will enable us to use the weights of MobileNetV2 trained on the ImageNet database. Through this approach, we can fine-tune the weights of the initial layers that identify generic features rather than randomly initializing them. The remaining layers are trained from scratch on the target database. The depth of NDTNet is heuristically determined after various experiments on the target database while examining the convergence of training and validation accuracy. The NDTNet architecture that was finalized is detailed in Table 5.1. When compared to other state-of-the-art network, NDTNet requires least number of parameters and the size of weights is considerably small. As seen in Table 5.2, NDTNet has one fifth the number of trainable parameters as compared to MobileNetV2. This significant parameter reduction provides us with a compact structure that reduces the training and, more importantly, the classification time; this makes it suitable for real-time implementation on the factory floor. Furthermore, the reduction in size and parameter will allow NDTNet to be implemented on inexpensive embedded devices like FPGA and TPUs.

TABLE 5.1: NDTNet Architecture

Input Tensor	Layer	Output Channel	Stride	Repeat Factor
224 x 224 x 3	SC	32	2	-
112 x 112 x 32	Bottleneck	16	1	-
112 x 112 x 16	Bottleneck	24	2	2
56 x 56 x 24	Bottleneck	32	2	2
28 x 28 x 32	Bottleneck	32	2	3
14 x 14 x 32	Bottleneck	72	2	2
7 x 7 x 72	Bottleneck	432	1	2
7 x 7 x 432	Avg. Pool	432	-	-
1 x 1 x 432	Dense	-	-	-
1 x 1 x 108	Dense	-	-	-
1 x 1 x 2	Dense	-	-	-

TABLE 5.2: Comparison of size and computational cost with other state-of-the-art CNNs

Network (Reference)	Parameters (Million)	Weight (MB)	Time (Images/sec)
VGG16 [103]	15 M	28	360
ResNet50 [104]	25 M	48	380
DenseNet121 [105]	8.6 M	20	375
MobileNetV2 [107]	3.6 M	8	410
NDTNet	0.7 M	2.5	425

5.3.2 AutoML based Convolution Neural Network

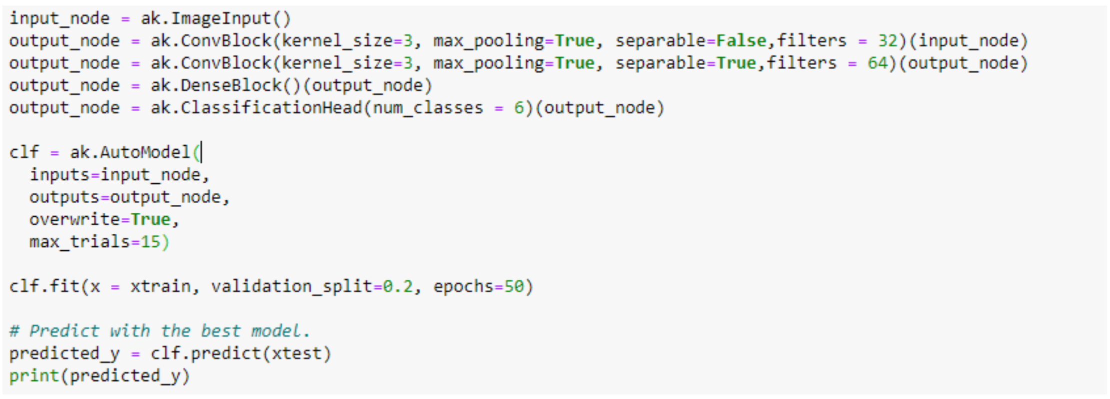
AutoML means automatic machine learning and its objective is to enable the users whose expertise is not machine learning to use it in their domains. There are several AutoML platforms and services that are predominantly available on large cloud computing infrastructure. But there are many reasons that prohibits users from using them. First, the AutoML that is implemented on cloud-computing infrastructure require complicated Docker container configurations and use of kubernetes. Some one who doesn't have a computer-science background would have tough time to using a cloud-based AutoML frameworks. Second, the cost to use the cloud-based depends upon the hours of service used. Someone that is new to this domain would end up spending more money than required. Finally, there is no guarantee of security or privacy of the data. If there is a AutoML platform which runs locally it would be idea for new users or to those whose applications are not complex. One such AutoML platform in AutoKeras [110], which is an open-sources software is ideal for such use case. It provides configurable application programming interface (API) which can be easily used by users without deep understanding of computer-science. For advanced user, AutoKeras allows to to configure details of the system like hyperparameters and layers to meet their requirements. In next subsection we will discuss how AutoKeras was used to generate compact convolution network. Only the front-end details are described as the back-end implementation is beyond the scope of this thesis. I encourage the readers to read the AutoKeras paper [110] for further details on backend implementation of the platform.

Using Autokeras

There are two important components of Autokeras viz., Automodel and Block. Automodel is the base class for the classifier and it combines the hypermodel with the tuner to tune the hypermodel. AutoKeras allows us to stack such blocks sequentially to create a classification pipeline. At the start of each classification pipeline one has to provide a 'Node' which tells what type of data is being passed. The 'Block' can be anything from pre-processing layer to a full-fledged convolution neural network.

The users can choose to use an existing neural network architecture in their pipeline or provide parameters to build their own network.

In the implementation, I used both standard convolution as well as separable convolution layers. I started with a ImageInput block as the convolution neural network is dealing with images. This was followed by convolution block with kernel size set to three. In order to add separable convolution layer, we just need to set the separable argument of the ConvBlock API to true. For separable layers, the kernel size was set to three as well. It is desirable to have more separable layers than standard convolution layers simply due to there effectiveness in making model compact as described above. The number of filters in standard convolution block were set to 32 and for separable convolution block were set to 64. Then a dense block was added to the pipeline which would also add a flatten layer. Finally a classification head is joined to classify into bins. The snippet of the basic code to build the model is shown in Figure 5.5. The AutoKeras will add the max-pooling layers wherever necessary. It will also add dropout layers in the dense block to avoid overfitting. One needs to set the maximum trials to 15 for the Autokeras to generate a model. One important thing about AutoKeras is the searching of a network and training of a network is done parallely by the CPU and GPU respectively. Hence it is not necessary that as the trial count increases we would get networks that perform better than previous. As a user one can choose to store all the network models generated by AutoKeras or overwrite the best performing model with even better performing model in consequent iterations.



```
input_node = ak.ImageInput()
output_node = ak.ConvBlock(kernel_size=3, max_pooling=True, separable=False,filters = 32)(input_node)
output_node = ak.ConvBlock(kernel_size=3, max_pooling=True, separable=True,filters = 64)(output_node)
output_node = ak.DenseBlock()(output_node)
output_node = ak.ClassificationHead(num_classes = 6)(output_node)

clf = ak.AutoModel(
  inputs=input_node,
  outputs=output_node,
  overwrite=True,
  max_trials=15)

clf.fit(x = xtrain, validation_split=0.2, epochs=50)

# Predict with the best model.
predicted_y = clf.predict(xtest)
print(predicted_y)
```

FIGURE 5.5: Psuedo Code snippet of how to build CNN using AutoKeras.

Although AutoKeras is based upon Keras platform, it does not natively support any model compression algorithms. One cannot use any tensorflow APIs to perform quantization or pruning on AutoKeras generated model. We wanted to optimize the model generated by AutoKeras further by implementing parameter pruning and data quantization as shown earlier in Algorithm I and II. But to do this we first need to convert the AutoKeras model into its Keras equivalent so that we can use the trained weights as well as use the Tensorflow APIs. As mentioned earlier we have an option to save only the best performing model. So once the initial training is done, we load the best performing model. We wrote a function which would take this model and based on its layers will create a sequential Keras model. The first layer is Input layer (from Keras) as opposed to ImageInput layer (from AutoKeras) with shape of (224, 224, 3). Then based on the layer i.e convolution, separable, flatten, dense, max-pooling we will add that layer with the same number of filters kernels in each layer and each filter kernel is of size 3. Once we succesfully build the model, then we can use the Tensorflow APIs to quantize or prune the model. For quantization, we used quantization aware APIs available from Tensorflow. In this network, we only quantized the dense layers. For parameter pruning, we performed pruning on all layers as followed the Algorithm I. The lower and upper bounds for sparsity was set to 30 and 80 percent respectively.

5.4 Implementation Details

The software part of the algorithm which involves created a neural network and its training or using Autokeras to generate a neural network as well as the model compression algorithms are coded in python 3.6 on deep learning platforms like Keras [109] and Tensorflow. We have used AutoKeras platform which is also based on Keras, to generate multiple CNN candidates for a given dataset. All the training was done on a single local machine which housed Intel core i7-8700k CPU with Nvidia GTX-1080 TI GPU with 11 GB of memory. In order to increase the training speed, the intermediate training weights were stored on M.2 PCIE NVMe SSD storage on the machine.

For heuristically built convolution neural network, we trained the CNN for 20 epochs with each epoch having $\frac{no.\,of\ Images}{Batch\ size}$ iterations. The batch size was set to 32 as higher batch size sometimes result in out of memory (OOM) exception from GPU. As mentioned in previous section, the ratio of flawed to unflawed images in the PDC dataset is highly unbalanced. When we use such datasets, we have to take certain measures so that the CNN does not develop a bias towards a class in dataset that is large in number. To tackle this issue we used class weights, which trains the model in such a way that it pays more attention towards the under-represented class of the dataset. The class weight is a parameter passed while training and it was set to 3:1 flawed to unflawed ratio. We used L1 regularization and dropout in dense layers to avoid overfitting. The initial learning rate for Adam optimizer was set to 4.5×10^{-3} with 2×10^{-5} weight decay. For quantization aware training we trained the model for 20 epochs with each epoch having 100 steps. Similarly, for parameter pruning the number of epochs were set to 20 but he iterations in each epoch was equal to $\frac{no.\,of\ Images}{Batch\ size*2}$. The important components in this parameter pruning algorithm is initial and final sparsity, begin and end step, and finally frequency at which to prune. For initial and final sparsity, as mentioned above we selected 30% and 80% respectively. The begin step is start of the training i.e. 0^{th} step and end step is $no.\ of\ iterations\ per\ epoch * (epoch - 2)$. The frequency at which the parameters are pruned is set to once every two epochs. This is done so that the network gets ample training to recover from the pruning. For AutoKeras implementation, the maximum trails to search a best performing candidate network was set to 15. This number is heuristically found and setting a very high number can result in unnecessary waste of training time. The model compression algorithms used after finding the best candidate follows the same strategy as discussed above.

The hardware implementation is done using verilog language on Xilinx Platform. We implemented on two types of microprocessor cores; first Xilinx proprietary Microblaze softcore and Arm core on Nexys Video FPGA and Zynq ZCU104 FPGA boards respectively. On both the boards we used a direct memory access Xilinx IP to remove the host intervention during memory transfer which would allow for faster and efficient access with the external DDR memory.

5.5 Results and Discussion

In this subsection we perform a quantitative evaluation of the NDTNet and compare it with the state-of-the-art CNNs. First, we compare the parameters of the proposed NDTNet with other networks. As seen in Table 5.2, the proposed NDTNet requires the least number of parameters and size of weights. Due to the size of NDTNet, it can sample the images faster than any other state-of-the-art networks. Hence, it is more suitable to enforce such a network on a real-time basis on the factory floor.

In classification schemes that involve imbalanced data sets, using the accuracy alone is not a satisfactory metric. For example, let us assume there are 100 samples of which 90 are unflawed and 10 are flawed. If the classification scheme determines each sample to be unflawed, we would still obtain 90% accuracy. For better quantitative evaluation of the proposed algorithm we are using statistical indicators like precision, recall, average precision and F1 score as defined [111]:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$Avg.\ Precision = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5.4}$$

Here TP denotes true positive, TN is true negative, FP is false positive and FN is false negative. TP and FP are the defect regions correctly and incorrectly identified. In Table 5.3, we evaluate the performance of the proposed NDTNet for different training methods. While the average precision and F1 score statistics give us an overview of a particular network, it is the precision and recall indicators that are of key importance to us. High precision means the algorithm returns more appropriate than inappropriate results, while high recall implies the algorithm returns mostly appropriate results.

5.5.1 Results for Heuristically built NDTNet

The proposed NDTNet achieves 98.7% precision and 99.3% recall, respectively, when partial transfer learning is adopted. When NDTNet is trained from scratch, a drop in performance is observed. A state-of-the-art network, like MobileNetV2, has a similar architecture, hence we evaluate it by training it from scratch as well as, using the same transfer learning. It can be seen that we achieve similar results as the propose NDTNet for both training methods. The best performance for surface flaw classification is achieved when the transfer learning method is adopted. This is because the weights in the initial layers are not randomly assigned. Most recent flaw evaluation techniques discussed in the literature that use neural networks employ VGG16 due to its simplicity. To assess the proposed NDTNet against other networks, we analyzed our data set on a VGG16 based network as proposed by Tao et.al. [114] and Sun et .al. [112] for classification accuracy and speed to implement. The results are similar in terms of classification accuracy; reference [112] reported that they can classify one image (224,224,3) in 39 ms, while the proposed NDTNet can classify one image in 2.35 ms. This is due to the architecture of the network used, as shown in Table 5.2, the parameters would make it difficult to implement a VGG16 based network on cost effective embedded devices like FPGAs. Further, we also tested our database on conventional feature extraction (Gabor) and classification (SVM) discussed in [113]. We observed poor classification accuracy, while the time required for feature extraction and classification is very large. This performance is expected as feature extraction and classification is useful when we are inspecting patterns or textures. Typically flaws on metallic surfaces are sparse and randomly distributed, hence conventional feature extraction and classification methods do not

TABLE 5.3: Performance Evaluation of NDTNet with MobileNetV2 for various training methods

Network	Train. Method	Pre. %	Rec. %	Avg. Pre. %	F1 Scr. %
NDTNet	Part. Trans. Learn.	98.7	99.3	97.6	99.0
MobileNetV2 [107]	Trans. Learn.	97.2	98.6	96.7	97.9
VGG16 [112]	Trans. Learn.	96.5	97.2	95.9	96.8
Gabor + SVM [113]	Scratch	80.4	81.6	79.5	80.8
NDTNet	Scratch	93.3	96.2	92.0	94.7
MobileNetV2 [106]	Scratch	93.2	96.4	91.9	94.8

perform well.

Now let us see the performance of proposed NDTNet when model compression algorithms discussed above are used. We demonstrate the performance of the proposed NDTNet for four cases. For the first case (Normal), the precision of parameters is kept 32-bit floating point. In second case (Weight Pruning), the parameters are pruned for up to 90% sparsity of the network. The parameters are quantized to 8-bit fixed point precision in the third case (Quantization). Finally, both weight pruning for 90% sparsity and quantization to 8-bit fixed point of parameters is done in case four. The results for all the cases is tabulated in Table 5.4. As it can be seen in Table 5.4, we achieve three times size reduction by weight pruning and up to eighteen times by weight pruning along with quantization. Despite of compromising on the precision as well as number of parameters, the performance is reduced by less than 2%. The comparison of the proposed NDTNet with other state-of-the-art CNNs, like VGG16 [112], which are widely used in inspection techniques [113] and MobileNetV2 [107] which was specifically developed to be implemented on mobile devices, is shown in Table 5.5. Although, there is no significant difference in precision, the size of NDTNet is much smaller than the state-of-the-art networks. We compare the proposed NDTNet, with a VGG16 based network [112] developed for optical nondestructive testing. Due to the fact that the type of flaws are different, it is difficult to compare the statistical indicators. Instead, we will compare the speed of inference Since VGG16 and MobileNetV2 were designed to classify images in thousands of classes, they are large in size. For a pass-fail classification scheme, a compact CNN like NDTNet performs just as well as the state-of-the-art. This performance paves the way to implement such a compact CNN on cost-effective embedded platforms like FPGAs.

TABLE 5.4: Performance Evaluation of NDTNet for Different Approaches

Approach	Size (MB)	Prec. %	Rec. %	Acc. %	F1 Scr. %	Speed (FPS)
Standard	3.6	98.7	98.35	97.79	98.49	176.54
Wgt. Prun.	1.3	98.34	97.8	97.12	97.89	181.37
Quant.	0.95	96.69	97.03	95.3	96.85	191.17
Wgt. Prun. with Quant.	0.2	98.01	97.64	96.8	97.82	198.01

TABLE 5.5: Comparison of NDTNet with other state-of-the-art CNNs

Network (Reference)	Parameters (Millions)	Weight (in MB)	Precision %
VGG16 [112]	15 M	28	97.30
MobileNetV2 [107]	3.6 M	10	98.70
NDTNet	0.25 M	3.6	98.85

5.5.2 Results for AutoML built NDTNet

We tested the proposed AutoML based NDTNet framework on two datasets mentioned above. The CNN architectures obtained from the AutoKeras platform are listed in Table 5.6. A slight difference in the architectures for the two datasets is observed, which means AutoKeras tuned the architecture according to the features in a dataset.

The efficiency of Winograd's algorithm when accelerating convolutions on some widely used CNNs like VGG16 [112] and MobileNetv2(MNetv2) [107] as well as on the NDTNet [115, 116] and NDTNetv2 is demonstrated in Table 5.7. NDTNet is the previous iteration of the proposed work in which the compact CNN model was developed heuristically along with model compression techniques. It is observed that

TABLE 5.6: NDTNETv2 Architecture generated by AutoKeras for Different Datasets

Dataset	Input Tensor	Layer	Output Channel	Stride	Repeat Factor
PDC Dataset	224x224x3	SC	32	2	2
	114x114x32	SC	32	2	2
	60x60x32	DSC	64	2	2
	32x32x64	DSC	64	2	2
	16x16x64	Pool	1024	-	-
	1x1x1024	FC	-	-	-
	1x1x32	FC	-	-	-
	1x1x2	FC	-	-	-
NEU Surface Defect Dataset [100]	224x224x3	SC	32	2	2
	114x114x32	SC	32	2	2
	60x60x32	SC	32	2	2
	32x32x32	DSC	64	2	2
	16x16x64	DSC	64	1	2
	16x16x64	DSC	64	1	2
	16x16x64	Pool	1024	-	-
	1x1x1024	FC	-	-	-
	1x1x32	FC	-	-	-
	1x1x6	FC	-	-	-

an operation reduction of around 60% in million operations MOPs for each CNN when the Winograd convolution accelerator is used compared to conventional convolutions. Transfer learning was adopted while training VGG16 and MobileNetv2, while trained NDTNet and NDTNetv2 from scratch. It can be seen in Table 5.7, the precision for NDTNetv2 is marginally better than other CNNs for both dataset 1 (DS1) consisting of PDC images and dataset 2 (DS2) which is NEU surface defect database [100].

I implemented the proposed Winograd convolution accelerator on Zynq ZCU104 and Nexys Video FPGA boards. Table 5.8 shows resource utilization of the proposed implementation on both the boards operating at 100 MHz. Since Winograd's algorithm uses less number of multiplications compared to a conventional approach, it enabled us to achieve higher throughput. We used 76% and 89% DSP slices (used for multiplications) on Zynq and Nexys FPGAs respectively. Each multiplication operation uses one DSP, we used 16-bit fixed point representation. After each convolution layer we have Relu6 which gives results between [0, 6]. We use QS3,8 fixed point when storing the result of each layer. However, we maintain precision in the intermediate stages. As Winograd's algorithm compensates reduction in multiplication by additions, we require more LUTs than usual to perform arithmetic operations seen in input and output transformation. We required over 90% of available LUT resources on both FPGA boards to implement the proposed accelerator. As mentioned earlier, on-chip data storage is the biggest bottleneck for CNN implementations on FPGA, we managed to use almost all of the available Block RAMs (BRAMs) to cache the input and output feature maps as well as to store the intermediate feature maps. The use of flip-flops (FFs) is done to achieve timing requirements.

In order to conduct a fair comparison between different Winograd convolution

TABLE 5.7: Performance of WINOGRAD based Convolution in different CNN Models

CNN Model	Convent. MOPs	Wino. MOPs	Saving Rate(%)	Precision(%) DS1	DS2
VGG16 [112]	30800.2	11597.6	62.3	96.5	95.7
MNetv2 [107]	585	244.82	58.1	97.2	96.5
NDTNet [115]	278.1	115.77	58.4	98.7	96.9
NDTNetv2	188.2	80.36	57.3	99.1	98.6

TABLE 5.8: Resource Utilization of NDTNETv2 on ZYNQ-ZCU104 and NEXYS VIDEO

Board	Resources	LUTs	BRAMs	DSP	FFs
ZYNQ-ZCU104	Available	230k	412	1728	460k
	Used	221k	381	1312	132k
	Utilization	91%	93%	76%	29%
NEXYS VIDEO	Available	134k	365	740	269k
	Used	129k	360	656	75k
	Utilization	96%	99%	89%	28%

accelerator implementations, we use DSP efficiency which is in giga operations per seconds per DSP (GOPS/DSP). The throughput of an implementation depends on resources used in parallel which can vary greatly depending on the FPGA board used. We managed to achieve 0.93 and 0.82 GOPS/DSP efficiency with Zynq and Nexys boards respectively. Even though both the boards use Winograd F(3 x 3, 3 x 3), we had to slightly adjust the implementation on the Nexys board in order to manage the resources while achieving desired throughput. Hence, the Nexys board achieve slightly less DSP efficiency than Zynq board. In Table 5.9, we compare our proposed Winograd implementation with previous state of-the-art Winograd implementations. We have comparable DSP efficiency with the previous works while having similar resource utilization. While reference [118] shows impressive DSP efficiency, their design is optimized for standard VGG16 convolutions. The presented implementation can perform in addition to standard convolution (SC), depthwise convolution (DC), pointwise convolution (PC), and depthwise separable convolution (DSC) as needed in modern CNNs. Furthermore, we can achieve 198 frames

TABLE 5.9: Performance Comparison with STATE-OF-THE-ART WINOGRAD based Convolution Accelerators for FPGA

	[116] 2017	[117] 2018	[118] 2020	Proposed Work	
Platform	ZYNQ ZC706	VCU 440	ARRIA 10	ZYNQ ZCU104	NEXYS VIDEO
Freq. [MHz]	100	200	250	100	100
CNN Model	VGG-16	VGG-16	VGG-16	NDTNetv2	NDTNetv2
Precision	16-bit	16-bit	16-bit	16-bit	16-bit
Winograd	F(4x4, 3x3)	F(2x2, 3x3)	F(2x2, 3x3)	F(3x3, 3x3)	F(3x3, 3x3)
LUTs	156k	189k	181k	210k	129k
DSP	725	1376	1344	1312	656
GOPS	660	821	1642	1218	534
GOPS/DSP	0.91	0.6	1.22	0.93	0.82

per second on the Zynq board, while the Nexys board can achieve 152 frames per second.

Chapter 6

Conclusion

6.1 Conclusion

This dissertation proposed a novel compact convolution neural network approach called NDTNet for metallic surface flaw detection. The NDTNet addresses important challenges such as consistency, accuracy and speed in nondestructive testing of metallic parts. The proposed system consists of image acquisition, pre-processing and classification. It proposes a framework based on AutoKeras to automatically design and optimize CNNs for metalllic surface flaw inspection. By using transfer learning, we find a compact CNN using AutoKeras platform for a particular dataset. The architecture of NDTNet consists of mostly depth-wise separable convolution layers which enables us to reduce the number of parameters. The result is a significant increase in the classification speed. When compared to other state-of-the art CNNs developed for nondestructive testing of metallic surfaces, NDTNet is almost 15 times faster. Second, we optimize the CNN using model compression techniques like parameter pruning and quantization. Finally, to accelerate the convolutions on hardware, a Winograd's Convolution Algorithm is employed. For 2-D convolutions, the $F(2 \text{ x } 2, 3 \text{ x } 3)$ Winograd is very popular. However, most Winograd base accelerators are capable of performing stride-1 convolutions only.This dissertation developed a $F(3 \text{ x } 3, 3 \text{ x } 3)$ Winograd so that we are able to accelerate both Stride-1 and Stride-2 convolutions. For stride-1 convolutions, the proposed Winograd is 3.25x computationally efficient, while stride-2 is 1.44x efficient than standard convolution algorithm. Without any optimizations, the classification statistics of NDTNet such as precision and recall are not compromised. Based on practical testing performed on approximately 100 PDCs, it can be seen that NDTNet achieves a precision of 98.7% and a recall of 99%. Thus, the results obtained so far suggest that the accuracy and speed of the proposed NDTNet is suitable for real-time optical nondestructive testing of metallic surfaces. When model compression techniques are used on NDTNet, we observe precision and recall statistics of 96-98%. For hardware implementation, the Winograd based convolution accelerator was implemented on Zynq ZCU104 and Nexys Video FPGA boards. The proposed system achieves a comparable 0.93 and 0.87 GOPS/DSP performance efficiency, respectively. When evaluated the proposed framework on two different datasets of metallic objects with varying degrees of flaws were employed. First dataset consisted of PDCs which had surface pitting flaws, while second dataset has six types of flaws like crazing, inclusion, patches, pitting, rolled-in surface and scratches. The framework was able to achieve 98% of precision on both the datasets, as discussed in Chapter 5.

6.2 Future Work

This work can be further enhanced by adding a localization of an object or flaw feature in addition to classification. Furthermore, AutoML optimizations can be incorporated to improve the time required to find the best candidate CNN. Finally, an automated script that finds optimized CNN candidate and produce TCL scripts to generate hardware depending on the available FPGA resources on board should be explored as part of future research efforts.

Bibliography

[1] J. Song, S. Kim, Z. Liu, N. N. Quang and F. Bien, "A Real Time Nondestructive Crack Detection System for the Automotive Stamping Process," in *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 11, pp. 2434-2441, Nov. 2016.

[2] N. Athi, S. R. Wylie, J. D. Cullen, A. I. Al-Shamma'a, T. Sun, "Ultrasonic non-destructive evaluation for spot welding in the automotive industry," in *Proc. IEEE Sensors*, pp. 1518-1523, Oct. 2009.

[3] M. N. Bassim, M. P. Dudar, R. Rifaat, R. Roller, "Application of acoustic emission for nondestructive evaluation of utility inductive reactors," in *IEEE Trans. Power Del.*, vol. 8, no. 1, pp. 281-284, Jan. 1993.

[4] H. N. Nagamani, T. B. Shanker, V. Vaidhyanathan, S. Neelakantan, "Acoustic emission technique for detection and location of simulated defects in power transformers," in *Proc. IEEE Russia Power Tech*, pp. 1-7, Jun. 2005.

[5] J. J. G. de la Rosa, R. Piotrkowski, J. E. Ruzzante, "Higher order statistics and independent component analysis for spectral characterization of acoustic emission signals in steel pipes," in *IEEE Trans. Instrum. Meas.*, vol. 56, no. 6, pp. 2312-2321, Dec. 2007.

[6] J. Lim, T. Kaewkongka, "Micro cracking in stainless steel pipe detection by using acoustic emission and crest factor technique," in *Proc. IEEE Instrum. Meas. Technol. Conf. (IMTC)*, pp. 1-3, May 2007.

[7] E. Coffey, "Acoustic resonance testing," in *2012 Future of Instrumentation International Workshop (FIIW)*, Gatlinburg, TN, 2012, pp. 1-2.

[8] V. Sankaran, “Low Cost Inline NDT System for Internal Defect Detection in Automotive Components Using Acoustic Resonance Testing," in *Proceedings of the National Seminar & Exhibition on Non-Destructive Evaluation, NDE 2011*, December 8-10, 2011.

[9] M. Cabrera, X. Castell and R. Montoliu, “Crack detection system based on spectral analysis of an ultrasonic resonance signals," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '03)*, 2003, pp. II-605-8 vol.2.

[10] L. Zhiyong, Z. Qinlan and L. Xiang, “New Magnetic Particle Cassette NDT Intelligent Detection Device," in *2013 Fourth International Conference on Intelligent Systems Design and Engineering Applications,* Zhangjiajie, 2013, pp. 403-406. doi: 10.1109/ISDEA.2013.496

[11] J. A. Buck, P. R. Underhill, J. E. Morelli and T. W. Krause, “Simultaneous Multiparameter Measurement in Pulsed Eddy Current Steam Generator Data Using Artificial Neural Networks," in *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 3, pp. 672-679, March 2016.

[12] L. S. Rosado, F. M. Janeiro, P. M. Ramos, M. Piedade, “Defect characterization with eddy current testing using nonlinear-regression feature extraction and artificial neural networks," in *IEEE Trans. Instrum. Meas.*, vol. 62, no. 5, pp. 1207-1214, May 2013.

[13] A. Bernieri, L. Ferrigno, M. Laracca, M. Molinara, “Crack shape reconstruction in eddy current testing using machine learning systems for regression," in *IEEE Trans. Instrum. Meas.*, vol. 57, no. 9, pp. 1958-1968, Sep. 2008.

[14] J. A. Buck et al., “Pulsed eddy current inspection of support structures in steam generators," in *IEEE Sensors J.*, vol. 15, no. 8, pp. 4305-4312, Aug. 2015.

[15] P. Horan, P. R. Underhill, T. W. Krause, “Pulsed eddy current detection of cracks in F/A-18 inner wing spar without wing skin removal using modified principal component analysis," in *NDT E Int.*, vol. 55, no. 1, pp. 21-27, Apr. 2013.

[16] C. A. Stott, P. R. Underhill, V. K. Babbar, T. W. Krause, "Pulsed eddy current detection of cracks in multilayer aluminum lap joints," in *IEEE Sensors J.*, vol. 15, no. 2, pp. 956-962, Feb. 2015.

[17] A. Sophian, G. Y. Tian, D. Taylor, J. Rudlin, "A feature extraction technique based on principal component analysis for pulsed eddy current NDT," in *NDT E Int.*, vol. 36, no. 1, pp. 37-41, 2003.

[18] D. R. Desjardins, G. Vallières, P. P. Whalen, T. W. Krause, "Advances in transient (pulsed) eddy current for inspection of multi-layer aluminum structures in the presence of ferrous fasteners," in *Proc. AIP Conf.*, vol. 1430, no. 1, pp. 400-407, 2012.

[19] D. Vasić, V. Bilas, D. Ambruš, "Pulsed eddy-current nondestructive testing of ferromagnetic tubes," in *IEEE Trans. Instrum. Meas.*, vol. 53, no. 4, pp. 1289-1294, Aug. 2004.

[20] N. H. Jo, H.-B. Lee, "A novel feature extraction for eddy current testing of steam generator tubes," *NDT E Int.*, vol. 42, no. 7, pp. 658-663, 2009.

[21] Y. Cheng, L. Bai, F. Yang, Y. Chen, S. Jiang and C. Yin, "Stainless Steel Weld Defect Detection Using Pulsed Inductive Thermography," in *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 7, pp. 1-4, Oct. 2016.

[22] E. Wu, Y. Ke and B. Du, "Noncontact Laser Inspection Based on a PSD for the Inner Surface of Minidiameter Pipes," in *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 7, pp. 2169-2173, July 2009.

[23] O. Dura, "Pipe inspection using a laser-based transducer and automated analysis techniques", in *IEEE/ASME Trans. Mechatron.*, vol. 8, no. 3, pp. 401-409, Sep. 2003.

[24] O. Duran, K. Althoefer and L. D. Seneviratne, "Experiments using a laser-based transducer and automated analysis techniques for pipe inspection," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 2003, pp. 2561-2566 vol.2.

[25] Tsubouchi, T. Takaki, S. Kawaguchi, Y and Yuta, S., “A straight pipe observation from the inside by laser spot array and a TV camera," in *Proc. of IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 82-87, 2000.

[26] R. G. Dragan, I. C. Rosca, S. A. Keo and F. Breaban, “Active thermography method using an CO2 laser for thermal excitation, applied to defect detection in bioceramic materials," in *2013 E-Health and Bioengineering Conference (EHB)*, Iasi, 2013, pp. 1-4.

[27] Huiyi Zhang and J. Jackman, “A feasibility study of wind turbine blade surface crack detection using an optical inspection method," in *2013 International Conference on Renewable Energy Research and Applications (ICRERA)*, Madrid, 2013, pp. 847-852.

[28] Mumtaz, M., “A new approach to aircraft surface inspection based on directional energies of texture," in *Pattern Recognition*, Aug. 2010, pp. 4404-4407.

[29] Quan Shi, Ning Xi and Yifan Chen, “Development of an automatic optical measurement system for automotive part surface inspection," in *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.*, Monterey, CA, 2005, pp. 1557-1562.

[30] F. M. Megahed, J. A. Camelio, “Real-time fault detection in manufacturing environments using face recognition techniques," in *Journal of Intelligent manufacturing*, vol. 23, no. 3, pp. 393-408, June 2012.

[31] Yi-Gang Cen *et al.*, “Defect inspection for TFT-LCD images based on the low-rank matrix reconstruction,” *Neurocomputing*, vol. 149, pp. 1206-1215, 2015.

[32] X. Gibert, V. M. Patel and R. Chellappa, “Deep Multitask Learning for Railway Track Inspection," in *IEEE Trans. on Intell. Transp. Syst.*, vol. 18, no. 1, pp. 153-164, 2017.

[33] Y. Li, W. Zhao and J. Pan, “Deformable Patterned Fabric Defect Detection With Fisher Criterion-Based Deep Learning," in *IEEE Trans. on Automation Science and Engineering*, vol. 14, no. 2, pp. 1256-1264, 2017.

[34] A. Majumder, “Image processing algorithms for improved character recognition and components inspection," in *Proc. World Congr. on Nature and Biol. Inspired Comput.*, Coimbatore, India, 2009, pp. 531-536.

[35] X. Tao *et al.*, “A Novel and Effective Surface Flaw Inspection Instrument for Large-Aperture Optical Elements," *IEEE Trans. on Instrum. and Meas.*, vol. 64, no. 9, pp. 2530-2540, 2015.

[36] X. Yuan, L. Wu and Q. Peng, “An improved Otsu method using the weighted object variance for defect detection,” *Appl. Surf. Sci.*, vol. 349, pp. 472-484, 2015.

[37] Huiyi Zhang and J. Jackman, “A feasibility study of wind turbine blade surface crack detection using an optical inspection method," in *Proc. Int. Conf. on Renewable Energy Res. and Appl. (ICRERA)*, Madrid, Spain, 2013, pp. 847-852.

[38] E. Gupta and R. S. Kushwah, "Combination of global and local features using DWT with SVM for CBIR,"*4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*, Noida, India, 2015, pp. 1-6.

[39] J. Sun *et al.*, “Surface Defects Detection Based on Adaptive Multiscale Image Collection and Convolutional Neural Networks," *IEEE Trans. on Instrum. and Meas.*, vol. 68, no. 12, pp. 4787-4797, Dec. 2019.

[40] Y. Cha *et al.*, “Autonomous structural visual inspection using Region-Based deep learning for detecting multiple damage types,” *Comput.-Aided Civ. Infrastruct. Eng.*, vol. 33, pp. 731-747, 2018.

[41] D. Soukup and R. Huber-Mork, “Convolutional Neural Networks for Steel Surface Defect Detection from Photometric Stereo Images," in *Advances in visual computing*, pp. 668-677, 2014.

[42] R. Borwankar and R. Ludwig, “A Novel Compact Convolutional Neural Network for Real-Time Nondestructive Evaluation of Metallic Surfaces," *IEEE Trans. on Instrum. and Meas.*, vol. 69, no. 10, pp. 8466-8473, Oct. 2020

[43] R. Borwankar and R. Ludwig, “NDTNet: Optical Nondestructive Evaluation with Compact Convolutional Neural Network," in *Proc. IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1-6.

[44] V. Natarajan *et al.*, "Convolutional networks for voting-based anomaly classification in metal surface inspection," in *Proc. IEEE Int. Conf. on Ind. Technol. (ICIT)*, Toronto, Canada, 2017, pp. 986-991.

[45] T. Wang *et al.*, "A fast and robust convolutional neural network-based defect detection model in product quality control," *Int. J. Adv. Manuf. Technol.*, vol. 94, pp. 3465-3471, 2018.

[46] Y. Cha *et al.*, "Autonomous structural visual inspection using Region-Based deep learning for detecting multiple damage types," *Comput.-Aided Civ. Infrastruct. Eng.*, vol. 33, pp. 731-747, 2018.

[47] J. Chen *et al.*, "Automatic Defect Detection of Fasteners on the Catenary Support Device Using Deep Convolutional Neural Network," in *IEEE Trans. on Instrum. and Meas.*, vol. 67, no. 2, pp. 257-269, Feb. 2018.

[48] Xian Tao *et al.*, "Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks," *Appl. Sci.*, vol. 8, no. 9, pp. 1575-1590, 2018.

[49] J. Zhang, J. Li, "Improving the performance of OpenCL-based FPGA Accelerator for Convolution Neural Networks," in*Proceedings of the ACMISIGDA International Symposium on Field-Programmable Arrays -FPGA '17*, pages 25-34, 2017.

[50] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong, "Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster," in *Proceedings of the International Symposium on Low Power Electronics and Design -ISLPED '16*, pages 326-331, 2016.

[51] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proceedings of the International Conference on Computer-Aided Design - ICCAD '16*, pages 1-8, New York, New York, USA, 2016.

[52] Jeremy Bottleson, Sungye Kim, Jeff Andrews, Preeti Bindu, Deepak N. Murthy, and Jingyi Jin, "ClCaffe:Open CL accelerated caffe for convolutional neural networks," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium - IPDPS '16*, pages 50-57, 2016.

[53] Chi Zhang and Viktor Prasanna, "Frequency domain acceleration of Convolutional Neural Networks on CPU-FPGA shared Memory System," in *Proceedings of the ACM/SIGDA International Symposium on Field programmable Gate Arrays -FPGA '17*, pages 35-44, 2017.

[54] Jong Hwan Ko, Burhan Ahmad Mudassar, Taesik Na, and Saibal Mukhopadhyay, "Design of an Energy-efficient Accelerator for Training of Convolutional Neural Networks using Frequency-Domain Computation," in *Proceedings of the Annual Conference on Design Automation -DAC '17*, 2017.

[55] Andrew Lavin and Scott Gray, "Fast Algorithms for Convolutional Neural Networks," arXiv:150, 9 2015.

[56] Liqiang Lu, *et al*, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proceedings of the IEEE Annual International Symposium on Field-Programmable Custom Computing Machines -FCCM '17*, pages 101-108, 2017.

[57] R. DiCecco, *et al*, "Caffeinated FPGAs: FPGA Framework foe convolutional Neural Networks," in *proceedings of the International Conference on Field programmable Technology-FPT '16*, 2016.

[58] C Farabet, C Poulet, J Y Han, Y LeCun, David R. Tobergte, and Shirley Curtis, "CNP: An FPGA-based processor for Convolutional Networks," in *Proceedings of the International Conference on Field Programmable Logic and Applications - FPL '09*, volume 53, pages 1689-1699, 2009.

[59] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR '14*, pages 696-701, 6 2014.

[60] Jiantao Qiu, *et al*, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays -FPGA '16*, pages 26-35, New York, NY, USA, 2016.

[61] Chen Zhang, *et al*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*, FPGA, pages 161-170, 2015.

[62] Mohammad Motamedi, *et al*, "Design space exploration of FPGA-based Deep Convolutional Neural Networks," in *Proceedings of the Asia and South Pacific Design Automation Conference -ASPDAC'16*, pages 575-580, 1 2016.

[63] Rahman Atul, Lee Jongeun, and Choi Kiyoung, "Efficient FPGA acceleration of Convolutional Neural Networks using logical-3D compute array," in *Proceedings of the Conference on Design, Automation and Test in Europe -DATE '16*, Dresden, Germany, 2016.

[64] Chen Zhang, *et al*, "Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster," in *Proceedings of the International Symposium on Low Power Electronics and Design -ISLPED '16*, pages 326-331, 2016.

[65] Xuechao Wei, *et al*, "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs," in *Proceedings of the Annual Conference on Design Automation - DAC '17*, pages 1-6, New York, New York, USA, 2017.

[66] J. Dennis and D. Misunas, "A Preliminary Architecture for a Basic Data-flow Processor," in *Proceedings of the International Symposium on Computer Architecture -ISCA '75*, pages 126-132, 1975.

[67] Li Lin, *et al*, "Low power design methodology for signal processing systems using lightweight dataflow techniques," in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing - DASIP' 16*, pages 82-89, 2016.

[68] Chung-Ching Shen, *et al*, “A lightweight dataflow approach for design and implementation of SDR systems," in *Proceedings of the Wireless Innovation Conference and Product Exposition*, pages 640-645, 2010.

[69] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR, vol. abs/1512.03385*, 2015.

[70] Yu Cheng, *et al.*, “A Survey of Model Compression and Acceleration for Deep Neural Networks,” *IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding*, 2017.

[71] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, “Compressing deep convolutional networks using vector quantization,” *CoRR*, vol. abs/1412.6115, 2014.

[72] Z. Cai, X. He, J. Sun, and N. Vasconcelos, “Deep learning with low precision by half-wave gaussian quantization,” in *CVPR. IEEE Computer Society*, 2017, pp. 5406–5414

[73] S. J. Hanson and L. Y. Pratt, “Comparing biases for minimal network construction with back-propagation,” in *Advances in Neural Information Processing Systems*, 1989, pp. 177–185.

[74] S. Srinivas and R. V. Babu, “Data-free parameter pruning for deep neural networks,” in *Proceedings of the British Machine Vision Conference 2015*, BMVC 2015, Swansea, UK, September 7-10, 2015, 2015, pp. 31.1–31.12.

[75] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems, ser. NIPS’15*, 2015.

[76] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick.” *JMLR Workshop and Conference Proceedings*, 2015.

[77] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *International Conference on Learning Representations (ICLR)*, 2016.

[78] V. Lebedev and V. S. Lempitsky, "Fast convnets using group-wise brain damage," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, Las Vegas, NV, USA, June 27-30, 2016, 2016, pp. 2554–2564

[79] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *European Conference on Computer Vision*, Amsterdam, the Netherlands, October 2016, pp. 662–677.

[80] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 2074–2082.

[81] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, 2014.

[82] Y. W. Q. H. Jiaxiang Wu, Cong Leng and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[83] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[84] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ser. ICML'15*, 2015, pp. 1737–1746.

[85] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," *CoRR*, vol. abs/1612.01543, 2016.

[86] 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.

[87] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[88] M. Courbariaux, Y. Bengio, and J. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, December 7-12, 2015, Montreal, Quebec, Canada, pp. 3123–3131.

[89] S. Han, H. Mao and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” *https://arxiv.org/abs/1510.00149*, 2015.

[90] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, “Deep fried convnets,” in *International Conference on Computer Vision (ICCV)*, 2015.

[91] T. S. Cohen and M. Welling, “Group equivariant convolutional networks,” *arXiv preprint arXiv:1602.07576*, 2016.

[92] S. Zhai, Y. Cheng, and Z. M. Zhang, “Doubly convolutional neural networks,” in *Advances In Neural Information Processing Systems*, 2016, pp. 1082–1090.

[93] W. Shang, K. Sohn, D. Almeida, and H. Lee, “Understanding and improving convolutional neural networks via concatenated rectified linear units,” *arXiv preprint arXiv:1603.05201*, 2016.

[94] H. Li, W. Ouyang, and X. Wang, “Multi-bias non-linear activation in deep neural networks,” *arXiv preprint arXiv:1604.00676*, 2016.

[95] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, “Exploiting cyclic symmetry in convolutional neural networks,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ser. ICML'16*, 2016.

[96] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, “Speeding-up convolutional neural networks using fine-tuned cpdecomposition,” *CoRR, vol. abs/1412.6553*, 2014.

[97] C. Tai, T. Xiao, X. Wang, and W. E, “Convolutional neural networks with low-rank regularization,” *CoRR, vol. abs/1511.06067*, 2015.

[98] J. Ba and R. Caruana, “Do deep nets really need to be deep?” in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, December 8-13 2014, Montreal, Quebec, Canada, pp. 2654–2662.

[99] G. E. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *CoRR, vol. abs/1503.02531*, 2015.

[100] Yu He, *et al.*, “An End-to-end Steel Surface Defect Detection Approach via Fusing Multiple Hierarchical Features,” *IEEE Transactions on Instrumentation and Measurements*, vol. 69, no. 4, pp. 1493-1504, 2020.

[101] Pavlo Molchanov, *et al.*, “Pruning convolutional neural networks for resource efficient transfer learning,” *International Conference on Learning Representations (ICLR)*, Apr. 2017.

[102] B. Jacob, *et al.*, “Quantization and training of neural networks for efficient integer-arithmetic-only inference", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, 2018, pp. 2704-2713.

[103] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. on Learn. Representation (ICLR)*, San Diego, CA, 2015.

[104] K. He *et al.*, “Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. on Comput. Vision and Pattern Recognit. (CVPR)*, Las Vegas, NV, 2016, pp. 770-778.

[105] G. Huang *et al.*, “Densely Connected Convolutional Networks," in *Proc. IEEE Conf. on Comput. Vision and Pattern Recognit. (CVPR),* Honolulu, HI, 2017, pp. 2261-2269

[106] Andrew G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861v1 [cs.CV]*, Apr 2017. Available: http://arxiv.org/abs/1704.04861v1

[107] Mark Sandler *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Proc. IEEE Conf. on Comput. Vision and Pattern Recognit. (CVPR)*, Salt Lake City, UT, 2018, pp. 4510-4520.

[108] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. on Comput. Vision and Pattern Recognit.*, Miami, FL, 2009, pp. 248-255.

[109] *keras*. (2015). [online]. Available: https://keras.io

[110] Haifeng Jin, Qingquan Song, and Xia Hu, "Auto-Keras: An Efficient Neural Architecture Search System," in *Proc. 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '19)*, New York, NY, 2018, pp. 1946–1956.

[111] Y. Lyu, L. Bai and X. Huang, "ChipNet: Real-Time LiDAR Processing for Drivable Region Segmentation on an FPGA," *IEEE Trans. on Circuits and Syst. I: Regular Papers*, vol. 66, no. 5, pp. 1769-1779, May 2019.

[112] Xian Tao *et al.*, "Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks," *Appl. Sci.*, vol. 8, no. 9, pp. 1575-1590, 2018.

[113] J. Chen *et al.*., "RLBP: Robust local binary pattern," in *Proc. British Mach. Vision Conf. (BMVC)*, Bristol, UK, 2013.

[114] R. Borwankar and R. Ludwig, "A Novel Compact Convolutional Neural Network for Real-Time Nondestructive Evaluation of Metallic Surfaces," *IEEE Trans. on Instrum. and Meas.*, vol. 69, no. 10, pp. 8466-8473, Oct. 2020

[115] R. Borwankar and R. Ludwig, "NDTNet: Optical Nondestructive Evaluation with Compact Convolutional Neural Network," in *Proc. IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1-6.

[116] Q. Xiao, *et al.*, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2017, pp. 1-6.

[117] J. Shen, *et al.*, “Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays-FPGA*, 2018, pp. 97-106.

[118] J. Yepez and S. Ko, “Stride 2 1-D, 2-D, and 3-D Winograd for Convolutional Neural Networks," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 853-863, 2020.