INTELLIGENT SIGNAL PROCCESSING AND EMBEDDED AI SYSTEMS

Editor

Pavan Kumar Potuganti

INTELLIGENT SIGNAL PROCESSING AND EMBEDDED AI SYSTEMS- 2026

ISBN: 978-625-93102-8-2
DOI: 10.5281/zenodo.18305502

Edited By
Pavan Kumar Potuganti

January/ 2026
İstanbul, Türkiye

Date: 19.01.2026

Halic Publishing House
İstanbul, Türkiye
www.halicyayinevi.com

adopted by ESRA KOÇAK
ISBN: 978-625-93102-8-2

INTELLIGENT SIGNAL PROCESSING AND EMBEDDED AI SYSTEMS

EDITOR
Pavan Kumar Potuganti

AUTHORS
Dr. Suhas S. PATIL
A. KAMARAJ
S. Selva NIDHYANANTHAN
J. SENTHIL KUMAR
Monisha KARMAKAR
Pratim KUMA
Mrunmayee V. DAITHANKAR

TABLE OF CONTENTS

PREFACE

This book brings together innovative research that highlights the growing role of intelligent systems, embedded platforms, and advanced signal analysis in modern engineering applications. The chapters reflect a convergence of hardware, software, and data-driven methodologies aimed at improving perception, performance, and decision-making in complex technological environments.

The chapter Road Sign Recognition Using BNN in PYNQ-Z2 Board explores the implementation of efficient neural networks on embedded hardware for real-time visual recognition, demonstrating practical solutions for intelligent transportation systems. Complementing this, Preparation and Characterization Studies of KNO_3-HTPB Based Solid Rocket Propellant with Different Plasticizers presents an experimental investigation into materials and formulation techniques critical for propulsion performance and reliability.

The final chapter, Signal to Insight: AI-Driven Signal Processing, broadens the scope by examining how artificial intelligence transforms raw signals into meaningful insights across diverse domains. Together, these chapters offer readers a concise yet comprehensive perspective on the integration of intelligent computation, hardware innovation, and advanced engineering analysis, making the book a valuable resource for researchers, engineers, and students alike.

Editorial Team
January 19, 2026
Türkiye

CHAPTER 1
ROAD SIGN RECOGNITION USING BNN IN PYNQ-Z2 BOARD

A. KAMARAJ[1]
S. Selva NIDHYANANTHAN[2]
J. SENTHIL KUMAR[3]

[1]Department of ECE, Mepco Schlenk Engineering College, Sivakasi, kamarajvlsi@gmail.com, ORCID ID: 0000-0001-6952-2374.
[2]Department of ECE, Mepco Schlenk Engineering College, Sivakasi, nidhyan@mepcoeng.ac.in, ORCID ID: 0000-0001-9131-8409.
[3]Department of ECE, Mepco Schlenk Engineering College, Sivakasi, senthilkumarj@mepcoeng.ac.in, ORCID ID: 0000-0002-9516-0327.

INTRODUCTION

Recognition is a fundamental and challenging task in computer vision, with wide-ranging applications such as autonomous driving, crowd counting, and face recognition (Bhatt et al., 2022), (Fang et al., 2022). It involves both object classification and localization, typically achieved by identifying objects in images and drawing bounding boxes around them. Object recognition can be implemented using machine learning and deep learning techniques. Traditional machine learning approaches rely on handcrafted features such as color histograms, edges, or texture descriptors to identify groups of pixels corresponding to objects (Saha et al., 2012). In contrast, deep learning techniques automatically learn hierarchical feature representations directly from raw image data, leading to superior accuracy and robustness (Hadjam et al., 2022), (Zhao et al., 2022).

PYNQ (Python Productivity for Zynq) is an open-source framework that enables high-level programming of FPGA-based systems using Python (Mándi et al., 2021). The PYNQ-Z2 is an FPGA development platform based on the Zynq-7000 XC7Z020 SoC, designed specifically to support the PYNQ framework. It integrates programmable logic (PL) with a processing system (PS), allowing designers to develop, deploy, and test hardware-accelerated applications using Python through Jupyter Notebook environments. This approach significantly simplifies FPGA programming and accelerates rapid prototyping of embedded vision applications (Mándi et al., 2021).

Binarized Neural Networks (BNNs) are a class of neural networks in which both weights and activations are constrained to binary values (Rastegari et al., 2016), (Jaiswal et al., 2021). This binarization dramatically reduces memory usage, computational complexity, and power consumption, making BNNs particularly well suited for deployment on resource-constrained platforms such as FPGAs (Zhang et al., 2022), (Jokic et al., 2018). Despite their simplicity, BNNs can achieve competitive performance for real-time applications such as road sign recognition, where low latency and energy efficiency are critical (Liang et al., 2018), (Fiscaletti et al., 2020).

The proposed work involves dataset collection, preprocessing, training, and deployment of BNN models using road-sign and street-view datasets (Bhatt et al., 2022), (Saha et al., 2012).

Two BNN architectures—one based on fully connected layers and the other using convolutional layers—are developed and trained on the same dataset to enable a fair comparison. The trained models are synthesized and deployed on the PYNQ-Z2 FPGA using the Vivado Design Suite. Performance metrics such as accuracy, memory utilization, processing time, and hardware resource consumption are analyzed to evaluate the trade-offs between fully connected and convolutional BNN architectures (Zhang et al., 2022), (Yuan & Agaian, 2023). This comparative study provides practical insights into selecting appropriate BNN structures for FPGA-based embedded vision applications.

Neural Networks

Neural networks are a subset of machine learning inspired by the structure and functioning of the human brain (Qin et al., 2020), (Zhu et al., 2020). They consist of interconnected processing units called neurons, organized into layers, which collectively learn to map inputs to outputs through training. Neural networks are widely used in tasks such as image recognition, speech processing, and pattern classification, often outperforming traditional algorithms in complex problem domains (Zhao et al., 2022). Each neuron processes input signals using weighted connections and a bias term, followed by an activation function such as sigmoid or ReLU. During training, the network adjusts weights and biases using backpropagation to minimize prediction error. Neural networks can efficiently model complex nonlinear relationships, enabling rapid and accurate decision-making in large-scale data-driven applications (Zhu et al., 2020).

Types of Neural Networks

Several neural network architectures are commonly used, including Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Binarized Neural Networks (BNNs) (Qin et al., 2020), (Zhu et al., 2020). ANNs are general-purpose models composed of fully connected layers. CNNs are optimized for image processing tasks by exploiting spatial locality through convolutional operations (Bhatt et al., 2022), (Zhang et al., 2021). RNNs are designed for sequential and time-dependent data.

BNNs, which use binary weights and activations, offer significant advantages in terms of hardware efficiency and energy consumption (Rastegari et al., 2016), (Jaiswal et al., 2021) (Jokic et al., 2018).

Fully Connected Layers

A fully connected (dense) layer connects every input neuron to every output neuron, performing a linear transformation followed by an activation function. While fully connected layers are flexible and simple to implement, they require a large number of parameters, leading to higher memory usage and computational cost (Zhang et al., 2022), (Zhang et al., 2021).

Convolutional Layers

Convolutional layers employ sparse connectivity and shared weights, significantly reducing the number of parameters compared to fully connected layers (Bhatt et al., 2022), (Zhang et al., 2021). By applying convolution kernels across the input, CNNs efficiently capture spatial features such as edges, shapes, and textures.

1. LITERATURE SURVEY

Hardware-Optimized CNN and BNN Architectures on FPGA

One of the primary challenges in Advanced Driver Assistance Systems (ADAS) is the high memory footprint and computational complexity of Convolutional Neural Networks (CNNs). To address this issue, early efforts focused on reducing numerical precision. Integer-based and quantized CNN implementations have demonstrated significant reductions in computational overhead while maintaining high classification accuracy, highlighting the feasibility of low-cost hardware CNN accelerators for real-time applications (Bhatt et al., 2022).

Rastegari et al. introduced XNOR-Net, demonstrating that binary convolutional networks can achieve competitive accuracy while drastically reducing memory usage and arithmetic complexity (Rastegari et al., 2016). Building upon this concept, Zhang et al. proposed a time-domain FPGA-based BNN architecture that reduced storage requirements by approximately 75% by maintaining intermediate computations in 1-bit form (Zhang et al., 2022).

Practical FPGA-based implementations using BNNs have also been demonstrated on PYNQ platforms. Mándi et al. implemented a hardware-accelerated image processing pipeline on the PYNQ-Z2 board, achieving reduced power consumption and memory usage (Mándi et al., 2021). However, increased latency was reported under complex road conditions.

System-Level Implementations and Hybrid Models

System-level design choices play a crucial role in real-time TSR performance. CNN-based real-time TSR systems using hybrid datasets have demonstrated effective trade-offs between accuracy and speed, though frame-rate adaptation and data throughput remain bottlenecks (Bhatt et al., 2022).

Hybrid neural architectures have also been explored to reduce complexity. Saha et al. combined local image sampling with Artificial Neural Networks, achieving approximately 98% accuracy for a limited set of traffic signs, but with limited scalability (Saha et al., 2012), (Mándi et al., 2021), (Jokic et al., 2018).

Recent surveys and reviews of BNNs emphasize ongoing research in architecture search, robustness enhancement, and mixed-precision optimization to close the accuracy gap between binary and full-precision models (Qin et al., 2020), (Zhu et al., 2020), (Shen et al., 2020), (Phan et al., 2020). These studies confirm that BNNs are well suited for FPGA-based embedded vision systems when carefully designed.

Table 1. Comparison Analysis of Existing Works

Ref.	Authors / Year	Model Type	Hardware Platform	Key Contribution	Limitations
[1]	Fang et al., 2022	CNN	FPGA / GPU	High-accuracy deep CNN for image classification	High computational and memory cost
[3]	Saha et al., 2012	ANN	CPU	Early NN-based TSR with ~98% accuracy	Limited scalability, handcrafted features
[4]	Bhatt et al., 2022	CNN	Embedded system	Real-time TSR using hybrid datasets	Frame-rate and throughput bottlenecks

[6]	Mándi et al., 2021	Image Processing + BNN	PYNQ-Z2 FPGA	Low power and memory-efficient FPGA pipeline	Increased latency in complex scenes
[7]	Rastegari et al., 2016	BNN (XNOR-Net)	GPU / FPGA-ready	Binary weights & activations, major complexity reduction	Accuracy drop vs full-precision CNN
[9]	Zhang et al., 2022	BNN (Time-domain)	FPGA	~75% storage reduction, low power design	Higher hardware design complexity
[10]	Liang et al., 2018	BNN	FPGA	Optimized dataflow for binary inference	Limited flexibility for deep models
[11]	Jokic et al., 2018	BNN	FPGA camera system	Real-time (20 kfps) on-device recognition	Task-specific architecture
[14]	Yuan & Agaian, 2023	Survey (BNN)	—	Comprehensive BNN review & challenges	No hardware implementation
[15]	Qin et al., 2020	Survey (BNN)	—	Analysis of accuracy–efficiency trade-offs	Theoretical focus

Table 1 provides the reviews of FPGA-based CNN and BNN implementations for real-time traffic sign recognition, highlighting that BNNs significantly reduce memory, power, and computation while maintaining competitive accuracy, making them suitable for embedded vision systems when carefully architected.

Identified Research Gap

A systematic, hardware-level comparison between fully connected and convolutional BNN architectures on PYNQ-Z2—evaluating accuracy, resource utilization, and latency—remains insufficiently explored.

2. BINARY NEURAL NETWORK

The majority of network binarization techniques follow the BNN methodology developed by Courbariaux.

Binarization is used in BNNs for both the weights and activations. Using bitwise operations, this lowers the memory requirement for BNNs and the computational complexity. Except for the fact that everything is binarized to either +1 or -1, the architecture of BNN is similar to any normal DNN design. Here is a straightforward inference pseudo-code. (forward operation)

```
for k = 1 to L do
                    $W_k^{-b}$ ←Binarize ($W_k$)
                    $S_k \leftarrow a_{k-1}^{b} W_k^{-b}$
                    $a_k$ ← BatchNorm ($s_k.\emptyset_k$)
if k < L then
                    $a_k^b$← Binarize ($a_k$)
                      end if
                      end for
```

The aforementioned pseudo-code illustrates how a BNNs network works in the forward direction. L stands for the number layer, k for the layer index, ak for the activation after batchnorm, Sk for the activation prior to batchnorm, and Wk for the binarized weight.

2.1 Binarization of Weights

First, Courbariaux offers a technique for training with binary weights utilising backpropagation and a gradient descent algorithm. As contrast to just binarizing a network once training is complete, using binary data during training results in a more representative loss to train against (Zhang et al., 2022) (Jokic et al., 2018) (Yuan & Agaian, 2023), (Qin et al., 2020). It is not difficult to compute the gradient of the loss with respect to binary weights using back propagation. Binary weights, however, make gradient reasonable approaches for updating the weights unfeasible. Gradient descent methods allow for minor weight value adjustments, which are not possible with binary values.

Courbariaux maintains a set of real valued weights, WR, which are binarized within the network to produce binary weights, WB, in order to resolve this issue. Then, WR can be modified using backprop and gradient descent for incremental updates. The only weights recorded and used during inference are the binary weights because WR is not required. A straightforward sign function is used for binarization.

WB=sign (WR) ....(1)

Generates a tensor with the values +1 and -1. Because to the sign function employed in binarization, calculating the gradient of the loss with respect to the real valued directly weights have no practical application. At every point, the gradient of the sign function is zero or undefined. Courbariaux employs a technique known as the straight through estimator to circumvent this issue. By skipping over the gradient of the layer in question, this technique approximates a gradient. Just transform the troublesome gradient into an identity function.

$$\frac{\partial L}{\partial WR} = \frac{\partial L}{\partial WB} \qquad ....(2)$$

where L is the output loss. The weights with real values are updated using this gradient approximation. Sometimes, this binarization is considered to be a layer unto itself. The weights are sent via a binarization layer that, during the forward pass, determines the sign of the values, and, during the backward pass, executes an identity function. An illustration of the Straight-Through Estimator with sign layer (STE). The gradient of the binary weights is simply passed through to the real valued weights, while the sign function processes the real values of the weights in the forward pass.

The real valued weights can be adjusted using the STE and an optimization technique like SDG or Adam. If values in WR are not constrained, they can add up to very large amounts because the gradient updates can change the real valued weights WR without modifying the binary values WB. For instance, if a positive value of WR is assessed to have a positive gradient over a significant chunk of training, every update will raise that value. This may result in high WR values. Because of this, BNNs cut WR values between 1 and +1. As a result, WR and WB values remain nearby.

2.2 Binarization of Activations

In his initial BNN study, Courbariaux introduced the binarization of the activation values. Similar to how the weights are binarized, the activations are binarized by passing them through a sign function with a STE in the backwards pass. The network's activation function is this sign function. If the input to the activation was too large, Courbariaux discover that they need to use the backwards pass to cancel out the gradient in order to get decent results.

$$\frac{\partial L}{\partial aR} = \frac{\partial L}{\partial aB} * 1_{|aR|\leq 1} \qquad .... (3)$$

where aR is the activation function's real-valued input and aB is the function's binarized output. The indicator function 1|aR|1 returns 1 when |aR|1 and 0 when it does not. If the input to the activation function is too large, this zeroes out the gradient. It is possible to add a hard tanh function before the sign activation function to provide this capability, however this layer would only be effective in the backwards pass and not the forward pass.

$$X^b = \sin(X) = \begin{cases} -1, otherwise \\ 1, if\ x \geq 0 \end{cases} \qquad .... (4)$$

2.3 Bitwise Operations

The dot product between weights and activations can be broken down into bitwise operations when employing binary values. There are two possible binary values: -1 and +1 for the Figure 3.1. The encoding of these signed binary values uses a 0 for -1 and a 1 for +1. In order to be unambiguous, by refer to the signed numbers 1 and +1 as binary "values" and 0 and 1 as binary "encodings" for these numbers. As seen in Table 3.1, applying an XNOR logical operation to the binary encodings is equal to multiplying the binary values.

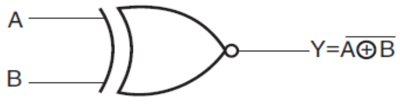


Figure 1. XNOR Gate

All the products between values must be added up to create a dot product. Bitwise multiplication can be accomplished with XNOR as shown in Figure 1; however, accumulating the results of the XNOR operation necessitates a summation. This can be done using the binary encodings produced by the XNOR operation by counting the number of 1 bit in a collection of XNOR products, multiplying this number by 2, and then taking away the number of bits that result in an integer value. To count the number of ones in a binary value, pop count instructions are frequently included in processor instruction sets as shown in Table 2.

Table 2. XNOR Operation's Counterpart

A	B	OUTPUT
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

A	B	OUTPUT
0	0	1
0	1	0
1	0	0
1	1	1

Comparatively to multi-bit floating-point or fixed-point multiplication and accumulation, these bitwise operations are substantially easier to compute. This may result in quicker execution times and/or a need for less hardware resources. It is not always easy to theories efficient speedups, though. For instance, some of the publications referred studied here use the number of instructions as a metric of execution time when examining the CPU's execution time. A CPU may perform a bitwise XNOR operation between two 64-bit registers thanks to the 64-bit x86 instruction set. One instruction is required for this procedure. Two 32-bit floating point multiplications could be achieved on a 64-bit CPU with a comparable architecture.

The bitwise operations would be 32 times faster than the floating-point operations, one could infer. However, the quantity of instructions does not reflect the speed of execution. The time it takes to complete each command can vary depending on the clock cycle. A modern CPU core's dynamic instruction and resource scheduling means that the number of cycles required to complete an instruction relies on the results of earlier instructions. Some kinds of instruction profiles are better suited to CPUs and GPUs than others. It is preferable to look at the actual execution times as a metric of efficiency rather than the total amount of instructions. While optimizing their code for bitwise operations, Courbariaux notice a 23 speedup. BNNs require less hardware in digital designs than bitwise operations, which also enable faster execution times in software-based implementations.

2.4 Batch Normalization

Deep learning commonly uses batch normalization (BN) layers. They serve as a sort of regularization and condition the values within a network for quicker training. They are viewed as crucial in BNNs.

BN layers contain gain and bias terms that the network learns as well as conditions for the values utilized during training. These acquired terms enable BNN become more complicated, without which it would suffer.

3. PROPOSED METHODOLOGY

Every input neuron and every output neuron are connected in a fully connected layer, also known as a linear layer, in commonly used neural networks. Bigger parameters typically enable improved efficiency and parallelization. Via the use of weights, the neuron makes a linear transformation to the input vector**.**

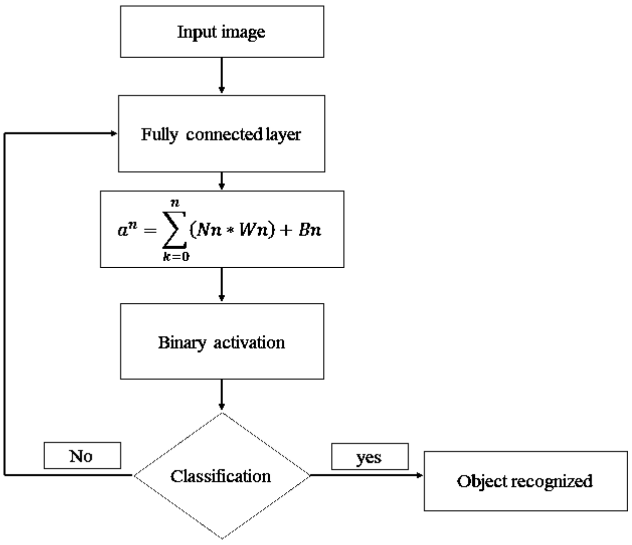


Figure 2. Fully Connected Layer Flow Chart

As shown in Figure 2, the input image is given the completely linked state from the above flowchart. In other words, the number of neurons employed in the network equals the number of pixels, and each node is given a different value. And after multiplying by the appropriate weights, each value is added together. The bias value is once more added to the additional value before being sent to the activation function, which determines whether the node is active or not. After classification is complete, a final check is made to see if the given image is recognized in the dataset or classes.

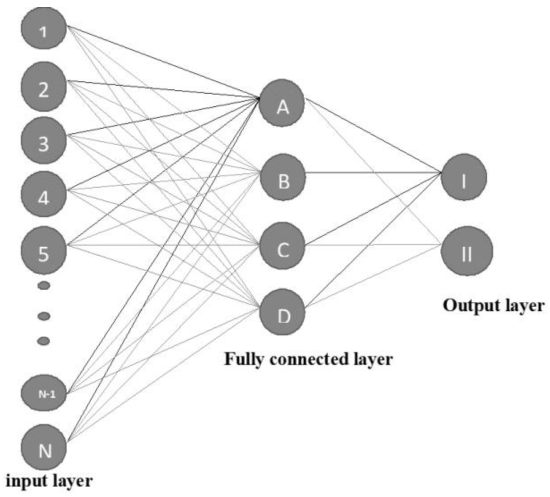


Figure 3. Fully Connected Layer

Layer to layer, all potential connections are present. Fully linked networks have the main benefit of being "structure agnostic," or not requiring any additional assumptions to be made about the input. As a result, these networks typically perform worse than special purpose networks that are customized to a problem's structure. Hence, there are many neurons as shown in Figure 3.

3.1 Using Convolutional Layer

In a neural network where not all input nodes are connected to output, there is a convolutional layer (Rastegari et al., 2016), (Jaiswal et al., 2021). Convolutional layers now have additional learning flexibility as a result. Moreover, there are a lot less weights per layer, which is beneficial for high-dimensional inputs like image data.

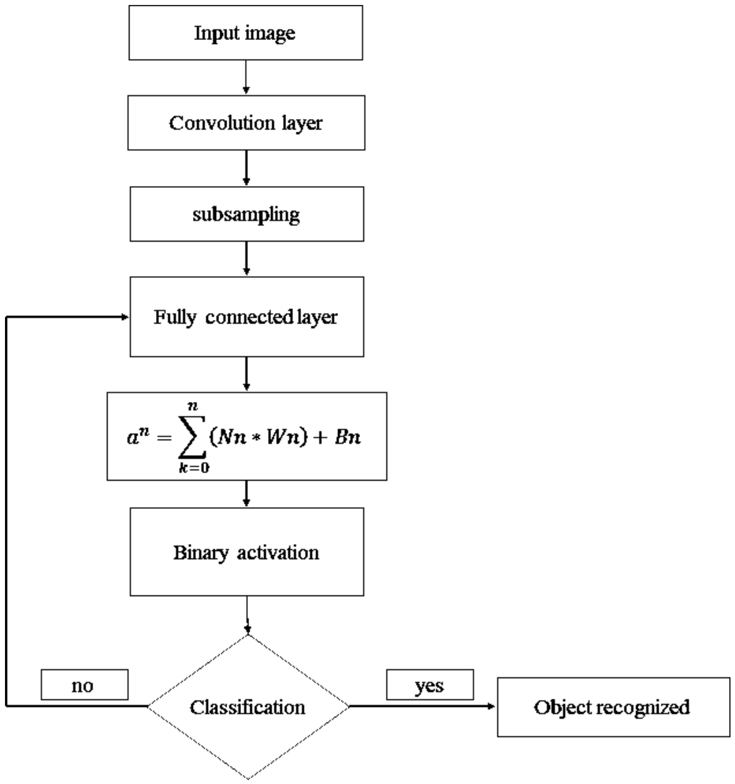


Figure 4. Convolutional Layer Flow Chart

As shown in Figure 4, the convolutional layer receives the input image from the previous computation; where the 3*3 kernel matrix is multiplied by the input image matrix to create images with distinct edges. To minimise the size of the image, pooling or sub-sampling is used. Thereafter, the number of pixels is decreased, and various values are assigned to each node. And after multiplying by the appropriate weights, each value is added together. The bias value is once more added to the additional value before being sent to the activation function, which determines whether the node is active or not. After classification is complete, a final check is made to see if the given image is recognised in the dataset or classes.

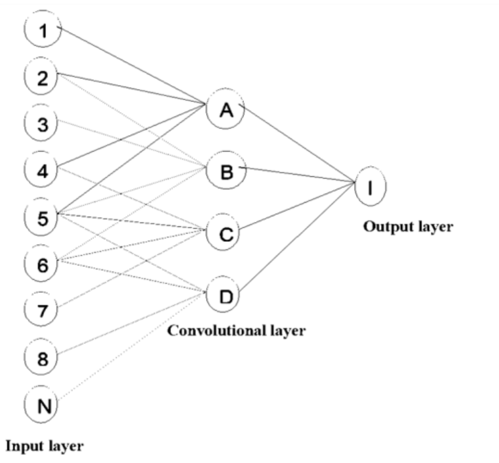


Figure 5. Convolutional Layer

With the kernel shifting along the input matrix and us taking the dot product between the two as though they were vectors, convolutional is essentially a sliding dot product. The model design can encode attributes since it explicitly assumes that the inputs are images. Every layer of a basic CNN is a sequence layer that translates activation volume from one layer to another using a differential function (Mándi et al., 2021), (Rastegari et al., 2016) as shown in Figure 5.

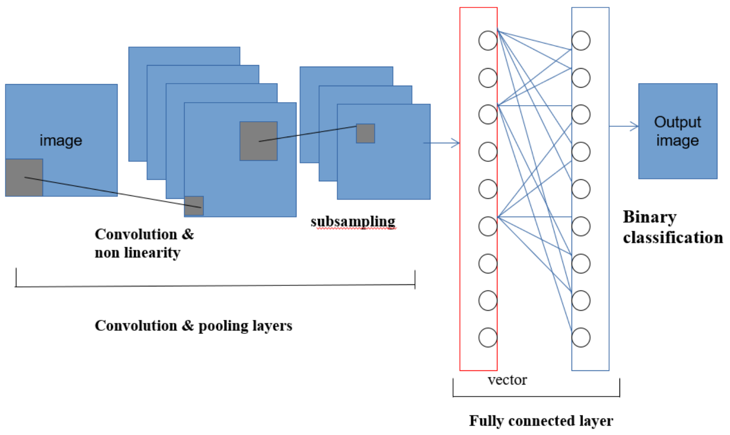


Figure 6. Convolutional Layer Using Subsampling

Sub-sampling shown in Figure 6 is a technique for reducing the amount of data by choosing only a portion of the original data. The process that identifies the highest values within each patch of a feature map and utilizes these values to generate down-sampled outputs. Typically, it comes after the convolutional layer. Sub sampling allows for faster processing and storage reduction.

3.2 Fully Connected Layer Vs Convolutional Layer

Compared to a fully linked layer, a convolutional layer is significantly more efficient and specialised. Each connection between neurons in a layer that is fully connected has its own weight since every neuron is linked to every other neuron in the layer above it. This connection design is entirely general-purpose and doesn't make any assumptions about the characteristics of the data. Also, the cost of memory (weights) and computation is relatively high (connections).

Contrarily, in a convolutional layer, every neuron has the same set of weights (and local connection structure) and is only connected to a small number of neighbouring (also known as local) neurons in the previous layer. This connection pattern only makes sense in situations when the data can be perceived as spatial, the features to be extracted are local in space (thus, only local connections are acceptable), and the likelihood of occurrence at any input position is equal (hence same weights at all positions OK). Convolutional layers are typically applied to image data where the features are local (e.g., a "nose" is made up of a group of neighbouring pixels rather than being dispersed throughout the entire image) and equally likely to occur anywhere.

The density of the connections is the primary distinction between the two types of layers. Every neuron in the output is coupled to every neuron in the input in the FC layers because of their high connectivity. The neurons in a convolutional layer, on the other hand, are only coupled to nearby neurons within the convolutional kernel's width and are not densely connected. Hence, a Conv layer is more appropriate when the input is a picture and there are many neurons. They also differ significantly in terms of how they share weight. Every output neuron in an FC layer is coupled to every input neuron by a unique weight(w).

The weights in a Conv layer, however, are distributed among various neurons. This is also another feature of Conv layers that makes them suitable for use when dealing with many neurons.

4. FUNCTIONAL MODULES

A data set is a collection of related, discrete items of related data that may be accessed individually or in combination or managed as a whole entity.

A data set is organized into some type of data structure. In a database, for example, a data set might contain a collection of business data (names, salaries, contact information, sales figures, and so forth). The database itself can be considered a data set, as can bodies of data within it related to a particular type of information, such as sales data for a particular corporate department.

The term data set originated with IBM, where its meaning was similar to that of file. In an IBM mainframe operating system, a data set s a named collection of data that contains individual data units organized (formatted) in a specific, IBM-prescribed way and accessed by a specific access method based on the data set organization. Types of data set organization include sequential, relative sequential, indexed sequential, and partitioned. Access methods include the Virtual Sequential Access Method (VSAM) and the Indexed Sequential Access Method (ISAM).

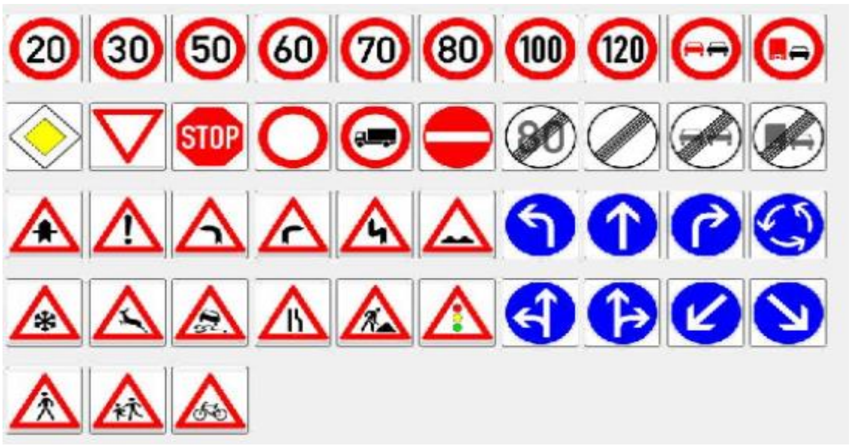


Figure 7. Sample Traffic Sign Dataset

A total of 50,000 images are used to test the detection phase of the models. Of these,43 traffic sign class from online sources with traffic sign having different viewing angle and position on image as shown in Figure 7. The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011.And cordially invite researchers from relevant fields to participate: The competition is designed to allow for participation without special domain knowledge. Our benchmark has the following properties:

- Single-image, multi-class classification problem
- More than 40 classes
- More than 50,000 images in total
- Large, lifelike database

4.1 Classifier Evaluation

The categorization displays how many layers were utilised to discover the picture and how much the image's size was decreased throughout the processing procedure. According to the table below, recognition is complete after the input picture, which has a resolution of 1024 pixels, has gone through five convolutional layers and been reduced to 256 pixels. The complete Classification of Convolutional Layer has been provided in Table 4.

Table 3. Set Of Images Given as Input

S. No	**Road sign**	**Name of the road sign**	**No. of images**	**S. No**	**Road sign**	**Name of the road sign**	**N.o of images**
1		20km/hr	10	18		End of all restrictions	10
2		30km/hr	10	19		End of no passi zone	10
3		50km/hr	10	20		Give away	10
4		60km/hr	10	21		Left turn	10
5		70km/hr	10	22		No entry	10

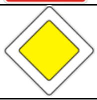
S. No	Road sign	Name of the road sign	No. of images	S. No	Road sign	Name of the road sign	N.o of images
6		80km/hr	10	23		No overtaking	10
7		100km/hr	10	24		No overtaking f large trucks	10
8		120km/hr	10	25		pass by left	10
9		Ahead only right	10	26		pass by right	10
10		Bicycle crossing	10	27		Pedestrian crossing	10
11		caution	10	28		Priority cross roads	10
12		Priority road	10	29		stop	10
13		Road about	10	30		Traffic signal ahead	10
14		Road work	10	31		Truck crossing	10
15		Slippery	10	32		Right turn	10
16		Snow	10	33		Watch for children	10
17		Speed breake	10	34		Wild animals crossing	10

Table 4. Classification of Convolutional Layer

Layers	**Size**
Input image resolution	32x32
Convolutional Layer 1	30x30
Convolutional Layer 2	28x28
Sub sampling / Maxpool 1	14x14
Convolutional Layer 3	12x12
Convolutional Layer 4	10x10

Sub sampling / Maxpool 2	5x5
Convolutional Layer 5	3x3
Fully Connected Layer 1	256
Fully Connected Layer 2	64
Fully Connected Layer 3	8

5. FPGA REALIZATION

By utilising a convolutional layer, one can may reduce the size of the picture and analyse it more quickly because employing a fully linked layer may require more data and processing. To decrease the size of the picture, there are three to five convolutional layers, and the BNNs technique is used to identify the road sign. Version 2019.1 of Vivado is the programme in use. (Jaiswal et al., 2021) The software is implemented using the PYNQ-Z2 board. The Xilinx Zynq-7000 SoC, which includes a dual-core ARM Cortex-A9 CPU and programmable logic that can be programmed using the Xilinx Vivado Design Suite, is installed on the PYNQ-Z2 board.

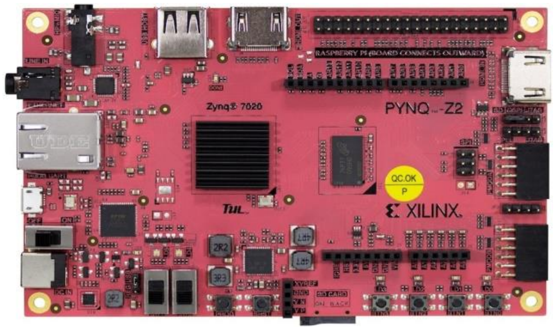


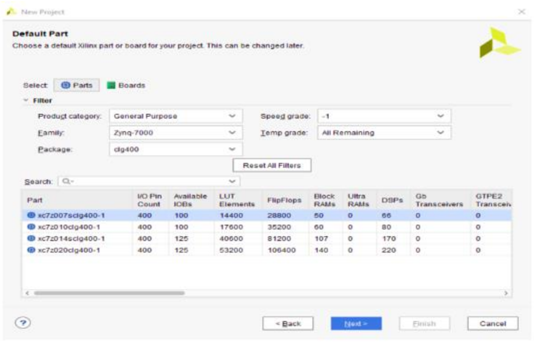
Figure 8. PYNQ-Z2 Board

5.1 Steps to Create and Implement a Project

Step 1: Create a Vivado Project

Vivado "projects" are directory structures that have every file a certain design requires. Several of these files are system files made by Vivado to control project design, simulation, and implementation. Others of these files are user-created source files that explain and limit the design. no need to worry about the user-created source files in a typical design.

But in the future, this may access to the other files as well if it require additional details about our design or more fine control over certain implementation aspects. Based on the sort of project need to develop, the "project type" configures certain design tools and the IDE look. They will often select "RTL Project" to set up the tools for creating a new design in all Real Digital courses. (RTL stands for Register Transfer Language, which is a phrase occasionally used to denote a hardware design language like Verilog).



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Product category: General Purpose

Family: Zynq-7000

Package: clg400

Speed grade: -1

Temp grade: All Remaining

Search: Q-

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	GTPE2 Transceiv
xc7z007sclg400-1	400	100	14400	28800	50	0	66	0	0
xc7z010clg400-1	400	100	17600	35200	60	0	80	0	0
xc7z014sclg400-1	400	125	40600	81200	107	0	170	0	0
xc7z020clg400-1	400	125	53200	106400	140	0	220	0	0

Figure 9. ZYNQ 7000 part

There are several components made by Xilinx, and the synthesizer needs to know precisely which one is using in order to create the appropriate programming file. The device family, packaging, and speed and temperature grades—which solely impact special-purpose simulation results and have no bearing on the synthesizer's capacity to build accurate circuits—must be known in order to identify the right item. It is necessary and check select the proper component for the equipment mounted on PYNQ Z2 BOARD as shown in Figure 9 and Table 5.

Table 5. PYNQ-Z2 Package in Vivado

Part number	Xc7z007sclg400-1
Family	Zynq-7000
Package	Clg400
Speed Grade	-1
Temperature Grade	C

Step 2: Edit the Project - Create source files

A constraints file that gives the synthesizer the details it needs to map the circuit into the target chip and an HDL file (Verilog or VHDL) that describes the circuit are both required for every project. It is possible to immediately replicate the Verilog source file after it has been prepared. Before spending the effort to create a circuit in a real device, and test its functionality using simulation (explained in greater depth later). The simulator enables allows to test that the outputs respond as anticipated in all circumstances by driving all of the circuit's inputs with a variety of patterns over time.

Step 3: Synthesize, Implement, and Generate Bitstream

By Synthesizing the design project after execution of Verilog and constraint files are finished. Verilog code is converted into a "netlist," which specifies all the necessary circuit components, during the synthesis process (these components are the programmable parts of the targeted logic device - more on that later). By selecting the Run Synthesis button in the Flow Navigator window as illustrated, this may begin the Synthesize procedure. When synthesis is active, can able to access the Project Manager log panel at the bottom to view a log of the processes that are now operating. The log will include a description of any synthesis-related mistakes that take place. When the design has been synthesized, the Implementation phase must be conducted. The synthesized design is mapped onto the Xilinx chip that it is intended for during the implementation phase. On the Flow Navigator window, click the Run Implementation button as displayed. The log panel at the bottom of Project Manager will provide information about any mistakes that happen while the implementation process is underway.

By selecting the Create Bitstream procedure in the Flow Navigator panel as shown, this may produce a bit file after the design has been successfully executed. The method converts the implemented design into a bitstream that can be directly programmable into the hardware on PYNQ Z2 BOARD.

5.2 Board connection and Download Bitstream

Use a micro-USB cord to connect Blackboard to your computer. Be careful to attach the micro-USB cable to the "PROG UART" port. By turning the switch in the top-left corner to the on position, now turn on the board. When it turns on, a red LED will start to glow near the switch. Make sure the blue jumper near the port marked "EXTP" is set to "USB" if the board won't turn on. The image depicts a Blackboard that is powered on and has the proper jumper settings.

The Hardware panel, which is found in the upper left corner of Hardware Manager, will display the board's logic device component number if Vivado successfully recognizes the board (For the Blackboard this will be xc7z007s). Right-click on the device to be programmed, then choose Program Device. The produced bit file will be chosen in the text box when a Program Device pop-up dialogue window appears. To download the bitstream to the board, choose Program.

5.3 Design Implemented and Block Generated

To test the picture, downloaded the "GTSRB" dataset. (Zhang et al., 2022), (Jokic et al., 2018) (Liang et al., 2018), (Fiscaletti et al., 2020) Following the creation of the HDL wrapper in Vivado, the processing system developed an IP and picked the ZYNQ7 chip since the PYNQ-Z2 board supports the ZYNQ 7000 series. AXI connection has been employed, which employs several slave and master nodes before using a reset system to restart the procedure. The output is then sent to the processing system once the IP block has been linked to two AXI interconnect blocks.

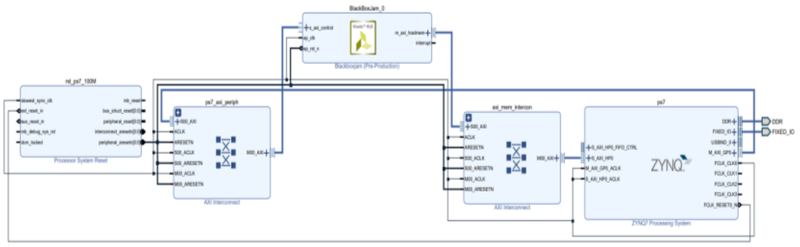


Figure 10. Generated Block Diagram

Synthesis and implementation follow the generation of the block as generated in Figure 10. The implementation results demonstrate how Vivado uses time, power, and register. The bit files are then copied to the board after the bit stream has been created. On our computer, intercommon port 5 is used to connect the PYNQ-Z2. As the board boots up, a default IP may be configured in the settings, and the Jupyter page will appear instantly. From there, in order to view results in real time application, must upload our files. Vivado generates a report for two ways, and a comparison is made.

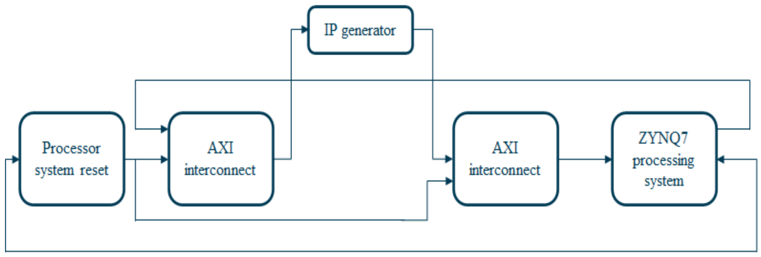


Figure 11. Simple Block Diagram

The ZYNQ7 Processing System and its peripherals, such as the AXI Interconnect and the IP Generator, which oversee facilitating communication between the processing system and the programmable logic of the PYNQ Z2 board, are reset using the processor system reset. Although the IP Generator oversees gathering data on the accelerator's performance, the AXI Connector handles communication between the processing systems. The simplified version of the generated block is shown in Figure 11.

6. RESULTS AND DISCUSSIONS

Binarized neural network (BNN) is designed and synthesised utilising a completely linked and convolutional layer. Implementation is carried out on the Vivado platform using the ZYNQ 70000 series PYNQ-Z2 board.

6.1 Synthesis and Implementation Results

Timing Summary

Worst Negative Slack (WNS): For maximum delay analysis, this number represents the worst slack of all time pathways.

It could be favorable or unfavorable. When considering only the worst violation of each timing route endpoint, Total Negative Slack (TNS) is the total of all WNS violations. When all timing requirements are satisfied for the max delay analysis, its value is 0 ns. The delay will be greatest if the worst negative slack value turns negative the total number of endpoints that failed (WNS< 0ns) is referred to as the number of failing endpoints. The total number of endpoints that have been examined.

Worst Hold Slack (WHS): Refers to the timing routes' worst slack for the min delay analysis. It could be favorable or unfavorable. When just considering the worst violation of each timing route endpoint, Total Hold Slack (THS) is the total of all WHS violations. When all timing requirements are satisfied for the minimum delay analysis, its value is 0 ns. The delay will be greatest if the worst negative slack value turns negative. The total number of endpoints that failed (WHS<0 ns) is known as the number of failing endpoints. The total number of endpoints that have been examined.

Worst Pulse Width Slack (WPWS): When utilizing both the min and max delays, corresponds to the worst slack of all the timing tests stated above. By just considering the worst violation of each pin in the design, Total Pulse Width Slack (TPWS) is the total of all WPWS infractions. When all pertinent restrictions are satisfied, its value is set to 0. The delay will be greatest if the worst negative slack value turns negative. The total number of pins with a violation (WPWS< 0 ns) is referred to as the number of failing endpoints. The total number of endpoints that have been examined.

Timing Summary of Fully Connected Layer

Worst Negative Slack (WNS) - 0.101ns
Total Negative Slack (TNS) Total Endpoints -131755
Worst Hold Slack (WHS) -0.009ns
Total Hold Slack (THS) Total Endpoints -131755
Worst Pulse Width Slack (WPWS) -3.750ns
Total Pulse Width Slack (TPWS) Total Endpoints -49520

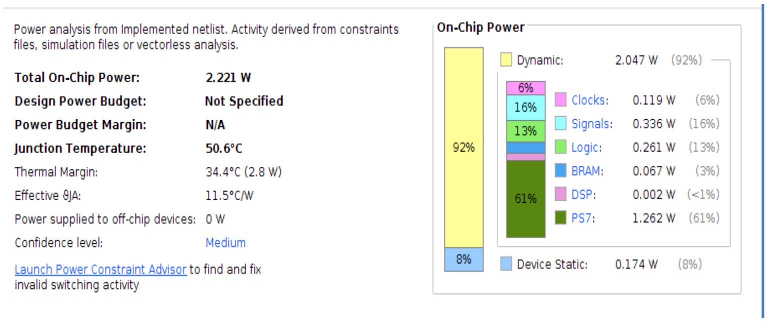
Timing Summary of Convolutional Layer

Worst Negative Slack (WNS) - 0.166ns

Total Negative Slack (TNS) Total Endpoints -101307
Worst Hold Slack (WHS) -0.019ns
Total Hold Slack (THS) Total Endpoints -101307
Worst Pulse Width Slack (WPWS) -3.750ns
Total Pulse Width Slack (TPWS) Total Endpoints -41013

Power Analysis

The post-synthesis, post-placement, and post-routing stages of the flow are all powered estimated using the Vivado ® power analysis tool. Since it can read the precise logic and routing resources from the implemented design, post-route is when it is most accurate. The Summary power report and the many perspectives of your design that you may explore—by clock domain, by resource type, and by design hierarchy—are shown in the accompanying image. You can modify environment settings and design activities in the Vivado Integrated Design Environment (IDE) to assess how to lower your design supply and thermal power usage. In order to evaluate and identify the design's high power-consuming hierarchy and resources, you may also cross-probe into the design from the power report.



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**2.221 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**50.6°C**
Thermal Margin:	34.4°C (2.8 W)
Effective θJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

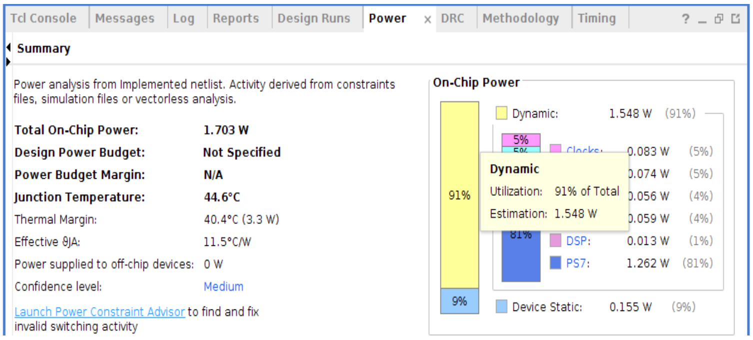
Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	2.047 W	(92%)
Clocks:	0.119 W	(6%)
Signals:	0.336 W	(16%)
Logic:	0.261 W	(13%)
BRAM:	0.067 W	(3%)
DSP:	0.002 W	(<1%)
PS7:	1.262 W	(61%)
Device Static:	0.174 W	(8%)

Figure 12. Power Analysis Using Fully Connected Layer

From Figure 12, the power analysis for a completely linked layer is successfully obtained from the given graphic. 2.21W of the chip's total power is utilized by clocks, signals, logic, Memory, DSP, and PS7. The junction temperature is 50.6°C, while the thermal margin temperature is 34.4°C, or 2.8W.



Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.703 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**44.6°C**
Thermal Margin:	40.4°C (3.3 W)
Effective θJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.548 W	(91%)
Clocks:	0.083 W	(5%)
	0.074 W	(5%)
	0.056 W	(4%)
	0.059 W	(4%)
DSP:	0.013 W	(1%)
PS7:	1.262 W	(81%)
Device Static:	0.155 W	(9%)

Dynamic
Utilization: 91% of Total
Estimation: 1.548 W

Figure 13. Power Analysis Using Convolutional Layer

From Figure 13, the power analysis for a completely linked layer is successfully obtained from the given graphic. 1.703W of the chip's total power is utilized by clocks, signals, logic, Memory, DSP, and PS7. The junction temperature is 44.4°C, while the thermal margin temperature is 40.4°C, or 3.3W.

Synthesis Results

The functionally verified HDL codes are implemented in FPGA platform for prototype hardware generation. The synthesis results Fully connected layer and convolutional layer are shown in Table 6 and Table 7.

Table 6. Implementation Result of Fully Connected Layer

Device Utilization Summary			
Site Type	**Used**	**Available**	**Util%**
Slice LUTs	40946	53200	76.97
LUT as Logic	36190	53200	68.21
LUT as Memory	4791	17400	27.53
LUT as Distributed RAM	4362		
LUT as Shift Register	429		
Slice Registers	45338	106400	42.61
Register as Flip Flop	45338	106400	42.61
F7 Muxes	903	26600	3.39
F8 Muxes	128	13300	0.96

To examined the outcomes and power use on the device utilizing fully linked layer BNNs.

In addition to using the F7 mux, which may multiply some inputs, the slice register and block ram memory are also utilized. The utilization statistics and design summary for the completely linked layer are shown in the implementation results above; the worst time is 0.101ns, and the power use is around 2.221W.

In order to examined the outcomes and power use on the device utilizing convolutional layer BNNs. In addition to using the F7 mux, which may multiply some inputs, the slice register and block ram memory are also utilized. The utilization statistics and design summary for the completely linked layer are shown in the implementation results below; the worst time is 0.166ns, and the power use is around 1.703W.

Table 7. Implementation Result of Convolutional Layer

Device Utilization Summary			
Site Type	**Used**	**Available**	**Util%**
Slice LUTs	24395	53200	45.86
LUT as Logic	22472	53200	42.24
LUT as Memory	1923	17400	11.05
LUT as Distributed RAM	1578		
LUT as Shift Register	345		
Slice Registers	38506	106400	36.19
Register as Flip Flop	38506	106400	36.19
F7 Muxes	857	26600	3.22
F8 Muxes	240	13300	1.80

The resource utilization for the BNN algorithm using the Fully connected layer and convolutional layer is listed in a Table 8.

Table 8. Comparison of Fully Connected and Convolutional Layer

Resources	**FCL**	**CNL**	**% of Reduction**
Slice LUTs	40946	24395	40.4
LUT as Logic	36190	22472	37.9
LUT as Memory	4791	1923	59.8
LUT as Distributed RAM	4362	1578	63.8
LUT as Shift Register	429	345	19.5
Slice Registers	45338	38506	15
F7 Muxes	903	857	5
POWER	2.221W	1.703W	23.3

The resource utilization for the BNN algorithm using the Fully connected layer and convolutional layer is compared using a bar graph in the Figure 14 and found that the resource is utilized minimally in Convolutional layer than the Fully connected layer which uses the resources comparatively higher. The usage of slice LUTs and LUT flip flop pairs are 31% comparatively lower in Convolutional layer than in Fully connected layer which utilizes the area efficiently. The timing constraints and area utilization ensures the efficiency of Convolutional layer.

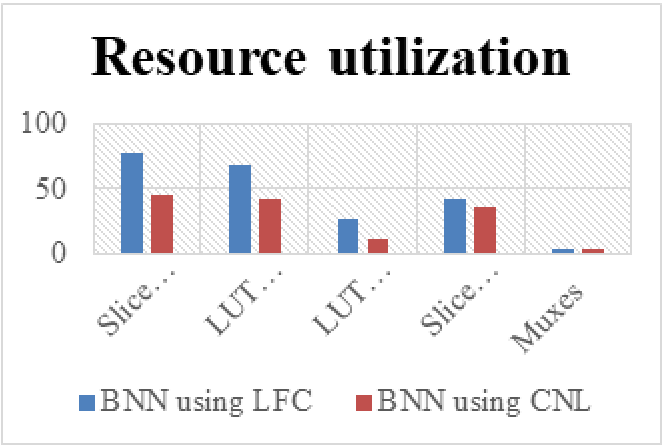
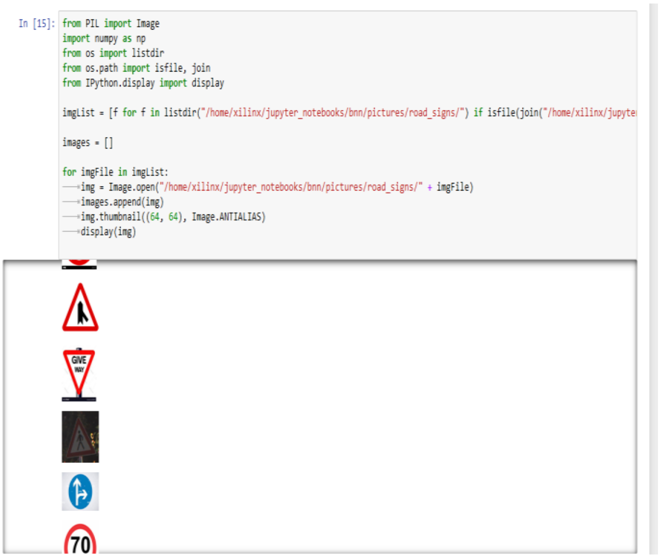


Figure 14. Comparison Of Two Layers

Jupyter Output for Traffic Sign Recognition

Using https://192.168.2.99 as the default IP, connect the PYNZQ-Z2 board to the Jupyter notebook. The Jupyter notebook requires the username and password to be entered after the connection. Both the username and password are Xilinx. After that, you must upload the zip file or folder containing the dataset and the codes. The PYNQ-Z2 board must have the BNN package installed in order to carry out the operation. First determine how many datasets there are and how many classes are included in the supplied dataset. So, the GTSTB dataset, which includes 50,000 photos and the 42 classes stated in the previous chapter, for determining the inference time and classification rate using the 340 input photos provided as shown in Figure 15. The sign's name and the class to which the image belongs will be included in the output. Also, the program's operating speed varies across hardware and software.



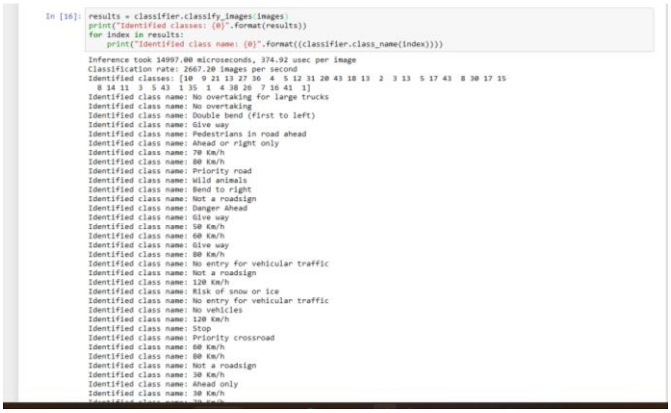
```
In [15]: from PIL import Image
         import numpy as np
         from os import listdir
         from os.path import isfile, join
         from IPython.display import display

         imgList = [f for f in listdir("/home/xilinx/jupyter_notebooks/bnn/pictures/road_signs/") if isfile(join("/home/xilinx/jupyte

         images = []

         for imgFile in imgList:
             img = Image.open("/home/xilinx/jupyter_notebooks/bnn/pictures/road_signs/" + imgFile)
             images.append(img)
             img.thumbnail((64, 64), Image.ANTIALIAS)
             display(img)
```

Figure 15. Input Images Uploaded



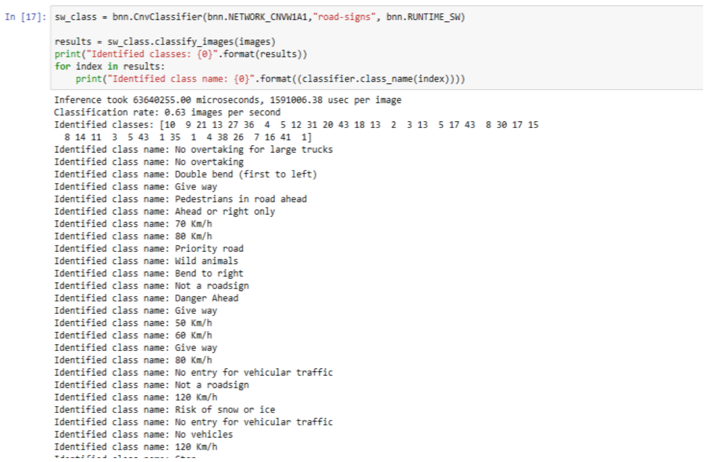
```
In [16]: results = classifier.classify_images(images)
         print("Identified classes: {0}".format(results))
         for index in results:
             print("Identified class name: {0}".format((classifier.class_name(index))))

         Inference took 14997.00 microseconds, 374.92 usec per image
         Classification rate: 2667.20 images per second
         Identified classes: [10  9 21 13 27 36  4  5 12 31 20 43 18 13  2  3 13  5 17 43  8 30 17 15
           8 14 11  3  5 43  1 35  1  4 38 26  7 16 41  1]
         Identified class name: No overtaking for large trucks
         Identified class name: No overtaking
         Identified class name: Double bend (first to left)
         Identified class name: Give way
         Identified class name: Pedestrians in road ahead
         Identified class name: Ahead or right only
         Identified class name: 70 Km/h
         Identified class name: 80 Km/h
         Identified class name: Priority road
         Identified class name: Wild animals
         Identified class name: Bend to right
         Identified class name: Not a roadsign
         Identified class name: Danger Ahead
         Identified class name: Give way
         Identified class name: 50 Km/h
         Identified class name: 60 Km/h
         Identified class name: Give way
         Identified class name: 80 Km/h
         Identified class name: No entry for vehicular traffic
         Identified class name: Not a roadsign
         Identified class name: 120 Km/h
         Identified class name: Risk of snow or ice
         Identified class name: No entry for vehicular traffic
         Identified class name: No vehicles
         Identified class name: 120 Km/h
         Identified class name: Stop
         Identified class name: Priority crossroad
         Identified class name: 60 Km/h
         Identified class name: 80 Km/h
         Identified class name: Not a roadsign
         Identified class name: 30 Km/h
         Identified class name: Ahead only
         Identified class name: 30 Km/h
```

Figure 16. Classification in Hardware

The provided Figure 16 illustrates the inference results and performance benchmarks of a convolutional neural network (CNN) trained for traffic sign classification, likely utilizing the German Traffic Sign Recognition Benchmark (GTSRB) dataset. The execution block demonstrates a batch processing approach where the classify_images function predicts class indices for a series of input images, which are subsequently mapped to human-readable labels such as "Priority Road," "Give way," and various speed limit indicators.

High-performance efficiency is a key highlight of this output, with the model achieving a classification rate of 2,667.20 images per second and a mean inference time of approximately 374.92 microseconds per image. Furthermore, the inclusion of a "Not a roadsign" classification category suggests the implementation of a robust filtering mechanism to handle non-relevant background data, which is critical for the reliability of real-time autonomous driving systems.



```
In [17]: sw_class = bnn.CnvClassifier(bnn.NETWORK_CNVW1A1,"road-signs", bnn.RUNTIME_SW)

         results = sw_class.classify_images(images)
         print("Identified classes: {0}".format(results))
         for index in results:
             print("Identified class name: {0}".format((classifier.class_name(index))))

         Inference took 63640255.00 microseconds, 1591006.38 usec per image
         Classification rate: 0.63 images per second
         Identified classes: [10  9 21 13 27 36  4  5 12 31 20 43 18 13  2  3 13  5 17 43  8 30 17 15
           8 14 11  3  5 43  1 35  1  4 38 26  7 16 41  1]
         Identified class name: No overtaking for large trucks
         Identified class name: No overtaking
         Identified class name: Double bend (first to left)
         Identified class name: Give way
         Identified class name: Pedestrians in road ahead
         Identified class name: Ahead or right only
         Identified class name: 70 Km/h
         Identified class name: 80 Km/h
         Identified class name: Priority road
         Identified class name: Wild animals
         Identified class name: Bend to right
         Identified class name: Not a roadsign
         Identified class name: Danger Ahead
         Identified class name: Give way
         Identified class name: 50 Km/h
         Identified class name: 60 Km/h
         Identified class name: Give way
         Identified class name: 80 Km/h
         Identified class name: No entry for vehicular traffic
         Identified class name: Not a roadsign
         Identified class name: 120 Km/h
         Identified class name: Risk of snow or ice
         Identified class name: No entry for vehicular traffic
         Identified class name: No vehicles
         Identified class name: 120 Km/h
```

Figure 17. Classification in Software

The following figure illustrates the performance characteristics of a traffic sign classification model specifically operating under a software-only runtime environment (bnn. RUNTIME_SW). While the model maintains consistent classification accuracy—correctly identifying a diverse array of regulatory and warning signs such as "No overtaking," "Give way," and various speed limits—the computational overhead of the software-based inference is substantial. The system recorded an inference time of approximately 63,640,255.00 microseconds, translating to a significantly reduced throughput of 0.63 images per second and a latency of 1,591,006.38 microseconds per image. This performance metric highlights the inherent limitations of standard software execution for complex neural network operations in real-time scenarios, serving as a critical baseline for evaluating the acceleration provided by dedicated hardware runtimes or FPGA-based implementations.

Table 9. Comparison between Software and Hardware BNN

Metric	**Software Runtime (SW)**	**Hardware Runtime (BNN)**	**Improvement**
Throughput	0.63 FPS	2667.20 FPS	~4,233x
Inference Latency	1,591,006.38 µs	374.92 µs	>99.9% reduction
Resource Efficiency	N/A	31% Reduction in LUTs	Optimized for FPGA
Power Consumption	N/A	1.703W	Energy Efficient

The performance comparison between the software runtime (SW) and hardware runtime (BNN) highlights significant improvements in both speed and resource efficiency as shown in Table 9. Throughput increases by approximately 4,233 times, with the hardware achieving 2,667.20 FPS compared to the software's 0.63 FPS. Inference latency is reduced by over 99.9%, from 1,591,006.38 µs in software to just 374.92 µs in hardware. Additionally, resource usage is optimized for FPGA, with a 31% reduction in LUTs. Power consumption is notably lower in the hardware implementation, with the system consuming just 1.703W, demonstrating its energy efficiency.

CONCLUSION

This method can be effectively applied to a variety of real-time applications, offering significant performance benefits. The approach demonstrates impressive efficiency, particularly in the classification of traffic signs, with the system capable of classifying them in under 0.5 seconds. A generalized recognition system for road signs was developed using a 3x3 kernel and a 64x64 feature matrix. A comparison of resource usage between the BNN algorithm's fully connected layer and convolutional layer revealed that the convolutional layer consumes significantly fewer resources. Specifically, the convolutional layer uses 31% less slice LUTs and LUT flip-flop pairs, effectively optimizing space usage. The convolutional layer's efficiency is further validated by its ability to meet strict time constraints and minimize area usage. Despite being less densely connected, with some input nodes not affecting all output nodes, convolutional layers outperform fully connected layers in terms of resource consumption and learning flexibility.

Implementation results in Vivado show that the fully connected layer has higher resource utilization, but the convolutional layer exhibits a lower power consumption of 1.703W. Additionally, the road sign recognition process is completed in just 3527 microseconds, with 1417.64 images classified per second. In conclusion, this approach proves to be both resource-efficient and effective for real-time applications, offering significant advantages in speed, power consumption, and overall performance.

REFERENCES

Bhatt, N., Laldas, P., & Lobo, V. B. (2022). A real-time traffic sign detection and recognition system on hybrid dataset using CNN. *Proceedings of the 7th International Conference on Communication and Electronics Systems (ICCES)*, 1354–1358. https://doi.org/10.1109/ICCES54183.2022.9835884

Fang, M., Lei, X., Liao, B., & Wu, F. X. (2022). A deep neural network for cervical cell classification based on cytology images. *IEEE Access, 10*, 130968–130980. https://doi.org/10.1109/ACCESS.2022.3227795

Fiscaletti, G., Speziali, M., Stornaiuolo, L., Santambrogio, M. D., & Sciuto, D. (2020). BNNsplit: Binarized neural networks for embedded distributed FPGA-based computing systems. *Design, Automation & Test in Europe Conference (DATE)*, 975–978. https://doi.org/10.23919/DATE48585.2020.9116482

Hadjam, T., Salah, A. M., Nedjma, M. B., Abdelmadjid, M., & Hamil, H. (2022). FPGA implementation of a convolutional neural network for image classification. *Proceedings of the 2nd International Conference on Advanced Electrical Engineering (ICAEE)*, 1–5. https://doi.org/10.1109/ICAEE54426.2022.9933827

Jaiswal, M., Sharma, V., Sharma, A., Saini, S., & Tomar, R. (2021). FPGA based implementation of binarized neural network for sign language application. *IEEE International Symposium on Smart Electronic Systems (iSES)*, 303–306. https://doi.org/10.1109/iSES52644.2021.00075

Jokic, P., Emery, S., & Benini, L. (2018). BinaryEye: A 20 kfps streaming camera system on FPGA with real-time on-device image recognition using binary neural networks. *IEEE International Symposium on Industrial Embedded Systems (SIES)*, 1–7. https://doi.org/10.1109/SIES.2018.8442096

Liang, S., Yin, S., Liu, L., Luk, W., & Wei, S. (2018). FP-BNN: Binarized neural network on FPGA. *Neurocomputing, 275*, 1072–1086. https://doi.org/10.1016/j.neucom.2017.09.087

Mándi, Á., Máté, J., Rózsa, D., & Oniga, S. (2021). Hardware accelerated image processing on FPGA based PYNQ-Z2 board. *Carpathian Journal of Electronic and Computer Engineering, 14*(1), 20–23.

Phan, H., Liu, Z., Huynh, D., Savvides, M., Cheng, K. T., & Shen, Z. (2020). Binarizing MobileNet via evolution-based searching. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13420–13429. https://doi.org/10.1109/CVPR42600.2020.01344

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., & Sebe, N. (2020). Binary neural networks: A survey. *Pattern Recognition, 105*, 107281. https://doi.org/10.1016/j.patcog.2020.107281

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-Net: ImageNet classification using binary convolutional neural networks. *European Conference on Computer Vision (ECCV 2016)*, 525–542. https://doi.org/10.1007/978-3-319-46493-0_32

Saha, S. K., Chakraborty, D., & Bhuiyan, M. A. A. (2012). Neural network-based road sign recognition. *International Journal of Computer Applications, 50*(10), 1–5. https://doi.org/10.5120/7780-1087

Shen, M., Liu, X., Gong, R., & Han, K. (2020). Balanced binary neural networks with gated residual. *ICASSP 2020 – IEEE International Conference on Acoustics, Speech and Signal Processing*, 4197–4201. https://doi.org/10.1109/ICASSP40776.2020.9054438

Yuan, C., & Agaian, S. S. (2023). A comprehensive review of binary neural networks. *Artificial Intelligence Review*, 1–65. https://doi.org/10.1007/s10462-023-10461-9

Zhang, L., Tang, X., Hu, X., Zhou, T., & Peng, Y. (2022). FPGA-based BNN architecture in time domain with low storage and power consumption. *Electronics, 11*(9), 1421. https://doi.org/10.3390/electronics11091421

Zhang, Y., Pan, J., Liu, X., Chen, H., Chen, D., & Zhang, Z. (2021). FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations. *Proceedings of the ACM/SIGDA International Symposium on FPGAs*, 171–182. https://doi.org/10.1145/3431920.3439287

Zhao, G., Wei, W., Xie, X., Fan, S., & Sun, K. (2022). An FPGA-based BNN real-time facial emotion recognition algorithm. *Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 20–24. https://doi.org/10.1109/ICAICA54656.2022.9782352

Zhu, B., Al-Ars, Z., & Hofstee, H. P. (2020). NASB: Neural architecture search for binary convolutional neural networks. *International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN48605.2020.9206691

CHAPTER 2
PREPARATION AND CHARACTERIZATION STUDIES OF KNO_3-HTPB BASED SOLID ROCKET PROPELLANT WITH DIFFERENT PLASTICIZERS

Monisha KARMAKAR[1]
Pratim KUMA[2]

---

[1]Department of Aerospace Engineering and Applied Mechanics, Indian Institute of Engineering Science and Technology, Shibpur, Howrah-711103, India, mkmonisha03@gmail.com, ORCID ID: 0009-0000-9489-2289.
[2]Department of Aerospace Engineering and Applied Mechanics, Indian Institute of Engineering Science and Technology, Shibpur, Howrah–711103, India, pratim.kumar.86@gmail.com, ORCID ID: 0000-0003-4151-836X.

INTRODUCTION

Composite solid propellants remain central to contemporary propulsion systems owing to their reliability, scalability, and tunable performance. The present work reports an expanded experimental investigation into potassium nitrate (KNO_3)–based composite propellants employing hydroxyl-terminated polybutadiene (HTPB) as the polymeric fuel–binder. In contrast to conventional ammonium perchlorate (AP) formulations, KNO_3 offers enhanced safety, reduced environmental impact, and improved handling characteristics, albeit at the expense of energetic performance. To address this limitation, catalytic additives and optimized curing systems were systematically explored. Propellant formulations were prepared using HTPB with plasticizers (dioctyl adipate, dioctyl phthalate, and dibutyl phthalate), cured with either toluene diisocyanate (TDI) or isophorone diisocyanate (IPDI), and catalyzed using cupric oxide (CuO) and cobalt (II, III) oxide (Co_3O_4).

A comprehensive experimental methodology encompassing controlled mixing, casting, vacuum degassing, and multi-stage thermal curing was adopted. Thermophysical and combustion-related properties—including density, moisture content, calorific value, burning rate, flame temperature, and emission spectra—were evaluated using standardized laboratory techniques. Results demonstrate that TDI-based formulations consistently outperform IPDI-based systems in terms of combustion temperature and calorific value, attributable to higher crosslink density and aromatic rigidity. Among the catalysts investigated, CuO significantly increased peak flame temperature (up to 1034 °C), while Co_3O_4 enhanced calorific value (up to 2598.8 cal g^{-1}) and promoted smoother combustion behavior. Spectral emission analysis confirmed the presence of characteristic K^+ and Cu^{2+} species, validating catalytic participation during combustion.

The expanded dataset and discussion provide deeper insight into structure–property–performance relationships in KNO_3–HTPB propellants. The findings highlight the feasibility of developing safer, chlorine-free, and environmentally benign solid propellants for educational, experimental, and small-scale aerospace applications. The work contributes to the growing body of research on green propulsion materials and establishes a foundation for future pressure-dependent and motor-scale studies.

Solid rocket propulsion has played a decisive role in the advancement of aerospace and defense technologies due to its inherent simplicity, mechanical robustness, and operational reliability. Unlike liquid propulsion systems, solid propellants integrate fuel and oxidizer into a single grain, eliminating complex feed systems and enabling long-term storability. Composite solid propellants, comprising a crystalline oxidizer dispersed within a polymeric binder matrix, dominate modern applications ranging from tactical missiles to space launch vehicle boosters (Kubota, 2002; Yang et al., 2000).

Hydroxyl-terminated polybutadiene (HTPB) has emerged as one of the most widely adopted binders for composite propellants since the 1960s. Its popularity stems from its favorable mechanical flexibility, chemical compatibility with a wide range of oxidizers, and ability to form polyurethane networks when cured with diisocyanates (Ramakrishna et al., 2002). Traditionally, ammonium perchlorate (AP) has been the oxidizer of choice due to its high oxygen balance and energetic output. However, AP-based propellants generate environmentally harmful chlorine-containing exhaust species and pose handling and disposal challenges.

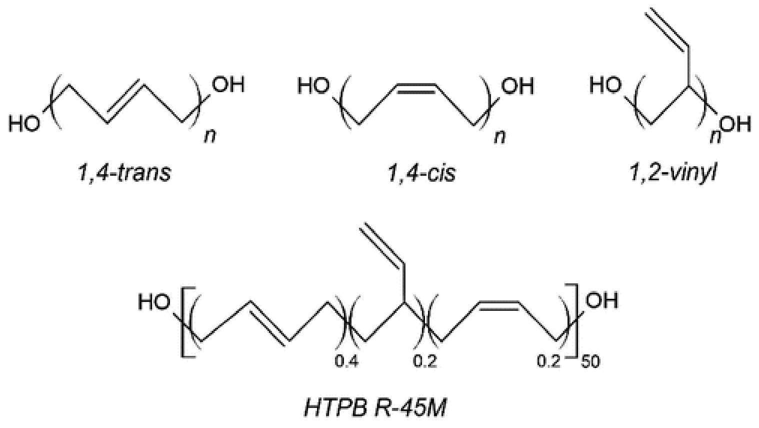


Figure 1. Chemical structure of HTPB.

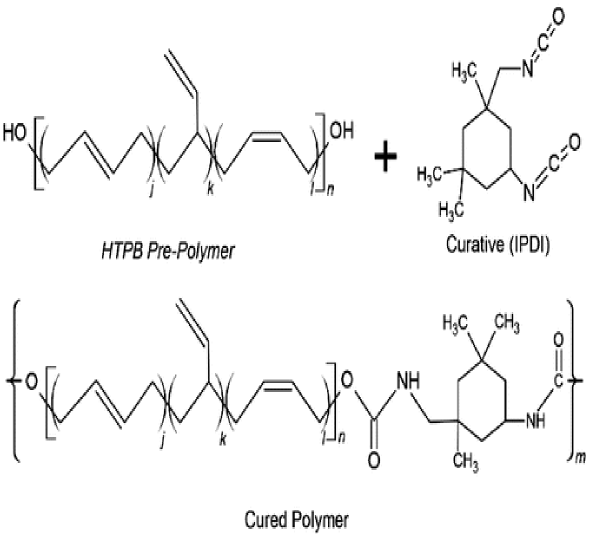


Figure 2. HTPB/IPDI

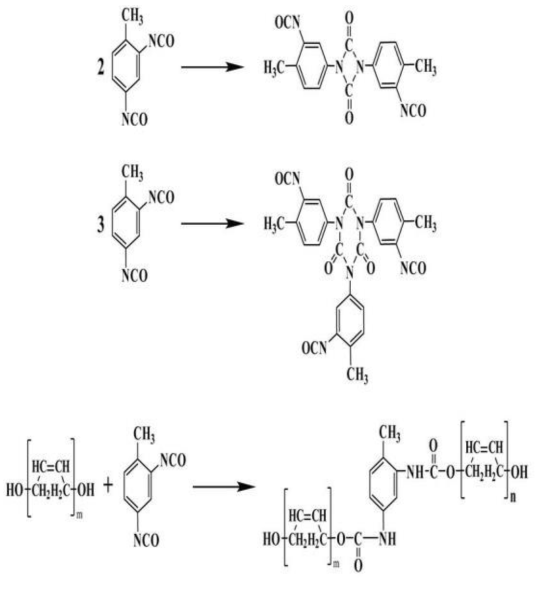


Figure 3. HTPB/TDI

In recent years, increasing emphasis on environmental sustainability and operational safety has motivated the exploration of alternative oxidizers such as ammonium nitrate (AN), ammonium dinitramide (ADN), and potassium nitrate (KNO_3) (Reddy et al., 2021).

KNO_3, though less energetic than AP, offers distinct advantages including low sensitivity to impact and friction, ease of availability, and reduced environmental impact. These attributes make KNO_3-based propellants particularly attractive for academic research, educational demonstrations, and small-scale propulsion systems.

Despite these advantages, KNO_3-based propellants often suffer from lower burning rates and reduced flame temperatures. To mitigate these drawbacks, burn rate modifiers and catalysts—typically transition metal oxides—are incorporated to tailor combustion characteristics. Cupric oxide (CuO) and cobalt oxide (Co_3O_4) are among the most studied catalysts due to their redox activity and ability to alter oxidizer decomposition pathways (Lee et al., 2011; Zhang et al., 2016).

The curing chemistry of HTPB also plays a critical role in determining final propellant properties. Diisocyanates such as toluene diisocyanate (TDI) and isophorone diisocyanate (IPDI) react with hydroxyl groups in HTPB to form a crosslinked polyurethane matrix. The molecular structure of the curing agent influences curing kinetics, crosslink density, mechanical integrity, and thermal stability (Nguyen & Wang, 2010).

The present study expands upon prior work by providing a detailed, data-rich investigation of KNO_3–HTPB composite propellants formulated with different plasticizers, curing agents, and catalysts. Beyond basic characterization, this paper emphasizes combustion diagnostics, spectral analysis, and comparative performance evaluation, with the aim of contributing a comprehensive reference for green composite propellant development.

1. CLASSIFICATION OF SOLID ROCKET PROPELLANTS

Solid rocket propellants are commonly classified based on their chemical composition, energetic mechanism, and physical structure. Broadly, they are categorized into homogeneous and heterogeneous (composite) propellants.

Homogeneous propellants consist of fuel and oxidizer combined at the molecular level. These include double-base propellants, primarily composed of nitrocellulose and nitroglycerin, which offer smooth combustion and low smoke but limited performance and scalability.

Modified double-base (MDB) propellants incorporate energetic additives such as RDX or HMX to enhance performance, though they often present increased sensitivity and processing complexity.

Heterogeneous or composite propellants consist of a crystalline oxidizer dispersed within a polymeric binder matrix that also acts as a fuel. This category dominates modern aerospace applications due to its superior mechanical strength, formulation flexibility, and higher specific impulse. Conventional composite propellants typically employ ammonium perchlorate (AP) as the oxidizer and hydroxyl-terminated polybutadiene (HTPB) as the binder. However, environmental and safety concerns associated with AP have driven research into alternative oxidizers such as ammonium nitrate (AN), ammonium dinitramide (ADN), and potassium nitrate (KNO_3). Based on oxidizer chemistry, composite propellants may further be classified as chlorine-based (e.g., AP systems) and chlorine-free or green propellants (e.g., AN-, ADN-, or KNO_3-based systems). Green propellants offer reduced environmental impact, lower toxicity, and enhanced handling safety, albeit at reduced energetic performance. Additionally, solid propellants can be classified by burning rate modifiers, where metallic or metal oxide catalysts are introduced to tailor combustion behavior, and by binder chemistry, depending on curing agents and plasticizers used to optimize mechanical and thermal properties. This study focuses on chlorine-free composite solid propellants, specifically KNO_3–HTPB formulations, representing a safer and environmentally benign alternative for small-scale and experimental propulsion applications.

2. LITERATURE REVIEW

Evolution of Composite Solid Propellants

The evolution of composite solid propellants has been closely linked to advances in polymer chemistry and materials science. Early propellants relied on asphalt and polysulfide binders, which were gradually replaced by synthetic polymers offering superior mechanical and thermal properties. HTPB emerged as a dominant binder due to its controllable molecular weight, terminal hydroxyl functionality, and excellent compatibility with energetic additives (Smith & Anderson, 2002).

Oxidizer Selection and Environmental Considerations

While AP-based propellants remain unmatched in performance, environmental concerns have prompted significant research into chlorine-free alternatives. Raghu et al. (2014) demonstrated that KNO_3-based formulations exhibit enhanced safety and stability, though at reduced energetic efficiency. Recent reviews emphasize the importance of balancing performance with environmental impact, particularly for future aerospace systems subject to stricter emission regulations (Ghosh & Sengupta, 2022).

Curing Chemistry of HTPB

The curing reaction between hydroxyl-terminated polymers and diisocyanates forms the structural backbone of composite propellants. TDI-based systems cure rapidly and produce rigid networks, whereas IPDI-based systems cure more slowly and impart improved flexibility and weather resistance (Nguyen & Wang, 2010; Shrivastava & Kulkarni, 2011). The NCO:OH ratio, curing temperature, and presence of catalysts critically influence the final properties.

Role of Burn Rate Catalysts

Transition metal oxides have been extensively studied as combustion catalysts. CuO has been shown to enhance flame temperature and energy release, while Co_3O_4 promotes smoother combustion and modifies burn rate behavior (Lee et al., 2011; Zhang et al., 2016). Spectroscopic studies reveal that these catalysts participate actively in redox reactions during combustion, altering gas-phase and condensed-phase kinetics (Tiwari & Jain, 2013).

Combustion Diagnostics and Spectroscopy

Advanced diagnostic techniques such as 2D and 3D emission spectroscopy provide valuable insight into combustion mechanisms by identifying intermediate species and tracking temporal evolution of flames. The detection of K^+, Cu^{2+}, and OH^* emissions has been widely used to correlate chemical reactions with macroscopic performance metrics (Jani & Shah, 2016).

3. MATERIALS AND METHODS

Materials

Table 1. Propellant Material Composition

Component	**Chemical Name**	**Formula**	**Function**	**Wt% Range**
HTPB	Hydroxyl-Terminated Polybutadiene	HO–$(C_4H_6)_n$–OH	Fuel & Binder	12%
KNO_3	Potassium Nitrate	KNO_3	Oxidizer	65–75%
CuO	Cupric Oxide	CuO	Burn Rate Catalyst	1%
Co_3O_4	Cobalt (II, III) Oxide	Co_3O_4	Burn Rate Catalyst	0.5%
DOA/DOP/DBP	Plasticizers	Various	Process Aid/Flexibility	8%
IPDI/TDI	Diisocyanates	Various	Curing Agent	1–2%

Preparation Procedure

The preparation process began with the drying of potassium nitrate (KNO_3) at 100 °C for 2 hours, after which it was sieved to obtain a particle size finer than 150 mesh. Hydroxyl-terminated polybutadiene (HTPB) was then blended with selected plasticizers, such as dioctyl adipate (DOA), dioctyl phthalate (DOP), or dibutyl phthalate (DBP), at 40 °C.

This was followed by the sequential addition of catalysts, including cupric oxide (CuO) and cobalt oxide (Co_3O_4), along with the sieved KNO_3, in a vacuum mixer to ensure homogeneous dispersion. The curing agent, either isophorone diisocyanate (IPDI) or toluene diisocyanate (TDI), was introduced at the final mixing stage, with dibutyltin dilaurate optionally added to accelerate the curing process. The resulting mixture was cast into Teflon-lined molds and subjected to vacuum degassing to remove entrapped air bubbles.

Finally, the samples were thermally cured, initially at room temperature for 24 hours, followed by an extended curing period of 48–72 hours at 50–60 °C to achieve complete polymerization.

Characterization Techniques

Density, moisture content, and calorific value were measured using standard analytical instruments. Burning rate was determined via strand burner tests. Combustion temperature and emission spectra were recorded using thermocouples and flame emission spectroscopy.

4. RESULTS AND DISCUSSION

Calorific Value: Measured using a LABTRONICS bomb calorimeter with oxygen at 400 psi, calibrated with benzoic acid.

Table 2. Calorific Value Results Of Propellant Formulations

Sample	**Max Temp Rise (°C)**	**Calorific Value (cal/g)**
IPDI Cupric	0.745	1849.23
IPDI Cobalt	0.746	1779.30
TDI Normal	0.939	2356.90
TDI Cupric	0.940	2275.50
TDI Cobalt	0.999	2598.80

The TDI–Cobalt formulation exhibited the highest calorific value (2598.8 cal/g), indicating superior energy release, likely due to enhanced catalytic activity of Co_3O_4.

Density: Determined using a WENSAR Digital Analytical Weighing Balance (accuracy 0.001 g) by dividing sample weight by volume.

Table 3. Density Of Propellant Samples

Sample	**Density (g/cm^3)**
IPDI Cupric	1.333
IPDI Cobalt	1.491
TDI Normal	1.234
TDI Cupric	1.285
TDI Cobalt	1.670

Higher densities in TDI–Cobalt suggests better packing of oxidizer and catalyst particles.

Moisture Content: Analyzed via a moisture balance using the loss-on-drying method.

Table 4. Propellant Moisture Results

Sample	Moisture (%)
IPDI Cupric	0.85
IPDI Cobalt	0.43
TDI Normal	0.90
TDI Cupric	0.96
TDI Cobalt	0.68

Low moisture content (<1%) indicates effective drying and sealing, critical for propellant stability.

Burning Rate: Calculated by timing the combustion of 1 cm and 2 cm cylindrical strands with a stopwatch.

Sample	Length (cm)	Time (s)	Burn Rate (cm/s)
TDI Cobalt	1	58.5008	0.0170940
TDI Cupric	1	45.9325	0.0217710
TDI Normal	1	35.705	0.0280073
IPDI Cobalt	1	68.6625	0.0145639
IPDI Cupric	1	43.4175	0.0230322

TDI Normal exhibited the highest burning rate (0.0280 cm/s), likely due to the absence of catalysts slowing decomposition. Co_3O_4 significantly enhanced burn rates in both TDI and IPDI systems, while CuO produced higher combustion temperatures.

Combustion Analysis: Conducted using 2D/3D emission spectroscopy (Avantes spectrometer) to identify species like K^+ and Cu^{2+}.

Tablo 5. Combustion Temperature

Sample	Temperature (°C)
TDI Cupric	1034
TDI Normal	900.4

Sample	**Temperature (°C)**
TDI Cobalt	927.5
IPDI Cupric	860.1
IPDI Cobalt	813

TDI–Cupric achieved the highest temperature (1034°C), confirming CuO’s role in enhancing thermal energy release. In below, the first graph, "Refined Burning Rate of Various Propellant Formulations", shows that the TDI Base formulation exhibits the highest burning rate at 0.028007 cm/s, followed by IPDI Cupric (0.023032 cm/s) and TDI Cupric (0.021771 cm/s). The lowest burning rates are observed in TDI Cobalt (0.017094 cm/s) and IPDI Cobalt (0.014564 cm/s), indicating that cobalt oxide generally slows down the combustion rate compared to cupric oxide.

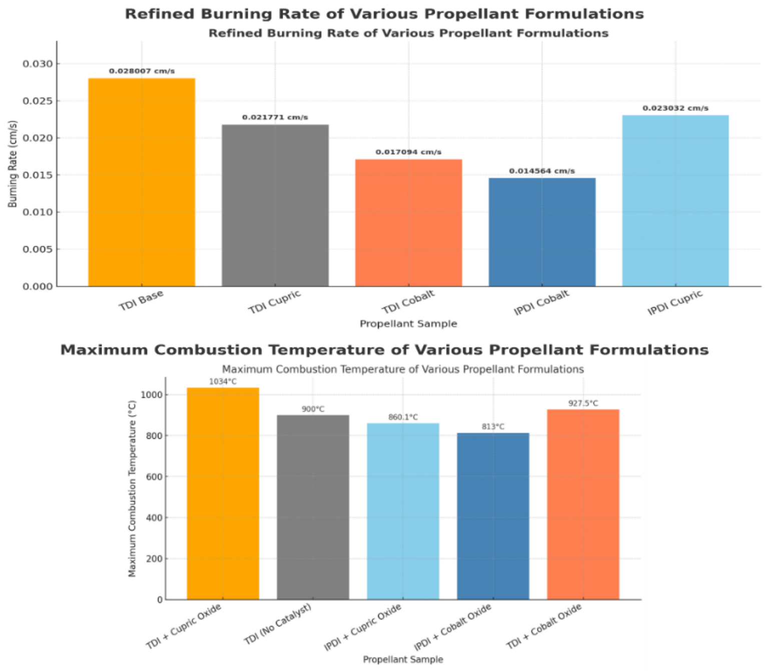


Figure 1. Combustion Characteristics Of Propellant Formulations

The second graph, "Maximum Combustion Temperature of Various Propellant Formulations", reveals that TDI + Cupric Oxide achieves the highest combustion temperature at 1034°C, while IPDI + Cobalt Oxide has the lowest at 813°C. Overall, formulations containing cupric oxide tend to produce higher combustion temperatures than those with cobalt oxide, and TDI-based propellants generally outperform IPDI-based ones in terms of thermal output.

From the graph we can say that-

TDI + Cupric Oxide (1034°C) has the highest temperature, confirming that: TDI as a binder and CuO as a catalyst, significantly improve combustion performance, confirming it as the most thermally energetic composition.

TDI without catalyst (900°C) still performs well, better than any IPDI-based formulation. This suggests that TDI alone contributes to higher energy release due to its aromatic structure and better crosslinking properties.

TDI + Cobalt Oxide (927.5°C) performs better than TDI alone.

IPDI + Cupric Oxide (860.1°C) performs moderately, confirming CuO helps, but IPDI's lower reactivity limits max temperature compared to TDI.

IPDI + Cobalt Oxide (813°C) is the least energetic formulation among the four. This shows both:

- IPDI is less reactive
- Co_3O_4 is a weaker catalyst than CuO in this system.

Spectral Analysis: Emission spectroscopy revealed K^+ (766 nm) and Cu^{2+} (510–530 nm) peaks, with TDI–Cupric showing stronger emissions, indicating more energetic combustion.

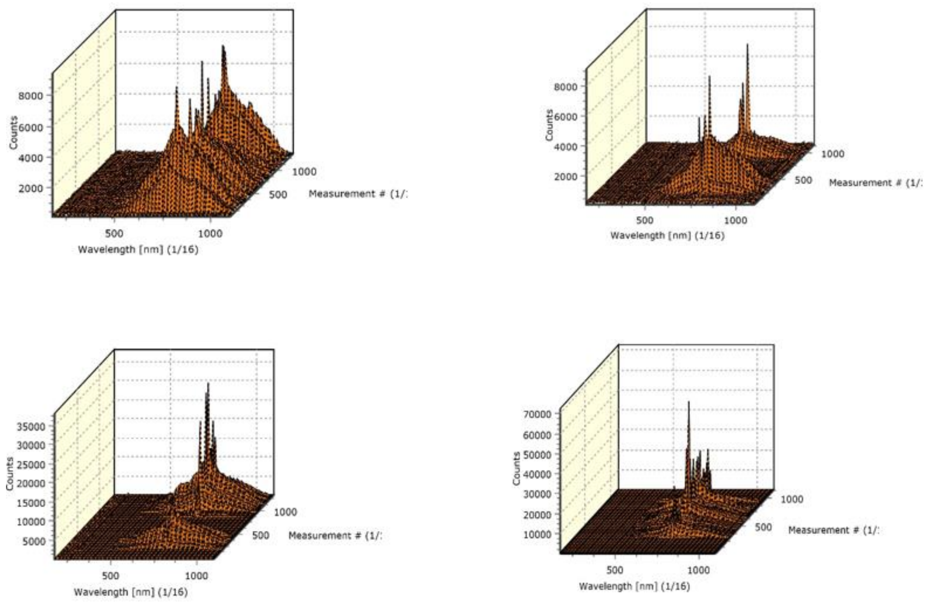


Figure 2. Combustion Emission Spectra Of Propellant Samples

The spectral analysis of different fuel formulations was carried out using 3D spectra (Wavelength vs. Intensity vs. Time or Measurement Index) offers a dynamic representation of how these emissions evolve throughout the combustion event. Among all, the catalyzed samples (especially TDI Cupric) exhibited more complex and multi-peaked spectra, indicating a more energetic and chemically diverse combustion environment due to catalytic enhancement. These observations affirm the role of metallic additives in modifying the thermal decomposition pathways and increasing energy release rates in composite propellants.

Table 6. Emission Peaks Of Propellant Samples

Sample	**Peak (nm)**	**Functional Group**
IPDI Cobalt	766	Potassium ion (K^+)
IPDI Cupric	766, 515–525	K^+, Cu^{2+} (from CuO)
TDI Base	765	K^+
TDI Cupric	766, 510–530	K^+, Cu^{2+}(stronger emission than IPDI Cupric)

Table 7. Functional Group / Species Reference Mapping

Species	**Wavelength (nm)**	**Comment**
Potassium (K^+)	766.5, 769.9	Strong doublet lines, dominant in combustion of KNO_3
Copper (Cu^{2+})	510–530	Green-blue emission of copper compounds
Cobalt (Co^{2+})	345–375	Weak violet-blue emission; not resolved in current graphs
CH (methylidyne)	~431	From hydrocarbon combustion HTPB not clearly visible.
OH Radical	~309	UV region, often seen in flame — not resolved here

5. FUTURE SCOPE OF THE PRESENT WORK

The present study has established a foundational understanding of the combustion characteristics of HTPB-based composite solid propellants using KNO_3 as the oxidizer and metallic catalysts such as CuO and Co_3O_4. While key parameters like calorific value, density, moisture content, and combustion temperature have been evaluated, several important aspects remain unexplored, which presents valuable opportunities for further research and development.

5.1 Unperformed Pressure Measurement Study

A significant experimental limitation in this study is the absence of chamber pressure measurement during combustion. Measuring pressure–time profiles is crucial for characterizing the real-world performance of solid propellants in rocket motors. This experiment, typically conducted using a closed bomb calorimeter or strand burner under controlled pressure, provides data such as:

- Peak pressure generation
- Burning rate dependence on pressure
- Ignition delay and pressure rise time
- Stability of combustion under confined conditions

These results would enable determination of the pressure exponent (n) in the empirical burning rate law:

$$\text{Burning Rate} = a \cdot P^n$$

Where a is the pre-exponential constant and n indicates how strongly burn rate depends on pressure. Future work should include pressure measurements to validate combustion efficiency, thermal stability, and ignition consistency across the various formulations studied here.

5.2 Suggested Future Experiments

To comprehensively evaluate and optimize the performance of these propellants, the following experimental extensions are recommended:

Strand Burner Testing Under Pressure

Conduct strand burn rate experiments across a range of pressures (1–10 atm) to obtain accurate pressure-dependent burn profiles. This would help assess whether a formulation is suitable for low-thrust or high-thrust applications.

Thermal Analysis: TGA & DSC

Use Thermogravimetric Analysis (TGA) and Differential Scanning Calorimetry (DSC) to study the decomposition kinetics, phase transitions, and thermal stability of each formulation. These techniques reveal critical information about heat release, ignition temperature, and compatibility of components.

Microstructural Studies (SEM/EDX)

Apply Scanning Electron Microscopy (SEM) and Energy Dispersive X-ray Analysis (EDX) to visualize and chemically analyze the dispersion of oxidizer and catalysts in the cured binder. Poor dispersion can cause uneven combustion and performance instability.

Spectral Flame Diagnostics at Different Conditions

Expand 2D and 3D emission spectroscopy to different pressure and temperature environments to study changes in species evolution. This will help in better identifying metal–ion interactions and combustion efficiency under flight-like conditions.

Chemical Compatibility & Stability Studies

Long-term storage tests under humidity and temperature cycles can help determine the shelf-life and chemical compatibility of propellant components.

5.3 Research Opportunities for Future Work

For researchers interested in continuing this work or exploring its practical applications, the following directions are highly recommended:

New Catalyst Exploration

In addition to CuO and Co_3O_4, catalysts like nano-Fe_2O_3, MnO_2, NiO, or TiO_2 can be studied for their ability to alter flame temperature, burn rate, and sensitivity.

Alternative Binder Systems

Exploration of other binders such as:

- GAP (Glycidyl Azide Polymer) – Energetic and gas-generating
- PBAN (Polybutadiene Acrylonitrile) – More rigid and thermally stable
- HTPE (Hydroxyl-Terminated Polyether) – Improved mechanical properties

These systems may improve performance or reduce environmental impact.

Incorporation of Metallic Fuels

Adding aluminum or magnesium powders to the formulation could significantly enhance energy output and specific impulse. However, this must be balanced with issues of slag formation and safety.

Green Propulsion Materials

Working toward chlorine-free oxidizers (such as Ammonium Nitrate (AN), Ammonium Dinitramide (ADN)) and lead-free catalysts would make the system eco-friendlier and more compliant with future regulations.

Computational Simulation

Use computational fluid dynamics (CFD) and reaction kinetics simulations to model flame spread, pressure wave propagation, and regression rate of the propellant. This can be useful for designing scalable rocket motors.

5.4 Potential Applications and Real-World Impact

The formulations developed in this study—particularly those using TDI-CuO and TDI-Co_3O_4—show promising combustion characteristics and thermal energy output. With further optimization, such propellants can be applied to:

- Model rocketry and student satellite launch vehicles (CanSat, Sounding rockets)
- Laboratory demonstration of safe solid propulsion
- Low-cost booster stages for UAVs or expendable drones
- Green propulsion systems in educational and test-bed missions

By refining the combustion parameters, improving safety, and reducing environmental impact, this study lays the groundwork for future work in the field of sustainable solid propulsion.

6. LIMITATIONS OF THE PRESENT STUDY

Despite providing valuable insights into the formulation and combustion behavior of KNO_3–HTPB based composite solid propellants, the present study is subject to certain limitations that should be acknowledged. The experimental investigations were primarily conducted under ambient pressure conditions, and pressure-dependent burning rate measurements were not performed. As a result, key combustion parameters such as the pressure exponent and steady-state burning behavior under realistic motor chamber pressures could not be evaluated. These parameters are critical for predicting performance in actual rocket motor applications.

The study was limited to laboratory-scale strand combustion tests, and no motor-scale or static firing experiments were conducted. Consequently, thrust characteristics, chamber pressure evolution, erosive burning effects, and nozzle–propellant interactions were not assessed.

In addition, the concentration range of burn rate catalysts (CuO and Co_3O_4) was restricted, and the influence of varying catalyst particle size, morphology, or nano-scale additives was not explored.

Mechanical properties such as tensile strength, elongation at break, and viscoelastic behavior of the cured propellant grains were not evaluated. These properties are essential for assessing structural integrity during handling, storage, and operation, particularly under thermal and vibrational loads. Furthermore, long-term aging, compatibility, and environmental stability studies were beyond the scope of the present work.

Spectral diagnostics were limited by the resolution of the available instrumentation, restricting the detection of certain transient radical species in the ultraviolet region. Finally, numerical modeling and combustion simulations were not incorporated, limiting the ability to generalize the experimental findings across a broader range of operating conditions.

Addressing these limitations in future investigations will enable a more comprehensive understanding of KNO_3–HTPB propellant systems and support their optimization for practical aerospace propulsion applications.

CONCLUSION

This experimental study successfully demonstrated the preparation, processing, and performance evaluation of HTPB-based composite solid propellants using potassium nitrate (KNO_3) as the oxidizer and various combinations of curing agents (TDI and IPDI) and burn rate catalysts (CuO and Co_3O_4). Through a series of controlled experiments, the effect of these variables on key performance metrics such as burning rate, combustion temperature, and calorific value was thoroughly assessed.

Among the formulations tested, the TDI–Cupric Oxide sample achieved the highest combustion temperature of 1034 °C, indicating the most thermally energetic behavior. However, the TDI–Cobalt Oxide sample demonstrated the highest calorific value of 2598.8 cal/g, confirming its superior energy output per unit mass. On the other hand, the IPDI-based formulations consistently showed lower combustion efficiency and energy release, with the IPDI–Cobalt Oxide sample producing the lowest combustion temperature (813 °C) and calorific value.

Burning rate analysis revealed that cobalt oxide catalysts significantly enhance the combustion rate, particularly in IPDI–Cobalt and TDI–Cobalt systems, suggesting a strong catalytic influence on decomposition kinetics. In contrast, cupric oxide, while slightly slower in burn rate, produced higher combustion temperatures and cleaner spectral signatures.

In summary, both the choice of isocyanate (curing agent) and metal oxide catalyst play a pivotal role in tailoring the performance of composite propellants. TDI-based systems, especially when combined with CuO or Co_3O_4, emerged as the most effective combinations for high-energy applications. These findings provide valuable insight into the optimization of green and stable propellant formulations for academic, industrial, and defense-related propulsion technologies.

REFERENCES

Ghosh, A., & Sengupta, S. (2022). Advanced composite propellants: Recent developments and challenges. *Materials Today: Proceedings, 61*, 140–148.

Jani, S. P., & Shah, R. P. (2016). Optical emission spectroscopy of composite propellant flames. *Defence Science Journal, 66*(2), 161–168.

Kubota, N. (2002). *Propellants and explosives: Thermochemical aspects of combustion* (2nd ed.). Wiley-VCH.

Lee, J., Kim, S. H., & Park, J. (2011). Catalytic effects of transition metal oxides on the combustion of composite solid propellants. *Journal of Energetic Materials, 29*(3), 219–234.

Nguyen, Q. T., & Wang, W. (2010). Influence of isocyanate curing agents on the mechanical and thermal properties of HTPB-based polyurethane binders. (2), 198–204.

Raghu, K., Rao, K. V. S., & Reddy, B. R. (2014). Studies on potassium nitrate based composite solid propellants. *Defence Technology, 10*(4), 345–351.

Ramakrishna, R. V., Rao, K. V. S., & Sarwade, D. B. (2002). HTPB-based composite solid propellants—A review. *Defence Science Journal, 52*(3), 335–349.

Reddy, T. S., Prasad, M. V. R., & Chakravarthy, S. R. (2021). Green propellants for future space propulsion systems. *Aerospace Science and Technology, 109*, 106416.

Shrivastava, P., & Kulkarni, P. S. (2011). Curing behavior of HTPB with aromatic and aliphatic diisocyanates. *Journal of Applied Polymer Science, 120*(3), 1565–1573.

Smith, A. L., & Anderson, R. L. (2002). Polymer binders in solid rocket propulsion. *Progress in Aerospace Sciences, 38*(8), 695–736.

Tiwari, M., & Jain, R. (2013). Spectral emission analysis of composite propellant combustion. *Defence Science Journal, 63*(2), 188–194.

Yang, V., Brill, T. B., & Ren, W. Z. (2000). *Solid propellant chemistry, combustion, and motor interior ballistics*. American Institute of Aeronautics and Astronautics.

Zhang, Y., Li, S., & Zhu, W. (2016). Effects of metal oxide catalysts on the burning rate of HTPB-based composite propellants. *Propellants, Explosives, Pyrotechnics, 41*(5), 837–844.

CHAPTER 3
SIGNAL TO INSIGHT: AI-DRIVEN SIGNAL PROCESSING

Mrunmayee V. DAITHANKAR[1]
Dr. Suhas S. PATIL[2]

[1]Electronics Engineering, Shivaji University, Kolhapur, Maharashtra, India, mrunmayeed30@gmail.com, ORCID: 0000-0003-3180 491X.
[2]Electronics Engineering, Ex. I/C Principal, KBP College of Engineering, Satara. Maharashtra, India.

INTRODUCTION

Intelligent Signal Processing (ISP) combines established signal processing methods with artificial intelligence and machine learning techniques to build systems that can understand and respond to complex forms of data, including speech signals, images, and sensor outputs. Rather than depending solely on predefined mathematical models, these systems learn patterns directly from data, allowing them to perform tasks such as pattern identification, fault analysis, adaptive control, and intelligent decision-making (Chen, 2023). As a result, ISP has found applications across diverse fields, like healthcare diagnostics, communication systems, energy monitoring, and automation.

The fundamental idea behind ISP lies in integrating traditional digital signal processing with learning-based models such as neural networks and deep learning frameworks. This integration enables systems to automatically extract relevant features from raw, high-dimensional signals and adapt their behaviour as operating conditions change (Wang, 2023). By learning system characteristics from observed data instead of rigid analytical formulations, ISP offers improved flexibility and resilience in uncertain environments, supporting advanced applications including smart power system supervision and enhanced speech recognition technologies.

1. LIMITATIONS OF CONVENTIONAL SIGNAL PROCESSING IN NON-LINEAR, NON-STATIONARY ENVIRONMENTS

Conventional signal processing methods are based on the hypotheses of linearity and stationarity. These assumptions become substantial limitations when dealing with real-world, complex signals that exhibit non-linear and non-stationary behaviour (Proakis, 2006).

Limitations in Non-Stationary Environments

Limited Representation of Temporal Variability: Conventional techniques such as the classical Fourier Transform focus solely on frequency information and implicitly assume that signal characteristics remain unchanged over time. As a result, they fail to reflect evolving spectral or statistical behaviour within dynamic signals.

Inherent Time–Frequency Compromise: To address partial non-stationarity, the Short-Time Fourier Transform introduces windowed analysis; however, the use of a fixed window length imposes an unavoidable compromise between temporal and spectral resolution. (Cohen, 1995) Longer windows improve frequency discrimination but obscure short-lived events, whereas shorter windows enhance time localisation at the expense of frequency detail (Sejdic, 2009). This constraint hampers effective analysis of signals containing both abrupt transients and slowly varying components.

Inadequacy of Classical Modelling Approaches: Linear statistical models, including autoregressive and ARMA frameworks, are generally insufficient for representing the complex, time-dependent nature of non-stationary data. Consequently, they provide oversimplified descriptions of real-world signals, such as biomedical recordings (e.g., EEG) or structural response measurements.

Limitations in Non-Linear Environments

Violation of the Superposition Assumption: Traditional linear models are built on the principle that system responses add linearly. However, many real-world systems, particularly in physical devices and biological processes, exhibit non-linearity, where the output generates additional frequency components such as harmonics and intermodulation terms that are absent in the original input (Akkaya, 2025).

Limitations of Conventional Linear Filtering: Linear filtering techniques are inherently incapable of accurately representing or suppressing these non-linearity-induced spectral components, which restricts their effectiveness in practical signal analysis.

Absence of a Generalised Non-Linear Framework: In contrast to linear system theory, non-linear systems lack a comprehensive, universally applicable theoretical foundation. As a result, modelling strategies are often tailored to specific forms of non-linearity, necessitating specialised and frequently complex analytical or computational methods.

Risk of Model Instability and Overfitting: While simple non-linear representations, such as polynomial-based models, offer increased flexibility, they often introduce challenges, including unstable behaviour, sensitivity to noise, and a heightened risk of overfitting. These issues limit their practical usability unless supported by advanced regularisation or optimisation techniques.

1.1 Motivation for AI Integration in Modern Signal Processing

The inspiration for AI integration in modern signal processing pipelines stems primarily from the demand to overcome the limitations of conventional, model-based methods when dealing with complex, real-world data and dynamic environments. AI offers *superior adaptability, enhanced accuracy, automation, and predictive capabilities* that transform raw data into actionable intelligence (Wang, 2009).

Significant Inspirations Include (Bishop, 2006)

Handling Complexity and Non-Linearity: Many contemporary signal sources exhibit irregular, non-linear behaviour that cannot be adequately described using fixed analytical models. In areas such as next-generation wireless systems and industrial sensing, data-driven AI methods—especially deep learning—are better suited to capturing such complexity by learning patterns directly from large-scale observations.

Superior Performance and Accuracy: Hybrid systems that combine AI with classical signal processing frequently deliver improved results in tasks including denoising, pattern classification, and feature discovery. Learning-based models are capable of identifying weak or hidden structures that may be overlooked by conventional algorithms, enhancing reliability in fields such as healthcare diagnostics and fault analysis.

Automation and Efficiency: AI reduces manual intervention by automating routine and computation-heavy processes such as preprocessing and feature extraction. This allows specialists to concentrate on interpretation and decision-making, leading to more efficient analytical workflows.

Adaptability and Real-Time Operation: AI-enabled signal processing systems can adjust continuously to evolving operating conditions, supporting real-time applications such as autonomous platforms, adaptive networks, and continuous health monitoring.

Enhanced Decision-Making and Predictive Insights: Through real-time analysis and forecasting, AI facilitates anticipatory actions, such as early detection of equipment degradation, thereby minimizing downtime and maintenance costs.

Feature Engineering and Dimensionality Reduction: Established transforms, including Fourier and wavelet methods, remain effective for compact signal representation. When integrated with AI models, they improve computational efficiency and robustness.

Addressing Data Volume Challenges: The rapid growth of sensor-driven data exceeds the capacity of traditional techniques, making AI essential for scalable processing and meaningful insight extraction.

1.2 Transition from Model-Driven to Data-Driven Paradigms *In AI-Based Signal Processing*

The shift from model-driven to data-driven AI signal processing moves from traditional mathematical models to learning patterns directly from vast datasets, driven by deep learning's success. It offers superior performance in complex tasks but demands huge data, while hybrid approaches combining both are emerging to gain robustness, efficiency, and explainability by integrating domain knowledge, moving towards "Smarter AI" for better real-world automation and insights (Gannot, 2024).

Model-Driven Paradigm (Traditional)

The core idea uses established scientific laws, equations (like Fourier and Laplace), and system understanding to build explicit models. It requires less data, offers strong guarantees (stability, performance), is interpretable, and leverages expert knowledge. The limitation of this approach is struggles with extreme complexity, noise, and unmodeled dynamics; building models is slow and expensive (requires scarce experts). The examples are traditional filters, spectral analysis based on known physics (Shlezinger, 2023).

Data-Driven Paradigm (Modern)

The core idea leverages machine learning (ML) and deep learning (DL) to learn features and mappings directly from data, often end-to-end. The advantage is the achievement of the state-of-the-art performance, automates feature extraction, and handles high complexity (e.g., computer vision, NLP). The difficulty is the need for massive, high-quality datasets; it can be black-box (hard to interpret); it struggles with out-of-distribution data (Razzaq, 2025). The examples are CNNs for image/audio, RNNs for time-series, etc.

Table 1: Comparison Of Model and Data-Driven Paradigms

Aspect	Model Driven Paradigm	Data Driven Paradigm
Core Principle	Relies on explicit mathematical and physical models of the signal generation process	Learns signal characteristics directly from data without predefined models
Knowledge Source	Domain expertise, physics-based equations, and analytical assumptions	Large volumes of labelled or unlabelled data
Flexibility	Limited adaptability to complex, time-varying, or nonlinear environments	Highly flexible and adaptive to diverse and evolving signal conditions
Handling non-linearity	Often struggles with strong non-linearities	Excels at modeling complex non-linear relationships
Interpretability	High interpretability due to transparent mathematical structure	Often low interpretability (black-box nature of deep models)
Data Requirement	Requires relatively small datasets	Requires large and representative datasets for effective learning
Computational Complexity	Typically, lower computational cost	High computational demand during training and inference
Robustness to Noise	Performance degrades if model assumptions are violated	Can be robust to noise when trained on diverse data
Generalization	Generalizes well within assumed model constraints	Generalization depends heavily on training data quality
Real-Time Implementation	Easier to deploy in real-time systems	Real-time deployment may be challenging due to latency
Adaptability	Limited self-learning capability	Strong self-learning and adaptation capabilities

Examples	Kalman filters, Wiener filters, matched filtering	Deep neural networks, CNNs, RNNs, transformers
Typical Applications	Radar, sonar, and classical communication systems	5G/6G, speech recognition, biomedical, and IoT signal analysis
Scalability	Less scalable to high-dimensional data	Highly scalable to large-scale, high-dimensional signals
Design Effort	Requires careful manual model design	Requires extensive data collection and training

2. MACHINE LEARNING PARADIGMS FOR SIGNAL PROCESSING

Machine learning paradigms in signal processing use core ML types- Supervised, Unsupervised, and Reinforcement Learning. They're applied to signals (audio, biomedical, sensor data) through stages: pre-processing (denoising), processing (feature extraction), and application (classification/clustering), often leveraging techniques like Fourier transforms for frequency analysis and neural networks for complex pattern recognition. ML models, especially deep learning, learn representations similar to Fourier transforms (spectral analysis) to understand frequency content in audio or communications. Models like Recurrent Neural Networks (RNNs) excel at capturing the time-dependent nature of signals (e.g., speech, sensor streams) (Razzaq, 2025). Instead of manual methods (like autocorrelation), ML automatically learns relevant features from raw signals for tasks like pattern recognition.

Comparison of Supervised, Unsupervised, Semi-Supervised, and Reinforcement Learning For Signals

Table 2 gives a detailed comparison of the machine learning paradigms for signal processing.

Table 2. Comparison of Supervised, Unsupervised, Semi-Supervised, and Reinforcement Learning

Aspect	**Supervised Learning**	**Unsupervised Learning**	**Semi-Supervised Learning**	**Reinforcement Learning**
Definition	Learns a mapping from	Discovers hidden	Combines a small	Learns by interacting

	inputs to outputs using labelled data	patterns or structures from unlabelled data	amount of labelled data with a large amount of unlabelled data	with an environment and receiving rewards or penalties
Data Requirement	Fully labelled dataset	Completely unlabelled dataset	Few labelled + many unlabelled samples	No labelled dataset; uses feedback (reward signal)
Learning Objective	Minimize prediction error	Identify inherent structure or distribution	Improve learning accuracy using limited labels	Maximize cumulative reward over time
Feedback Mechanism	Direct feedback via correct labels	No explicit feedback	Partial feedback from labelled data	Delayed and scalar reward feedback
Typical Tasks	Classification, regression	Clustering, dimensionalit y reduction, anomaly detection	Classificatio n with scarce labels	Decision-making, control, optimization
Common Algorithms	Linear/Logisti c Regression, SVM, k-NN, Decision Trees, Neural Networks	k-Means, Hierarchical Clustering, DBSCAN, PCA, Autoencoders	Self-training, Co-training, Label Propagation, Semi-Supervised SVM	Q-Learning, SARSA, Deep Q-Networks (DQN), Policy Gradient
Computationa l Complexity	Moderate to high (depends on dataset size)	Generally lower, but may scale poorly with high dimensions	Higher than supervised due to hybrid processing	High due to continuous interaction and exploration
Advantages	High accuracy when labels are reliable	No labelling cost; reveals hidden data structure	Reduces labelling cost while improving performance	Suitable for sequential and dynamic environments
Limitations	Requires large labelled datasets	No direct prediction targets	Sensitive to incorrectly labelled data	Training can be unstable

				and time-consuming
Real-World Applications	Image recognition, medical diagnosis, spam detection	Market segmentation, topic modeling, and fraud detection	Speech recognition, web content classificatio n	Robotics, game playing, and autonomous vehicles
Suitability for Real-Time Systems	Limited (depends on model)	Limited	Limited	Highly suitable

Feature-Based ML Vs. End-To-End Learning

Feature-based machine learning (ML) relies on manual feature engineering to extract relevant information from raw data, while end-to-end (E2E) learning automatically learns features directly from the raw input using deep neural networks.

Feature-Based ML: In feature-based ML, domain experts use their knowledge to manually identify, extract, and select valuable features from the raw input data. These crafted features are then fed into a traditional ML algorithm (e.g., support vector machines, logistic regression, decision trees) to train a model. The key characteristics are:

- Requires Human Expertise: Performance heavily relies on the quality of human-engineered features and domain-specific knowledge.
- Interpretable: The models are often considered "white boxes" because the features used in decision-making are explicit and understandable.
- Data Efficiency: Can work effectively with smaller datasets, as the manual feature engineering helps focus the model's learning on relevant aspects.

End-to-End (E2E) Learning: E2E learning, typically using deep learning models like Convolutional Neural Networks (CNNs), bypasses manual feature engineering. The model takes raw data as input and learns hierarchical feature representations through its many layers, from basic features in early layers to complex, abstract features in deeper layers, producing an output directly. The key characteristics are:

- **Automated Feature Extraction:** Features are learned automatically, reducing the need for manual intervention and extensive domain expertise during the feature engineering phase.
- **Less Interpretable**: These models are often considered "black boxes" due to their complex, non-linear structure, making it difficult for humans to fully understand how decisions are made.
- **High Performance in Complex Tasks:** Excels in complex tasks like image recognition and natural language processing, often achieving state-of-the-art results (Aburakhia, 2024).

Bias - Variance Trade-Off in Signal Datasets

The Bias-Variance Trade-off in signal datasets is the fundamental challenge of balancing a model's simplicity (low complexity, high bias, low variance) against its ability to capture intricate patterns (low bias, high variance, potential overfitting), aiming for minimal total prediction error on new data by finding a "sweet spot" in model complexity, crucial for building robust signal processing models that generalize well. High bias means underfitting (too simple), while high variance means overfitting (too sensitive to training noise) (Geman, 1992).

- **Bias:** Error from overly simplistic assumptions; the model consistently misses the true signal (underfitting).
- **Variance:** Error from model sensitivity to training data fluctuations; predictions change wildly with new data (overfitting).
- **Total Error:** Sum of Bias^2, Variance, and Irreducible Error (noise). In signal datasets, the following are some important concepts related to bias and complexity.
- **Low Complexity (High Bias):** A simple model (e.g., linear) might miss subtle signal trends but will be consistent.
- **High Complexity (High Variance):** A complex model (e.g., deep neural net) can fit the training signal perfectly but might mistake noise for real patterns, failing on new signals.

3. DEEP LEARNING ARCHITECTURES FOR SIGNAL PROCESSING

Convolutional Neural Network For 1D, 2D, And 3D Signals

Convolutional Neural Networks (CNNs) are adapted for signals by matching the dimensionality of the learnable kernels to the inherent structure of the data. While 2D CNNs are standard for images, 1D and 3D variants are essential for sequential and volumetric signal processing.

1D CNNs: Sequential Signals: 1D CNNs use kernels that slide along a single axis, making them ideal for time-series and periodic data where local temporal patterns are critical. The architecture consists of 1D convolutional layers, pooling (typically max-pooling), and fully connected layers (Kiranyaz, 2021). They are computationally efficient and suitable for real-time mobile or edge device applications. The key applications are: Biomedical signals like ECG, EEG, and EMG, etc. Audio/Speech signals, industrial signals like vibration analysis or fault detection, etc.

2D CNNs: Spatial and Time-Frequency Signals: 2D CNNs process grid-like data by sliding kernels across height and width. Beyond standard images, they are often applied to signals that have been converted into 2D representations. In architectures, standard deep frameworks like Res-Net or LeNet-5 are common. The key applications are; Spectrogram Analysis, remote sensing, healthcare, etc. (Krizhevsky, 2012)

3D CNNs: Volumetric and Spatiotemporal Signals: 3D CNNs use cubic kernels (width × height × depth) to capture spatial dependencies across multiple slices or time steps simultaneously. The architecture is computationally intensive due to the extra dimension. Modern 2025 approaches often utilize "Integrated CNNs" (3D-2D-1D hybrids) to strike a balance between accuracy and efficiency, reducing training time by up to 60% compared to pure 3D models. The key applications are; Medical imaging, video analytics, autonomous systems, etc. (Ji, 2013)

Recurrent Neural Networks (Rnns), LSTM, And GRU For Temporal Signals

Recurrent Neural Networks (RNNs) and their advanced variants, LSTMs and GRUs, are specifically designed to process temporal signals where the order and timing of data points are critical. Best for modeling temporal dependencies in sequential signals, crucial for tasks like predicting remaining useful life (RUL) or understanding time-series data. LSTMs & GRUs are popular extensions that solve the vanishing gradient problem, allowing the model to capture long-term dependencies in signals like speech or vibration data. (Mienye, 2024)

Vanilla RNNs (Classic): Simple RNNs maintain a hidden state that captures information from previous time steps, acting as an internal "memory".

- **Architecture:** Uses feedback loops to pass information from one step to the next.
- **Limitation:** Highly susceptible to the vanishing gradient problem, which prevents them from learning long-term dependencies in signals.
- **Best For:** Short sequences and low-resource environments.

Long Short-Term Memory (LSTM): LSTMs solve the vanishing gradient problem by introducing a specialized cell state and a complex gating mechanism. (Chambers, 2024).

Gating Mechanism:

- **Forget Gate:** Decides which information to discard from the cell state.
- **Input Gate:** Determines which new information to store in the cell state.
- **Output Gate:** Controls what parts of the cell state are passed to the next hidden state.

Best For: Complex signals with very long-range dependencies, such as full speech sentences or long-term financial trends.

Gated Recurrent Units (GRU): GRUs are a streamlined, more efficient version of LSTMs that offer comparable performance with fewer parameters.

- **Simplified Architecture:** Merges the input and forget gates into a single update gate and uses a reset gate.
- **Performance Benefits:** Typically trains 25–40% faster and uses roughly 25% less memory than LSTMs, making them ideal for real-time edge devices.

- *Best For:* Real-time signal processing, resource-constrained IoT devices, and short-to-medium-length sequences. (Rivas, 2025)

Transformers For Long-Range Signal Dependencies

Transformers have surpassed traditional Recurrent Neural Networks (RNNs) in signal processing due to their self-attention mechanism, which explicitly computes relationships between all-time steps in a signal simultaneously. Unlike LSTMs, which can "forget" information over long sequences due to sequential processing, Transformers maintain a global view, making them ideal for identifying long-range patterns in complex signals like vibration, audio, and RF. (Thundiyil, 2025)

Key Advantages for Signal Processing

Global Contextual Awareness: Transformers do not rely on a hidden state passed step-by-step. Instead, they use multi-head attention to "see" the entire signal window at once, capturing dependencies between events that may be separated by thousands of samples.

Parallelization: Because they process the entire sequence in one forward pass, Transformers are significantly faster to train than RNNs on modern GPU hardware.

Handling Non-Stationarity: Advanced variants like Auto-former and FED-former are specifically designed to decompose complex signals into seasonal and trend components, making them more robust for long-term forecasting than standard models. (Nazari, 2025)

Standard Transformers face a "quadratic complexity" challenge, where memory usage grows exponentially with signal length. In 2025, specialized versions address this for real-time signal processing:

Informer: Uses a "ProbSparse" attention mechanism that only focuses on the most significant signal features, reducing computational cost from $O(L^2)$ to $O(L\log L)$.

Auto-former: Replaces standard self-attention with an Auto-Correlation block, utilizing the inherent periodicity of signals (via Fast Fourier Transforms) to find dependencies more efficiently. (Wu,2021).

Autoencoders and Latent Signal Representations

Autoencoders (AEs) are critical in signal processing for their ability to compress high-dimensional raw signals into a low-dimensional latent representation (or "bottleneck"). This latent space captures the most essential, underlying features of a signal while discarding noise and redundancies. Acts as a "proxy" for the signal's core parameters. For instance, in MRI signal evolution, a single latent variable can represent complex tissue relaxation properties as effectively as multiple linear coefficients. The decoders reconstruct the original signal from the latent vector. In 2025, decoders are increasingly integrated directly into forward models for tasks like high-speed MRI reconstruction. (Ahmadi, 2025)

Specialised Autoencoder Variants are;

- **Denoising Autoencoders (DAE):** Trained to recover clean signals from inputs corrupted by synthetic or real-world noise (e.g., ambient underwater noise or sensor interference). Recent 2025 research shows DAEs can improve signal-to-noise ratios (SNR) in sonobuoy systems by effectively "modulating" data into more secure, low-bit-rate latent vectors.
- **Variational Autoencoders (VAE):** Unlike standard AEs, VAEs learn a probabilistic latent space (mean and variance). This allows them to generate new, synthetic signal samples that mirror the distribution of real-world data, which is useful for data augmentation in medical studies or simulating wireless channel effects. (Liu, 2025).
- **Convolutional Autoencoders (CAE):** Use 1D or 2D convolutional layers to extract local temporal or spectral features, making them highly effective for compressing complex biomedical signals like EMG or ECG.

Generative Adversarial Networks (GANs)

GANs are a transformative class of deep learning architectures in signal processing, primarily used for data augmentation and signal enhancement. They consist of two competing neural networks: a **generator** that creates synthetic signals and a **discriminator** that attempts to distinguish them from real-world data. (Chakraborty, 2024).

Common Architectures for Signals:

- **1D-DGANs:** Specifically tailored 1-dimensional denoising GANs for temporal data like vibration sensors or heart rhythms.
- **WGAN-GP:** Utilizes Wasserstein distance with a gradient penalty to ensure more stable training and avoid "mode collapse," a common issue where the generator produces a limited variety of signals.
- **Conditional GANs (cGANs):** The most common variant in 2025; they use additional information (like class labels or clean signal templates) to guide and control the generation process.
- **SynSigGAN:** An emerging architecture in 2025 specifically designed for biomedical signal synthesis (EEG, PPG, EMG).

Key Performance Benefits (2025 Benchmarks):

- **Accuracy Improvement:** GAN-driven denoising has been shown to reduce Mean Squared Error (MSE) by over 30% in industrial sensor data compared to traditional filters.
- **Realism:** In medical imaging Turing tests, GAN-generated signals often prove indistinguishable from real data, allowing them to effectively supplement training sets and improve diagnostic sensitivity by nearly 10%.

Table 3. Comparative Summary of Deep Learning Architectures

Architecture	**Primary Signal Type**	**Temporal Reach**	**Efficiency** (Training)	**Key Advantage in 2025**	**Core Limitation**
CNN	Spatial/ Spectrogram	Short (Local)	**Very High**	Superior at local feature extraction and pattern recognition in 2D spectrograms.	Lacks inherent long-range temporal memory.
Vanilla RNN	Sequential	Short	High	Simplest design; good for short	Susceptible to vanishing/exploding gradients; poor

				sequences on low-power edge devices.	long-term memory.
LSTM	Sequential	Long	Moderate	Robustly handles long-term dependencies; mitigates gradient issues.	High computational complexity and memory usage.
GRU	Sequential	Moderate-Long	**High**	Achieves LSTM-like accuracy with ~25% fewer parameters and faster training.	May capture long-term dependencies slightly less effectively than LSTMs in very complex signals.
Transformer	Contextual/Global	**Infinite**	Low (High GPU)	Global contextual awareness via self-attention; no "forgetting" over long sequences.	Quadratic computational cost; requires massive datasets to outperform RNNs.
Autoencoder	Compressed/Latent	Variable	Moderate	Efficient signal denoising and compression into low-dimensional latent spaces.	Quality highly sensitive to the bottleneck size (latent dimension).
GAN	Synthetic/Generative	N/A	Low	SOTA for high-quality	Hardest to train due to instability and

				data augmentation and synthetic signal generation (e.g., radar, MRI).	potential mode collapse.

4. AI-EMBEDDED SIGNAL PROCESSING APPLICATIONS

AI-embedded signal processing is integrated into edge devices across many fields, enabling *real-time autonomous decision-making* with low latency and enhanced privacy. These applications process various signals, such as images, sound, and biomedical data, locally on resource-constrained hardware, including FPGAs and specialized microcontrollers.

AI in Image and Video Signal Processing

AI-powered image and video processing uses machine learning algorithms, particularly Convolutional Neural Networks (CNNs) and Vision Transformers, to enable computers to understand, interpret, and generate visual content. This technology has a wide range of applications, from medical diagnostics to autonomous vehicles. (Tian, 2025). The computer vision that gives machines the ability to "see" and interpret information from images and videos. The machine learning models are CNNs, Vision Transformers, and GANs. The key techniques include object detection, image segmentation, activity recognition, image enhancement, and video compression.

AI is used in many sectors to automate and improve visual tasks. These include:

- **Surveillance and Security:** Automating real-time video analysis for change detection and security monitoring.
- **Content Creation:** Generating new images and videos from text prompts (e.g., using tools like OpenAI's Sora or Google's Gemini) for media production and marketing.

- **Healthcare:** Improving the accuracy of medical imaging analysis and diagnostics.
- **Autonomous Systems:** Enabling self-driving cars and drones to recognise objects and navigate their environment in real-time.
- **Image and Video Restoration and Enhancement:** Current models utilise diffusion models and transformers to outperform traditional GANs in denoising and deblurring. In 2025, research is specifically targeting RAW image restoration to integrate AI earlier in the image signal processing (ISP) pipeline, handling unknown noise directly from sensor data. Improving the quality of existing media, such as upscaling low-resolution video, denoising old footage, and enhancing low-light performance. Modern architectures, such as SISR (Single-Image Super-Resolution) , utilize deep residual networks and self-attention to synthesize realistic textures rather than merely interpolating pixels. A major focus is on Real-Time SR for IoT, optimising these heavy models for deployment on low-power devices. (Chakraborty, 2024)
- Several software libraries and commercial tools facilitate AI-powered image and video processing: Open-Source Libraries like OpenCV, TensorFlow, PyTorch, Scikit-Image, and Commercial Software Topaz Video AI for video enhancement and upscaling, RunwayML for generative video creation from text, Midjourney, and Google Gemini for AI image generation.

Biomedical Signal Processing Using AI

Biomedical signal processing using Artificial Intelligence (AI) leverages machine learning (ML) and deep learning (DL) algorithms to analyse complex physiological data for enhanced diagnosis, monitoring, and personalised treatment. AI systems can optimise traditional signal processing tasks, such as noise reduction and feature extraction, and provide computer-aided diagnosis (CAD) to assist physicians. The core concepts in biomedical signal processing are as follows:

- **Signal Acquisition & Preprocessing:** Biomedical signals (e.g., ECG, EEG) are often weak, noisy, and distorted.

Traditional methods like filters (e.g., FIR, Butterworth) are used for noise reduction, but AI can employ adaptive filtering to improve signal quality based on real-time data dynamically.

- **Feature Extraction & Analysis:** AI algorithms, particularly neural networks, can automatically identify complex patterns and extract vital information from raw data that might be difficult for human experts to discern.
- **Modelling & Interpretation:** Machine learning models can be trained to classify signals, detect anomalies, and predict health outcomes, enabling proactive health monitoring and reducing the need for continuous high-power processing.

AI in biomedical signal processing has numerous applications across various medical fields:

- **Cardiology:** Analysis of electrocardiogram (ECG) signals to detect heart abnormalities and identify specific heartbeats with greater precision.
- **Neurology:** Processing of electroencephalogram (EEG) signals for diagnosing conditions like epilepsy, sleep disorders, Alzheimer's, and Parkinson's disease.
- **Rehabilitation:** Using AI to interpret EEG signals for motor imagery in stroke patients, aiding neurological rehabilitation and personalised therapy planning.
- **Patient Monitoring:** Real-time analysis of various signals (blood pressure, respiration) in intensive care units (ICUs) or through remote monitoring systems to guide treatment decisions and optimize patient care. (Alqudah, 2025)

AI for Communication and Radar Signal Processing

Artificial intelligence (AI), particularly deep learning, is revolutionizing communication and radar signal processing by providing data-driven solutions to complex challenges that exceed the capabilities of traditional model-based techniques. AI is used to address the need for robust, efficient, and adaptive systems in modern applications like autonomous vehicles, 5G networks, and electronic warfare.

- **Applications in Communication Systems:** AI-driven signal processing enhances communication systems by improving efficiency, robustness, and spectral efficiency. (Li, 2025).
- **Channel Coding and Modulation Optimization:** AI can optimize signal parameters for different dynamic transmission environments and hardware impairments, improving the reliability and efficiency of information exchange.
- **Spectrum Management**: AI helps manage the radio frequency spectrum more efficiently by identifying available bands, predicting usage, and dynamically allocating resources. Antenna design and beamforming: AI aids in designing complex antenna arrays and optimizing beamforming techniques, especially for advanced systems like 5G networks, enabling targeted and efficient signal transmission. Noise reduction and signal restoration: Machine learning models can adaptively eliminate various types of background noise, leading to much clearer and higher-fidelity signals than traditional fixed methods.
- **Applications in Radar Signal Processing:** AI is used across the entire radar signal processing chain, from raw data interpretation to target identification and resource management (Ayaz, 2025).
- **Automatic Target Recognition (ATR):** Deep learning enables improved classification, identification, and recognition of targets (e.g., in automotive radar for self-driving cars) by extracting complex patterns from radar data.
- **Imaging Techniques:** AI enhances synthetic aperture radar (SAR) and inverse SAR (ISAR) imaging, providing better resolution and more accurate mapping of target characteristics.
- **Waveform and Array Design:** AI assists in the design and optimization of radar waveforms and antenna arrays to achieve better performance in specific scenarios. Clutter and jamming suppression: Traditional anti-jamming techniques often struggle with sophisticated, deceptive active jamming. AI algorithms help in recognizing and suppressing complex jamming signals and unwanted sea/ground clutter.

5. RESEARCH FRONTIERS IN AI-BASED SIGNAL PROCESSING

Research frontiers in AI-based signal processing (SP) focus on bridging the gap between classical mathematical signal modeling and the black-box nature of deep learning to create more efficient, interpretable, and robust systems.

Generalization Challenges in Real-World Signal Environments

Generalization challenges in real-world signal environments stem primarily from the inherent variability, uncertainty, and complexity of these settings compared to controlled laboratory or simulated conditions (Rohlfs, 2024). Key issues include:

- **Sensor Noise and Artifacts:** Real-world data from sensors (e.g., physiological signals, environmental sensors) are susceptible to various types of noise, missing values, and artifacts caused by movement, device positioning, or environmental interference. Models trained on clean, simulated data often fail to perform reliably when faced with this inherent "noisiness".
- **Data Discrepancies (Sim-To-Real Gap):** A significant hurdle is the difference in data distributions between simulated environments (where models are often initially trained) and actual deployment settings. Features that are relevant in simulation may not hold the same importance in reality, leading to performance degradation.
- **Dynamic and Unpredictable Conditions**: Real-world environments are constantly changing, with dynamic factors such as varying weather, different user behaviours, or unexpected obstacles. Models must be robust enough to handle these unseen, out-of-distribution (OOD) scenarios without compromising performance.
- **Partial Observability and Hidden States:** In many real-world scenarios, the system or agent may not have access to the full state of the environment, operating instead on partial or incomplete observations. This hidden information makes it difficult for models to reason effectively and generalize their behaviour.

- **Communication Reliability and Latency:** In distributed systems, such as smart traffic networks, the reliability of communication channels can be a major issue. Poor signal strength often results in data loss or increased latency, which can severely impact the real-time performance and coordination of signal processing systems.
- **Resource Constraints:** Real-time systems deployed in edge computing environments, like IoT devices or autonomous vehicles, often have limited computational resources. Models need to be highly efficient to operate within these constraints while maintaining accuracy and low latency, a challenge for complex models.
- **Ethical and Safety Constraints:** In high-stakes domains like healthcare or autonomous systems, it can be difficult to generate data for all possible abnormal or dangerous conditions due to safety and ethical constraints. This data sparseness limits the models' exposure to critical scenarios during training, impacting their generalization to rare but important events.

Current Research Frontiers

Researchers are moving beyond pure data-driven approaches by integrating mathematical priors (e.g., physical laws or geometric constraints) with generative learning:

- **Diffusion Models for 3D Imaging:** Scaling diffusion models to 3D and multimodal imaging for medicine and astronomy.
- **Physics-Informed Neural Networks (PINNs):** These are used to ensure that signal reconstructions adhere to physical reality, reducing data requirements and computational costs.

The convergence of foundational models (like Large Language Models) and multimedia signal processing is a primary 2025 research topic:

- **Cross-Modal Data Fusion:** Developing algorithms that can simultaneously process and correlate video, audio, and sensor data (e.g., radar and LiDAR for autonomous driving).
- **Multimodal VLMs in Healthcare:** Using Vision-Language Models (VLMs) for smart healthcare to reason across medical images and textual patient records.

Deep integration between neuroscience and signal processing is leading to "Brain-in-the-Loop" technologies:

- **Closed-Loop Neuromodulation:** Real-time decoding of cortical signals for neuro-prosthetics to mitigate neurological injury.
- **Neuromorphic Engineering:** Developing AI hardware that mimics the brain's energy-efficient signal processing, suitable for wearable ECG or EEG devices.

Trustworthy, Explainable, and Responsible AI (XAI): A major frontier involves "unveiling the decision veil" to make AI signal processing transparent, especially in high-stakes fields. Current models often explain *what* they detected but not *why*. Future research must develop frameworks that provide causal reasoning for SP decisions, especially in medical diagnostics (e.g., detecting cognitive decline or heart anomalies).

Evaluation Metrics and Performance Analysis in AI-Based Signal Processing

Evaluating AI models in signal processing requires a combination of quantitative metrics (specific to the task, such as classification or regression) and performance analysis techniques (focusing on speed, resource utilization, and reliability). The choice of metrics depends heavily on the specific application and its goals (e.g., medical diagnosis vs. spam detection).

Evaluation Metrics

Metrics are generally divided by the type of machine learning task:

Classification Metrics: These metrics are used when the AI model categorizes signals, such as detecting a pattern or identifying a disease. Key metrics include (Powers, 2011):

- **Accuracy:** Overall proportion of correct predictions.
- **Precision:** Proportion of correctly classified positive instances among all predicted positives, important when false positives are costly.
- **Recall (Sensitivity):** Proportion of actual positive instances correctly identified, critical when false negatives are costly.
- **F1-Score:** A balanced measure using the harmonic mean of precision and recall, especially for imbalanced datasets.

- **Specificity:** Proportion of true negatives correctly identified.
- **AUC-ROC:** Evaluates the model's ability to distinguish between classes across thresholds.

Regression Metrics: These are applied when models predict continuous values, such as forecasting time series or estimating signal amplitudes. (Willmott, 2005) Common metrics include:

- **Mean Absolute Error (MAE):** Average magnitude of errors.
- **Mean Squared Error (MSE) / Root Mean Squared Error (RMSE):** Average of squared errors, sensitive to outliers.
- **R^2 Score:** Proportion of variance in the target variable explained by the model.

Domain-Specific Signal Processing Metrics: Some metrics are specific to signal processing (Kay, 1993) tasks:

- **Signal-to-Noise Ratio (SNR):** Measures signal clarity after processing.
- Echo Return Loss Enhancement (ERLE): Evaluates echo cancellation.
- **Peak Signal-to-Noise Ratio (PSNR):** Used for quality evaluation in image/video processing.
- **Perceptual Evaluation of Speech Quality (PESQ):** A human-centric measure for speech quality.

Performance Analysis in AI Systems

Performance analysis examines operational aspects beyond prediction accuracy (Sokolova, 2009). This includes:

- **Latency:** Time for processing input and producing output, vital for real-time systems.
- **Throughput:** Number of tasks handled per unit time.
- **Resource Utilization:** Monitoring CPU, GPU, and memory usage for efficiency.
- **Reliability & Robustness:** How well the model handles unexpected inputs or noisy conditions.
- **Scalability:** Ability to maintain performance with increased workload or data.

Effective performance analysis involves defining clear goals, using diverse datasets, combining multiple metrics for a comprehensive view, incorporating human judgment for subjective tasks, and continuous monitoring after deployment. The advanced performance analysis can be done with the help of the following metrics:

- **Closed-Loop Evaluation:** In 2025, systems use "auto-raters" (AI acting as a judge) to detect and correct errors in real-time before they cascade.
- **Robustness & Fairness:** Models are stress-tested against adversarial inputs and audited for demographic parity to ensure ethical compliance.
- **Drift Detection:** Continuous monitoring is used to identify performance degradation caused by changes in input signal patterns over time.

CONCLUSION

This chapter presents a comprehensive overview of intelligent signal processing, emphasizing the evolution from conventional, model-driven techniques to modern AI-enabled, data-driven approaches. It begins by highlighting the limitations of classical signal processing in handling non-linear, non-stationary, and high-dimensional signals, motivating the integration of machine learning and deep learning methods. Various learning paradigms—supervised, unsupervised, and self-supervised learning—are discussed alongside the trade-offs between feature-based models and end-to-end learning, with attention to the bias–variance dilemma in signal datasets. The chapter then explores key deep learning architectures, including CNNs, RNNs, transformers, autoencoders, and GANs, illustrating their suitability for spatial, temporal, and generative signal modeling. Practical AI-embedded applications in image and video processing, biomedical signals, and communication and radar systems are reviewed. Finally, the chapter addresses research frontiers, generalization challenges in real-world environments, and the role of evaluation metrics and performance analysis in assessing AI-based signal processing systems.

REFERENCES

Aburakhia, S., Shami, A., & Karagiannidis, G. K. (2024). *On the intersection of signal processing and machine learning: A use case-driven analysis approach*. arXiv. https://doi.org/10.48550/arXiv.2403.17181.

Ahmadi, S., Rahmani, A. M., Abbasi, A., & Fathy, M. (2025). Deep autoencoder neural networks: A comprehensive review and new perspectives. *Archives of Computational Methods in Engineering, 32*(5), 16871–16882. https://doi.org/10.1007/s11831-025-10260-5.

Akkaya, S. (2025). Wavelet-based denoising strategies for non-stationary signals in electrical power systems: An optimization perspective. *Electronics, 14*(16), 3190. https://doi.org/10.3390/electronics14163190

Alqudah, A. M., & Moussavi, Z. (2025). A review of deep learning for biomedical signals: Current applications, advancements, future prospects, interpretation, and challenges. *Computers, Materials & Continua, 83*(3), 3753–3841. https://doi.org/10.32604/cmc.2025.063643A.

Ayaz, F., Alhumaily, B., Hussain, S., Imran, M. A., Arshad, K., Assaleh, K., & Zoha, A. (2025). Radar signal processing and its impact on deep learning-driven human activity recognition. *Sensors, 25*(3), 724. https://doi.org/10.3390/s25030724 .

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Chakraborty, T., Reddy, U. R. K. S., Naik, S. M., Panja, M., & Manvitha, B. (2024). *Ten years of generative adversarial nets (GANs): A survey of the state-of-the-art. Machine Learning: Science and Technology, 5*(1), 011001. https://doi.org/10.1088/2632-2153/accb4a.

Chambers, J. D., Cook, M. J., Burkitt, A. N., & others. (2024). *Using long short-term memory (LSTM) recurrent neural networks to classify unprocessed EEG for seizure prediction. Frontiers in Neuroscience*. https://doi.org/10.3389/fnins.2024.1472747.

Chen, K.C. J., Peng, W.H., & Lee, C. G. G. (2023). Overview of intelligent signal processing systems. *APSIPA Transactions on Signal and Information Processing, 12*(1), e36. https://doi.org/10.1561/116.00000053

Cohen, L. (1995). *Time-frequency analysis*. Prentice Hall.

Gannot, S., Kellermann, W., Koldovsky, Z., Araki, S., & Richard, G. (2024). *Special issue on model-based and data-driven audio signal processing [From the guest editors]. IEEE Signal Processing Magazine, 41*(6), 8–11. https://doi.org/10.1109/MSP.2024.3497727

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*(1), 1–58. https://doi.org/10.1162/neco.1992.4.1.1.

Ji, S., Xu, W., Yang, M., & Yu, K. (2013). 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 35*(1), 221–231. https://doi.org/10.1109/TPAMI.2012.59.

Kay, S. M. (1993). *Fundamentals of statistical signal processing: Estimation theory*. Prentice Hall.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., & Mahoor, M. (2021). 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing, 151*, 107398. https://doi.org/10.1016/j.ymssp.2020.107398.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097–1105.

Li, X., & Others. (2025). Signal processing technology of mobile communication systems based on artificial intelligence. *Procedia Computer Science, 261*, 1284–1290. https://doi.org/10.1016/j.procs.2025.05.003.

Liu, C., Chen, Y., & Tao, D. (2025). Variational and adversarial autoencoders: Latent signal representation and generation. *Pattern Recognition, 145*, 109291. https://doi.org/10.1016/j.patcog.2025.109291.

Mienye, I. D., Swart, T. G., & Obaido, G. (2024). *Recurrent neural networks: A comprehensive review of architectures, variants, and applications. Information (Switzerland), 15*(9), 517. https://doi.org/10.3390/info15090517.

Nazari, M., & Cui, W. (2025). *Transformers in speech processing: Overcoming challenges and paving the future. Computer Science Review, 58*, 100768. https://doi.org/10.1016/j.cosrev.2025.100768.

Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies, 2*(1), 37–63.

Proakis, J. G., & Manolakis, D. G. (2006). *Digital signal processing: Principles, algorithms, and applications* (4th ed.). Prentice Hall.

Razzaq, K., & Shah, M. (2025). *Machine learning and deep learning paradigms: From techniques to practical applications and research frontiers. Computers, 14*(3), 93. https://doi.org/10.3390/computers14030093

Rivas, F., Sierra-Garcia, J. E., & Camara, J. M. (2025). *Comparison of LSTM- and GRU-type RNN networks for attention and meditation prediction on raw EEG data from low-cost headsets. Electronics, 14*(4), 707. https://doi.org/10.3390/electronics14040707.

Rohlfs, C. (2024). *Generalization in neural networks: A broad survey. Neurocomputing, 611*, 128701. https://doi.org/10.1016/j.neucom.2024.128701.

Sejdić, E., Djurović, I., & Jiang, J. (2009). Time-frequency feature representation using energy concentration: An overview of recent advances. *Digital Signal Processing, 19*(1), 153–183. https://doi.org/10.1016/j.dsp.2008.03.004

Shlezinger, N., & Eldar, Y. C. (2023). *Model-based deep learning*. arXiv. https://arxiv.org/abs/2306.04469.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management, 45*(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002.

Thundiyil, S., Shalamzari, S. S., Picone, J., & McKenzie, S. (2025). *Transformers for modeling long-term dependencies in time series data: A review. Proceedings of IEEE SPMB Conference.*

Tian, C., Cheng, T., Peng, Z., Zuo, W., Zhang, Q., Wang, F.-Y., & Zhang, D. (2025). A survey on deep learning fundamentals and applications across image, video, and vision tasks. Artificial Intelligence Review, 58, 381–412. https://doi.org/10.1007/s10462-025-11368-7.

Wang, B., Jiang, Z., Sun, Y., et al. (2023). An intelligent signal processing method against impulsive noise interference in A IoT. *EURASIP Journal on Advances in Signal Processing, 2023*, 104. https://doi.org/10.1186/s13634-023-01061-8

Wang, Y., Li, Y., & Ding, Z. (2020). Deep learning in signal processing: A survey. *IEEE Signal Processing Magazine, 37*(4), 36–51. https://doi.org/10.1109/MSP.2020.2987025

Willmott, C. J., & Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research, 30*(1), 79–82. https://doi.org/10.3354/cr030079.

Wu, H., Xu, J., Wang, J., & Long, M. (2021). *Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting*. arXiv. https://arxiv.org/abs/2106.13008.

ISBN: 978-625-93102-8-2