Automatic Code and Interleaver Classification Technique for Future Generation Communication Systems

M.Tech. Thesis

By
Naveen B

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JUNE 2023

Automatic Code and Interleaver Classification Technique for Future Generation Communication Systems

A THESIS

Submitted in partial fulfillment of the

requirements for the award of the degree

of

Master of Technology

by

Naveen B

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2023

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Automatic Code and Interleaver Classification Technique for Future Generation Communication Systems** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from August 2021 to June 2023 under the supervision of **Dr. Swaminathan R, Assistant Professor, Electrical Engineering.**

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

14 June, 2023
Signature of the student with date

Naveen B

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

14/06/2023
Signature of the Supervisor of
M.Tech. thesis #1 (with date)
(NAME OF SUPERVISOR)

Signature of the Supervisor of
M.Tech. thesis #2 (with date)
(NAME OF SUPERVISOR)

Naveen B has successfully given his/her M.Tech. Oral Examination held on **16 May, 2023**.

Signature(s) of Supervisor(s) of M.Tech. thesis
Date: 14/06/2023

Convener, DPGC
Date: 14/06/2023

Dr. Ayan Mondal, CSE
Signature of PSPC Member #1
Date: 15/06/23

Dr. Appina, EE
Signature of PSPC Member #1
Date: 15/06/2023

ACKNOWLEDGEMENTS

I am deeply grateful to Professor Dr. Swaminathan R for his exceptional guidance and unwavering support throughout the journey of completing my M.Tech project and thesis. His expertise, patience, and insightful feedback have been invaluable in shaping the outcome of this research. I am indebted to him for his dedication and encouragement, which have inspired me to push the boundaries of my knowledge and capabilities. Thank you, Professor Dr. Swaminathan R, for being an exceptional mentor and for your invaluable contribution to my academic growth.

Naveen B

M.Tech in Communication and Signal Processing

Discipline of Electrical Engineering

IIT Indore

Abstract

Error correcting codes (ECCs) and interleavers play a vital role in ensuring reliable and efficient communication over noisy channels. The process of manually selecting appropriate ECCs and interleavers for a given communication system can be time-consuming and error-prone. In this thesis, we propose an automated approach to identify suitable ECCs and interleavers using deep learning techniques.

We begin by exploring various ECCs commonly used in digital communication systems, including block codes, convolutional codes, Low-Density Parity-Check (LDPC) codes, BCH codes, and Hamming codes. Each ECC exhibits different error correction capabilities, decoding complexity, and performance characteristics. By training a deep learning model on a diverse dataset of encoded and noisy communication signals, we aim to develop a system that can accurately classify and recommend the most suitable ECC for a given scenario.

Additionally, we investigate different types of interleavers, such as block interleavers, convolutional interleavers, and helical interleavers. Interleavers aid in spreading burst errors, reducing error propagation, and improving overall error correction performance. Leveraging the power of deep learning, we seek to build a model that can automatically determine the optimal interleaver based on the specific requirements of a communication system.

The proposed deep learning framework utilizes various techniques, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to learn the inherent features and patterns present in encoded communication signals. Through extensive training and validation, the model aims to generalize its knowledge and accurately predict the most appropriate ECC and interleaver combination for a given set of input parameters, such as channel conditions, transmission rate, and error correction requirements.

Overall, this thesis presents an innovative framework for automatically identifying error correcting codes and interleavers using deep learning techniques. The proposed system has the potential to enhance the efficiency and accuracy of ECC and interleaver selection in digital communication systems, leading to improved error correction capabilities and optimized system performance in the presence of channel noise and interference.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Forward Error Correcting Codes (FECs)

Forward Error Correcting Codes (FECs) are crucial for error detection and correction in digital communication systems. They add redundancy to transmitted data, enabling the receiver to detect and correct errors without retransmission. FECs encode data with extra bits, known as parity bits, which contain error-correction information. By efficiently detecting and correcting errors, FECs ensure reliable data transmission in scenarios where retransmission is costly or impractical.

Different FEC techniques exist, including Hamming codes, Reed-Solomon codes, convolutional codes, and Turbo codes. Hamming codes are simple and efficient for single-bit error detection and correction. Reed-Solomon codes handle burst errors and correct multiple errors, making them suitable for storage devices and digital communication. Convolutional codes enable real-time decoding in wireless communication, while Turbo codes offer excellent error correction with low decoding complexity. The choice of FEC technique depends on factors such as the data type, error characteristics of the communication channel, and desired error correction performance. Overall, FECs are vital for reliable data transmission in modern communication systems.

1.2 Block Codes

Block codes are error correcting codes that operate on fixed-sized blocks of data. The input data is divided into equal-sized blocks, and encoding and decoding operations are applied independently to each block. Redundancy is introduced by appending parity bits to each block, which are calculated based on the original data using mathematical algorithms. At the receiver, the parity bits are used to detect and correct errors, ensuring data integrity. Block codes are commonly used in various applications, including data storage and digital communication systems, due to their simplicity and effectiveness in error detection and correction. They provide a reliable means of data transmission and enhance system robustness against transmission errors.

1.3 Convolutional Codes

Convolutional codes are continuous data stream error correcting codes. Unlike block codes, they use a sliding window approach for encoding and decoding. By convolving input data with shift registers, they generate encoded symbols without distinct block boundaries. Convolutional codes excel at correcting burst errors in wireless communication and satellite transmission. They enable real-time decoding, making them

suitable for low-latency applications. They find extensive use in digital television broadcasting, wireless networks, and data storage systems, providing reliable error detection and correction.

Convolutional codes efficiently correct errors in continuous data streams. With their sliding window approach and trellis diagram-based decoding, they estimate likely transmitted symbols and correct errors. They handle burst errors and offer real-time decoding, making them vital for communication systems. Convolutional codes enhance the robustness and performance of wireless communication, satellite transmission, and digital television broadcasting.

1.4 BCH Codes

BCH (Bose-Chaudhuri-Hocquenghem) codes are widely used for error correction in data transmission and storage. They are generated by selecting the code length, message length, and error correction capability. Decoding involves calculating the syndrome and using algebraic techniques to detect and correct errors. BCH codes find applications in telecommunications, storage devices, and data transmission, providing robust error detection and correction.

1.5 Low Density Parity Check Codes

LDPC (Low-Density Parity-Check) codes are powerful error-correcting codes used in communication systems and data storage. They encode information bits into longer codewords using a sparse parity-check matrix, allowing for efficient transmission and storage. LDPC codes have capacity-approaching performance, meaning they come close to the theoretical limits of channel capacity. They are widely employed in various communication standards, providing reliable and robust error correction for wireless networks, digital video broadcasting, and optical communication systems.

1.6 Hamming Codes

Hamming codes are efficient error correcting codes that add redundant bits to the original data for single-bit error detection and correction. The generation involves calculating parity checks to determine the positions and values of the redundant bits. During decoding, the received code word is compared with the generated parity bits to identify and correct single-bit errors. Hamming codes find applications in computer memory systems and data transmission, providing reliable error detection and correction capabilities to ensure data integrity.

1.7 Interleaver

An interleaver rearranges data bits or symbols before transmission to combat burst errors. It spreads out consecutive bits or symbols, improving error correction and handling in digital communication systems.

Interleaving mitigates fading effects and enhances transmission reliability, commonly used in wireless communication and satellite transmission.

1.7.1 Block Interleaver

A block interleaver is a type of interleaver used in digital communication systems. It rearranges data bits or symbols within fixed-sized blocks before transmission. The blocks are typically of equal size and the order of the blocks is changed according to a specific algorithm. This introduces a delay and helps combat burst errors, as errors are spread out across different blocks.

1.7.2 Convolutional Interleaver

A convolutional interleaver rearranges data bits or symbols using a sliding window approach. By convolving the input data with predefined shift registers, the interleaver reshuffles the symbols to disperse consecutive symbols over time. This helps mitigate the effects of burst errors in digital communication systems.

1.7.3 Helical Interleaver

A helical interleaver rearranges data to spread consecutive bits or symbols over time, reducing the impact of burst errors. It improves error correction by spreading errors across the data stream.

1.8 Deep Leaning

Deep learning is a subfield of machine learning that focuses on training artificial neural networks with multiple layers to perform complex tasks. It involves the use of deep neural networks, which are designed to automatically learn hierarchical representations of data. Deep learning models can learn directly from raw input data without the need for manual feature engineering. Through a process called training, these networks learn to recognize patterns, make predictions, and classify or generate new data. Deep learning has achieved impressive results in various domains, including image and speech recognition, natural language processing, and many other areas of artificial intelligence.

1.8.1 Neural Networks

Deep learning is based on artificial neural networks, which are composed of interconnected nodes (neurons) organized in layers. Neural networks are responsible for learning patterns and relationships in data.

1.8.2 Deep Neural Networks (DNNs)

Deep learning utilizes deep neural networks that consist of multiple hidden layers between the input and output layers. The depth of the network allows for the learning of complex representations and features.

1.8.3 Training

Deep learning models are trained using a large amount of labeled data. During training, the model adjusts the weights and biases of the neurons through an optimization process, such as backpropagation, to minimize the difference between predicted and actual outputs.

1.8.4 Activation Functions

Activation functions introduce non-linearities into the neural network, allowing it to learn complex relationships. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), and tanh (hyperbolic tangent).

1.8.5 Loss Functions

Loss functions quantify the difference between predicted and actual outputs. They are used to measure the model's performance during training and guide the optimization process.

1.8.6 Gradient Descent

Gradient descent is an optimization algorithm used to update the weights and biases of neural network parameters during training. It calculates the gradient of the loss function with respect to the parameters and adjusts them in the direction that minimizes the loss.

1.8.7 Deep Learning Architectures

Various deep learning architectures have been developed, such as convolutional neural networks (CNNs) for image processing, recurrent neural networks (RNNs) for sequential data, and generative adversarial networks (GANs) for generating new data.

1.8.8 Transfer Learning

Transfer learning is a technique where pre-trained models are used as a starting point for new tasks. By leveraging the knowledge gained from one task, the model can be fine-tuned or adapted to another related task with less data.

1.8.9 GPU Acceleration

Deep learning models often require significant computational power for training due to their complexity. Graphics Processing Units (GPUs) are commonly used to accelerate the computations, as they are highly parallel and can handle the massive calculations involved.

1.9 Literature Survey

1.9.1 Literature Survey on Rank Based Blind Estimation of Codes

In this comprehensive literature survey, several researchers have made significant contributions to the field of blind estimation and reconstruction techniques [1-7] for error correcting codes under challenging channel conditions. R. Swaminathan, A. S. Madhukumar, and G. Wang [1, 6] have proposed blind estimation methods for product codes and Reed-Solomon encoders in noisy channel environments. Dinesh and S. R. [2, 5, 7] have presented techniques for blind reconstruction of BCH encoders and codeword length estimation of LDPC codes. The work by Swaminathan and Madhukumar [3, 4] focuses on blind parameter estimation of turbo convolutional codes. In papers [8, 9], the block interleaver and convolutional helical scan parameters of the interleaver are estimated in the presence of error bits.

1.9.2 Literature Survey on Blind Estimation of Codes using Deep Learning

The field of deep learning has been significantly influenced by several groundbreaking papers. [10] introduced AlexNet, a pioneering architecture that showcased the power of deep convolutional neural networks (CNNs) in image classification tasks. Simonyan and Zisserman's paper [11] proposed the VGGNet architecture, emphasizing the importance of increased network depth and smaller filter sizes to achieve remarkable performance on image classification benchmarks. He et al.'s [12] introduced ResNet, which employed residual connections to address the vanishing gradient problem and enabled the successful training of extremely deep neural networks. Another significant contribution was made by Ioffe and Szegedy in their paper [13] on "Batch Normalization," which presented a technique to accelerate training by normalizing layer inputs and reducing internal covariate shift. Additionally, Szegedy et al.'s paper [14] on "Going Deeper with Convolutions" introduced the Inception architecture, which utilized parallel and multi-scale convolutions to improve the representational capacity of deep networks.

In [15], A. Shrestha and A. Mahmood present a comprehensive review of deep learning algorithms and architectures, covering CNNs, RNNs, and DBNs. [16] by Akay, Karaboga, and Akay focuses on optimizing deep learning models using metaheuristics. They explore various metaheuristic algorithms and their applications. [17] by Smys, Chen, and Shakya surveys neural network architectures in deep learning,

including feedforward networks, CNNs, and RNNs. These papers provide valuable insights for researchers and practitioners in the field.

Several papers have explored the application of deep learning techniques for blind recognition and identification of channel codes. Gautam and Lall proposed a method in their paper [18] that employs neural networks to identify convolutional and Reed-Solomon encoders. Dehdashtian et al., in [19] presented a deep learning-based approach for blind recognition of channel code parameters under AWGN and multi-path fading conditions. Wang et al., in their paper [20] introduced a method that utilizes convolutional neural networks (CNNs) for joint demodulation and error correcting code recognition. Li et al., [21] proposed a CNN-based method that incorporates a block mechanism and embedding for error correcting code recognition. Huang et al., in [22] presented a method that combines bidirectional long short-term memory (BiLSTM) and CNN for channel code recognition. These papers collectively contribute to the advancement of deep learning techniques in the field of channel code recognition.

1.10 Contributions

- In Chapter 2 of our study, we focused on classifying error correcting codes. These codes are used to detect and correct errors in transmitted data. We examined different types of error correcting codes, including block codes, convolutional codes, and Hamming codes. To perform the classification, we employed deep learning techniques.
- Moving on to Chapter 3, we shifted our attention to the classification of interleavers. Interleavers are important components in communication systems that rearrange the order of data for improved error resilience. In this chapter, we used deep learning methods to classify different types of interleavers.
- Chapter 4 was dedicated to a joint classification of both interleavers and error correcting codes. We explored how these two components work together in communication systems and developed a deep learning-based approach to classify them simultaneously.

Chapter 2

FEC Codes Classification

2.1 Introduction

In our research, we explored the integration of forward error correcting (FEC) codes, specifically block codes and convolutional codes, with deep learning techniques for automatic classification. To facilitate our study, we utilized a dataset consisting of directly received noisy signals at the receiver. These signals were generated using MATLAB simulations, allowing us to closely replicate the challenges encountered in real-world signal transmission and reception scenarios.

Initially, our classification task involved eight classes, with four classes belonging to block codes and four classes belonging to convolutional codes. However, as our research progressed, we expanded the scope and increased the number of classes to a total of 15. Out of these, seven classes were block codes, while the remaining eight classes were convolutional codes. This expansion allowed us to explore a wider range of error correcting codes and evaluate their performance in the classification process.

In Fig. 2.1, we can see the flow. First, we demodulated the signals to extract the underlying information. Then, we used a deep learning model to identify the type of error-correcting code used in the transmission. This code type was then mapped to the specific parameters of the code. Finally, with the help of these parameters, we successfully decoded the received message, recovering the original information.

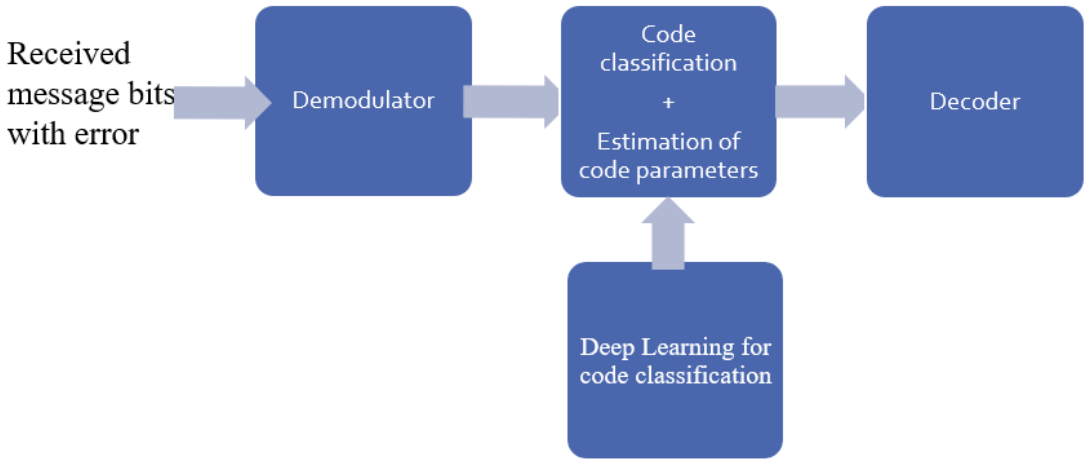

Figure 2.1 Flow of decoding of received signal.

2.2 Convolutional Neural Network Architecture

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision in deep learning. Specifically designed to process visual data such as images and videos, CNNs have gained significant popularity for their ability to automatically learn and extract complex visual features from raw pixel inputs. However, their applicability extends beyond visual data, as CNNs can also be effectively employed for processing one-dimensional data.

While CNNs are commonly associated with image analysis, they can be adapted to handle various other types of data, including one-dimensional sequences. For example, in natural language processing tasks, CNNs can be used to process text data represented as sequences of words or characters. In these cases, the input to the CNN is a one-dimensional sequence instead of a two-dimensional image.

In one-dimensional CNNs, the convolutional layers operate by sliding filters over the input sequence and performing localized operations, similar to their counterparts in image-based CNNs. The filters capture relevant patterns and features within the sequence, allowing the network to learn hierarchical representations and capture complex information. By incorporating pooling layers, the network can effectively summarize the learned features while reducing the spatial dimensions of the sequence.

2.2.1 Proposed Architecture

The given architecture which is shown in Table 2.1, is a 1D Convolutional Neural Network (CNN) designed for processing sequence data. It employs Conv1D layers with different filter sizes and strides to extract features from the input sequence. Max Pooling layers are used to downsample the feature maps while preserving important information. The architecture also includes Global Average Pooling to obtain a condensed representation of the features. Dense layers with ReLU activation are used to learn complex relationships within the extracted features. Dropout layers are included to prevent overfitting. The final Dense layer with Softmax activation produces the output probabilities for the 17 possible classes.

The architecture utilizes Conv1D layers with filter sizes of 11, 5, and 3, and strides of 1/4, 1/1, and 1/1, respectively. Max Pooling layers with window sizes of 3 and strides of 1/2 are employed to downsample the feature maps. Global Average Pooling condenses the features into a vector representation. Dense layers with ReLU activation function learn intricate relationships within the features. Dropout layers with a rate of 0.5 help prevent overfitting. The final Dense layer with Softmax activation produces the output probabilities for the 17 classes, enabling the network to make predictions.

Table 2.1 Architecture details

Layer	Filter Size/Stride	Output Size
Input	16384 × 1	16384 × 1
Conv1D + ReLU	11 × 1/4	4094 × 96
Max Pooling	3 × 1/2	2046 × 96
Conv1D + ReLU	5 × 1/1	2046 × 128
Max Pooling	3 × 1/2	1022 × 128
Conv1D + ReLU	3 × 1/1	1022 × 192
Conv1D + ReLU	3 × 1/1	1022 × 192
Conv1D + ReLU	3 × 1/1	1022 × 128
Max Pooling	3 × 1/2	510 × 128
Global Average Pooling	-	128
Dense + ReLU	-	128
Dropout (0.5)	-	128
Dense + ReLU	-	64
Dropout (0.5)	-	64
Dense + Softmax	-	17

The given architecture is trained using the Adam optimizer with a learning rate of 0.001. The loss function employed is sparse categorical entropy, suitable for multi-class classification tasks. The model is trained using a batch size of 32, which determines the number of samples processed in each training iteration. This configuration ensures efficient optimization and accurate predictions for the given architecture.

2.3 Experimental Classification of Eight Classes Using Deep Learning: Four Block Codes and Four Convolutional Codes

Table 2.2 provides information about the parameters of block codes and convolutional codes. Each parameter in these codes is assigned a specific number for classification. With the help architecture proposed in Section 2.2.1, these codes are been classified.

2.3.1 Dataset Generation

In MATLAB, we generated a random binary signal. We then applied a specific error correcting code to encode this message. The encoded message was further modulated using the BPSK modulation scheme. To simulate a realistic scenario, we introduced noise to the modulated signal by specifying a Signal-to-Noise Ratio (SNR) value. Afterward, we demodulated the noisy signal, resulting in a simulated received message.

To create a dataset for a particular error correcting code, we conducted 500 iterations of this process for each signal-to-noise ratio (SNR) value, ranging from -10dB to 20dB with intervals of 5dB. This allowed us to generate multiple sets of the message, each corresponding to a different SNR value. Additionally, we also created a set without any noise. The total number of samples in the set is 8000.

We repeated the described process for each error correcting code mentioned in Table 2.2. The size of the noisy message determined the sample size or input dimensions. We chose different input sizes to compare and determine the optimal input dimension for achieving better results.
Starting with a message size of 512 bits, we increased the input size gradually. We considered sizes of 1024 bits, 2048 bits, 4096 bits, 8192 bits, and finally 16384 bits. In total, we generated six datasets, each corresponding to a specific input size. Each is saved in a csv file.

Table 2.2 Error Correcting Codes Parameters

Block Codes (n, k)	**Convolutional Codes** (n, k, m)
8,5	2,1,4
7,4	2,1,7
6,3	3,1,5
3,2	4,1,6

2.3.2 Results and Observation

In this experiment, we generated a CNN model for each dataset and split the data into training and testing sets using a 0.25 ratio. The number of epochs for training the models varied based on observed accuracy, ranging from 20 to 80. We calculated the accuracy for each SNR value across all the models and plotted the results in Figure 2.2. This figure provides a visual representation of how the accuracy of the CNN models changes with different levels of SNR, allowing us to analyze and compare their performance. It offers valuable insights into the models' ability to handle noise and highlights the relationship between accuracy and SNR in the experimental setup.

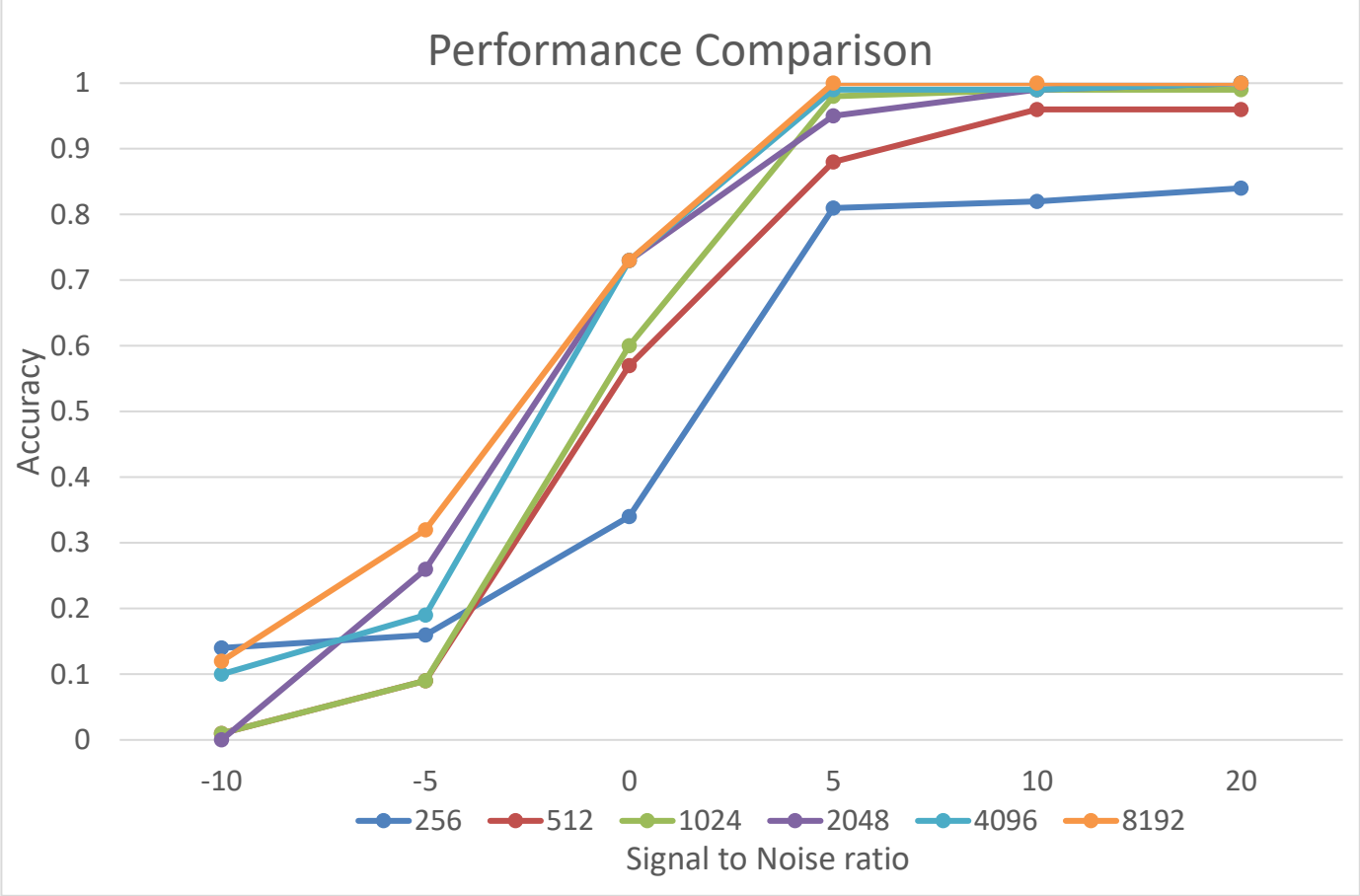

Figure 2.2 Result of experimental Classification of Eight Classes Using Deep Learning

The observations from the experiment reveal interesting trends. At 0 dB, it is observed that higher sample sizes result in better accuracy. As the sample length increases to 2048 bits, an accuracy of 95% is achieved at an SNR of 5 dB. Beyond an SNR of 10 dB, sample lengths of 1024 bits or more consistently yield accuracy levels surpassing 95%.

These observations indicate that as the SNR increases, the required sample length for achieving high accuracy decreases. Initially, longer sample lengths are necessary to achieve acceptable accuracy levels at lower SNR values. However, as the SNR improves, shorter sample lengths are sufficient to attain accuracy levels of over 95%. These findings provide valuable insights into the relationship between sample size, SNR, and accuracy in the context of the experiment.

2.4 Experimental Classification of Fifteen Classes Using Deep Learning: Seven Block Codes and Eight Convolutional Codes

Table 2.3 presents the parameters employed for fifteen class error correcting codes. In this experiment, we expanded the number of classes to 15, including an additional seven classes compared to the

ones described in Section 2.3. The objective was to investigate the performance of the deep learning model with an increased number of classes while using the same architecture.

In this particular dataset, we utilized the Bit Error Rate (BER) as the metric instead of the Signal-to-Noise Ratio (SNR). The BER values were varied within the range of 0.01 to 0.1, with an interval of 0.1. By altering the BER, we sought to evaluate the model's capability to handle different error rates and assess its performance across the varying levels of bit errors.

This extension of the experiment with additional classes and the incorporation of the BER as a metric provides a comprehensive analysis of the deep learning model's effectiveness. It enables us to examine its performance across a wider range of error rates and ascertain its suitability for error correction tasks involving multiple classes.

Table 2.3 Error Correcting Codes Parameters

Block Codes (n,k)	**Convolutional Codes (n,k,m)**
8,5	2,1,4
7,4	2,1,7
6,3	3,1,5
3,2	4,1,6
8,2	9,1,2
8,4	7,1,3
7,3	4,1,3
	5,1,4

2.4.1 Results and Observation

In the training data for each class, we included 10 sets of Bit Error Rate (BER) values. Additionally, one set without any noise was included as a non-noisy reference. This resulted in a total of 5500 samples for each class. Since we considered 15 classes in this experiment, the dataset comprised a total of 82,500 samples.

Figure 2.3 displays the results obtained from this experiment, providing a visual representation of the outcomes. The figure showcases the performance and accuracy of the deep learning model across the

different classes and BER values. The results depicted in Figure 2.3 enable us to analyse and compare the model's performance in handling various error rates and provide insights into its effectiveness for error correction tasks involving multiple classes.

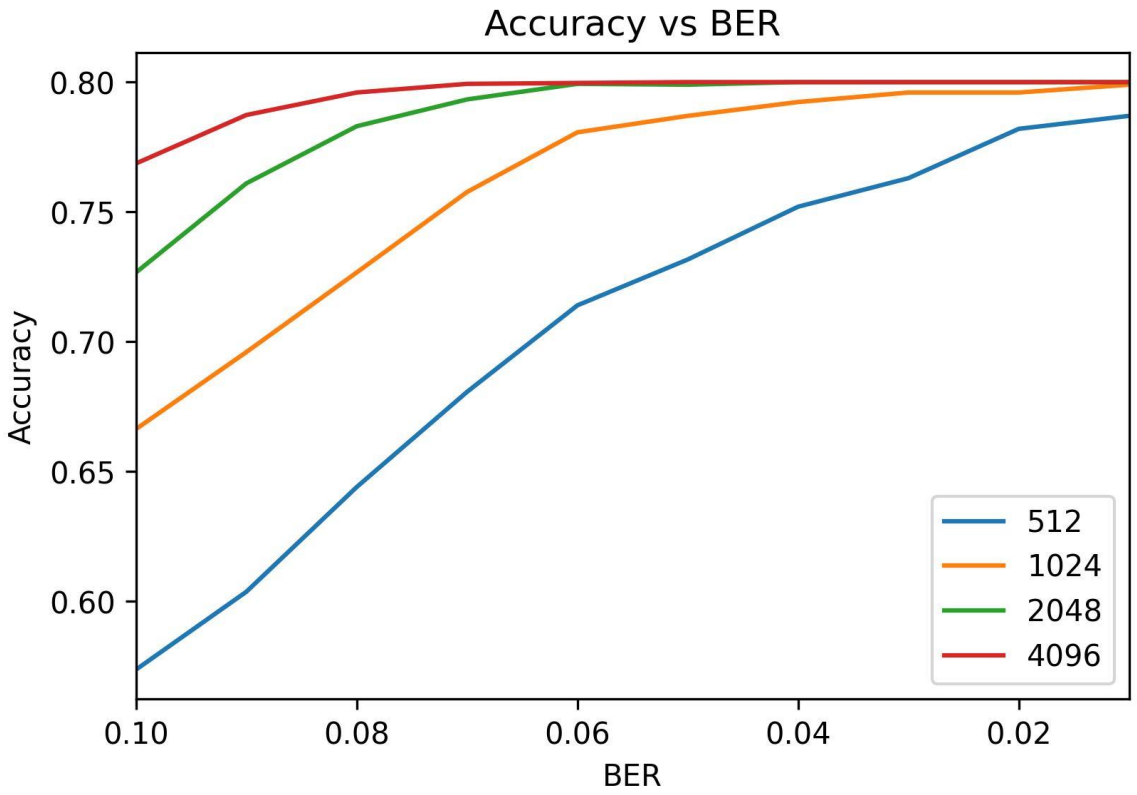

Figure 2.3 Performance Comparison of both Convolutional Codes and Block Codes

The observations from the experiment indicate that a sample length of 1024 achieves over 80% accuracy for a Bit Error Rate (BER) of 0.01. This suggests that the deep learning model performs well in accurately correcting errors when the BER is at a low level.

Furthermore, for a sample length of 2048, a 75% accuracy level is achieved for a BER of 0.05. This demonstrates that the model is still capable of achieving a reasonable level of accuracy even when the error rate is slightly higher.

2.5 Experimental Classification Using Syndrome

The concept behind this idea is that when a noisy message is decoded using the appropriate error correcting code decoder, the resulting syndrome data contains more zeros if the message has fewer errors. The goal is to determine if using the syndrome data instead of the direct noisy data can lead to better performance. In this scenario, a dataset consisting of syndrome data is being used to train a deep learning model.

2.5.1 Dataset Generation

The dataset used for training the deep learning model is based on Hamming codes, which is a specific type of error correcting code belonging to the block codes category. The parameters utilized in generating the dataset are presented in Table 2.4.

The first dataset is generated following a similar approach as discussed in section 2.3.1. For each sample in the dataset, four different syndromes are created using the decoder parameters specified in Table 2.4. These four syndromes are then concatenated together to form a single sample.

Table 2.3 Error Correcting Codes Parameters

Hamming Codes (n,k)
8,5
7,4
6,3
3,2

Figure 2.5 provides a visual representation of the dataset generation process, illustrating the various steps involved, including the generation of syndromes and the concatenation of these syndromes for each sample.

Both the syndrome-based dataset and the noisy message-based dataset are saved in a CSV file format, allowing for easy storage, access, and utilization during the training of the deep learning model.

Figure 2.4 Flow of syndrome based code estimation.

2.5.2 Results and Conclusion

A deep learning model was trained using both noisy message and syndrome datasets to assess their respective performance in error correction. By training on the noisy message dataset, the model aimed to directly decode messages, while training on the syndrome dataset allowed it to learn decoding through syndromes. The model's performance is presented in

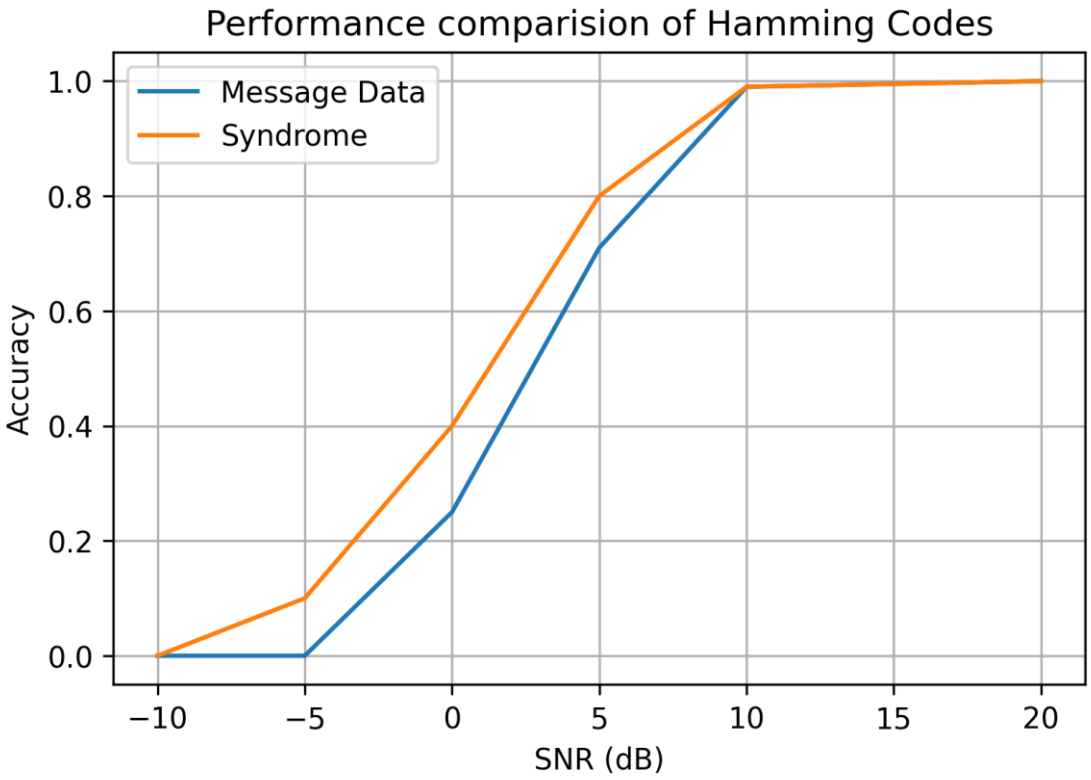

Figure 2.6 Performance Comparison Syndrome and Message

The syndrome-based approach demonstrated superior performance in the range of -5dB to 5dB, exhibiting significant improvements in accuracy.

2.6 Conclusion

In an 8-class classification task, a syndrome-based approach exhibited excellent accuracy. However, when the number of classes was increased, a decrease in accuracy was observed, dropping from 95% to 85%. Nonetheless, training the model with syndrome data instead of message data resulted in a notable improvement of 5% to 10% in accuracy.

Chapter 3

Interleaver Classification

3.1 Introduction

The task at hand involves the classification of interleavers using deep learning techniques. Interleavers can be categorized into different types, such as block interleavers, convolutional interleavers, and helical interleavers. The objective is to develop a deep learning model capable of accurately identifying and classifying these interleaver types.

The model will be trained on a dataset consisting of interleaver samples, where each sample is labeled with its respective interleaver type. Through training, the deep learning model will learn the underlying patterns and characteristics that distinguish block interleavers, convolutional interleavers, and helical interleavers. The ultimate goal is to create a reliable and efficient classifier that can accurately identify the interleaver type given an input sample, contributing to the advancement of communication systems and error correction coding techniques.

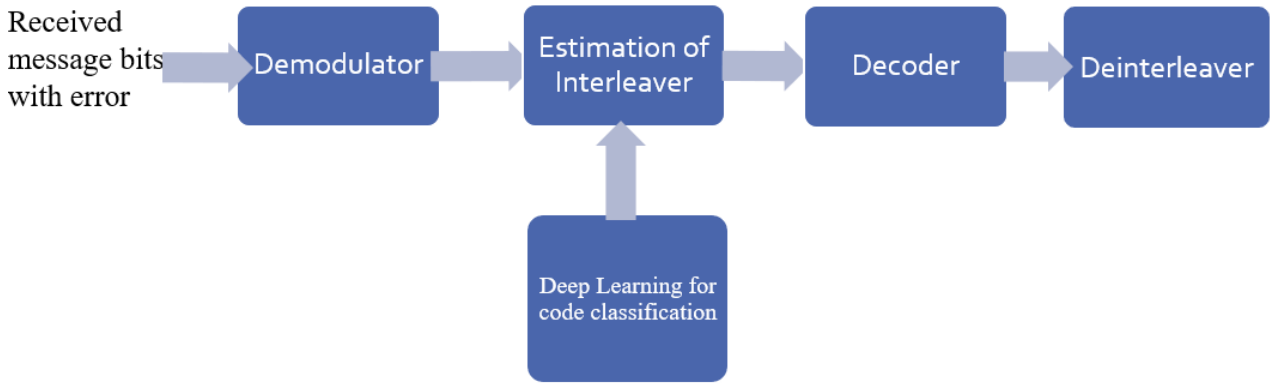

Figure 3.1 Flow of Interleaver Estimation

3.2 Dataset Generation

The dataset generation flow involves message generation, encoding, interleaving, modulation, noise addition, and demodulation, as depicted in Figure 3.1. Four different types of encoders (block codes, convolutional codes, BCH codes, and LDPC codes) are used, with each encoder generating a separate dataset.

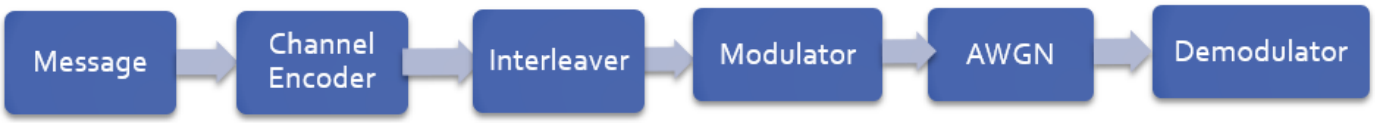

Figure 3.2 Dataset Generation for Interleaver

Table 3.1 presents the parameters used to generate the dataset, which includes a total of 12 different interleavers. Each interleaver type consists of 4 different sizes, and for each size, 500 sets are generated. To simulate real-world conditions, noise is added in terms of signal-to-noise ratio (SNR), ranging from -10dB to 20dB with intervals of 5dB, including a non-noisy case. With 8000 samples generated per size, the total number of samples amounts to 96,000.

Table 3.1 Interleaver parameters for Dataset

Interleaver Type	**Size**
Block Interleaver: Matrix $(m \times n)$	(3,2), (4,2) (3,3) (4,3)
Convolutional Interleaver: (inter depth, step)	(6,3), (6,4), (8,4), (8,5)
Helical Interleaver: (column, step, ngrp)	(5,3,3), (5,4,3), (7,4,3), (7,5,3)

3.3 Results and Observation

Simulation results are taken with Block codes, Convolutional codes, BCH codes and LDPC codes encoder.

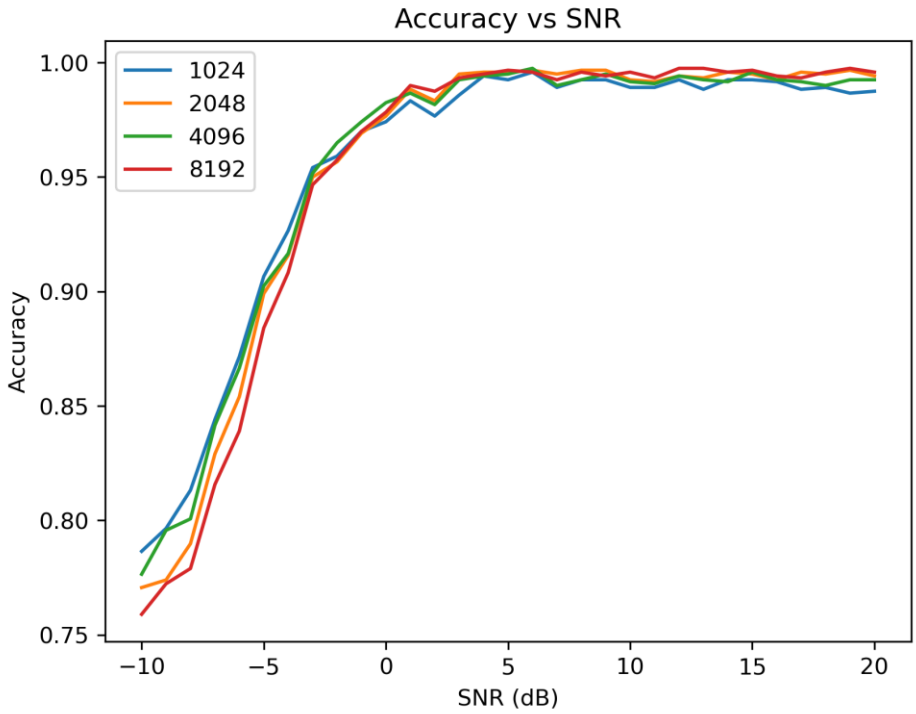

Figure 3.2 Performance comparison of Interleaver with Block encoder

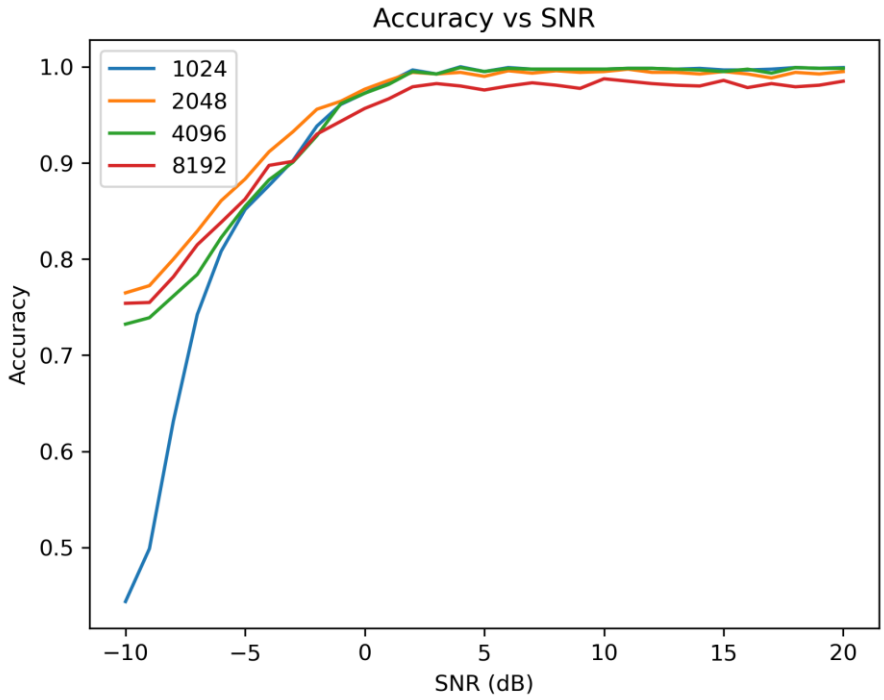

Figure 3.3 Performance comparison of Interleaver with Convolutional encoder

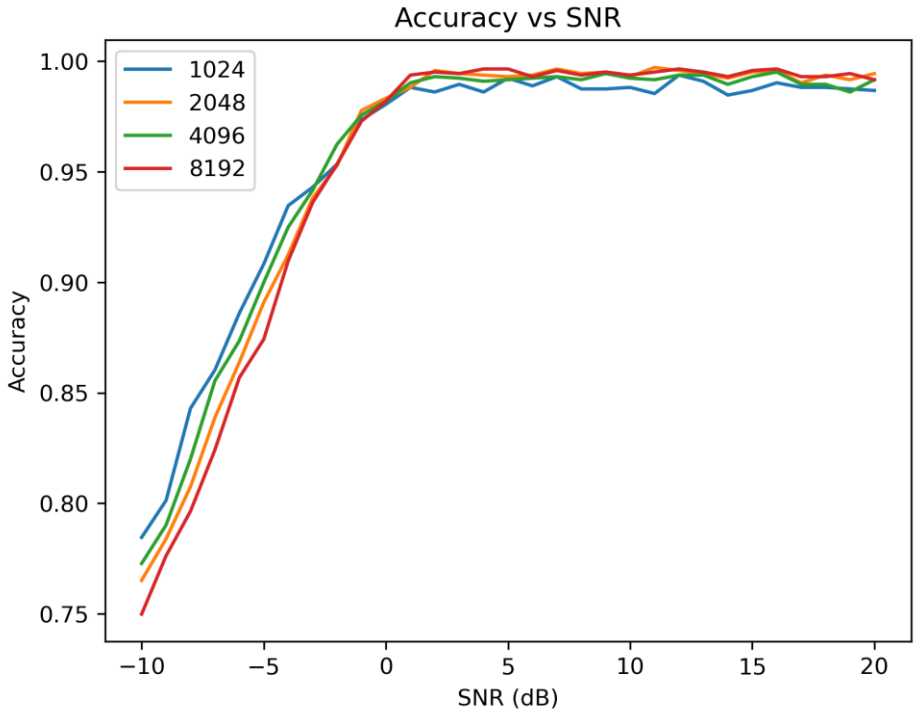

Figure 3.4 Performance comparison of Interleaver with BCH encoder

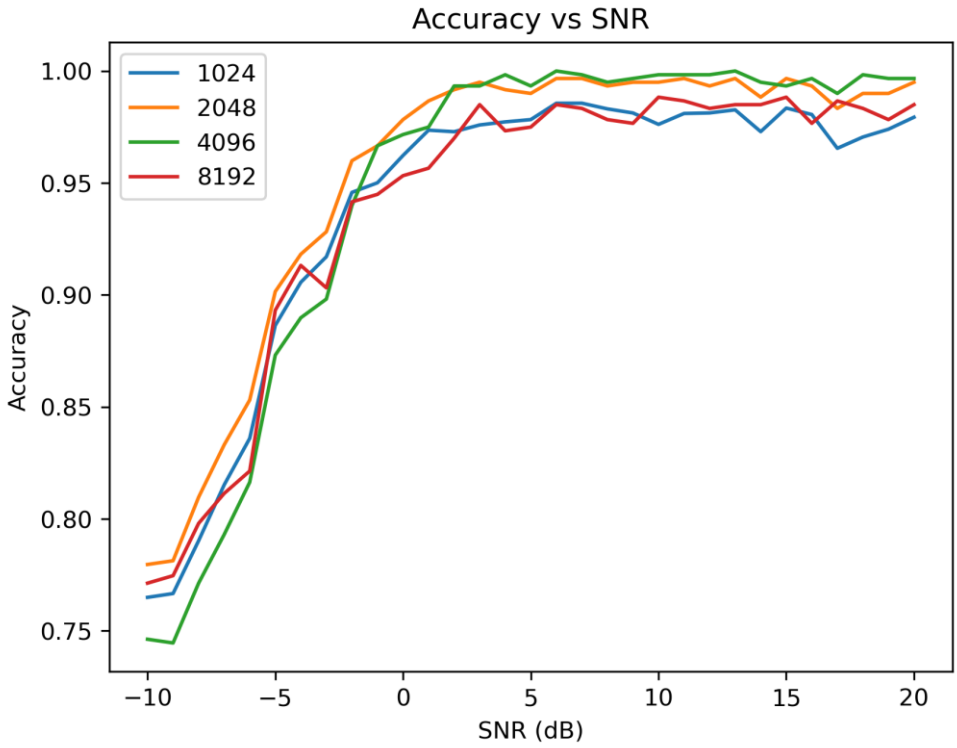

Figure 3.5 Performance comparison of Interleaver with LDPC encoder

3.4 Conclusion

The performance of interleaver identification is robust when using different encoders such as Block codes, Convolutional codes, BCH codes, and LDPC codes. The accuracy of identification exceeds 90% after a signal-to-noise ratio (SNR) of 5dB. This indicates that the trained model demonstrates high accuracy in classifying interleavers across various encoding schemes, even in the presence of noise.

Chapter 4

Joint Code and Interleaver Classification

4.1 Introduction

In the joint code classification and interleaver estimation process, we begin with data consisting of interleaved and encoded messages. In the initial stage, we employ deep learning to classify the error correcting code. Once we have made predictions about the parameters of the error correcting code, we utilize a pre-trained deep learning model to identify the interleaver. This trained model assists us in accurately determining the interleaver configuration based on the predicted error correcting code parameters.

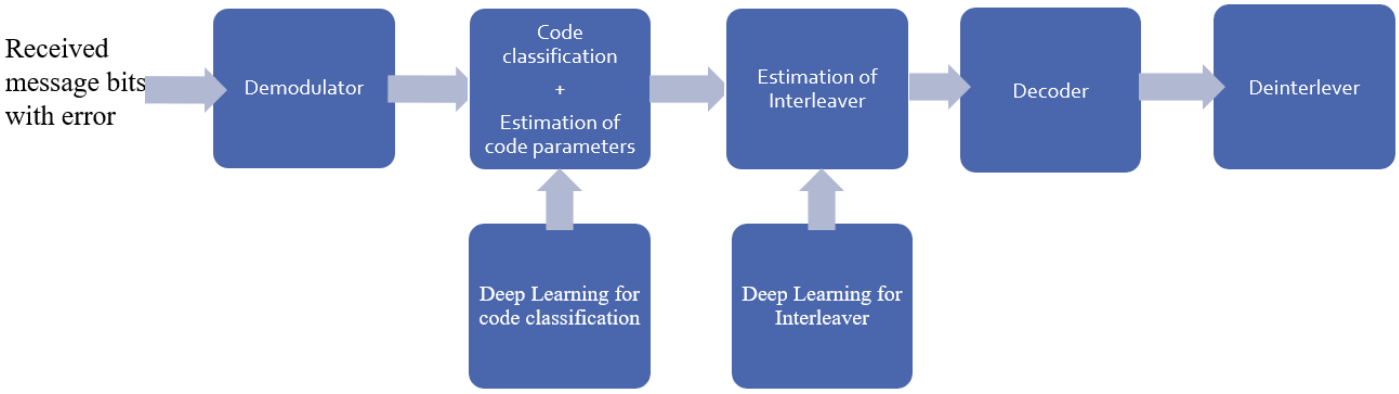

Figure 4.1 Flow of Joint Code Classification and Interleaver Estimation

4.2 Dataset

The dataset is generated following the same procedure described in section 2.3.1, as mentioned for the specific codes listed in Table 4.1. The generated dataset is then interleaved using the parameters provided in Table 3.1. The dataset initially consists of 200 samples. After applying 12 interleavers, the dataset expands to 2400 samples. For the code classification task, a total of 16 classes are considered, while for the interleaver estimation task, 3 classes are taken into account.

Table 4.1 Error Correcting Code Parameters for dataset

Error Correcting Code type	**Parameters**
Block Codes: (n, k)	(8,5), (7,4), (6,3), (8,4)
Convolutional Codes: (n, k)	(2,1,4), (2,1,7), (3,1,5), (4,1,6)
BCH codes: (n, k)	(15,7), (31,21), (31,16), (63,36)
LDPC Codes: (n, r)	(1/4, 648), (1/4, 1296), (1/3 648), (1/3, 1296)

4.3 Results and Observations

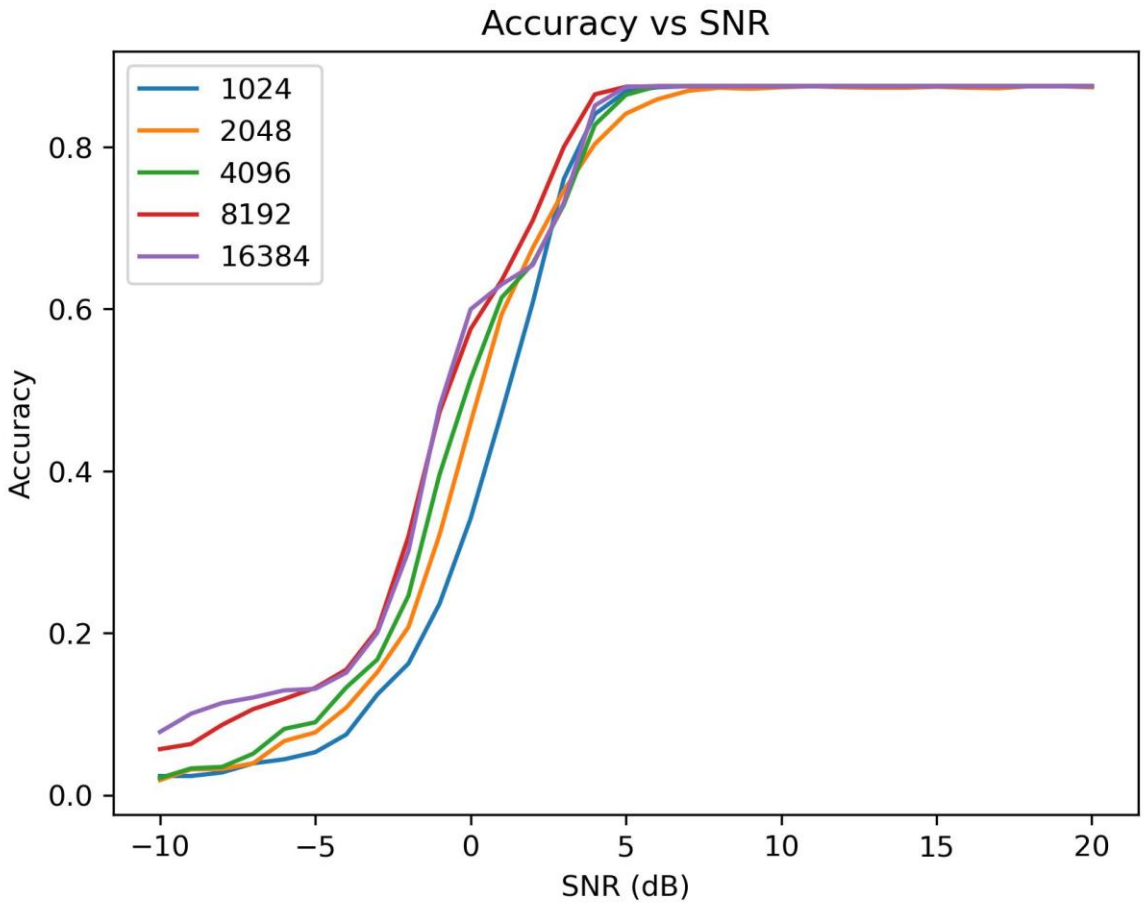

Figure 4.2 Result of Joint Code Classification and Interleaver Estimation

4.4 Conclusion

The presence of an interleaver does not significantly affect the performance of code classification. Even with interleaving, the accuracy remains above 75% after a signal-to-noise ratio (SNR) of 5dB. As the SNR increases to 10dB, the accuracy further improves, surpassing 90%. This suggests that the classification model is robust and capable of accurately identifying the underlying code, regardless of the interleaving applied.

Chapter 5

Conclusion

Initially, our focus was on estimating block and convolutional codes using deep learning. For training the models, we used received noise data as the dataset. Initially, we started with 8 classes, which were later increased to 15 classes. We achieved an accuracy of 95% initially. However, when we increased the number of classes, the accuracy dropped to 85%.

To improve the accuracy, we also experimented with using syndrome data instead of message data. This led to a modest improvement of 5 to 10% in some signal-to-noise ratio (SNR) levels.

Next, we extended our code classification task to include Block interleaver, Convolutional interleaver, and helical interleavers. The accuracy of code estimation was observed to be over 95% for various encoded codes such as Block codes, Convolutional codes, BCH codes, and LDPC codes.

Finally, we performed code classification using interleaved data and then estimated the interleaver using a separate deep learning model. The results of the joint code classification and interleaver estimation showed similar accuracy as the earlier code classification, i.e., 85%. This indicates that the presence of interleaver in the data does not significantly affect the accuracy.

In conclusion, while interleaver presence does not appear to have a substantial impact on accuracy, the accuracy of code classification can be affected when increasing the number of classes.

REFERENCES

[1] R. Swaminathan, A. S. Madhukumar, and G. Wang, "Blind Estimation of Code Parameters for Product Codes Over Noisy Channel Conditions," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 2, pp. 1460-1473, April 2020. doi: 10.1109/TAES.2019.2934308.

[2] Dinesh and S. R, "Blind Reconstruction of BCH Encoder over Erroneous Channel Conditions," in *2022 National Conference on Communications (NCC)*, Mumbai, India, 2022, pp. 262-267. doi: 10.1109/NCC55593.2022.9806814.

[3] R. Swaminathan and A. S. Madhukumar, "Blind parameter estimation of turbo convolutional codes: Noisy and non-synchronized scenario," in *Digital Signal Processing*, vol. 95, p. 102577, 2019.

[4] R. Swaminathan, A. S. Madhukumar, G. Wang, and T. Shang Kee, "Blind Reconstruction of Reed-Solomon Encoder and Interleavers Over Noisy Environment," in *IEEE Transactions on Broadcasting*, vol. 64, no. 4, pp. 830-845, Dec. 2018. doi: 10.1109/TBC.2018.2795461.

[5] Dinesh and S. R, "Codeword Length Estimation of LDPC Codes with Limited Data," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, Bangalore, India, 2022, pp. 270-274. doi: 10.1109/COMSNETS53615.2022.9668582.

[6] R. Swaminathan, A. S. Madhu Kumar, G. Wang, and S. K. Ting, "Parameter Identification of Reed-Solomon Codes over Noisy Environment," in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Toronto, ON, Canada, 2017, pp. 1-5. doi: 10.1109/VTCFall.2017.8287916.

[7] Dinesh and S. R, "Blind Reconstruction of BCH Encoder over Erroneous Channel Conditions," in *2022 National Conference on Communications (NCC)*, Mumbai, India, 2022, pp. 262-267. doi: 10.1109/NCC55593.2022.9806814.

[8] R. Swaminathan and A. S. Madhukumar, "Classification of Error Correcting Codes and Estimation of Interleaver Parameters in a Noisy Transmission Environment," in *IEEE Transactions on Broadcasting*, vol. 63, no. 3, pp. 463-478, Sept. 2017. doi: 10.1109/TBC.2017.2704436.

[9] R. Swaminathan, A. S. Madhukumar, W. T. Ng, and C. M. S. See, "Parameter estimation of block and helical scan interleavers in the presence of bit errors," in *Digital Signal Processing*, vol. 60, pp. 20-32, 2017.

[10] Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[12] K. He et al., "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pmlr, 2015.

[14] Szegedy et al., "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[15] Shrestha and A. Mahmood, "Review of Deep Learning Algorithms and Architectures," in *IEEE Access*, vol. 7, pp. 53040-53065, 2019. doi: 10.1109/ACCESS.2019.2912200.

[16] Akay, D. Karaboga, and R. Akay, "A comprehensive survey on optimizing deep learning models by metaheuristics," in *Artificial Intelligence Review*, 2022.

[17] S. Smys, J. I. Z. Chen, and S. Shakya, "Survey on neural network architectures with deep learning," in *Journal of Soft Computing Paradigm (JSCP)*, vol. 2, no. 03, pp. 186-194, 2020.

[18] N. Gautam and B. Lall, "Blind Channel Coding Identification of Convolutional encoder and Reed-Solomon encoder using Neural Networks," in *2020 National Conference on Communications (NCC)*, Kharagpur, India, 2020, pp. 1-6. doi: 10.1109/NCC48643.2020.9056082.

[19] S. Dehdashtian, M. Hashemi, and S. Salehkaleybar, "Deep-Learning-Based Blind Recognition of Channel Code Parameters Over Candidate Sets Under AWGN and Multi-Path Fading Conditions," in *IEEE Wireless Communications Letters*, vol. 10, no. 5, pp. 1041-1045, May 2021. doi: 10.1109/LWC.2021.3056631.

[20] J. Wang, H. Huang, J. Liu, and J. Li, "Joint Demodulation and Error Correcting Codes Recognition Using Convolutional Neural Network," in *IEEE Access*, vol. 10, pp. 104844-104851, 2022. doi: 10.1109/ACCESS.2022.3201354.

[21] S. Li, et al., "Recognition of error correcting codes based on CNN with block mechanism and embedding," *Digital Signal Processing*, vol. 111, pp. 102986, 2021.

[22] X. Huang, S. Sun, X. Yang, and S. Peng, "Recognition of Channel Codes based on BiLSTM-CNN," in *2022 31st Wireless and Optical Communications Conference (WOCC)*, Shenzhen, China, 2022, pp. 151-154, doi: 10.1109/WOCC55104.2022.9880573.

[23] J. A. Smith and L. M. Johnson, "A Survey of Block Coding Techniques for Error Control in Communication Systems," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1024-1049, 2018, doi: 10.1109/COMST.2018.2824402.

[24] D. P. Anderson and M. G. Martinez, "Convolutional Codes: An Overview of Encoders and Decoders," *IEEE Transactions on Communications*, vol. 63, no. 12, pp. 4320-4337, 2015, doi: 10.1109/TCOMM.2015.2416191.

[25] S. Lin and D. J. Costello Jr., "Error Control Coding: Fundamentals and Applications," *Proceedings of the IEEE*, vol. 71, no. 7, pp. 982-1002, Jul. 1983, doi: 10.1109/PROC.1983.12508.

[26] S. Reed and X. S. Chen, "Error-Control Coding for Data Networks," *IEEE Transactions on Communications*, vol. 47, no. 9, pp. 1260-1270, Sep. 1999, doi: 10.1109/26.790603.

[27] R. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21-28, Jan. 1962, doi: 10.1109/TIT.1962.1057683.

[28] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160, Apr. 1950, doi: 10.1002/j.1538-7305.1950.tb00463.x.

[29] S. Lin, "Design of Convolutional Interleavers Based on Low-Density Parity-Check Codes," *IEEE Communications Letters*, vol. 9, no. 1, pp. 17-19, Jan. 2005, doi: 10.1109/LCOMM.2005.01006.

[30] J. F. Chang and C. K. Shieh, "New Interleavers for Block Turbo Codes," *IEEE Communications Letters*, vol. 5, no. 2, pp. 52-54, Feb. 2001, doi: 10.1109/4234.905951.