2 Deep Neural Networks for Joint Source-Channel Coding

David Burth Kurka, Milind Rao, Nariman Farsad, Deniz Gündüz, and Andrea Goldsmith

2.1 Introduction

Digital communication systems typically entail separate steps for source coding and channel coding. In source coding, the source signal, e.g., text, image, or video, is mapped to a sequence of symbols that compresses the data for efficient transmission while guaranteeing a certain reconstruction quality. This involves stripping the original source signal of any redundancies. In channel coding, redundant symbols are systematically added to the compressed source sequence prior to transmission. These redundant symbols enable the receiver to detect or correct any errors that may be introduced during the transmission of data over the channel. Surprisingly, Shannon showed that this two-step approach to data communication is optimal for ergodic sources and channels when infinite blocklength codes are allowed [1]. Known as Shannon’s separation theorem, this has been extended to a larger class of source, channel, and network scenarios [2–4].

Optimality of separation in Shannon’s theorem assumes no constraint on the complexity of the source and channel code design. However, in practice, having large blocklengths may not be possible due to computational complexity as well as delay constraints. Thus, in challenging communication scenarios with more stringent power or latency constraints, or in the presence of multiple users or rapidly changing channels, the limitations of this separation-based approach become more apparent, significantly limiting the performance with respect to its fundamental information theoretic limit [5–7]. The alternative is to design the mapping from the source signal directly to the channel input, which is called joint source-channel coding (JSCC). There have been significant research efforts on JSCC over the years; however, these have focused either on the theoretical analysis under some idealistic source and channel distributions, e.g., [8–13], or on the joint optimization of the component parameters (vector quantizer, index assignment, channel code, and modulator) of an inherently separate design, e.g., [14–21]. Nonetheless, despite the suboptimality of separate source and channel coding in many practical settings, the lack of powerful JSCC techniques with reasonable coding and decoding complexities has prevented the emergence of alternatives to the modular separation-based approach.

In this chapter, we show that deep neural networks (DNNs) can be used to design JSCC solutions with impressive results. We illustrate the potential of DNN-based JSCC through concrete examples for various source and channel distributions.
In particular, we show that DNN-based JSCC schemes (a) achieve performance comparable or superior to state-of-the-art separation-based schemes, (b) provide graceful degradation upon deterioration of channel conditions, (c) have the versatility to adapt to different channels and source domains, (d) allow successive refinement with almost no performance loss, (e) exploit channel output feedback, and (f) support variable-length encoding.

The rest of this chapter is organized as follows. First, we review separate source and channel coding in Section 2.2. We then present how DNNs have been used for compression and channel coding in Section 2.3. A neural network–based JSCC for the transmission of text or natural language over discrete channels is presented in Section 2.4, and Section 2.5 deals with transmission of images over continuous channels. Section 2.6 concludes the chapter.

2.2 Source and Channel Coding

Consider a source signal \( x \in \mathbb{R}^n \) to be transmitted over \( k \) uses of a noisy channel, where \( k/n \) is denoted as the bandwidth ratio. Conventional wireless point-to-point communication systems follow a modular design approach (see Fig. 2.1(a)), consisting of two steps: a source encoder followed by a channel encoder. The source encoder \( f_s: \mathbb{R}^n \to \mathbb{B}^m, \mathbb{B} = \{0, 1\} \) maps \( x \) into as few bits as possible, while the source decoder \( g_s: \mathbb{B}^m \to \mathbb{R}^n \) reconstructs the original source signal from the compressed bits. When designing the source encoder and decoder, the goal is to minimize \( m \) by compressing the source signal while allowing for reconstruction of the original source within the allowed distortion under a prescribed distortion measure.

Let \( b \in \mathbb{B}^m \) be the compressed bits. The channel encoder \( f_c: \mathbb{B}^m \to \mathbb{Z}^k \) maps the compressed bit sequence into a sequence of symbols transmitted over the channel, where \( \mathbb{Z} \) denotes the channel input alphabet. In principle, the channel encoder introduces structured redundancy to correct any errors that may be introduced during transmission over the channel. Typically, the set \( \mathbb{Z} \) is finite in digital communication systems (i.e., the channel input is discrete). However, in this work we also consider continuous-input channels, where \( \mathbb{Z} \) is the set of complex numbers, \( \mathbb{C} \). The channel introduces errors and distortion, and the received symbol sequence at the receiver is denoted by \( \hat{z} \in \mathbb{Z} \). The channel decoder \( g_c: \mathbb{Z}^k \to \mathbb{B}^m \) estimates the original \( b \), potentially correcting the errors that are introduced during transmission. When designing the channel encoder and decoder, the goal is to use the smallest \( k \), or add the fewest number of redundant symbols, while guaranteeing reliable communication of the bit sequence \( b \).

Alternatively, in JSCC, the source \( x \in \mathbb{R}^n \) is directly mapped to the channel input vector using a JSCC encoder \( f_\theta: \mathbb{R}^n \to \mathbb{Z}^k \). Similarly, the channel output is directly mapped to an estimate of the source using a JSCC decoder \( g_\phi: \mathbb{Z}^k \to \mathbb{R}^n \) (see Fig. 2.1(b)).

Practical wireless communication systems today almost exclusively rely on a separate design of the source and channel codes. The separate design provides modularity;
that is, the design and optimization of the source and channel components can be carried out independently, which simplifies the design process as the source and channel coding problems individually are much simpler to design and are better understood. Moreover, we have highly specialized source codes for different types of information sources, e.g., JPEG2000/ BPG for images, MPEG-4/ WMA for audio, or H.264 for video, which have been engineered by domain experts over many decades and many generations of standards. There are also universal source encoders such as gzip, which are designed to compress any type of data. Similarly, highly optimized channel coding techniques have been developed for additive white Gaussian noise (AWGN) channels, such as turbo, low-density parity-check (LDPC), and polar codes.

Shannon’s Separation Theorem establishes the theoretical optimality of the separate design in the asymptotic infinite blocklength regime [22]. However, as we move toward less conventional communication paradigms, we are reaching the limits of this separate design. Particularly, for machine-type communications within the Internet of Things (IoT), this modular approach is increasingly limited in meeting the stringent transmission power and latency constraints of the devices and the underlying applications. For example, compression delay is currently the main bottleneck in ultra-low-latency communications, which is essential for many emerging applications such as virtual reality or tactile Internet. Moreover, due to the overcrowding of the wireless spectrum, communications increasingly take place over more challenging

Figure 2.1 The components of (a) separation-based and (b) JSCC communication systems. In separation-based systems, each component is optimized independently. In JSCC, the input source is transformed directly into channel inputs that are transmitted and restored in a single step.
environments that do not follow traditional channel models. Fading and interference can have more degrading effects on the transmission compared to AWGN. Existing coding techniques perform poorly in such channel environments, and adapting their design to these complex channel statistics is extremely challenging, if not impossible.

Even within the scope of existing channel models, separation-based schemes are extremely sensitive to the channel parameters and can suffer severely when the channel conditions differ from those for which the codes have been optimized. For example, if the signal-to-noise ratio (SNR) in an AWGN channel is worse than the one for which the channel code rate is chosen, the error probability increases rapidly, provoking errors in both the channel and the source decoders, which can compromise the reconstruction quality significantly. Additionally, because the source and channel code rates are fixed, the reconstruction quality remains the same regardless of how much the channel SNR improves. These two characteristics are known as the “cliff effect” in digital communications. This also has implications when broadcasting to multiple receivers: those with worse channel conditions than the one targeted by the channel code are not able to reconstruct the source, while those with much better channel conditions do not obtain a better source reconstruction as a result.

In recent years, DNNs have been employed to improve both the source coding and the channel coding components of the conventional digital communication systems. In addition, the aforementioned limitations of the existing separate source and channel coding approach have motivated the use of deep learning (DL) to solve the JSCC problem. Before summarizing the proposed solution approaches in the next section, we will first give a brief overview of how DNNs have been used for source compression and channel coding.

### 2.3 DL-Based Source and Channel Coding

The design of source and channel codes has traditionally relied on human ingenuity. Several decades of intensive research have resulted in capacity achieving codes for the AWGN channel, such as turbo codes, LDPC codes, and polar codes. In parallel, numerous standards have been developed for specialized compression techniques for different types of information sources, e.g., JPEG2000 for images, as well as H.264 and H.265 for videos. However, with the recent developments in DL, a new data-driven approach for source and channel coding individually as well as for JSCC has emerged, as shown in Fig. 2.2. In this new paradigm, the encoders and decoders are replaced by DNNs, which are trained directly from data for source and channel coding, as we now describe in more detail.

- **DL for source coding**: Data-driven methods, such as principal component analysis (PCA), have long been used for dimensionality reduction and feature extraction. More recently, neural networks paired to form an autoencoder network have been shown to provide much better performance for dimensionality reduction and feature extraction [23]. This naturally led to employing autoencoders for source
compression by incorporating a quantization layer as shown in Fig. 2.2(a). In this architecture, the neural network encoder and decoder are jointly trained as autoencoders together with the quantization component, using sample source data (e.g., image, audio, or video samples).

- **DL for channel coding**: For a fixed channel encoder, the decoding operation at the receiver is a standard classification problem, where the received noisy channel output is classified into one of the input channel codewords. Many recent works have shown that DNN-based decoders can outperform existing conventional decoder architectures, e.g., the belief propagation decoder [24, 25]. On the other hand, both the encoder and the decoder can also be replaced by DNNs, which results in an autoencoder architecture similar to source compression. However, in the case of channel coding, we need to introduce the modulation and the channel as an untrainable neural network layer between the encoder and the decoder. This is illustrated in Fig. 2.2(b). Hence, as opposed to source coding, where the goal of the autoencoder is to learn the most efficient representation of the source sequence, the goal here is to learn channel codewords that can be recovered reliably despite channel impairments.

- **DL for JSCC**: Neural networks can also be used to design the mapping from the source signals directly to the channel inputs and the reconstruction of the source signal directly from the noisy channel output, i.e., \( f_\theta \) and \( g_\phi \) in Fig. 2.1(b), jointly. This method is similar to the channel code design: it incorporates the channel as an untrainable layer in the autoencoder architecture but uses sample source data for training the network. This architecture is illustrated in Fig. 2.2(c).

Next, we discuss each of these approaches in more detail.

### 2.3.1 Source Coding Using DL

Lossy source coding using DL has primarily relied on an autoencoder-type architecture. In this architecture, an encoder neural network compresses the input into a lower-dimensional embedding and a decoder reconstructs the input from this lower-dimensional embedding. For lossy source coding, the lower-dimensional embedding needs to be quantized. This process is shown in Fig. 2.2(a).
For end-to-end training of lossy source coding methods using DL, prior works have typically relied on the optimization of a rate-distortion loss given by
\[
\mathcal{L} = -\log_2 P(\lceil f_s(x) \rceil) + \lambda \ d(x, g_s(\lceil f_s(x) \rceil)),
\]
where \( P \) is a probability mass function that depends on the network parameters, distortion \( d(\cdot, \cdot) \) measures the discrepancy between the input signal and its reconstruction at the decoder, and the parameter \( \lambda > 0 \) controls the trade-off between the distortion and the number of bits. Here \( \lceil \cdot \rceil \) is the rounding function used for quantization.

Since quantization is not a smooth function, it can suppress the backpropagation of gradients during training. Several methods have been proposed to deal with this challenge. In [26], a stochastic binarization method is proposed for quantization. This stochastic binarization was used in [27] along with recurrent neural networks to achieve a reconstruction performance that outperforms BPG, WebP, JPEG2000, and JPEG. Later in [28], stochastic binarization was generalized to stochastic rounding to the nearest integer. In this method, during the forward pass, the following random variable is used:
\[
b = \lceil f_s(x) \rceil + \epsilon, \quad \epsilon \in \{0, 1\}, \quad P(\epsilon = 1) = f_s(x) - \lceil f_s(x) \rceil,
\]
where \( \lceil \cdot \rceil \) is the floor operation. In the backward pass, the derivative is replaced with the derivative of the expectation, which is equal to 1, and hence, the derivative passes through the quantization layer unchanged.

A smooth approximation of vector quantization that was annealed toward hard quantization during training is used in [29]. Another approach in [30] adds uniform noise during training as an approximation to rounding at test time. In this scheme, the loss in Eq. (2.1) becomes
\[
\mathcal{L} = -\log_2 p(f_s(x) + u) + \lambda \ d(x, g_s(f_s(x) + u)),
\]
where \( p \) is now a probability density function, and \( u \) is a random vector with independent and identically distributed (i.i.d.) elements drawn from a uniform distribution \( u_i \sim U(-0.5, 0.5) \). If the distortion measure is the mean-squared error, then this approach is equivalent to a variational autoencoder [31] with a uniform encoder. Later in [32–34], the spatial dependencies that might exist in the source (e.g., spacial correlations between pixels in an image) were further exploited by transmitting such information as side information to the decoder, which is in parallel to what is being used by conventional compression methods to improve performance. These DNN-based methods surpass conventional compression standards in various performance metrics such as peak SNR (PSNR) and also in terms of subjective perceptual quality [35].

### 2.3.2 Channel Coding Using DL

Channel codes can also be designed using autoencoders or variational autoencoders. In this approach, the output of the encoder neural network is passed on to a modulator and transmitted over the channel. The channel output is then passed to a decoder,
which estimates the input bits, \( b \), from the noisy channel output. The network can be trained end-to-end to learn new channel codes, but to allow joint training of the encoder and decoder neural networks, the modulation scheme used and the stochastic channel both must be modeled or approximated as neural network layers. Using this approach, the system learns a reversible transformation from the input data to a latent space, and then from noisy observations of the latent space back to the estimate of the original input data. This process is illustrated in Fig. 2.2(b).

The design of the encoder and decoders using autoencoders was first proposed in [36, 37] for short blocklength codes. For such codes it was shown that the DL-based channel code design can achieve the same or better performance compared to Hamming codes when used over AWGN channels. These results were then extended to OFDM channels in [38], convolutional codes over AWGN channels in the short blocklength regime in [39], and turbo codes over AWGN and additive T-distribution noise (ATN) channels in [40]. Autoencoders were used in [41, 42] for novel code design for the feedback channel, for which no practical coding scheme exists even under known channel statistics. Despite the promising performance achieved by DL-aided channel code design over short to moderate blocklengths, extending this result to longer codes has been a challenge due to the exponentially growing codebook size with the blocklength.

Due to this difficulty, some of the prior works have explored using DNNs to decode existing channel codes, such as linear [24], convolutional [43], turbo [44], or polar [45] codes. Since the decoder of any known channel code can be treated as a classifier on the noisy channel output, this approach has provided promising results in either improving the error probability or reducing the computational complexity of the decoder. For example, in [46] a multiple-in multiple-out (MIMO) channel decoder using DNN was proposed that achieves a performance close to approximate message passing and semidefinite relaxation at a much reduced computational complexity. Even more impressive results have been achieved through DNN-based designs in settings where current codes fall short of the fundamental theoretical limits. For example, this is the case for channels that are harder to model, such as optical [47] and molecular [48] communications, or even blind channel equalization [49].

Autoencoder-based channel codes do not yet provide significant improvements over existing conventional codes, especially at longer blocklengths. However, state-of-the-art channel codes are the result of decades-long intense efforts and expertise, whereas data-driven DNN techniques that were introduced only a few years ago have already achieved impressive results, especially for short blocklengths. Moreover, channel codes designed using DL tend to be more resilient than conventional codes, such as when the channel conditions change with respect to the underlying model they were designed for. This effect is also observed when DNNs are used for decoding of conventional channel codes.

### 2.3.3 JSCC with DL

As we have mentioned earlier, there is a long history of research on JSCC, and there are numerous studies both on information theoretical limits and on practical
code design. However, existing results have either been for specific designs exploiting the properties of a particular source signal or were too complex to be used in practice. Moreover, such code designs have not provided significant improvements in practice to justify the introduced complexity in their design. The lack of low-complexity, high-performance JSCC solutions together with the recent advances in DL-aided coding schemes for source compression and channel coding have motivated the application of DL to design novel JSCC schemes.

In DL-aided JSCC, the input to the DNN encoder is the source signal, while its outputs are the symbols transmitted over the channel. The channel is modeled or approximated as another layer in the DNN architecture, where the output of the channel is passed to the DNN decoder. The encoder and the decoder for JSCC can be trained end-to-end using a data-driven approach. This process is illustrated in Fig. 2.2(c).

One of the first works that proposed JSCC using neural networks is [50], where simple neural network architectures were used as encoder and decoder for Gaussian-Markov sources over the additive white Gaussian noise channel. More recently in [51–56], autoencoder-based solutions for end-to-end design and optimization of JSCC were proposed.

Specifically, [51, 57] focus on text as the information source that is communicated over discrete channels, [52] considers JSCC for lossy data storage, and image transmission over an AWGN wireless channel is studied in [54–56, 58, 59]. In [54], the authors propose a fully convolutional autoencoder architecture, which maps the input images directly to channel symbols, without going through any digital interface. The authors show that the proposed DeepJSCC architecture not only improves upon the concatenation of state-of-the-art compression and channel coding schemes in a separate architecture [55], but it also provides graceful degradation as the channel SNR degrades. This latter property, which is common to analog transmission schemes, provides significant benefits compared to digital schemes, which exhibit catastrophic error when the channel SNR significantly deviates from an expected value determined on system’s design. This is particularly common when broadcasting to multiple receivers or when transmitting over a time-varying channel. DeepJSCC is also shown in [56, 58] to be almost successively refinable; that is, an image can be transmitted in stages, where each stage refines the quality of the previous stages at almost no additional cost. Finally, in [59] JSCC of images transmitted over binary symmetric and over binary erasure channels are considered. To overcome the challenges imposed by the nondifferentiability of discrete latent random variables (i.e., the channel inputs), unbiased low-variance gradient estimation is used, and the model is trained using a lower bound on the mutual information between the images and their binary representations.

One of the other benefits of JSCC using DL is the ability to jointly optimize the encoder and the decoder for the downstream DL task. For example, if the receiver is interested in object detection using DL on an image received from the transmitter, the JSCC encoder and decoder can be trained together with the image-detection network in an end-to-end manner, thereby optimizing the encoder and decoder for the downstream task rather than just for image reconstruction.
In the rest of this chapter, we separately focus on JSCC for discrete and continuous channels and provide specific examples on how the general approach proposed in this section can be applied for JSCC design in different domains. Specifically, we first consider a specific JSCC design for text transmission over discrete input channels in Section 2.4 and then a specific JSCC design for image transmission over continuous input channels in Section 2.5.

2.4 DL-Aided JSCC for Text

In this section, we focus on the JSCC of text over discrete channels using a DL approach. In several applications, system performance is not measured by fidelity to the transmitted data but by the performance of downstream systems that use the received data. That is, the receiver is less interested in the exact recovery of the transmitted data than in the relevant information of interest or a facsimile of the data that would be used in downstream applications. In the case of text data, the receiver would be interested in recovering the semantic content of a sentence including facts, relations, topics, or keywords as opposed to the exact sentence that can include non-informative carrier phrases. Downstream natural language processing tasks of summarization, topic classification, sentiment detection, intent, and named-entity extraction would use these artifacts. We declare decoded sentences error free if they convey the equivalent information as the original sentence, even if they are paraphrased (e.g., “the car stopped” and “the automobile came to a halt”). The neural network architecture for JSCC of text we develop is inspired by recent state-of-the-art results of DL in natural language processing tasks such as machine translation, summarization, and semantic understanding.

Our model is composed of a recurrent neural network (RNN) encoder, a stochastic binarization layer, the channel layer, and a decoder based on RNNs. We use this architecture to train a JSCC encoder-decoder pair and show that it is possible to obtain different but equivalent sentences that preserves the semantic content of the transmitted sentence. We introduce schemes where a fixed-length binary encoding is produced for an input sentence as well as one in which encodings of variable lengths are produced for sentences that vary in length and complexity. The latter variable-length architecture would improve on the performance of the fixed-length encoding scheme with a less strict average sentence encoding length constraint expending more bits to encode longer sentences than frequently occurring shorter ones.

The performance of our DL encoder and decoder is contrasted with separate source and channel coding design. In the separate design, channel coding is done using Reed-Solomon codes. For compression or source coding, we consider three different methods: a universal source coding scheme, Huffman coding, and a 5-bit character ASCII encoding. We show that the proposed DL encoder and decoder does better than the separate design on the metric of word error rate (WER) or edit distance, when each sentence is encoded using fewer bits. In several cases, the DL decoder may insert, replace, or substitute words that preserve the semantic content of the sentence in a
qualitative sense, but this will not be reflected in the edit distance that penalizes these transformations. In order to capture the impact of replacing words with synonyms, a new metric is also proposed that scores word substitutions with a similarity score between them.

2.4.1 System Model

The system model in this particular application is defined as follows: Let \( \mathcal{V} \) be the entire vocabulary, indexing the set of all the words in the language. Then the source in our model is \( x \in \mathcal{V}^n \), where \( n \) is the length of the sentence and \( x = [w_1, w_2, \ldots, w_n] \) is a vector representing the sequences of words in the sentence. Note that, although the source is not a real number in this setup, \( \mathcal{V} \subset \mathbb{R} \). As the channel, we consider the binary erasure channel (BEC), the binary symmetric channel (BSC), and the AWGN channel. Let \( \hat{x} = [\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_{\hat{n}}] \) be the output of the JSCC decoder (i.e., the recovered sentence). With this framework, the number of words in the decoded sentence can differ, or equivalently, we can have \( n \neq \hat{n} \). Specifically, we design the JSCC encoder and decoder (i.e., \( f_\theta \) and \( g_\phi \)) such that the meaning between the transmitted sentence \( x \) and the recovered sentence \( \hat{x} \) is preserved. Therefore, the transmitted and recovered sentences may have different words and different lengths. We now further describe the component modules in the system.

Neural JSCC Architecture for Text

The DL architecture we implement is inspired from the sequence-to-sequence learning framework [60]. The end-to-end neural JSCC architecture is shown in Fig. 2.3. It has primarily three components: the encoder, the channel, and the decoder. The encoder \( f_\theta \) takes a sentence \( x \) as input and produces a binary encoding \( z \in \mathbb{Z}^k \), where \( \mathbb{Z} = \{0, 1\} \). The channel transforms this bit vector \( z \) to realize an output vector \( \hat{z} \) at the receiver. This module is stochastic. We will consider different cases, where the channel output alphabet is either binary, ternary, or continuous. The channel output vector \( \hat{z} \) is the input to the decoder \( g_\phi \), and the output of the decoder is the estimated sentence \( \hat{x} \). We now describe each of these modules in detail.
Encoder

The encoder first applies an embedding layer to generate a continuous vector representation for each word of the input sentence. We make use of pretrained Glove word vectors [61] to initialize the embedding layer. Glove word vectors have been obtained using joint co-occurrence statistics of words from a large text corpora; they have been shown to capture the semantic meaning of words as demonstrated by performance in the word analogy task. The embedding is represented by $E = \theta_e(s)$, where $E = [e_1, e_2, \ldots, e_n, e_{eos}]$ is the $n + 1$ embeddings of words in the sentence. We have $n + 1$ words in the sentence as an additional end of sentence symbol is affixed in the data preparation process.

In the next step of the encoder, the word embeddings are inputs to a stacked bidirectional long short term memory (BLSTM) network [62]. LSTM cells with peepholes have been used in this work similar to that used in [63]. We can represent the BLSTM layers by

$$r = \theta_{BLSTM}(E), \quad (2.4)$$

where $r$ is the output state of the BLSTM stack. Each individual layer of the stack comprises two states from the forward network and backward network. These states from all the layers are concatenated to form $r$.

The output state $r$ is then fed to a feed-forward dense layers with tanh activation or a multilayer perceptron (MLP). This layer is used to modulate the dimension of the binary encoding of the sentence. Or, the MLP is used to increase or decrease the dimension of $r$ to $\ell_{max}$, the maximum number of bits used to encode the sentence. This is represented by

$$v = \theta_{MLP}(r), \quad (2.5)$$

where $v \in [-1, 1]^\ell_{max}$.

The final step in the encoder is to binarize $v$ from the interval $[-1, 1]$ to binary values $\{-1, 1\}$. We define a scalar stochastic binarization function as

$$\theta^{sto}_\beta(x) = x + Z_x, \quad (2.6)$$

where $Z_x$ is a random variable with distribution

$$Z_x \sim \begin{cases} 1 - x & \text{w.p. } \frac{1 + x}{2} \\ -x - 1 & \text{w.p. } \frac{1 - x}{2} \end{cases}. \quad (2.7)$$

This final binarization step is

$$z = \theta^{sto}_\beta(v) \quad (2.8)$$

in the forward pass. A custom operation is defined to back-propagate gradients through the binarization layer during training. The gradients are defined to pass straight through or unchanged through $\theta^{sto}_\beta$. This is obtained by using the derivative with respect to the expectation $[\theta^{sto}_\beta(v)] = v$ to calculate the gradient [64]. When the trained
The variable length encoder that can produce binary codes with lengths proportional to the sentence length. If we are constrained by the average length of encoding of each sentence, it is optimal to encode shorter sentences with encodings of a shorter length and expend more bits to encode longer sentences. This would also occur if we are to transmit a block of text or collection of sentences using as few bits as possible.

Figure 2.4 describes the architecture of the variable length encoder. This is accomplished by transmitting the first $\ell_n = L(n)$ encoded bits, where $L$ is a function that maps the sentence length $n$ to the length of binary encoding $\ell_n$. We implement this code puncturing in the neural network model by zeroing out the last $\ell_{\text{max}} - \ell_n$ bits in $x$. As bits are represented using -1 and 1, symbol 0 is equivalent to not transmitting the bit.

Channel
The next module in the system models the communication channel. The transform function of the channel that maps the input to the output at the receiver must be differentiable to facilitate joint training of the encoder and decoder using variants of stochastic gradient descent. In this chapter, we consider three different channels: the BEC, the BSC, and the AWGN channels. The framework can be extended to models with memory such as inter-symbol interference (ISI) channels or channels with nonlinearities.

The BEC can be implemented in training via a dropout layer [65],

$$\hat{z} = \eta_{\text{bec}}(z, p_d),$$  \hspace{1cm} (2.9)

where $\hat{z}$ is the received vector, and $p_d$ is the probability of erasing a bit. The elements of $\hat{z}$ are in the ternary set $\{-1, 0, 1\}$, where 0 indicates an erasure. Each bit in $z$ may or may not be dropped independent of other bits.
The BSC can be modeled as
\[
\hat{z} = \eta_{\text{bsc}}(x, p_e) = n_{p_e} \odot z,
\]
(2.10)
where \(n_{p_e}\) is the noise introduced by the channel with an element of the vector equal to 
\(-1\) with probability \(p_e\) and \(1\) otherwise, and \(\odot\) denotes element-wise multiplication.
In other words, bits may be inverted with probability \(p_e\).

Finally, the real-valued AWGN channel is represented by
\[
\hat{z} = \eta_{\text{awgn}}(z, \sigma^2) = z + n_{\sigma^2},
\]
(2.11)
where \(n_{\sigma^2}\) is an additive noise term with elements arising from i.i.d. Gaussian random
variables with zero mean and variance \(\sigma^2\). Note that the output \(y\) consists of a vector
of real values here.

**Decoder**

At the receiver, the first step in the decoding process is to change the dimension of the
observation vector \(\hat{z}\) using an MLP or a feed-forward network:
\[
c = \phi_{\text{MLP}}(\hat{z}).
\]
(2.12)
The MLP is an adapter module that transforms the received vector so that it can serve
as the initial state of the next submodule in the decoder, a stacked LSTM decoder.
The stacked unidirectional decoder with initial state \(c\) emits each word of the decoded
sentence auto-regressively. This is given by
\[
\hat{x} = \phi_{\text{LSTM}}(c, \langle \text{sos} \rangle),
\]
(2.13)
where \(\hat{s}\) represents the decoded sentence. The embedding vector for the special start
of the sentence symbol \(\langle \text{sos} \rangle\) serves as the first input to the LSTM stack. At each
decoding step, the word is sampled from the distribution over words in the vocabulary
represented by the logits of the output layer and this embedding of this word serves
as the input for the next decoding step. During training, we set up a schedule for
annealing the probability of using the previously decoded word as the decoder input
or using the ground truth data. During model deployment and inference, we have a
choice of greedy decoding that uses the previously decoded word or a beam search
algorithm that maintains a fixed number of likely beams. Beam decoding ensures that
a high probability decoding error early on does not cascade to the entire sentence that
results in overall lower beam probability.

### 2.4.2 Experimental Setup

**Dataset**
The sentences that are transmitted are drawn from the News Crawl 2015 dataset
[66]. This dataset was constructed by crawling through articles of various online
English publications. Our vocabulary is selected using the 20,000 most popular words
in the dataset. We filter sentences from lengths 4 to 30 with less than 20% of the

https://doi.org/10.1017/9781108966559.004 Published online by Cambridge University Press
vocabulary words. We used 18.5 million sentences, or 90%, for model training and validation, and the remaining 2 million sentences form the test set.

**Model Training Details**

The loss function used to train the model is categorical cross-entropy function that minimizes the Kullback-Leibler (KL) divergence between the probabilities of the decoded words, represented by the logits emitted by the decoder final layer, and the one-hot distribution of the ground truth sequence.

To initialize the encoder word embedding layer, 200-dimensional pretrained Glove embeddings [61] are used. The special symbols out-of-vocabulary, padding, start of sentence, and end of sentence are randomly initialized using the uniform distribution scaled inversely be embedding length. The data processing pipeline prepares batches of 512 sentences bucketed by their sequence lengths as input to the JSCC encoder. The encoder BLSTM has two layers of 256 units with peephole connections.

To obtain a constant length encoding, the encoder MLP transforms the concatenated end states of the BLSTM layers to a vector of dimension $\ell$, the number of transmission bits. To obtain variable length embeddings, we log the histogram of sentence lengths. To map the sequence length to binary encoding length, the smallest bucket (of length 4–7) are allotted 250 bits. Subsequent buckets of width 4 are allotted 50 more bits linearly. The average number of bits per sentence amounts to 400. The encoder MLP maps the BLSTM states to the maximum bit length, which is then punctured. The decoder LSTM stack has 2 512-unit layers with peephole connections.

An Adam optimizer with initial learning rate of $10^{-3}$ is used for 6 epochs on the training dataset. At inference time, a beam decoder with width 10 is used.

**Baselines**

Separate source and channel coding schemes serve as baselines for comparison. As previously noted, the separate design is optimal for arbitrarily large blocklengths with no constraints on delays for the channels we consider. We consider the following source coding approaches:

1. Universal compressors: Lempel-Ziv universal compression [67] implemented in gzip is the first method. Universal compression asymptotically reaches the entropic compression limit for any arbitrary source of data. Empirically, large blocks of sentences are required for good compression performance, and we use blocks of 30 sentences for evaluation. This method is unsuitable for the transmission of single sentences unlike the other baselines and the proposed DL JSCC encoder.

2. Huffman coding: The character frequencies obtained using the validation set are used in a Huffman coding scheme to encode characters of single sentences.

3. Fixed-length character encoding: This is the computationally simple baseline that assigns a fixed 5-bit ASCII coding for the characters including the lower case alphabet and the special symbols. We implement this baseline as the receiver can retrieve the sentence partially when the channel introduces noise to the source code. This partial retrieval capability is not available in the other baselines.

https://doi.org/10.1017/9781108966559.004 Published online by Cambridge University Press
In the separate design baselines, channel coding is done via Reed-Solomon codes [68] after source coding. With \(x\) bits of added redundancy, this can correct up to \(x\) erasures in the binary erasure channel or \(\lfloor x/2 \rfloor\) inversion errors of the binary symmetric channel or the maximum likelihood decoded binary vector in the AWGN channel. The message cannot be decoded at all with gzip or Huffman codes if the channel introduces more erasures or errors than the channel code can correct for. We tune the number of redundant bits added to maximize word throughput based on channel statistics via the following trade-off: too few parity bits would result in a high probability of decoding failures of the entire sentence and too many parity bits will require words to be dropped so that the sentence can be transmitted with the required bit budget.

Performance is measured via the normalized edit or Levenshtein distance. This metric is obtained by a dynamic programming algorithm that finds the shortest sequence of insert, delete, or substitute operations to map the reference sentence to the hypothesis. To capture the semantic similarity more accurately, we underweight the substitution cost in a new metric by using the Wu-Palmer score for relatedness of words [69].

2.4.3 Results

In Fig. 2.5(a), we first observe the effect of the bit erasure rate of the BEC on the word error rate (WER). Gzip is the best performing baseline with large batches of sentences. In the regime where the binary erasure probability is large, the DL JSCC model proposed outperforms the baselines, suggesting that the former is robust to channel errors and exhibits graceful degradation in performance. In Fig. 2.5(b) we use the BSC and observe an increase in WER with sentence lengths across the board with the performance of the proposed method comparable to baselines in the limited bit regime for longer sentences. We reiterate that word errors for the baselines imply that a word was lost at the receiver as it either was not transmitted or incorrectly decoded, whereas the DL JSCC may preserve the semantic content of the sentence. Figure 2.5(c) reinforces the conclusions of the earlier plots with the AWGN channel model.

Figure 2.6(a) first compares the fixed-length encoding architecture to the variable length one. Longer sentences are allotted more bits in the latter scheme, leading to significantly fewer errors without impacting the performance for shorter sentences encoded with fewer bits. As can be seen, the increased bit allotment for longer sentences results in fewer errors without much loss in performance for shorter sentences. Finally, in Fig. 2.6(b) we investigate the impact of source and channel coding in JSCC by contrasting a DL JSCC network with 450 bits/sentence for the binary erasure channel with a scheme where the neural network is used only for source coding that generates a 400 bit representation after which Reed-Solomon coding with 50 added parity bits is used. The plots suggest that the latter scheme performs fractionally better, implying that source coding accounts for most performance gains for DL JSCC. As noted in Section 2.3.2, DNN based channel coders outperform conventional
techniques for channels that are hard to model such as optical or molecular channels, and we similarly expect the gap between DL JSCC and source coding with Reed-Solomon coding to reduce or invert with such channels.

In the next section, we describe JSCC for images over continuous input channels.

**Figure 2.5** Performance plots of DL JSCC with constant length encoding: (a) Word error rate (WER) increases with erasure probability in BEC for 400 bit encoding, (b) WER for sentences of different lengths for a binary symmetric channel with error rate 0.5, and (c) WER for sentences of different length with an AWGN channel with a standard deviation of 0.6.

**Figure 2.6** (a) Contrasting WER with sentence length for fixed and variable length encoding with the BEC channel with erasure probability 0.1 and 400 bit average sentence encoding length, and (b) comparing DL JSCC with neural network source coding and Reed-Solomon coding.
2.5 DL-Aided JSCC for Images

An increasing number of applications involve transmission of images over wireless channels. This is not only for the traditional human-to-human (e.g., social networks, messaging, online content) communications, but also for human-to-machine (e.g., artificial or virtual reality, telepresence) and machine-to-machine (e.g., surveillance, pattern recognition) applications that are increasingly connected through wireless links. This exponentially growing demand for high quality image communication under strict latency constraints present new challenges on the wireless infrastructure.

This section reviews recent developments on DeepJSCC, an autoencoder-based solution for generating robust and compact codes directly from images pixels. The channel input symbols produced by DeepJSCC are not constrained to a specific constellation, so it is able to operate with continuous values (instead of discrete, as in the previous section). This property ensures that DeepJSCC can present analog behavior such as graceful degradation. We demonstrate with experimental results that DeepJSCC achieves superior performance over the state-of-the-art digital communication schemes (BPG/ JPEG2000 compression followed by LDPC+QAM for transmission) on static channels, presenting graceful degradation as the channel quality degrades, and can successfully adapt to time-varying channels. Finally, we present a hardware implementation of the scheme, showing the model’s application to real physical channels.

2.5.1 System Model

Consider an image source with height $H$, width $W$, and $C$ color channels, represented as a vector of pixel intensities $\mathbf{x} \in \mathbb{R}^n$; $n = H \times W \times C$. An encoder $f_{\theta_i} : \mathbb{R}^n \rightarrow \mathcal{C}^{k_i}$ maps $\mathbf{x}$ into a block of channel input symbols $z_i \in \mathcal{C}_i^{k_i}$. The transmission of $k$ symbols is split over $L$ layers, so the total bandwidth $k$ is achieved by accumulating $L$ transmissions ($\sum_{i=1}^{L} k_i = k$). Unlike in the previous section, the encoder outputs continuous complex symbols, that is, $\mathcal{Z} = \mathcal{C}$. These symbols are transmitted over a noisy channel, characterized by a random transformation $\eta : \mathcal{C}_i^{k_i} \rightarrow \mathcal{C}^{k_i}$, resulting in the received symbols $\hat{y}_i$. The decoder $g_{\phi_i} : \mathbb{C}^{k_i} \rightarrow \mathbb{R}^n$ attempts to reconstruct the original image by minimizing the reconstruction error $\mathcal{L}(\hat{y}_i, \mathbf{x})$. The overall encoder and decoder architectures used in experiments are shown in Figure 2.7.

![Figure 2.7 Encoder and decoder architectures used in experiments.](https://doi.org/10.1017/9781108966559.004)
in the corrupted channel output \( \hat{z}_i = \eta(z_i) \). We consider \( L \) distinct decoders, where the channel outputs for the first \( i \) layers are decoded using \( g_{\phi_i} : c^{k_i} \rightarrow \mathcal{R}^n \) (where \( I = \sum_{j=0}^{i} k_j \)), creating reconstructions \( \hat{x}_i = g_{\phi_i}(\hat{z}_1, \ldots, \hat{z}_i) \in \mathcal{R}^n \), for \( i \in 1, \ldots, L \).

Figure 2.7 shows the component blocks of the chosen architecture and its hyperparameters. The encoder and decoder are composed of a series of trainable convolutional neural network (CNN) blocks, using generalized normalization transformations (GDN/IGDN) [70], followed by a parametric rectified linear unit (PReLU) [71] activation function (or a sigmoid, in the last decoder block). This architecture was inspired by [72] and improved with ablation studies. Intuitively, convolutional layers extract image features, GDN apply local divisive normalization, and the nonlinear activation allows the learning of nonlinear mapping from the source signal space to the coded signal space. The communication channel is incorporated into the model as a nontrainable layer. Although different channel models are considered, as described in the experimental section, all of them are differentiable transfer functions, allowing their inclusion in the general architecture and enabling gradient computation and error back propagation.

Before transmission, the latent vector \( z'_i \) generated at the encoder’s last convolutional layer is normalized to enforce an average power constraint so that
\[
\frac{1}{k_i} \mathbb{E}[z'_i z'_i] \leq P,
\]
where \( \tilde{z}_i^* \) is the conjugate transpose of \( \tilde{z}_i \).

The model can be optimized to minimize the average distortion between input \( x \) and its reconstructions \( \hat{x}_i \) at each layer \( i \):
\[
(\Theta^*_i, \phi^*_i) = \arg \min_{\Theta_i, \phi_i} \mathbb{E}_{p(x, \hat{x}_i)}[d(x, \hat{x}_i)],
\]
where \( d(x, \hat{x}_i) \) is a specified distortion measure, usually the mean squared error (MSE), although other metrics are also considered. Since the true distribution of \( p(x) \) is unknown, an analytical form of the expected distortion in Eq. (2.15) is also unknown, so we estimate the expected distortion by sampling from a dataset.

When \( L > 1 \), we have a multiobjective problem. However, we simplify it so that the optimization of multiple layers is done either jointly, by considering a weighted combination of distortions, or greedily, by optimizing \((\Theta_i, \phi_i)\) successively. Please see [56, 58, 73] for more details.

### 2.5.2 Evaluation Metrics

In order to measure the performance of the proposed JSCC algorithm and alternative schemes, we use the peak signal-to-noise ratio (PSNR), given by
\[
\text{PSNR} = 10 \log_{10} \frac{\text{MAX}^2}{||x - \hat{x}||^2} \quad (dB),
\]
where \( \text{MAX} \) is the maximum value of the input signal.
where $\text{MAX}^2$ is the maximum power our input signal can have. When evaluating 24-bits RGB images, $\text{MAX} = 255$ is given by the maximum value a pixel can represent.

The quality of the channel is measured by the average signal-to-noise ratio (SNR) given by

$$\text{SNR} = 10 \log_{10} \frac{P}{\sigma^2} \ (dB),$$

(2.17)

representing the ratio of the average power of the channel input signal to the average noise power. $P$ is set to 1 in all experiments.

To compare the performance of DeepJSCC to traditional separation-based digital schemes, we consider different well established source codes followed by LDPC codes for error correction. For the problem of image transmission, we use as source codes JPEG, WebP, JPEG2000, and BPG, and we discount the header information for BPG and JPEG2000 when computing bit rates and transmission sizes for fair comparison. For the channel code, we consider all possible combinations of $(4096, 8192)$, $(4096, 6144)$, and $(2048, 6144)$ LDPC codes (which correspond to $1/2$, $2/3$, and $1/3$ rate codes) with BPSK, 4-QAM, 16-QAM, and 64-QAM modulation schemes.

For each channel code configuration, we can define the maximum rate $R_{\max}$ (bits per pixel) at which we can transmit an image (using the channel code rate) and empirically evaluate the frame error rate $\epsilon$ for each channel model and condition we consider. Then, we compress the images (using the different codecs) at the largest rate $R$ that satisfies $R \leq R_{\max}$. We consider that the transmission can either be successful or fail, with probability of failure $\epsilon$. When the transmission fails, we consider that the reconstruction at the receiver is set to the mean value for all the pixels. When the transmission is successful the distortion is dictated by $R$ and the compression scheme used. We then measure the average performance over the evaluation dataset.

### 2.5.3 Experimental Setup

Using the model described in the previous section we perform a series of experiments using RGB source images. We train and evaluate the model with distinct datasets and all plotted performance values are averaged from 10 realizations of the channel for every image on the evaluation dataset. All results presented use the same compression ratio of $k/n = 1/6$, although similar results apply to other ratios.

The model was implemented in Tensorflow [74] and optimized using the Adam algorithm [75]. We used a learning rate of $10^{-4}$ and a batch size of 16. Models were trained until convergence, when the loss does not decrease after new iterations. The loss function used for training of the model is the average mean squared error (MSE) over $N$ samples:

$$\mathcal{L} = \frac{1}{N} \sum ||\mathbf{x} - \hat{\mathbf{x}}||^2.$$  

(2.18)
2.5.4 Results

Baseline

Our first set of results demonstrates the base case in which an image $x$ is encoded by a single encoder and a single decoder and thus $L = 1$. As channel model we consider a complex AWGN channel, which is a common model for static wireless links. Its transfer function is given by

$$\eta_{\text{cawgn}}(z) = z + n,$$

(2.19)

where $n \in \mathbb{C}^k$ is a vector with i.i.d. elements sampled from a circularly symmetric complex Gaussian distribution $n \sim \mathcal{CN}(0, \sigma^2 I_k)$, where $\sigma^2$ is the average noise power.

In Fig. 2.8, we compare DeepJSCC at different channel SNRs. For comparison, we also plot the performance of well established separation-based schemes with JPEG, JPEG2000, WebP, and BPG codecs followed by LDPC channel coding. We see that the performance of DeepJSCC is either above or comparable to the performance of these separate source and channel coding schemes for a wide range of channel SNRs. These results show that we can obtain significant performance gain by a joint design.

In Fig. 2.9 we plot a visual comparison between the reconstructed output of DeepJSCC and a separation-based scheme using BPG+LDPC, for transmission over an AWGN channel, with channel SNR equal to 1 dB and $k/n = 1/24$. DeepJSCC achieves considerably higher PSNR and multiscale structural similarity (MS-SSIM) [76]. The performance difference can be seen clearly in the images, in which the DeepJSCC output presents more richness in details and sharpness, especially in high frequency components, such as background trees and leaves, while the BPG+LDPC output exhibits blurry artifacts over the whole image.

Graceful Degradation

These results are obtained by training a different encoder/decoder model for each SNR value evaluated in the case of DeepJSCC, and considering the best performance achieved by the separation-based scheme at each SNR.
Figure 2.9 Visual comparison of reconstructed output for transmission under AWGN channel, SNR = 1 dB, and $k/n = 1/24$. 

(a) Original image

(b) DeepJSCC: PSNR = 24.40 dB / MS-SSIM = 0.907

(c) BPG+LDPC: PSNR = 22.27 dB / MS-SSIM = 0.779
Here, we consider the situation where there is a mismatch between the channel conditions during design and deployment. In practice this might occur for several reasons. First of all, it is impractical to assume a different model can be stored and employed at the transceivers for each SNR value. Instead, the same model needs to be used at least for a range of SNR values. There may also be mismatch between the SNR estimate at the encoder and the real SNR value during transmission, either due to time variations in the channel or due to imperfect channel estimation and feedback.

For this, we consider different DeepJSCC models trained for specific SNRs and show the evaluation of the test dataset for a range of SNRs, lower and higher than that is used for training. In Fig. 2.10(a) we show a separate curve for each model trained at a specific channel SNR (SNR$_{\text{train}}$), and the performance of each model is plotted against the test SNR (SNR$_{\text{test}}$). We also plot the performance of the separation-based scheme with BPG, showing the best performing LDPC code at each SNR.

It can be clearly seen that DeepJSCC presents graceful degradation; that is, the performance gradually decreases as the channel SNR deteriorates, while the digital scheme presents a cliff-effect when the quality of the channel goes below the capacity for which the code was designed, resulting in indistinguishable transmission output. Thus, we can see that DeepJSCC not only produces high quality transmissions (when compared to digital schemes), but also analog behavior, being more robust to nonergodic channels.

**Channel Versatility**

A big advantage of DeepJSCC being data-driven is the possibility of training for different channel models, objective functions, or specific domains.

To better illustrate the advantage of graceful degradation, we consider transmission over the more challenging Rayleigh fading channel, which models variations in
channel quality over time due to physical changes in the environment. The channel is modelled by a random channel gain:

\[
\eta_{\text{fading}}(z) = h z + n,
\]

(2.20)

where \( h \sim \mathcal{CN}(0, H_c) \) is a complex normal random variable.

Here we consider a slow Rayleigh fading channel and assume that the channel gain \( h \) remains constant during the transmission of a single image and changes to an independent value for the next image following the Rayleigh distribution. We do not assume channel state information (CSI) either at the receiver or the transmitter, but we consider that the phase shift introduced by the channel is known at the receiver, making the model equivalent to a real fading channel with bandwidth \( 2k \). In order to emulate and measure the average channel SNR, we define \( H_c = 1 \) and vary the noise variance \( \sigma^2 \) over transmissions.

In Fig. 2.10(b) we present results for models trained at different average channel SNRs and tested at different average SNR values. We also compare our performance with different LDPC configurations, using BPG as compression. We do not consider any explicit channel estimation or feedback in the case of DeepJSCC, whereas we assume that the channel gain is known by the receiver for the digital transmission scheme, providing a clear advantage for the latter. Yet, DeepJSCC still outperforms the digital scheme significantly for the whole range of average SNRs considered. This is mainly due to the graceful degradation property of DeepJSCC, which means that accurate CSI knowledge at the transmitter is not required. On the other hand, we stipulate that the network architecture learns to employ channel estimation sufficient for the decoder network to correctly reconstruct the image over varying channel gains.

We can also consider a channel model with bursty noise, which can model a scenario in which individual symbols of the transmitted signal, apart from being perturbed by AWGN noise \( n \), can also be perturbed by a high variance noise with probability \( p \). Formally, this is a Bernouille-Gaussian noise channel with the transfer function

\[
\eta_{\text{bursty}}(z) = z + n + B(k, p) w,
\]

(2.21)

where \( B(k, p) \) is the binomial distribution, and \( w \sim \mathcal{CN}(0, \sigma^2_b I) \) the high variance noise component with \( \sigma^2_b \gg 0 \). In practice, this models an occasional random interference from a nearby transmitter. This channel model is also used in Kim et al. [43] to illustrate the robustness of DNN-based channel encoders with channel output feedback.

Figure 2.11(a) shows the effect of the probability \( p \) on the model’s performance for a bursty channel with AWGN component’s SNR equal to 10 dB. We consider both a low-power (\( \sigma_b = 0.5 \)) and a high-power (\( \sigma_b = 3.5 \)) burst and compare the performance with a digital scheme with BPG+LDPC. As expected, the performance degrades as \( p \) increases, but the DeepJSCC scheme is much more robust against the increasing power of the burst noise. This improved robustness of DeepJSCC is particularly obvious for \( \sigma_b = 3.5 \). A high-power burst degrades the performance of the digital scheme very quickly, even if the burst probability is very low, completely
destroying the signal when $p > 0.15$. In this region, all combinations of compression rates and codes are considered to have failed. In contrast, DeepJSCC exhibits a graceful degradation even in the presence of bursty channel noise, showing its advantages in practical scenarios, particularly for communications over unlicensed bands, where occasional burst noise is common.

Figure 2.11(b) shows the effect in performance of the bursty channel at different channel conditions, considering fixed values of $p = 0.10$ and $\sigma_b = 3.5$. The SNR displayed is with regards to the fixed AWGN component of the channel. We also plot the AWGN performance without any noise bursts, for comparison. The results show that, although impacted by the bursty noise, DeepJSCC still achieves a reasonable performance in the whole range of SNRs considered.

Interestingly the gap between the AWGN performance and the bursty channel increases with the SNR; quite surprisingly, we notice that in the low SNR regime (SNR $\leq 2.5$) the model trained and evaluated on the bursty channel achieves higher performance than the one trained and evaluated with only AWGN. This could be explained due to the fact that the network trained on the bursty channel better generalizes its latent vector representation given its probability of erasure – an effect similar to the use of dropout layers [77] as a regularization technique. For comparison, we also plot the performance of the model trained on the bursty channel and evaluated with fixed AWGN, where we can see that this generalization effect is even stronger at higher SNRs (SNR $\leq 5$).

**Domain Specific Communication**

Traditional image compression schemes are independent of the type of images being compressed. However, in principle, statistical properties of different types of images can be exploited to obtain a more efficient compression algorithm. A trivial example of this would be compressing black-and-white images instead of color images. But, a learning based image transmission strategy can exploit even less obvious statistical properties of the dataset that may not even be visible to human eye. To test the
capability of the DeepJSCC approach to adapt to a more specific class of images, we experimented by training our model with satellite image data [78], a plausible application of our model. Here we use the distortion measure of MS-SSIM [76] — a widely accepted image quality measure that better represents human visual perception than pixel-wise differences. The results, presented in Fig. 2.12 show that, when more specific image domains are considered, DeepJSCC can better adapt to it, significantly increasing the performance gap to conventional separation-based techniques.

**Successive Refinement**

Yet another advantage of DeepJSCC is its flexibility to adapt the transmission to different paths or stages. We now consider a model with $L > 1$, in which the same image is transmitted progressively in blocks of size $k_i$, $i = 1, \ldots, L$ and $\sum_{i=1}^{L} k_i = k$. We aim to be able to reconstruct the complete image after each transmission, with increasing quality, thus performing *successive refinement* [79, 80]. Progressive transmission can be applied to scenarios in which communication is either expensive or urgent. For example, in surveillance applications, it may be beneficial to quickly send a low-resolution image to detect a potential threat as soon as possible, while a higher resolution description can be later received for further evaluation or archival purposes. Or, in a multiuser communication setting, one could send a different number of components for different users, depending on the available bandwidth.

We therefore expand our system by creating $L$ encoder and decoder pairs, each responsible for a partial transmission $z_i$ and trained jointly (see [56] for implementation details and alternative architectures). Figure 2.13(a) presents results for the case $L = 2$, for $k_1/n = k_2/n = k/12$ and shows the performance of each layer for different channel SNRs, for the AWGN channel. Results show that the loss of dividing the transmission into multiple stages is not significant; when compared to a single transmission with $k/n = 1/6$ (dotted black curve in Fig. 2.13(a)), the model performs with approximately the same quality for most channel conditions. Moreover, we observe that every layer of the layered transmission scheme preserves all features of the single transmission, such as graceful degradation and adaptability to different channel models.
Figure 2.13 (a) Successive refinement with $L = 2$; (b) layered transmission with channel output feedback, for $L = 4$; and (c) comparison between simulated and hardware performance.

Channel Output Feedback

Another interesting direction to be explored by DeepJSCC is the use of channel output feedback, when it is available. Suppose that alongside the forward communication channel considered so far, there is also a feedback channel able to send back to the transmitter an estimation of the channel output $\hat{z}_i$ after its realization. In a multi-layered transmission, this information can be used to inform subsequent layers and enhance the reconstruction at the receiver. Thus, a transmission of a source $x$ is done sequentially in $L$ steps, in which each step $i$ a channel input $z_i$ is generated from input $x$ and feedback $\hat{z}_{i-1}$ (for $i > 1$), transmitted and decoded to generate successively refined representations $\hat{x}_i$ (see [73] for specific architecture and implementation details). There has also been recent advances in the use of channel output feedback to improve the performance of channel coding [41]; however, the design is for a specific blocklength and code rate, whereas the proposed DeepJSCC scheme can transmit large content, such as images.

Figure 2.13(b) shows the results for a scenario considering noiseless feedback (i.e., $\hat{z}_i = \hat{z}_i$) and three uses of the feedback channel ($L = 4$), for channel inputs with size $k_i/n = 1/12$, $i = 1, \ldots, 4$. We see that by exploiting the feedback information, DeepJSCC can further increase its performance, establishing its superiority over other schemes. Note that we compare DeepJSCC with feedback with a theoretical capacity achieving channel code and can still outperform the separation-based scheme.

This architecture enables other communication strategies, such as variable length coding, in which a minimum number of layers $z_i$ are transmitted and the quality of the reconstruction is estimated through feedback, until a target quality is achieved and the further transmission is interrupted. This scheme can provide gains of over 50 percent in bandwidth, when compared to separation-based approaches [73]. Further experiments also demonstrate that our model successfully operates under noisy channel feedback and even present graceful degradation when the feedback channel changes between training and evaluation.
Hardware Implementation

Finally, to validate the real world performance of the proposed architecture, we implemented our DeepJSCC scheme on a real communication channel, enabled by a software defined radio platform. We used models trained on the AWGN model, with different SNRs. Results can be seen in Fig. 2.13(c) and show that the simulated performance closely matches the hardware performance, especially in higher SNRs.

The JSCC approach to image transmission also provides significant gains in terms of end-to-end encoding and decoding delays. We observed that the average encoding and decoding time per image with DeepJSCC is 6.40 ms on a GPU, or 15.4 ms on a CPU, while a scheme with JPEG2000+LDPC and BPG+LDPC takes on average 4.53 ms and 69.9 ms on a CPU, respectively, using the same hardware as the used for DeepJSCC evaluation. This shows the competitive processing times of DeepJSCC together with its outstanding performance. Note that, as with all CNNs, both the encoder and decoder components of DeepJSCC are parallelized and benefit from multicore hardware such as GPUs; this is not true for the standard implementations of source and channel codes considered, hence only CPU times are shown. Moreover, the DeepJSCC implementation uses standard and generic Tensorflow libraries and could still be further optimized for speed, if necessary.

2.6 Conclusion

Optimality of separation between compression and channel coding promised by Shannon’s theorem assumes no constraint on the complexity of the system or the associated delay. However, in practice, large blocklength source and channel codes may not be feasible due to the computational complexity and delay constraints. Therefore, many practical communication systems can benefit from designing the source/channel codes jointly. However, despite the suboptimality of separate design, the lack of low-complexity JSCC schemes that can notably outperform the alternative modular design approach has prevented the adoption of JSCC in practice. Recent successes of machine learning, in particular, DNNs, for a variety of complex tasks in image processing and natural language processing have motivated their use for JSCC design.

In this chapter, we reviewed DL-aided JSCC design over both continuous and discrete input channels for the transmission of different information sources, particularly focusing on images and text. The results show that for both types of channels, DL-aided JSCC design achieves a performance comparable or superior to the state-of-the-art separation-based schemes. Moreover, when the channel conditions deteriorate, DL-based JSCC achieves graceful degradation in performance, in contrast to the cliff effect observed in separation-based schemes. This data-driven approach is versatile and can be applied to different channels and domains. In the case of text data, this technique can also be used to recover sentences that preserve semantic meaning at the receiver relevant for downstream tasks rather than the exact sentence. Moreover,
the JSCC design based on DL can be successively refined, and can exploit channel output feedback in order to improve the communication. It is also possible to build support for variable length encoding for the JSCC using DL to further optimize performance with average encoding length constraints. These properties, coupled with its lower computational complexity, make deep JSCC code design an attractive approach for signal transmission in wireless communications, particularly for low-latency and power-limited constraints.

References


