BottleNet++: An End-to-End Approach for Feature Compression in Device-Edge Co-Inference Systems

Jiawei Shao, Jun Zhang
Department of Electronic and Information Engineering
The Hong Kong Polytechnic University, Hong Kong
Email: jiawei.shao@connect.polyu.hk, jun-eie.zhang@polyu.edu.hk

Abstract—The emergence of various intelligent mobile applications demands the deployment of powerful deep learning models at resource-constrained mobile devices. The device-edge co-inference framework provides a promising solution by splitting a neural network at a mobile device and an edge computing server. In order to balance the on-device computation and the communication overhead, the splitting point needs to be carefully picked, while the intermediate feature needs to be compressed before transmission. Existing studies decoupled the design of model splitting, feature compression, and communication, which may lead to excessive resource consumption of the mobile device. In this paper, we introduce an end-to-end architecture, named BottleNet++, that consists of an encoder, a non-trainable channel layer, and a decoder for more efficient feature compression and transmission. The encoder and decoder essentially implement joint source-channel coding via lightweight convolutional neural networks (CNNs), while explicitly considering the effect of channel noise. By exploiting the strong sparsity and the fault-tolerant property of the intermediate feature in a deep neural network (DNN), BottleNet++ achieves a much higher compression ratio than existing methods. Furthermore, by providing the channel condition to the encoder as a parameter, our method enjoys a strong generalization ability in different channel conditions. Compared with merely transmitting intermediate data without feature compression, BottleNet++ achieves up to 64 × bit compression ratio in the binary erasure channel, with less than 2% reduction in accuracy. With a higher compression ratio, BottleNet++ enables splitting a DNN at earlier layers, which leads to up to 3 × reduction in on-device computation compared with other compression methods.

Index Terms—Deep Learning, Device-Edge Co-Inference, Network Compression, Joint Source-Chanell Coding

I. INTRODUCTION

Recently, applications enabled by various mobile and Internet of Things (IoT) devices have profoundly changed our daily life [1]. One primary driver of these applications is the recent breakthrough in Deep Neural Networks (DNN) [2]. Unfortunately, DNN-based applications typically require a tremendous amount of computation, so they cannot be directly deployed on resource-constrained mobile devices. A common approach to solve this problem is to transmit the raw data to be processed at the cloud or edge computing platforms [3], [4]. The main disadvantage of this approach is the huge amount of communication overhead, which leads to considerable latency and energy consumption [5]. Recently, on-device inference via compressing and deploying DNN models to be deployed at mobile terminals has received lots of attention, whose performance, however, is severely constrained by the limited resources on the device. This has driven the development of other alternatives, among which device-edge co-inference is a promising solution [6], [7].

By splitting a DNN and deploying the front part of the network on an edge device and the remaining part on an edge server, device-edge co-inference brings the following benefits. First, since the feature volume of each layer will shrink in size as the neural network goes deep, the communication overhead can be effectively reduced by carefully selecting a model splitting point. Second, the DNN can extract the embedding feature of the input data and lead to the sparsity of the intermediate feature [6], [8], which will create an excellent opportunity for compression to reduce the communication overhead further. The third benefit is that a flexible tradeoff can be achieved among on-device computation and communication by splitting at different layers. While model splitting has received lots of attention [9], feature compression is less well studied. In DNNs, such as ResNet [10], there is an in-layer data amplification phenomenon, i.e., the output data size of early layers may be larger than the original input data [9]. Thus, without effective feature compression, the splitting point needs to be deep enough until the communication overhead of the intermediate feature is small enough, which leads to more layers deployed on the device, thus more on-device computation. So feature compression affects both the communication overhead and the amount of on-device computation and should be carefully investigated.

In this paper, we propose an end-to-end trainable architecture, named BottleNet++, for efficient feature compression in device-edge co-inference systems. BottleNet++ consists of an encoder, a non-trainable channel layer, and a decoder. Both the encoder and decoder adopt lightweight convolutional neural networks (CNNs). They essentially implement the idea of joint source-channel coding and provide a stronger ability to compress the intermediate network features compared with the hand-crafted compression algorithms like JPEG and JPEG2000. Furthermore, by training the encoder and decoder in an end-to-end manner, while explicitly considering the communication channel effect, BottleNet++ effectively exploits the fault-tolerant property of the DNN for a higher compression ratio. To further improve the model generalization ability in different channel conditions, the channel condition is added as a parameter to the encoder. One method similar to our
To deploy a DNN on source-constrained devices, there are three primary solutions: the server-based method \cite{3}, the on-device processing \cite{4}, and the hybrid deployment \cite{12}. In the server-based method, the data is transmitted to be processed at the server, preferably a proximate edge server. However, the latency is limited by the delay caused by uploading a large amount of original data. On-device processing with model compression can significantly reduce latency, but will also degrade the accuracy due to model over-compression to meet the on-device computation and memory constraint. The hybrid deployment, called device-edge co-inference in this paper, utilizes both the mobile device and the edge server for the execution to achieve a better tradeoff between latency and on-device computation. Our study endeavors to increase the compression ratio of the intermediate feature, which will help to reduce the communication overhead and on-device computation. It is achieved by exploiting the compression potential from model compression, joint source-channel coding, as well as the fault-tolerant property of neural networks.

### II. Preliminary

DNN has become a powerful method for many applications, but it typically requires a tremendous amount of computation. Feature compression has attracted lots of recent attention. The feature coding method proposed in \cite{6} applied JPEG coding and Huffman coding to compress the intermediate data. The method in \cite{11} combined network splitting and model pruning, which first prunes the weights of the network and then selects the splitting point based on the pruned model. However, the pruning step would consume plenty of time for a deep network in the pruning process. Distilled Split DNN \cite{13} applies the idea of network distillation to the head network deployed on the device and compresses its parameters to reduce communication overhead. Nevertheless, this method suffers from massive time consumption to retrain the distilled network. More recently, BottleNet \cite{11} proposed to encode the intermediate feature by a neural network and use a compression-aware training approach to reduce the accuracy loss. A key difference from existing studies is that BottleNet applied a learning-based method for feature compression, which was demonstrated to be very useful.

All of the works mentioned above only considered source coding (i.e., compression), while assuming reliable communication over the wireless channel, i.e., they adopted the separate principle of source and channel coding. Moreover, their design objective is to recover the intermediate feature at the edge server, either perfectly (with lossless source coding) or with tolerable distortion (with lossy compression). Nevertheless, the overall design objective is to optimize the inference task. In other words, feature compression and communication do not have their own goals but rather serve for the inference of the DNN. Motivated by the above discussion, we propose an end-to-end design approach based on joint source-channel coding, while exploiting the fault-tolerant property of DNNs. In the next two subsections, the two-ingredient, i.e., joint source-channel coding and the fault-tolerant property of DNNs, are introduced. The proposed framework will be presented in Section III.

#### B. Joint Source-Channel Coding

According to Shannon’s source-channel separation theorem \cite{15}, it is optimal to separate the design of source coding and channel coding, which is the underlying principle of modern communication systems. However, the optimality of the separate design holds only in the asymptotic limit of infinitely long source and channel blocks. In practice, considering the decoding complexity and delay constraints, joint source-channel coding will achieve better performance. As a consequence, many works have attempted to design a joint source-channel coding method \cite{16, 17}. However, given a finite bit-length budget, the hand-crafted design of joint source-channel coding is challenging.

Recently, many works have tried learning-based methods for joint source-channel coding. The work \cite{18} considered using the deep convolutional neural network to encode and decode the image over the AWGN channel, which compresses the transmitted data size. Furthermore, in \cite{19}, the authors proposed a learning-based NECST code that is robust to channel noise to transmit images. The work \cite{20} implemented...
the joint source-channel coding on the text transmission over a binary erasure channel, which uses the sequence-to-sequence learning framework to encode and decode the text.

However, the above studies focused on restoring the transmitted message at the destination, which is a challenging communication problem but is not fully aligned with the overall design objective in our considered problem. For device-edge co-inference, the final target is to run a DNN for a machine learning task. As DNNs enjoy a fault-tolerant property, the high reliability in intermediate feature transmission is not needed, which gives further room for compression. To the best of our knowledge, this is the first work that exploits this unique opportunity for feature compression.

C. Fault-Tolerance Property of Neural Networks

Fault tolerance is frequently cited as a significant property of neural networks \([21]\) since these networks contain more neurons or processing elements than necessary to solve a given problem. This property can be leveraged for reducing the communication overhead in the device-edge co-inference system by relaxing the reliability requirement of transmitting the intermediate feature to the server. For this purpose, we propose to add the channel effect directly to the neural network during the training process, to exploit the fault-tolerant capability of the DNN. Meanwhile, having the channel condition as input also improves the generalization ability to different channel conditions. As to be shown in the experiments, thanks to the fault-tolerance property, even if the channel noise corrupts the transmitted data, the DNN performance degrades gracefully.

III. PROPOSED ARCHITECTURE

We propose an end-to-end architecture, namely, BottleNet++, as shown in the box of Fig. 1. It consists of an encoder, a non-trainable channel layer, and a decoder. BottleNet++ is deployed at the splitting point of DNN, i.e., behind the feasible convolutional layer, to compress and transmit the cubelike intermediate feature tensors. The encoder and decoder are a pair of complementary lightweight CNNs, where the structure is inspired by \([11]\). The extra computation introduced by the encoder and decoder are negligible compared to the whole network. The wireless channel is modeled as a non-trainable layer in DNN and represented by a transfer function, which follows the configuration in \([19], [18]\).

A. Encoder

The encoder plays the role of feature compression and joint source-channel coding, which consists of a convolutional layer, a batch normalization layer, and an activation layer. In order to reduce the communication overhead, the encoder applies lossy compression to reduce the dimension of the intermediate feature of DNN, which can realize channel-wise reduction, width-wise reduction, and height-wise reduction. The convolutional layer uses different numbers of filters to control the output channel. The stride of the convolutional operation and the kernel size define the spatial size of the output feature, i.e., the width and the height. Concretely, the intermediate feature can be represented by a tensor as \((\text{channel}, \text{width}, \text{height})\). The four cubes presented in Fig. 1 are feature tensors with size \((C, W, H)\) or \((C', W', H')\). To compress \(H\) channels to \(H'\) channels, the convolutional layer adopts \(H'\) filters. To realize width-wise reduction from \(W\) to \(W'\) and height-wise reduction from \(H\) to \(H'\), the stride of convolutional operation is set to \(\left[\frac{W}{W'}, \frac{H}{H'}\right]\). Because the wireless channel is time-varying, in order to improve the encoder’s generalization ability, the encoder is designed to be adaptive to different channel conditions. Assuming the channel condition information is available, we will add it as a parameter to the encoder.

The convolutional layer is followed by a batch normalization layer and a Sigmoid activation function to add non-linearity features in the encoder. We use a Sigmoid function as the activation function because the output values are constrained to \([0, 1]\), which can be scaled to satisfy the transmitter output.
power constraint and further benefit to quantify data in the
digital communication system.

B. Channel Model

We consider two different types of channel models. By
assuming that the compressed feature is transmitted within
one coherent block, an AWGN model is adopted for each fading
block. To simplify the notation, the transfer function of
the Gaussian channel is written as \( f(x) = x + n \), with \( n \sim \mathcal{N}(0,\sigma^2) \). The parameter \( \sigma^2 \) captures the noise variance and the channel condition. The analog communication is adopted over the AWGN channel, i.e., symbols of the compressed feature will be directly modulated without digi-
tizing or channel encoding. We also consider a channel model commonly adopted in DNN-based studies of communication systems [20], namely, the binary erasure channel (BEC), which could model wireless channels with deep fades or burst errors. A BEC uses the bit erasure rate \( p \) to describe the channel condition, which has binary input and ternary output. When simulating the BEC, the value of the erasure bit is set to the average of the bit taking 0 and 1 in the receiver. For instance, converting a binary number to a decimal number, the value of 110 is 6. However, if the channel erases the leftmost bit, the receiver would assign the average of 010 and 110, which is 4, to this binary sequence. For the BEC model, the floating-point data must be quantized to \( n \)-bit numbers before transmission. The range of floating-point data is constrained in \([0,1]\) by Sigmoid function in encoder, so the quantizer is \( \tilde{X} = \text{round}(X \cdot (2^n - 1)) / (2^n - 1) \), where the float-point \( X \) is quantized to \( \tilde{X} \), presented by \( n \)-bit sequence. The AWGN and BEC represent two typical kinds of channel transfer functions, and the following discussion can be generalized to other channel models.

The AWGN model can be used directly in the end-to-end training because the transfer function is differentiable, and thus, back-propagation can be applied to update the weights in the training process. However, the transfer function of the BEC, as well as the quantization function, are non-differentiable, which makes the back-propagation inapplicable. In order to solve this problem, the intermediate feature directly uses the gradient of the corrupted feature as its gradient in the training process. The intermediate feature and corrupted feature are shown at the bottom of Fig. 1. In the experiment, we find that ignoring the quantization and channel corruption is feasible, and the model still converges when the value of \( p \) is not too large.

C. Decoder

The decoder deployed at the receiver is a joint source-channel decoder to map the corrupted feature to the restored feature. As shown in the box of Fig. 1, it consists of a deconvolutional layer, a batch normalization layer, and a ReLU activation function. The decoder aims to restore the bitstream to the feature tensor with the same dimension as the intermediate feature. The number of filters in the deconvolutional layer determines the output channel number. To recover \( C' \) channels from \( C \) channels, the deconvolutional layer use \( C \) filters. Width-wise and height-wise restoration can adopt \((|W/W'|,|H/H'|)\) stride in convolutional operation, where the tuple value is the same as the encoder.

In our implementation, because the width and height of the feature tensor are even numbers, we tend to set up the convolutional kernel size to \( 2 \times 2 \) and stride to \((2,2)\) in the encoder, which realize \( 2 \times \) width-compression and \( 2 \times \) height-compression, and the decoder also uses \( 2 \times 2 \) size kernel and \((2,2)\) stride.

For the encoder and decoder, we use the convolutional/deconvolutional network to compress the intermediate feature but not using a fully connected layer. This is because, although the fully connected layer has more parameters and thus more compressible, its memory and computation cost is unacceptable. Compared to traditional coding algorithms, e.g., the LDPC code [22] and the Huffman code, our method enjoys a higher computational efficiency, and the encoding and decoding models are much easier.

D. Training Strategy

As illustrated in Section III.B, the proposed BottleNet++ can be trained in an end-to-end manner with channel noise. However, direct training of the whole architecture would suffer from the problem of slow convergence. To overcome this problem, we propose a three-step approach to train our end-to-end architecture. The first step is to train the DNN, e.g., VGG16 [23] or ResNet50 [10], to reach the desired accuracy of the task. The DNN consists of many layers, like the top of Fig. 1 excluding BottleNet++. The second step is to select the splitting point to deploy the BottleNet++, and then train and update the weights of the encoder and decoder while fixing other parameters in the DNN unchanged. In this step, training the encoder and decoder in different channel conditions, i.e., with different values of \( \sigma \) in the AWGN channel, and \( p \) in the BEC will benefit its generalization ability. In the last step, we fine-tune the whole network to increase the accuracy further. In this process, all the parameters in both the DNN and BottleNet++ are updatable, and we train the whole BottleNet++ with a low learning rate.

Compared with the architecture similar to our proposal, i.e., BottleNet [11], our method explicitly models the wireless channel as a non-trainable layer in DNN, while BottleNet assumes reliable communication over the wireless channel and ignores the bandwidth expansion caused by channel coding. As BottleNet++ considers channel conditions and exploits the fault-tolerance property of DNNs, the encoded bitstream does not need to be protected by a powerful channel en-
coder, and thus it enjoys a higher compression capability compared with BottleNet. For the training process, BottleNet adopts a compression-aware training approach, which trains the DNN and the compression module commonly. In contrast, our method adopts a three-step approach to train the whole architecture, which improves the convergence rate, and when we need to adjust the compression ratio to different channel
of VGG16 and ResNet50. Note that not all layers in DNN can be used as a splitting point. For sequential DNNs like VGG and AlexNet, input signals flow layer by layer. However, the latest deep models like ResNet introduce branchy network structures rather than sequential models. So, the splitting points are different for these two kinds of networks. In our evaluation, each res-unit in ResNet is regarded as a possible splitting point, while for VGG16, each convolutional layer can be regarded as a splitting point.

B. Compression Capability Comparison

In this part, we compare the compression capability of BottleNet++ with other methods, in both the BEC and AWGN channels. Three baseline methods are considered: the method in [6] that adopts the JPEG algorithm for lossy compression, denoted as “JPEG”. The approach in [9] that quantizes floating-point data to \( n \) bit-depth and then encodes the result with Huffman coding denoted as “Quantization + Huffman”, and BottleNet [11] that encodes the intermediate feature by a neural network and uses JPEG compression-aware training to reduce the accuracy loss. Furthermore, we consider the communication overhead of transmitting the raw PNG image from device to edge, denoted as “PNG image”. All of the baseline methods assumed reliable communication over the wireless channel, i.e., with perfect channel coding. In contrast, our method explicitly takes the channel effect into account during the training process, so the encoded data are robust to the channel noise, and no extra channel coding is required.

We first compare the compression capability of these methods on different splitting points in ResNet50 and VGG16 over the BEC with the bit erasure rate \( p = 0.01 \). The accuracy loss threshold is set to 2%. In order to ensure the fairness of the comparison with our method, we apply the 1/2 rate LDPC [22] code to other methods as channel coding. The compression

\[
\text{PSNR} = 10 \log_{10} \frac{1}{\sigma^2} \quad (\text{dB})
\]  

(1)

To evaluate our proposed method, we split the DNN and compress the intermediate feature at different splitting points.
capability is illustrated via the resulting on-device computation cost and communication overhead at each splitting point, as shown in Fig. 2(a) and Fig. 2(b). The computation cost approximated by the number of floating-point calculations in the convolutional layers deployed on the device. The extra computation introduced by BottleNet++ is negligible and thus is ignored. The size of the compressed feature determines the communication overhead.

From both Fig. 2(a) and Fig. 2(b) we note that at any splitting point, BottleNet++ achieves the lowest communication overhead. It realizes up to 256× bit compression ratio in the last convolutional layer of ResNet, where BottleNet++ compresses 8192 32bit-floating numbers to 128 8bit-integers. A higher compression ratio means the possibility of earlier model splitting, which means a lower on-device computation cost with a target communication overhead. In other words, BottleNet++ achieves a better tradeoff among on-device computation and the communication overhead. For other methods, only when splitting at very deep layers, the communication overhead of the intermediate feature can be lower than transmitting the original PNG image. This aspect is illustrated in Table I, which shows the on-device computation of different methods. In any case, our BottleNet++ achieves the minimum on-device computation. For example, for VGG16, BottleNet++ can reduce the on-device computation by \( \sim 2 \times \) and \( \sim 3 \times \) for VGG16 and ResNet50 in the AWGN channel compared with other methods.

C. Generalization Ability and Robustness Analysis

In the real communication scenario, the channel condition is time-varying, and in this part, we test the generalization ability of BottleNet++ when the channel state changes. We conduct the experiment based on the ResNet50 model deployed behind the last convolutional layer, with 64× compression ratio for the AWGN channel and with 256× bit compression ratio for BEC, respectively. We evaluate the performance of BottleNet++ in three different cases:

- **Case 1:** The encoder knows the channel condition information, bit erasure rate or PSNR, in both the training and testing processes.
- **Case 2:** The encoder only knows the channel condition information in the training process. In the testing process, the encoder assumes the channel condition to be 15dB in the AWGN channel or 0.125 in the BEC.

2Note that this gives a performance upper bound for the baseline methods, and thus in practice, the performance gain of BottleNet++ will be more prominent. Current channel coding methods are not effective for low-latency short-packet transmission, e.g., to transmit compressed feature, and new codes will be needed.

---

Fig. 3. Accuracy degradation of BottleNet++ over the (a) AWGN channel and (b) BEC.
• Case 3: The encoder does not know the channel condition information in neither the training nor the testing process. Fig. 3(a) and Fig. 3(b) presents the accuracy loss in different channel conditions of the three cases. Case 1 achieves the highest accuracy under any channel condition. Case 2 is very close to case 1, which shows the robustness of BottleNet++ to the variation of channel conditions. Case 3 has a noticeable accuracy drop compared to Case 1, which means that considering channel conditions during encoding can improve the generalization ability. Remarkably, the performance of BottleNet++ is robust to channel variations in all three cases. Specifically, the accuracy drops less than 1% when PSNR changes from 25 dB to 10 dB in the AWGN channel, or when the bit erasure rate changes from 0.01 to 0.15 in BEC.

V. CONCLUSIONS

In this paper, we propose an end-to-end deep learning architecture, named BottleNet++, for device-edge co-inference with resource-constrained mobile devices. By exploiting the strong sparsity and the fault-tolerant property of the intermediate feature in a deep neural network, BottleNet++ achieves a much higher compression ratio than existing methods, which leads to a significant reduction in the communication overhead, which makes it feasible to split a neural network in the earlier layer to reduce the on-device computation. For the communication theoretic aspect, our study casts new light on the two fundamental problems in the setting of device-edge co-inference: What to transmit? How to transmit? The results indicate that transmitting highly compressed features with the analog communication system becomes attractive for edge-assisted inference.

REFERENCES

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Region-based convolutional networks for accurate object detection and segmentation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142–158, 2015.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," Open Journal of Cloud Computing (OJCC), vol. 2, no. 1, pp. 16–24, 2015.

[4] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “Model compression and acceleration for deep neural networks: The principles, progress, and challenges,” IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 126–136, 2018.

[5] A. Eshratifar, M. S. Abrishami, and M. Pedram, “Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services,” arXiv preprint arXiv:1801.08618, 2018.

[6] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, “Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms,” in 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). IEEE, 2018, pp. 1–6.

[7] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, “Neurossegnet: Collaborative intelligence between the cloud and mobile edge,” in ACM SIGARCH Computer Architecture News, vol. 45, no. 1. ACM, 2017, pp. 615–629.

[8] H. Choi and I. V. Bajić, “Deep feature compression for collaborative object detection,” in 2018 25th IEEE International Conference on Image Processing (ICIP). IEEE, 2018, pp. 3743–3747.

[9] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, “Jalad: Joint accuracy-aware latency-aware tradeoff for edge-cloud execution,” in 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2018, pp. 671–678.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[11] A. E. Eshratifar, A. Esmaeli, and M. Pedram, “BottleNet: A deep learning architecture for intelligent mobile cloud computing services,” arXiv preprint arXiv:1902.01000, 2019.

[12] H. Choi and I. V. Bajić, “Near-lossless deep feature compression for collaborative intelligence,” in 2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP). IEEE, 2018, pp. 1–6.

[13] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, “Improving device-edge cooperative inference of deep learning via 2-step pruning,” arXiv preprint arXiv:1903.03472, 2019.

[14] Y. Matusbara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, “Distilled split deep neural networks for edge-assisted real-time systems,” in Proc. the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges. ACM, 2019, pp. 21–26.

[15] C. E. Shannon, “A mathematical theory of communication,” Bell system technical journal, vol. 27, no. 3, pp. 379–423, 1948.

[16] A. Goldsmith, “Joint source/channel coding for wireless channels,” in 1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century, vol. 2. IEEE, 1995, pp. 614–618.

[17] F. Zhai, Y. Eisenberg, and A. K. Katseggelos, “Joint source-channel coding for video communications,” Handbook of Image and Video Processing, 2005.

[18] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz, “Deep joint source-channel coding for wireless image transmission,” IEEE Transactions on Cognitive Communications and Networking, 2019.

[19] N. Farsad, M. Rao, and A. Goldsmith, “Deep learning for joint source-channel coding of text,” in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018, pp. 2326–2330.

[20] K. Choi, K. Tawawadi, A. Grover, T. Weissman, and S. Ermon, “Neural joint source-channel coding,” in International Conference on Machine Learning, 2019, pp. 1182–1192.

[21] M. D. Emmerson and R. I. Damper, “Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application,” IEEE Transactions on Neural Networks, vol. 4, no. 5, pp. 788–793, 1993.

[22] A. Shokrollahi, “Ldpc codes: An introduction,” Digital Fountain, Inc., Tech. Rep., vol. 2, p. 17, 2003.

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

[24] A. Krizhevsky, G. Hinton et al., “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.

[25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.