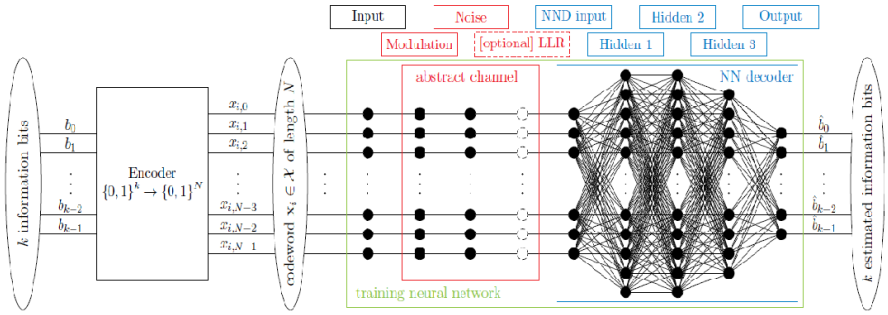
University of Stuttgart
Germany

Institute of Telecommunications
Professor Dr.-Ing. Stephan ten Brink

Seminar Thesis

Learning to Decode-Curse of Dimensionality

Vivek Ramalingam Kailasam



Date of hand out: May 15, 2018
Date of hand in: July 19, 2018
Supervisor: Jakob Hoydis
Stephan ten Brink
Sebastian Dörner
Sebastian Cammerer

Abstract

The main shortcoming of the deep learning-based channel decoding is the curse of the dimensionality. To overcome this shortcoming, the neural network must learn some form of decoding algorithm. This learning is possible only for structured codes not random codes. In this project work, attempt for construction of such decoder is done using convolutional neural network (CNN) and residual network inception model. The illustration of decision of hyperparameters for NN is explained and comparison of the CNN, residual neural network inception model and the dense layer network is done. For higher values of message length in the codeword, the performance of the CNN and residual network inception model decoders are explained.

Title page image: The message of k bit length is encoded into the N-bit length codeword at the transmitter. These encoded words are modulated and transmitted over a channel which is noisy. This noisy codeword is received at the receiver. The decoder has to decode the received codeword and find out the original message which was transmitted.

Contents

Acronyms

CNN	Convolution Neural Network
NN	Neural Network
SNR	signal-to-noise-ratio
BPSK	binary-phase-shift-keying

Notations

1 Introduction

Artificial Intelligence which is one of the current hot topics, has extended its hand in the field of communications too. The decoding process of the communication can be carried out using the deep neural networks. In [1], the deep learning-based channel decoding was performed for short codes, and the following were stated. Practically it is very difficult to fully train the neural networks (NN), since the deep learning-based channel decoding is doomed by the curse of dimensionality. For example, when code has the length N=100 and rate r = 0.5, there are totally 2^{50} codeword exists, training all these 2^{50}codewords is costly and time consuming. But if the NN learns some form of algorithm to decode, then it would be possible to decode the whole codebook only by training the small part of the whole codebook.

The NN will have the ability to learn the decoding algorithm only if the code has some structure which is based on some simple encoding rules as in the convolutional codes or algebraic codes, rather than being random without any structure. In [1], the dense NN were used for decoding the structure codes. The goal of this study thesis is to improvise the work done in [1].

In this study thesis, the following are done. Different NN structures like Convolutional Neural Networks (CNN) and Residual-Inception network are performed for the same task of decoding. The learning process in the construction of the NN structure was discussed. The hyperparameter tuning of the NN and the training procedure of the networks are also discussed. Finally, the comparison between the dense NN structure of [1] and NN structures of this project are done.

2 Fundamentals

In this section, the general concepts of neural networks and brief overview of CNN and Residual-Inception will be seen.

2.1 Convolutional Neural Network

Basic concepts of CNN with reference to [2] is given in this subsection. There are 1D, 2D and 3D convolutions available, since this study thesis project uses 1D convolution, CNN is explained using 1D convolution. The main advantage of CNN when compared with fully connected layer is that CNN uses fewer parameters, so it is easy to train the NN. The input layer of length n is convolved with filter or kernel of size w and then activation function is applied. After convolution, the feature map of size n-w+1 is generated. The filter will have weights and bias. After convolution, pooling can be applied. Pooling is the nonlinear down sampling technique, it is mainly used for reducing the size of the feature map. Average pooling and max pooling are types of pooling possible. In max pooling the max value in the pool is extracted and in the average pooling, the average of the pool is extracted.

2.2 Residual-inception Network

On referring [2], some fundamental concepts of residual-inception network model is explained. The residual network helps to avoid the degradation problem in the very deep network. Usual neural network, receives the inputs from one layer (say layer 1), and weights and bias values of second layer (layer 2) are applied to the inputs, then activation function is applied, and output of the activation function is given to the layer 2. Likewise, the third layer (layer 3) gets the input from layer 2, apply the weights and activation function and gives the output. But in the residual network, the input of the layer 1 is added to the values (only weights are applied in these values) of output of layer 2 before applying the activation function. And then the output of the activation function is given to layer 3. Refer to the figure 2.1.

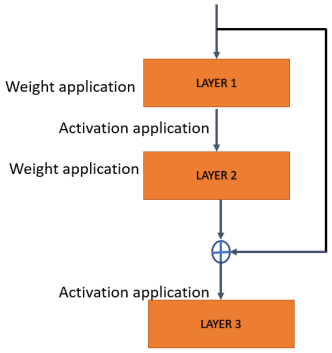


Figure 2.1: An example of Residual network [2]

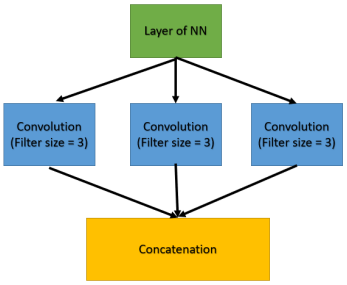


Figure 2.2: An example of inception model [2]

In inception model filter level sparsity blocks were used. Usually in the Convolution Neural Network (CNN), between 2 layers, only one convolution with a same filter size is done, but the inception model uses many filter layers with different filter sizes, which takes the input from the same layer.

The output of these filter layers were concatenated, and this concatenated layer can be used as input for next layer. Refer to the figure 2.2. The combination of both residual network and inception network is the residual-inception network.

3 Deeping Learning Setup for Channel Coding

In [1], the following setup was used. The decoder which is made with neural networks are meant for decoding the noisy codewords. The message of k bit length is encoded into the N-bit length codeword at the transmitter. These encoded words are modulated and transmitted over a channel which is noisy. This noisy codeword is received at the receiver. Now the decoder has to decode the received codeword and find out the original message which was transmitted. One main advantage of this task is that the labelled data collection for training is not as hard as other machine learning tasks. Since the signal which are dealt with here are man-made signals, it is easy to generate the training samples required.binary-phase-shift-keying (BPSK) modulation and additive white gaussian noise (AWGN) channel is used. Additional layers of Neural Network (NN) are used for modulation and noise addition, and these layers don't have trainable parameters. Activation function of the output layer is always sigmoid, since the output of sigmoid will lie between 0 to 1, it can be considered as the probability of "1" being transmitted. During validation, when the output, which has the probability, is close to the corresponding correct output value, then loss will be less, else the loss will be high. Mean square error (MSE) and binary cross entropy (BCE) are the examples of such loss functions. For this study thesis project, BCE is used. Log likelihood ratio (LLR) also can be implemented as an additional layer as shown in the figure 2, but for this study thesis project, LLR is omitted. The performance of the neural network decoder is measured from the signal to noise ratio (SNR) for which the bit error rate (BER) is computed.

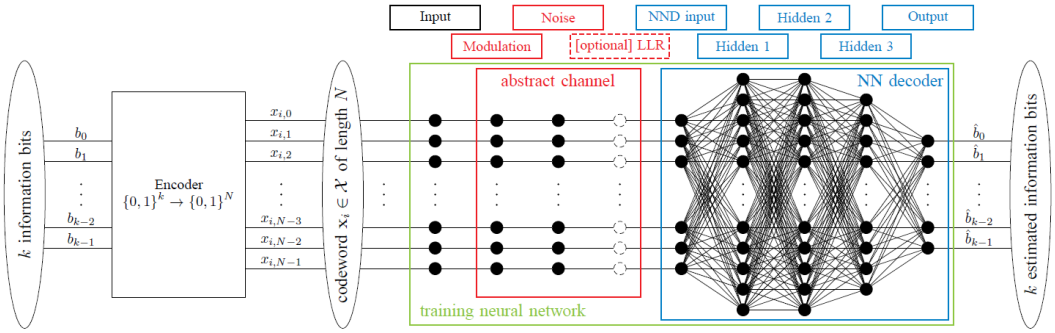


Figure 3.1: *Deep* learning setup for channel coding

4 Learning Process

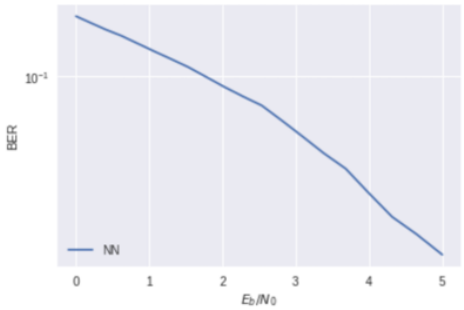
As mentioned earlier, the deep-learning based channel decoding is doomed by the curse of dimensionality. When the length of the message k is increased, the computation complexity increases drastically. So, the approach to overcome this problem is designing the NN structure and determining the hyperparameters using the smaller values of k and then that NN with same hyperparameters is used for code which has higher value of k. This is based on the assumption that, if the neural network doesn't seem to work for small value of k, then that neural network will not work for large values of k. Point to be noted that, there is no guarantee that if the NN works for small value of k, then it will work for higher values of the k. One of the tasks of this study thesis is that to determine till what value of k, the NN will work.

4.1 Using CNN

On using the below code, which shown in the figure 4.1a, which has two layers of CNN network. First layer has 64 channels, kernel size used is 5 and max pooling with pool size 3 and 1 step strides is done. The second layer has 32 channels, kernel size used is 3 and max pooling with pool size 2 and 1 step strides is done. The output of the second layer is flattened. After flattening one dense layer with 500 nodes is placed. Finally, the output layer with k nodes is placed. Let us call this code set up shown in the figure 4.1a as neural network A.

```
#######REPLACE WITH CNN Structure here#################
decoder_layers = [Reshape((N,1),input_shape=(N,))]
decoder_layers.append(Conv1D(64,kernel_size = 5,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 3, strides =1))
decoder_layers.append(Conv1D(32,kernel_size = 3,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 2, strides =1))
decoder_layers.append(Flatten())
decoder_layers.append(Dense(500, activation='relu'))
decoder_layers.append(Dense(k, activation='sigmoid'))
decoder = compose_model(decoder_layers)
```

(a) Code for decoder model in NN A



(b) Performance curve by NN A for k=8, N=16

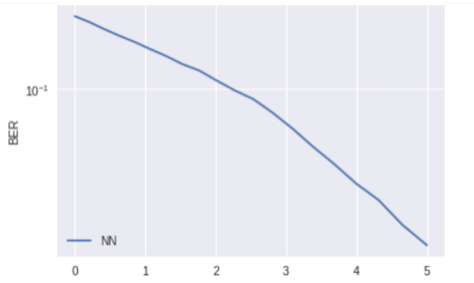
Figure 4.1: NN A

On implementing the neural network A, with N=16, k=8, number of epochs = 4000, training signal-to-noise-ratio (SNR) = 1 dB. The performance curve looks like the curve in the figure 4.1b

When modified the neural network A by just changing the channel size of the CNN layers. Like first layer have 32 channels and second layer have 16 channels as shown in the figure 4.2a. Let us name this construction as neural network B. On implementing the neural network B, with N=16, k=8, number of epochs = 4000, training SNR = 1 dB. The performance curve looks like the curve in the figure 4.2b. On the comparing the performance of the NN A with NN B, NN A looks better. So the channel sizes of NN A is taken into consideration and channel sizes in the NN B is omitted.

```
#######REPLACE WITH CNN Structure here#################
decoder_layers = [Reshape((N,1),input_shape=(N,))]
decoder_layers.append(Conv1D(32,kernel_size = 5,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 3, strides =1))
decoder_layers.append(Conv1D(16,kernel_size = 3,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 2, strides =1))
decoder_layers.append(Flatten())
decoder_layers.append(Dense(500, activation='relu'))
decoder_layers.append(Dense(k, activation='sigmoid'))
decoder = compose_model(decoder_layers)
```

(a) Code for decoder model in NN B



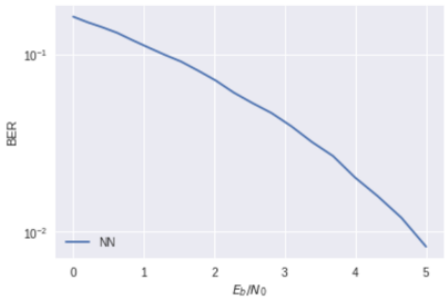
(b) Performance curve by NN B for k=8, N=16

Figure 4.2: NN B

In neural network A there is only one dense layer between the CNN layers and output layer. Now one more fully connected layer with 200 nodes is placed before the output layer as shown in the figure 4.3a. Let us name this construction as neural network C. On implementing the neural network C, with N=16, k=8, number of epochs = 4000, training SNR = 1 dB. The performance curve looks like the curve in the figure 4.3b. The performance of NN C is far better than that of NN A, the curve moves down crossing 10^{-2}. So, it is better to have two dense layers instead of just one dense layer.

```
############REPLACE WITH CNN Structure here#################
decoder_layers = [Reshape((N,1),input_shape=(N,))]
decoder_layers.append(Conv1D(64,kernel_size = 5,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 3, strides =1))
decoder_layers.append(Conv1D(32,kernel_size = 3,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 2, strides =1))
decoder_layers.append(Flatten())
decoder_layers.append(Dense(500, activation='relu'))
decoder_layers.append(Dense(200, activation='relu'))
decoder_layers.append(Dense(k, activation='sigmoid'))
decoder = compose_model(decoder_layers)
```

(a) Code for decoder model in NN C



(b) Performance curve by NN C for k=8, N=16

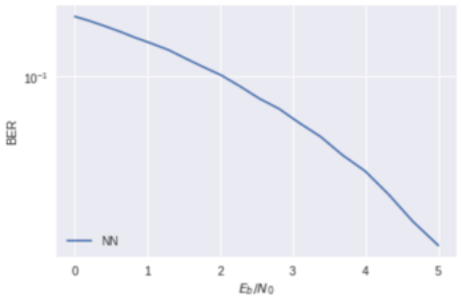
Figure 4.3: NN C

In NN C, there are two CNN layers and two dense layers before output layer. Now construction of neural network D is done as using three CNN layers and two dense layers as shown in figure 4.4a. The third CNN layer has 24 channels, its kernel size is 3 and then max pooling is done

with pool size as 2 and 1 step strides. On implementing the neural network D, with N=16, k=8, number of epochs = 4000, training SNR = 1 dB. The performance curve looks like the curve in the figure 4.4b. The performance curve didn't cross the 10^{-2} BER, thus here on increasing the number of CNN layers to three, doesn't improve the performance. Thus, only two CNN layers is placed.

```
###########REPLACE WITH CNN Structure here###################
decoder_layers = [Reshape((N,1),input_shape=(N,))]
decoder_layers.append(Conv1D(64,kernel_size = 5,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 3, strides =1))
decoder_layers.append(Conv1D(32,kernel_size = 3,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 2, strides =1))
decoder_layers.append(Conv1D(24,kernel_size = 3,activation='relu'))
decoder_layers.append(MaxPooling1D(pool_size = 2, strides =1))
decoder_layers.append(Flatten())
decoder_layers.append(Dense(500, activation='relu'))
decoder_layers.append(Dense(200, activation='relu'))
decoder_layers.append(Dense(k, activation='sigmoid'))
decoder = compose_model(decoder_layers)
```

(a) Code for decoder model in NN D



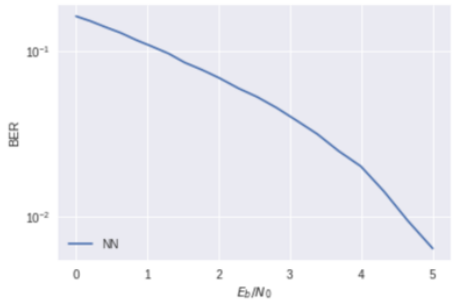
(b) Performance curve by NN D for k=8, N=16

Figure 4.4: NN D

Construction of NN F is made as same as construction of NN C, but with the difference that average pooling is done instead of max pooling. Refer to the code shown in the figure 4.5a. On implementing the neural network D, with N=16, k=8, number of epochs = 4000, training SNR = 1 dB. The performance curve looks like the curve in the figure 4.5b. The performance of the NN C and NN F are almost the same. Now NN C and NN F are trained and validated for N=32 and k=16, with epochs = 4000 and training SNR = 1dB, the performance of NN C and NN F are given in the figure 4.6. In the comparison of NN C and NN F with performance curves, it is identified that the performance of NN F is better.

```
###########REPLACE WITH CNN Structure here##########
decoder_layers = [Reshape((N,1),input_shape=(N,))]
decoder_layers.append(Conv1D(64,kernel_size = 5,activation='relu'))
decoder_layers.append(AveragePooling1D(pool_size = 3, strides =1))
decoder_layers.append(Conv1D(32,kernel_size = 3,activation='relu'))
decoder_layers.append(AveragePooling1D(pool_size = 2, strides =1))
decoder_layers.append(Flatten())
decoder_layers.append(Dense(500, activation='relu'))
decoder_layers.append(Dense(200, activation='relu'))
decoder_layers.append(Dense(k, activation='sigmoid'))
decoder = compose_model(decoder_layers)
```

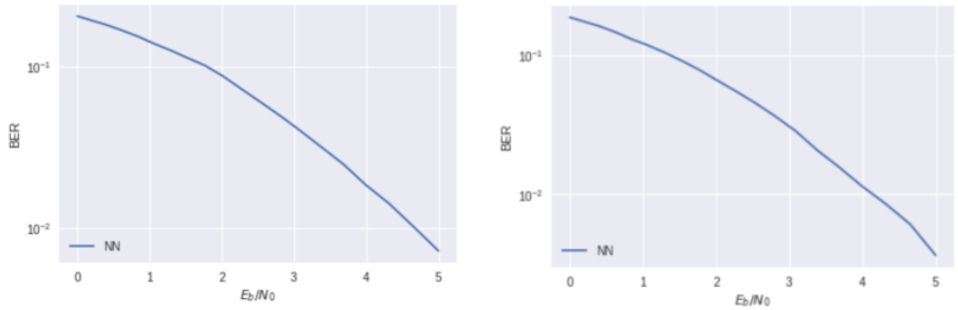
(a) Code for decoder model with NN F



(b) Performance curve by NN F for k=8, N=16

Figure 4.5: NN F

Now on trying the channel decoding with NN F for higher value of k, the curve shown in the figure 4.7 is attained. Figure 4.7 shows the performance curve on implementing NN F with k = 20, N=32, training SNR =3 dB and epochs = 2000. The performance curve looks better taking the long length of the codeword N and long length of k.



(a) Performance curve by NN C for k=16, N=32 (b) Performance curve by NN F for k=16, N=32

Figure 4.6: NN F and C

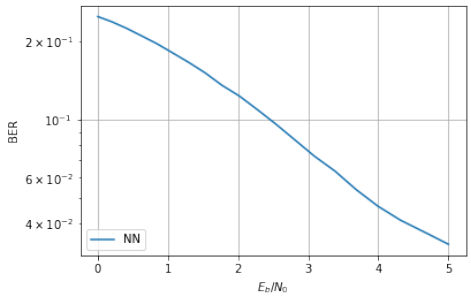


Figure 4.7: Performance curve by NN F for k=20, N=32

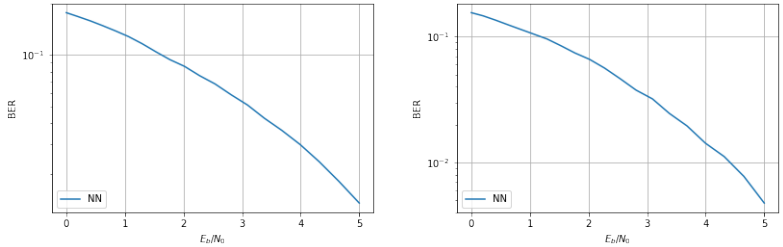
4.2 Using Residual Inception Model Network

The residual inception model as shown in figure 4.8 is implemented. Let us name this as ResNet 1. In Resnet 1, four convolutions (via 4 branches) are done, number of filters used in each convolution is 16. 2,4,8,16 are the sizes of kernel used in the branch 1,2,3,4 respectively. In the first iteration, the four branches are concatenated and then added with input layer of any branch (All branches have same input layer). This layer which is added, is the output of the that iteration, and it is given as input to branches in the next iteration. Value of NStages variable gives the number of iterations. After the last iteration, the output layer of the last iteration is flattened. The flattened layer is followed by two dense layers with nodes 60 and 35 and finally after these layers, output layer with k neurons are placed.

On implementing the neural network Resnet 1, with N = 16, k = 8, Nstages = 1, number of epochs = 4000, training SNR = 3 dB. The performance curve looks like the curve in the figure 4.9a On attempt to further improve the NN, for Resnet 1, value of Nstages is made as 2, the performance of this setup is given in figure 4.9b. The performance was improved by making the Nstages = 2. Proceeding with this neural network Restnet 1 with Nstages = 2, for higher values of k, the computation time is observed to be very high. For k = 20 and N = 32, it took 278s for one epoch, for 2000 epochs it would take around 6.5 days. Thus, this structure of ResNet 1 is omitted, only by considering the computation expense.

```
#--------------ResNet decoder--------
#define conv net
Nstages=1 #5
incept_dim=[16, 16, 16, 16, 16, 16, 16]
incept_size=[2, 4, 8, 16, 32, 64, 32]

ch_out=Reshape((N,1))(n1)
dec_in=Input(shape=(N,))
lt_out=Reshape((N,1))(dec_in)

for i in range(0,Nstages):
    #inception block
    branch1 = Conv1D(filters=incept_dim[0], kernel_size=incept_size[0], activation='relu', padding='same')(lt_out)
    branch2 = Conv1D(filters=incept_dim[1], kernel_size=incept_size[1], activation='relu', padding='same')(lt_out)
    branch3 = Conv1D(filters=incept_dim[2], kernel_size=incept_size[2], activation='relu', padding='same')(lt_out)
    branch4 = Conv1D(filters=incept_dim[3], kernel_size=incept_size[3], activation='relu', padding='same')(lt_out)
    lt=Concatenate()([branch1, branch2, branch3, branch4])

    lt_short=lt_out

    lt_out=Add()([lt, lt_short])

lt_out=Flatten()(lt_out)
l_out_pre1 = Dense(60,activation='relu')(lt_out)
l_out_pre2 = Dense(35,activation='relu')(l_out_pre1)
l_out=Dense(k,activation='sigmoid')(l_out_pre2)
```

Figure 4.8: Code for ResNet 1



(a) Performance curve by ResNet 1 for k=8, N=16, Nstages=1

(b) Performance curve by ResNet 1 for k=8, N=16, Nstages=2

Figure 4.9: ResNet 1

On an attempt to design a neural network with less computational expense and reasonable performance, neural network Resnet 2 is formed as shown in the figure 4.10. In Resnet 2, instead of having 16 number of filters in each convolution operation, only 8 number of filters are used and instead of 4 branches per iteration, only 2 branches per iteration were used. And also in each iteration, different set of branches were executed. In first iteration, branch 1 and branch 2 use kernel sizes of 2 and 4 respectively, whereas in second iteration, branch 1 and branch 2 use kernel sizes of 8 and 16 respectively. The three dense layers of 80, 60 and 35 are used.

On implementing the neural network Resnet 2, with N = 16, k = 8, NStages = 2, number of epochs = 4000, training SNR = 3 dB. The performance curve looks like the curve in the figure 4.11. It gives a good performance, curve goes down crossing 10^{-2}.

On implementing Resnet 2 for higher values of k with Nstages = 2, the computation time has reduced reasonably, on relative comparison with ResNet 1. For ResNet 2, it took 74s for one epoch, with k=20, N=32, training SNR = 3dB. The computation time might be relatively good, but it is also high. So ResNet 2 is implemented for k = 16 and k=18, with N =32, epochs = 1200 and SNR = 3 dB. The performance curve k=16 and k = 18 are shown in figure . Note: the time value given for an epoch in this subsection, is based on the training happened in the dedicated server provided (not on colab).

```
#-------------ResNet decoder--------
#define conv net
Nstages=2 #5
incept_dim=[8, 8, 8, 8, 8, 8, 8]
incept_size=[2, 4, 8, 16, 32, 64, 32]

ch_out=Reshape((N,1))(n1)
dec_in=Input(shape=(N,))
lt_out=Reshape((N,1))(dec_in)

for i in range(0,Nstages):
    #inception block
    if i == 0:
        branch1 = Conv1D(filters=incept_dim[0], kernel_size=incept_size[0], activation='relu', padding='same')(lt_out)
        branch2 = Conv1D(filters=incept_dim[1], kernel_size=incept_size[1], activation='relu', padding='same')(lt_out)
    if i == 1:
        branch1 = Conv1D(filters=incept_dim[2], kernel_size=incept_size[2], activation='relu', padding='same')(lt_out)
        branch2 = Conv1D(filters=incept_dim[3], kernel_size=incept_size[3], activation='relu', padding='same')(lt_out)
    if i == 2:
        branch1 = Conv1D(filters=incept_dim[2], kernel_size=incept_size[2], activation='relu', padding='same')(lt_out)
        branch2 = Conv1D(filters=incept_dim[4], kernel_size=incept_size[3], activation='relu', padding='same')(lt_out)
    if i == 3:
        branch1 = Conv1D(filters=incept_dim[2], kernel_size=incept_size[2], activation='relu', padding='same')(lt_out)
        branch2 = Conv1D(filters=incept_dim[3], kernel_size=incept_size[3], activation='relu', padding='same')(lt_out)

    lt=Concatenate()([branch1, branch2])

    lt_short=lt_out

    lt_out=Add()([lt, lt_short])

lt_out=Flatten()(lt_out)
l_out_pre1 = Dense(80,activation='relu')(lt_out)
l_out_pre2 = Dense(60,activation='relu')(l_out_pre1)
l_out_pre3 = Dense(35,activation='relu')(l_out_pre2)
l_out=Dense(k,activation='sigmoid')(l_out_pre3)
```

Figure 4.10: Code for ResNet 2

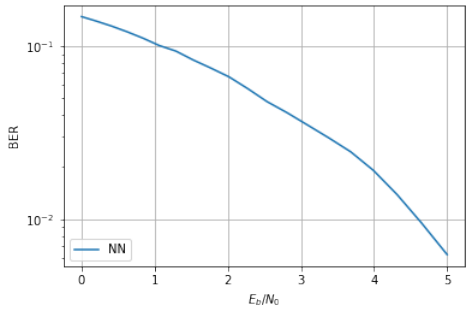
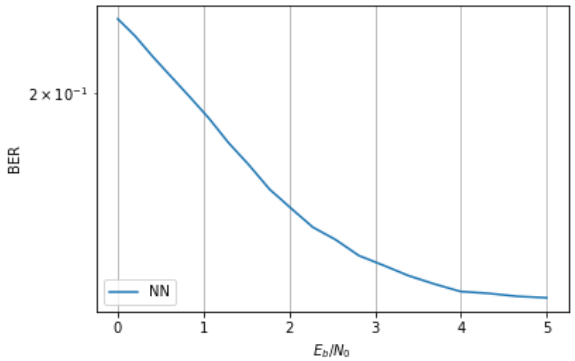
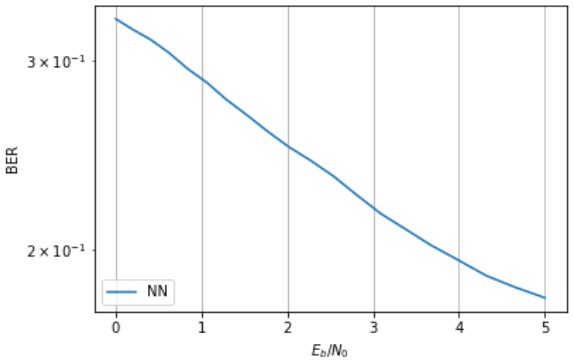


Figure 4.11: Performance curve by ResNet 2 for k=8, N=16, Nstages=2



(a) Performance curve by ResNet 2 for k=16, N=32, Nstages=2



(b) Performance curve by ResNet 2 for k=18, N=32, Nstages=2

Figure 4.12: Performance curve for Resnet 2

5 Comparison of Different Neural Network Structures

The table 1 shows the comparison of decoders structured using dense layer network model, CNN model, residual network inception model with respect to number of trainable parameters and time taken for one epoch. In this comparison, dense layer decoder has three hidden layers with 128, 64 and 32 nodes, the CNN decoder has structure as shown in the NN F and the residual network inception model decoder has the structure shown in the ResNet 2 with Nstages = 2. The data given in the below table is based on the NN training happened in the Colab notebook (not in dedicated server). In Colab notebook, hardware accelerator mode is set as GPU. The table 1 is based on the parameter values of N=32, k=20, training SNR = 3dB, batch size =256.

Model	No of trainable parameters	Approx Time taken for one epoch (seconds)
dense layer network	15220	26
CNN	479280	40
residual network inception	51907	110

Table 5.1: Comparison of decoders based on different NN structures

6 Conclusion

In the process of constructing a decoder with CNN and residual network inception model, both of them showed good performance for the shorter values of message length k. When value of the k gets higher, performance of the both models get degraded. The main problem with the residual-inception NN is the computation time. Since the computation time is very high for higher values of k, it has become very difficult to explore the full capability of the NN (Since reasonable number of epochs is required for a neural network to learn the decoding). The residual-inception networks performance for k = 16 and k = 18, is not that impressive as shown in the figure 4.12 . The CNN showed moderate performance for higher values of k (= 20) as shown in figure 4.7. Overall, the CNN works better than the residual-inception model for the higher values of k with the limited computational capability. May be if the computation capability is more, residual-inception model may show better performance.

Bibliography

[1] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," *CoRR*, vol. abs/1701.07738, 2017. [Online]. Available: http://arxiv.org/abs/1701.07738 (p. 1)

[2] S. Pouyanfar, S. C. Chen, and M. L. Shyu, "An efficient deep residual-inception network for multimedia classification," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, July 2017, pp. 373–378. DOI: 10.1109/ICME.2017.8019447 (p. 2)

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Das elektronische Exemplar stimmt mit den gedruckten Exemplaren überein.

Stuttgart, July 19, 2018

Vivek Ramalingam Kailasam