ELEC5306 Video Intelligence and Compression Project Report 3

Hangyu Chen (490075948)

Abstract

With the development of deep learning and the computer version, convolutional neural networks (CNNs) got some achievements in the image processing area. However, just like what Feng (2017) [1] said, the most successful achievements for CNNs are in high-level vision tasks (including object classification and recognition). Low-level problems such as image/video compression are rarely been solved.

In this project, an end-to-end convolutional neural network was used to complete the video compression problem. The aims are:

- Meet the requirements of the benchmark test (including the maximum complexity of the network and the minimum compression ratio)
- Get the compressed-decoded Peak-Signal-to-Noise-Ratio (PSNR) value as higher as possible after training the network in 30 min

The best network is chosen after comparing the results (**Section III**) with different structure networks and hyper-parameters. Then, their usage and effects will be analyzed in **Section II**. In **Section IV**, the discussion is mainly focused on how the network is improved step-by-step and why some of the methods will cause negative effects on this low-level task. And a table containing all the results is in the **Appendix**.

In this project, rather than developing a more complex CNN, the question is simplified because:

- There is no specific compression ratio, as long as it is larger than 70% to meet the benchmark.
- There is no need for a quantization process, which means the compressed images could directly be used for decoding.

We got the best result which is the final accuracy (represented in PSNR) for the test set as **36.82dB**. All the results with their parameters could be seen in the **Appendix**.

Section I: Aims and Background

It has been claimed that more than 80% of the internet traffic is made up of video content nowadays [5], and it is still increasing especially due to some popular software such as TikTok and YouTube.
However, large video files require more storage space and consume more bandwidth, which will slow down the speed of uploading and file transfer. According to iStock (2022) [3], a simple 1080 HD video clip can take up approximately 11 GB of space per minute of video, which could be considered a high cost.

In this case, it becomes essential to build up a method to generate higher quality frames with a limited bandwidth [5]. And based on the powerful ability of CNN, an end-to-end network is built up so that the images could be compressed as frames with a **compression ratio of 70%** and the decoded images (8-bit) will have an average PSNR of about **36.8dB**, which is high enough for human eyes so that the differences cannot be told.

This project mainly focused on improving the baseline network by changing the structure and choosing proper hyper-parameters. The **main process** includes but is not limited to:

1. Load the training and test datasets files from Google drive and keep the same data augmentation as in the template which crops the size of the images into (256 x 256).
2. Train the baseline network and record the result.
3. Within each modified model, changing the structure or the value of hyper-parameters such as batch size, learning rate, etc. and saved the model with the best performance (minimum loss and highest accuracy).
4. Using curves to show the Loss and accuracy (PSNR) for each epoch during training. And visualize the results by comparing the input and output images for the baseline and best performance models.
5. Compare the differences and similarities in the structure between these models and analyze the effect of each change.

Related Works

Some related works have been referred to in this project. Feng Jiang, et.al [1] combined two CNNs into one system. The first CNN is used to compress the images so that the optimal features were extracted and it also preserved the structural information, the second CNN is used to reconstruct the compressed images in high quality. Besides, Guo Lu. Et.al [5] improved the compressed system so that it could use the residual information for video compression. And the network used generalized divisive normalization (GDN) and IGDN is used.

In our network, the network structure is referred to [1] and there will be an analysis of why Batch-Normalization will induce a negative effect on image compression.

As for the results, the **compression ratio will be 70.36%** and the decoded images (8-bit) will have an average PSNR of about **36.8dB.**

Section II: Methodology

To achieve the compression task, we have to implement the techniques/algorithms needed first. In this part, the techniques used in each network with background knowledge are introduced in detail and some of the comparisons are mentioned with the results in **Appendix**.

1. Network structure

The structure of the network used in the project is shown in the figure below. It contains two parts of CNNs, the first part (encoder) contains 5 convolution layers which will compress the input images of 3*256*256 into 56*32*32 by extracting the features. The second part (decoder) contains 3 deconvolution layers which will reconstruct the compressed images into their original size.

Rather than developing a complex network, this self-defined network aims to simplify as long as it meets the minimum requirement of the compression ratio (70%).

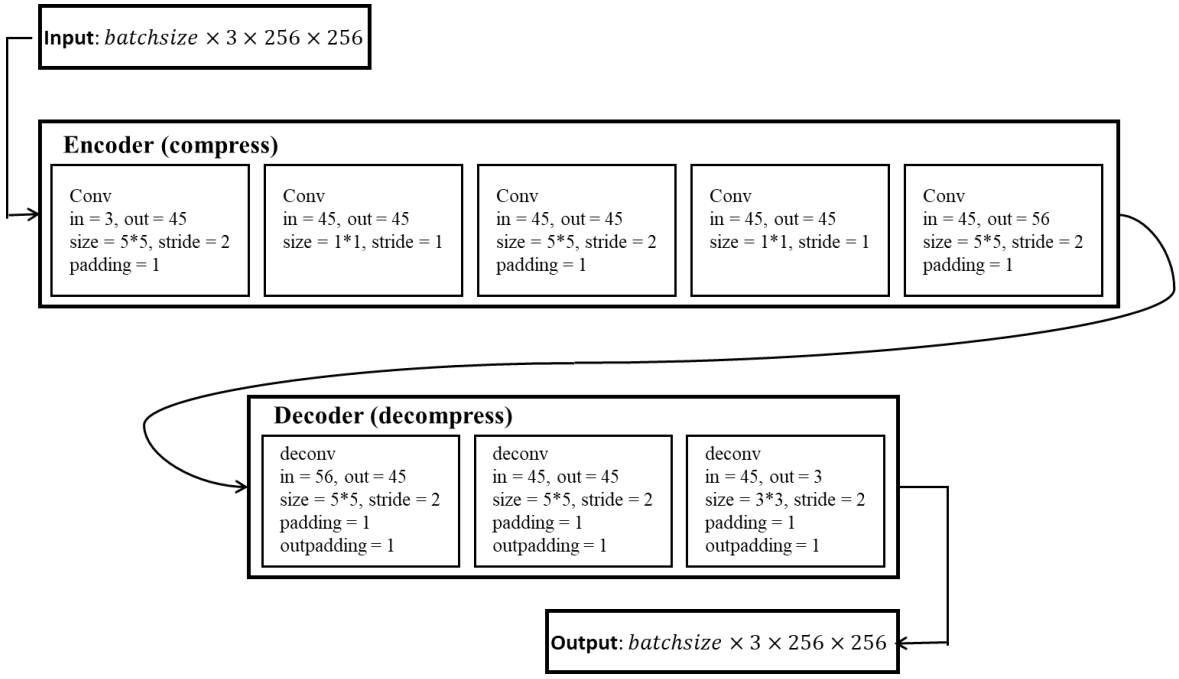


Figure 1 Network Structure

It could be figured out that except for the normal convolutional layers used in the baseline network, two 1*1 kernel-size convolution layers are used which is considered a “bottleneck”.

The 1*1 convolution kernels were firstly used in the Network in Network (NIN), and some classical networks such as Resnet and GoogleNet used the Inception block which contains a 1*1 kernel to reduce the dimension and minimize the parameters needed.

However, in this self-defined network, 1*1 kernel convolution is used which could act similarly with a

full-connected layer but in 2-dimension (figure below). And it has been claimed by Yan LeCun [6]. That there are actually no “fully-connected layers” in Convolutional Nets and the items that act as “fully connected” are convolution layers with 1x1 convolution kernels and a full connection table.

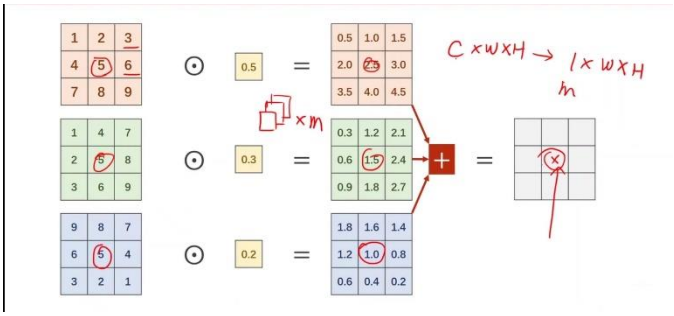


Figure 2 1*1 kernel convolution

In this case, the linearity after each convolution layer has been increased so that the performance will be improved (it can be proved by the results of **Modified 2** and **Modified 5** shown in **Appendix**).

2. Data Augmentation

Data Augmentation is a technique to increase the diversity of your training set by applying random (but realistic) transformations [2]. If you have a lot of parameters, you need to give your model a sufficient proportion of samples. Again, the number of parameters you need is proportional to the complexity of your task. In this project, except for **Random Cropping** and **Center Cropping** of the data in the training and test set respectively, we also tried to use other data augmentations in the modified network.

However, different from the data augmentation used for image classification, the data augmentation for this low-level task will bring **negative effects**. For example, the random horizontal flipping is used for data augmentation, however, the PSNR of the test set decreased from 36.82 to 36.159 (see **Modified 7** and **Modified 7(data augmentation)** in **Appendix**). It is mainly because the images in the test set have no relationship with the horizontal features (different from some classification tasks such as the MNIST dataset), so a random horizontal flipping will bring additional noise so that the network did not learn enough features.

3. Data Loader and batch size

Data loader is an iterable class used for loading part of data into our model during training, and this part of data is called “batch” with the number of samples named “batch size” inside. A training dataset can be divided into one or more batches.

There are three different choices for batch size:

- Batch Gradient Descent: Batch Size = Size of Training Set
- Stochastic Gradient Descent: Batch Size = 1
- Mini-Batch Gradient Descent: 1 < Batch Size < Size of Training Set

We normally use the Mini-Batch algorithm in Deep Learning so that we do not need to load all the samples into the network at one time (save the computation) and can update parameters multiple times in one epoch to accelerate convergence.

As for our project, in baseline model training, we set the batch size as 8 after comparing the result using 16 as the batch size (proved by the results of **Selections Modified 4** and **Selections Modified 5** shown in **Appendix**) which shows that batch size of 8 will have a better performance.

4. Loss function

The basic idea of a deep learning neural network is learning to map a set of inputs to a set of outputs from training data. As it is almost impossible for us to calculate the perfect weights for the CNN, there must exist a difference between the predicted output and the ground truth. To minimize the difference (error), the **objective function** is often referred to as a **loss function** and the value calculated by the loss function is referred to as simply “loss” [8].

Some fluently used loss functions are Least Square Loss, Soft-margin Loss and Cross-Entropy Loss. The aim of the project decides which loss function will be used. And in this project, we used the Mean-Square-Error to compare the difference between the original and decoded images.

4. Adam optimizer and learning rate

The optimizer is used to minimize the loss as we have mentioned, which can be considered as finding the global minima of the loss function. The method that we find the minima is calculating and assigning the **gradient** of the loss function as zero by applying the **chain rule**.

There are many optimizers with different algorithms, and the most two widely used are Adam and SGD optimizers. In this project, we simply used Adam to minimize the loss function because hyper-parameters have intuitive interpretation and typically require little tuning in Adma [7].

The **learning rate** is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated [9]. A proper learning rate is important.

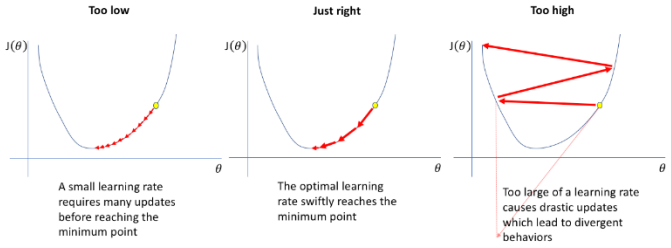


Figure 3 Learning rate effect

As the figure above, a small learning rate will spend more time and steps on reaching the minima. However, if the learning rate is too high, it will lead to divergent behaviours.

In this case, different learning rates (0.5e-3, 1e-3, and 2e-3) have been tested with the same batch size and the optimal one is when the learning rate is 1e-3 (it can be proved by **Selections Modified 2**, **Selections Modified 3**, and **Selections Modified 5**).

5. Forward and Back Propagation

1) Forward Propagation is the way to move from the input layer to the output layer. Different from CNNs for classification tasks, in this project, the ground truth to compare with is the original image without compression. As we just mentioned, the loss between the original images and compressed-decoded images could be calculated after the forward propagation.

2) Backward Propagation is the method of adjusting or correcting the weights in the connections of the network so that it could minimize the loss [10]. The weights of the network will be updated in the backward propagation in each epoch.

In this project, **convolution layers** are not only used to extract the features of the image but also aim to compress the images (**down-sampling**). The weights of the convolution kernel can be learned during the training process. The convolution kernels are used as a "**filter**" to extract the features from the input images into a feature map (Figure below).

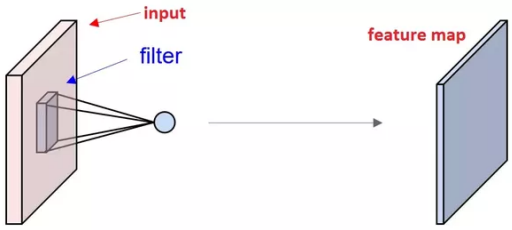


Figure 4 Convolution method

Padding is adding extra rows and columns on the outer dimensions of the image (figure below). It is used to increase the size of the output so that the output and input size is the same.

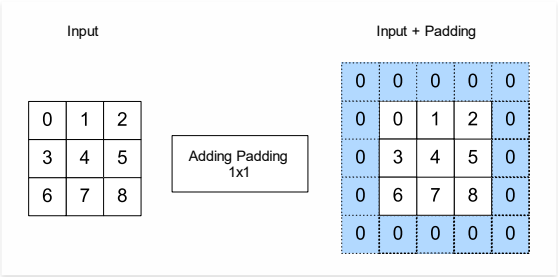


Figure 5 Padding

The **output width of the convolution layer** is calculated by the equation:

$$W_{out} = \frac{W_{in} - W_{filter} + 2 \times padding}{stride} + 1$$

As the height and width of the input image and filter are all the same, the height of the output H_{out} is also the same as the W_{out}.

Similarly, the **deconvolutional method** will also use filters. But it is mainly used to reconstruct the compressed images in our network (**up-sampling**). So that the output will have the same size as the input images.

The output width of deconvolution can be calculated by:

$$W_{out} = stride \times (W_{in} - 1) + W_{filter} - 2 \times padding + outpadding$$

And the H_{out} is also the same as the W_{out}in our project.

The algorithm below shows how we calculate the

parameters needed (e.g., stride, padding, out-padding) in the last deconvolutional layer:

```
input=torch.randn(1, 3, 256, 256)

a1 = conv(3, N)
a2 = nn.Conv2d(N, N-2,kernel_size=1)
a3 = conv(N-2, N-2)
a4 = nn.Conv2d(N-2, N,kernel_size=1)
a5 = conv(N, M)

output = a1(input)
output = a2(output)
output = a3(output)
output = a4(output)
output = a5(output)

b1 = deconv(M,N)
b2 = nn.ConvTranspose2d(N, N, kernel_size = 1)
b3 = deconv(N,N)
b4 = nn.ConvTranspose2d(N, N, kernel_size = 1)
b5 = nn.ConvTranspose2d(N, 3, kernel_size=3, stride=2,padding = 1,output_padding=1)
output1 = b1(output)
output1 = b2(output1)
output1 = b3(output1)
output1 = b4(output1)
output2 = b5(output1)

print(output.size())
print(output1.size())
print(output2.size())
```

```
torch.Size([1, 56, 32, 32])
torch.Size([1, 45, 128, 128])
torch.Size([1, 3, 256, 256])
```

Figure 5 Parameters calculation

The input is a tensor with a size of [B, C, H, W]. where B is the batch size, C is the channel of the input images, and H and W represent the height and weight of the image.

The input images in the dataset (after data augmentation) are all [C, H, W] = [3, 256, 256], so we set up a random torch with size [1, 3, 256, 256]. With the algorithm above, we could make sure that the decoded images will have the same size as the input.

Section III: Results

In this part, the results for different networks are shown in order (p.s. Loss represents the **MSE Loss** for the test set, and **Accuracy represents the PSNR value** between original and compressed-decoded images in the test set). **A table that combines all the information for each network is shown in the Appendix**.

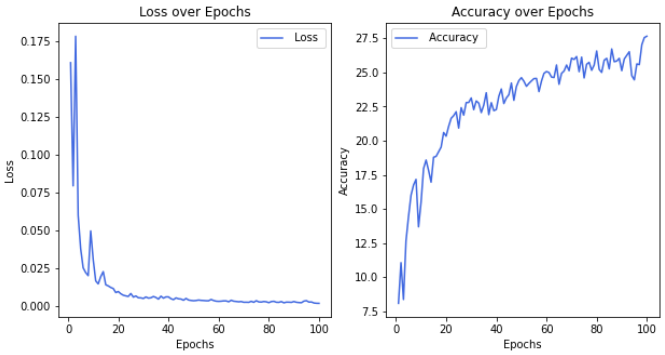
Notice that the PSNR value for an 8-bit image which is what we used in this project could be calculated by:

$$20 \cdot \log_{10} 255 - 10 \cdot \log_{10} MSE$$

Where 255 means the maximum pixel value is 255 and MSE represents the Mean-Square-Error. And typical values for the PSNR in the lossy image and video compression are between 30 and 50 dB.

1. Baseline network:

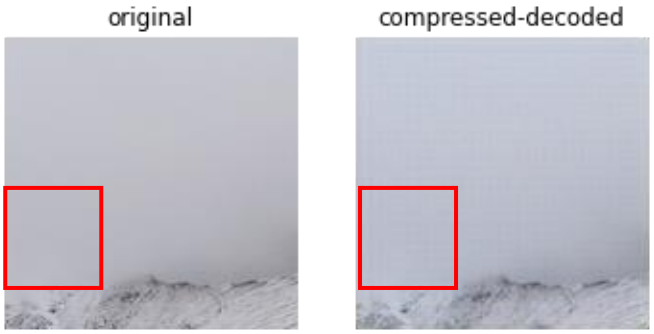
◆ Results are shown in the curve:



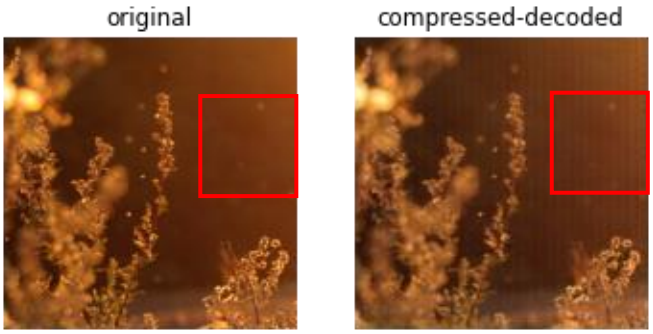
Where the maximum PSNR for the test set is 27.644 dB.

◆ Visualization for random three images in the dataset:

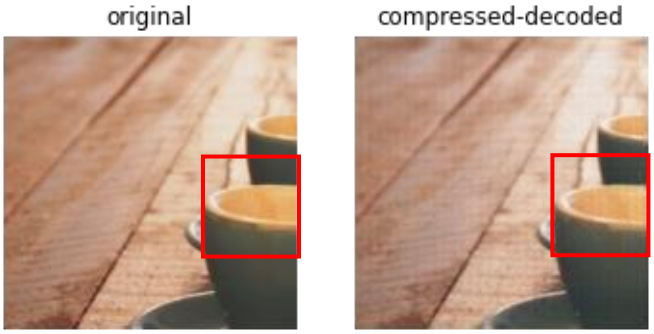
1. PSNR = 24.35 dB



2. PSNR = 21.46 dB



3. PSNR = 23.42 dB

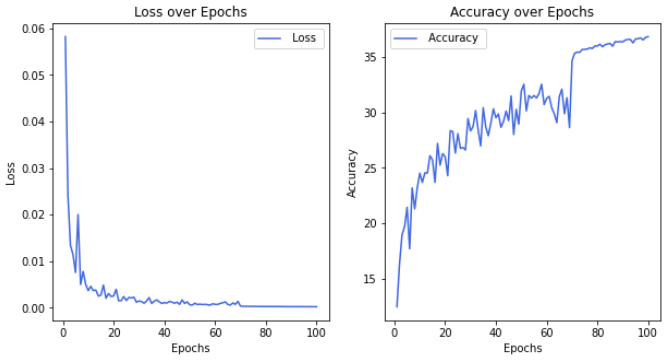


As human eyes are **more sensitive** to low-frequency signals, it could be seen that the parts of low-frequency are made up of some small pixels (red box), which

means the decoded result is not good enough.

2. Modified network 7 (final network):

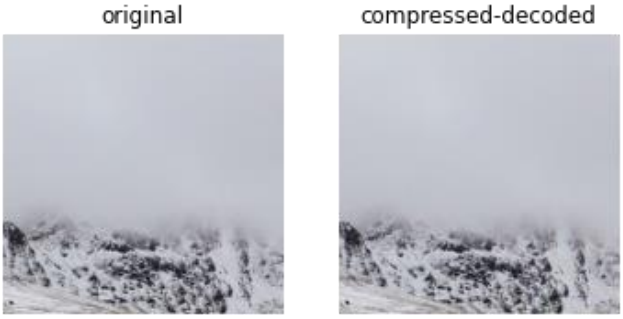
◆ Results are shown in the curve:



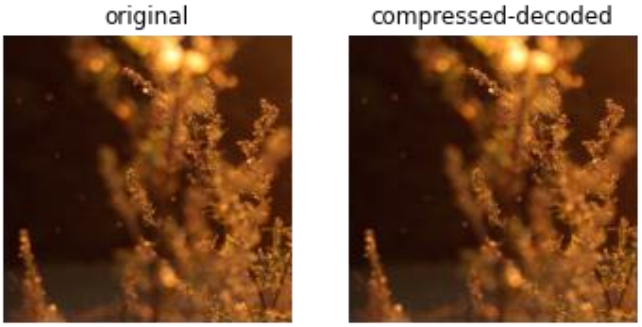
Where the maximum PSNR for the test set is 36.82 dB.

◆ Visualization for random three images in the training dataset:

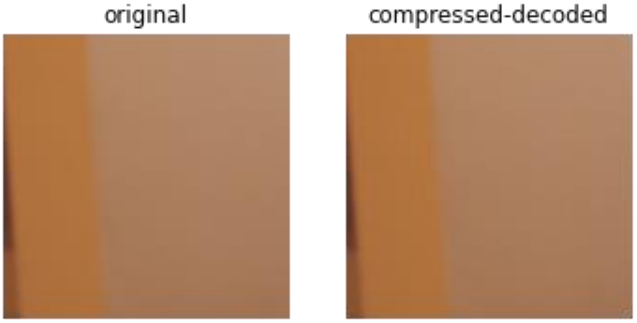
1. PSNR = 34.07 dB



2. PSNR = 36.35 dB

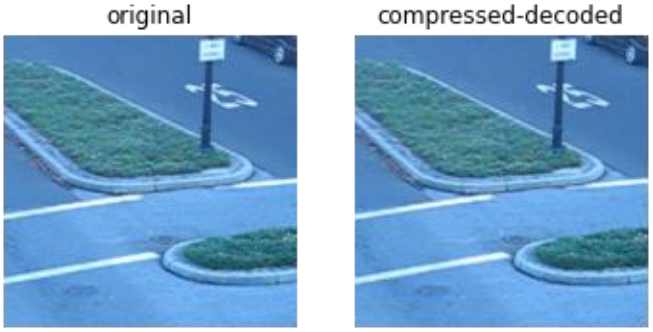


3. PSNR = 47.17 dB



◆ Visualization for random three images in the test dataset:

1) PSNR = 38.40 dB



2) PSNR = 39.91 dB

3) PSNR =38.40 dB

Based on the results above, it could be claimed that the difference between the original and decoded images is hard to tell by human eyes, which means the result is acceptable.

3. Rest of the networks:

As shown in the table in **Appendix**.

Section IV: Discussion

As in the previous parts, some of the networks with their effects have already been discussed (e.g., overall structure, data augmentation, learning rate, batch size, etc.). In this part, we mainly focus on analyzing **the rest of the results** from the other networks we used. And it

could be summarized as the following questions.

1) **Why there will be some drawbacks to using Batch Normalization in this project?**

Based on the result of **Modified 3** (shown in the **Appendix**), it could be figured out that with the batch normalization, the performance becomes worse than without using it (**Modified 4**). Some of the discussions have mentioned this problem [11] [12]. In summary, the batch normalization will destroy the original contrast information of the image because the mean value of each pixel will become zero and the variance will become one. It could be considered as increasing the generalization ability by increasing some noise. As for tasks such as classification, contrast information is a useless feature so that it could improve the performance. However, for this task which aims to reconstruct the image, batch normalization will negatively affect the result.

2) **Why activation functions (i.e., PReLU) do not increase the final performance in this task?**

Comparing the result of Modified 8 (shown in Appendix) and Modified 7, the final PSNR was reduced from 36.82 to 26.67 dB if a PReLU layer (figure below) is added in the Encoder (first part of the network).

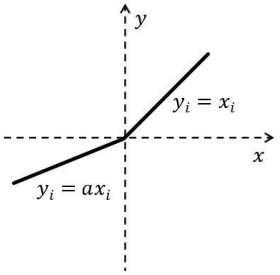


Figure 6 PReLU

The activation function is one of the main functions that could increase the non-linearity. There are not too many discussions about why activation functions will have a negative effect on CNNs. However, we believe that, as the self-defined network is a shallow network that only contains 3 layers for reconstruction, an activation will bring more noise to the compressed image so that it becomes difficult for the decoder to reconstruct.

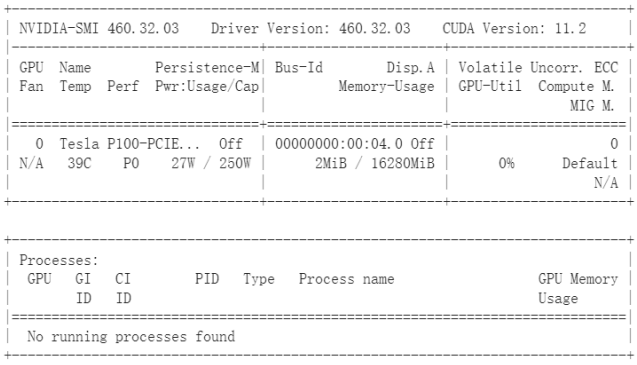
Section V: Conclusion

In this project, an End-to-End CNN model for image compression and reconstruction is developed by improving the baseline network. Different modifications with their results have been analysed and shown in **Section III** and Appendix.

The network with the best performance (**Modified 7**) has a final accuracy (PSNR) of **36.8dB** on the test set. Three of the images in the test set are also been visualized so that it could be found that the performance is higher enough so that human eyes cannot tell the difference between the original and decoded images.

Appendix

1) GPU used:



2) Table for all the results

Name	(N,M)	Compression ratio	Batch size	lr	epoch + time	PSNR	Structure
Baseline	(64,96)	(12, 0.868928)	16	lr = 1e-3	40 epochs	23.817	4 conv + 4 deconv
Baseline	(64,96)	(12, 0.868928)	16	lr = 1e-3	99 epochs (28min)	27.644	4 conv + 4 deconv
Modified Network 1	(32,48)	(3, 0.745984)	16	lr = 1e-3	40 epochs	27.2492	3 conv + 3 deconv
Modified 2	(40,54)	(5, 0.714232)	16	lr = 1e-3	40 epochs	27.6815757	3 conv + 3 deconv
Modified 3	(40,54)	(5, 0.714232)	16	lr = 1e-3	40 epochs	25.19012	3 conv+ 2 PeRelu + 1 BatchNorm + 3 deconv
Modified 4	(40,54)	(5, 0.714232)	16	lr = 1e-3	40 epochs	25.19048	3 conv+ 2 PeRelu+ 3 deconv
Modified 5	(40,54)	(4, 0.714232)	16	lr = 1e-3	40 epochs	27.783	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Hyper-parameter chosen based on Modified 5	(40,54)	(4, 0.714232)	32	lr = 1e-3	40 epochs	25.09	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Modified 6	(40,54)	(4, 0.714232)	8	lr = 1e-3	40 epochs	28.114	5 conv (including 2 1*1 bottleNeck) + 1 PReLU + 3 deconv
Modified 6	(40,54)	(4, 0.714232)	8	lr = 1e-3	79 epochs (25min)	32.511	5 conv (including 2 1*1 bottleNeck) + 1 PReLU + 3 deconv
Selections	(40,54)	(4, 0.714232)	8	lr = 1e-3	81 epochs (25min)	35.488	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Selections modified1 (increase epochs)	(40,54)	(4, 0.714232)	8	lr = 1e-3	98 epochs (28min)	35.854	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Selections modified2	(40,54)	(4, 0.714232)	8	lr = 1e-4	97 epoch (28 min)	28.833	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Selections modified3	(40,54)	(4, 0.714232)	8	lr = 0.5e-3	98 epoch (28 min)	34.796	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Selections modified4	(40,54)	(4, 0.714232)	16	lr = 2e-3	98 epoch (28 min)	28.079	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Selections modified5	(40,54)	(4, 0.714232)	8	lr = 2e-3	89 epochs (28 min)	36.031	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Modified 7	(45,56)	(5, 0.703648)	8	lr = 1e-3	99 epochs (28min)	36.82	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Modified 7 (data augmentation)	(45,56)	(6, 0.703648)	8	lr = 1e-3	86 epchs (28min)	36.159	5 conv (including 2 1*1 bottleNeck) + 3 deconv
Modified 7 (bottleneck)	(45,56)	(6, 0.703648)	8	lr = 1e-3	94 epochs (28min)	36.136	5 conv (including 2 1*1 bottleNeck) + 5 deconv (including 2 1*1 bottleNeck)
Modified 8	(45,56)	(5, 0.703648)	8	lr = 1e-3	87 epochs (28min)	26.67	5 conv (including 2 1*1 bottleNeck) + 1 PReLU + 3 deconv

References:

[1] F. Jiang, W. Tao, S. Liu et.al. 2017. *An End-to-End Compression Framework Based on Convolutional Neural Networks*. In IEEE Transactions on Circuits and Systems for Video Technology. Volume: 28, Issue: 10, Oct. IEEE. 2018

[2] TensorFlow Core. *Data augmentation*. Referred from: https://www.tensorflow.org/tutorials/images/data_augmentation

[3] iStock Staff, Jan 2022. *What is video compression and what does it do?*. Available at: https://marketing.istockphoto.com/blog/what-is-video-compression/

[4] Brownlee, J., 2019. *How Do Convolutional Layers Work in Deep Learning Neural Networks?*. Available at: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks

[5] Guo Lu, Xiaoyun Zhang, Wanli Ouyang, et.al, 2019. *DVC: An End-to-end Deep Video Compression Framework*. IEEE Trans. Pattern Anal. Mach. Intell. (PAMI), accepted, Apr, 2020

[6] Yan LeCun (2015). In Convolutional Nets, there is no such thing as "fully-connected layers". Available at: https://www.facebook.com/yann.lecun/posts/10152820758292143

[7] Jason Brownlee. (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[8] Jason Brownlee. (2019). *Loss and Loss Functions for Training Deep Learning Neural Networks*. Available at: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[9] Jason Brownlee. (2019). *Understand the Impact of Learning Rate on Neural Network Performance.* Available at: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[10] Neha Seth. 2021. *How does Backward Propagation Work in Neural Networks?*. Available at: https://www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/

Discussions about Batch Normalization problems:
[11] https://zhuanlan.zhihu.com/p/81193829

[12]https://blog.csdn.net/AIchipmunk/article/details/54234646

Figure2 1*1 kernel convolution
https://www.pianshen.com/images/251/86f696a7999a4ca94c7093d2719d650b.JPEG

Figure3 Learning rate effect:
https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png

Figure4 Convolution method:
https://qph.cf2.quoracdn.net/main-qimg-134024e4e35d7c7cbc4ccbe3a62dc8b2

Figure6 PReLU:
https://www.researchgate.net/publication/338560486/figure/fig1/AS:847054389645313@1578964912579/Parametric-Rectified-Linear-Unit-PReLU-28-and-the-proposed-Reverse-Parametric_Q640.jpg