COEN 240 Machine Learning

Term Project

Name: Jinhao Wang   ID: 1588747

Name: Pengwei Lin   ID: 1588140

Name: Yukun Zhang

Task1:

Code for Task1 is in MLprojectOne.py

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
60000/60000 [==============================] - 37s 614us/step - loss: 0.4906 - accuracy: 0.8195
Epoch 2/5
60000/60000 [==============================] - 34s 567us/step - loss: 0.3198 - accuracy: 0.8822
Epoch 3/5
60000/60000 [==============================] - 34s 572us/step - loss: 0.2761 - accuracy: 0.8988
Epoch 4/5
60000/60000 [==============================] - 36s 595us/step - loss: 0.2458 - accuracy: 0.9100
Epoch 5/5
60000/60000 [==============================] - 36s 592us/step - loss: 0.2229 - accuracy: 0.9182
10000/10000 [==============================] - 2s 232us/step
0.9035000205039978
[[825   1  23  23   4   1 117   0   6   0]
 [  3 976   1  13   3   0   3   0   1   0]
 [ 10   0 847   7  64   0  72   0   0   0]
 [  8   3  11 884  46   0  47   0   1   0]
 [  2   1  41  13 884   0  59   0   0   0]
 [  0   0   0   1   0 968   0  18   0  13]
 [ 94   0  58  17  66   0 757   0   8   0]
 [  0   0   0   0   0   7   0 953   0  40]
 [  3   1   4   6   4   4   1   4 972   1]
 [  0   0   0   0   0   4   1  26   0 969]]
```

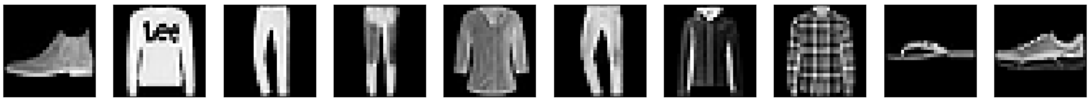
Task2:

Code for Task2 is in MLprojectOne.py

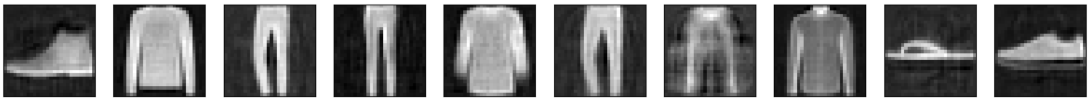
```
60000/60000 [==============================] - 16s 262us/step - loss: 0.0225 - accuracy: 0.1175
Epoch 2/5
60000/60000 [==============================] - 16s 270us/step - loss: 0.0164 - accuracy: 0.1481
Epoch 3/5
60000/60000 [==============================] - 16s 270us/step - loss: 0.0153 - accuracy: 0.1599
Epoch 4/5
60000/60000 [==============================] - 18s 292us/step - loss: 0.0148 - accuracy: 0.1664
Epoch 5/5
60000/60000 [==============================] - 17s 289us/step - loss: 0.0145 - accuracy: 0.1722
P= 10 , PSNR= 19.09739
```

```
60000/60000 [==============================] - 17s 285us/step - loss: 0.0153 - accuracy: 0.1753
Epoch 2/5
60000/60000 [==============================] - 18s 302us/step - loss: 0.0098 - accuracy: 0.2322
Epoch 3/5
60000/60000 [==============================] - 18s 299us/step - loss: 0.0088 - accuracy: 0.2496
Epoch 4/5
60000/60000 [==============================] - 18s 307us/step - loss: 0.0083 - accuracy: 0.2605
Epoch 5/5
60000/60000 [==============================] - 18s 302us/step - loss: 0.0080 - accuracy: 0.2674
P= 50 , PSNR= 21.97014
```

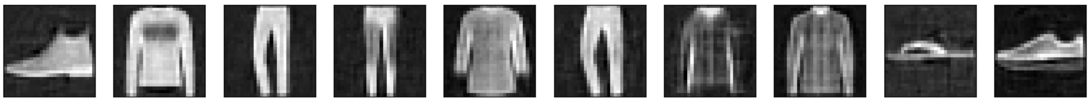
```
60000/60000 [==============================] - 19s 322us/step - loss: 0.0112 - accuracy: 0.2476
Epoch 2/5
60000/60000 [==============================] - 21s 348us/step - loss: 0.0055 - accuracy: 0.3359
Epoch 3/5
60000/60000 [==============================] - 21s 345us/step - loss: 0.0046 - accuracy: 0.3602
Epoch 4/5
60000/60000 [==============================] - 21s 351us/step - loss: 0.0043 - accuracy: 0.3719
Epoch 5/5
60000/60000 [==============================] - 21s 346us/step - loss: 0.0040 - accuracy: 0.3824
P= 200 , PSNR= 25.11773
```



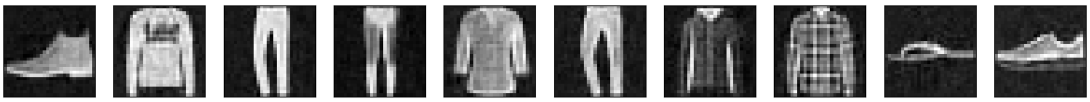
Original Images



P=10



P=50



P=200

2.a. What do you observe from the results? Give your comments.

When P is larger, PSNR is higher.

P represents the number of nodes in the compression layer. More nodes mean more features and more information are taken into consideration when compressing the image. Thus, when P is large, compression rate will be low, and the reconstructed image quality will be high. Therefore, PSNR will be high.

2.b. What do you observe from the decompressed images (the visual quality of the decompressed images of different P values)? With the same P value, which kind of images do you think are more difficult to decompress, and why?

Decompressed images with higher P values are closer to the original image.

From the 10 examples, our team believes that images with more detail are more difficult to decompress. For example, image No2 and No8, there are patterns on the shirt, and with a low P value, the decompressed images of these two have a distinct difference with their original images, compared with the other 8 examples.

This is because, with the same P value, images with more detail can lose more feature information than those with less detail, and thus become more difficult to decompress. Because information is already lost during the compression stage.

Task3:

We run code for task3 on Google Colab.

Introduction

At present, public safety is becoming an important topic worldwide. More and more surveillance cameras appear in crowded areas to monitor suspicious action or person. However, how to store such a large amount of camera footage becomes a big problem. A 30-second traveling video recorded by phone may not be very large, but a 10-minute video of a break-in at the bank could be extremely large. Therefore, video compression begins to play an essential role on the stage, since it can compress raw files into a smaller format. In this task, our team is going to propose a color video compression system, implemented with 2 different convolutional neural networks.

Proposed Methods

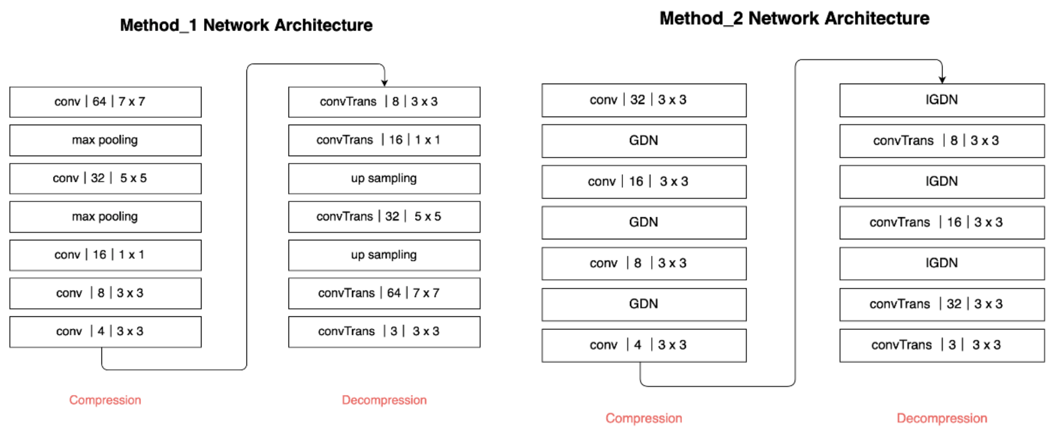


Figure 1 Two proposed methods

Description of methods

Method 1 Methodology

We apply a five-layer convolutional network and two max-pooling layers to encode image information.

We use MSE as our loss function and Relu activation function in each layer.

Also, the stride of max pooling depends on the compression ratio we apply in the current network.

Decompress procedure is the symmetric part of the image compression. Besides, we add a three-channel layer to retrieve 3 color channels for each pixel.

Method 2 Methodology

In this method, we train images with a four-layer convolutional network. Specifically, there is a GDN transformation applied to information between each layer.

As the final decompression process, we add a three-channel layer to retrieve the three-color channel as well.

The stride of the first of the second layer depends on the compression ratio we choose. Similarly, decompression is symmetric and we use MSE as our loss function and Relu activation function in each layer.

In addition, we train this network with TensorFlow version 1.13.1 because the GDN function does not compatible with a version higher than 1.13.

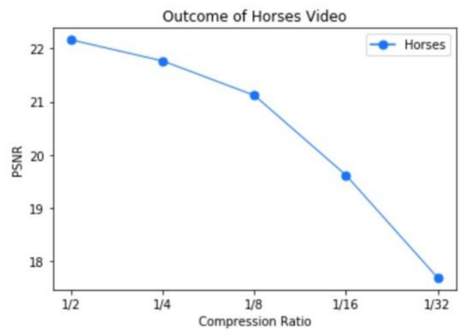
Description of Datasets

We trained three different datasets (Horses 416x240_300, Bubbles 416x240_500 and Basketball 832x480_500) with method1 and method2.

For each method, we train and test in the same datasets with train size of first 80% total number of images, and the rest of images are test datasets.

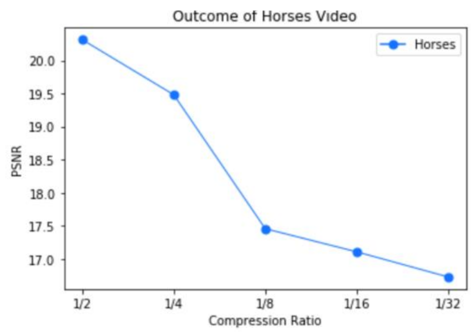
Due to the high volume of calculation, we split the 832x480 image into two 416*480 images while training the network, and concatenate two successive test images horizontally as output image to make comparison with the original image.

Quantitative evaluation



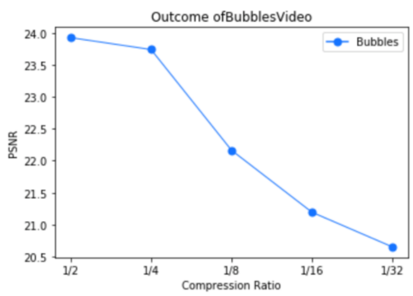

```
__Method1__
CR = 1/2, PSNR_with_ 30 Epochs =  22.16081
CR = 1/4, PSNR_with_ 30 Epochs =  21.76403
CR = 1/8, PSNR_with_ 30 Epochs =  21.11815
CR = 1/16, PSNR_with_ 30 Epochs =  19.61829
CR = 1/32, PSNR_with_ 30 Epochs =  17.6926
```

2.1-Model1-RaceHorses



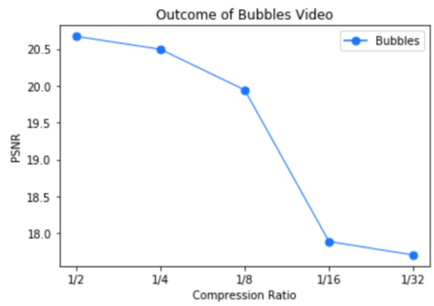

```
__Method2__
CR = 1/2, PSNR_with_ 30 Epochs =  20.31574
CR = 1/4, PSNR_with_ 30 Epochs =  19.48209
CR = 1/8, PSNR_with_ 30 Epochs =  17.45489
CR = 1/16, PSNR_with_ 30 Epochs =  17.1043
CR = 1/32, PSNR_with_ 30 Epochs =  16.72378
```

2.2-Model2-RaceHorses



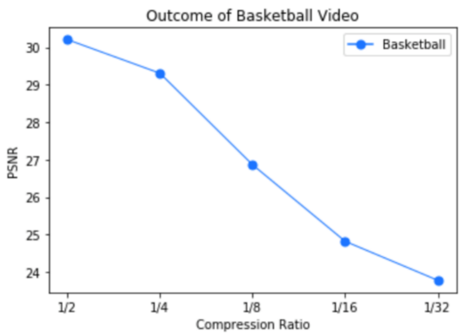

```
__Method1__
CR = 1/2, PSNR_with_ 30 Epochs =  23.92636
CR = 1/4, PSNR_with_ 30 Epochs =  23.7402
CR = 1/8, PSNR_with_ 30 Epochs =  22.16358
CR = 1/16, PSNR_with_ 30 Epochs =  21.19353
CR = 1/32, PSNR_with_ 30 Epochs =  20.65077
```

2.3-Model1-BlowingBubbles



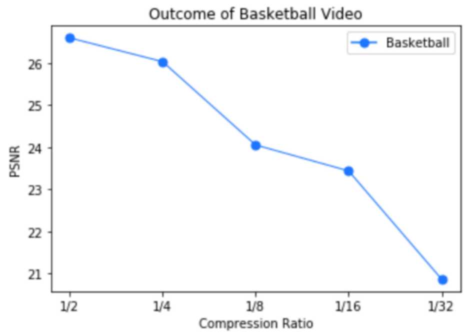

```
__Method2__
CR = 1/2, PSNR_with_ 30 Epochs =  20.6689
CR = 1/4, PSNR_with_ 30 Epochs =  20.49401
CR = 1/8, PSNR_with_ 30 Epochs =  19.94318
CR = 1/16, PSNR_with_ 30 Epochs =  17.89059
CR = 1/32, PSNR_with_ 30 Epochs =  17.70753
```

2.4-Model2-BlowingBubbles




```
__Method1__
CR = 1/2, PSNR_with_ 30 Epochs =  30.204
CR = 1/4, PSNR_with_ 30 Epochs =  29.30551
CR = 1/8, PSNR_with_ 30 Epochs =  26.86311
CR = 1/16, PSNR_with_ 30 Epochs =  24.81964
CR = 1/32, PSNR_with_ 30 Epochs =  23.77844
```

2.5-Model1-BasketballDrill




```
__Method2__
CR = 1/2, PSNR_with_ 30 Epochs =  26.60414
CR = 1/4, PSNR_with_ 30 Epochs =  26.03721
CR = 1/8, PSNR_with_ 30 Epochs =  24.0552
CR = 1/16, PSNR_with_ 30 Epochs =  23.43755
CR = 1/32, PSNR_with_ 30 Epochs =  20.86127
```

2.5-Model2-BasketballDrill

Figure 2.0 PSNR vs Compression Ratio

For both model one and model two, PSNR values decrease as compression ratio increases. This may because with higher compression ratio, more feature and information is discarded during the compression stage, which leads to a poor reconstruction quality.

For both model one and model two, BasketballDrill dataset achieves a better overall PSNR than the other two datasets. This may because our team deploys block-wise strategy on this dataset, while our initial intention is to help training process be faster.

Reconstruction quality provided by model one is better than model two. This may because our team deploys the max-pooling layer to extract 'the most important' feature to reduce dimensionality of data in model one. While in model two, we simply increase the stride to reduce dimensionality, which may lead to the loss of important training features, which finally results in poor reconstruction quality.

Perceptual quality evaluation

Original image:

Index: 19 Index: 39 Index: 59

Index: 19 Index: 39 Index: 59

Index: 19 Index: 39 Index: 59

1/2：

PSNR=22.53207 PSNR=22.30590 PSNR=22.76551

PSNR=20.78019 PSNR=20.97589 PSNR=20.17262

PSNR=23.52409 PSNR=23.54137 PSNR=24.12562

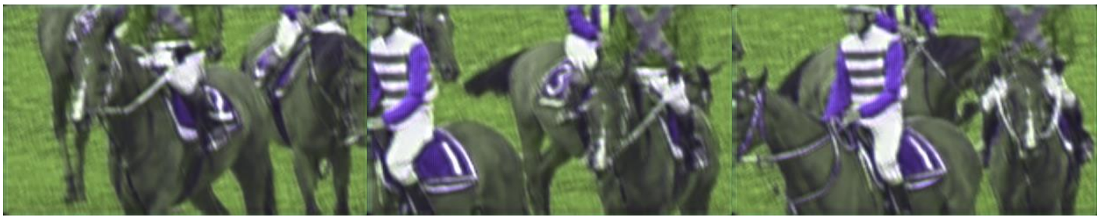
PSNR=20.98942 PSNR=20.80686 PSNR=20.34340

PSNR=29.58990 PSNR=29.87545 PSNR=29.68222

PSNR=26.38442 PSNR=26.90761 PSNR=26.79430

1/4:

PSNR=22.18376 PSNR=21.80203 PSNR=21.50798



PSNR=19.91414 PSNR=19.17890 PSNR=19.70927

PSNR=24.34340 PSNR=24.36271 PSNR=24.22319

PSNR=20.46673 PSNR=20.62885 PSNR=20.13989

PSNR=29.60451 PSNR=29.17378 PSNR=29.15452

PSNR=25.95958 PSNR=26.13443 PSNR=25.71210

1/8：

PSNR=21.21642 PSNR=21.23303 PSNR=20.95189

PSNR=17.19802 PSNR=17.96798 PSNR=17.65804

PSNR=22.24556 PSNR=22.25339 PSNR=22.45749

PSNR=19.52662 PSNR=20.37509 PSNR=19.59641

PSNR=27.15143 PSNR=26.62290 PSNR=27.45417

PSNR=23.52556 PSNR=24.11956 PSNR=24.25908

1/16:

PSNR=19.57192 PSNR=19.56580 PSNR=19.63909

PSNR=16.55229 PSNR=17.34676 PSNR=16.99265

PSNR=20.92151 PSNR=20.86619 PSNR=21.40923

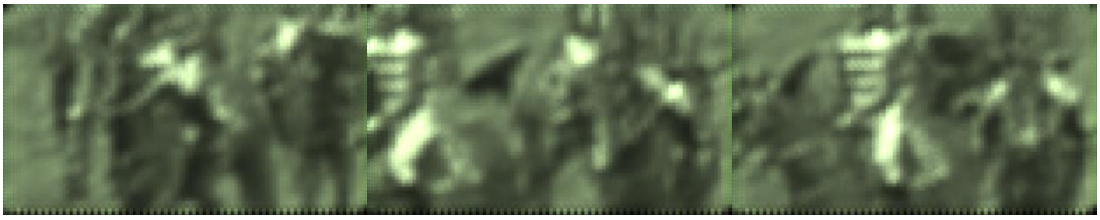
PSNR=17.75852 PSNR=17.52027 PSNR=18.20914

PSNR=25.33240 PSNR=24.69617 PSNR=24.59438

PSNR=23.10391 PSNR=23.90631 PSNR=23.48961

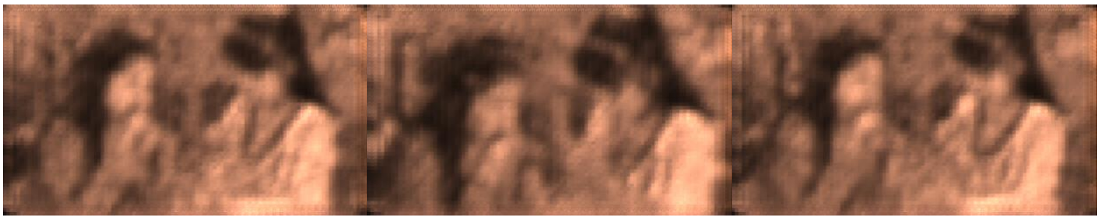
1/32：

PSNR=17.31037 PSNR=17.83613 PSNR=17.45683



PSNR=16.28889 PSNR=16.50618 PSNR=16.69475

PSNR=20.69782 PSNR=20.84468 PSNR=20.51396



PSNR=17.59992 PSNR=18.13453 PSNR=18.18766

PSNR=23.76988 PSNR=23.68020 PSNR=23.78677

PSNR=21.34539 PSNR=20.99794 PSNR=20.62797

Figure 3 Test frames under different compression ratio

For RaceHorses and BlowingBubbles datasets, we observe a quite poor reconstruction quality at compression ratio of 1/16 and 1/32 in model two. While in model one and BasketballDrill dataset, the reconstruction quality is still acceptable. This may because:

1. For RaceHorses and BlowingBubbles, our team deploys frame-wise strategy, while for BasketballDrill, we deploy block-wise strategy. In frame-wise strategy, since the image is proceeded as a whole, under higher compression ratio, feature and information are likely to be lost;
2. For model one, our team deploys the max-pooling layer to extract important features and information to reduce dimensionality, while in model two, we simply modify stride to reduce. Under high compression ratio, loss of essential features will definitely result in extremely poor reconstruction quality.

Complexity and model size analysis

Model One:

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 480, 416, 64)	9472
max_pooling2d_2 (MaxPooling2	(None, 120, 104, 64)	0
conv2d_6 (Conv2D)	(None, 120, 104, 32)	51232
max_pooling2d_3 (MaxPooling2	(None, 120, 104, 32)	0
conv2d_7 (Conv2D)	(None, 120, 104, 16)	528
conv2d_8 (Conv2D)	(None, 120, 104, 8)	1160
conv2d_9 (Conv2D)	(None, 120, 104, 4)	292
conv2d_transpose_5 (Conv2DTr	(None, 120, 104, 8)	296
conv2d_transpose_6 (Conv2DTr	(None, 120, 104, 16)	1168

up_sampling2d_2 (UpSampling2	(None, 120, 104, 16)	0
conv2d_transpose_7 (Conv2DTr	(None, 120, 104, 32)	544
up_sampling2d_3 (UpSampling2	(None, 480, 416, 32)	0
conv2d_transpose_8 (Conv2DTr	(None, 480, 416, 64)	51264
conv2d_transpose_9 (Conv2DTr	(None, 480, 416, 3)	9411

===

Total params: 125,367
Trainable params: 125,367
Non-trainable params: 0

Model Two:

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 60, 104, 32)	896
gdn_25 (GDN)	(None, 60, 104, 32)	1056
conv2d_17 (Conv2D)	(None, 60, 52, 16)	4624
gdn_26 (GDN)	(None, 60, 52, 16)	272
conv2d_18 (Conv2D)	(None, 60, 52, 8)	1160
gdn_27 (GDN)	(None, 60, 52, 8)	72
conv2d_19 (Conv2D)	(None, 60, 52, 4)	292
gdn_28 (GDN)	(None, 60, 52, 4)	20
conv2d_transpose_16 (Conv2DT	(None, 60, 52, 8)	296
gdn_29 (GDN)	(None, 60, 52, 8)	72
conv2d_transpose_17 (Conv2DT	(None, 60, 104, 16)	1168
gdn_30 (GDN)	(None, 60, 104, 16)	272
conv2d_transpose_18 (Conv2DT	(None, 240, 416, 32)	4640
conv2d_transpose_19 (Conv2DT	(None, 240, 416, 3)	867

===
Total params: 15,707
Trainable params: 15,707
Non-trainable params: 0

To calculate number of parameters:
For Conv2D and Conv2DTr layer, parameters = (width of filter * height of filter * number of channels + 1) * number of filters. For example, for the first Conv2D layer in model one, filter size is 7*7, number of channels of input is 3, 1 represents the bias term, number of filters is 64. Thus, total number of parameters for the first Conv2D layer in model one is (7*7*3+1)*64=9472.
For MaxPooling and Upsampling layer, parameter is 0, since there is no back-propagation learning involved.
For GDN and IGDN layer, they are generalized divisive normalization layers. Parameters of this layer can be calculated as the (number of input channel + 1) * number of input channel.

To calculate complexity:
Method1:

```
O(64*7*7*64*7*7        #layer1
+
32*5*5*32*5*5          #layer2
+
16*1*1*16*1*1          #layer3
+
8*3*3*8*3*3            #layer4
+
4*3*3*4*3*3            #layer5
+
8*3*3*8*3*3
+
16*1*1*16*1*1
+
32*5*5*32*5*5
+
64*7*7*64*7*7)
```

Method2:

```
O(32*3*3*32*3*3        #layer1
+
16*3*3*3*3*16          #layer2
+
8*3*3*3*3*8            #layer3
+
4*3*3*3*3*4            #layer4
```

+
8*3*3*3*3*8
+
16*3*3*3*3*16
+
32*3*3*32*3*3
)

Conclusions and Future Work

Video compression aims to achieve a high compression ratio while keeping the reconstruction quality high. In the 2 models our team proposes, both reconstruction quality reduces when the compression ratio grows higher. The decline rate of the 2 models is similar, while model one has an overall higher reconstruction quality.

Based on this observation, we conclude that model one performed better than model two. This may result from different dimensionality reduction approaches the 2 models deploy. In model one, our team uses the max-pooling layer to extract 'the most important' feature to reduce dimensionality of data, while in model two, we simply increase the stride to reduce, which may lead to the loss of important training features, which finally results in poor reconstruction quality.

We also conclude that both models perform well on BasketballDrill dataset than the other two. This may because our team deploys block-wise strategy on BasketballDrill dataset, thus more features and information can be saved. While in frame-wise strategy, since the image is processed as a whole, features and information are more likely to be discarded, resulting in a bad performance.

For future work, our team believes that memory and computation efficiency is the most important one that we need to address. At present, model two takes an extremely long time to train on large images. This leads to many other problems. For example, due to the large time consumption, our training epoch cannot be set to be too large. When we set epoch to 100 on model one on the first dataset (RaceHorses), the outcome PSNR value is satisfying, reaching approximately 28 (block-wise strategy is not applied). Therefore, if we can improve efficiency, and train models with more epoch, the model should work better.

Furthermore, with the improvement of efficiency, our team believes that we can deploy block-wise strategy more often, to avoid loss of features and information in order to achieve a better reconstruction quality. Meanwhile, maintain an acceptable time consumption.

References

Ball´e, J., et al., “Density Modeling of Images Using a Generalized Normalization Transformation”. In: arXiv e-prints. Presented at the 4th Int. Conf. for Learning Representations, 2016. arXiv: 1511.06281.

Ball´e, J., et al., “End-to-end Optimized Image Compression”. In: arXiv e-prints. Presented at the 5th Int. Conf. for Learning Representations, 2017. arXiv: 1611.01704

Ball´e, J., “Efficient Nonlinear Transforms for Lossy Image Compression”. In: arXiv e-prints. arXiv: 1802.00847

Chen, T., et al., “DeepCoder: A Deep Neural Network Based Video Compression”. Presented at Visual Communications and Image Processing, 2017.

Chen, Z., et al., “Learning for Video Compression”. In: arXiv e-prints. arXiv: 1804.09869

Huang, C., et al., “Extreme Image Coding via Multiscale Autoencoders with Generative Adversarial Optimization”. In: arXiv e-prints. arXiv: 1904.03851

He, K. and Sun, J., “Convolutional Neural Networks at Constrained Time Cost”. In: arXiv e-prints. arXiv: 1412.1710

Koratana, A., et al., “LIT: Block-wise Intermediate Representation Training For Model Compression”. In: arXiv e-prints. arXiv: 1810.01937

Mardani, M., et al., “Deep Generative Adversarial Networks for Compressed Sensing (GANCS) Automates MRI”. In: arXiv e-prints. arXiv: 1706.00051

Ma, S., et al., “Image and Video Compression with Neural Networks: A Review”. In: arXiv e-prints. arXiv: 1904.03567

Contribution of Team Members

Jinhao Wang: 50% (Q1, Q2, Q3)
Pengwei Lin: 40% (Q3)
Yukun Zhang: 10%