ELEC5306 Video Intelligence and Compression

Project 3

Optimization and Innovation: Design and Performance of Model Improvements

Group 48

Tianze (Harry) Zeng SID: 510215695

Junkai (Kevin) Mei SID: 500210899

Zhiyi Shi SID: 500256298

Contents

1 Introduction

Nowadays, videos take up more than 80% of the internet bandwidth [1]. To cope with the rising bandwidth costs, it is important to build an efficient video compression system that ensures higher video quality.

In the past, video compression algorithms were usually based on Discrete Cosine Transform (DCT), block segmentation, and entropy coding techniques. For example, the H.264 standard specified by ITU-T and ISO functions achieved high compression rates at the time through DCT and variable block size segmentation [2]. Other compression algorithms like MPEG-4, H.265, etc., also followed these principles. However, these methods share a common limitation: they are not end-to-end optimized, which restricts their compression rates.

Recently, deep neural network (DNN)-based image autoencoders have increasingly achieved better performance than JPEG and BPG [3] [4]. DNN image compression utilizes large-scale end-to-end training and high nonlinearity [5]. These have not been used in traditional methods and have shown excellent results.

In this project, we established a new DNN model that achieves better performance in video compression compared to the original model in terms of PSNR and loss metrics. The performance has been greatly improved.

2 Method

In this project, our model consists of two main parts: the encoder and the decoder. The encoder compresses the input video frames into a lower-dimensional latent representation, while the decoder reconstructs the original frames from this compressed data. This structure allows efficient video compression while maintaining high reconstruction quality. The network is shown in figure 1.

2.1 Encoder

In the encoder part, the encoder is composed of a series of convolutional layers and other layers. These layers extract useful features from the input video and remove redundant data.

In the encoder, the first layer is a convolutional layer. It is used to extract features from the image. Through a series of convolution operations, it captures low-level features like edges and textures in the image. After the second convolutional layer, there is `batchnorm`. `batchnorm` is used to normalize the input, making the data distribution more stable in each layer. By reducing internal covariate shift, `batchnorm` can lessen the impact of parameter changes during training. It also allows for larger learning rates, speeding up the training process and improving

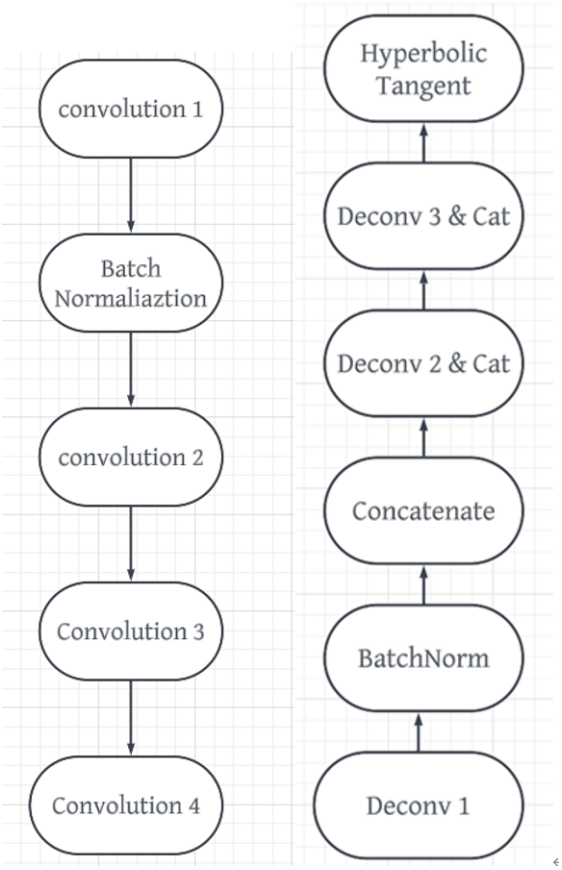

Figure 1: Network Layers

model stability and convergence speed. Next is the `LeakyReLU` layer. `LeakyReLU` prevents the problem of "dead neurons" caused by negative inputs being set to zero. Additionally, `LeakyReLU` keeps the gradient flow and non-linearity, helping the model avoid the vanishing gradient problem while maintaining non-linearity. After this, there is the second convolutional layer with `LeakyReLU`, then the third convolutional layer with `LeakyReLU`, and finally the fourth and last convolutional layer with `LeakyReLU`.

2.2 Decoder

The decoder restores the input data back to video quality that is close to the original. First, the decoder relies on a series of deconvolutional layers and other layers to decode the features and gradually restore the video data. Then it reconstructs the data, using the decoder to fill in the missing data. The final output is the video image.

The decoder starts with the first deconvolutional layer. It up samples the low-dimensional feature maps to high-dimensional feature maps, gradually restoring the structure and texture information. The following `batchnorm` stabilizes the gradients, preventing gradient vanishing, and ensures gradient stability during backpropagation. `LeakyReLU` maintains gradient flow and non-linearity. After this, there is the second deconvolutional layer with `LeakyReLU`, then the third deconvolutional layer with `LeakyReLU`. Finally, there is the fourth deconvolutional layer with Tanh. This tanh function compresses the output range to `[-1, 1]`, which is convenient for subsequent processing.

3 Experiment and Analysis

3.1 Experimental Setup

3.1.1 Dataset and Data Preprocessing

In our experiment, we followed the steps and parameters below to set up and execute the training process, as well as to preprocess our dataset:

We used a dataset provided by the course, which contains multiple videos and their frame-by-frame images. The dataset has been pre-divided into training and test sets, each containing frames from different videos.

To ensure consistency and reproducibility in model training, we set a random seed `seed = 123`.

For data preprocessing, we applied the following transformations:

- **Training Data**: Random cropping of the images to a size of `(96, 128)`, followed by conversion to tensor format.
- **Test Data**: Center cropping of the images to a size of `(96, 128)`, followed by conversion to tensor format.

3.1.2 Data Loading

We used the `SequenceFolder` class to read the data. This class traverses all images in a specified directory and organizes them into sequences, facilitating subsequent model training and testing. We used `DataLoader` for batch loading. The batch size for the training data loader was set to `32`, and the batch size for the test data loader was also set to `32`. To improve data loading efficiency, we used `128` worker processes for parallel data loading.

3.1.3 Model Architecture

We used our designed convolutional neural network `CompressionNetwork` for video compression. The model parameters were set to `N = 128` and `M = 192`.

3.1.4 Training Configuration

Model training was conducted on a GPU. We used the Adam optimizer, an adaptive learning rate optimization algorithm that combines the benefits of momentum and RMSProp. It can effectively handle sparse gradients and speed up convergence. The initial learning rate was set to `3e-4`, a value validated through multiple experiments to ensure model convergence while avoiding instability caused by too rapid learning rates. To dynamically adjust the learning rate, we used the `ReduceLROnPlateau` scheduler, which reduces the learning rate when the loss stops decreasing. The loss function chosen was Mean Squared Error (MSE).

3.1.5 Training Process

During training, we performed `50` epochs. We used the `train_one_epoch` function to train each epoch. To avoid the exploding gradient problem, the maximum gradient norm was set to `1.0`. After each epoch, we evaluated the model performance on the test dataset using the `test_epoch` function and adjusted the learning rate based on the test loss.

If the loss for the current epoch was lower than the previous best loss, we saved a model checkpoint, including the current epoch number, model state dictionary, optimizer state dictionary, and learning rate scheduler state dictionary.

3.1.6 Hyperparameters

- Random Seed: `123`
- Training Epochs: `50`
- Maximum Gradient Norm: `1.0`
- Crop Size: `(96, 128)`
- Initial Learning Rate: `3e-4`
- Training Batch Size: `32`
- Test Batch Size: `32`
- Data Loading Worker Processes: `128`
- Model Parameters (Input and Output Channels): `N = 128`, `M = 192`

3.2 Experimental Results

3.2.1 Performance Metrics

For this project, we used model computational complexity, compression ratio, peak signal-to-noise ratio (PSNR), and mean squared error (MSE) loss as performance metrics to evaluate the model's performance.

3.2.2 Benchmarking

Model	Compute Complexity	MSE Loss	PSNR	Compression Ratio
Template	47 GFLOPS	0.001	33.498	71.56%
Improved	69 GFLOPS	< 0.001	36.413	73.79%
ResNet	88 GFLOPS	0.002	31.560	45.5%

Table 1: Benchmarking Results

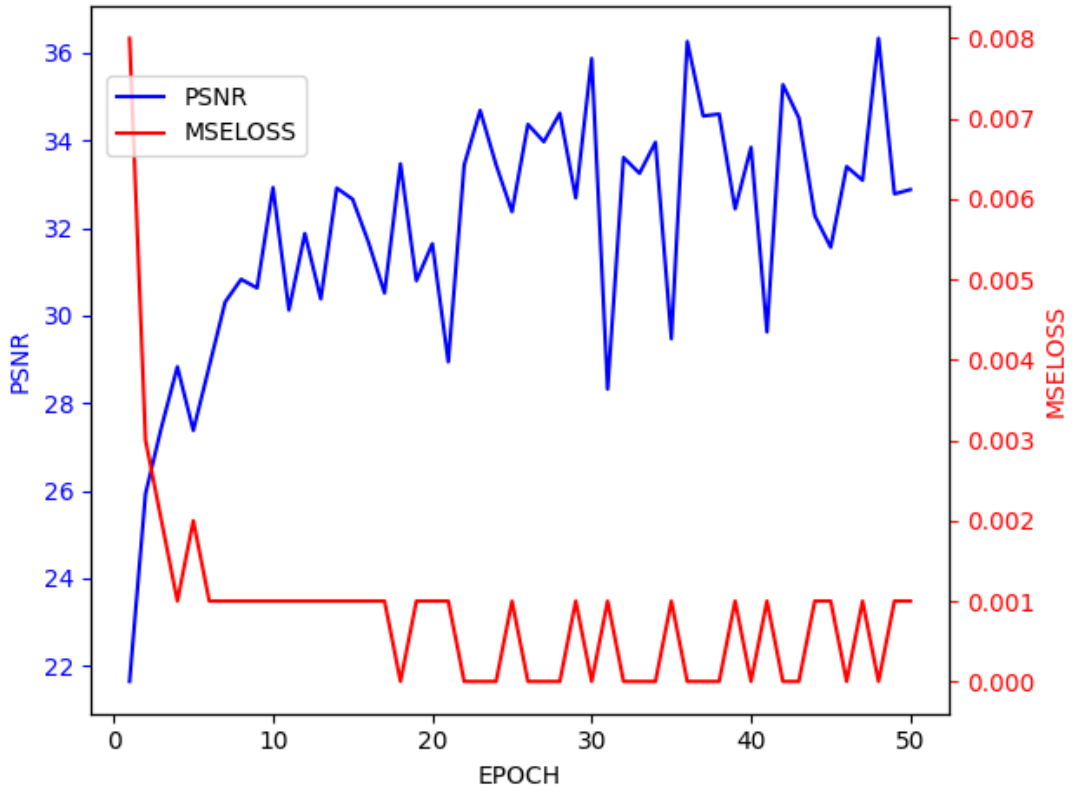

Figure 2: MSE Loss and PSNR during training

3.3 Results Analysis

3.3.1 Quantitative Analysis

From the experimental results, it can be seen that our improved model performs well on multiple performance metrics. Compared to the template model, the improved model increases in computational complexity (from 47 GFLOPS to 69 GFLOPS), but shows significant improvement in PSNR and MSE loss.

Specifically, the improved model increases the PSNR by nearly 3 dB (from 33.498 dB to 36.413 dB) and reduces the MSE loss to below 0.001. Additionally, the improved model significantly improves image quality while maintaining a high compression ratio (73.79%). This indicates that by reasonably increasing model complexity, we can effectively enhance video compression performance.

3.3.2 Qualitative Analysis

To further verify the model's performance, we compared the visual effects of the original video frames and the compressed video frames. As shown in Figure 3, our model excels in preserving image details, especially in maintaining edge and texture details. Compared to the template model, the improved model shows a clear improvement in visual quality.

Figure 3: Comparison of Original Video Frames and Compressed Video Frames

4 Conclusion

The convolutional neural network model we designed performs well in video compression tasks, significantly improving PSNR and maintaining a high compression ratio, while keeping computational complexity moderate. Experimental results show that the model performs excellently under various test conditions. Future work can further optimize the model architecture to improve video compression performance and efficiency.

5 References

[1] C. V. Networking Index. Forecast and methodology, 2016.

[2] 2021, white paper. San Jose, CA, USA, 1, 2016.

[3] Toderici, George, et al. "Full resolution image compression with recurrent neural networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[4] Toderici, George, et al. "Variable rate image compression with recurrent neural networks." arXiv preprint arXiv:1511.06085 (2015).

[5] Lu, Guo, et al. "Dvc: An end-to-end deep video compression framework." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.