Journal of Neonatal Surgery
ISSN(Online): 2226-0439
Vol. 14, Issue 32s (2025)
https://www.jneonatalsurg.com

Deep Learning Approaches in Video Compression and Transmission Efficiency

Swagata Sarkar[1], Janani Selvam[2], Divya Midhun Chakkaravarthy[3]

[1] Post Doctoral Fellow, Lincoln University College, Malaysia
Email ID: ssacademiczone@gmail.com
[2] Faculty of Engineering, Lincoln University College, Malaysia
Email ID: vijayjanani.s@gmail.com
[3] Faculty of Engineering Lincoln University College, Malaysia.
Email ID: divya@lincoln.edu.my

.Cite this paper as: Swagata Sarkar, Janani Selvam, Divya Midhun Chakkaravarthy, (2025) Deep Learning Approaches in Video Compression and Transmission Efficiency. *Journal of Neonatal Surgery*, 14 (32s), 5179-5194.

ABSTRACT
In recent years, video files have significantly increased in size, posing challenges for storage and transmission. One effective solution has been to reduce the length of videos. In this study, we introduce Deep Bi VC, a dual-branch framework designed to improve video compression. The model was developed to address these challenges through two distinct compression strategies. Initially, video sequences were pre-processed by segmenting them into groups of five consecutive frames to enable efficient processing. For the first stage, we implemented an Invertible Neural Network (INN) to develop an image compression module. This component focuses on compressing the first and last frames of each group. Subsequently, a video compression module was developed, utilizing motion prediction techniques to interpolate the intermediate frames between key frames. Experimental evaluations using PSNR and MS-SSIM metrics demonstrated that Deep Bi VC outperformed several state-of-the-art methods. On the VUG dataset, the model achieved a PSNR of 37.16 and an MS-SSIM of 0.98 at 3.2 bits per pixel, indicating superior compression performance.

Keywords: *Dataset, Feature Enhancement, Hyper-Prior, Light Field Video Compression, Light Field Video Compression.*

1. INTRODUCTION
Video makes up almost all of the data being made right now. It is now built into many digital phones that people use to talk on the phone. Video files can be changed, saved, and played more quickly if the quality stays the same. For a long time, many people have made and used AV1, HEVC, and H.264 to make pictures smaller. To keep the quality high, these tools shrink video files in three ways (Chen, et al. 2020). You can guess what will happen between frames, within frames, and between frames. "With minimal perceptible quality degradation" means you can send or save data very close to the quality it had at the start. One of the most essential parts of video compression is ensuring the process and the movie work well together. There are better ways to shrink pictures now than there were a long time ago. They're better at both of these. That being said, ML can help us do better by changing how we think about why we shrink movies (Kovtun, V, et al. 2022).

There should be a better way to cut movies since more people want better video files and frame rates. One way to fix these problems is to use machine learning to make pictures smaller. They can learn new skills by working on more films and TV shows. These ways will help you figure it out. According to a new study, Machine learning might help solve this problem. When old frames are shrunk a lot, DNNs can make new frames that look the same. Tools for reducing data these days are better because they use machine learning (Izonin, I.; et al. 2023). They work like this: movie clips are linked by when and where they were shot. This is the first study to examine how machine learning-based methods compare to simple ways of reducing movies. There needs to be a way to make pictures smaller that doesn't lose too much clarity. Many people have used the same method to make pictures for a long time. But machine learning is still very new, and it's constantly getting better (Shao, D., et al. 2024). The study will compare both types' good and bad points to find the best one for this situation.

This study tries to find the best way to reduce the size of pictures while keeping the quality as high as possible by using both machine learning and regular methods (Shilpa, B, et al. 2022). That's the end of it.

- ➢ We looked at six ways to get something done: We tested H.265, VP9, AV1, CNN, RNN, and DAE to see how well they reduced data, how hard they were to load, how good the pictures were, and how people felt about them.

- This page will discuss the following: Part 2 discusses a new study done in the area. As little quality loss as possible can be achieved by cutting movies down in the third part. This can be done using standard video compression and machine learning methods. Part 4 is a close look at how the ideas talked about turned into real things. This part (Mustafa, D.I.; et al. 2022)
- also shows the data that was used.

When you send as little as possible, try not to lose too much quality. Real-time systems need a lot more compression than other types of systems. There is a way for cars to drive themselves right now. They need to change how they shrink data right away. When you send something from one computer to another, it doesn't change. Remember this as you look for a better way to help self-driving cars get less camera data (Rakhmanov, A.; et al. 2023). That's how it works; some information is lost before the movie is sent. There is less information than there used to be because of better compression (Wiseman, Y. et al. 2024). A number is said to be round when you add different amounts to it. Some people decided what details to leave out to make it possible for people to send pictures that they could watch later. We miss some things when we do things the old way (Moffat, A, et al. 2019) because our eyes can't see them or can only see part of them. We need to change how we think about self-driving cars because what we see is not how they work.

- When you work on video compression for self-driving cars, here are some things to keep in mind:
- So they can decide quickly, self-driving cars need to be able to cut up pictures right away.
- In as little time as possible, self-driving cars should be able to send where they are from their cams.

Things could go wrong and make a lot of noise since you can change many settings. This noise and changes need to be fixed by the compression method as quickly as possible with as little data loss as possible. Because it knows how people see things, perception video code can cut pictures even more. A computer drawing of how our eyes focus is one way to do this. A new study made this award possible. These days, when we code videos, we usually record a small area around the look spots that is clearer than other areas that aren't as clear. Space is only evident in a small circle around the centre point, about the view angle. This is one reason why picking like this happens in space. Different people have different kinds of photoreceptors in their eyes (Bruce, et al. 2005). This way shrinks the picture even more, blocking out parts of it from view. This way, perceptual video coding can get better compression in the parts the user doesn't care about while keeping the quality of the parts they care about. The grade wouldn't have changed much after the move (Zhao, Y., et al. 2021). After the move, things would get even worse.

For some parts of a taped screen to fit better, they need to be squished in tighter. Some parts require more care, so you need to ease up on the squeeze. This idea is so crucial that many teachers and professors recorded their lessons during the COVID-19 plague (Khatoonabadi, et al. 2015). This has led to a lot of films based on the show. They all did better after our advice. This training might make a big difference in how people teach (Serrano-Carrasco, et al. 2021). Classrooms may record lessons on computers more and more. The pictures should be more significant for some reason.

The remainder of the paper is organized as follows: Section 2 reviews related work on light field video compression. Section 3 presents the proposed method developed in this study. Section 4 discusses the experimental results. Finally, Section 5 provides the conclusion of the research.

2. RELATED WORK

We will now talk about how to utilize afterimages. Next, we'll look at the parts and bits of the video files used to find moves. We'll also look at old-school projects that used keyframe selection and afterimages. When making afterimages, it's essential to pick the proper keyframes. A past study shows that this new method is not the same as other ones that have been used. To show how important the process is, this is done. There are two kinds of video files: movies that have been changed and movies that have not. Studies of various types are needed for multiple groups (Sullivan, G.J., et al. 2012). Keyframes show a cut scene in a film with an action or event. This picture can be used immediately to learn moves or make videos with others. The files UCF-101, HMDB51, and ActivityNet all have a lot of writing like this. These files are different because they don't have much extra information. They make it easy to train models without doing much work first. Movie study projects like C3D, mI3D, and Two-Stream CNN often use cut-down video files to find movie moves.

It's clear what's essential. This works best when the background doesn't change—like a crime report. With a still background, things that are moving stand out more. You know you have an afterimage when you keep seeing the same thing (Mochurad, L. et al. 2024). The places where things change over time are stacked on top of each other in this idea. People and cars can be tracked with afterimage techniques when you compare frames. You can make moving picture trails. This worked, but it was faster and took up 28% less room than the videos. Remember that this method didn't delete data (Chen, Y., et al. 2020) because it took afterimages from every frame, not just one. Illustrative Motion Smoothing is the name of a technique that was shown in a study. This is one hard way to make 3D pictures in multiple ways. You can also hide movements and smooth shapes. The thoughts that lead to these steps are the same ones that lead to afterimages. Things move more slowly, there is less background noise, and the time is shown. It helps you remember what's important.

Afterimages are a way to reduce the amount of time information in movies that could work very well. But we haven't used this much to learn or watch fewer movies. No one has considered picking the most critical frames for afterimages (Santamaria, et al. 2021). Someone who knows how to select frames for video summarization can teach you how to make afterimages. It was better to use CNN to help you choose keyframes. It learns how to read frames.

2.1. Light Field Compression

It could only shrink flat pictures when it was first made. Now, some tools can shrink four-dimensional light fields as well. Each of these ideas is meant to build on the one before it. We can see that the JPEG Pleno proof model for lenslet light field compression works in the real world. The discrete cosine transform (DCT) in four dimensions changes the data and divides it into four-dimensional blocks. It works well but uses a lot of CPU power (Malamal Vadakital, et al. 2012), so computers that need to start up quickly shouldn't use it. A lot of changes have been made to help kids learn numbers better (Jeon, et al. 2024). LFMs can be split in two different ways. One way is to make two-dimensional changes to four-dimensional blocks one at a time instead of all at once. This is simple to do. With the three-dimensional discrete cosine transform (DCT), you can move the SAIs in the light field along a third dimension and put blocks together in that dimension. Putting this together helps me with my maths. These two methods work better now (TranQ.N. et al, et al. 2022) but might not work with more complex light field models.

Cutting down on light fields is another thing that the discrete wavelet transform (DWT) can do. This is mainly meant to fix the weird things stop signs do when using DCT. We can improve compression and lessen harmful effects with 4-D discrete wavelet transform (DWT). Since the discrete wavelet transform (DWT) is used in JPEG2000, people have tried to figure out how to use it for light field compression. It would make models work better and less likely to go wrong. The Karhunen–Loève transform's ability to turn micro-images into sample vectors for vector quantization (H. M. Kwan, et al. 2023) is a big step forward. There wasn't as much noise to information the first time around. When data rates are cut down, the difference is even more significant. There is a new way to use the graph's Fourier function. It displays the colour, brightness, and form of the light field. After this change, the mean squared fix error may stay the same or get better. You can also cut the number of change factors by up to 21.92%.

Simple graphs are used in ways no one thought were possible before light fields were cut down. You can write code for good storage apps faster if you don't do the demos. We can talk about important things now that we have more time. Changing the lifts for plots is a great way to improve compression. One more way to do it is with the ideal intra-prediction method. The MSNR has gone up, and the bit rates have gone down. Graph learning and dictionary-guided sparse coding are to blame for this (Helen K, et al. 2023). This is true when you use taught graph adjacency vectors to make new views and save essential ones. Not long ago, the JPEG Pleno method got better. This change has also made light field compression better. From now on, it warps forward based on samples instead of backwards based on shapes. People can now guess and plan better about the view. They also say that the rate-distortion performance of coding depth maps is even better with the breakpoint-dependent discrete wavelet transform (DWT) (C.-Y. Lu et al. 2013). Another great new idea is the checked 4D-DCT code mode. The 4-D DCT is changed first, and its shape can be changed. In many types of light sources, this means the BD rate is less than 31%.

2.2. Light Field Video Compression

Light fields can be shrunk in many ways, but few people have tried them. A concept for an LFV compression gadget was created with deep learning. This method works well with the shape of LFVs. Sparse coding and a convolutional neural network reduce the number of saved views. The learning synthesis methods are then used to make the rest of the views again. This is possible with choice-based between-frame prediction (Pfaff, J., et al. 2018) and prediction models that can be changed. This changed how often things were repeated between frames and views in a big way. You need much more information for other well-known ways to code pictures so they can be watched more than once. Learn-based compression works excellent for movies like this one with slow frame rates (LFVs). You could use this method on light fields with five dimensions if you made the step where the light is squeezed last longer. It had been used before on light fields with four dimensions. For low-frequency video compression to work, this had to be done.

The images are then moved around in time so that well-known video codecs like HEVC can shrink them even more. When they use the VVC codec's hexagonal grid accuracy, rate twisting is much easier. It looks like this next to the old integer-pel and half-pel resolutions. LFV compression works much better now that hexagons have been added to the grid. With this new method, we can now guess motion vectors better (Bienik, J., et al. 2023). Sometimes, the quarter-pel fix is still valid. On the other hand, our way gives equal weight to both getting high compression rates and making computers run faster. This is how we shrink LFV files when the maths is easy, and the shrinking works well. This is how we can finish these two tasks (Zhang, Z.T., et al. 2017). If you don't use floating-point math, ADCT can help you get close to the right DCT. You need to know how to add and change. Now, the ideas have a better chance of working.

3. PROPOSED METHOD

3.1. Dataset

For this study, we got things from two different places. The first one is called Type A. Videos from Vimeo-90K were used to make it. This set has 89,800 movie clips. It's 448 by 256 pixels and has seven frames. There was no need to cut the video clips, and the size was set to 256 x 256 before training (Ma, C., et al. 2020). We got more information from UVG1080p to ensure the first set was correct. This group of data is just known as "Dataset B." This set has seven movies that are 1920x1080. 80% of dataset A was cut in half to make training and testing datasets. This was done because of what our study showed.

3.2. The Overall Architecture of the Model

Deep Bi VC might be unable to watch as many movies because it is learned in two steps. Many of the shots at the beginning and end of the first step were shrunk. The movie shrunk in the frames between the first and second steps. Figure 1 shows the house.

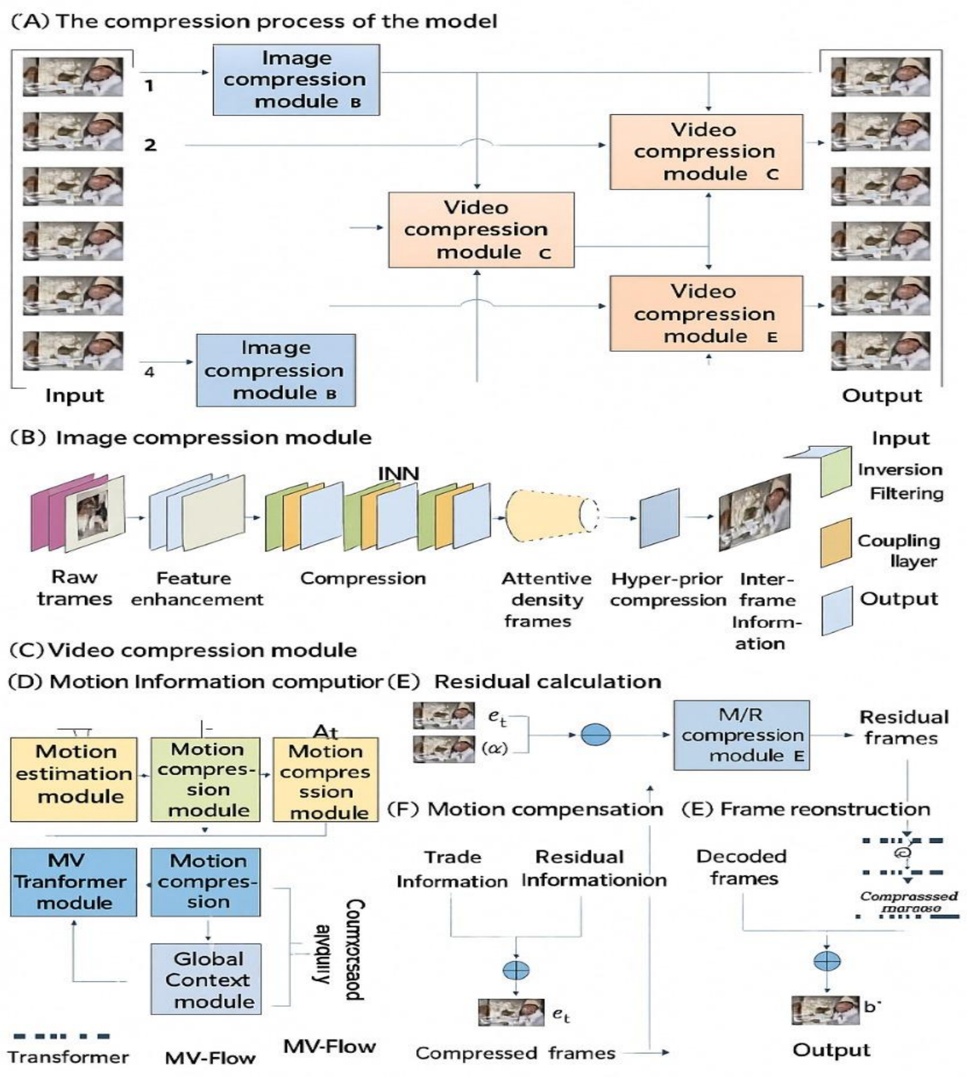


Figure 1. Main model architecture. DeepBiVC compresses frames—compressing images. C: Video-compression. Use b as the input frame and a' and c' as compressed surrounding frames to calculate motion information. E) A motion estimation network created Mv-flow motion vectors between two photo frames. Mv-bits and Res-bits were obtained by compressing Mv-flow and residual between predicted frame (b) and original frame b. The current frame b was predicted using bidirectional motion vectors $Ma' \rightarrow b$ and $Mc' \rightarrow b$. H) Calculated and compressed residual between predicted frame (b) and original frame b. Reconstruction of the original frame using residual Res-bits and bidirectional motion vectors $Ma' \rightarrow b$ and $Mc' \rightarrow b$ yielded the compressed frame Movement, waste.

We split the video data into sixty frames per second of separate picture data to fix it before we worked with it. After that, the frames had to be put together in fives. Each group got frames 1, 2, 3, 4, and 5.

3.3. Image Compression Module

The module ensured that each group's first and last frames were not too big. You can see all four parts in Figure 3B: making features better, making them smaller, making the attention path tighter, and giving them more weight than they would usually have. since you can flip it over. On the other hand, it showed how to get back info that had been compressed.

3.3.1. Feature Enhancement

We used a tool to improve the features, making it better at nonlinear models. This part had three convolutional layers and two big blocks. One, three, and one were the three seeds. It was there, as shown in Figure 2. A 3x3-inch picture frame was used for it. But 3 naira showed up. The pictures on the left and right are called H and W. With the thick blocks, the photographs and their parts no longer don't fit together right. People sent pictures that had 3D layers added to them to help them show more of who they are. This is how you can describe the result k for layer l of the thick block:

$$x_l = \text{LeakyReLu}(\textstyle\sum_{m-0}^{M} \sum_{n=0}^{N} \text{concat}\left[x_0, x_1 \cdots, x_{(l-1)}\right]. w_l(m,n)) \qquad (1)$$

$$LeakyReLu(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leqslant 0 \end{cases} \qquad (2)$$

Where $c\boldsymbol{U}ncat$ $[x0, x1, x(\boldsymbol{T}-1)]$ is the sum of the outcomes from the layers that came before it, $\boldsymbol{T}-1$, it was set to $\boldsymbol{X}eee\ ee\boldsymbol{X}u$ $(\cdot)$, and 0.01 was used to fix the problem where there was no gradient when x was negative and the slope was minimal. At layer l, the writing for the two-dimensional convolution kernel was $Ul(m, \boldsymbol{T})$. The đ and signs told us where the seed was.

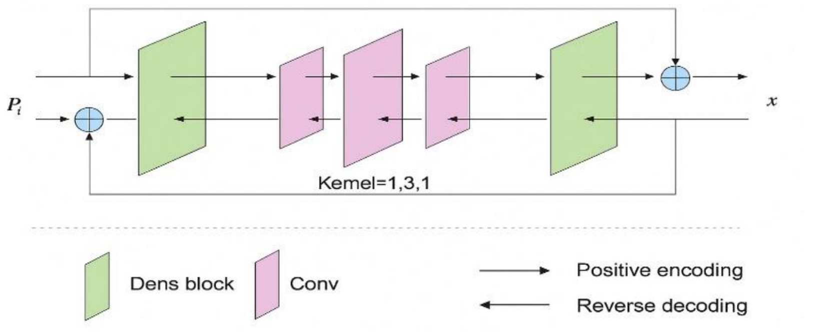


Figure 2. Feature enhancement module. Three convolutional layers with different kernel sizes and two dense blocks. Positive encoding extracts qualities x from image Pi, while reverse decoding recreates the original image.

3.3.2. Compression

The first and last frames were changed by adding and taking away frames. During the reduction step, the first picture's area was changed from space to frequency. The frequency parts were changed back to how they were initially in this method. Each INN building had four blocks that could be turned around. You could connect three levels to each block to make it smaller by one level. Each seed was three times bigger than the last one. It had a layer and a 1x1 convolution that could be turned around. It was given a tensor with twice as much information and four times less room for each channel. Four blocks could be turned around. After that, what I put in wasn't as sharp. A tensor could be sent and shown with the sign $x{\times}HHU$.

$$d(rh + i, rw + j, c) = x(h, w, c \cdot r^2 + i \cdot r + j) \qquad (3)$$

The output channel's index was shown by c, points in the tensor x were shown by $(\hbar, e)$, and i and j showed indices within the range of the scale factor r. The four blocks that can be turned around could then be linked together. The kernel size was set to five, five, three, three. The affine coupling layer gave us the number $d{=}ui1$:B. It could be any number from 0 to A. It split in half at point A, so the answer was A=$_e$($_e$+1)1:A. People called it the affine coupling layer to describe what it did:

$$u_{1:a}^{(i+1)} = u_{1:a}^{(i)} \odot exp[\sigma_a(g_2(u_{a+1:A}^{(i)}))] + h_2(u_{a+1:A}^{(i)}) \quad (4)$$

3.3.3. Attentive Channel Squeeze

A channel squeeze layer was added to reduce the number of pixels in the input tensor. It got people's attention, changing each outlet's value. The largest size that could be reached was that. It was shrunk down to kilobytes using kibibits. The channel squeeze layer changed the input I into L∈($_e$^{2}**Ω**2)$\boldsymbol{T\Omega}{\times}\boldsymbol{U\Omega}$. After the changes were made, things were sent to the attention function. Anyone could look at that part. Two attention blocks could send and receive in that part of the system.

$$y = forw_{attention}(C = softmax\left(\frac{W_Q \cdot C \cdot (W_K \cdot C)^T}{\sqrt{d_k}}\right) \cdot (W_V \cdot C) \quad (5)$$

$$C = back_{attention}(y) = softmax\left(\frac{W'_Q \cdot y \cdot (W'{}_K V \cdot y)^T}{\sqrt{d_k}}\right) \cdot (W'{}_V . y) \quad (6)$$

The parameter matrices for the forward focus method were shown by the letters UN, $\boldsymbol{UW}$, and UV. The backward attention method showed that they were $U'N$, $U'K$, and $U'V$. Someone or something kept the dot's size just right. It wasn't too big or too small.

3.3.4. Hyper-Prior

It was helpful to use the hyper-prior tool to eliminate extra information in the hidden form. This made entropy coding more stable and made compression work even better. Because someone else brought it up, we picked the one with a Gaussian distribution, mean, and size for this job. So that it could get z, it sent out settings for the hidden traits y. This is how y′ was set up after that.

$$z' = h_s(z) = h_s(h_a(y)) \quad (7)$$

For y, the analytical Gaussian transformation was used, and for z, the synthetic Gaussian transformation was used. The following equation shows how the two are connected:

$$p_{y'|z'} \leftarrow \{h_s(z'), C_m(y')\} \quad (8)$$

When the sign h□ (^′) is used, something has changed. Something new was added when the sign Xm (e') was used. We didn't know much about the entropy decomposition prior model, so we used it to show how e' was spread out as $Oe'|$ff̄ŋ:

$$py'|\theta = \theta(z') \quad (9)$$

The last step was to use entropy to send X′ and X′ using different ANS on each side. They didn't lose anything when they were turned into a bitstream. That's how the little picture frame ($'M$) was made.

3.4. Light Field Video Compression

A reverse compression device was also used to cut down on the space between the frames of each video series. There were four key parts to the program. One of them was process knowledge about motion. The other three fixed motion problems and put together new frames on computers. The movie had to be shrunk to see how the motion changed between this frame and those beside it.

3.4.1. Putting together motion information

The module had two parts: the motion estimate module and the M/R compression module. The people who did the M/R reduction had to figure out how to make the light work in both directions. After that, this information was squished together on visual flow. That machine had to learn how to send and receive light. First, we made a machine that could guess what would happen. It had three main parts: a transformer layer that changed parts of the image, a correlation Softmax layer that matched parts of the image, and a self-attention layer that made the image move better. Figure 3 shows how this network is put together. It has many parts.

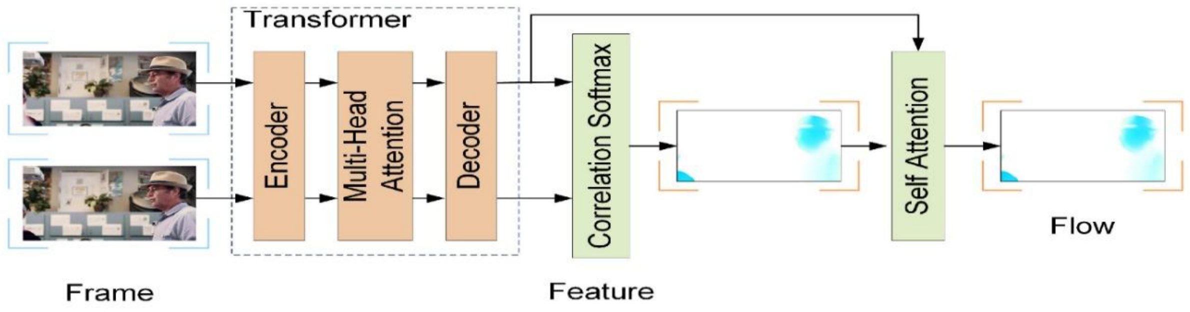


Figure 3. Module for motion estimation. This module processed incoming pictures using a transformer layer and multi-head attention to enhance data discrimination—calculated optical flow using Softmax correlation. The self-attention layer increased optical flow lastly.

There were two frames next to each other. They were worked on with these two pictures, named X1 and X2. Once we had X1 and X2, we started to take traits from them. Then, we got X1 and X2 with a smaller set of data. After that, the features were sent to the change layer to be made better. This caused better features I′1 and I′2 to be made. The following things happened when the multi-head focus device in the transformer layer was used:

$FU' = MultiHead(Q, K, V) = Concat(head_1, \dots, head_i)W^O$ (10)

Q, K, and V represent this context's question, key, and value matrices. The encoder analyzed the down-sampled data, generating these matrices. The digit 0 denoted the matrix of the linear transformation.

$R = \frac{(FU'_1)(FU'_2)^T}{\sqrt{D}}$ (11)

Everything changed simultaneously, as shown by the shape $N \in RT—U \times N—V$. Every piece of the picture could be put together ($_e$). Any spot on the feature map could be demonstrated by the letter "i." Then, we found the link N' between the feature maps by calculating the average of the chances that the two-dimensional pixel locations would match. These things helped us figure them out:

$N' = NP$ (12)

As you can see in picture 2, each point on the feature map was in a particular place. What was different about the two areas where the matched dots were found? That was the goal of the visual flow.

$Flow = N' - P$ (13)

It was also used to shrink the information that came out so that you could see how fast something moves. This part is put together in Figure 4. The encoder and the decoder were put together with a group of Resblocks. The decoder was made up of three Resblocks. The thing's job was to stop the motion vector or track from moving. In the secret area, smooth data had to be broken up into narrow data. The form that was there before it was done was changed. This lets the motion vector or trace be spread out again after it has been squished together.

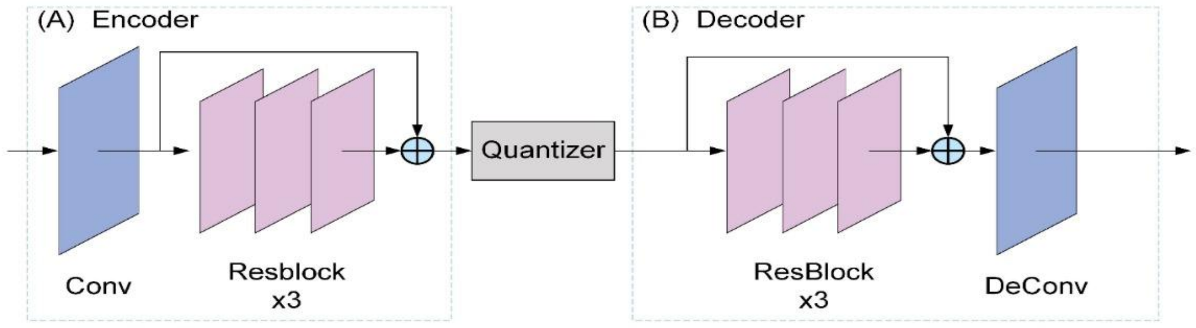


Figure 4. Modular compression. The encoder module featured a convolutional layer and residual blocks, while the decoder reconstructed quantized latent space data to obtain compressed motion or residual. Quantizers separated latent and continuous data.

There are several ways that the data from the Kth layer could be shown in the Resblock.

$x_K = x_k + \sum_{I=K}^{K-1} F(x_i, W_i)$ (14)

3.4.2. Motion Compensation

There are three frames in this stop-motion movie. To show the order in which they were worked on, they were named "before" (b), "current" (c), and "after" (a). The seventh picture shows how the two-way motion control system is put together. We used the two-way light flow estimates $M\boldsymbol{X}(c \rightarrow b)$ and $M\boldsymbol{X}(c \rightarrow b)$ to move the frames from before (Mb) to after ($M\boldsymbol{X}$) to now (Mc). Two-line drawing tools were used to make this change. Things from the frames before and after this one were mixed with this one's stuff. The last step was to find the forecast frame by putting together the masks and reference frames from before and after:

$\hat{P}_c = b \cdot M_v(c \rightarrow b) + (1 - b) \cdot M_v(c \rightarrow a)$ (15)

In frame $M\acute{}c$, the filtering coefficient was given by the number b. From one frame to the next, $M\boldsymbol{X}(c \rightarrow b)$ showed the direction of motion. From one frame to the next, $M\boldsymbol{S}(c \rightarrow b)$ showed the direction of motion.

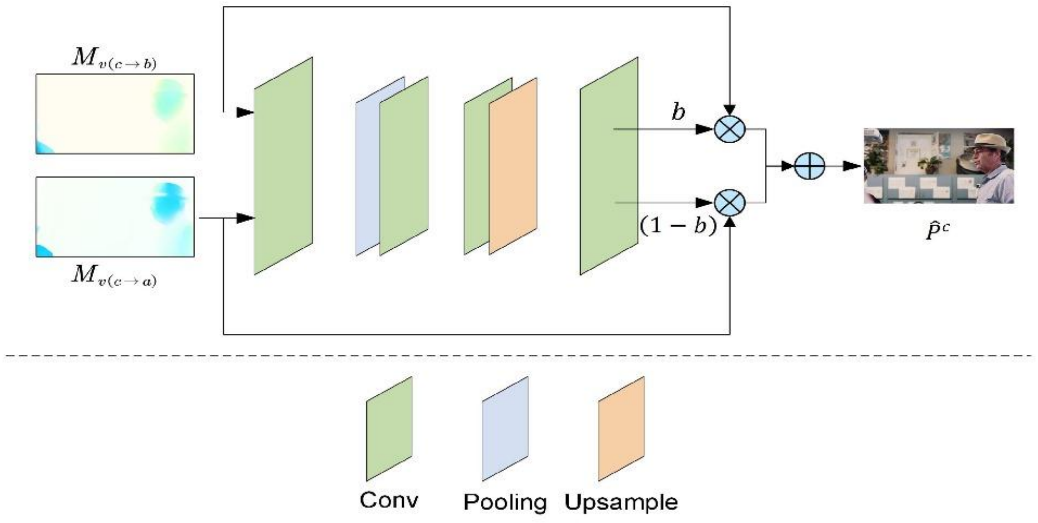


Figure 5. Motion-correction module. The inputs were forward and backward motion data; the output was masked image reconstruction.

3.4.3. Residual Calculation

There should have been that small change between the first frame (X) and the last frame. After that, the M/R reduction tool was used to fill in this space. The letter G is now here.

$$Res_{(x,y)} = C_{\frac{m}{r}}(\hat{P}_{c(x,y)} - P_{c(x,y)}) \qquad (16)$$

So, N-(x, y) was the value that was left over at position (x,y), and Mc(x,y) was the value of the pixel at position (x,y) from the first frame. The pixel value of the prediction frame at the point (,) was written on it.

3.5. Loss Function

Why lose? The goal of the project is to find a way to lower the bit rate needed to compress video while also making the frames that are played better:

$$L = \alpha[R_i + \lambda_i d(P_i - \hat{P}'_i)] + \beta[R_m + R_V + \lambda_b d(P_c - P'_c)] \qquad (17)$$

Three-bit counts were used to save the first and last frames, motion vectors, and residuals. There were three of them. The picture was shrunk, so the first frame (θϖ) and the second frame ($M'i$) were not straight. It was called $d(Mi{-}M'i)$. The word $d(Mc{-}M'c)$ showed the change from the first frame (Mc) to the second frame ($M'c$). The number $_e(\cdot)$ showed the mean squared error (MSE). A long time ago, people believed that the factors i and b managed the balance between rate and distortion. It was their job to train people and do other things. Before **Ω** was 1, they didn't know how to make things smaller. People were only shown how to shrink pictures when 0 and 1. The contrast, colour, and form of the two pictures could tell MS-SSIM how close they were to each other. It was also used to rate how well the shots were made. The MS-SSIM tests were good but not as good as those before them. This is how it was done:

$$MS\_SSIM(x,y) = [l_M(x,y)]^{\alpha} M \cdot \prod_{J=1}^{M}[\,c_j(x,y)]\beta_j \cdot [s_j(x,y)]^{\gamma j} \qquad (18)$$

The two pictures could be compared with things named x and y. These signs—$\boldsymbol{T}M(x,e)$, $cj(x,e)$, and $sj(x,e)$—showed that the colours, sizes, and similarities were all the same and different. People thought about how significant the $\boldsymbol{\Omega}M$, δj, and γj were. It was shown by the number M how many times the process had been done. The last thing we did was find the MS-SSIM average of all the frames from all the shows. One way to tell how well something worked was by how fast it could do. This method found the difference in bit rates between two encoders in a specific quality range. This is how it was done:

$$BDrate = \frac{\int_{Q_{min}}^{Q_{max}}(f_2(Q) - f_1(Q))dQ}{\int_{Q_{min}}^{Q_{max}}(f_1(Q)dQ} \times 100\% \qquad (19)$$

PSNR is the value of the picture after the encoder has shrunk it down. That's what Q means here. The number of bits was tied to a set term called B, which showed the launch level of the model. The picture's bit rate caused a significant change in quality Q. To find the middle number, the number $_e(\cdot)$ was used. It was written as $f(Q)$=10$\boldsymbol{V}\cdot N$+b. It was possible to compare the two encoders by writing them as $f1(Q)$ and $f2(Q)$. They also checked out other methods to see how their model compared to those.

DVC: This one is the first deep-learning method that shrinks pictures. People who can't figure out how to shrink pictures on their computers might find this helpful. It's the first model like this. Cute pictures are often used in this way to help kids learn. It's liked by many.

Open DVC: It was able to build DVC and use TensorFlow to improve it. It didn't just copy DVC's plan to enhance PSNR, though. A form of the DVC model also worked best with MS-SSIM.

RLVC: It compressed video much like how we do it now. It was called macro encoding. In the past, time links between frames that came before and after this one made it smaller. This method is a lot like how we work with nearby frame links.

FVC: There was nothing that the machine couldn't do in the picture frame area. We picked this one to compare in the feature space processing setting so we could be sure of our video compression method's pros and cons.

LHBDC: This model was also good because it used a stacked bidirectional video codec to watch the movie and guess how things would move in small steps. People have done work like this before.

H.264/AVC and H.265/HEVC were the two most common ways to slim down videos back then. H.264 and H.265 are often used as standards when discussing video compression. They put our way next to theirs to see how well it worked. We could see how well they saved data and how good the movie looked.

4. EXPERIMENTAL RESULTS

4.1. Dataset

These three LFVs are open to everyone. In this study, they are used to check how well LFV compression works. They're named "Car," "David," and "Toys." A lot of what you see in these LFVs is real. You can test the method on the car LFV to see how well it works for quick changes. A car goes slowly from left to right in this case. It's 512x352, so you can see how well the method works. The David LFV has a green background with a single person in a group. It's played on a turntable. This picture has 480x320 pixels. Two toys are on a turntable in this 480x320 movie from Toys LFV (Sritharan, et al. 2025). The background is green. This movie has a lot going on, but the plot is simple. Fifteen cameras were set up in a square for the three low-frequency video (LFV) shots. Each SAI has 420 YUV files, but the study only looks at 8x8 views. That's the first and last number in this table for each LFV channel. There are also smaller parts that make up that number. There is reason to believe the ADCT-based compression method since these LFVs show different amounts of motion, angles, and real-life settings. This part makes it more likely for the technique to work and be used elsewhere. You can read more about what was found here. Many places can use the way the method was found. Not many big things change when people move around a lot.

Table 1. Specifications of experimental LFVs.

LFV	Channel	Size	Min	Max
Car	U	8×8×176×256×248×8×176×256×24	25	181
	V	8×8×176×256×248×8×176×256×24	80	193
David	U	8×8×160×240×248×8×160×240×24	49	122
	V	8×8×160×240×248×8×160×240×24	107	151
Toy	U	8×8×160×240×248×8×160×240×24	3	160
	V	8×8×160×240×248×8×160×240×24	50	224
	Y	8×8×320×480×248×8×320×480×24	12	255

4.2. Evaluation Metrics

PSNR and SSIM are the ways we rate how well compression works. For each SAI of the decompressed LFVs, these two numbers are used. The first raw LFVs are used as a guide or "ground truth." Follow these steps to find the PSNR between

SAI A and SAI A′:

$PSNR = 20log_{10}(\frac{2^N-1}{\sqrt{MSE}})$, (20)

This time, N=8 tells us how many bits are used to describe an LFV. You need to do some math to determine how SAI A′ differs from SAI A.

$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(A(i,j) - A'(i,j))^2$. (21)

Here's how to find the SSIM for A and A′s:

$SSIM = \frac{(2\mu_A\mu_{A'}+C_1)(2\sigma_{AA'}+C_2)}{(\mu_A^2+\mu_{A'}^2+C_1)(\sigma_A^2+\sigma_{A'}^2+C_2)}$, (22)

What do D, D′, D, and DD′ mean? They show the means, standard deviations, and cross-covariance for SAIs A and A′. That's how they are displayed. Adding up the PSNR and SSIM for each SAI is one way to find the LFV's PSNR and SSIM. Light is in LFV. You only need to know the light part (Y) to get the full LFV. For the bright part of the colour, PSN stands for it. For the blue and red parts, PSN and PSN stand for them. Adding them all up lets you get the PSNR of the low-frequency voltage (LFV).

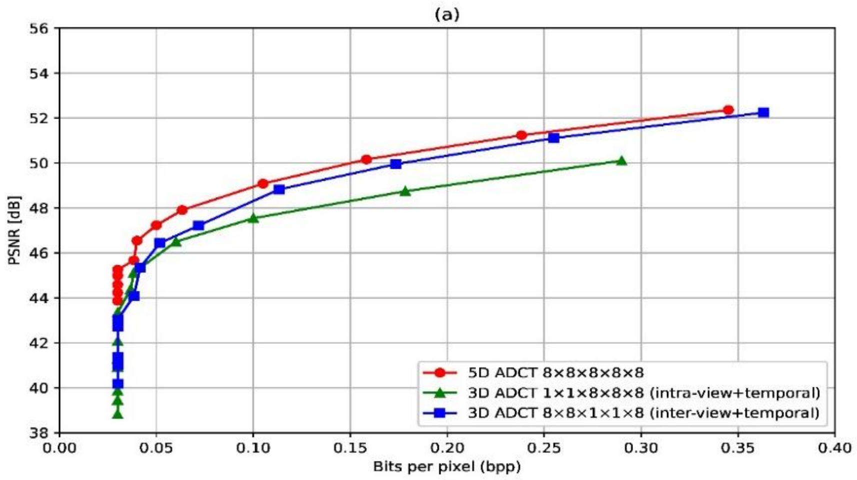
$PSNR_{YCbCr} = \frac{6PSNR_Y+PSNR_{Cb}+PSNR_{Cr}}{8}$.

We also care about how hard it is to understand the method given. We rate it by how well it does math. When the LFV has n data, these things happen: (1) the discrete cosine transform (DCT) methods for the LFV with n samples (2) A discrete cosine transform (DCT) operation in two dimensions; and (3) a discrete cosine transform (DCT) operation in one dimension. Don't forget that the DCT methods are hard to grasp because they need a lot of math. You must multiply by 64 and add by 56 to get the 1-D DCT. The exact DCT method is used to find this. To do the 1-D DCT process, this way only needs to add 14 numbers together, not multiply them. You can do math ten times faster now.

4.3. Results and Discussion

It tries to use the extra information in both the inside and outside views of an LFV. It's called a 5-D ADCT LFV compression method. Cutting down on 3-D ADCT can be done in three ways. After the first two, you'll see something faster. The third will see something else quicker, and the fourth will see something faster. All of them can be seen in Figure 6. We do this for ADCT. Three-dimensional ADCT that only shrinks time and space within a view is more minor than three-dimensional ADCT that only shrinks time and space between views. In an 8x8 square, the pictures are closer together, so this is the case. There are various ways that light can come from the same part of a picture. They look more like the blocks that show how light moves in a picture up close. The light in these blocks comes from the same place in the scene, so they all work. Of those ways, that is the best for both PSNR and SSIM. Copies are used in the talks and the parts that don't talk because they get smaller. The LFV PSNR and SSIM David's Y channel have changed, as shown in Table 2. It also shows how the various quantizations were connected. With a compression setting of 80 or more, a picture can only use 0.03 bits. The PSNR is about 45 dB, and the SSIM is 0.95 when this many bits are used.

When only 0.03 bits are used for each picture, it's clear that the file size is about 266 times smaller. This is because there are 8 bits in each colour pair. This picture shows Frames 35, 40, and 45 of the Car LFV's SAIs. This is a list of numbers: 1, 10, and 200. There are 4 SAIs altogether. This does prove that the LFv compression method works. It looks like the real thing when ₫=1 and ₡=10 are used to rebuild the frames. But the 200-won shots aren't as bright. You can still get good LFVs back if you compress them the suggested way, even if you speed up the process.



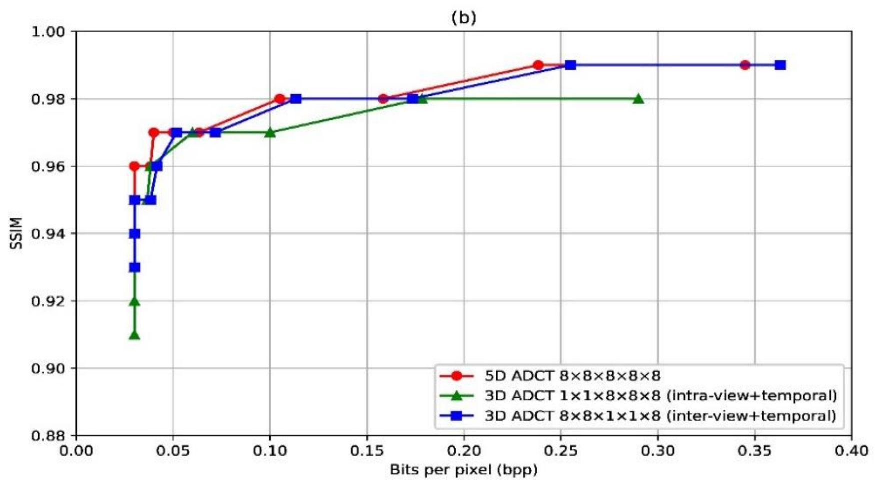


Figure 6. PSNR and SSIM are examined for 5-D ADCT, intra-view + temporal only 3-D DCT, and inter-view + temporal alone DCT. The ADCT algorithm is utilized.

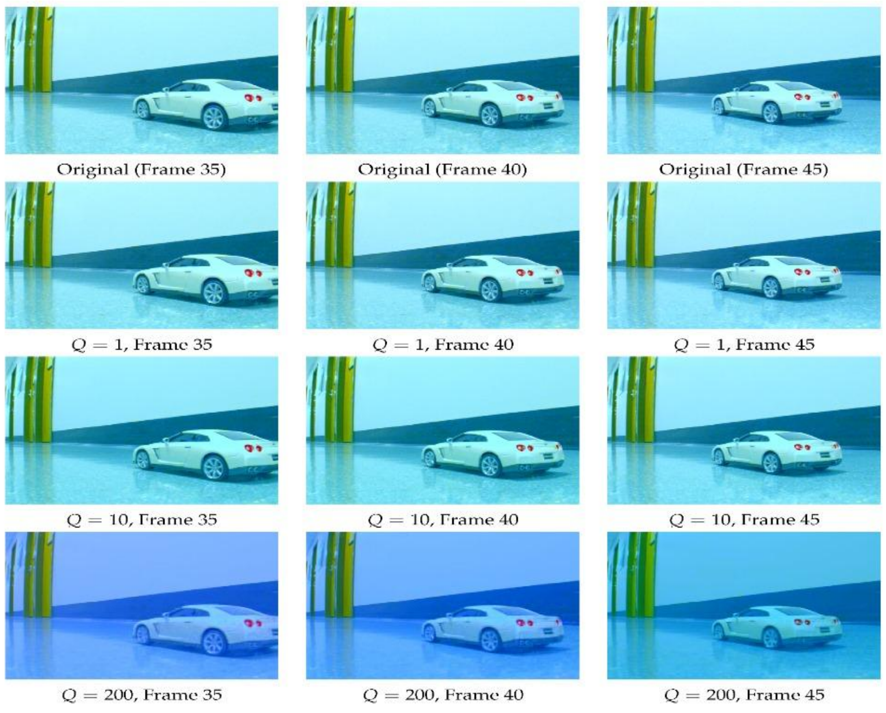


Figure 7. The SAIs (Nx=Ny=4) of the Car LFV at frames 35, 40, and 45. The first row displays the original pictures, while the second, third, and fourth rows display the reconstructed images at Q=1, Q=10, and Q=200 quantization levels.

Table 2. Different quantization levels affect PSNR and SSIM.

Value of Quantization	Number of bits in a pixel	PSNR (dB)	SSIM	Keeps energy in
1	0.22	51.21	0.00	0.00
3	0.36	44.15	0.88	0.00
5	0.17	42.42	0.88	0.00

Value of Quantization	Number of bits in a pixel	PSNR (dB)	SSIM	Keeps energy in
7	0.10	41.30	0.88	0.00
10	0.03	40.15	0.88	0.88

The exact DCT can be seen in Figure 8. You can do ADC in six different ways. There is the least amount of noise in the PSNR. For every frame of compression, it was a different bit. It was found that the PSNR drops for all three LFVs when the number of bits per frame goes down. It's also known as "greater compression." The picture below shows the PSNR and bits per frame for all three LFVs. A lot of the forms are the same. Car and Toy have the lowest LFV PSNR, which means they are also the lowest. It sounds like 41 dB. LFV David has a PSNR of about 45 dB and about 0.03 bits per frame. Table 2 shows that the LFV David has a smaller range than the others. This is why PSNR numbers are higher when frames have fewer bits. In Figure 9, you can see how SSIM changes when a frame has more or fewer bits. The SSIM going up at the same rate as the tax rate is unfair. The SSIM is better because the PSNR is better for the LFV David than for the Car and Toy LFVs. This LFV has the best PSNR and SSIM of the three tried. Still, this ADCT method has more difficult math than some of the others shown. But the SSIM doesn't work as well as some other ADCT tips. This is the case because they don't have many steps.

This ADCT method doesn't have a great PSNR, but it still works. You can get more done this way if you don't use exact DCTs. The Modified CB-2011 and PMC 2014 ways are more than ten times easier to give you an idea. We can see that ADCT usually makes the new LFVs worse than PSNR and SSIM. This is next to the whole DCT.

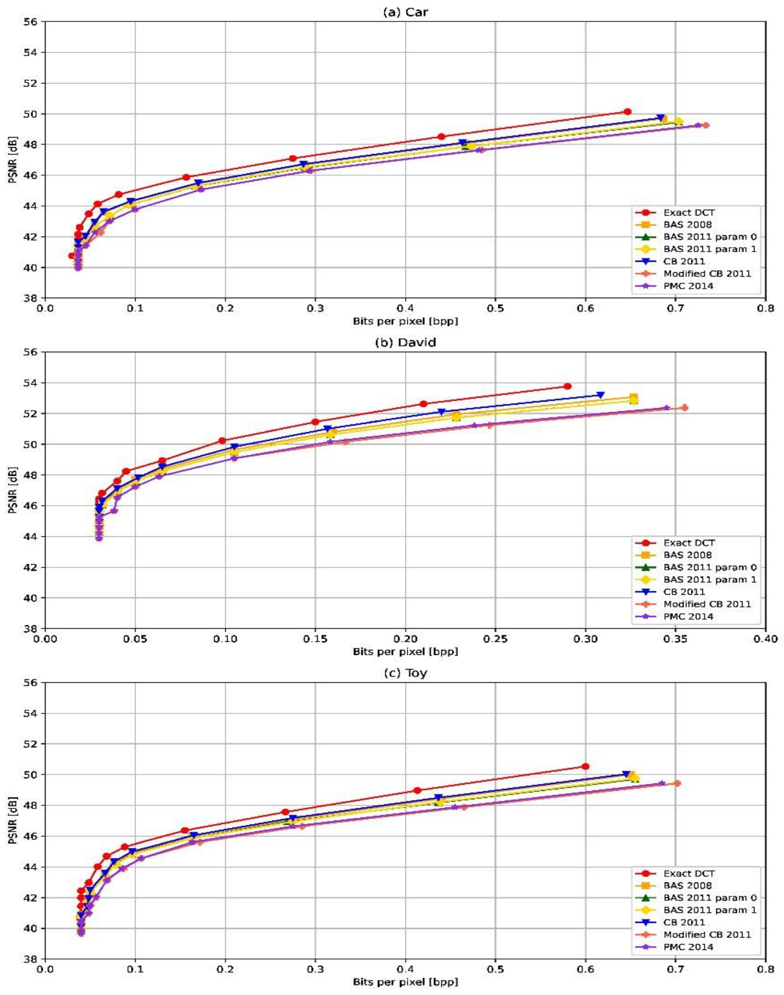


Figure 8. PSNR for bits per pixel utilizing BAS-2008, BAS-2011 for parameter 0, BAS-2011 for parameter 1, CB-2011, Modified CB-2011, PMC 2014, and the accurate 5-D DCT for LFVs (a) Car, (b) David, and (c) Toy.

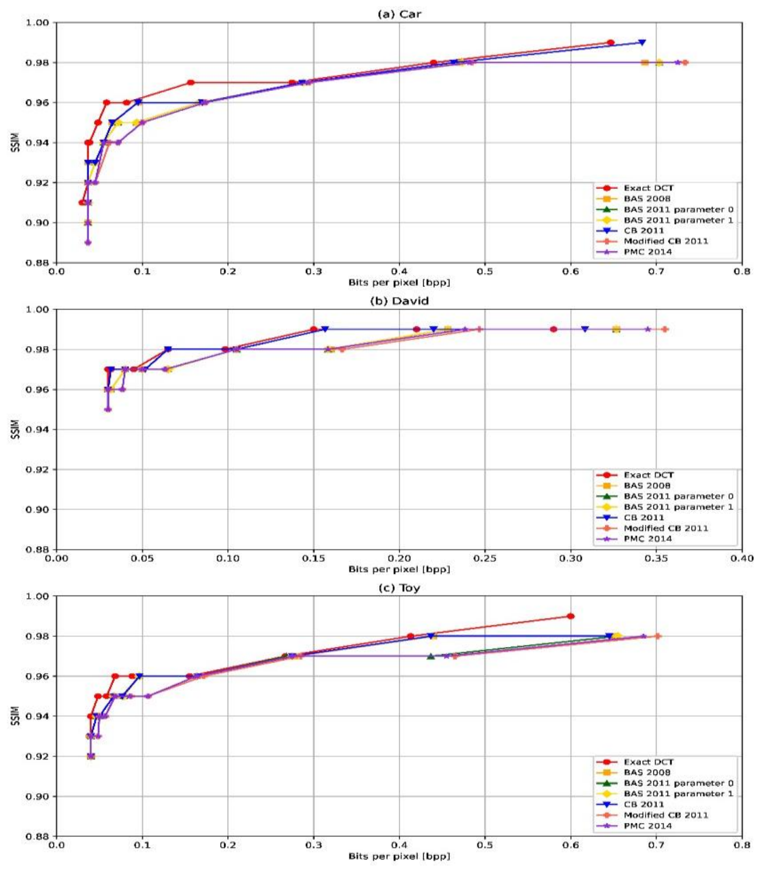
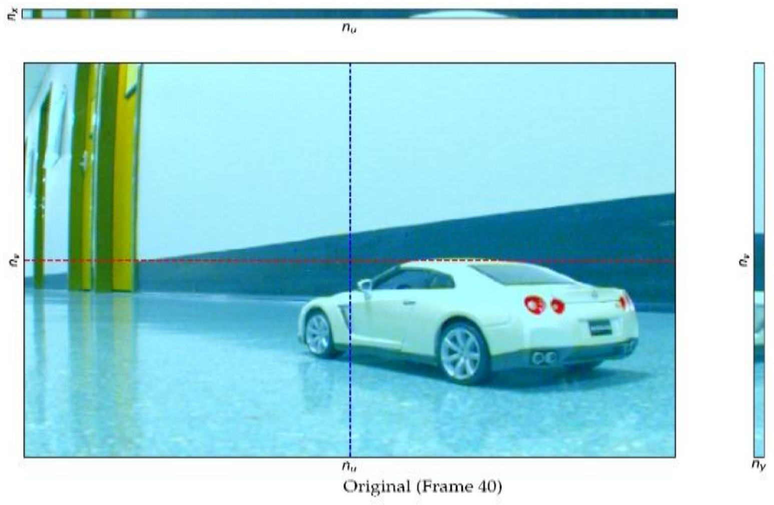


Figure 9. SSIM with bits per pixel for BAS-2008, BAS-2011 for parameter 0, BAS-2011 for parameter 1, CB-2011, CB-2011 PMC 2014, and the correct 5-D DCT for LFVs (a) Car, (b) David, and (c) Toy.

Before we use the suggested method to make them smaller, we look at a picture of the epi-polar plane to ensure the new LFVs are accurate. Remember that straight lines with changing slopes show the epipolar plane. These LFVs show edge and depth more. Figure 10 shows the epi-polar plane from the 40th LFV frame. It also shows the LFV that was made again, but this time, the number of compressions was 200. When closely examined, the new frame still has a lot of the straight-line shape. Even when the compression rate goes up, it is clear that the spin and depth are still right because of this difference. The PSNR and SSIM readings for ten frames of epi-polar plane images were added together to find the best compression setting. The results can be seen in Table 3. This is done so that more checks can be made. The PSNR number is over 65 dB, and the SSIM number is over 0.804. There is no doubt that the suggested method is a good way to keep track of the depth and angle information.



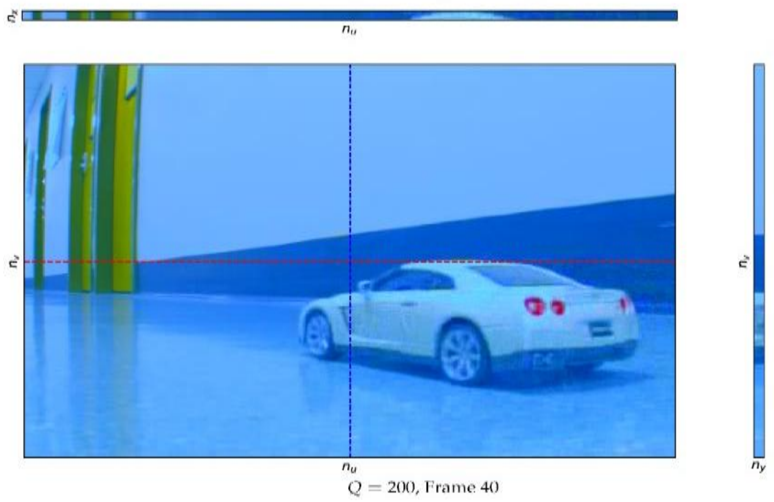


Figure 10. Epi-polar images of the original (top row) and reconstructed (bottom row) Car LFV at frame 40, quantized at Q=200. The reconstructed LFV preserves epi-polar plane images as straight lines.

Table 3. PSNR and SSIM values for horizontal and vertical epi-polar planar pictures with varied quantizations.

Val	**PSNR (dB)**		**SSIM**	
	Side to side	**Vertical**	**Side to side**	**Vertical**
1	00.032	97.640	0.000	1.000
5	77.811	84.948	0.850	0.854
10	72.636	79.816	0.800	0.826
50	62.456	58.565	0.724	0.764
200	60.174	54.772	0.703	0.720

5. CONCLUSIONS

LFVs record much information, making them hard to use in real time. That's too bad because sometimes they're helpful. 5-D ADCT makes shrinking 5-D linear functional vectors easy, and you don't need to use factors. Another way to make things easier is to break them up into parts. Things that get smaller are time, inside the view, between views, and inside the view itself. The suggested 5-D ADCT-based compression works better than the 2-D inter-view + 1-D temporal compression and the 2-D intra-view + 1-D temporal compression alone. When 0.03 bits per pixel, more than 250 times compression, were used, all three LFVs had PSNRs of more than 41 dB and SSIM indices of more than 0.9. An FPGA or other type of hardware must be used for this to work in or close to real-time. We're going to do these things to get what we want.

Declarations:

All authors declare that they have no conflicts of interest.

REFERENCES

[1] Bienik, J., Uhrina, M., Sevcik, L., & Holesova, A. (2023). Impact of packet loss rate on quality of compressed high resolution videos. Sensors, 23(5), 2744.

[2] Bruce, N. D. B., & Tsotsos, J. K. (2005). Saliency based on information maximization. In: Proceedings of the

18th International Conference on Neural Information Processing Systems, NIPS'05, pp. 155–162. MIT Press, Cambridge, MA, USA.

[3] C.-Y. Lu et al. Optimized projections for sparse representation-based classification Neurocomputing (2013)

[4] Chen, W.G.; Yu, R.; Wang, X. Neural Network-Based Video Compression Artifact Reduction Using Temporal Correlation and Sparsity Prior Predictions. IEEE Access 2020, 8, 162479–162490.

[5] Chen, Y., Mukherjee, D., Han, J., Grange, A., Xu, Y., Parker, S., et al. (2020). An Overview of Coding Tools in AV1: the First Video Codec from the Alliance for Open Media. APSIPA Trans. Signal Inf. Process. 9.

[6] Coding of Moving Video: High-Efficiency Video Coding (HEVC) ITU-T Recommendation H.265. Available online: https://handle.itu.int/11.1002/1000/14107 (accessed on 1 May 2023).

[7] H. M. Kwan, G. Gao, F. Zhang, A. Gower, and D. Bull, "HiNeRV: Video compression with hierarchical encoding based neural representation," in NeurIPS, 2023.

[8] Jeon, Minseong, and Kyungjoo Cheoi. "Efficient Video Compression Using Afterimage Representation." Sensors 24, no. 22 (2024): 7398.

[9] Joy, Helen K., Manjunath R. Kounte, Arunkumar Chandrasekhar, and Manoranjan Paul. "Deep Learning Based Video Compression Techniques with Future Research Issues." Wireless Personal Communications 131, no. 4 (2023): 2599-2625.

[10] Khatoonabadi, S. H., Bajić, I. V., & Shan, Y. (2015). Compressed-domain correlates of human fixations in dynamic scenes. Multimedia Tools and Applications, 74(22), 10057–10075.

[11] Kovtun, V.; Izonin, I.; Gregus, M. Model of functioning of the centralized wireless information ecosystem focused on multimedia streaming. Egypt. Inform. J. 2022, 23, 89–96.

[12] Ma, C., Liu, D., Peng, X., Li, L., & Wu, F. (2020). Convolutional neural network-based arithmetic coding for HEVC intra-predicted residues. IEEE Transactions on Circuits and Systems for Video Technology, 30(7), 1901–1916.

[13] Mochurad, L. (2024). A Comparison of Machine Learning-Based and Conventional Technologies for Video Compression. Technologies, 12(4), 52.

[14] Moffat, A. Huffman coding. ACM Comput. Surv. (CSUR) 2019, 52, 1–35.

[15] Mustafa, D.I.; Ali, I.A. Error Resilience of H. 264/Avc Coding Structures for Delivery over Wireless Networks. J. Duhok Univ. 2022, 25, 114–128.

[16] Pfaff, J., Helle, P., Maniry, D., Kaltenstadler, S., Samek, W., Schwarz, H., Marpe, D., & Wiegand, T. (2018). Neural network based intra prediction for video coding, in Applications of Digital Image Processing XLI, vol. 10752. International Society for Optics and Photonics, 2018, p. 1075213. 28.

[17] Rakhmanov, A.; Wiseman, Y. Compression of GNSS Data to speed up Communication to Autonomous Vehicles. Remote Sens. 2023, 15, 2165.

[18] Santamaria, M., Malamal Vadakital, V. K., Kondrad, L., Hallapuro, A., and Hannuksela, M. M. (2021). "Coding of Volumetric Content with MIV Using VVC Subpictures," in Proceeding of the IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP), Tampere, Finland, Oct. 2021 (IEEE).

[19] Serrano-Carrasco, D. J., Diaz-Honrubia, A. J., & Cuenca, P. (2021). Video Compression for Screen Recorded Sequences Following Eye Movements. Journal of Signal Processing Systems, 93(12), 1457-1465.

[20] Shao, D., Wang, N., Chen, P., Liu, Y. and Lin, L., 2024. A Novel Video Compression Approach Based on Two-Stage Learning. Entropy, 26(12), p.1110.

[21] Shilpa, B.; Budati, A.K.; Rao, L.K.; Goyal, S.B. Deep learning based optimized data transmission over 5G networks with Lagrangian encoder. Comput. Electr. Eng. 2022, 102, 108164.

[22] Sritharan, Braveenan, Chamira US Edussooriya, Chamith Wijenayake, R. J. Cintra, and Arjuna Madanayake. "Computationally Efficient Light Field Video Compression Using 5-D Approximate DCT." Journal of Low Power Electronics and Applications 15, no. 1 (2025): 2.

[23] Sullivan, G.J., Ohm, J.-R., Han, W.-J., Wiegand, T.: Overview of the high-efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. 22(12), 1649–1668 (2012)

[24] Terms of Reference of the Joint Collaborative Team on 3D Video Coding Extension Development ITU-T SG 16 document TD 532 (Plen/16) and ISO/IEC MPEG document N12830 Geneva Switzerland, Apr. 2012.

[25] TranQ.N. et al. Video frame interpolation via down–upscale generative adversarial networks Comput. Vis. Image Underst. (2022)

[26] Wiseman, Y. (2024). Video compression prototype for autonomous vehicles. Smart Cities, 7(2), 758-771.

[27] Zhang, Z.T., Yeh, C.H., Kang, L.W., & Lin, M.H. (2017). Efficient CTU- based frame coding for HEVC based on deep learning, in Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). IEEE, pp. 661–664

[28] Zhao, Y., Po, L. M., Cheung, K. W., Yu, W. Y., & Rehman, Y. A. U. (2021). SCGAN: saliency map-guided colourization with the generative adversarial network. IEEE Transactions on Circuits and Systems for Video Technology, 31(8), 3062–3077. https://doi.org/10.1109/TCSVT.2020.3037688.