# Video Compression using Generative Models

**Himank Goel**
301435635
hga50@sfu.ca

**Pranav Sharma**
301445726
psa131@sfu.ca

**Heng Zhou**
301474872
heng_zhou@sfu.ca

## Abstract

Video Compression is an important technique used to reduce the video storage burden or to ease the bandwidth requirements to transfer these videos. In an industrial setting, any video compression algorithm should be able to reconstruct the video frames as close as possible so as to not negatively affect the performance of downstream tasks. In this report, we evaluate the efficiency of compression techniques under industrial constraints. Further, we examine two methods of improving the baselines - using IMLE objective and introduction of Total Variation Loss. Extensive experimental results demonstrate the benefits and pitfalls of using these approaches for video compression tasks.

## 1 Introduction

Currently, the amount of video traffic is growing exponentially and so is the availability of high-resolution videos from mobile devices. However, this also increases the storage burden and bandwidth requirements to transfer these videos. For example, article [11] states that we require an approximately 12 Gbps bitrate to carry uncompressed 4K videos. So video compression techniques are crucial for video transfer, processing, and storage. In recent literature, deep learning has introduced a new line of thinking in end-to-end learned image compression. These works generally involve learning a generative network jointly with a prior distribution network by maximizing the ELBO on the image likelihood. However, adapting these methods from a single image to the video domain is still an open problem. Moreover, utilizing the potential of deep learning approaches for video compression on real-world mobile devices is an immense challenge due to the computational complexity of deep networks and the limitations of mobile hardware.
We designed a system to perform video compression and reconstruction to overcome high-resolution video storage and transfer issues. The information richness of the reconstructed video should match the original video. The primary input and output are both high-resolution videos as demonstrated in Figure 2. The only difference is that the output video is a reconstructed high-resolution video generated from the compressed video. Since high-resolution videos contain lots of helpful information, the compression-reconstruction process should keep as much original information as possible, and the reconstructed videos should be able to be used for downstream tasks and meet industrial requirements.

## 2 Related Work

Various frameworks have been introduced to solve the problem of video compression. We divide these existing methods into two relevant categories: Traditional Methods, and Deep Learning Methods.

### 2.1 Traditional Methods

Traditional methods for Video compression involved working with downscaling resolutions, and manipulation on the interframe levels devised from Group Of Pictures (i.e. I-Frame, B-Frame, P-frames). MPEG-4 [12] is one such traditional method widely used to date. H.265[2, 13] is another

Preprint. Under review.

codec very successful with video compression tasks that relies on hand-crafted transformations to analyze the redundancy within the videos. We use H.265 as one of the baselines for our experiments.

### 2.2 Deep Learning Methods

Deep learning methods are usually a step up from the traditional methods, involving the manipulation of spatial-temporal domain data. Some approaches also involve transform and entropy encoding for lossless compression strategies. Motion compensation is also one of the problem domains successfully handled by predictive deep learning methods. Deep learning methods use different coding strategies, one such type is residual coding which is highly successful for video compression. End-to-end learned video compression was given the impetus from DVC [7] which was the first such framework and was based on temporal predictive coding. Liu et al. [6] propose a GAN-based approach to compress individual frames to a latent space and then each subsequent frame is conditioned on the previous representations. Ma et al. [8] uses an enhanced residual non-local block to improve representational capacity along with a novel relativistic sphere GAN loss function. Ho et al. [3] propose the use of conditional augmented normalizing flows to form a purely conditional coding framework.
For our experiments, we used Self-Conditioned Probabilistic Learning of Video Rescaling(SelfC) [14], which uses a self-conditioned framework for a video downscaling task, which is a precursor to the actual video compression task. We also took a look at Hybrid Spatial-Temporal Video Compression(HSTVC) [4] which is a state-of-the-art, pure conditional coding deep learning model and acts as a nice comparative model for our results.

## 3 Baseline Experiments

In order to evaluate the existing methods, we devised a set of experiments that allowed us to establish baseline results. These experiments provided not only a starting point against which we could compare the results of our main experiment but also validated some of the hypotheses involved in our project. For these baseline experiments, we choose a traditional compression technique - H265, a rescaling-based compression method - SelfC, and a state-of-the-art deep learning-based video compressor - HSTVC.

### 3.1 Dataset

For our project, we chose a proprietary dataset of short videos of trucks containing rocks. Each video consists of multiple bursts of five frames. We managed to gather 174 videos of varying lengths which resulted in a total of 7434 individual frames. These videos are of size $[2048 \times 1536]$. In addition to this, we created a validation dataset with six video sequences that each contained 40 frames.

### 3.2 Out-of-the-box Comparison

The first experiment we conducted was to test the three methods using their out-of-the-box implementations on our validation dataset. This allowed us to understand the efficacity of these methods on a common dataset thus allowing us to compare them directly. In order to compare the performance of these models, we chose three metrics - Peak Signal to Noise Ratio (PSNR), Structural Similarity (SSIM), and Bits per Pixel (BPP). The first two of these metrics are to evaluate the quality of the reconstructed video frames and hence the higher the value of these metrics the better. BPP, however, should be as low as possible since it is a quantitative measure of the compression ratio achieved.
Shown in Table 1. are the results of this experiment for all three methods against all three metrics. From the table, it was inferred that H265 performs the best reconstruction since it had the highest values for PSNR and SSIM. However, at the same time, it also had the worst compression ratio since it had the highest BPP. In the contrast, HSTVC performs the poorest reconstruction but at the best compression ratio. The results for SelfC indicated that it splits the difference in both reconstruction quality and the compression ratio and hence suited our requirements the best.

### 3.3 Memory Consumption vs. Performance Tradeoff

The next experiment we conducted was to verify the ability of our model to work on an edge device i.e., with limited GPU memory consumption. For this experiment, we set a constraint of 4GB as the

highest GPU memory available in an edge device. After consulting with our industrial partners, we determined that in production, only the encoding part of our model needs to run on the edge device, the decoding, however, can run on a server at a later point in time. So in this experiment, we only made modifications to our encoding pipeline while keeping the decoding part exactly the same.
In the encoding pipeline, we found two possible avenues to reduce memory usage -

- We reduced the number of frames that were processed in a single iteration from three to one.
- We switched our model from full precision (32 bits) to half-precision (16 bits).

We dubbed this newer model with both of these changes in the encoding pipeline as SelfC-lite. However, in general, both of these techniques also have an adverse impact on the model's performance. In order to validate that, we measured the PSNR, SSIM, BPP, and peak GPU memory usage for the original model and SelfC-lite the results of which are shown in Table 2.
We observed that there was little to no difference in the reconstruction quality of the two models. This was probably because the decoding pipeline was still running at the same parameters. We did, however, observe a slight increase in BPP, which makes sense due to changes in the encoding pipeline. However, we observed more than 50% decrease in GPU usage for SelfC-lite which is more than enough to satisfy our constraint. Hence, we determined that this tradeoff is favorable enough to proceed in this direction.

| | **H265** | **SelfC** | **HSTVC** |
|---|---|---|---|
| PSNR ↑ | 46.8 | 41.24 | 38.17 |
| SSIM ↑ | 0.996 | 0.9907 | 0.97 |
| BPP ↓ | 0.24 | 0.128 | 0.064 |

Table 1: Out-of-the-box Experiment Results

| | **SelfC** | **SelfC-lite** |
|---|---|---|
| PSNR ↑ | 41.24 | 41.23 |
| SSIM ↑ | 0.9907 | 0.9908 |
| BPP ↓ | 0.128 | 0.134 |
| GPU Mem (MiB) ↓ | 8148 | 3845 |

Table 2: Memory Consumption vs. Performance Tradeoff

### 3.4 Model Training

The next step was to train and fine-tune our SelfC-lite model using the proprietary dataset. In order to train the model we used 1062 video sequences, which had 7 frames each. We used the pre-trained model to initialize the weights and we used Adam Optimizer with an initial learning rate set to $10^{-4}$. Further, we evaluated our newest trained model using the same metrics as before to give us our updated baseline which is shown in Table 3.

| | **SelfC-lite (Trained)** | **SelfC-lite (Pre-trained)** |
|---|---|---|
| PSNR ↑ | 41.94 | 41.23 |
| SSIM ↑ | 0.9920 | 0.9908 |
| BPP ↓ | 0.147 | 0.134 |

Table 3: Results after training the model

### 3.5 Downstream Performance

The last baseline experiment we performed was to establish a starting point for the performance of the reconstructed video frames in a downstream application. As mentioned earlier, the goal of any such compression algorithm is to be compatible with any downstream tasks that an organization might have. To that end, we picked rock instance segmentation as our sample application, the performance on which, we evaluated our reconstructed video frames. We chose this application because it is a substantially challenging task and hence would be a good measure of the ability of the compression algorithms.
This task, however, is just a qualitative assessment of the performance. Hence, we only present a visual comparison of the instance segmentation quality on - ground truth video frames, reconstructed frames from H265, and reconstructed frames from the trained SelfC-lite model in Figure 5.
As can be observed from the figure, the segmentation quality on the ground truth image is quite superior to that of both H265 reconstructed and SelfC reconstructed images.

# 4 Method

Based on the experimental results to establish the baselines, we concluded that, SelfC provides the desired amount of compression at a reasonably good reconstruction performance. Hence, we decided to use SelfC as our underlying algorithm on top of which we implemented our improvements in pursuit of better reconstruction performance. As we already mentioned, the compression ratio was under the target requirements, hence we did not make any changes to the encoding pipeline. To obtain better reconstruction results we proposed the following two approaches -

- training the decoding pipeline using Implicit Maximum Likelihood Estimation (IMLE) objective
- modifying the training loss function by incorporating the Total Variation Loss

In the next few subsections, we first explain the architecture of vanilla SelfC and then dive into the integration of the above two proposed methods into this architecture.

## 4.1 SelfC Architecture

The SelfC Architecture can be divided into two components: Encoder and Decoder. The encoding step (Figure 3) for SelfC involves analyzing a subset of video frames and decomposing it to high frequencies and low frequencies. The high frequencies are discarded and the low frequencies are sent for quantization where we store the low-resolution video frames.
The decoding step (Figure 4) involves reading the low-resolution frames and passing them to the STP-Net which is a feature extraction module. These features are based on - (a) local modeling of frames that extract information from the prior frame and subsequent frame for the current frame and (b) spatial aggregation to figure out long-range temporal dependencies for the images. This information is fed to a 3-layer multi-layer perceptron(MLP) that outputs our predicted Gaussian Mixture Model Parameters. These parameters are used to generate high frequencies that are fed into the synthesizer along with the low-resolution video frames giving out our reconstructed video frames.

## 4.2 Implicit Maximum Likelihood Estimation (IMLE)

Implicit Maximum Likelihood Estimation (IMLE)[5] is a training objective used to train implicit generative models. IMLE can trivially be extended to model conditional distributions by separately applying IMLE to each member of a family of distributions [10]. Given a generator parameterized by $\theta$ as $T_\theta(.)$, which takes in the input $x_i$ and a random code $z_{i,j}$ and outputs from a sample from $p(.|x_i)$, the IMLE objective optimizes the following:

$$\min_\theta \mathbb{E}_{\mathbf{z}1,1,\ldots\mathbf{z}n,m} \sim \mathcal{N}(0,I)[\sum_{i=1}^{n} \min_{j\in 1,\ldots,m} d(T_\theta(x_i, z_{i,j}), y_i)], \tag{1}$$

where $y_i$ is the observed output that corresponds to $x_i$, $d(.,.)$ is a distance metric and $m$ is a hyperparameter. We use a simple Euclidean distance as our distance metric. Figure 6 shows the conditional IMLE training procedure.
Training IMLE can be a very memory-intensive task due to the generation of the random code and using nearest neighbor search to establish matches between generated samples and real data. In order to get around this, we froze parts of the SelfC architecture and only trained the core generator module from the model. As shown in Figure 7, where we have the entire SelfC architecture(encoding and decoding). In this image, we have faded out the parts of the architecture we freeze which are - the analyzer, quantization module, and the temporal feature estimation of the STP-Net. What we are left with is the small three-layer MLP which takes in the temporally encoded feature vector of the low-resolution image and returns the high-resolution feature vector. In this case, our input becomes the output of the STP-Net $f_l^{'}$(shown in yellow), and our ground truth $f_h$ (shown in red) is the output from the analyzer.

## 4.3 Total Variation Loss

To analyze our generated frames, we plotted difference maps between the reconstructed frames from SelfC and the ground truth frames. We show a couple of the difference maps in Figure 8 in the three

color channels. We observed that the fact that these maps were quite noisy was probably the reason that the downstream processes get affected. In order to get around this we found Total Variation(TV) Loss. TV loss was introduced into super-resolution by Aly et al[1]. It is defined as the sum of the absolute differences between neighboring pixels and measures how much noise is in an image.

$$l_{TV} = \frac{1}{HWC} \sum_{i,j,k} \sqrt{(I_{i,j+1,k} - I_{i,j,k})^2 + (I_{i+1,j,k} - I_{i,j,k})^2}, \tag{2}$$

where, $H$, $W$, and $C$ represent the height, width, and the number of channels in the image.

# 5 Experiments

To test our above-proposed methods we used the SelfC-lite model which was pre-trained on our proprietary dataset as a starting point. We simply refer to this model as SelfC from here on. To evaluate the performance of our methods, we used the validation data we created earlier as mentioned in Section 3.1. The two approaches were trained using different experimental settings as listed below.

## 5.1 Experimental Setting - SelfC + IMLE

As mentioned earlier, we froze part of our SelfC architecture and only trained the three-layer MLP. In order to do this, we exported the low-res feature vector and the corresponding high-res feature vector for individual frames. This resulted in a total of 7434 training pairs. The low-res vector is of the size $[24 \times W \times H]$ and the high-res feature vector has the dimension $[12 \times W \times H]$. We used a latent code of dimension $[8 \times W \times H]$. We used some train time augmentations including - random crops to $432 \times 432$, rotation, and flips. We generated 120 samples per training pair as part of the IMLE algorithm. We trained this model using Euclidean Loss for 250,000 iterations.

## 5.2 Experimental Setting - SelfC + TV Loss

The loss function used in the baseline SelfC model is shown below -

$$\mathcal{L}_{selfc} = \lambda_1 \mathcal{L}_c + \lambda_2 \mathcal{L}_{mimic} + \lambda_3 \mathcal{L}_{pen} + \lambda_4 \mathcal{L}_{recons}, \tag{3}$$

where $\lambda_1$, $\lambda_2$, $\lambda_3$, and $\lambda_4$ are the balancing parameters and $\mathcal{L}_c$, $\mathcal{L}_{mimic}$, $\mathcal{L}_{pen}$, and $\mathcal{L}_{recons}$ are the various components of the loss function details of which can be referred to in the original paper. For integrating the TV loss into the existing loss function we develop the following formula -

$$\mathcal{L}_{selfc\_tv} = \mathcal{L}_{selfc} + \lambda_5 \mathcal{L}_{TV}, \tag{4}$$

where we $\lambda_5$ is a hyperparameter set to 0.4 in our experiments. Unlike [9] which adds this loss function on the reconstructed image, we calculate the loss on the difference between the ground truth video frame and reconstructed video frame.

## 5.3 Quantitative Results

We evaluate our two methods on three metrics, Peak Signal to Noise Ratio (PSNR), Structural Similarity (SSIM), and Bits per Pixel (BPP). The former measures the reconstruction quality of the video frames, while BPP measures the number of bits per pixel in the compressed images. As shown in Table 4, training SelfC with the IMLE objective degrades the performance in terms of the reconstruction quality by a significant amount. On the other hand, training SelfC using TV loss improves the reconstruction quality. The BPP values remain almost consistent for the two methods in comparison to the baseline which makes sense since we do not modify the encoding pipeline.

| | **SelfC** | **SelfC + IMLE** | **SelfC + TV loss** |
|---|---|---|---|
| PSNR ↑ | 41.94 | 32.81 | 43.08 |
| SSIM ↑ | 0.9920 | 0.9012 | 0.9957 |
| BPP ↓ | 0.147 | 0.151 | 0.144 |

Table 4: Results from the two methods

### 5.4 Qualitative Results

We show the results of our method and baselines in Figure 9. Visually the results from all four sources look identical. Therefore shown in Figure 1 are the histograms of all four results in the RGB color space. In this visualization, we can see that vanilla SelfC tends to intensify the changes in the histogram and hence is very noisy. This is resolved by the SelfC model trained with the TV loss where we can see the spikes are not quite intensified. SelfC trained with the IMLE objective also removes the aberrations but it makes significant changes to the histogram plot for e.g., the blue channel differs significantly between the values 50 and 100.

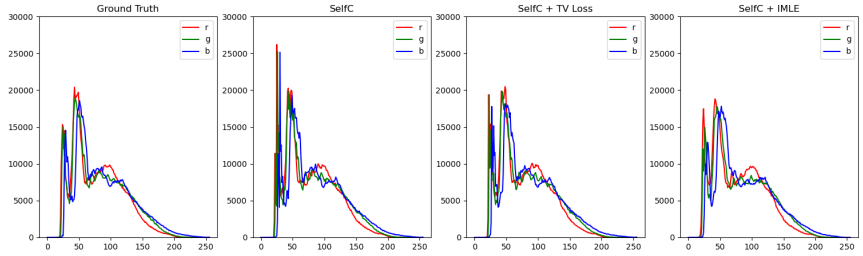


Figure 1: Qualitative Results: Histogram

We also show the qualitative results from the downstream task of instance segmentation in Figure 10. As can be observed, the segmentation result of the reconstructed image from the model trained using TV Loss is quite comparable to the ground truth. On the contrary, the model trained with IMLE loss produces images that break down the big rock into many smaller components thus failing to identify any boulder (big rock) in that spatial region.

### 5.5 Analysis

From the results we presented above, we inferred that vanilla SelfC suffered from introducing some noise into the reconstructed frames, which led to over-sharpening of the image, and hence all the edges got amplified - even those on a single big rock which led to incorrect segmentations. Training the same model with Total Variation loss mitigates this issue since it enforces the removal of noise. Hence only the appropriate edges get reconstructed during the decoding phase. Training using IMLE also smoothens the image as we show in the histograms, but it does so at the cost of shifting the color space of the image. This might be due to the ability of IMLE to generate unobserved realities which in this case is detrimental to the purpose of the model.

## 6 Conclusion

In this project, we proposed a video compression framework for edge devices based on an existing model. Extensive experimentation demonstrates the usability of adding Total Variation Loss to this architecture. We also demonstrate the inefficiency of using the IMLE objective for this task. We believe that with this survey as a basis, future research and prototyping could improve the results further.

## 7 Contribution

Heng Zhou was involved extensively in performing experimentations with the traditional compression techniques out of which we chose H265 as our baseline. Apart from this, he worked on data pre-processing for training the models. Pranav Sharma worked with the Hybrid Spatio Temporal Video Compression - generating metrics for the validation data to form a baseline. Himank Goel designed the SelfC experiments and improvements in terms of the introduction of TV loss and IMLE.

## References


[1] Hussein A Aly and Eric Dubois. Image up-sampling using total-variation regularization with a new observation model. *IEEE Transactions on Image Processing*, 14(10):1647–1659, 2005.

[2] High Efficiency Video Coding (HEVC) Family, H.265, MPEG-H Part 2. Sustainability of digital formats: Planning for Library of Congress Collections. `https://www.loc.gov/preservation/digital/formats/fdd/fdd000530.shtml`, Nov 2020. Online; accessed 14 December 2022.

[3] Yung-Han Ho, Chih-Peng Chang, Peng-Yu Chen, Alessandro Gnutti, and Wen-Hsiao Peng. Canf-vc: Conditional augmented normalizing flows for video compression. *arXiv preprint arXiv:2207.05315*, 2022.

[4] Jiahao Li, Bin Li, and Yan Lu. Hybrid spatial-temporal entropy modelling for neural video compression. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 1503–1511, 2022.

[5] Ke Li and Jitendra Malik. Implicit maximum likelihood estimation. *arXiv preprint arXiv:1809.09087*, 2018.

[6] Bowen Liu, Yu Chen, Shiyu Liu, and Hun-Seok Kim. Deep learning in latent space for video prediction and compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 701–710, 2021.

[7] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[8] Di Ma, Fan Zhang, and David R Bull. Cvegan: a perceptually-inspired gan for compressed video enhancement. *arXiv preprint arXiv:2011.09190*, 2020.

[9] Hai Nguyen-Truong, Khoa NA Nguyen, and San Cao. Srgan with total variation loss in face super-resolution. In *2020 7th NAFOSTED Conference on Information and Computer Science (NICS)*, pages 292–297. IEEE, 2020.

[10] Shichong Peng and Ke Li. Generating unobserved alternatives. *arXiv preprint arXiv:2011.01926*, 2020.

[11] B. Samis. Back to basics: GOPs explained | Amazon Web Services, 2022.

[12] Thomas Sikora. The mpeg-4 video standard verification model. *IEEE Transactions on circuits and systems for video technology*, 7(1):19–31, 1997.

[13] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[14] Yuan Tian, Guo Lu, Xiongkuo Min, Zhaohui Che, Guangtao Zhai, Guodong Guo, and Zhiyong Gao. Self-conditioned probabilistic learning of video rescaling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4490–4499, 2021.

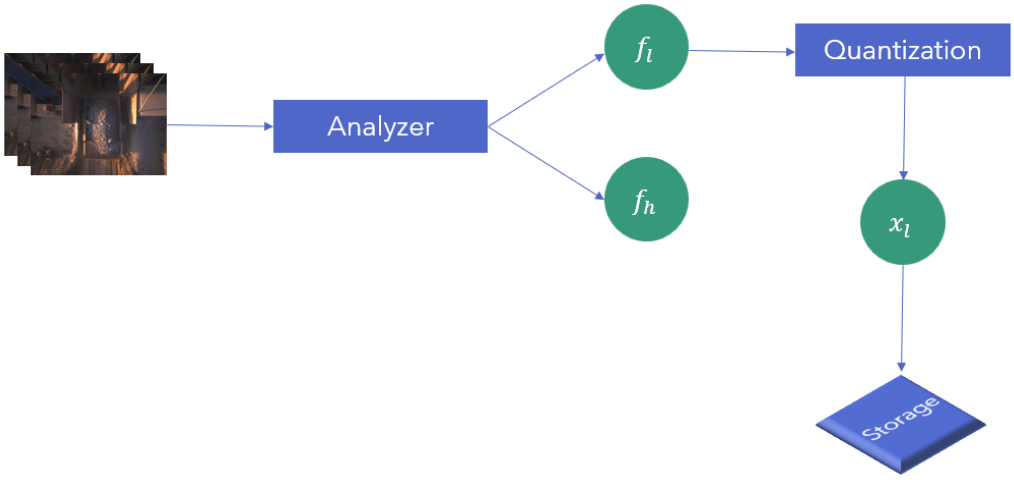


Figure 3: SelfC Encoder Architecture

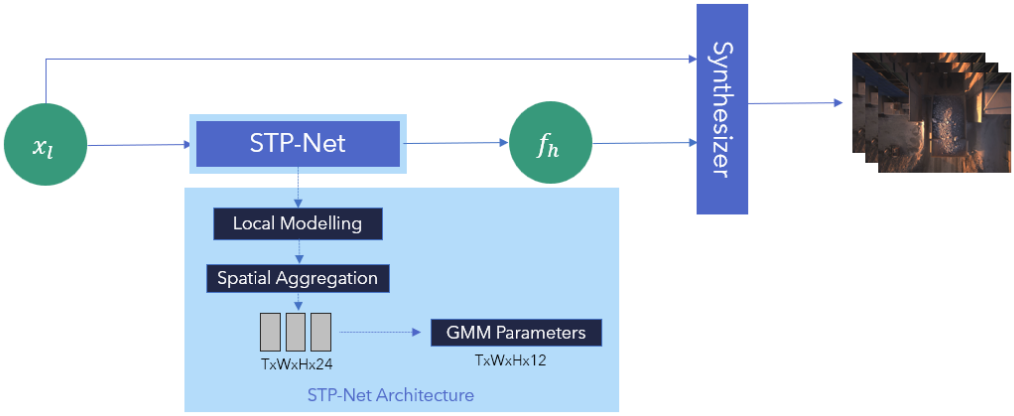


Figure 4: SelfC Decoder Architecture

# A Appendix

## A.1 Supplementary Images

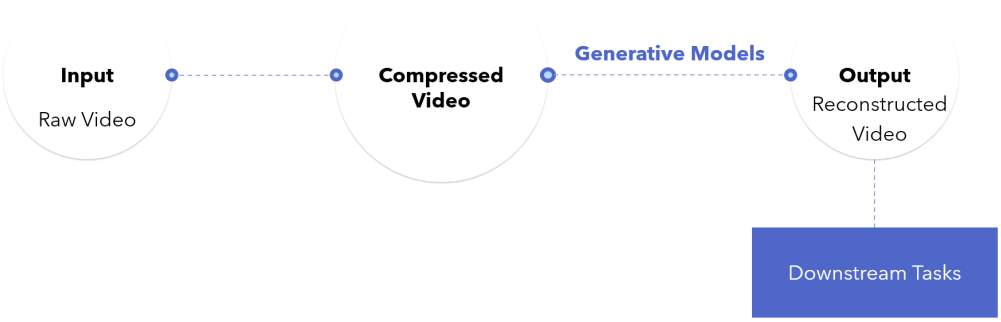


Figure 2: General Pipeline

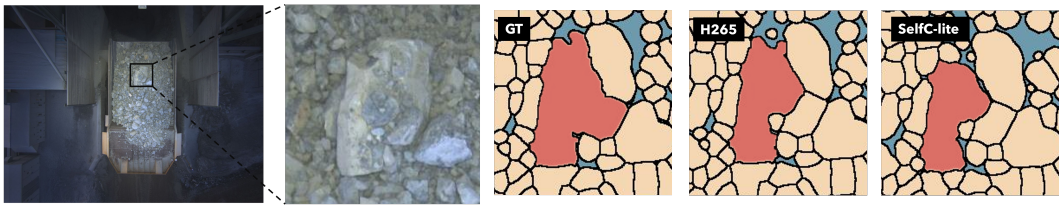


Figure 5: Instance Segmentation comparison on a sample rock from an image from the validation dataset. From left to right: Image from the validation dataset, zoomed-in rock, instance segmentation result on ground truth image, H265 reconstructed image, SelfC-lite reconstructed image

**Algorithm 1** Conditional IMLE Training Procedure

**Require:** The set of inputs $\{\mathbf{x}_i\}_{i=1}^{n}$ and the set of corresponding observed outputs $\{\mathbf{y}_i\}_{i=1}^{n}$
Initialize the parameters $\theta$ of the generator $T_\theta$
**for** $p = 1$ **to** $N$ **do**
  Pick a random batch $S \subseteq \{1, \ldots, n\}$
  **for** $i \in S$ **do**
    Randomly generate i.i.d. $m$ latent codes $\mathbf{z}_1, \ldots, \mathbf{z}_m$
    $\tilde{\mathbf{y}}_{i,j} \leftarrow T_\theta(\mathbf{x}_i, \mathbf{z}_j)\ \forall j \in [m]$
    $\sigma(i) \leftarrow \arg\min_j d(\mathbf{y}_i, \tilde{\mathbf{y}}_{i,j})\ \forall j \in [m]$
  **end for**
  **for** $q = 1$ **to** $M$ **do**
    Pick a random mini-batch $\tilde{S} \subseteq S$
    $\theta \leftarrow \theta - \eta \nabla_\theta \left( \sum_{i \in \tilde{S}} d(\mathbf{y}_i, \tilde{\mathbf{y}}_{i,\sigma(i)}) \right) / |\tilde{S}|$
  **end for**
**end for**
**return** $\theta$

Figure 6: Conditional IMLE training procedure

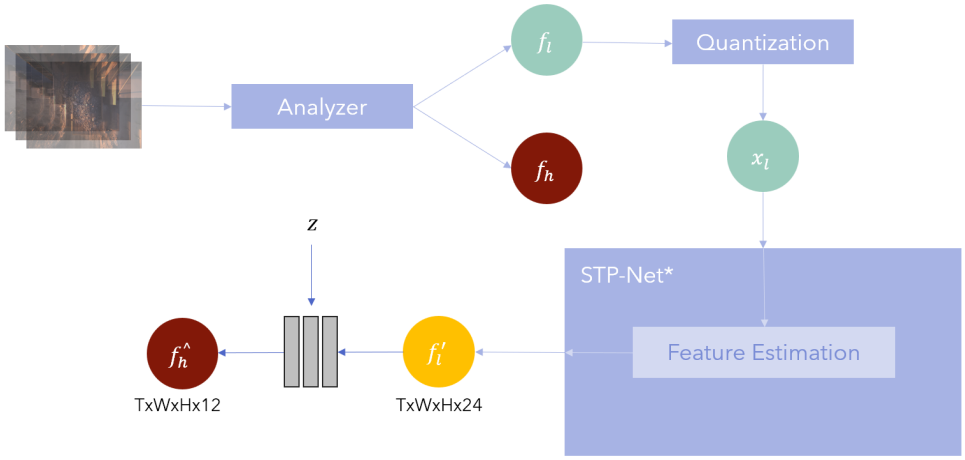


Figure 7: SelfC plus IMLE architecture

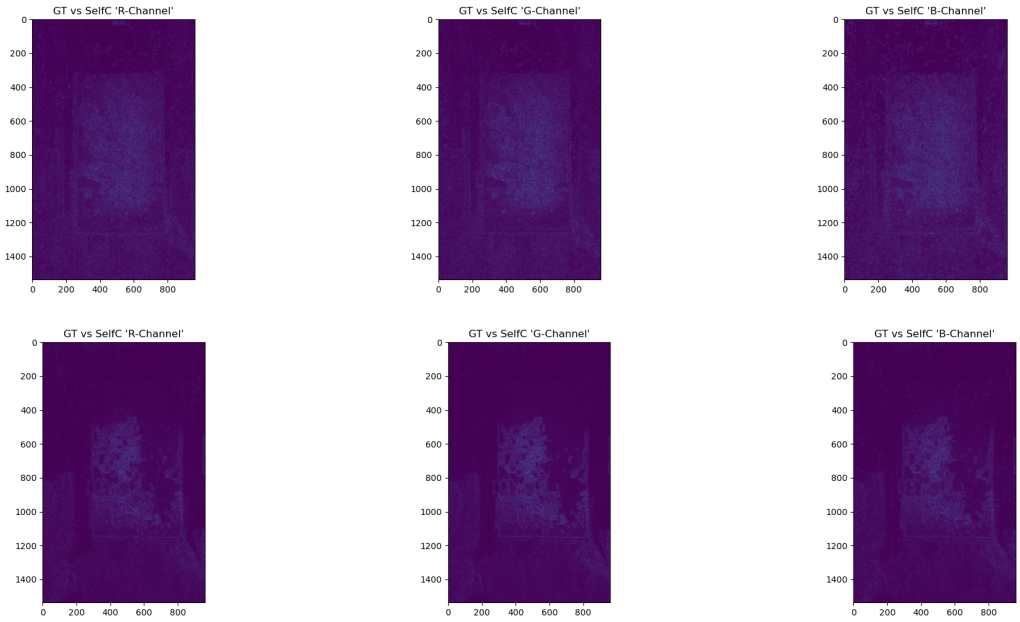


Figure 8: Difference maps for two frames in three color spaces

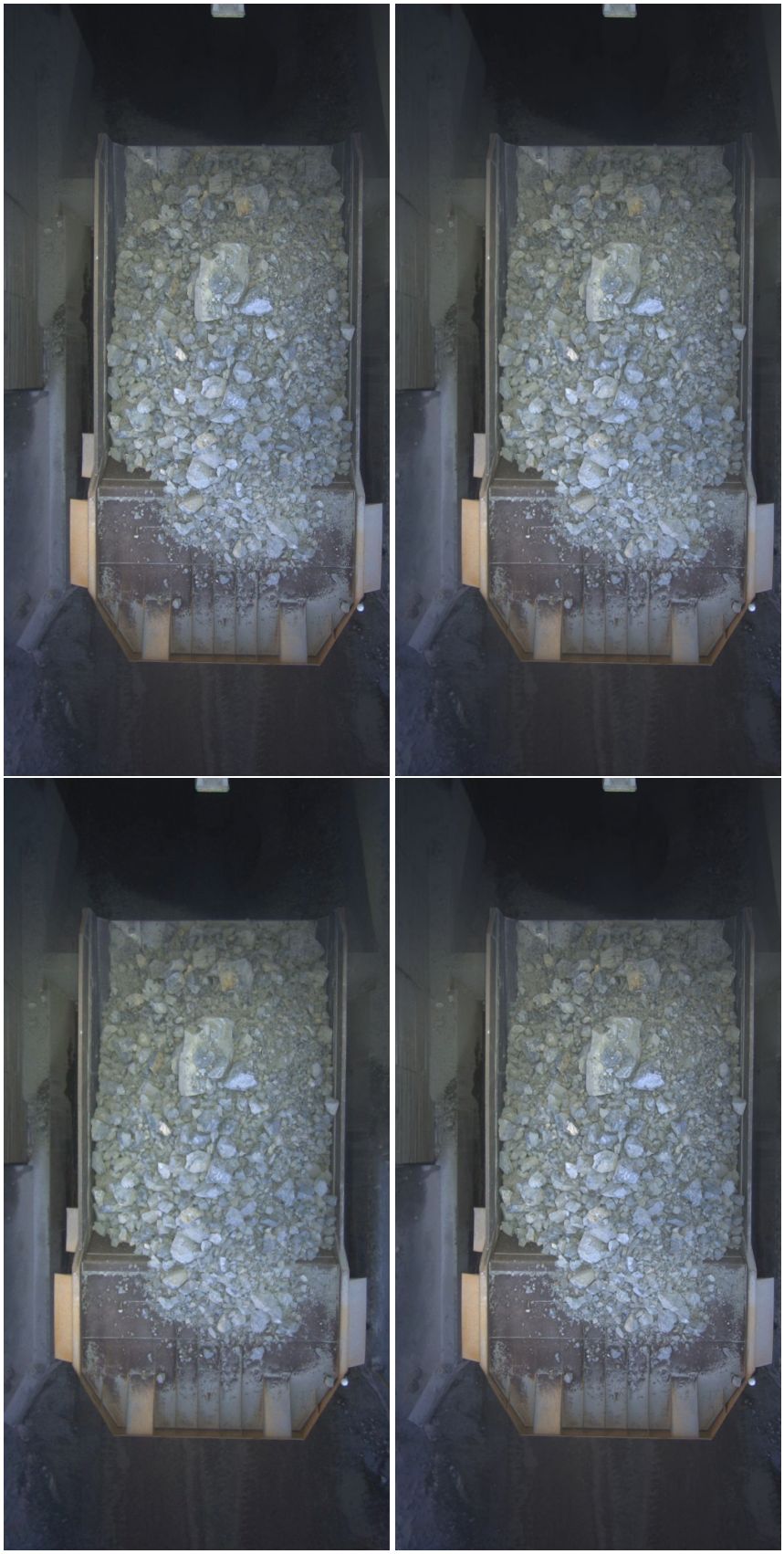

Figure 9: Qualitative Results: Top-left - Ground Truth, Top-Right - SelfC, Bottom-Left - SelfC + IMLE, Bottom-Right - SelfC + TV Loss

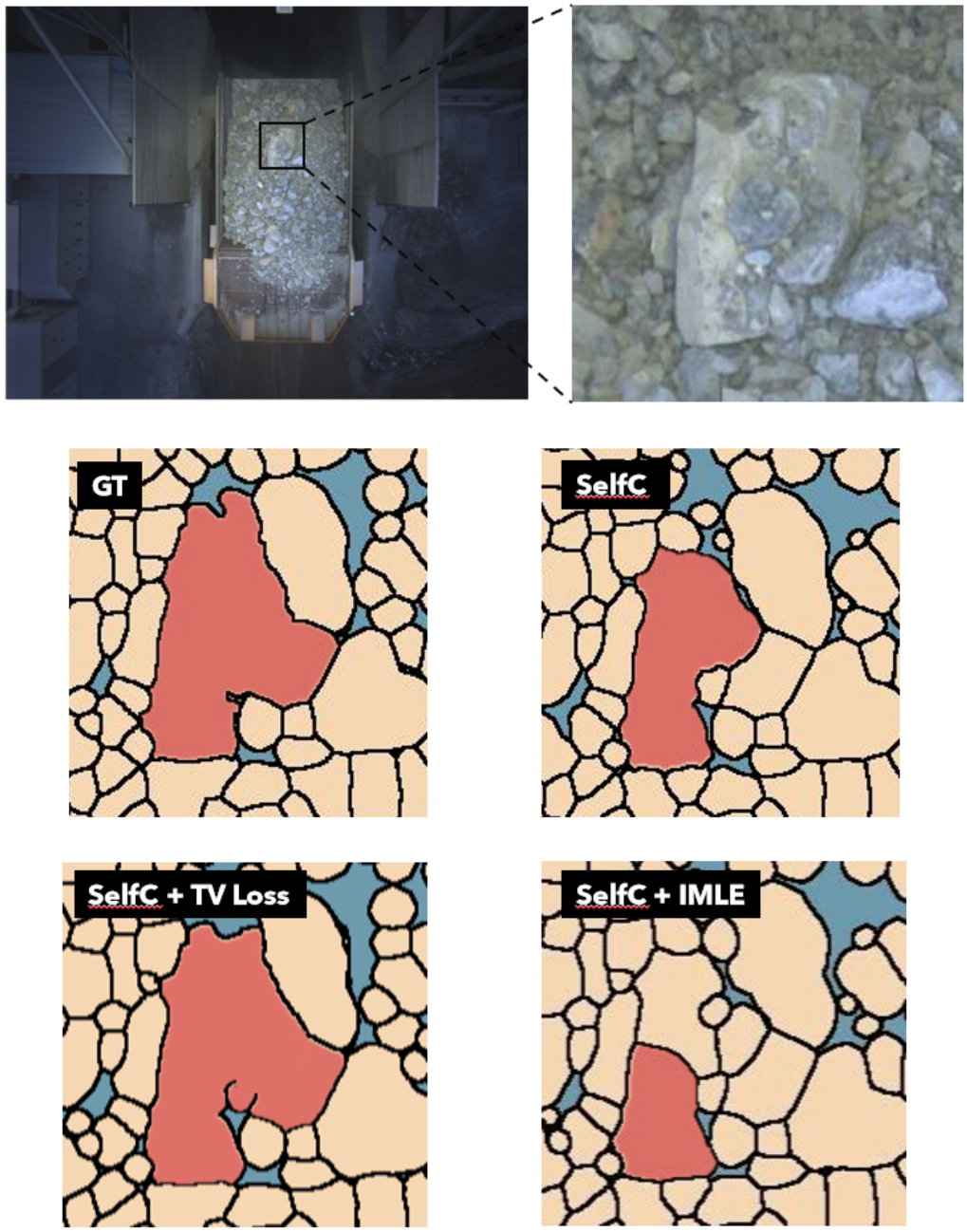


Figure 10: Instance Segmentation Results: Top-left - Ground Truth, Top-Right - SelfC, Bottom-Left - SelfC + IMLE, Bottom-Right - SelfC + TV Loss