Conditional GANs for Video Prediction

Visual model for robot control

Giulia Ghisolfi, g.ghisolfi@studenti.unipi.it, 664222

Master Degree in Computer in Science (AI Curriculum), University of Pisa
Robotics course (387AA)
Academic Year: 2023-2024

Outline

Introduction

Problem

TASK:

Develop a visual model for robot control that utilizes visual feedback from recorded images of the robot's movements, using previous frames as input to inform future predictions.

AGENT:

Soft robot interacting in a simulated environment, with and without objects.

OUTPUT:

Given a video sequence representing the robot's movements in the environment, predict future frames.

Solution Approach

To generate future frames, we use a **GAN model conditioned** on the **robot's actions**.

By conditioning the GAN on robot actions, the model **generates realistic future frames**.

Data Preparation

Dataset Overview

We worked with **two datasets**, one **with objects and one without**.

The images are **64x64 pixels**, in **RGB scale**, and each transition between two images is **conditioned** by a **real-valued vector of dimension 6**.

The datasets contained sequences of various lengths, from two-image sequences with one conditioning vector to sequences of 26 frames.

Dataset Overview - No Objects

The original dataset contains 5,431 distinct sample frames, organized into a total of 1,405 sequences, with variable lengths ranging from 2 to 26 frames.

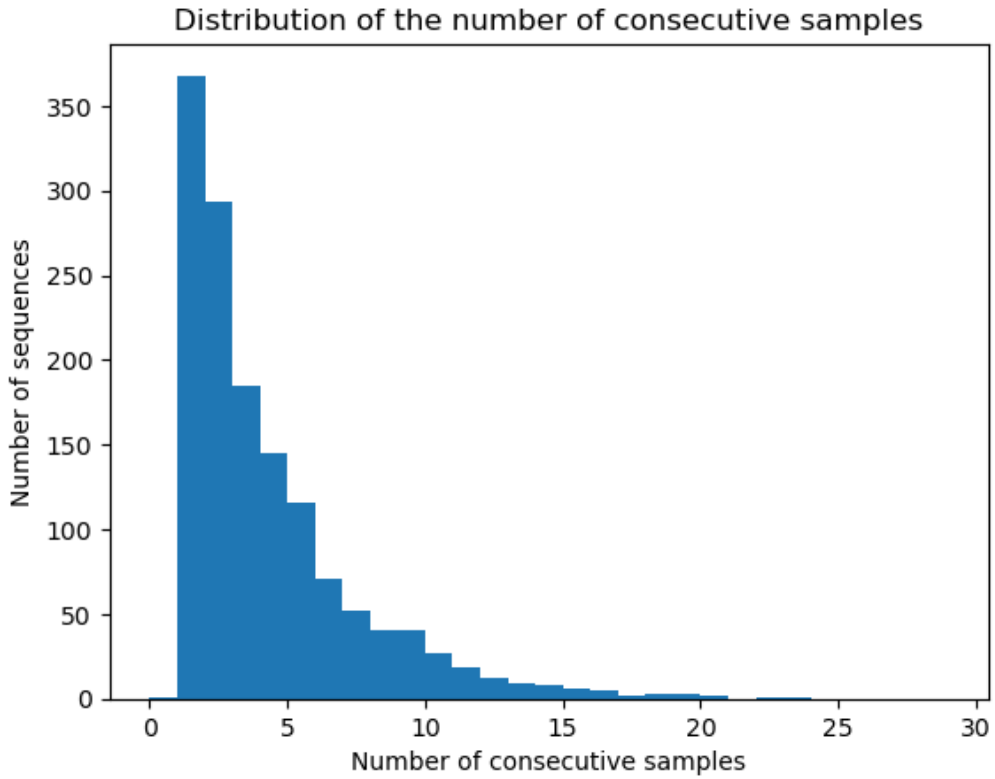

Figure 1: Distribution of Sequence Lengths in the Original Dataset

Original Dataset Overview - Objects

The original dataset contains 5413 distinct sample frames, organized into a total of 1,412 sequences, with variable lengths ranging from 2 to 23 frames.

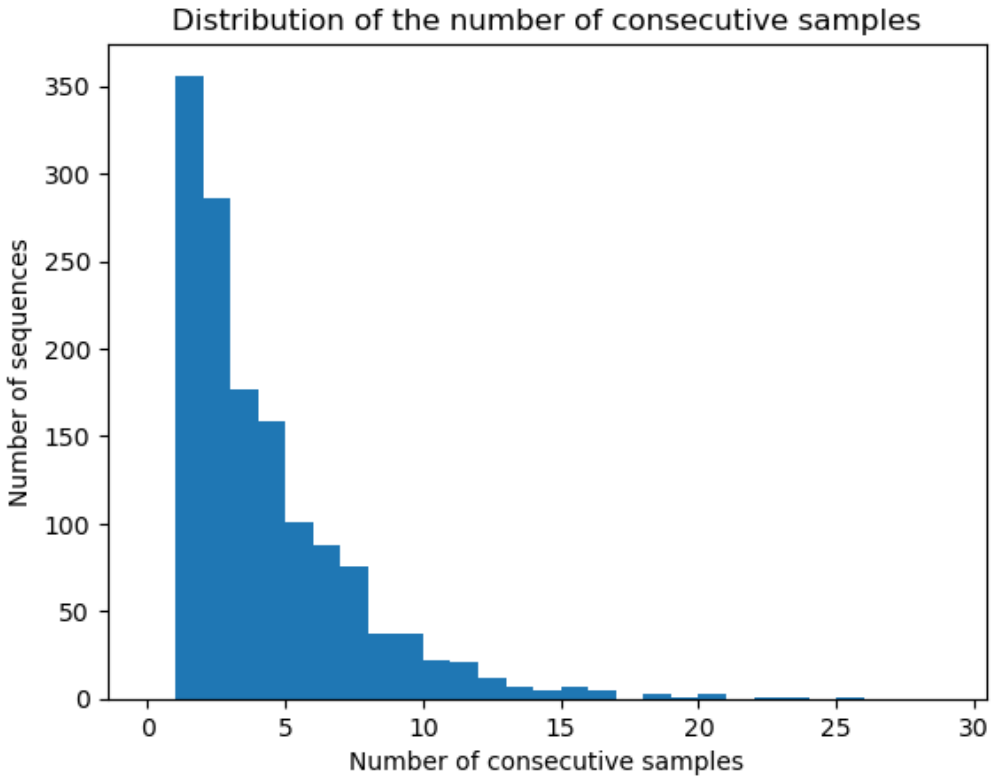

Figure 2: Distribution of Sequence Lengths in the Original Dataset

Data Preparation

We focused on sequences of **6 frames**, each **with 5 conditioning vectors**. Longer sequences were trimmed so there was no overlap, and sequences shorter than 6 frames were not used.

The choice of **6 frames** was a **trade-off** between **not losing too much data** and having **sequences long enough** for the model to **capture movement patterns**.

Final Datasets

The images and conditioning vectors were saved as **TensorFlow tensors**, with images stored in their RGB representation.

The final datasets consist of:

- **718 sequences** (6 frames with 5 conditioning vectors) for the dataset **without objects**.
- **703 sequences** (6 frames with 5 conditioning vectors) for the dataset containing **objects**.

Final Datasets

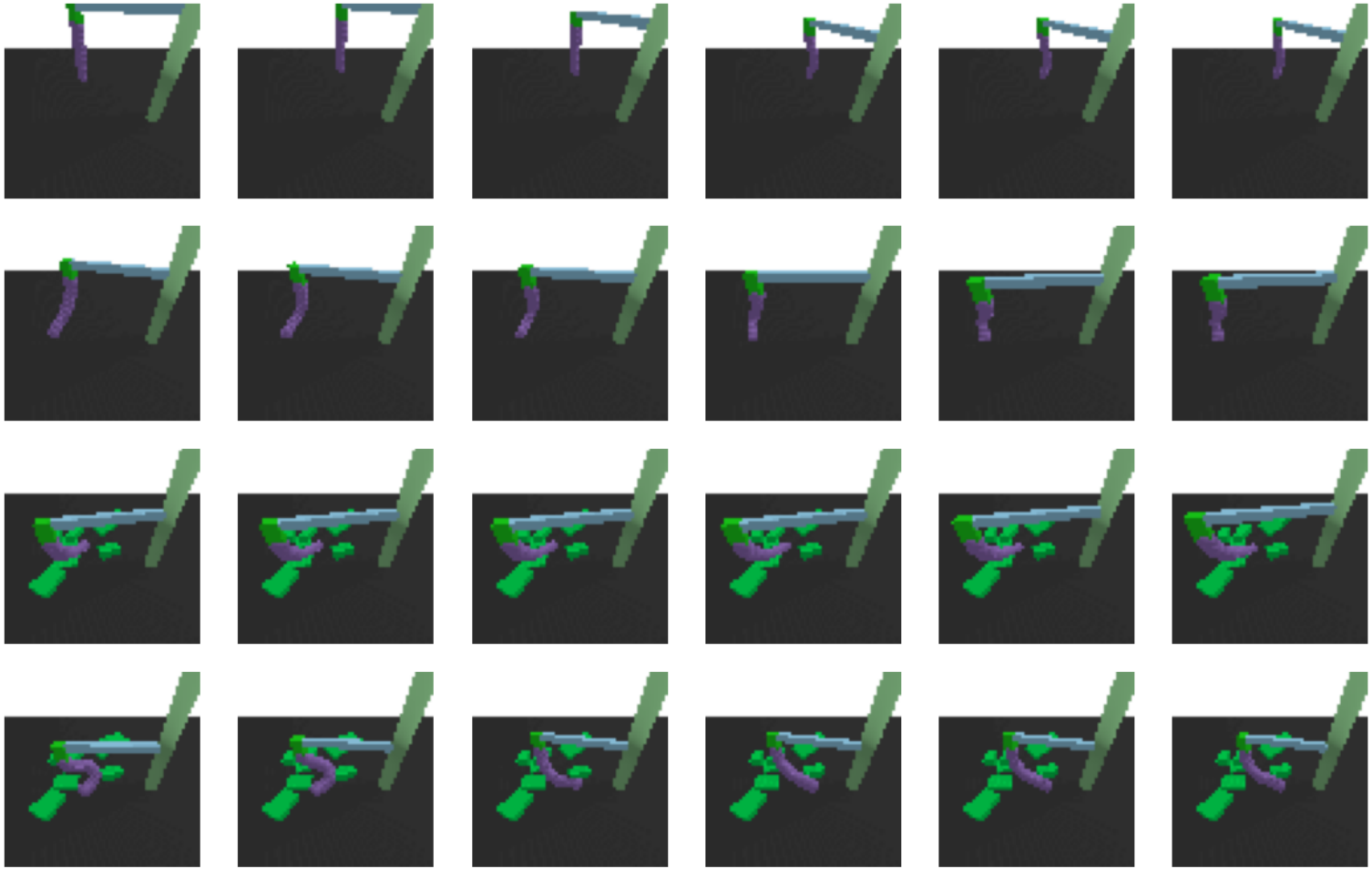

Figure 3: Dataset samples

Data Compression

Data Compression with Latent Vectors

For data compression, we used the model presented in *Face Aging With Conditional Generative Adversarial Networks* (Antipov et al., 2017).

This model enables the generation of a **latent representation** of the data **independently of the conditioning.**

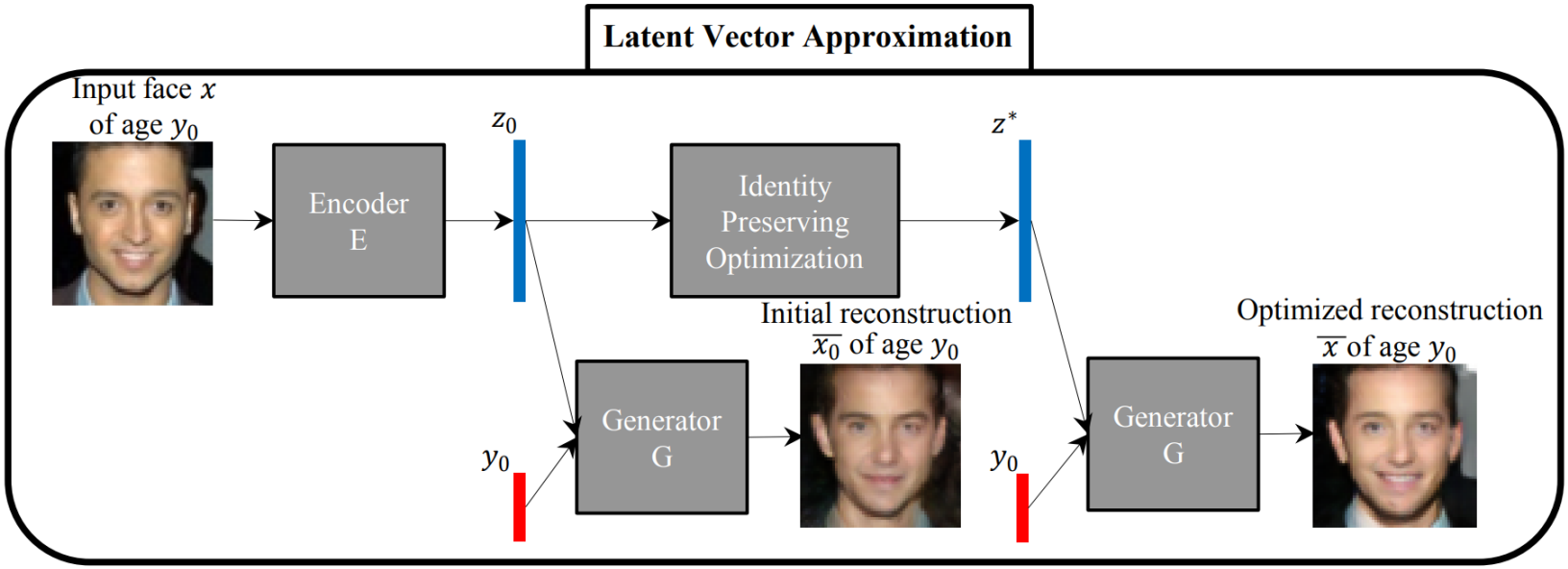

Figure 4: Latent Vector Approximation Process

Data Compression with Latent Vectors

Model Components:

- **Variational Autoencoder (VAE):** Learns the distribution of the latent space to encode the input image.
- **First Generator:** Generates an image from the latent vector produced by the VAE.
- **Identity Preserving VAE (IPVAE):** Adjusts the latent vector to preserve the identity of the input image while modifying it.
- **Second Generator:** Produces an optimized image from the latent vector obtained by the IPVAE.

Data Compression with Latent Vectors

Training Process:

1. **Pre-training:** The VAE, the generators, and the IPVAE are trained separately to optimize their specific tasks.
2. **Fine-tuning:** The entire model is assembled using the pre-trained components and fine-tuned to improve overall performance.

Data Compression with Latent Vectors

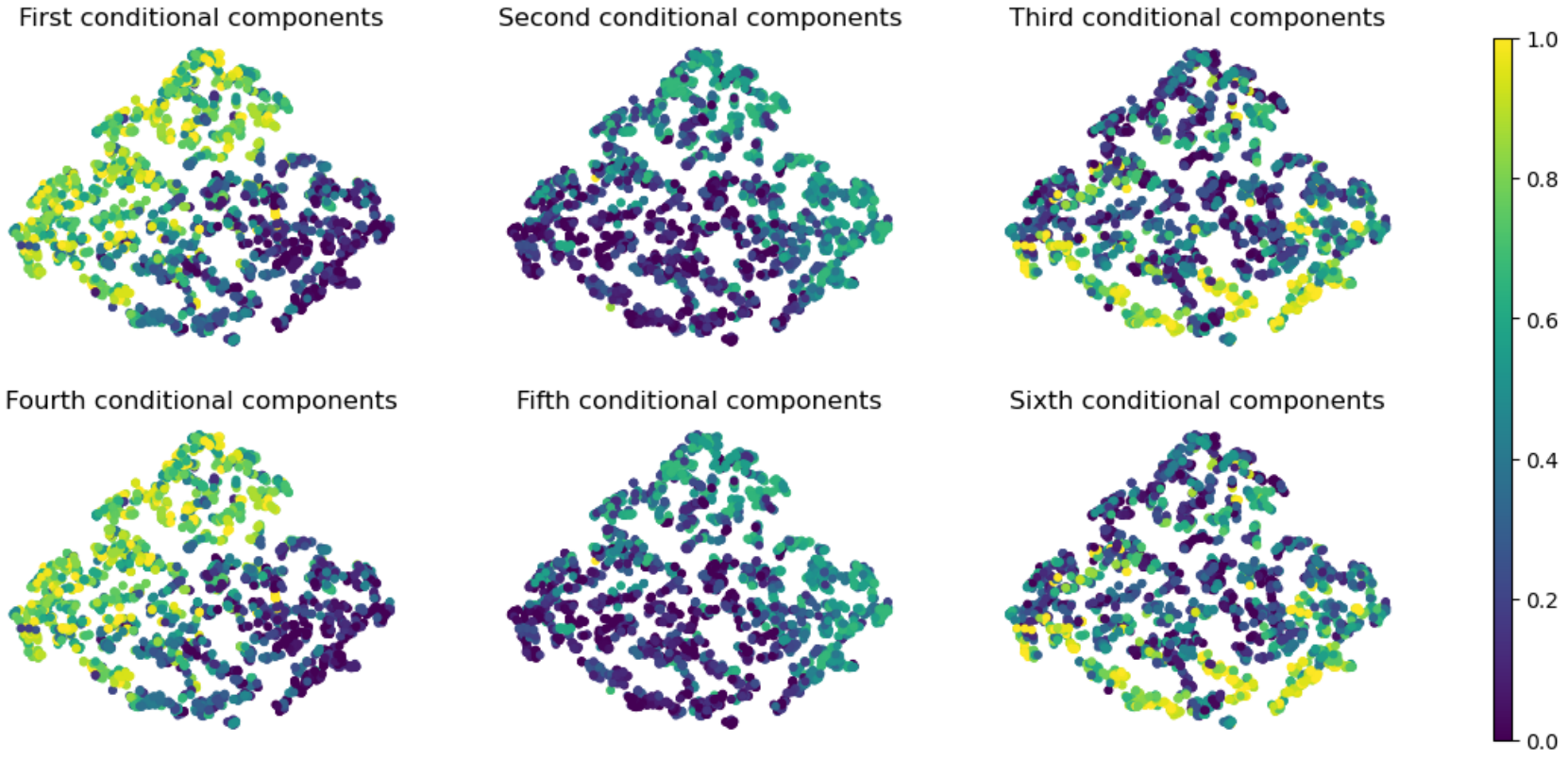

Figure 5: TSNE (T-distributed Stochastic Neighbor Embedding) of the encoded images with conditional components

Data Compression with Latent Vectors

The table below shows statistics for the latent representation of data from datasets with and without objects.

	No objects	**With objects**
Mean	8.429336	5.731796
Std	24.844608	23.135963
Min	-9.623224	-7.175397
Max	396.770782	463.580902

Architecture

WGAN Architecture Choice

Issue with Classic GANs:

- Unstable training due to vanishing gradients.
- Difficulty in generating high-quality images consistently.

Solution: WGAN Architecture

- **Wasserstein GAN (WGAN)** improves stability by using the Wasserstein distance as a measure of the difference between the generated and real data distributions.
- This distance provides smoother gradients, helping the model converge better than classic GANs.

$$
\begin{aligned}
G^* &= \arg\min_G W(\mu, \mu_G) \\
&= \arg\min_G \sup_{\|D\|_L \leq 1} \left(\mathbb{E}_{x\sim\mu}[D(x)] - \mathbb{E}_{x\sim\mu_G}[D(x)]\right)
\end{aligned}
$$

WGAN-GP Training for Stability

Training Approach:

- The critic is trained 5 more times than the generator to provide more accurate feedback.
- **WGAN-GP (Gradient Penalty)** is used to enforce the Lipschitz constraint, ensuring that the critic function is smooth.
- Clipping of critic weights or applying gradient penalty to ensure convergence and stability in training.

Outcome:

- This approach **stabilizes the training** and results in more realistic image generation, making it ideal for complex tasks like video prediction in robot control.

Generator Architecture

Purpose: The generator creates realistic future frames of a video based on the context of previous frames and external conditions.

Input:

- A sequence of 6 real images.
- Latent vectors for each frame.
- Conditional information.

Output:

- A sequence of 6 generated frames predicting the next steps in the video.

Generator Training

Architecture Overview:

- **Context Generation:** Combines latent vectors and conditional information through dense layers to create a context vector.
- **Frame Prediction:** Uses the context vector and previous frames to generate the next frame in the sequence via transposed convolutional layers.

Training Process:

- The context vector serves as input to the CNN, which generates each subsequent frame in the sequence.
- Each frame is processed by a series of transposed convolutional layers that use the context to predict the next frame.

Generator Training

Teacher Forcing: Ground truth frames are fed as input for the first 5 generated frames, while generated frames are used for subsequent ones.

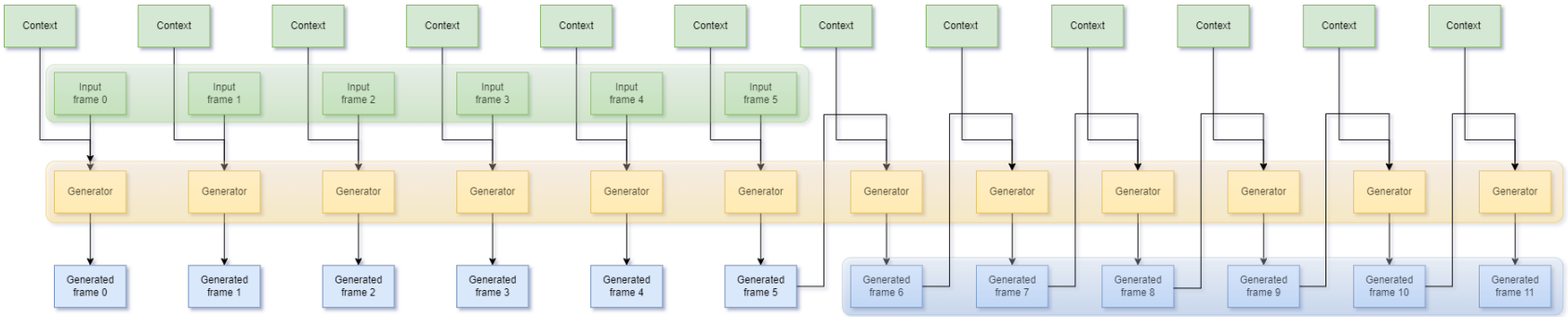

Figure 6: CNN Generator Training Flow

Discriminator Architecture

Purpose: Classifies input sequences between generated by the **generator** and **real images**.

Input:

- Takes input sequences and processes them through convolutional layers to extract meaningful features.

Output:

- The output layer consists of a single unit that predicts whether the input image is real or fake.
- Utilizes a sigmoid activation function to output a probability value between 0 and 1.

Discriminator Architecture

Training Strategy:

- **Feature Extraction:** Series of convolutional layers learn patterns from images, followed by fully connected layers for classification.
- Leverages dropout and **LeakyReLU** for better generalization and stable gradient flow during training.
- Utilizes a sigmoid activation function to output a probability value between 0 and 1.

Loss Function Definition

Discriminator Loss:

- **Wasserstein Loss:** Maximize the ability to discriminate between real and fake data.
- **Gradient Penalty:** Enforces Lipschitz constraint to stabilize training.

$$L_D = \underbrace{-\mathbb{E}_{x\sim\text{real}}[D(x)] + \mathbb{E}_{z\sim\text{noise}}[D(G(z))]}_{\text{Wasserstein Loss}} + \underbrace{\lambda\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]}_{\text{Gradient Penalty}}$$

Loss Function Definition

Generator Loss:

- **Wasserstein Loss:** Minimizes the critic's output on generated (fake) data.
- **Mean Squared Error (MSE):** Reduces the difference between real images and consecutive generated images.
- **Optical Flow Loss:** Ensures temporal consistency by comparing motion in real and fake images.
- **Color Loss:** Maintains color similarity between real and generated images.

Loss Function Definition

$$L_G = \underbrace{\mathbb{E}_{x\sim\text{real}}[D(x)] - \mathbb{E}_{z\sim\text{noise}}[D(G(z))]}_{\text{Wasserstein Loss}}$$
$$+ \underbrace{\alpha \cdot \frac{1}{n}\sum_{i=1}^{n}(x_i - G(z_i))^2}_{\text{MSE}} + \underbrace{\beta \cdot \frac{1}{n}\sum_{i=1}^{n}\text{Flow}(x_i, G(z_i))}_{\text{Optical Flow Loss}}$$
$$+ \underbrace{\gamma \cdot \frac{1}{n}\sum_{i=1}^{n}\sum_{c\in\{r,g,b\}}\|x_{ic} - G(z)_{ic}\|}_{\text{Color Loss}}$$

Experiments Results

Model Selection

The best model was selected through a series of preliminary tests in which, in addition to loss values, the generated sequences in the output were observed.

The **model selection** was made qualitatively by **assessing the quality of the results** obtained.

Model Selection

Due to the **computational cost**, and given the **already satisfactory results**, an **exhaustive grid search** was **not performed**. There is still room for future investigations to achieve better performance.

For the same reasons, the same set of hyperparameters was used to train both the model with the dataset containing objects in the background and the one without.

Model Selection

Model selection considered not only the model's hyperparameters and layers but also the generator's architecture.

Various experiments were conducted for the generator, initially using an LSTM, which did not yield good results.

Selected Model Hyperparameters

Parameter	Value
Discriminator Learning Rate	0.01
Generator Learning Rate	0.005
Patience	3
Number of Critic Updates per Generator	5
Image Dimension	[3, 64, 64]
Latent Dimension	1280
Conditions Dimension	6
Input Sequence Length	6
Output Sequence Length	6

Selected Model Hyperparameters

Generator Parameter	**Value**
Number of Filters	[4, 8, 8, 16]
Kernel Size	[5, 3, 3, 1]
Stride	[1, 1, 1, 1]
Padding	zero padding to preserve dimension
Hidden Dimensions	[128, 128]
Dense Layers Dropout	0.3
Activation Function	Leaky Relu
LeakyReLU Coefficient	0.2
Weight Initialization Mean	train set mean
Weight Initialization Std Deviation	train set std dev
Bias Initialization	0
Generator Loss Function	
Wasserstein Loss Weight	0.2
Optical Flow Loss Weight	0.15
MSE Weight Input-Generated Frames	0.6
Color Loss Weight	0.05

Selected Model Hyperparameters

Discriminator Parameter	**Value**
Number of Filters	[4, 4]
Kernel Size	[3, 3]
Stride	[1, 2]
Padding	zero padding to preserve dimension
Hidden Dimensions	[512, 256, 128, 64]
Dense Layers Dropout	0.4
Activation Function	Relu
Weight Initialization Mean	0
Weight Initialization Std Deviation	0.1
Bias Initialization	0
Discriminator Loss Function	
Gradient Penalty Weight	0.02
Clip Value	0.01

Learning Curves - Dataset without Objects

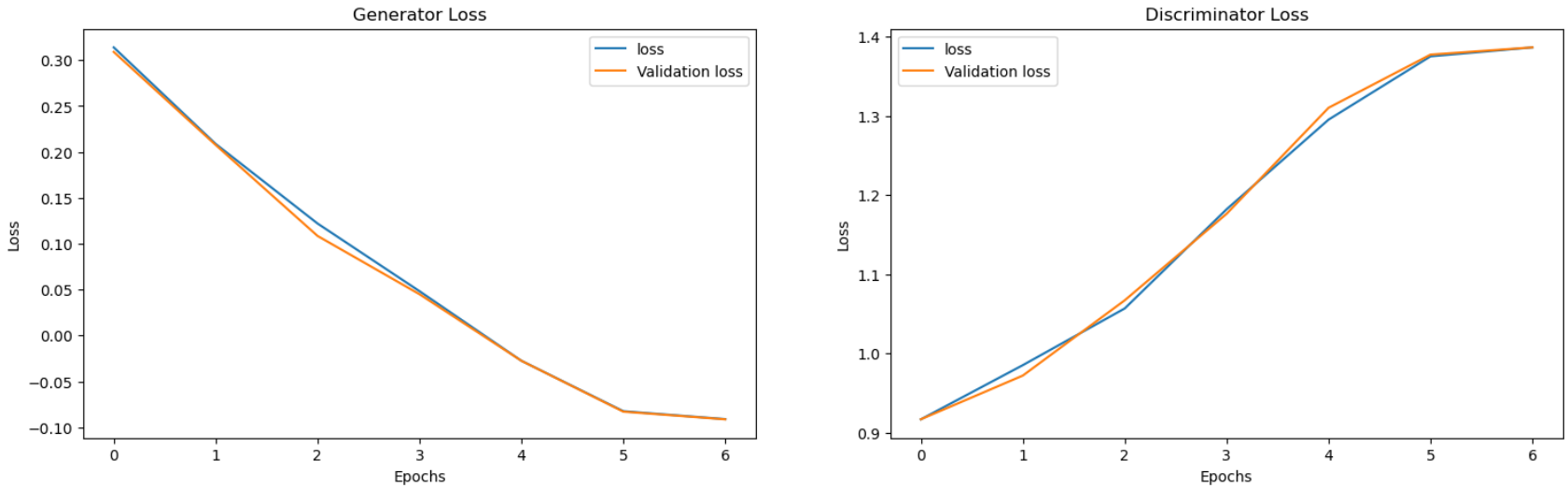

Figure 7: Learning curves for the generator and discriminator on train and validation sets for the dataset without objects. The curves show model convergence during training and accuracy on unseen data.

Learning Curves - Dataset without Objects

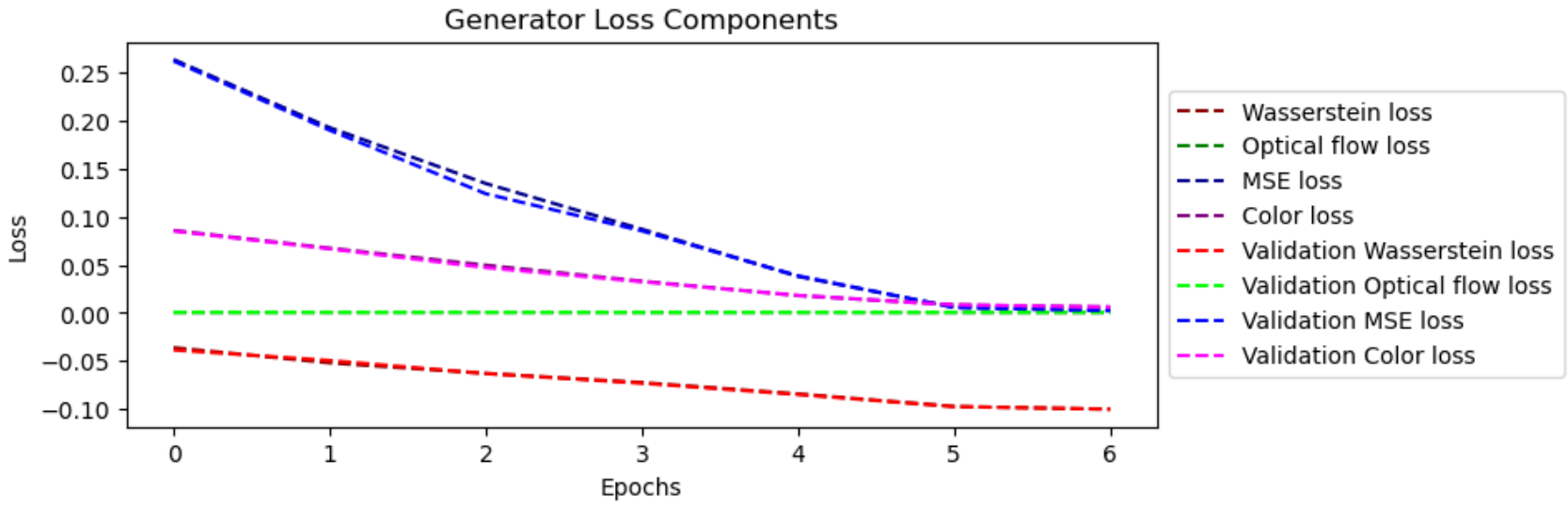

Figure 8: Components of the generator's learning curve on the training set, showing the variation of different losses (Wasserstein, MSE, Optical Flow, and Color Loss) throughout training.

Learning Curves - Dataset with Objects

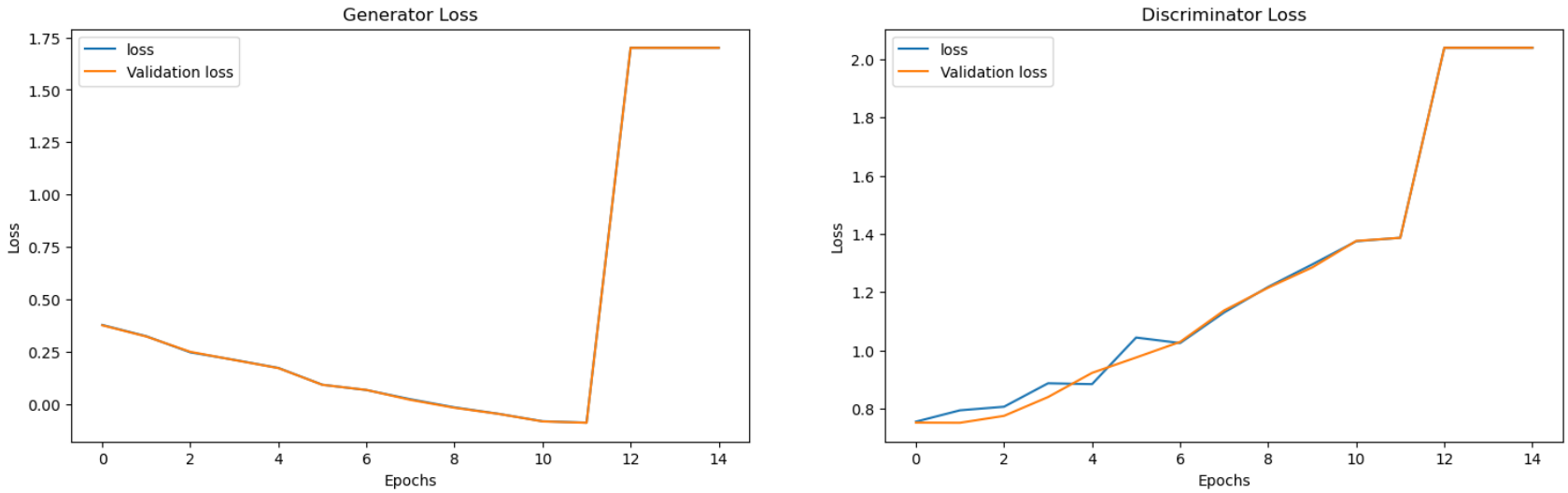

Figure 9: Learning curves for the generator and discriminator on train and validation sets for the dataset with objects. The curves indicate that training was stopped because the model diverged towards infinity.

Learning Curves - Dataset with Objects

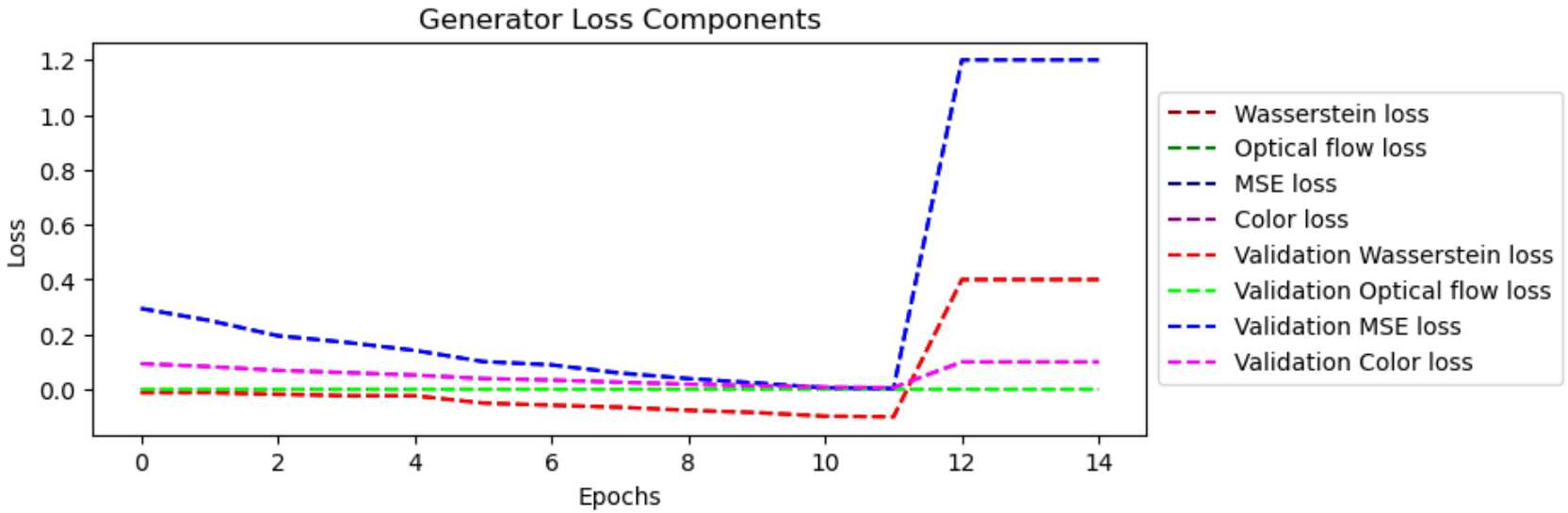

Figure 10: Components of the generator's learning curve on the training set, showing the variation of different losses (Wasserstein, MSE, Optical Flow, and Color Loss) throughout training.

Validation Examples - Dataset without Objects

Figure 11: Examples from the validation set across various epochs from the first to the seventh (algorithm convergence). The images display the inputs and the images generated by the model at each training epoch.

Validation Examples - Dataset with Objects

Figure 12: Examples from the validation of images generated by the model at each training epoch.

Validation Examples - Dataset with Objects

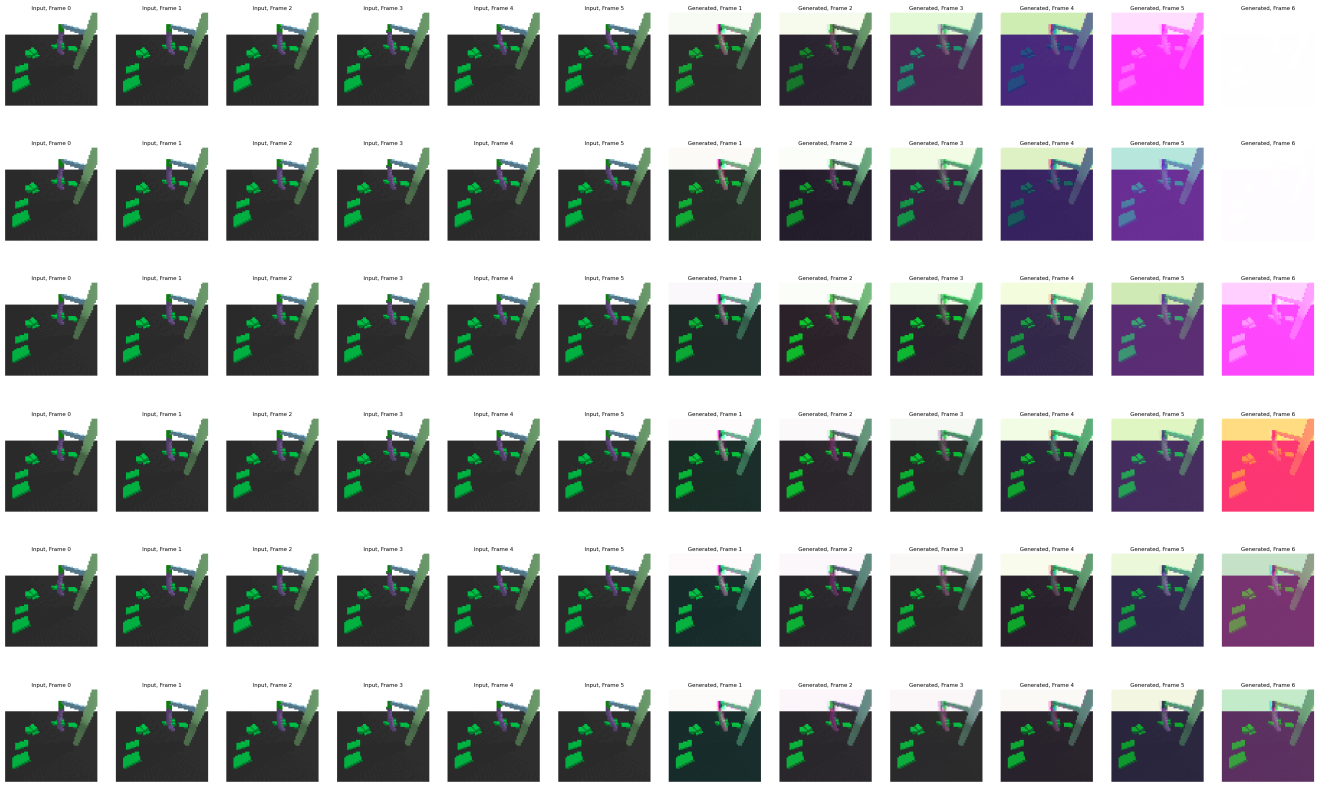

Figure 13: Examples from the validation of images generated by the model at each training epoch.

Results - Dataset without Objects

Figure 14: Results on the test set.

Results - Dataset with Objects

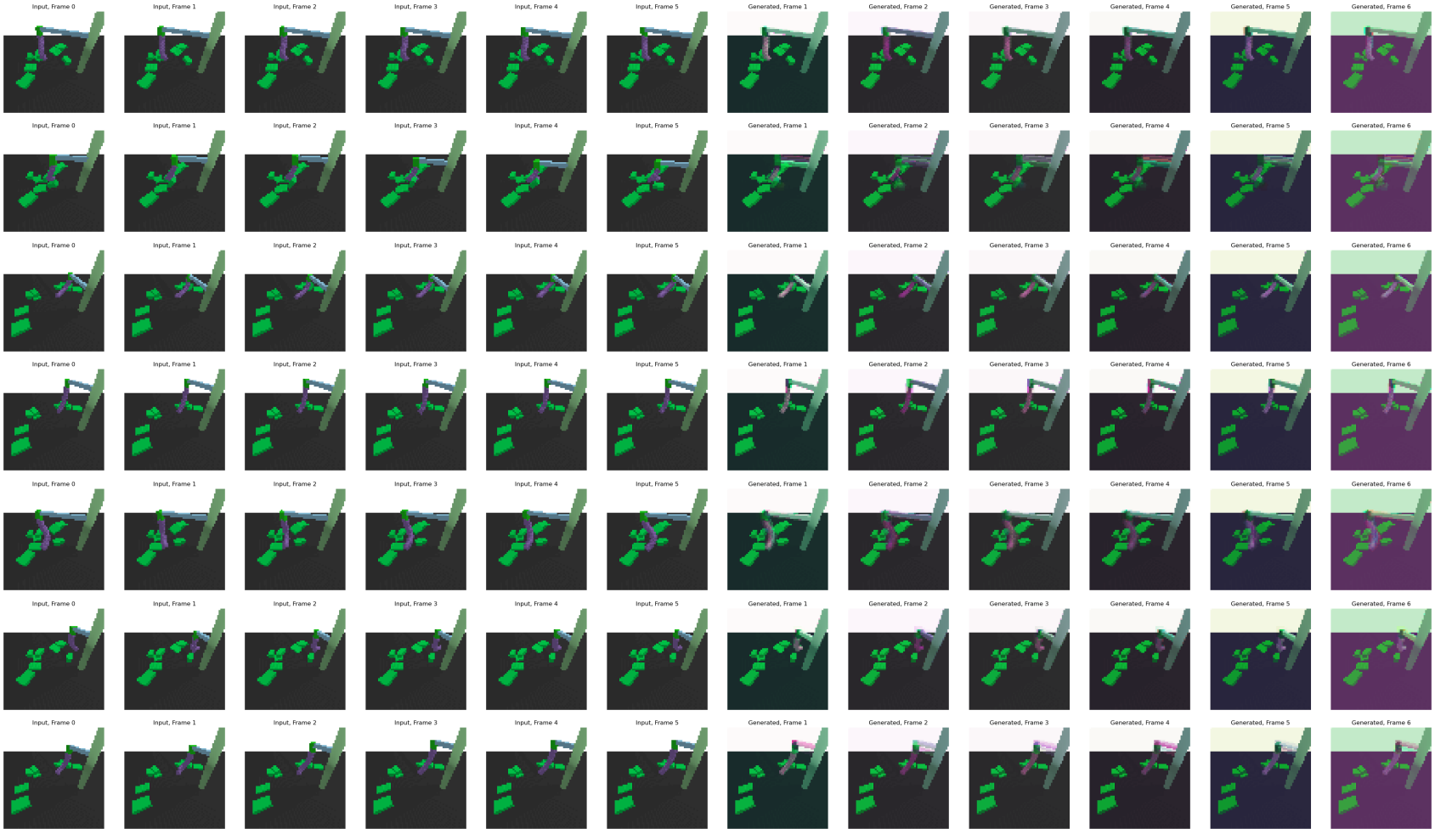

Figure 15: Results on the test set.

Recursive Results

We also experimented with **using the model** in a **recursive manner** for video prediction tasks.

By feeding the model's **previous predictions as inputs** for subsequent frames, we can **generate extended sequences** of future frames, allowing the model to iteratively build upon its own outputs.

This approach provides insight into the model's ability to maintain consistency and accuracy over extended time steps in the generated video sequence, highlighting how the noise present in the generated frames propagates to subsequent predictions.

Recursive Results

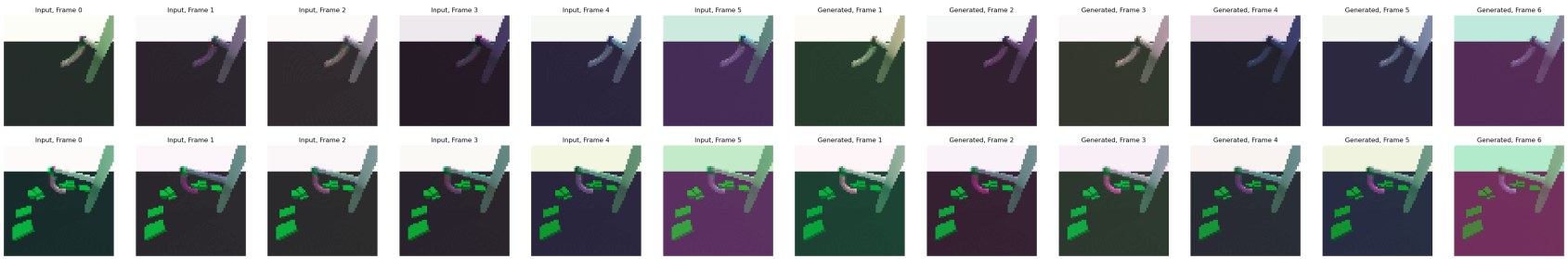

Figure 16: The figures illustrate the frames predicted by the model in a recursive manner, where previous predictions are used as input for subsequent frames. It is evident that noise propagates through the predictions, indicating opportunities for model improvements to enable the generation of high-quality images in a recursive fashion.

Limitations and Future Works

Limitations

- The model's **computational expense** and slow processing speed have made it challenging to conduct a thorough grid search and fine-tuning of the parameters; however, a **more accurate hyperparameter search** could lead to **significant improvements** in the model's performance.
- There is a challenge in generating longer sequences, as this tends to result in a loss of accuracy. **Longer input frames** are necessary to **improve performance**.

Future Works

- **Explore** further combinations of hyperparameters for improved frame prediction.
- Apply **data augmentation**.
- **Integrate the model's predictions** into a loop for real-time decision-making.
- Predict **conditioning variables** for each frame as well.
- Adjust the **discriminator's input data** to include a **greater proportion** of **real samples** over generated ones, and modify the sigmoid threshold accordingly.
- Different **conditioning strategies**, such as using sensor data.

Conclusion

Conclusions

- The proposed model shows promising results in sequence generation, effectively capturing temporal dependencies in the dataset.
- Qualitative analysis suggests that frame prediction **quality is satisfactory** for **sequences of moderate length**, though accuracy decreases with longer predictions.
- Due to computational limitations, the current hyperparameter setup was not extensively optimized, indicating **potential for performance improvements**.

Supplementary Material

Original Dataset - No Objects

Length	Frequency	Cumulative Frequency	#generated sequences
1	356	356	
2	286	642	
3	177	819	
4	158	977	
5	101	1078	
6	88	1166	1
7	76	1242	1
8	37	1279	1
9	37	1316	1
10	22	1338	1
11	21	1359	1
12	12	1371	2
13	7	1378	2
14	5	1383	2
15	7	1390	2
16	5	1395	2
18	3	1398	3
19	1	1399	3
20	3	1402	3
22	1	1403	3
23	1	1404	3
25	1	1405	4

The table above shows Sequence Lengths, Frequency, Cumulative Frequency, and the number of 6-frame sequences generated from each sequence of that length. Only sequences with a length of 6 or greater were used for generation.

Original Dataset - Objects

Length	Frequency	Cumulative Frequency	# Generated Sequences
1	368	369	
2	294	663	
3	185	848	
4	145	993	
5	116	1109	
6	71	1180	1
7	52	1232	1
8	41	1273	1
9	41	1314	1
10	27	1341	1
11	19	1360	1
12	12	1372	2
13	9	1381	2
14	8	1389	2
15	6	1395	2
16	5	1400	2
17	2	1402	2
18	3	1405	3
19	3	1408	3
20	2	1410	3
22	1	1411	3
23	1	1412	3

The table above shows Sequence Lengths, Frequency, Cumulative Frequency, and the number of 6-frame sequences generated from each sequence of that length. Only sequences with a length of 6 or greater were used for generation.

Loss Function Definition

Discriminator Loss:

- **Binary Cross-Entropy:**

$$L_D = -\mathbb{E}_{x\sim\text{data}}[\log D(x)] - \mathbb{E}_{z\sim\text{noise}}[\log(1 - D(G(z)))]$$

- **Gradient Penalty:**

$$L_{\text{GP}} = \lambda\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]$$

Loss Function Definition

Generator Loss:

- **Wasserstein Loss:**

$$L_G = -\mathbb{E}_{z \sim \text{noise}}[D(G(z))]$$

- **Mean Squared Error (MSE):**

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (x_i - G(z_i))^2$$

- **Optical Flow Loss:**

$$L_{\text{OF}} = \frac{1}{n} \sum_{i=1}^{n} \|\text{Flow}(x_i) - \text{Flow}(G(z_i))\|_2$$

- **Color Loss:**

$$L_{\text{color}} = \sum_{c \in \{r,g,b\}} \|x_c - G(z)_c\|_1$$

Data Distribuition - No Objects

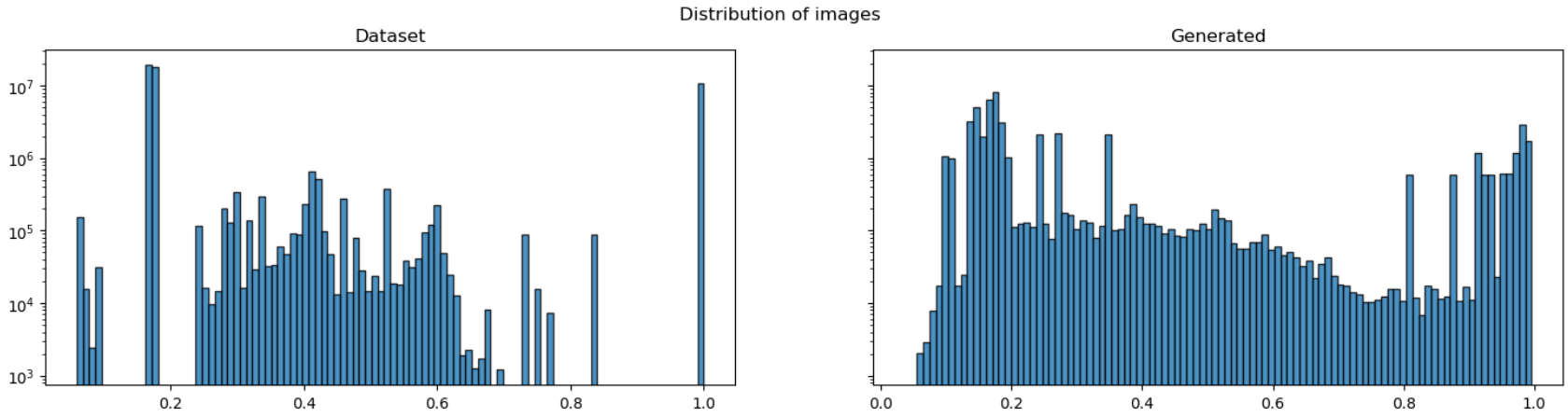

Figure 17: Distribution of densities for the real dataset (left) and the generated dataset (right). The graph illustrates the comparison between the empirical distribution of the actual data and the synthetic data produced by the model, highlighting the effectiveness of the generation process.

Data Distribuition - No Objects

Statistical comparison between the original dataset and the generated dataset.

Statistic	Dataset	Generated
Mean	0.364613	0.360628
Std Dev	0.330093	0.312250
Median	0.180420	0.180013
Min	0.058838	0.036363
Max	1.000000	0.995566
25% Quantile	0.168579	0.162237
75% Quantile	0.415771	0.411832

Data Distribuition - Objects

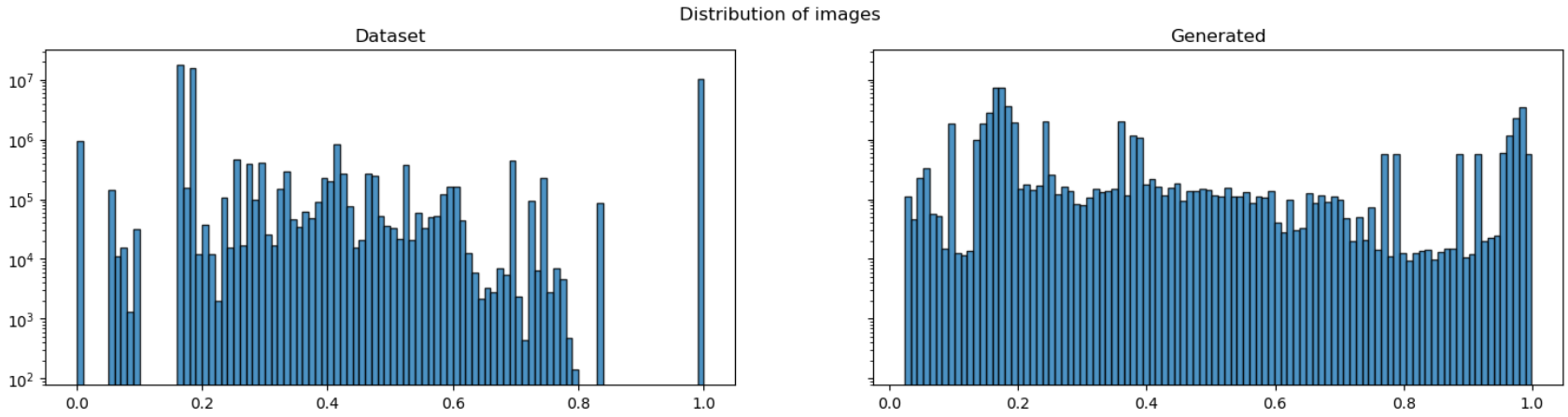

Figure 18: Distribution of densities for the real dataset (left) and the generated dataset (right). The graph illustrates the comparison between the empirical distribution of the actual data and the synthetic data produced by the model, highlighting the effectiveness of the generation process.

Data Distribuition - Objects

Statistical comparison between the original dataset and the generated dataset.

Metric	Dataset	Generated
Mean	0.371405	0.374729
Std Dev	0.333534	0.311762
Median	0.180420	0.182286
Min	0.000000	0.022939
Max	1.000000	0.999084
25% Quantile	0.168579	0.166446
75% Quantile	0.462646	0.491995