VQ-VAE Transformers Transfer Learning for Video Interpolation

Tanay Bennur (tbennur), Cathleen Chen (cathleec),
Nithya Kemp (nrkemp)

April 2024

1 Introduction

With the advent of image generation follows an extension into videos. Video generation is actively being pioneered and refined, particularly through the use of diffusion models. We sought to take a different approach in this space. Our project performs a video interpolation task. Provided beginning and end frames or clips of a video, we provide context-appropriate material for the middle of a video. A brief literature survey indicated that this task has been done with diffusion models, similar to general video generation. We aim to do this by training a combination of a VQ-VAE and a Transformer model with ample context (31 frames) and attempting to predict missing frames in a lower context (2 or 6 frame) setting. There is a precedent for using this combination in video generation, but not for interpolation. Using a modified version of the Vimeo90k dataset, our model is able to cohesively interpolate video for provided inputs, approaching though not currently meeting industry levels, given time and compute constraints.

2 Dataset and Task

Our model's task was to interpolate missing central frames from a larger sequence of frames. We based our dataset on the Vimeo90k dataset, a common video interpolation and generation dataset composed of ninety thousand videos from the Vimeo website. We explored long-context interpolation, where we trained the model to perform interpolation with 31 frame sequences. This allowed us to leverage Transformer models' ability to capture long distance dependencies but required us to create our own dataset.

We used webscraping and image processing techniques to create a version of this dataset with 31 frames of context - our model is trained to predict all even indexed frames from a 31-frame sequence. We tested this model's transfer-ability by predicting missing frames on standardized Vimeo90k septuplet datasets. Our ablations focused on finding the ideal transfer method.

We also periodically evaluated our model (and its variants) with 2 metrics, PSNR and SSIM. PSNR, or peak signal-to-noise ratio, is an image reconstruction quality metric largely based on MSE that approximates human perception. SSIM, or the structural similarity index measure, calculates the reconstruction error between various structural features (extracted by kernels) of each image. The combination of these metrics should give a holistic and objective insight into our model's interpolation quality.

3 Related Work

Video Generation Prior to experimentation, we surveyed the general video generation space. Text-based prompts are the norm. The most common form of video generation is using diffusion models. A 2022 paper detailing "Imagen Video" is a prime example of a video generation model utilizing text-based prompts [2]. The paper details the use of a cascade of diffusion models. In general, this form of video generation is found to generate high quality videos along with providing the user with a higher amount of control. Though the standard, diffusion models are not the only method explored for generation. In the same year, a "Make-A-Video" approach was published that utilizes existing text-to-image models to accelerate the training of a text-to-video model [9]. It utilizes existing representations and doesn't require labeled data. This idea of transferring existing models from other applications extends beyond the text-to-image space. In a paper from 2021, VQ-VAEs were utilized for the video generation task [10]. The power of the VQ-VAE allowed the model to compress videos to discrete latent representations, reducing dimensionality, which aids significantly in video generation, given intense compute and time constraints. This work provided higher-resolution videos than any work before its publication. VideoGPT is a model released shortly after combining the power of the latent representations provided by VQ-VAEs with Transformer models [11]. With a simple architecture, the model was able to compete with other models at the time of development, demonstrating room for this model to outperform given larger model architecture.

Video interpolation Video interpolation is a slightly different task from text-to-video generation, but the basis remains very similar, and hence video generation methods can easily be applied here. However, this field of research is more recent. Last year, Danier et al.'s LDMVFI paper handles the task of interpolation from a generative perspective, and does so utilizing diffusion models [1]. The paper demonstrated better results than state-of-the-art non-generative models at that time. More recently, a paper released this past month details using diffusion models for the video interpolation task [4]. Provided a start and end frame, a generative model can be applied to generate high-quality models. The paper is novel not only in its reapplication but also in the added modifications that mean the model requires fewer parameters, improving scalability.

But diffusion models aren't the sole approach that have been explored, just

the most recent. In 2021, this task was handled by using transformer models [5]. The idea is that the Transformer model can capture long-range dependencies that can be useful in a video interpolation task. The paper also features a modified and novel attention mechanism that aids the video interpolation task. The result is a model comparable to state-of-the-art models.

Despite these findings, no one has sought the combination of VQ-VAEs and Transformer models for the interpolation task. We believe there is potential for the VQ-VAE and Transformer combination to improve the video interpolation task, given the simplicity of the model compared to others in the field, the power of the latent representations provided by VQ-VAEs, and the general potential of the Transformer architecture to lend itself well for interpolation tasks. Hence, we seek to build off the work in the VideoGPT paper for the interpolation task and assess the effectiveness relative to other methods in the field.

Transfer Learning We can extend this task further by applying transfer learning. Pan et al. demonstrate the power of transfer learning in reducing parameters [7], and a paper from Mahum et al. provides a framework for applying transfer learning for video generation tasks [6].

Measure of Image Quality We examined several papers regarding evaluating image quality, including the following comparative study from Sara et al., assessing different image quality measures [8]. We ultimately landed on utilizing the Structured Similarity Indexing Method (SSIM) and the Peak Signal to Noise Ratio (PSNR) from this paper, due to the mix of interpretability and detail that this combination provides.

4 Methods

Our model approaches video generation using the architecture of a VQ-VAE and a transformer. The VQ-VAE compresses the frames of the video into a lower-dimensional latent representation as input to the transformer, and then decodes the generated latent representations from the output of the transformer back into video frames,

Our key contributions are the extension of the VQ-VAEs and Transformer architecture to an interpolation task (the combination of these methods was previously only used for a prediction task) and the introduction of the transfer learning aspect (changing context length).

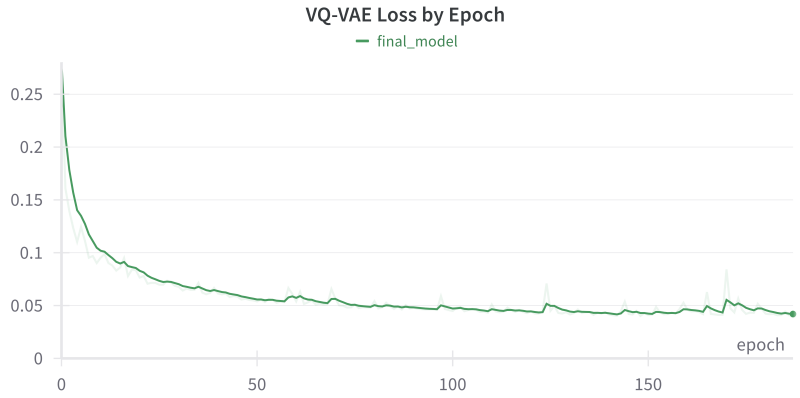
We begin with sequences of video frames and quantize them using a VQ-VAE. We apply a mask to remove the frames being predicted and pass the quantized frames through a video transformer and generate quantized predictions of the most likely masked frames. Finally, we reconstruct the masked frames from our quantizations.

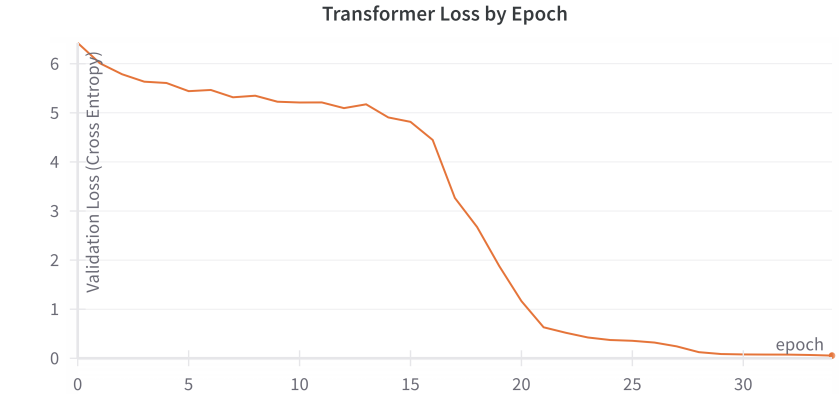
In a transfer learning context (i.e. predicting fewer frames) we extend the context sequence by padding non-existent frames or repeating pre-existing frames for a more accurate prediction method. The bulk of our experimentation will lie here in testing the effectiveness of different frame ratios.

The methods requiring adaptation for the novel task would thus be the video

data preprocessing step due to the use of another dataset, the attention masking step, the frame conditioning step, and the generation step. We will also create a method to evaluate the final results of our interpolated frames through metrics such as PSNR and SSIM.

5 Experiments





The graphs above show the loss for our models over time. We were intrigued by the transformer loss - the sigmoidal nature wasn't something we had seen before and could merit further investigation.

Our original approach was trained on predicting 31 frame inputs, with the even-numbered frames masked out for prediction. We tested our framework on Vimeo90k's septuplet dataset with 64x64 resolution and 7 total frames and compared it to the performance of pre-existing transformer and diffusion models. We experimented with various

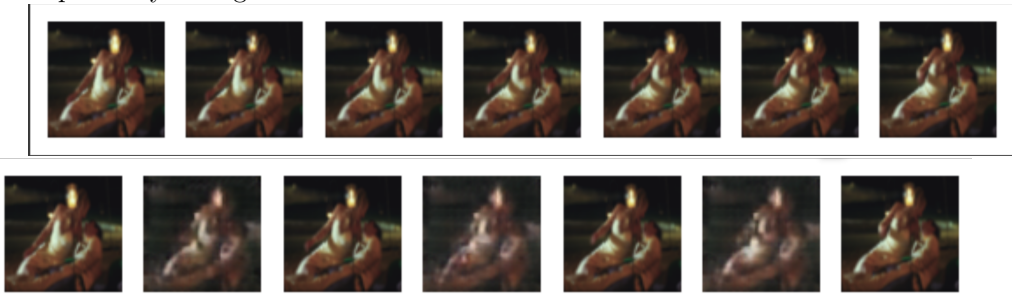
Quantitative Comparisons with Vimeo90K Benchmark			
Method	Parameters (M)	PSNR	SSIM
SuperSloMo	39.6	32.9	0.957
DAIN	24.0	33.35	0.945
SepConv	21.6	33.60	0.944
BMBC	11.0	34.76	0.965
CAIN	42.8	34.83	0.970
AdaCoF	21.8	35.40	0.971
QVI	29.2	35.15	0.971
SoftSplat	7.7	35.76	0.972
FLAVR	42.4	36.30	0.975
VFIT-S	7.5	36.48	0.976
VFIT-S	29.0	36.96	0.978
Our Best	33.3	20.05	0.731

The above table contains a comparison on quantitative features such as the amount of parameters and runtime, as well as evaluation metrics such as PSNR and SSIM on different models taken from [3]. For the intermediate frames, the metric values will be the average of the predicted frames. Unfortunately, our model does not seem to be comparable to any state-of-the-art techniques.

Our ablations revolved around testing different transfer functions - the table below shows a comparison between 3 different transfer function signatures. As seen below, transfer functions that incorporated many equal sized copies of several frames performed the best.

Quantitative Comparisons with Different Transfer Functions		
Signature (M)	PSNR	SSIM
10, 10, 10	20.05	0.731
10, 10, 5	19.70	0.730
2, 2, 1	19.59	0.718
3, 1, 2	19.57	0.712
3, 1, 1	19.44	0.715

We were not able to obtain this data for the baseline model due to current incompatibility with generation methods.



A sample output of our model is shown above. The first image demonstrates the original video frames, and the second demonstrates our model's interpolated frames.

6 Code Overview

There were 4 main components to our code.

Dataset and Data Collection

This code was written in the `VideoTransformerDataset.ipynb` file, and can be referred to in Figures 1-6 in the appendix. This code was primarily written from scratch. Because we wanted to use a Transformer architecture for an image generation task, we sought to capitalize on long-range dependencies which meant we needed more context than we could find in any existing dataset. In Figure 1, we wrote the setup and installation code. In Figures 2-5, we wrote the actual code necessary to fetch the videos, download them, and preprocess them, in addition to making our own custom Dataset class. Figure 6 just addresses saving the data.

VQ-VAE

This code was written in the `VQ-VAE.ipynb` file and can be referred to in Figures 7-10 in the appendix. Most of this code was borrowed directly from the VideoGPT github. The repository had a VQ-VAE implementation that worked well for our task. The only new things we wrote were installations, a custom dataset classes given that we created our own data, and code to actually train VQ-VAE.

Transformer

This code was written in the `VideoGPT.ipynb` file and can be referred to in Figures 11-14 in the appendix. Most of the code for the transformer was taken directly from the VideoGPT github, as the transformer they implemented worked well for our task. The code that we have written primarily consists of installations necessary to run the code, custom dataset classes given our custom datasets, as well as modified code in order to actually train the transformer.

Transfer Learning and Generating Results

All of this code was written in the `evaluation.ipynb` file. The code can be referred to in Figures 15-20 in the appendix. Most of the code in this file was written from scratch. We did make use of the VideoGPT class again which was borrowed from the VideoGPT paper [11]. The primary contributions here consisted of creating a dataset to handle septuplets, which will allow us to compare our code with other state-of-the-art models. We included functions

that allow us to obtain results and visualizations of those results. We also included a section for obtaining metrics.

7 Timeline

The below shows a table of time spent on different tasks during this project:

Task	Hours	Start Date	End Date
Literature Review	5	04/03	04/05
Proposal Writing	6	04/04	04/05
Code/Documentation Review	12	04/09	4/18
Dataset Scraping	30+	04/13	04/30
Baseline VQVAE Training	12	04/16	04/17
Baseline Transformer Training	8	04/17	04/18
Midway Summary Writing	4	4/15	4/18
Coding	25	4/20	5/1
VQVAE Training	15	4/30	5/1
Transformer Training	20	5/1	5/3
Poster Writing	3	5/1	5/3
Final Summary Writing	8	5/3	5/3

8 Research Log

We initially started out wanting to use Transformers for videos. This extended into using Transformers for video generation tasks. Given the large size of image datasets, and even more so for video datasets, we sought to reduce dimensionality through the use of a VQ-VAE. After investigating the space, we found VideoGPT which performed our initial idea. However, during our exploration, we realized how little work there was on the interpolation task. So we refocused our task to performing video interpolation through the use of a VQ-VAE and Transformer model. After initial experimentation and baseline model setup prior to the midway report, we realized that modifying code that was generating videos into interpolating videos presented its own set of challenges, and we chose to handle this, quite last minute, by pivoting our project entirely and incorporating a new aspect of our project: transfer learning. We sought to experiment with the selection of frames and scaling downwards to handle datasets that state-of-the-art machines also use. This last minute pivot ended up meaning that we had more work to do prior to our actual experimentation. This meant there was a delay in getting results. Hence, we wish we were able to get more ablations and experiments with the model architecture, but due to how much we pivoted from our original idea in such a short time frame, this was not as feasible. Instead, we experimented with different transfer functions

9 Conclusion

Our findings indicated that VQ-VAE and Transformer are somewhat applicable to video interpolation tasks. Additionally, correctly selecting transfer function will somewhat improve performance. However, other architectures (i.e. architectures without a VQ-VAE) might be more applicable to video interpolation tasks. Due to the limitations we reached due to time and monetary constraints, this area of research requires further exploration with more resources to get a better picture of the capabilities of this architecture compared to state-of-the-art models.

References

[1] Duolikun Danier, Fan Zhang, and David Bull. Ldmvfi: Video frame interpolation with latent diffusion models, 2023.

[2] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P. Kingma, Ben Poole, Mohammad Norouzi, David J. Fleet, and Tim Salimans. Imagen video: High definition video generation with diffusion models, 2022.

[3] Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. Real-time intermediate flow estimation for video frame interpolation, 2022.

[4] Siddhant Jain, Daniel Watson, Eric Tabellion, Aleksander Hołyński, Ben Poole, and Janne Kontkanen. Video interpolation with diffusion models, 2024.

[5] Liying Lu, Ruizheng Wu, Huaijia Lin, Jiangbo Lu, and Jiaya Jia. Video frame interpolation with transformer, 2022.

[6] Rabbia Mahum, Aun Irtaza, Marriam Nawaz, Tahira Nazir, Momina Masood, and Awais Mehmood. A generic framework for generation of summarized video clips using transfer learning (sumvclip). In *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, pages 1–8, 2021.

[7] Junting Pan, Ziyi Lin, Xiatian Zhu, Jing Shao, and Hongsheng Li. St-adapter: Parameter-efficient image-to-video transfer learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 26462–26477. Curran Associates, Inc., 2022.

[8] Umme Sara, Morium Akter, and Mohammad Shorif Uddin. Image quality assessment through fsim, ssim, mse and psnr—a com- parative study, 2019.

[9] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-a-video: Text-to-video generation without text-video data, 2022.

[10] Jacob Walker, Ali Razavi, and Aäron van den Oord. Predicting video with vqvae, 2021.

[11] Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. Videogpt: Video generation using vq-vae and transformers, 2021.

Appendix

Figure 1

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
[ ]  # Installs
     !pip install av
     !pip install --upgrade youtube_dl
     !pip install --upgrade pytorch_lightning
     !apt -y install ffmpeg lame
```

```
[ ]  # Imports
     import requests
     import os

     import os.path as osp
     import math
     import random
     import pickle
     import warnings

     import glob
     import h5py
     import numpy as np

     import torch
     import torch.utils.data as data
     import torch.nn.functional as F
     from torchvision.datasets.video_utils import VideoClips
     import pytorch_lightning as pl
```

```
[ ]  # Setup Dataset
     os.makedirs("video_dataset")
     os.makedirs("video_dataset/trainTANAY")
     os.makedirs("video_dataset/testTANAY")
```

```
[ ]  # CONFIG
     # Tanay is 1400 - 3000
     # Nithya is 3000 - 4800 (Start at 30)
     # Cathleen is 4800 - END (Roughly 6300) (Start at 48)
     KEEP_PROB = 0.2
     # Increment every time
     OVERALL_CHUNK_TRAIN = 15
     OVERALL_CHUNK_TEST = 14
     # Increase by between 100 and 250
     START_TRAIN = 2000
     START_TEST = 2200
     END_TEST = 2250 # 80% Train, 20% Test
```

Figure 2

```
[ ]  # Fetch video list
     video_list_url = "https://data.csail.mit.edu/tofu/dataset/original_video_list.txt"
     r = requests.get(video_list_url, auth=('user', 'pass'))
     full_video_list = r.text.split()

     def splits(video_list):
       train_list = video_list[START_TRAIN:START_TEST]
       test_list = video_list[START_TEST:END_TEST]
       return train_list, test_list

     # Test and train splits
     train_list, test_list = splits(full_video_list)

     print(len(full_video_list))
```

```
[ ]  # download train videos
     for ind,url in enumerate(train_list):
       !youtube-dl $url -f 'bestvideo[height<=480]' -o 'video_dataset/trainTANAY/%(title)s.mp4' -q

     # download test videos
     for ind,url in enumerate(test_list):
       !youtube-dl $url -f 'bestvideo[height<=480]' -o 'video_dataset/testTANAY/%(title)s.mp4' -q
```

```
[ ]  # Copied directly from https://github.com/wilson1yan/VideoGPT
     def preprocess(video, resolution, sequence_length=None):
         # video: THWC, {0, ..., 255}
         video = video.permute(0, 3, 1, 2).float() / 255. # TCHW
         t, c, h, w = video.shape

         scale = resolution / min(h, w)
         if h < w:
             target_size = (resolution, math.ceil(w * scale))
         else:
             target_size = (math.ceil(h * scale), resolution)
         video = F.interpolate(video, size=target_size, mode='bilinear',
                               align_corners=False)

         # center crop
         t, c, h, w = video.shape
         w_start = (w - resolution) // 2
         h_start = (h - resolution) // 2
         video = video[:, :, h_start:h_start + resolution, w_start:w_start + resolution]
         video = video.permute(1, 0, 2, 3).contiguous() # CTHW

         video -= 0.5

         return video
```

Figure 3

```
    # Custom dataset class
    class VideoDataset(data.Dataset):

        def __init__(self, data_folder, sequence_length, resolution=64):

            super().__init__()
            self.sequence_length = sequence_length
            self.resolution = resolution
            folder = osp.join(data_folder, 'trainTANAY')
            files = glob.glob(osp.join(folder, '**', f'*.mp4'), recursive=True)

            print(f"Found {len(files)} files")

            warnings.filterwarnings('ignore')
            clips = VideoClips(files, clip_length_in_frames=sequence_length, frames_between_clips=sequence_length, num_workers=1)
            print("files converted")

            self.clips = []

            for i in range(clips.num_clips()):
              # Sample with probability
              if random.random() > KEEP_PROB:
                continue
              try:
                video, _, _, idx = clips.get_clip(i)
                self.clips.append(preprocess(video, resolution))
              except:
                continue

        def __len__(self):
            return len(self.clips)

        def __getitem__(self, idx):
            return self.clips[idx], self.clips[idx][:, ::2]
```

```
[ ] train_dataset = VideoDataset('./video_dataset', 31, resolution=64)
    print((train_dataset[0][0].shape, train_dataset[0][1].shape), len(train_dataset))
```

Figure 4

```
[ ] # Copied directly from https://github.com/wilson1yan/VideoGPT
    def preprocess(video, resolution, sequence_length=None):
        # video: THWC, {0, ..., 255}
        video = video.permute(0, 3, 1, 2).float() / 255. # TCHW
        t, c, h, w = video.shape

        scale = resolution / min(h, w)
        if h < w:
            target_size = (resolution, math.ceil(w * scale))
        else:
            target_size = (math.ceil(h * scale), resolution)
        video = F.interpolate(video, size=target_size, mode='bilinear',
                              align_corners=False)

        # center crop
        t, c, h, w = video.shape
        w_start = (w - resolution) // 2
        h_start = (h - resolution) // 2
        video = video[:, :, h_start:h_start + resolution, w_start:w_start + resolution]
        video = video.permute(1, 0, 2, 3).contiguous() # CTHW

        video -= 0.5

        return video
```

Figure 5

```python
# Custom dataset class
class VideoDataset(data.Dataset):

    def __init__(self, data_folder, sequence_length, resolution=64):

        super().__init__()
        self.sequence_length = sequence_length
        self.resolution = resolution
        folder = osp.join(data_folder, 'testTANAY')
        files = glob.glob(osp.join(folder, '**', f'*.mp4'), recursive=True)

        print(f"Found {len(files)} files")

        warnings.filterwarnings('ignore')
        clips = VideoClips(files, clip_length_in_frames=sequence_length, frames_between_clips=sequence_length, num_workers=1)
        print("files converted")

        self.clips = []

        for i in range(clips.num_clips()):
          # Sample with probability
          if random.random() > KEEP_PROB:
            continue
          try:
            video, _, _, idx = clips.get_clip(i)
            self.clips.append(preprocess(video, resolution))
          except:
            continue

    def __len__(self):
        return len(self.clips)

    def __getitem__(self, idx):
        return self.clips[idx], self.clips[idx][:, ::2]
```

```python
[ ] test_dataset = VideoDataset('./video_dataset', 31, resolution=64)
    print((test_dataset[0][0].shape, test_dataset[0][1].shape), len(test_dataset))
```

Figure 6

```python
[ ] # Save to disk
    train_file = f'train_dataset_part{OVERALL_CHUNK_TRAIN}.pt'
    test_file = f'test_dataset_part{OVERALL_CHUNK_TEST}.pt'
    torch.save(torch.stack(train_dataset.clips).to(torch.float16), train_file)
    torch.save(torch.stack(test_dataset.clips).to(torch.float16), test_file)
```

```python
    train_loc = f'/content/drive/MyDrive/GenAIFinalProject/Data/Train/{train_file}'
    test_loc = f'/content/drive/MyDrive/GenAIFinalProject/Data/Test/{test_file}'

    !mv $train_file $train_loc
    !mv $test_file $test_loc
```

```python
!rm -rf video_dataset
```

Figure 7

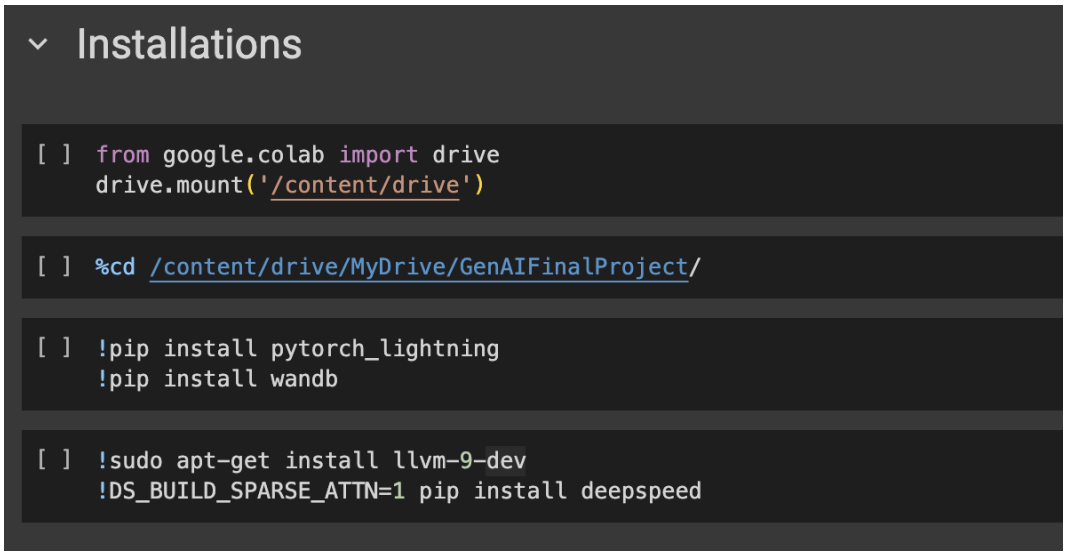

Installations
[ ] from google.colab import drive
drive.mount('/content/drive')
[ ] %cd /content/drive/MyDrive/GenAIFinalProject/
[ ] !pip install pytorch_lightning
!pip install wandb
[ ] !sudo apt-get install llvm-9-dev
!DS_BUILD_SPARSE_ATTN=1 pip install deepspeed

Figure 8

```
[ ]  import torch.utils.data as data
     import glob
     import os.path as osp
     import torch
     import pytorch_lightning as pl
     import os
     from pytorch_lightning.callbacks import ModelCheckpoint

     root = "./Data/"

     # Custom dataset class that collates saved video sequences for processing
     class ClipDataset(data.Dataset):

         def __init__(self, root):

             super().__init__()
             self.items = []
             files = glob.glob(osp.join(root, '**', f'*.pt'), recursive=True)

             print(f"Loading {len(files)} stores")
             for file in files:
               vals = torch.load(file).to(torch.float)
               clips = torch.unbind(vals)
               self.items += clips
             self.sequence_length = self.items[0].shape[1]
             self.resolution = self.items[0].shape[2]

         def __len__(self):
             return len(self.items)

         def __getitem__(self, idx):
             return self.items[idx]
```

Figure 9

```
    # Adapater for torch lightning
    class VideoData(pl.LightningDataModule):

        def __init__(self, root):
            super().__init__()
            self.train_root = osp.join(root, "Train")
            self.test_root = osp.join(root, "Test")

        def _dataloader(self, files):
            dataset = ClipDataset(files)
            dataloader = data.DataLoader(
                dataset,
                batch_size=4,
                num_workers=16,
                pin_memory=True,
                shuffle=True
            )
            return dataloader

        def train_dataloader(self):
            return self._dataloader(self.train_root)

        def val_dataloader(self):
            return self._dataloader(self.test_root)

        def test_dataloader(self):
            return self.val_dataloader()

    all_data = VideoData(root)
    train_dl = all_data.train_dataloader()
    test_dl = all_data.test_dataloader()
```

Figure 10


Training the VQ-VAE
[ ]
# Borrowed and modified from the VideoGPT github: https://github.com/wilson1yan/VideoGPT.git
import argparse
import pytorch_lightning as pl
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.loggers import WandbLogger
import wandb
import gc
gc.collect()
torch.cuda.empty_cache()
wandb.login(key="f9e8ebaee8545d7ff6e3b1afa4f192d01079b34a")
def main():
pl.seed_everything(1234)
parser = argparse.ArgumentParser()
data = VideoData(root)
model = VQVAE()
callbacks = []
callbacks.append(ModelCheckpoint(monitor='val/recon_loss', mode='min'))
kwargs = dict()
wandb_logger = WandbLogger(project="VideoGPT VQ-VAE")
trainer = pl.Trainer(accelerator="gpu", max_epochs=200, logger = wandb_logger)
trainer.fit(model, data)
if __name__ == '__main__':
main()

Figure 11

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

[ ]  %cd /content/drive/MyDrive/GenAIFinalProject/

[ ]  root = "/content/drive/MyDrive/GenAIFinalProject/"

[ ]  !pip install -r /content/drive/MyDrive/GenAIFinalProject/requirements.txt
     !pip install importnb
     !pip install wandb
     !pip install import-ipynb


[ ]  !pip install ipynb

[ ]  import os
     import itertools
     import numpy as np
     import argparse
     import gdown
     import torch
     import torch.nn as nn
     import torch.nn.functional as F
     import torch.optim.lr_scheduler as lr_scheduler
     import pytorch_lightning as pl
     import os.path as osp
     import wandb
     import gc

     from tqdm import tqdm
     from torchmetrics.image import PeakSignalNoiseRatio
     from pytorch_lightning.callbacks import ModelCheckpoint
     from pytorch_lightning.loggers import WandbLogger


     from resnet import resnet34
     from attention import AttentionStack, LayerNorm, AddBroadcastPosEmbed
     from utils import shift_dim
     from importnb import imports
     import import_ipynb

     from videotransformerdataset import VideoDataset

     from vq_vae import VQVAE
```

Figure 12

```
[ ]  import torch.utils.data as data
     import glob
     import os.path as osp
     import torch
     import pytorch_lightning as pl
     import os
     from torch.utils.data import SubsetRandomSampler
     from pytorch_lightning.callbacks import ModelCheckpoint

     # Custom dataset class
     class ClipDataset(data.Dataset):

         def __init__(self, root):

             super().__init__()
             self.items = []
             files = glob.glob(osp.join(root, '**', f'*.pt'), recursive=True)

             print(f"Loading {len(files)} stores")
             for file in files:
               vals = torch.load(file).to(torch.float)
               clips = torch.unbind(vals)
               self.items += clips
             self.sequence_length = self.items[0].shape[1]
             self.resolution = self.items[0].shape[2]

         def __len__(self):
             return len(self.items)

         def __getitem__(self, idx):
             return self.items[idx]
```

Figure 13

```
    class VideoData(pl.LightningDataModule):

        def __init__(self, root):
            super().__init__()
            self.train_root = osp.join(root, "Train")
            self.test_root = osp.join(root, "Test")

        def _dataloader(self, files):
            dataset = ClipDataset(files)
            dataloader = data.DataLoader(
                dataset,
                batch_size=4,
                num_workers=16,
                pin_memory=True,
                sampler=SubsetRandomSampler(torch.randint(high=len(dataset), size=(int(0.1 * len(dataset)),)))
            )
            return dataloader

        def train_dataloader(self):
            return self._dataloader(self.train_root)

        def val_dataloader(self):
            return self._dataloader(self.test_root)

        def test_dataloader(self):
            return self.val_dataloader()
```

Figure 14

```
[ ]  import gc

     gc.collect()
     torch.cuda.empty_cache()

     wandb.login(key="6839b73984bcab4842ff37c1f24655fdb496167e") #API Key is in your wandb account, under settings (wandb.ai/settings)

     class Args(argparse.Namespace):
         data_path = root+'Data'
         vqvae = root+'VideoGPT VQ-VAE/jpn9i0q0/checkpoints/epoch=44-step=29340.ckpt'
         n_cond_frames = 16
         resolution = 64
         sequence_length = 31
         batch_size = 1
         num_workers = 8
         hidden_dim = 288
         max_steps = 100
         max_epochs = 100
         heads = 4
         layers = 5 # 6
         dropout = 0.2
         attn_type = "full"
         attn_dropout = 0.3
         class_cond = None
         gpus = 1

     def main():
         pl.seed_everything(1234)
         args = Args()

         data = VideoData(args.data_path)
         # pre-make relevant cached files if necessary
         data.train_dataloader()
         data.test_dataloader()

         args.class_cond_dim = data.n_classes if args.class_cond else None
         model = VideoGPT(args)

         callbacks = []
         callbacks.append(ModelCheckpoint(monitor='val/loss', mode='min', save_top_k=-1))

         kwargs = dict()
         if args.gpus > 1:
             # find_unused_parameters = False to support gradient checkpointing
             kwargs = dict(distributed_backend='ddp', gpus=args.gpus,
                           plugins=[pl.plugins.DDPPlugin(find_unused_parameters=False)])

         wandb_logger = WandbLogger(project="VideoGPT VQ-VAE")

         trainer = pl.Trainer(accelerator="gpu", max_epochs=200, logger = wandb_logger)

         trainer.fit(model, data)


     if __name__ == '__main__':
         main()
```

Figure 15

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

[ ] %cd /content/drive/MyDrive/GenAIFinalProject/

[ ] !pip install import-ipynb
    !pip install importnb
    !pip install ipynb
    !pip install transformers==4.29.2
    !pip install pytorch-ignite

[ ] !pip install -r requirements.txt

[ ] !pip install wandb

[ ] import os
    import itertools
    import numpy as np
    import argparse
    import gdown
    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    import torch.optim.lr_scheduler as lr_scheduler
    import pytorch_lightning as pl
    import os.path as osp
    import wandb
    import gc

    from tqdm import tqdm
    from torchmetrics.image import PeakSignalNoiseRatio
    from pytorch_lightning.callbacks import ModelCheckpoint
    from pytorch_lightning.loggers import WandbLogger


    from resnet import resnet34
    from attention import AttentionStack, LayerNorm, AddBroadcastPosEmbed
    from utils import shift_dim
```

Figure 16

```
[ ] !wget data.csail.mit.edu/tofu/testset/vimeo_super_resolution_test.zip

[ ] !unzip vimeo_super_resolution_test.zip

[ ] !rm vimeo_super_resolution_test.zip

[ ] !rm -rf vimeo_super_resolution_test/low_resolution/
    !rm -rf vimeo_super_resolution_test/target/

[ ] import random

    f = open("./vimeo_super_resolution_test/sep_testlist.txt", "r")
    l = list(f.read().split())
    random.shuffle(l)
    to_remove = l[1000:]
    all_remove = ["vimeo_super_resolution_test/input/" + seq for seq in to_remove]
    keep = l[:1000]

[ ] keep = l[:1000]

[ ] for seq in all_remove:
      !rm -rf seq
```

Figure 17

```
[ ]  import os
     from torch.utils.data import Dataset, DataLoader
     from torchvision.transforms import Resize
     from torchvision.io import read_image
     from PIL import Image
     import random
     import torch

     class VimeoSepTuplet(Dataset):
         def __init__(self, data_root, seqs):
             """
             Creates a Vimeo Septuplet object.
             Inputs.
                 data_root: Root path for the Vimeo dataset containing the sep tuples.
                 is_training: Train/Test.
                 input_frames: Which frames to input for frame interpolation network.
             """
             self.sequences = []

             for i in range(len(seqs)):
               try:
                 print(i)
                 seq = torch.zeros((3, 7, 64, 64))
                 for j in range(7):
                   img = read_image(data_root + seqs[i] + "/im" + str(j + 1) + ".png")
                   img = Resize((64, 64))(img)
                   seq[:, j, :, :] = img
                 self.sequences.append(seq)
               except:
                 continue
             self.data_root = data_root

         def __getitem__(self, index):
           return self.sequences[index]

         def __len__(self):
             return len(self.sequences)


     dataset = VimeoSepTuplet("vimeo_super_resolution_test/input/", keep[:200])
     dataloader = DataLoader(dataset, batch_size=1, shuffle=True, pin_memory=True)
```

Figure 18

```
   # This reformats shorter sequences using pattern as the transfer function
   def restructure(seq, pattern):
     start = 31 - sum(pattern)
     full = torch.zeros((1, 3, 31, 64, 64))

     aves = torch.mean(seq, 0)

     for i in range(seq.shape[0]):
       full[:, i, :, :, :] = aves[i]

     offset = 0
     for i in range(len(pattern)):
       for j in range(pattern[i]):
         full[:, :, start + offset, :, :] = seq[:, 2 * i, :, :]
         offset += 1
     return (full / 255) - 0.5
```

```
[ ]  gc.collect()
     torch.cuda.empty_cache()
     # This predicts the next sequence
     def predict(model, seq, patterns):
       copy = seq.detach().clone()
       original = seq
       seq = copy

       for i in range(len(patterns)):
         reversed = bool(i >= ((len(patterns) + 1) // 2))
         pattern = patterns[i]
         cutoff = seq.shape[1] - 2 * min(i, len(patterns) - i) - 2 + 2 * int(reversed)

         temp = seq
         if reversed:
           temp = torch.flip(temp, [1]).detach().clone()
         full = restructure(temp[:, :cutoff, :, :], pattern).to("cuda")
         model = model.to("cuda")

         next = model.training_step(full, full)
         vals = torch.argmax(next[1], dim=4)
         recon = model.vqvae.decode(vals)
         seq[:, seq.shape[1] - 2 - 2 * i, :, :] = (recon[:, :, 30, :, :] - torch.min(recon)) / (torch.max(recon) - torch.min(recon)) * 255

       return seq, original

     seq, original = predict(model, dataset[7], [[2, 1, 2], [2, 1], [2, 1, 2]])
```

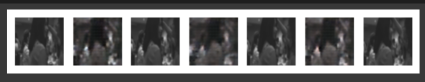
Figure 19

```
[ ]  # Visualization code
     import matplotlib.pyplot as plt

     seq_vis = torch.round(seq.permute(1, 2, 3, 0)) # THWC
     original_vis = torch.round(original.permute(1, 2, 3, 0))

     f, axarr = plt.subplots(1, seq_vis.shape[0])

     for i in range(seq_vis.shape[0]):
       axarr[i].axis("off")
       plot = axarr[i].imshow(seq_vis[i, :, :, :].numpy() / 255)
```



```
[ ]  f, axarr = plt.subplots(1, original_vis.shape[0])
     for i in range(seq_vis.shape[0]):
       axarr[i].axis("off")
       plot = axarr[i].imshow(original_vis[i, :, :, :].numpy() / 255)
```

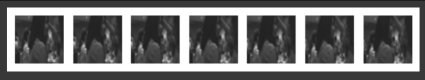


Figure 20

```
[ ]  # Loops through and evaluates data
     from ignite.metrics import PSNR, SSIM
     from ignite.engine import *

     PATTERN = [[2, 3, 1], [2, 1], [2, 3, 1]]

     def eval_step(engine, batch):
         return batch

     default_evaluator = Engine(eval_step)
     PSNR(data_range=1.0).attach(default_evaluator, 'psnr')
     SSIM(data_range=1.0).attach(default_evaluator, 'ssim')

     n = 0
     running_psnr = 0
     running_ssim = 0

     print(model.device)

     for x in tqdm(dataloader):
       seq, original = predict(model, x[0], PATTERN)
       state = default_evaluator.run([[seq / 255, original / 255]])
       running_psnr = (n * running_psnr + state.metrics['psnr']) / (n + 1)
       running_ssim = (n * running_ssim + state.metrics['ssim']) / (n + 1)
       n += 1
       print(f"PSNR: {running_psnr}, SSIM: {running_ssim}")
```