Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Cuda Visin Lab Report

Predicting Video Frames Using Transformer Models

Aidin Masroor

Instructor

Angel Angel Villar-Corrales

October 2025

1 Problem Statement

- In this project, we will explore the use of transformer models for predicting future frames in video sequences. The goal is to develop a model that can accurately forecast upcoming frames based on a series of preceding frames.
- So we train a transformer model on a dataset of video sequences, where the input is a sequence of frames and the output is the predicted next frame.
- The model will be evaluated based on its prediction accuracy, using metrics such as Least Absolute Deviations (L1), Structural Similarity Index (SSIM) and Learned Perceptual Image Patch Similarity (LPIPS).
- The model contains and encoder, a predictor (encoder) and a decoder.
- The encoder reads video frames and maps them to a feature space, the predictor reads five feature encoded vectors, which are corresponding to five consecutive video frames, and generates the next feature vector, which corresponds to the next video frame. In this way the predictor generates multiple feature vectors as predicted feature vectors. The decoder reads them and generates images for each of those feature vectors.
- This is how the model reads video frames and predicts the following video frames.
- The dataset consists of 250 videos for validation and 9737 videos for training, each video contains 24 frames of size 128x128 pixels.

2 Approach

- The model architecture is based on transformer model, and will be explained in detail in here.

2.1 Encoder

- There are two types of encoders implemented, one is object based encoder and the other is image based encoder.

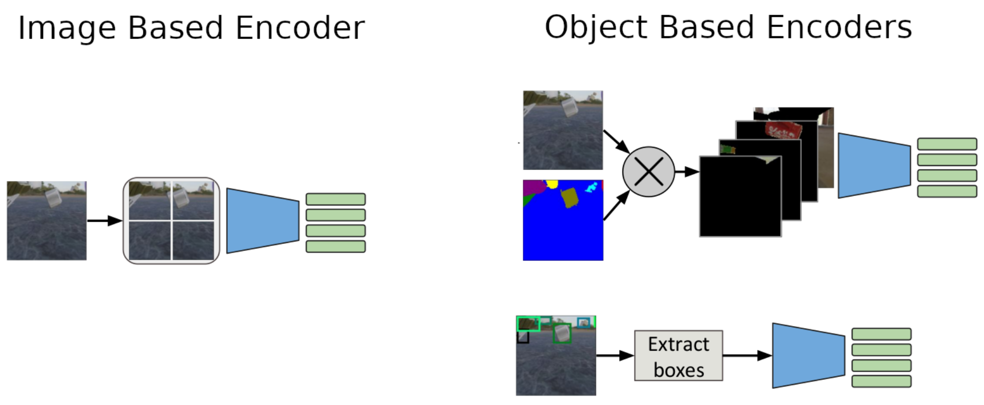

Figure 1: Two encoder types

2.1.1 Object Based Encoders

- The chosen encoder model here is Vision Transformer (ViT) based encoder.
- To handle input, either we use bounding boxes of objects in the video frames, or we use masks to encode the video frames. In fact, bounding boxes could be treated as rectangular masks, which have overlaps sometimes.
- Whether we pass bounding boxes or masks along side the rgb images to the encoder, the encoder extracts each object in the frame, also encoder behaves the background as an object.
- There are up to 10 objects in each frame and one background, the encoder extracts 11 objects `(maximum 10 objects + 1 background)` from each frame. If less than 10 objects are present in the frame, the encoder extracts images filled with zeros.
- The implemented encoder is on the main branch and accessible on the file `src/models/transformers/encoders/vit_encoder.py`. And its usage is in the file `src/models/transformers/encoders/VIT_Encoder.ipynb`.

- Data shapes are as following, bounding boxes have data shape of `[batch_size, number_of_frames_per_video, number_of_objects, 4]`, masks have data shape of `[batch_size, number_of_frames_per_video, image_height, image_width]`, and the rgb images have data shape of `[batch_size, number_of_frames_per_video, 3, image_height, image_width]`.
- The encoder outputs one feature vector for each rgb scene, so output of the encoder is a feature vector with data shape of `[batch_size, number_of_frames_per_video, feature_vector_dimension]`.

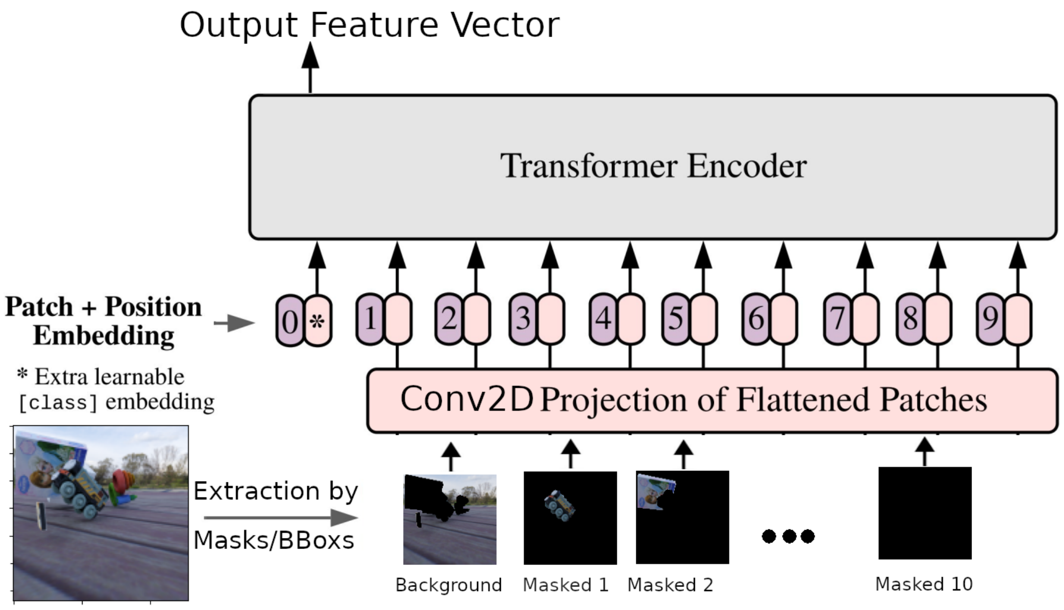
Output Feature Vector
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Conv2D Projection of Flattened Patches
Extraction by Masks/BBoxs
Background
Masked 1
Masked 2
Masked 10

Figure 2: Implemented ViT encoder structure.

2.1.2 Image Based Encoder

- To see the codes for this encoder please refer to the branch *Image_Based_Branch*, and the file `src/models/transformers/encoders/image_level/ vit_like_encoder.py`.
- The structue here is similar to the object based encoder, but here we do not pass bounding boxes or masks to the encoder, instead we pass patched

images to the ViT encoder.

2.2 Decoder

- The implemented decoder is on the main branch and accessible on the file `src/models/transformers/decoders/vit_decoder.py` and its usage is in the file `src/models/transformers/decoders/VIT_Decoder.ipynb`.
- In the decoder, we initialize a learnable parameter (Q), which is a feature vector with the same dimension as the output of the encoder.
- The decoder consists of a multihead attention layer, a cross attention layer and a feed forward layer.
- The output of the encoder is passed as key and value to the cross attention layer.
- The decoder reconstructs the images from feature space so the output shape of decoder is `[batch_size, number_of_frames_per_video, 3, img_height, img_width]`.

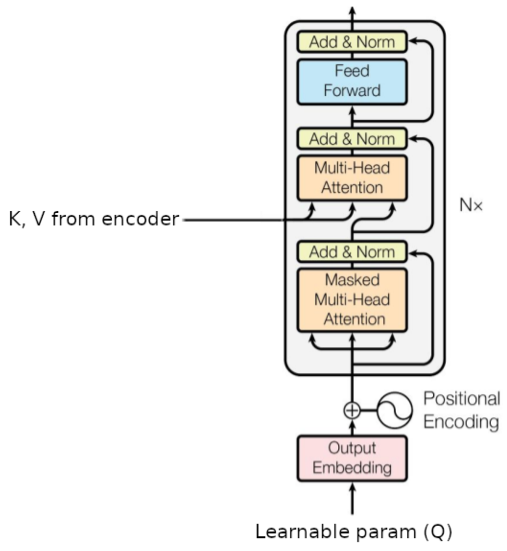

Figure 3: Decoder model.

2.3 Predictor

- This part of the project is not implemented yet, but the structure is similar to the encoder.
- The predictor reads five feature encoded vectors, and generates the next feature vector.
- It reads $(X_1, \ldots, X_5)$ and generates $\hat{X}_6$. Then it reads $(X_2, \ldots, X_5, \hat{X}_6)$ and generates $\hat{X}_7$. In this regard, it generates the next 5 frames. Additionally, metrics such as the `cosine loss function` or `MSE` can be used to train the predictor.

2.4 Transformer

- The transformer model consists of the explained above encoder and decoder.
- On branch main, please refer to the file `src/models/transformers/CustomTransformer.py` to see the code.

2.5 Training & Evaluation

- To see the training code and usage of the model (on branch main) please refere to the file `src/demo/ViT_Full_Transformer_Demo.ipynb`
- For evaluation we use three metrics, Least Absolute Deviations (L1), Structural Similarity Index (SSIM) and Learned Perceptual Image Patch Similarity (LPIPS). Each have their owm characteristics and advantages, and their contribution to the final loss should be weighted. The weights used for L1, SSIM and LPIPS are 1.0, 0.5 and 0.1 respectively.
- During training the loss value when we used masks instead of bounding boxes, dropped quicker. So we mostly continued our evaluation using mask extractors.
- After 60 epochs of training the model using masks, the loss value decreased but the reconstruction of images by decoder was not satisfactory. mostly images were nosity shape.

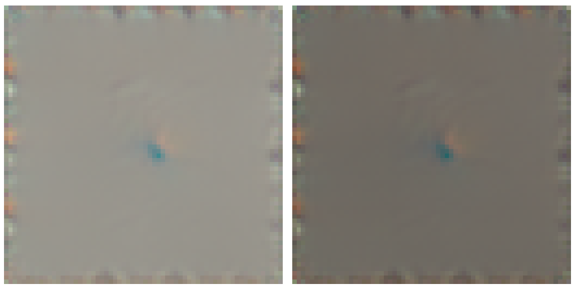
(a) (b)

Figure 4: Reconstructed images.

- One promising point is that the intensity of the reconstructed images is similar to the intensity of the correspond rgb images. It meanse brighter rgb images, produce brighter reconstruction and darker rgb images produce darker reconstructed images.
- In branch main, under the path `src/saved_images` you may see the reconstructed images for object based model.

3 Todos

- Still there are improvements to be done on the encoder-decoder, to have better image reconstruction.
- The predictor model is not implemented yet, although the structue is similar to the encoder. After obtaining satisfactory results from encoder-decoder, we will implement the predictor.

4 Challenges

- First, reading all 24 frames of each videos, made the iteration through the dataset very slow, so we limited the number of frames to 4.
- Reconstructed images by decoder were almost gray like images, after some debugging I found maybe the layer normalization in the encoder through

the three channels of rgb images (channels*img_height*img_width) could be problematic. I removed that and also replaced the Linear patch projections of encoder, decoder with conv2d and conv2dTranspose layers respectively. The results became better but still they are not satisfactory