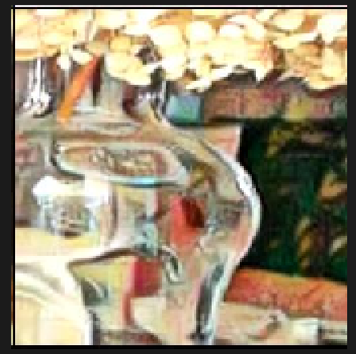
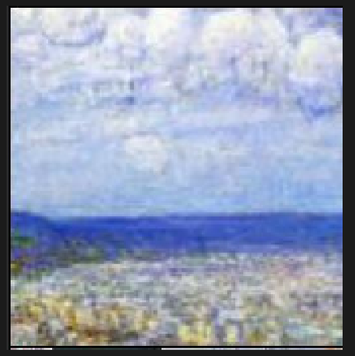




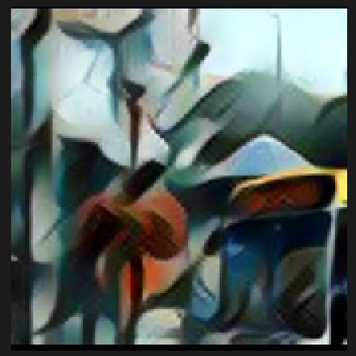



IMAGE STYLE TRANSFER USING TRANSFORMERS

Making a transformer based architecture for style transfer of images

Reimplementation of the paper:
Image Style Transfer with Transformers
(https://arxiv.org/pdf/2105.14576)

Sarthak Chittawar **2021111010**
Sanika Damle **2021115005**

ARCHITECTURE OVERVIEW

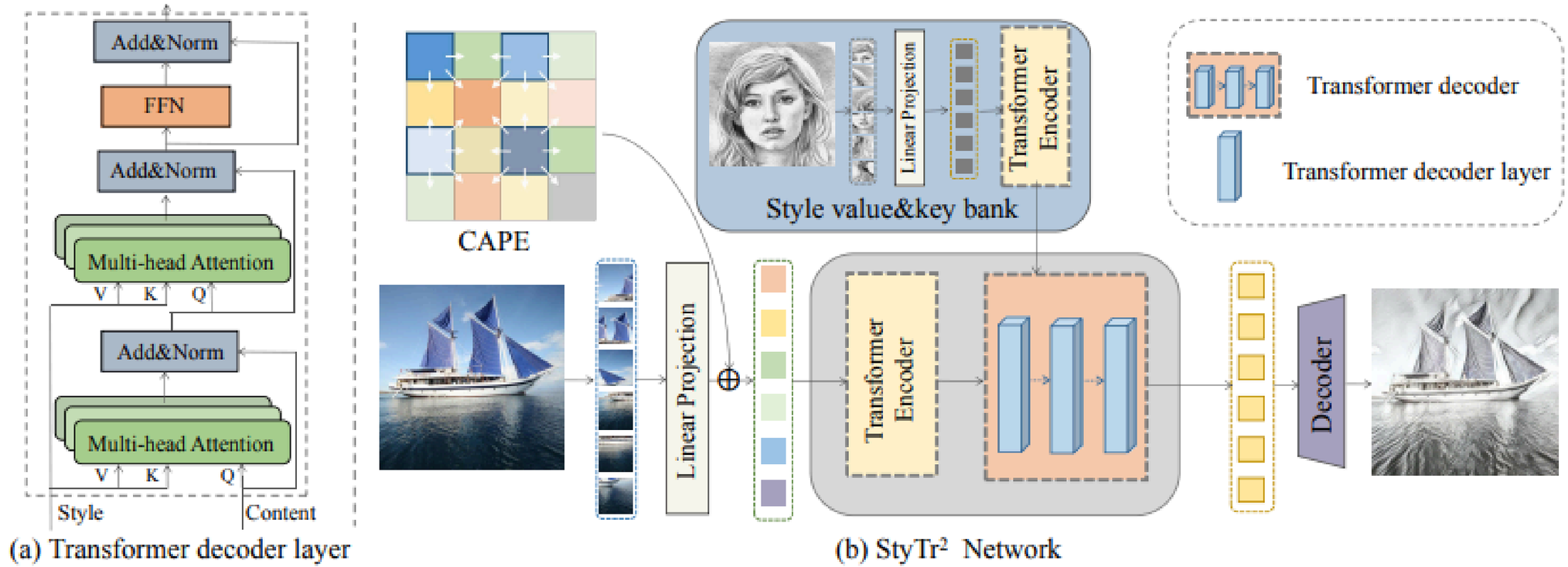


Fig. 2. Overall pipeline of our StyTr^2 framework. We split the content and style images into patches and use a linear projection to obtain patch sequences. Then, the content sequences added with CAPE are fed into a content transformer encoder, while the style sequences are fed into a style transformer encoder. Following the two transformer encoders, a multi-layer transformer decoder is adopted to stylize the content sequences according to the style sequences. Finally, we use a progressive upsampling decoder to obtain the final output.

Image credits: https://arxiv.org/pdf/2105.14576

CAPE

(Content Aware Positional Encoding)

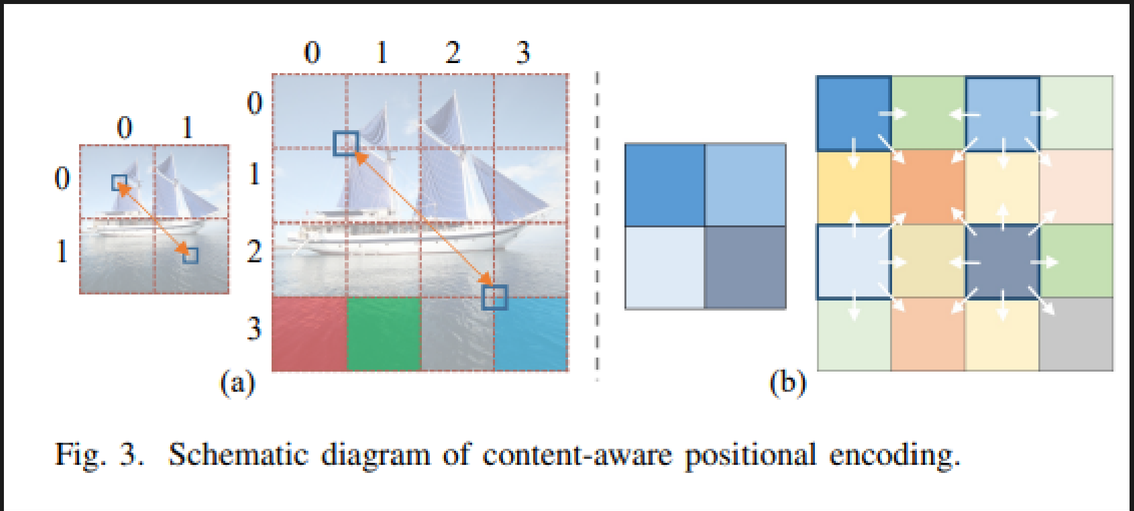


Fig. 3. Schematic diagram of content-aware positional encoding.

$$\mathcal{P}_{\mathcal{L}} = \mathcal{F}_{\text{pos}}(\text{AvgPool}_{n\times n}(\mathcal{E})),$$

$$\mathcal{P}_{\mathcal{CA}}(x,y) = \sum_{k=0}^{s}\sum_{l=0}^{s}(a_{kl}\mathcal{P}_{\mathcal{L}}(x_k,y_l)),$$

Image credits: https://arxiv.org/pdf/2105.14576

Traditional positional encoding only depends on the distance between patches

$$\mathcal{P}(x_i,y_i)^T\mathcal{P}(x_j,y_j) = \sum_{k=0}^{\frac{d}{4}-1}[\cos(w_k(x_j-x_i))+\cos(w_k(y_j-y_i))],$$

where $w_k = 1/10000^{2k/128}, d = 512$

This is not scale invariant, designed for language tasks

P_L is a learnable positional encoding

s is the number of neighbouring patches

This is added to the sequential feature encoding for each ith (x, y) patch

TRANSFORMER ENCODER

Two transformer encoders are used, one for the content image and one for the style.

Each of the 3 layers has 2 multi-head self attention modules and a feedforward network.

There is layer normalization after each sequence, along with dropout layers.

Below equation shows how the Multi-head self-attention is computed:

$$\mathcal{F}_{\mathrm{MSA}}(Q,K,V) = \mathrm{Concat}(\mathrm{Attention}_1(Q,K,V), \ldots, \mathrm{Attention}_N(Q,K,V))W_o,$$

Image credits: https://arxiv.org/pdf/2105.14576

The Query & Key are taken as the source image (content/style) with added position embedding, while Value is the source image.

TRANSFORMER DECODER

The content encoding is used to generate the query and the style is used for the key and value.

$$Q = \hat{Y}_c W_q, \quad K = Y_s W_k, \quad V = Y_s W_v.$$

Image credits: https://arxiv.org/pdf/2105.14576

$\hat{Y}_c, \hat{Y}_s$ are the encoded content & style images

W_q, W_k, W_v are the query, key, value weights

Two multi-headed self-attention modules with feed-forward networks are used in each of the 3 Decoder layers.

The output sequence is passed through a pretrained CNN decoder to get the final image.

LOSS CALCULATION

There are 4 components that go into the loss

1. Content Loss

This calculates the MSE between the content image and the image that is generated

$$L_c = \frac{1}{N_l} \sum_{i=0}^{N_l} ||\phi_i(I_o) - \phi_i(I_c)||_2$$

Here, $\phi_i()$ refers to the features extracted from the ith layer from the pretrained VGG19 model.

I_c is the initial content image
I_o is the final output image
N_l is the number of VGG19 layers considered

2. Style Loss

This calculates the MSE between the style image and the image that is generated

$$L_s = \frac{1}{N_l} \sum_{i=0}^{N_l} ||\mu(\phi_i(I_o)) - \mu(\phi_i(I_s))||_2 + ||\sigma(\phi_i(I_o)) - \sigma(\phi_i(I_s))||_2$$

Here, $\phi_i()$ refers to the features extracted from the ith layer from the pretrained VGG19 model.

I_s is the initial style image
I_o is the final output image
N_l is the number of VGG19 layers considered

The style loss is calculated differently as we do not want to superimpose the style image directly onto the final content image

LOSS CALCULATION

There are 4 components that go into the loss

3. Identity Loss 1

Two same content images are passed as style and content, and the MSE loss of the resulting output with the original content image is calculated. This is done for style as well.

$$L_{id1} = ||I_{cc} - I_c||_2 + ||I_{ss} - I_s||_2$$

4. Identity Loss 2

The variables used are the same as Identity Loss 1, but here the different between feature maps is calculated.

$$L_{id2} = \frac{1}{N_l} \sum_{i=0}^{N_l} ||\phi_i(I_{cc}) - \phi_i(I_c)||_2 + ||\phi_i(I_{ss}) - \phi_i(I_s)||_2$$

Final Loss

$$L = 10L_c + 7L_s + 50L_{id1} + 1L_{id2}$$

RESULTS

Dataset Details

Content Dataset: COCO2017 Dataset (original paper used COCO2014)

Style Dataset: Collected from WikiArt

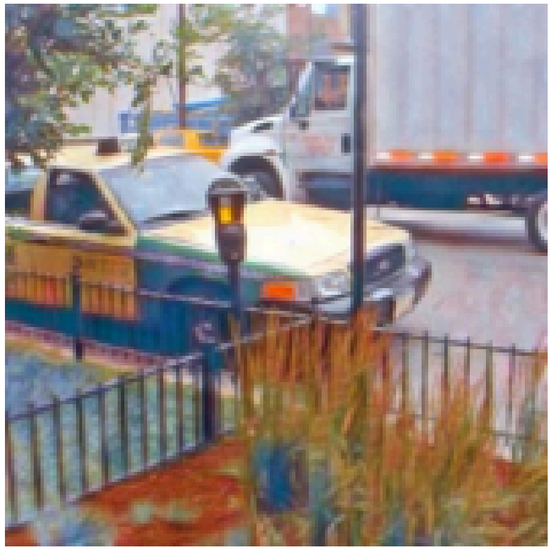
Training Details

- **Epochs :** 160000
- **Batch Size :** 8
- **Embedding size :** 512
- **Learning Rate :** 5e-4 with a decay of 1e-5 after 1000 epochs

- **Compute used :** Multi GPU training on 4 NVIDIA GeForce RTX 2080 Ti GPUs for 20 hours.

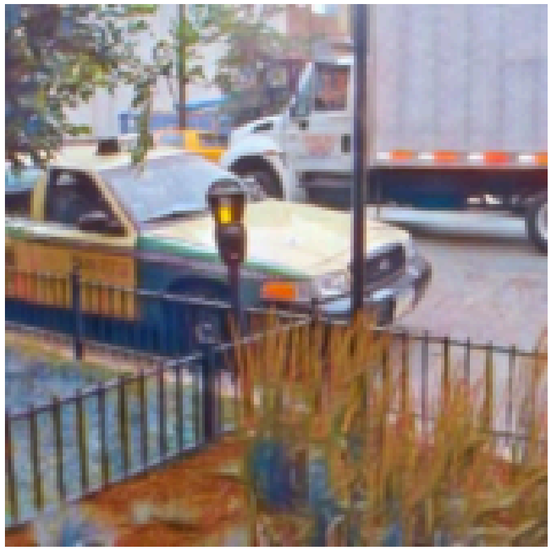
Content

Without CAPE



Style

With CAPE




ABLATIONS

TRANSITION OF OUTPUT IMAGES ACROSS TRAINING EPOCHS

Content Image

Style Image

TRANSITION OF OUTPUT IMAGES ACROSS TRAINING EPOCHS

Without CAPE

With CAPE

ACTIVATION FUNCTION:
ReLU v/s GeLU

ReLU ←

GeLU →

QUALITATIVE COMPARISON WITH CNN-BASED METHODS

Content Image

Style Image

QUALITATIVE COMPARISON WITH CNN-BASED METHODS

Image Style Transfer Using Convolutional Neural Networks

Content Loss: 9.704
Style Loss: 7.467

StyTR

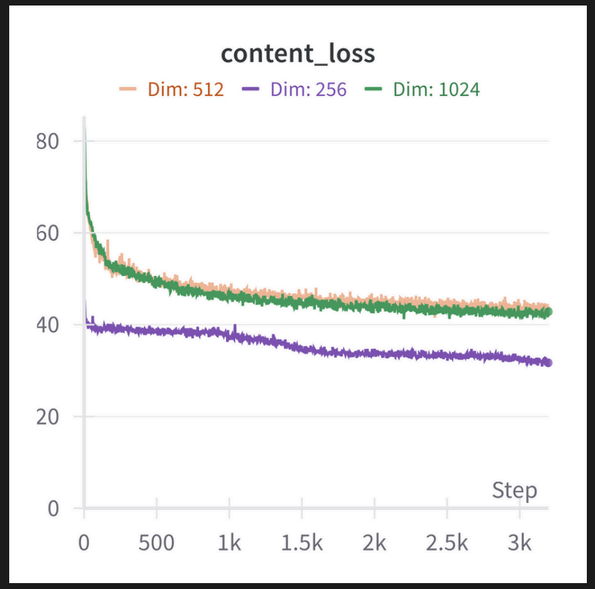
Content Loss: 10.052
Style Loss: 4.301

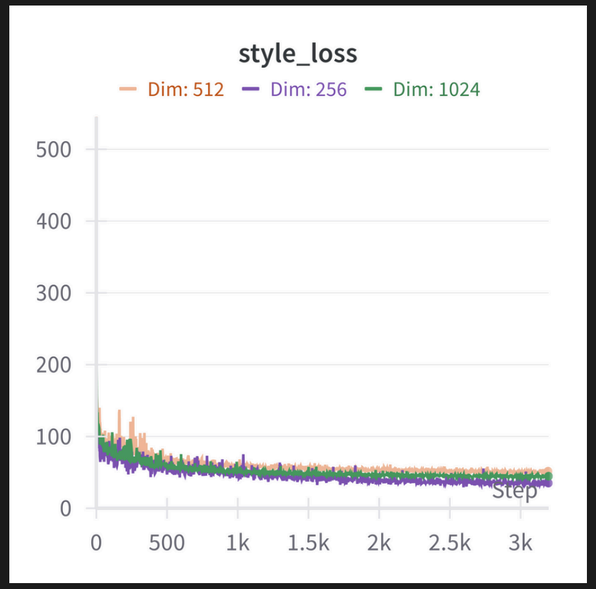
Fast Style Transfer (CNN)

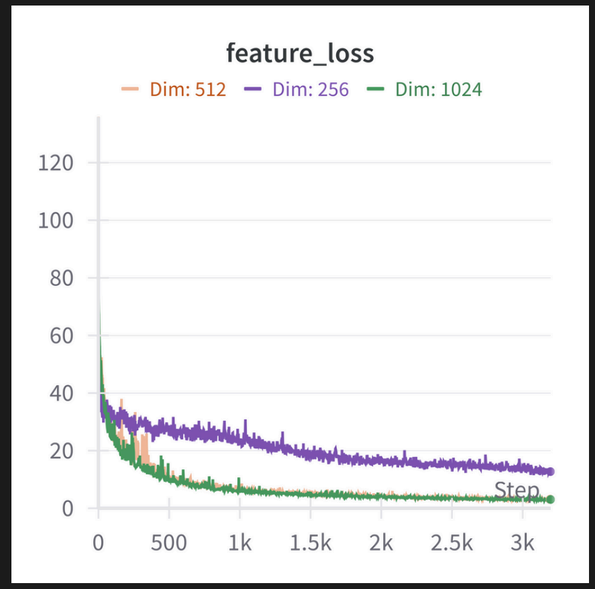
Content Loss: 9.903
Style Loss: 5.073

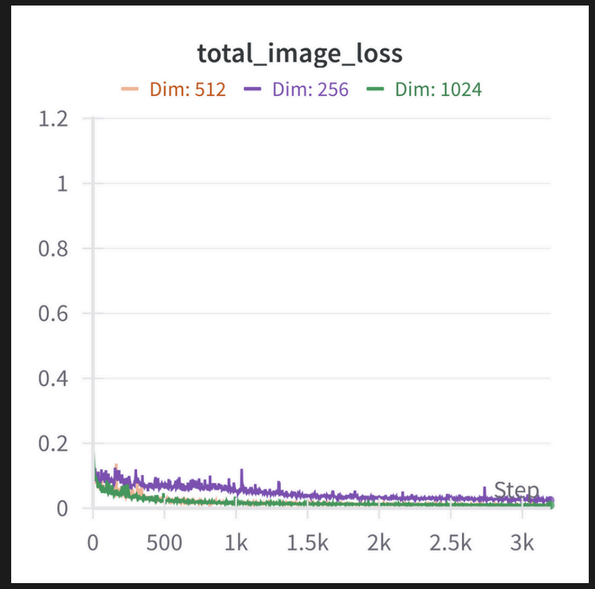
QUANTITATIVE RESULTS

We studied the effect of hidden dimension size in terms of losses:





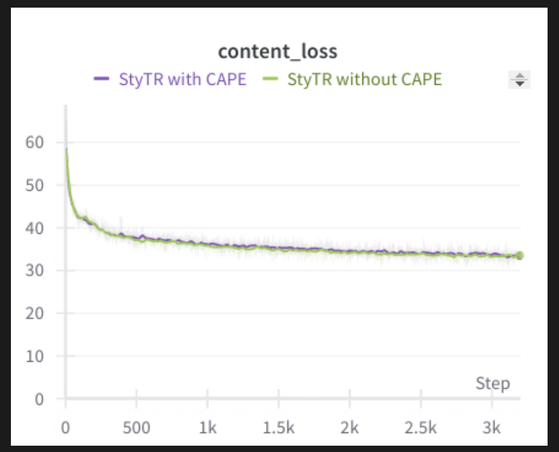


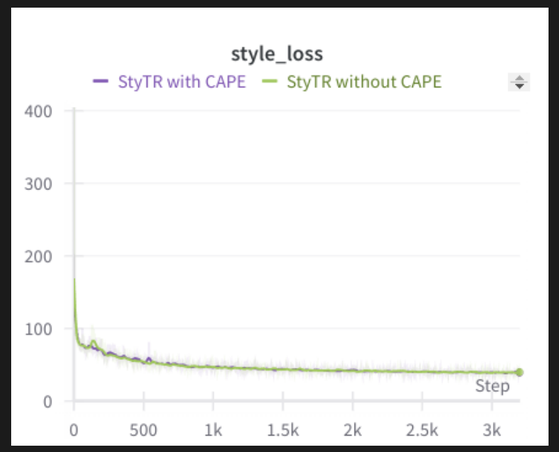


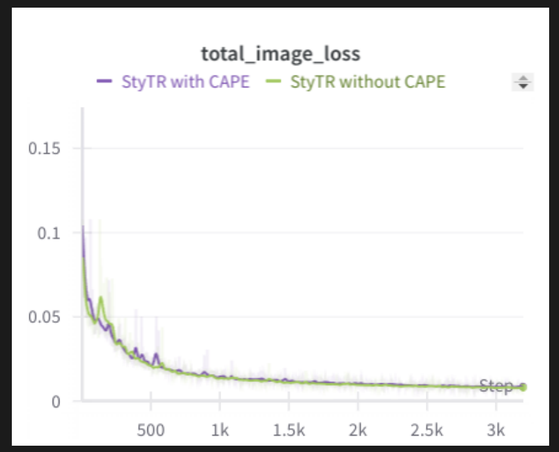
Note: Style transfer is a subjective comparison and therefore most research papers perform surveys to quantify their outputs as most preferred.

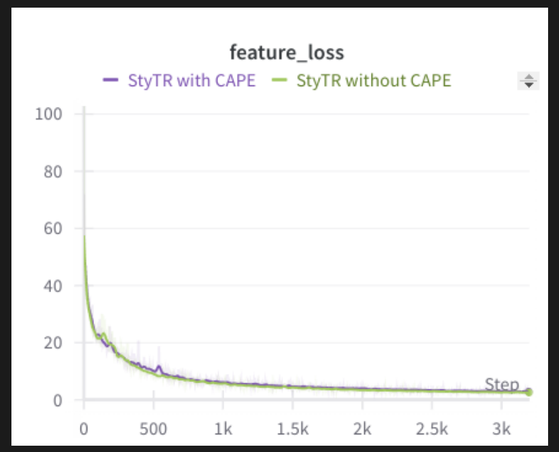
QUANTITATIVE RESULTS

Effect of CAPE









CAPE encoding affects the image quality, but not the loss as much. Because of the way that the loss is calculated, the intensity of the content image matters more than small features. However, when we visualize images that have been generated by both models, the difference is apparent.


NOVELTY

USING CLIP TO RETRIEVE IMAGES

1. Getting tags for wikiart images

Images had tags like Realism, Pointilism, etc along with the artist name and a short description of the objects in the painting

2. Converting tags to captions

Got short descriptions for each art style and concatenated that with the artist and the description for each painting

3. Fine-tuning CLIP

Fine-tuned a pretrained CLIP model with the art dataset and the captions produced

When the user inputs a description of a style that they want, the style is retrieved using the CLIP model and then applied to the specific content image

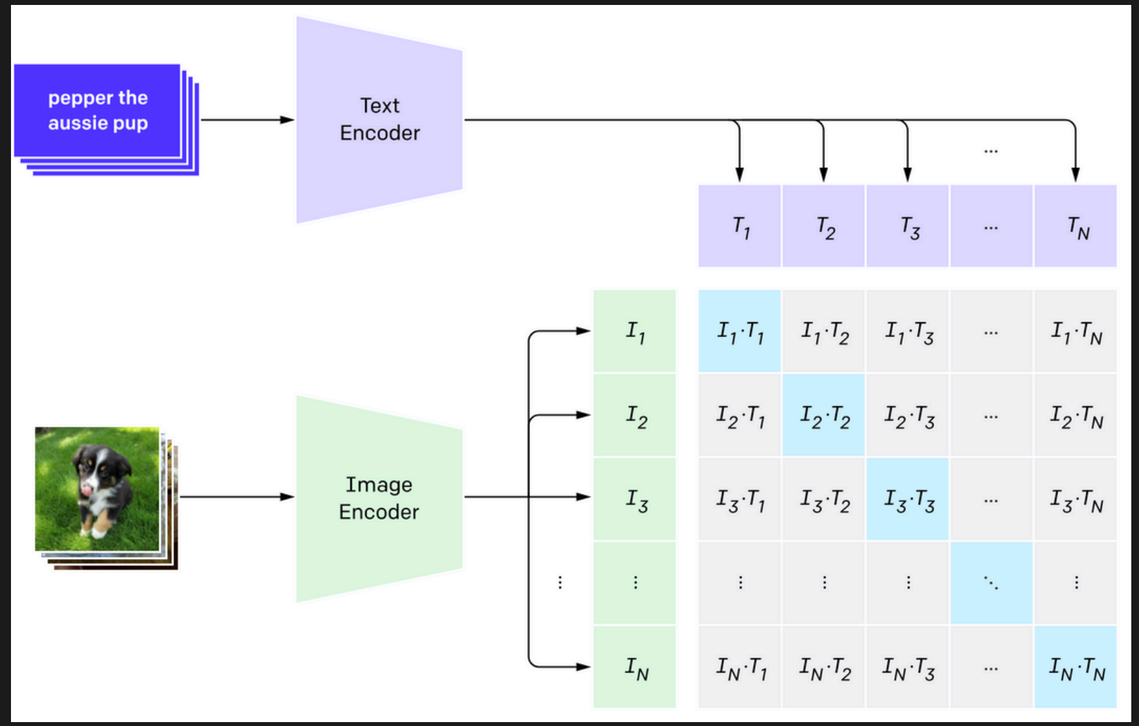
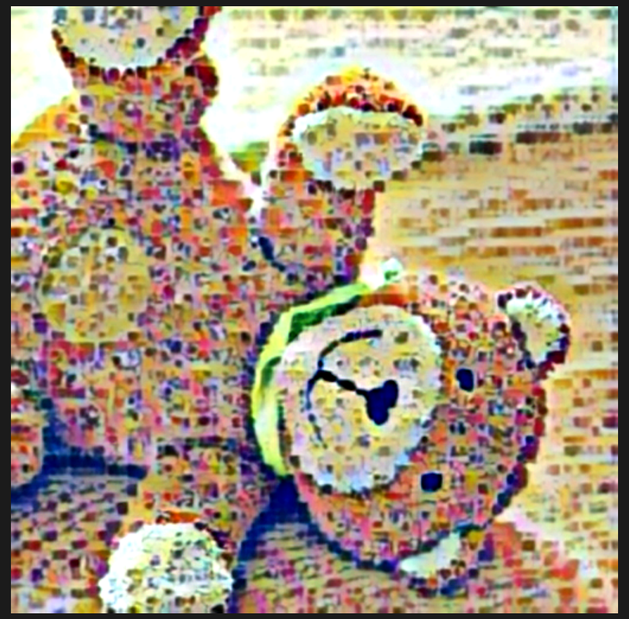


Image credits: https://openai.com/index/clip

“abstract art with dots”



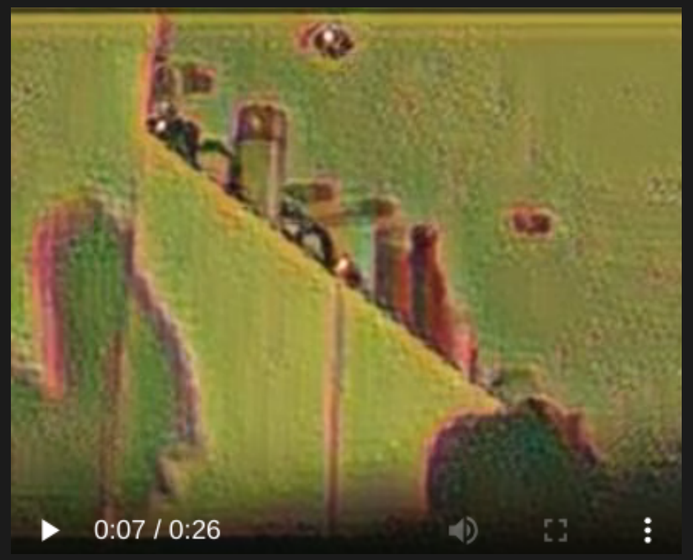
“realism with dark colors”

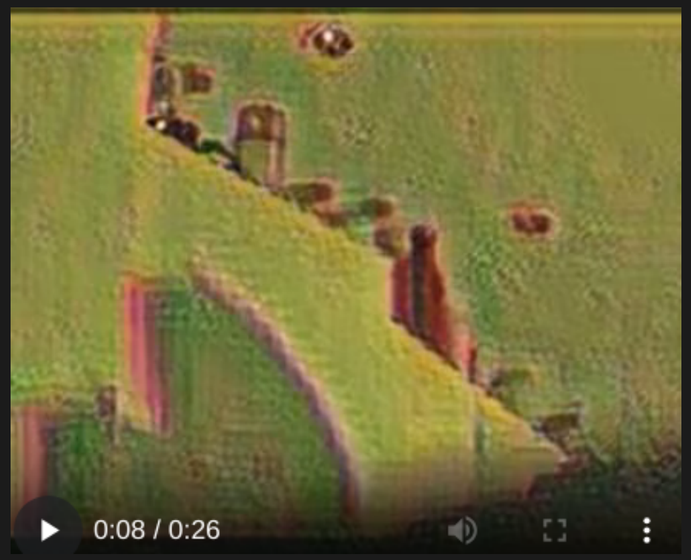
content

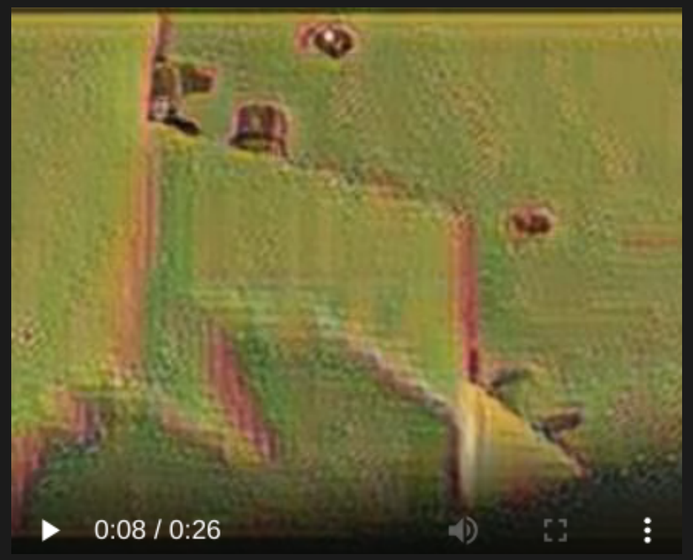
STYLE TRANSFER FOR VIDEOS

Split video into frames, applied style transfer to each individual frame, stitched video back together

Noticed that consecutive frames have differences in textures in the background if the subject of the video moves slightly, especially for pointilism or mosaic styles







STYLE TRANSFER FOR VIDEOS

If a real artist had made this, the still background would not change as much when the subject of the shot changes.

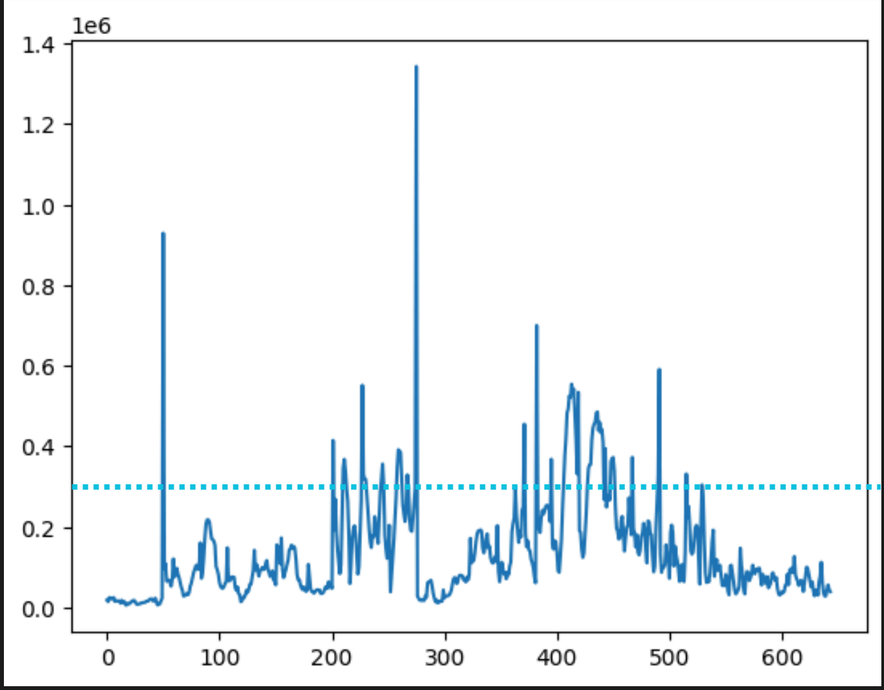
To make it believable, consecutive frames need to maintain some kind of consistency, which is not directly possible with the transformer model, as for the model, the two frames are completely different pictures.

The pattern of the subject changing is okay for now, but the background needs to be smoothened.

STYLE TRANSFER FOR VIDEOS

To try and smoothen this, we calculated the optical flow between consecutive frames of the original video to try and map it to this one.

We take consecutive frames of the style transferred video and try to reconstruct the next frame using the optical flow of the original video.



If there is a shot change, the value of the optical flow is pretty high, and if it goes above a threshold, we use the original image.

The threshold for this particular video is decided as 300000

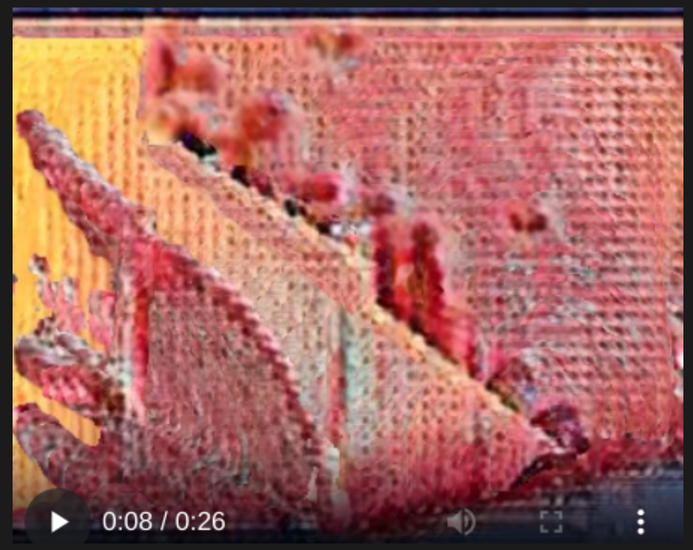
STYLE TRANSFER FOR VIDEOS

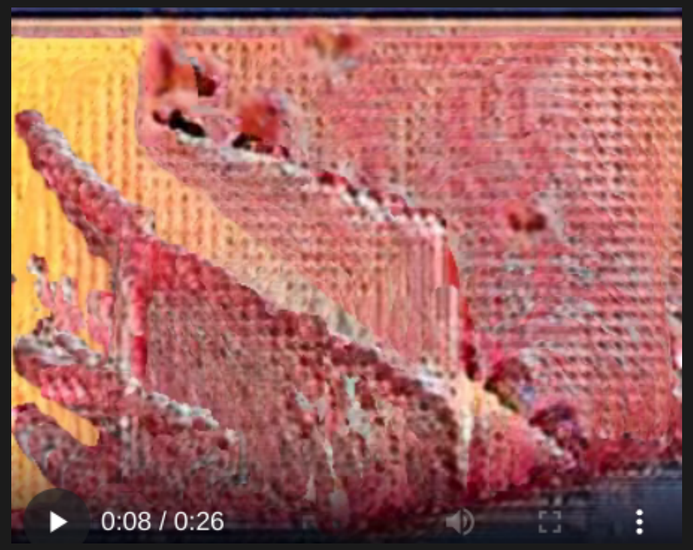
That method was still not perfect, as objects in the foreground move a lot more than the background.

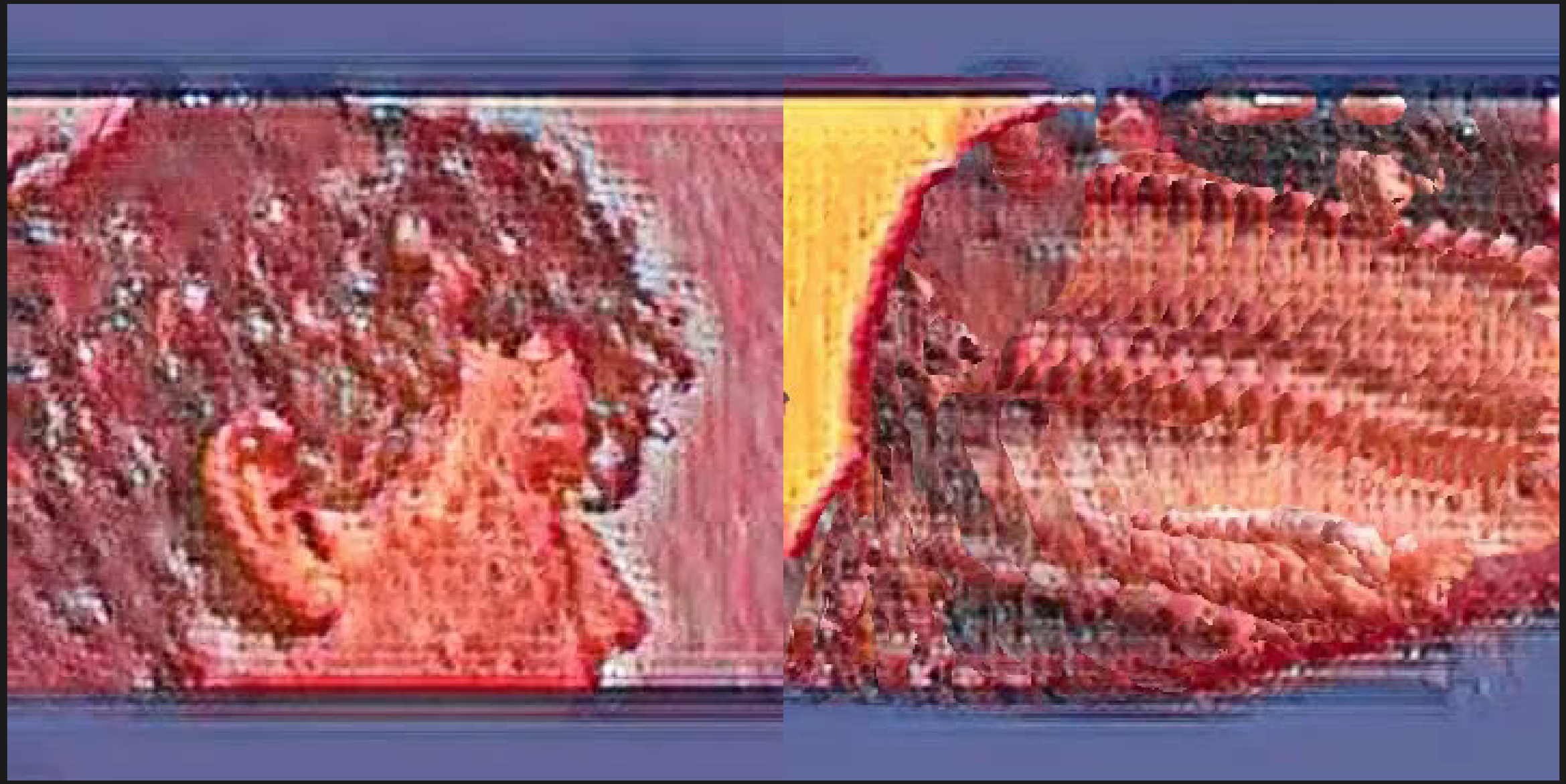
Next, we tried to only use optical flow on pixels where the values of the flow were below a certain threshold. The foreground that moved significantly, could be taken from the style-transferred video, but the background from the old frames + optical flow

This worked better than the last method, but there are still a few problems that arise when the person in the foreground is one block of solid color.

One fix to this could be to segment out the foreground using some other technique.







Without smoothing

With smoothing

REFERENCES

1. Original paper: https://arxiv.org/pdf/2105.14576
2. Original paper github repository: https://github.com/diyiiyiii/StyTR-2
3. Image Style Transfer Using Convolutional Neural Networks: https://github.com/superb20/Image-Style-Transfer-Using-Convolutional-Neural-Networks?tab=readme-ov-file
4. Fast Style Transfer (CNN): https://github.com/lengstrom/fast-style-transfer
5. CLIP: https://github.com/openai/CLIP
6. YouTube video: https://youtu.be/51buCKJZPG4?si=Y31BXtotJB5J-Lqr
7. WikiArt: https://huggingface.co/datasets/huggan/wikiart/viewer

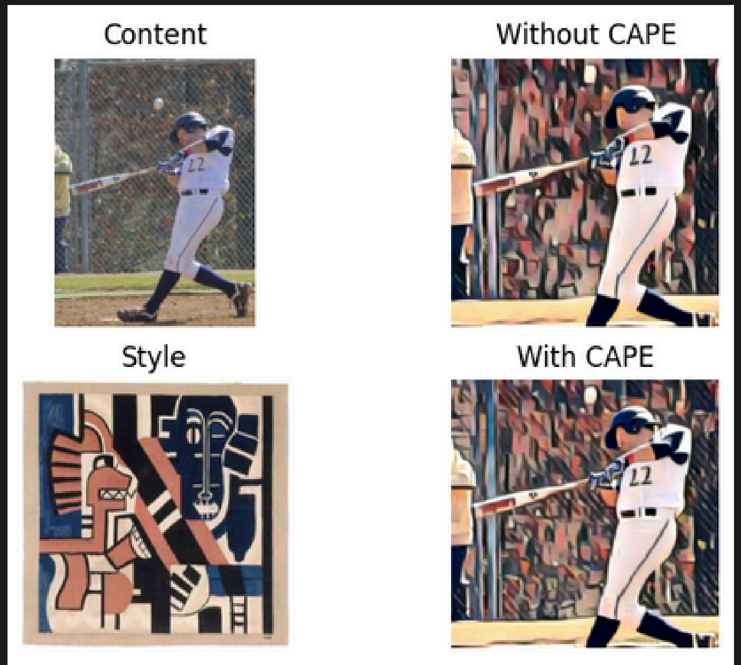
APPENDIX

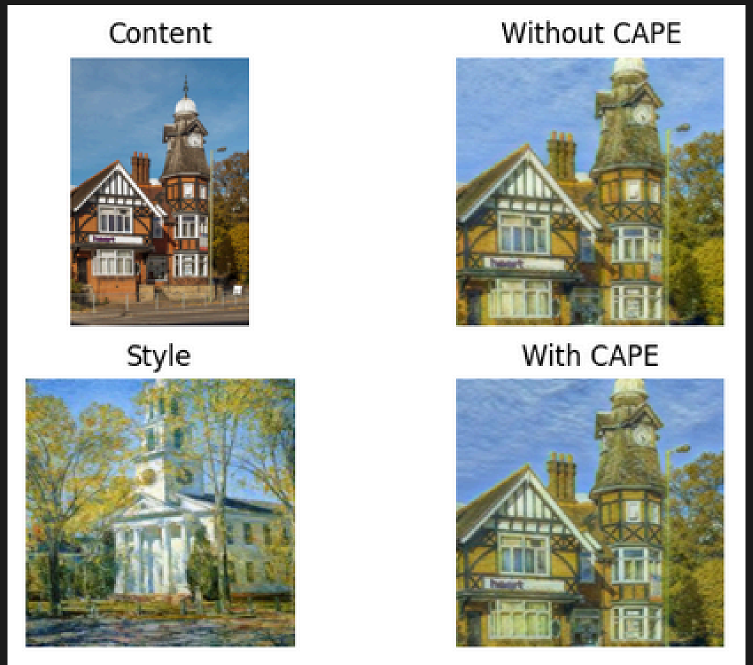
OUR CODEBASE

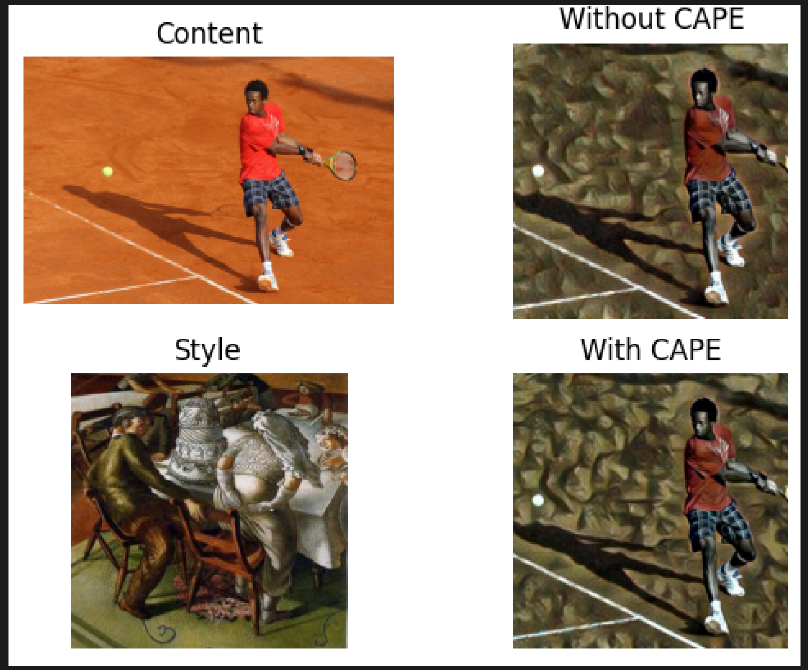
Github Link

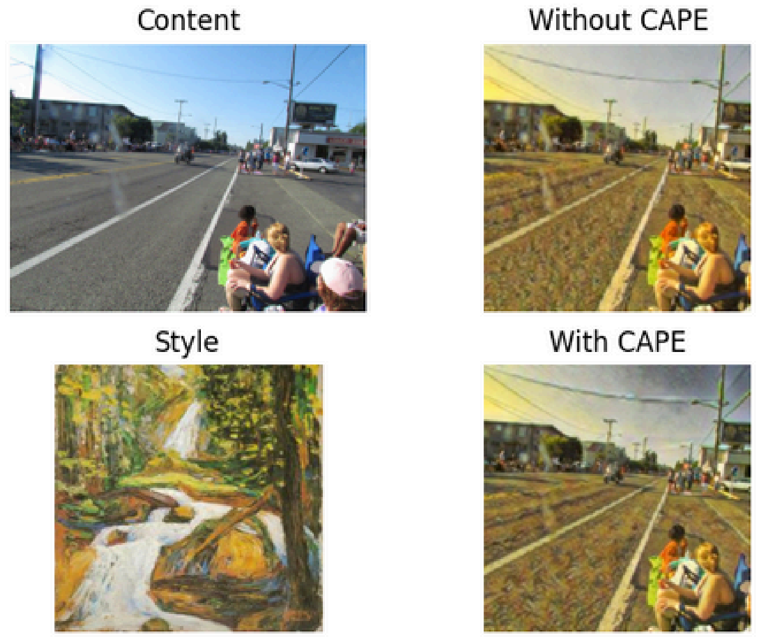
```
├──── README.md
├──── transformer files
├──── clip
|     ├──── finetuneclip.py (finetune clip)
|     ├──── metadata.json (metadata of all the wikiart images)
|     ├──── styles.txt (different styles)
|     ├──── styles_desc.txt (description of styles)
├──── content (content images)
├──── style (style images)
├──── output (output images)
├──── videos
|     ├──── video_Stabilisztion.ipynb (python notebook with all the relevant code)
|     ├──── video files
```

MORE RESULTS









STYLE TRANSFER FOR VIDEOS

Link to all videos

pointilism_smoothened.mp4	Video with red background, smoothened with optical flow
style_transferred_pointilism.mp4	Just plain style transform, no smoothing (red style)
threshold_for_shot_change.mp4	Video with just a threshold for shot change, no background separation
style_transfered.mp4	Just plain style transform, no smoothing (yellow style)
without_threshold_optical_flow.mp4	Video without a threshold for shot change, it warps eventually
original_video_(no_style_transfer).mp4	Original video used
smoothened_video_with_yellow_backgroun...	Video with yellow background, smoothened with optical flow