HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Report

Course: Computer Vision
Topic: Style Transfer

Instructors: Prof.Dinh Viet Sang

Group 11: Nguyen Ngoc Khanh 20204915

Bui Hong Nhat 20204890

Nguyen Van Hung 20204913

Nguyen Xuan Nam 20200422

Dinh Ngoc Hanh Trang 20204928

Hà Nội - 2023

Contents

Source code: https://github.com/khanhnn20042002/cv20222

1. Introduction

Image style transfer is a technique in computer vision that allows us to recompose the content of an image in the style of another. This is done by using a convolutional neural network to extract the content and style features from two images, and then blending them together to create a new image that has the content of the first image and the style of the second image.

In this project, our objective is to try to modify the loss function in the paper A Neural Algorithm of Artistic Style[1] to control the smoothness of the output image. We also use the method proposed in the paper Perceptual Losses for Real-Time Style Transfer and Super-Resolution[2] to build a simple web application that is able to transfer an input image to several styles in a short time

2. Related works

2.1. A Neural Algorithm of Artistic Style

Convolutional neural networks (CNNs) are a type of deep learning algorithm that are commonly used for image recognition tasks. CNNs are able to learn features from images in a hierarchical manner, starting with simple features at the early layers and progressing to more complex features at the later layers. This allows CNNs to achieve near-human performance on object and face recognition tasks. In 2015, Leon A. Gatys and fellows published a paper named A Neural Algorithm of Artistic Style that proposed a method that apply CNN to automatically transfer the style of an image. The key finding of the paper is that the representations of content and style in the CNN are separable

The idea of the method is to use a pretrained CNN to extract content representation of the content image and style representation of the style image.

Higher level layers in CNN produce feature maps which capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction. Therefore, the author chooses the feature reponses in higher layers of the network as the content representations.

To obtain a representation of the style of an input image,the author use a feature space originally designed to capture texture information. This feature space is built on top of the filter responses in each layer of the network. It consists of the correlations between the different filter responses over the spatial extent of the feature maps. The feature spaces built on different layers of the CNN produces texture information of different complexity. All of them form style representation of the image.

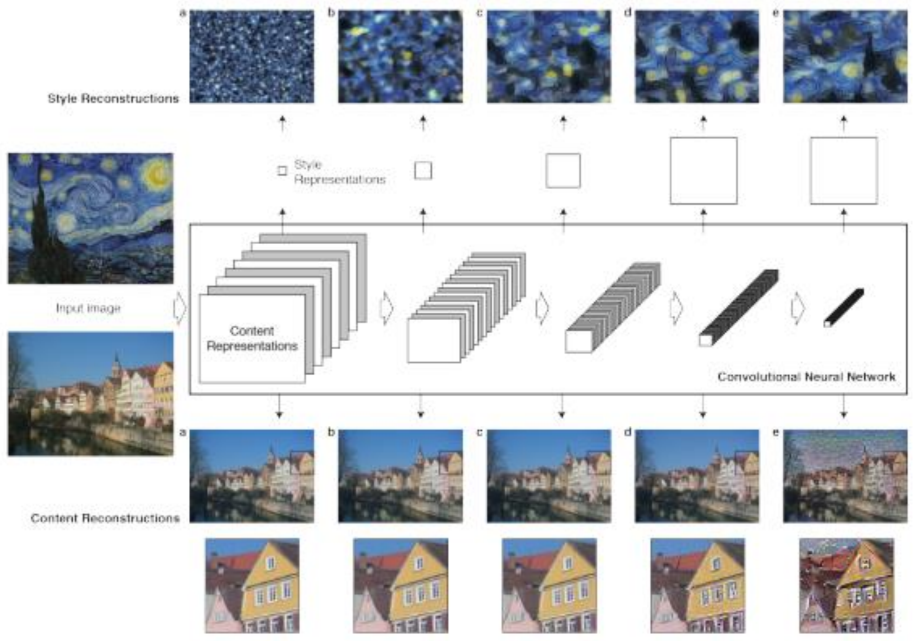


Figure: **Convolutional Neural Network** (CNN). A given input image is represented as a set of filtered images at each processing stage in the CNN. While the number of different filters increases along the processing hierarchy, the size of the filtered images is reduced by some downsampling mechanism (e.g. max-pooling) leading to a decrease in the total number of units per layer of the network. **Content Reconstructions**. We can visualise the information at different processing stages in the CNN by reconstructing the input image from only knowing the network's responses in a particular layer.**Style Reconstructions.** On top of the original CNN representations, we built a new feature space that captures the style of an input image. The style representation computes correlations between the different features in different layers of the CNN.

Generally, each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer in the network. Hence a given input image $\overline{x}$ is encoded in each layer of the CNN by the filter responses to that image. A layer with N_1 distinct filters has N_1 feature maps each size ze M_1 , where M_1 is the height times the width of the feature map. So the responses in a layer l can be stored in a matrix $F^l \in R^{N_i \cdot M_1 j}$ where F^l_{ij} is the activation of the i th filter at position j in layer l. To visualise the image information that is encoded at different layers of the hierarchy we perform gradient descent on a white noise image to find another image that matches the feature responses of the original image. So let $\overline{p}$ and $\overline{x}$ be the original image and the image that is generated and P^i and F^j be their respective feature representation in layer l. We then define the squared-error loss between the two feature representationss

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F^l_{ij} - P^l_{ij}\right)^2 \ .$$

The derivative of this loss with respect to the activations in layer l equals

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} \left(F^l - P^l\right)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0\,. \end{cases}$$

On top of the CNN responses in each layer of the network, we built a style representation that computes the correlations between the different filter responses, where the expectation is taken over the spatial extend of the input image. These feature correlations are given by the Gram matrix $G^l \in R^{N_l \cdot M_l}$, where G_{ij}^l is the inner product between the vectorised feature map i and j in layer l:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

The contribution of that layer to the total loss is then

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

and the total loss is

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

. The derivative of E_l with respect to the activations in layer l can be computed analytically:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left(\left(F^l\right)^{\mathrm{T}} \left(G^l - A^l\right)\right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0\,. \end{cases}$$

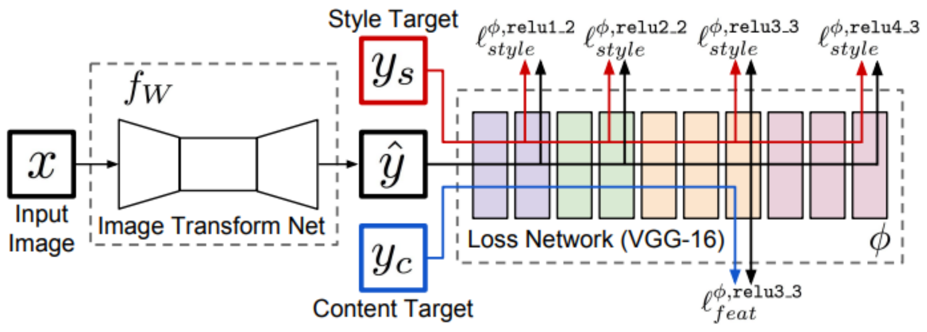
To generate images that mix the content of a photograph with the style of a painting we jointly minimise the distance of a white noise image from the content representation of the photograph in one layer of the network and the style representation of the painting in a number of layers of the CNN. So let $\overline{p}$ be the photograph and $\overline{a}$ be the artwork. The loss function we minimise is

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

where α and β are the weighting factors for content and style reconstruction respectively.

2.2. Perceptual Losses for Real-Time Style Transfer

2.2.1 System Overview


Style Target
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu4\_3}$
y_s
f_W
x
$\hat{y}$
Input Image
Image Transform Net
y_c
Loss Network (VGG-16)
ϕ
Content Target
$\ell_{feat}^{\phi,relu3\_3}$

We train an image transformation network to transform input images into output images. Then, a loss network pretrained for image classification is used to define perceptual loss functions that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.

2.2.2 Image Transformation Networks

The image transformation networks do not have any pooling layers, instead using strided and fractionally strided convolutions for in-network downsampling and upsampling. The network body consists of five residual blocks. All non-residual convolutional layers are followed by spatial batch normalization and ReLU nonlinearities except for the output layer, which instead uses a scaled tanh to ensure that the output image has pixels in the range [0, 255]. Other than the first and last layers which use 9 × 9 kernels, all convolutional layers use 3 × 3 kernels.

Inputs and Outputs: The input and output are both color images of shape 3 × 256 × 256. Since the image transformation networks are fully convolutional, at test-time they can be applied to images of any resolution.

Downsampling and Upsampling: The networks use two stride-2 convolutions to down sample the input followed by several residual blocks and then two convolutional layers with stride 1/2 to up sample. Although the input and output have the same size, there are several benefits to networks to do this.

The first is computational. With a naive implementation, a 3×3 convolution with C filters on an input of size C × H × W requires $9HWC^2$ multiply-adds, which is the same cost as a 3 × 3 convolution with DC filters on an input of shape DC × H/D × W/D. After downsampling, we can therefore use a larger network for the same computational cost.

The second benefit has to do with effective receptive field sizes. High-quality style transfer requires changing large parts of the image in a coherent way; therefore, it is advantageous for each pixel in the output to have a large effective receptive field in the input. Without downsampling, each additional 3 ×3 convolutional layer increases the effective receptive field size by 2. After downsampling by a factor of D, each 3×3 convolution instead increases effective receptive field size by 2D, giving larger effective receptive fields with the same number of layers.

Residual Connections: are used to train very deep networks for image classification. They make it easy for the network to learn the identify function; this is an appealing property for image transformation networks, since in most cases the output image should share structure with the input image. The body of the network thus consists of several residual blocks, each of which contains two 3 × 3 convolutional layers.

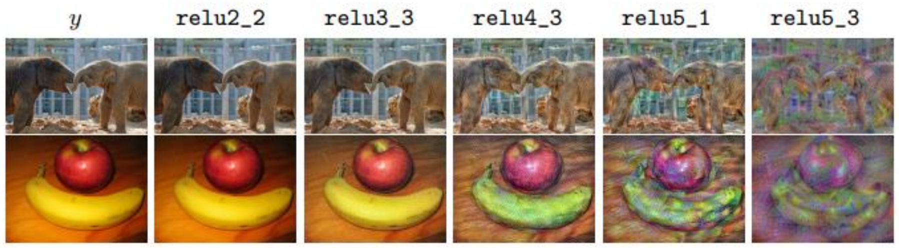
2.2.3. Perceptual Loss Functions

We define two perceptual loss functions that measure high-level perceptual and semantic differences between images. They make use of a loss network φ pre-trained for image classification, meaning that these perceptual loss functions are themselves deep convolutional neural networks

Feature Reconstruction Loss:
The pixels of the output image $\hat{y} = f_W(x)$ will have similar feature representations as computed by the loss network φ instead of exactly matching the pixels of the target image y. Let $\varphi_j(x)$ be the activations of the jth layer of the network φ when processing the image x; if j is a convolutional layer then $\varphi_j(x)$ will be a feature map of shape $C_j \times H_j \times W_j$. The feature reconstruction loss is the (squared, normalized) Euclidean distance between feature representations:

$$\ell_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

Finding an image ŷ that minimizes the feature reconstruction loss for early layers tends to produce images that are visually indistinguishable from y. As we reconstruct from higher layers, image content and overall spatial structure are preserved but color, texture, and exact shape are not.



Style Reconstruction Loss:
The feature reconstruction loss penalizes the output image ŷ when it deviates in content from the target y. We also wish to penalize differences in style: colors, textures, common patterns, etc.
Define the Gram matrix $G_j^{\varphi}(x)$ to be the $C_j \times C_j$ matrix whose elements are given by

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}.$$

If we interpret $\varphi_j(x)$ as giving C_j -dimensional features for each point on a $H_j \times W_j$ grid, then $G_j^{\varphi}(x)$ is proportional to the uncentered covariance of the C_j -dimensional features, treating each grid location as an independent sample. It thus captures information about which features tend to activate together. The Gram matrix can be computed efficiently by reshaping $\varphi_j(x)$ into a matrix ψ of shape $C_j \times H_jW_j$;
then $G_j^{\varphi}(x) = \psi\psi^T / C_jH_jW_j$.

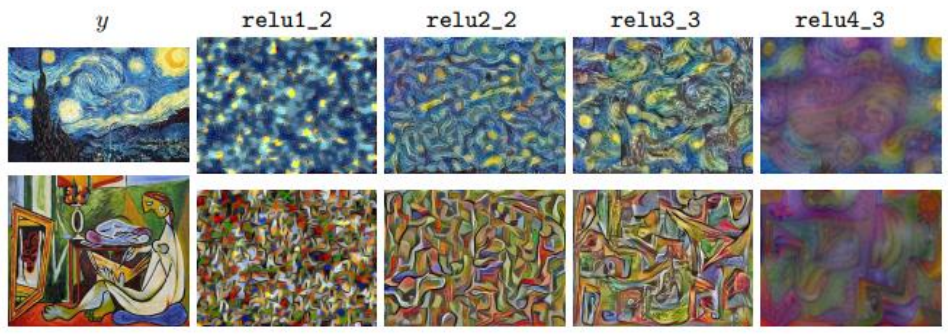
The style reconstruction loss is then the squared Frobenius norm of the difference between the Gram matrices of the output and target images:

$$\ell_{style}^{\phi,j}(\hat{y}, y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2.$$

The style reconstruction loss is well-defined even when ŷ and y have different sizes, since their Gram matrices will both have the same shape.

Generating an image ŷ that minimizes the style reconstruction loss preserves stylistic features from the target image, but does not preserve its spatial structure. Reconstructing from higher layers transfers larger-scale structure from the target image.

Perceptual Losses for Real-Time Style Transfer and Super-Resolution



To perform style reconstruction from a set of layers J rather than a single layer j, we define $\ell_{style}^{\varphi,j}$(ŷ, y) to be the sum of losses for each layer j ∈ J

Final Loss:

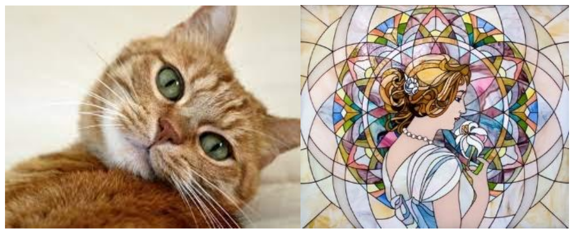
As a baseline, we reimplement the method of Gatys et al - "A neural algorithm of artistic style". Given style and content targets y_s and y_c and layers j and J at which to perform feature and style reconstruction, an image ŷ is generated by solving the problem

$$\hat{y} = \arg\min_{y} \lambda_c \ell_{feat}^{\phi,j}(y, y_c) + \lambda_s \ell_{style}^{\phi,J}(y, y_s) + \lambda_{TV} \ell_{TV}(y)$$

where λ_c, λ_s, and λ_{TV} are scalars, ℓ_{TV}(y) is total variation regularizer.

3. Experiment

The two images below are used to describe the results of our experiment.



Content Style

3.1. A Neural Algorithm of Artistic Style

Training details:

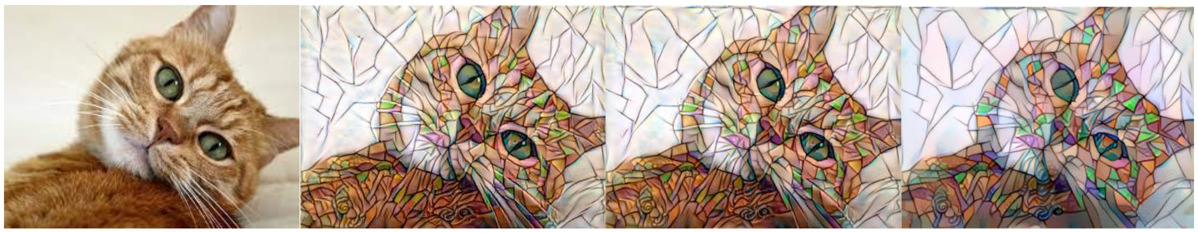
- $\alpha = 1, \beta = 10^3$
- Adam optimizer: learning rate = 0.01
- Iterations = 1000
- Use the content image instead of the white noise image as in the paper
- Network: VGG19 pretrained on the ImageNet dataset
- Content representation layer: conv4_2
- Content layer weights = 1
- Style representation layers: conv1_1, conv2_1, conv3_1, conv4_1, conv5_1
- Style layer weights = 0.2, 0.2, 0.2, 0.2, 0.2

We also add a total variation(tv) loss component to the loss function of the model and do an experiment to see the effect of it.

Given an 2D image y, total variation(tv) loss of that image is defined:

$$l_{tv} = \Sigma_{i,j}|y_{i+1,j} - y_{i,j}| + |y_{i,j+1} - y_{i,j}|$$

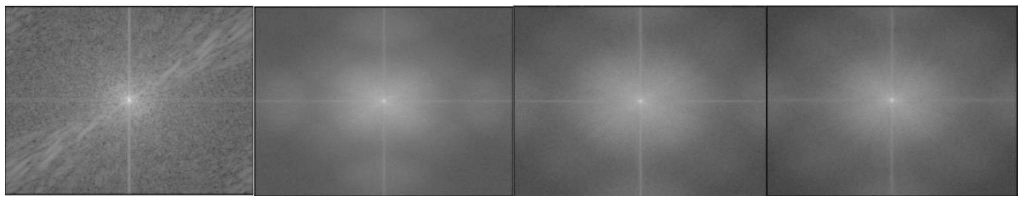
And we weight l_{tv} in the loss by a hyperparameter γ



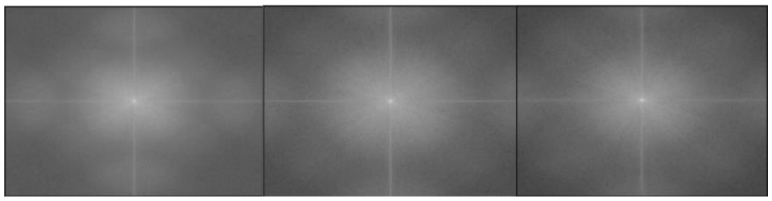
Content $\gamma = 0$ $\gamma = 10$ $\gamma = 100$

We can see that high value of γ reduces noise so that it gives the smoother image

We use Fourier transform of each of the image above to get theirs amplitude spectrum. From that, we see clearer the effect of γ



Content $\gamma = 0$ $\gamma = 10$ $\gamma = 100$



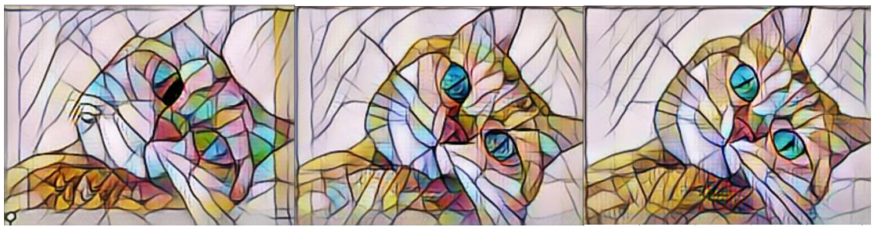
The high value of γ gives the darker value of high frequency position

3.2. Perceptual Losses for Real-Time Style Transfer

Training details:

- Dataset: Microsoft COCO training set which contains 82,783 training images
- Preprocess: before being fetched to the network, images are resized into 256x256x3
- Adam optimizer: learning rate = 1e-3
- Batch size = 8
- Epochs = 2
- $\lambda_c = 2$
- $\lambda_s = 40$
- $\lambda_{TV} = 200$
- Loss network: VGG19 pretrained on the ImageNet dataset
- Content representation layer: conv4_2
- Content layer weights = 1
- Style representation layers: conv1_1, conv2_1, conv3_1, conv4_1, conv5_1
- Style layer weights = 0.2, 0.2, 0.2, 0.2, 0.2

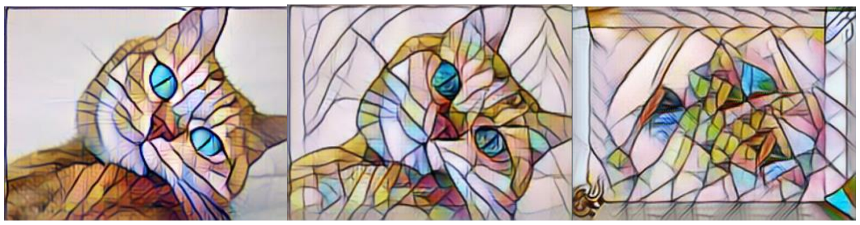
We try some different values of the weight of content loss λ_c



$\lambda_c = 0.5$ $\lambda_c = 1$ $\lambda_c = 2$

The higher the λ_c, the clearer the cat

We try some different values of the weight of style loss λ_s

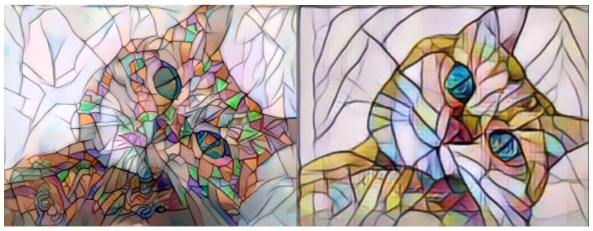


$\lambda_s = 4$ $\lambda_s = 40$ $\lambda_s = 400$

When the value is too small ($\lambda_s = 4$), there is a small difference between the input image and the output image. When the value is too big ($\lambda_s = 400$), we cannot see the cat anywhere in the output image

3.3. Comparison

We call the resulting model of part 3.1 is Vanilla Style Transfer and the resulting model of part 3.2 is Fast Style Transfer



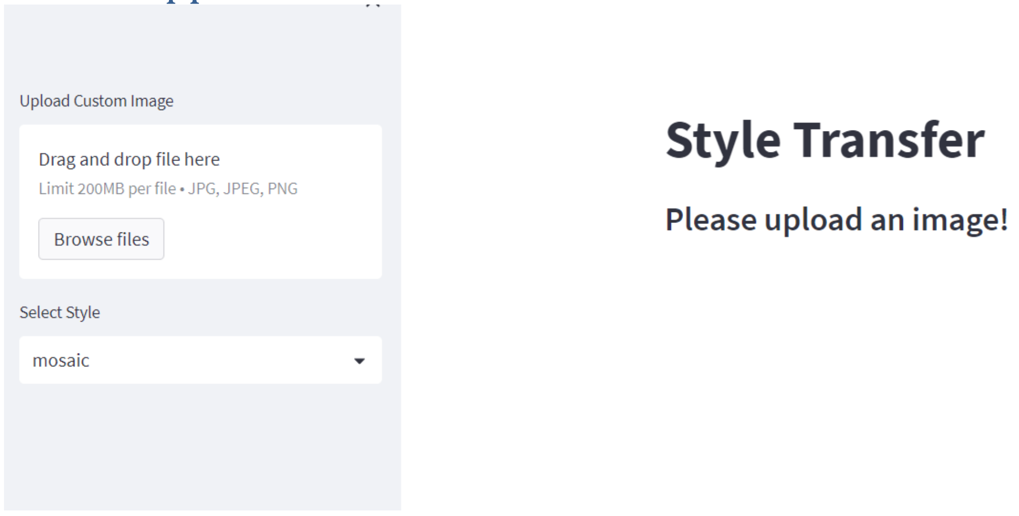
Vanilla Style Tranfer Fast Style Tranfer

Model	**Time (seconds)**
Vanilla Style Transfer	49.43
Fast Style Transfer	0.04

It is clear that one forward pass through the Fast Style Transfer is much faster than the optimization-based method like Style Transfer Model. The experimental result showing that Fast Style Transfer model is 1000x faster than Vanilla Style Transfer model also confirms that fact.

Which model gives a better style transfer is based on personal perspective. However, in our opinion, our Fast Style Transfer model does a better job in converting an image to another style.

4. Web Application

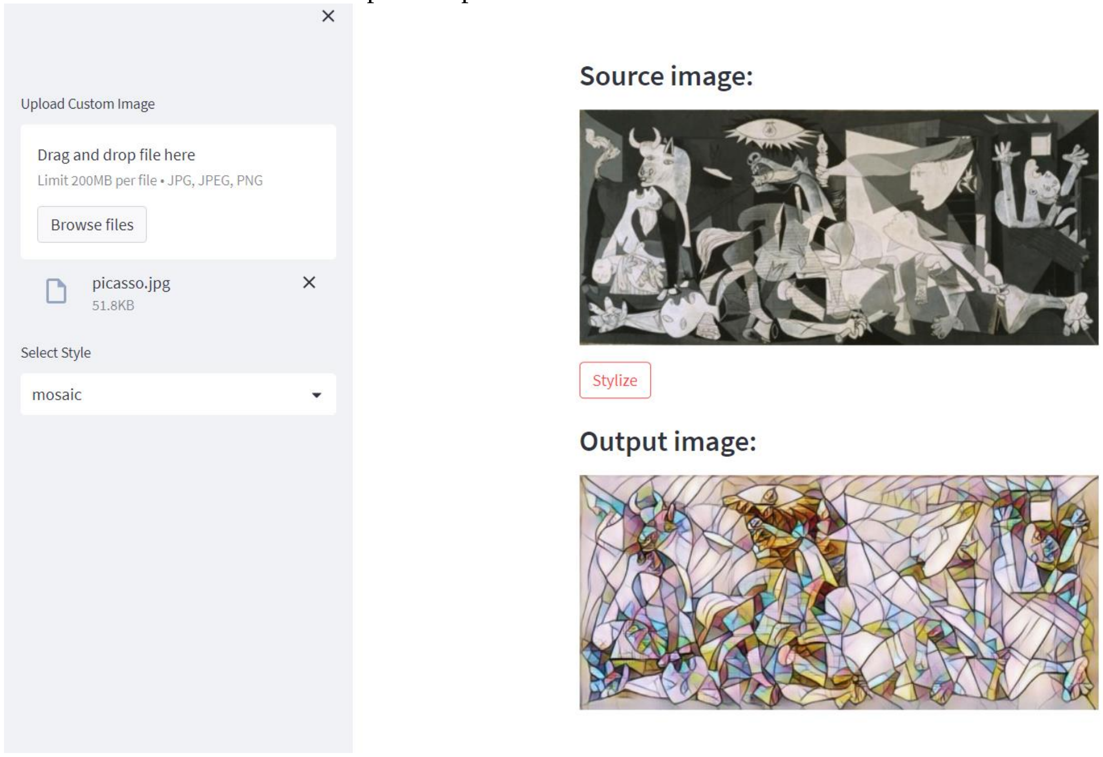


Style Transfer

Please upload an image!

We use Streamlit as a python framework to deploy our project into the internet.
Our interface just like:

We created an area to browse the files and an area to select the style.
There are 4 style to select: mosaic, autumn, udnie and wave. Each style goes with 1 trained model and we use that model to create the output and present outside.



So the link to our deployed project: [Streamlit (cv20222-1-rdopile8jk8.streamlit.app)](cv20222-1-rdopile8jk8.streamlit.app)

Reference

[1] Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576 (2015)
[2] Justin Johnson, Alexandre Alahi, Li Fei-Fei: Perceptual Losses for Real-Time Style Transfer and Super-Resolution. arXiv preprint arXiv:1603.08155 (2016)