CS 194-26: Image Manipulation and Computational Photography, Fall 2022

Poor Man's Augmented Reality and A Neural Algorithm of Artistic Style

Ethan Gnibus

Poor Man's Augmented Reality

Overview

Using a box I drew on and a live video, I attempt to recreate an Augmented Reality scene where I can render using 3 dimensions. Limiting myself to only a box with known 3D coordinates and a video as input, this means I must proceed without knowing intrinsic parameters of the camera.

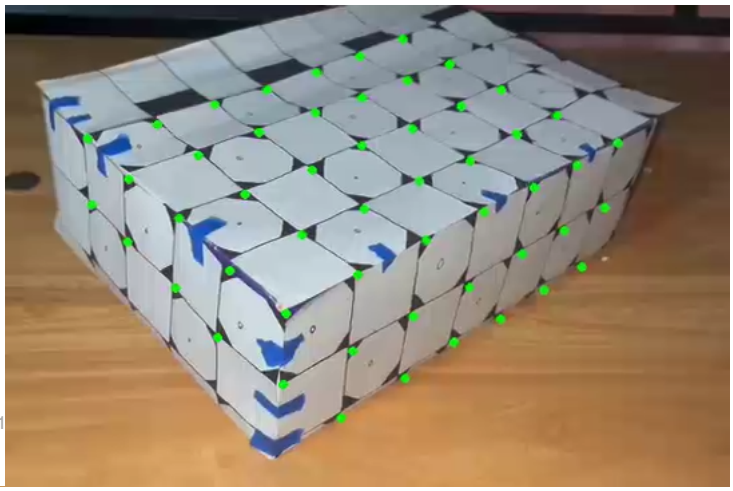
Part 1: Importing a video feed

To complete this project I decided to use cv2's VideoCapture function. This allowed me to read in video data in real-time. For the sake of this proof-of-concept, I chose to use the video clip below.

Recorded on my phone

Part 2: Tracking Points

To make a 3D scene, can correspond 2D (x, y) points in image-space to 3D (x, y, z) points in world-space. The first step in this process is to label 2D (x, y) points in image-space. I used matplotlib.pyplot.ginput() to manually label the 2D (x, y) points in the first frame of the video. To get the points in the rest of the frames, I used off the shelf tracking from cv2! To do this I initialized a separate Median Flow tracker on every point from the first frame (cv2.TrackerMedianFlow_create()) and tracked until the last frame. My results are below:



Tracked Points

Part 3: Finding the Projection Matrices

Using the known 3D (x, y, z) world-space points that correspond to the 2D (x, y) image-space points we just retrieved by tracking, we can recover a transformation matrix for every frame that enables us to project 3D meshes onto our 2D video!
We proceed by solving least squares on our 2D and 3D points based on the following diagram:

Ax=0 form

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

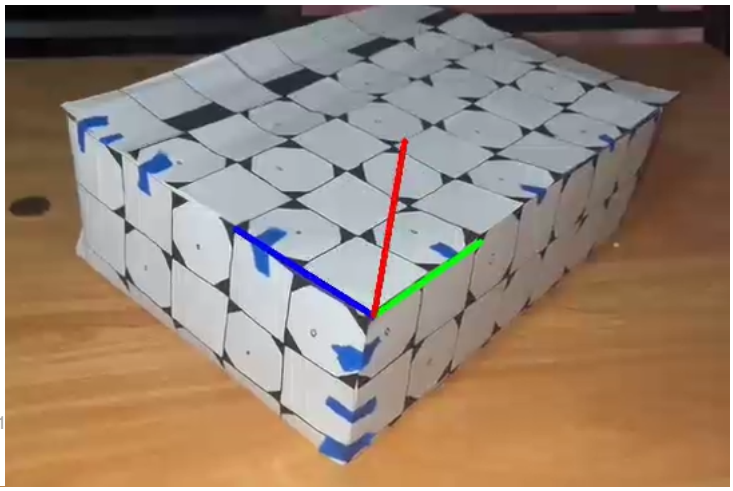
Linear Least Squares for Camera Calibration

Where (X_i, Y_i, Z_i) are the 3D world-space points that correspond to the (u_i, v_i) 2D image-space points. We can reshape the resulting vector to get the following matrix filled with m_i entries. This matrix is the projection matrix that allows us to go from 3D world-space to 2D image-space.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The Projection Matrix

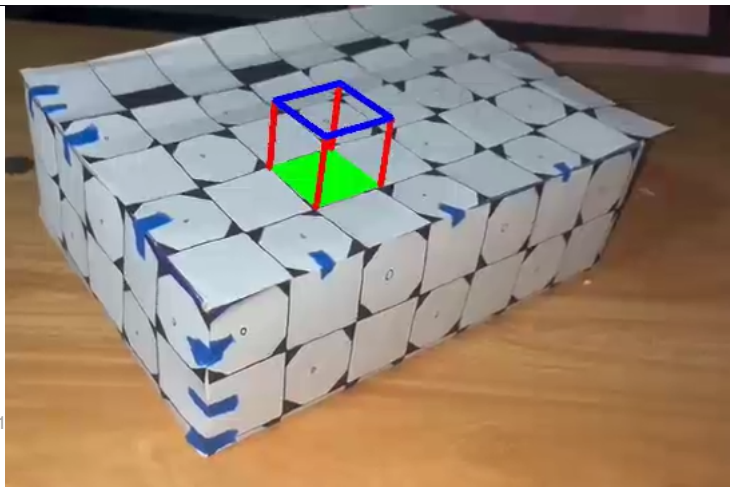
Here I display the 3D world-space's axes by defining points in 3D world-space, projecting them into 2D-image space, then using cv2's line() function to draw to the screen! I repeat this process separately for every frame.



Displaying (x, y, z) world-space axes

Part 4: Displaying the Box

I use the method from above to define a box object in our scene. I use cv2's drawContours function to draw the box's green bottom using trianges.

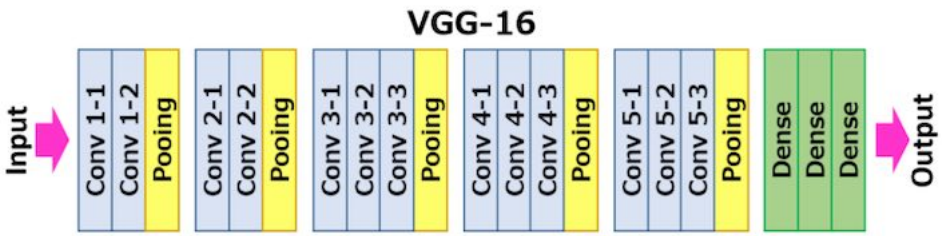


Box Mesh

A Neural Algorithm of Artistic Style

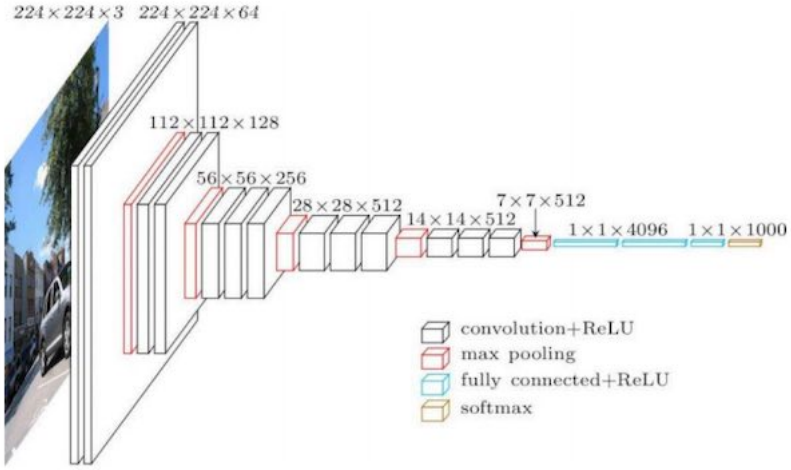
Overview

In this project based on A Neural Algorithm of Artistic Style, I use the content and style features from VGG-Network to transfer the style from one image to another. I take these content and style features from 5 convolutional layers in VGG-Network, then train a separate neural network to balance the loss between an output images content from one image and style from another image. Using the following VGG-Network architecture, A Neural Algorithm of Artistic Style, uses conv1_1, conv2_1, conv3_1, and conv4_1 as style features and conv4_2 as the content feature. A VGG-Network can be seen below:



The Architecture

The architecture depicted below is VGG16.



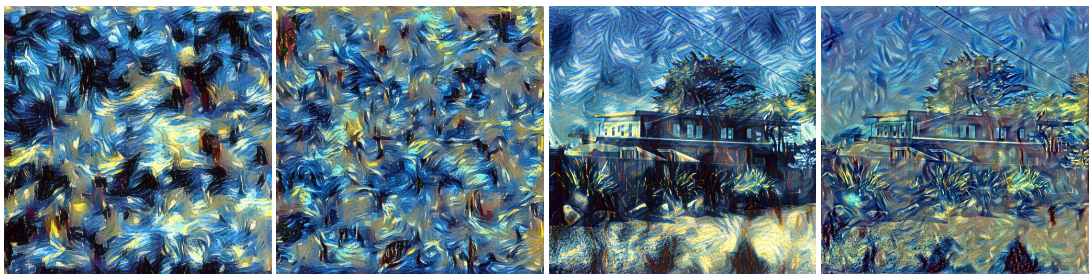
VGG-Network Architecture

Part 1: Building The Network

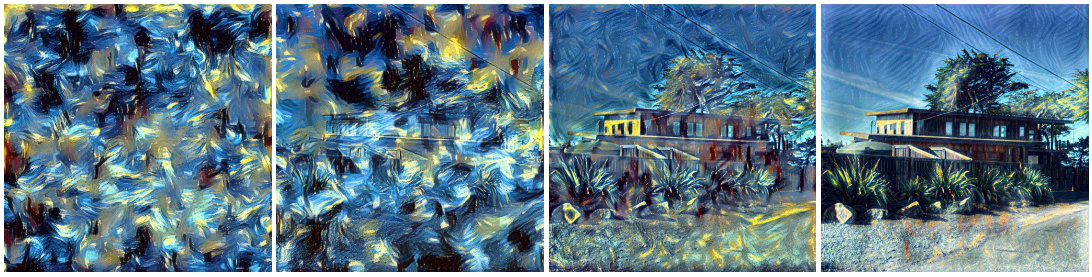
To implement a paper, I trained a neural network to take in two images and output one image. The goal of the style transfer neural network is to minimize the sum of the losses L_{style} and $L_{content}$. L_{style} aims to minimize the mean squared error between the Gram Matrices of the style image and the output image respectively. $L_{content}$ aims to minimize the mean squared error between the content features of the content image and output image respectively. I used a variety of hyperparameters as seen below.

Part 2: Varying Style Ratio and Style Layers

Here I show how adjusting the style to image ratio affects outputs. From left to right I adjust from loss favoring content to loss favoring style. From top to bottom I change which layers the style features come from.



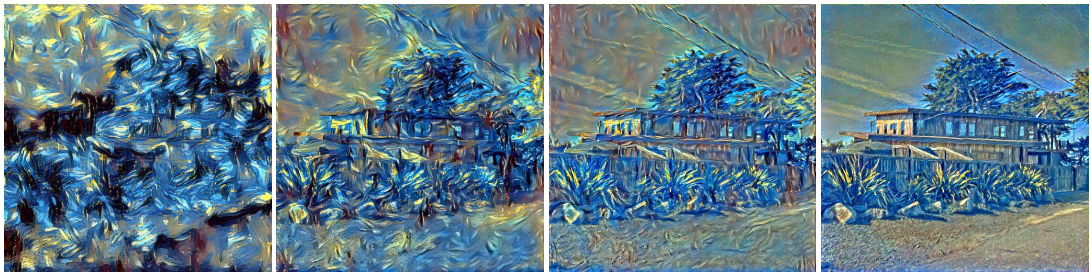
Conv1_1: 10^{-5} | Conv1_1: 10^{-4} | Conv1_1: 10^{-3} | Conv1_1: 10^{-2}



Conv2_1: 10^{-5} | Conv2_1: 10^{-4} | Conv2_1: 10^{-3} | Conv2_1: 10^{-2}

Conv3_1: 10^{-5} | Conv3_1: 10^{-4} | Conv3_1: 10^{-3} | Conv3_1: 10^{-2}

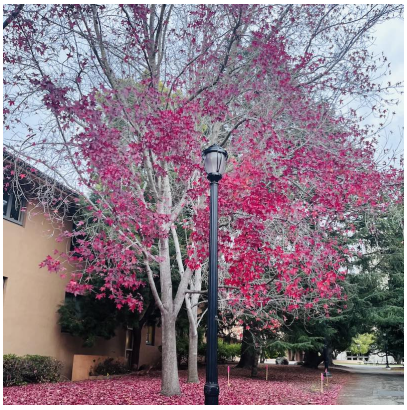
Conv4_1: 10^{-5} | Conv4_1: 10^{-4} | Conv4_1: 10^{-3} | Conv4_1: 10^{-2}



Conv5_1: 10^{-5} | Conv5_1: 10^{-4} | Conv5_1: 10^{-3} | Conv5_1: 10^{-2}

Part 3: Multiple different styles on one image

Here I show this project's capabilities by styling a picture of me eating a muffin on the beach:



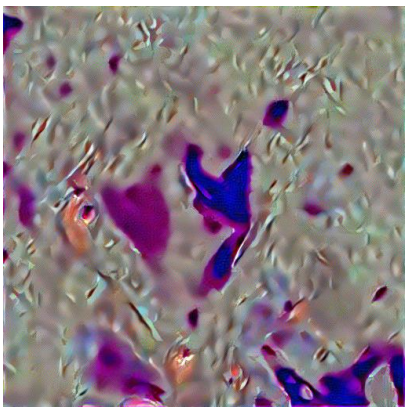
Style

Content

Output

Style

Content



Output


Style


Content


Output


Style

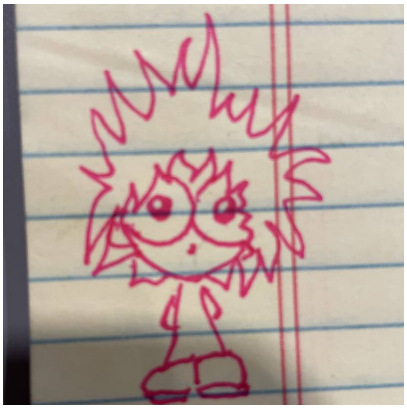

Content


Output

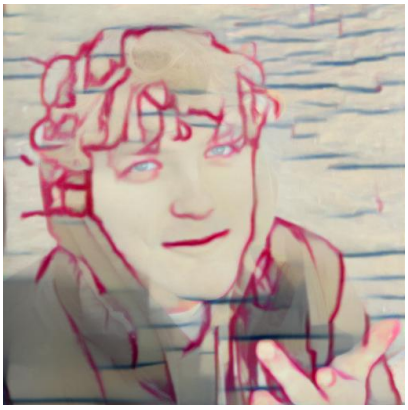

Style


Content


Output


Style


Content


Output

Part 4: Repetitive Styling

Just for fun, I decided to use the output of a style transfer as the style image in the next style transfer. Below are my results:

Style Content Output

Style Content Output

Style Content Output

Style Content Output

Style Content Output

https://inst.eecs.berkeley.edu/~cs194-26/fa22/upload/files/projFinalAssigned/cs194-26-ahn/