COMP 451: Vision with Deep Learning

End Term Project

STYLE TRANSFER

Sitare Arslantürk & Genco Levi

Contents

Introduction

In our project, we created new images by mixing the style of one picture with the content of another picture. To perform this, we used a pre-trained convolutional neural network model called VGG-19. The 19 stands for the 19 layers within the network. We performed gradient descent on the pixel values of the input image to minimize the combined distance measures of the context and the style input simultaneously. We changed the style weights of the neural network layers after 250 iterations but kept the content weight same throughout the computations different from the traditional approach and modified the input image's pixel values as parameters. To transform the inputs, backpropagation to the inputs was performed on the resulting gradients with respect to the distance. We also used average pooling rather than max-pooling because it was used by the authors of the article to do so for its better looking results. Figure 1 shows the overall aim of the project.

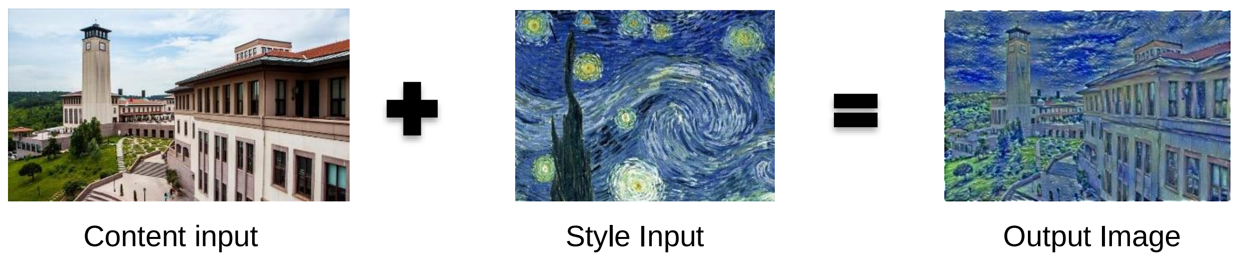


Figure 1

Utilities

For the usage of pre-trained models, torchvision package was imported. The load-image function was defined in order to upload the image we want to transform. This function takes in image path, maximum size and optional shape arguments and outputs image's Pytorch tensor to be used as input for training. In the algorithm we used, the tensor returned is Vector(4)[1, 3, 512, 768] corresponding to the batch dimension, the RGB channels, the height and the width. The function performs transformations on the image by first resizing it, transforming it into Pytorch tensors and then normalizing it. The normalization is based on statistics of the ImageNet dataset and we used them as given.

We also have a function called im_convert in which the opposite of load_image is performed. The image tensor is converted back into a numpy array which we can display. The dimensions are rearranged and the normalization is cancelled. According to Gatsy et al. the input image is transformed into representations that increasingly care about the actual content of the image compared to its detailed pixel values. The style and content of images are captured in deep convolutional neural network's certain layers by feature maps. Thus, these certain layers can be considered as content features of the model.

Layers

To obtain the features of the style and content layers, we used get_features function which takes the image and the model as input arguments and performs a forward pass one layer at a time through the model. The feature map responses are stored if the name of the layer matches the keys of predefined layer dict which serves as a mapping from the VGG-19's layer indices to the layer names. Both content and style layers will be used if no layers are specified. The graphical representation of the layers can be found in Figure 2.

We need to measure the correlation between different feature map responses of a given layer in order to obtain style representations. The gram_matrix function computes the Gram matrix of the vectorised feature map by taking a feature map tensor as input argument. The tensor is reshaped to be one vector and the inner product of the reshaped tensor is calculated.

We resized the style and content image to match with each other, so we will not have to modify dimensions later on. After loading the images, the feature map responses of the layers are computed. The gram matrices are computed for all style layers and kept them in a style_gram dictionary. A third image which is called target is created to serve as the starting point of the image transformation. We start this generation from random noise, which we thought gave better results in terms of style compared with the results of the generation from original content image. So now we have a target tensor image to update and a fixed model.

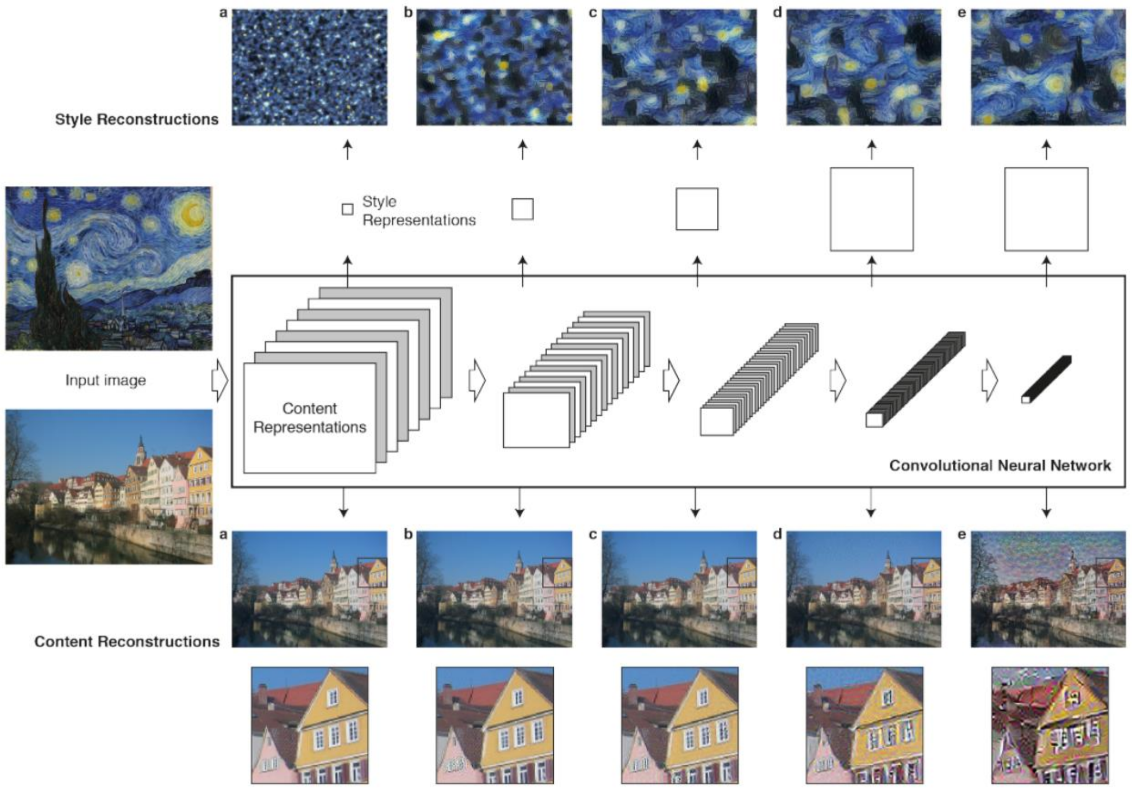


Figure 2: Layers

Loss Function

The loss function depends on feature map responses of certain content and style layers. For each layer, we assign multiplicative weights to tune style artifacts. We start with a larger weight in the earlier layers to produce larger artifacts. We also have weights assigned for the contribution of style and content to the individual losses. The content loss function is calculated via mean squared error loss between two feature map responses of the target and the content image. The style loss is calculated via gram matrices and dividing the mean squared error loss by the total number of elements in the respective future map.

This loss function produces good results, but the output can be too noisy. The solution to this is to add an additional term to the loss function to punish noisy outputs - total variational loss. This loss is calculated by taking the difference between horizontally and vertically adjacent pixels of the output, and taking the L2 norm of this “difference image”. It is similar to edge detection by convolution. This way any unnecessary variation in the image, (i.e. noise) is discouraged. It’s effect together with the other loss terms is like an edge preserving blur. This effect can of the variational loss can observed in Figure 4 and 5 together with the new loss function’s graph in Figure 3.

Figure 3: Loss Function

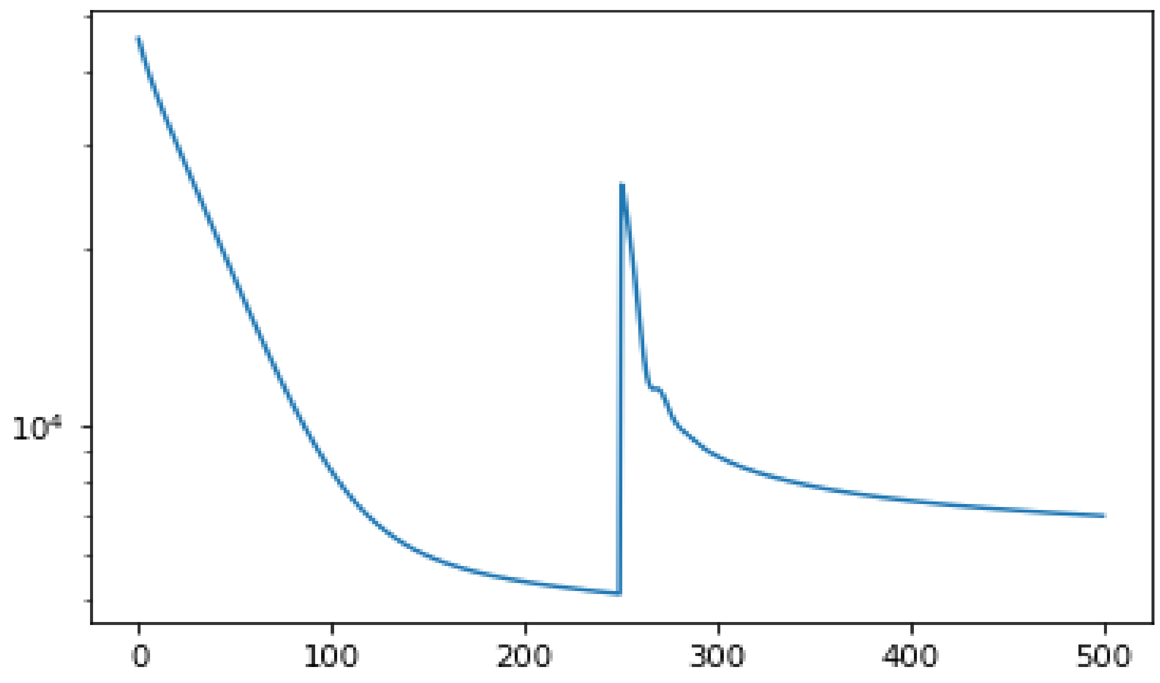


Figure 4: Without variational loss

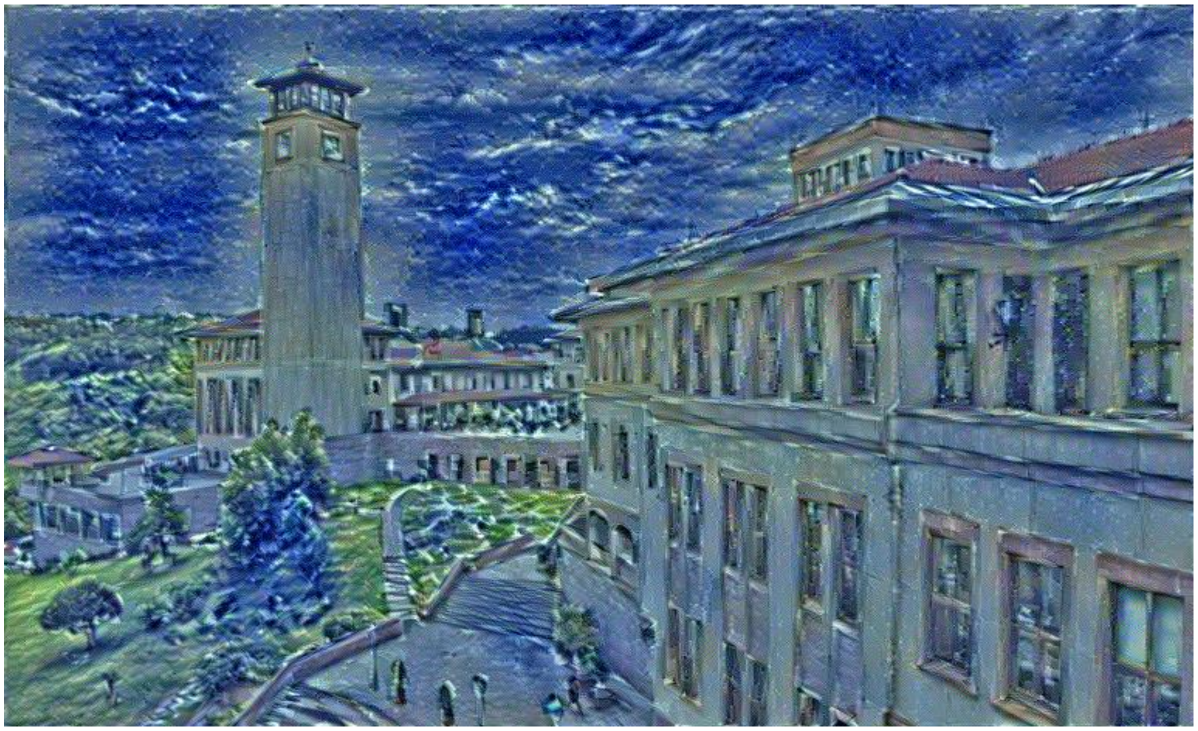
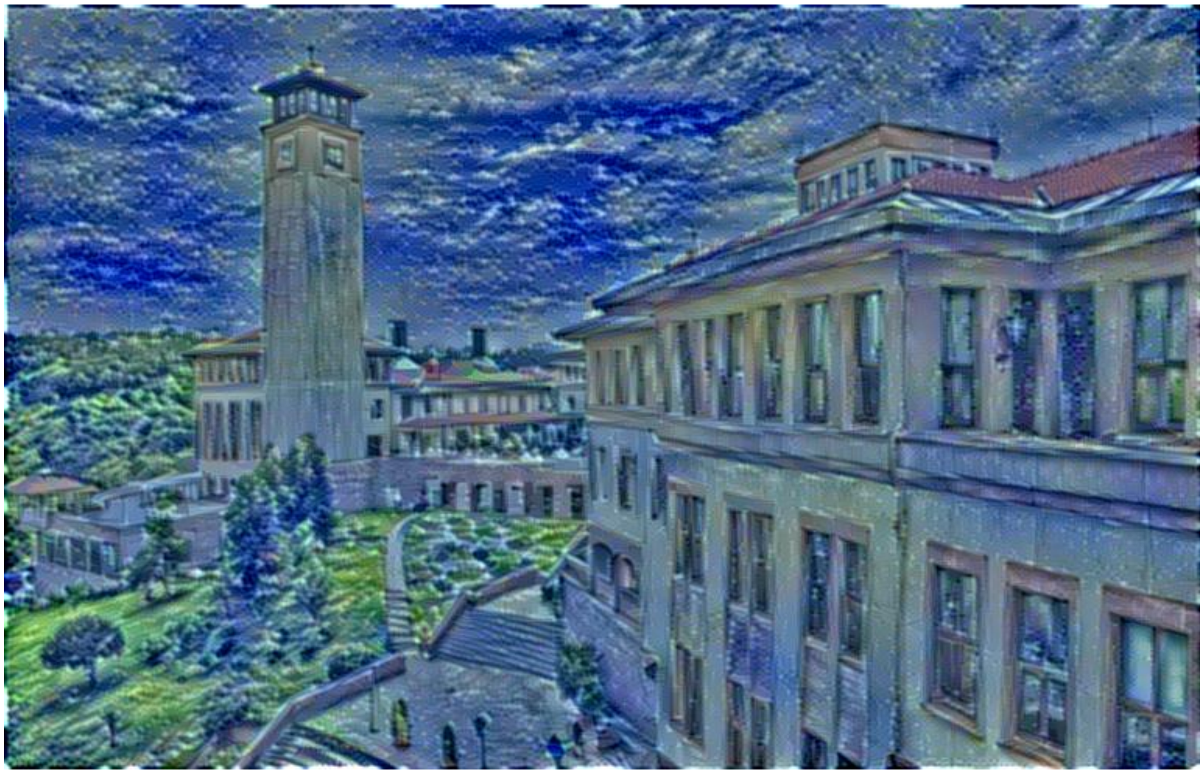


Figure 5: With variational loss



Optimizer

We choose the Adam optimizer as our optimizer and 400 iterations as the number of loops. To track the progress we printed the total loss with the content and style loss composition details and also outputted the image to see the difference as the iteration continues. Inside the loop, the content and style losses are computed and multiplied with their respective weights and added to the total loss. Then the image's pixel values are updated iteratively using backpropagation performed on the total loss.

Results

In the end, the final image is outputted and visualized from the numpy array after the style transfer loop is finished. The output png image is saved in the results file. The different results with different style input with the same content input (Figure 6) are observed in Figures 7, 8, 9 and 10.

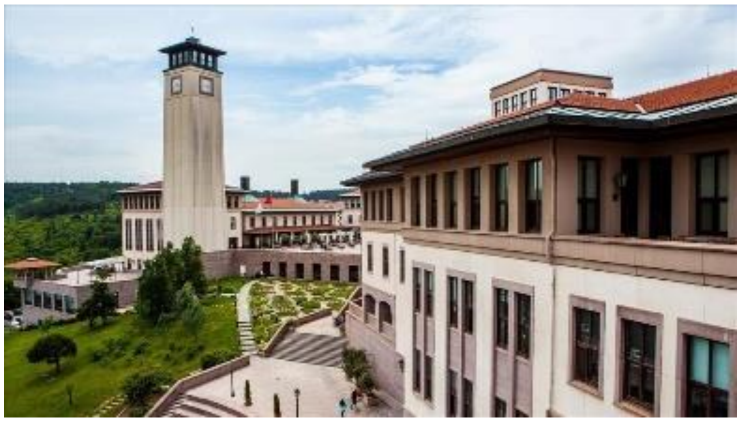


Figure 6: Content Image Input

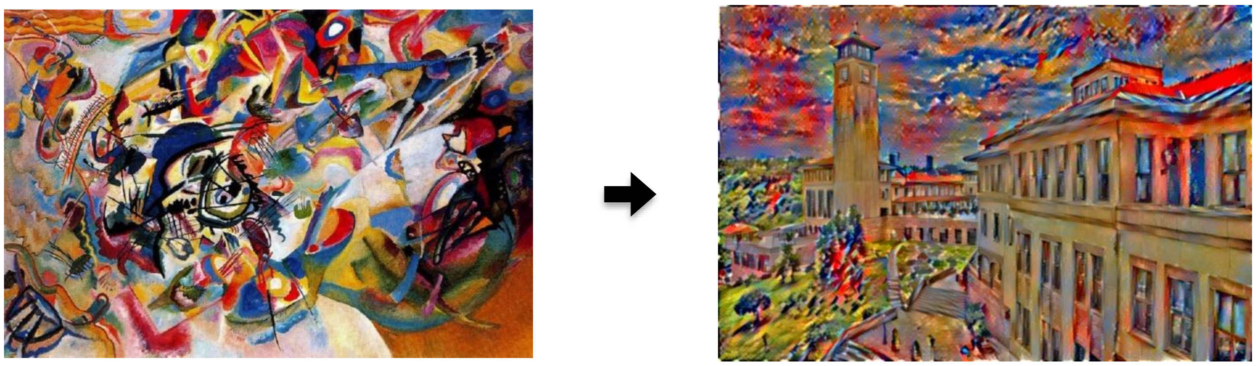


Figure 7: Style Image Input as Kandinsky

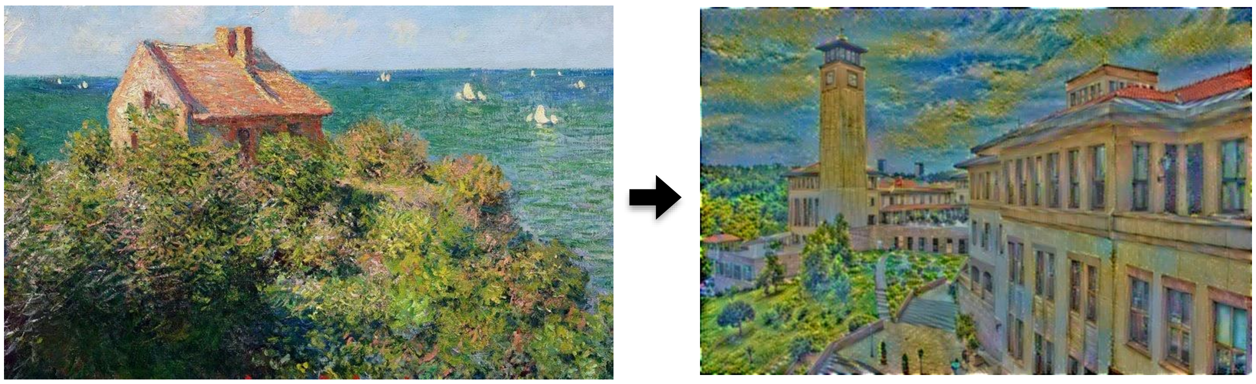


Figure 8: Style Image Input as Monet

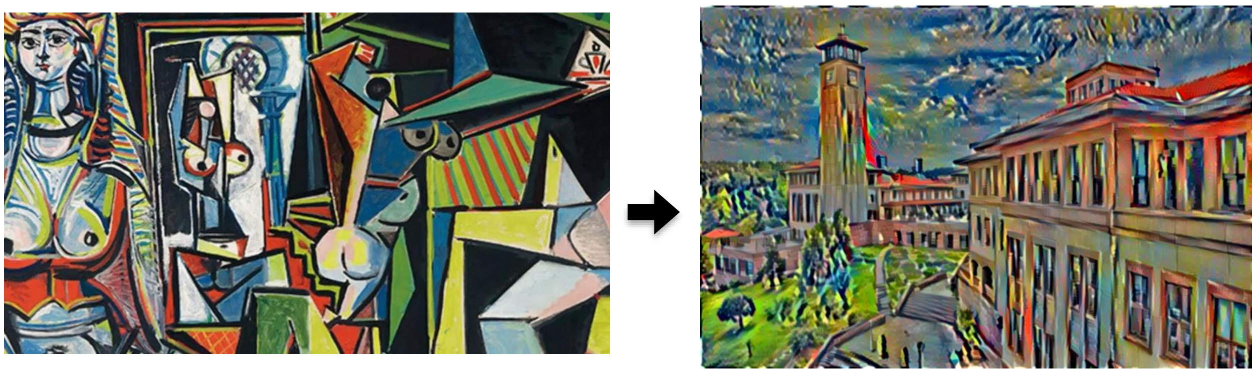


Figure 9: Style Image Input as Picasso

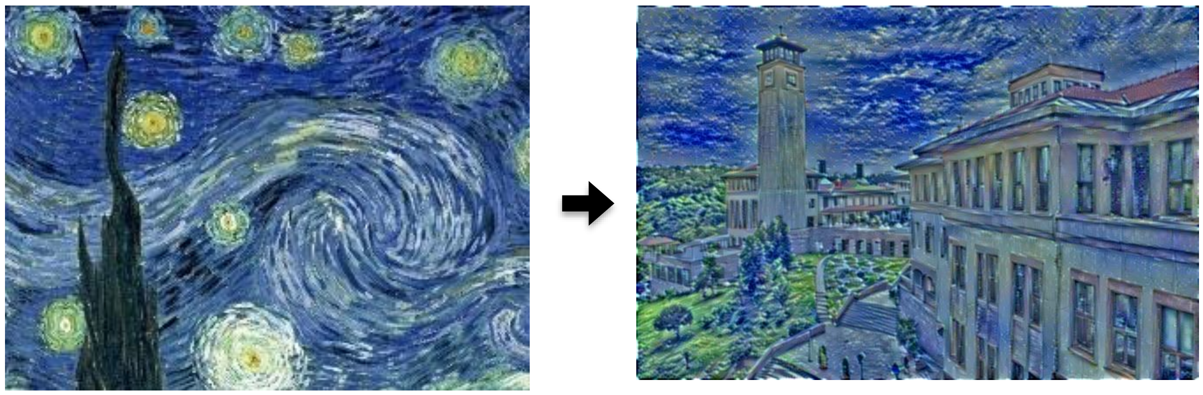


Figure 10: Style Image Input as Van Gogh

References

[1] https://nextjournal.com/gkoehler/pytorch-neural-style-transfer

[2] https://towardsdatascience.com/neural-networks-intuitions-2-dot-product-gram-matrix-and-neural-style-transfer-5d39653e7916

[3]https://www.vincentvangogh.org/starry-night.jsp.

[4]https://www.pablopicasso.org/the-women-of-algiers.jsp

[5]https://www.normandythenandnow.com/claude-monet-by-himself-part-4-the-power-of-friendship/

[6]https://www.overstockart.com/preframed/composition-vii-1913-pre-framed-6