COMS W 4995 006 (Fall 2022)

Deep Learning for Computer Vision

Project Report

Automated Photo Editing using Neural Style Transfer

Participants

Rohit V Gopalakrishnan (rvg2119)

Adithya K Krishna (akk2188)

Tanav Hitendra Shah (ts3469)

Links

GitHub repository containing our working project

Link to the working demo

Presentation Video

Presentation Link

Contributions:

Rohit V Gopalakrishnan (rvg2119)

- Writing sections 1, 5 (excluding 5.2), 6, 7, 8 in the report.
- Automation in selecting style images in the code.
- Integrated everyone's part and compiled the report together.
- Prepared the presentation slides.

Adithya K Krishna (akk2188)

- Writing sections 2,3.1,3.2 and section 4 in the report
- Created dataset used in our implementation

- Developed a web scraping algorithm in python to extract images used by the style transfer model.
- Ideated and Implemented an unsupervised semantic segmentation model using a Convolutional network and using SLIC features
- Incorporated superpixel refinement to improve the segmentation results

Tanav Hitendra Shah (ts3469)

- Writing sections 3.3, 3.4, 5.2, 5.3 in the report
- Setting up the base code from the reference paper for style transfer (DPST) and getting it to work
- Upgrading the code base from Tensorflow 1 to Tensorflow 2.4 and python 3.8
- Thinking about and implementing the novel ideas in the project
- Implementing and integrating the novel loss function, transfer learning procedure, automation in selecting style image in the code
- Generating the demonstration results and compiling the demo notebook

1. Introduction

(Written by Rohit V Gopalakrishnan (rvg2119))
Today, photo editing has become an essential part of photography. With mobile phone photography on the rise, naturally there is a rise in the number of users who use photo editing applications to enhance their images. However it is not as easy to make photos look professional. A beginner to photography simply lacks the experience and knowledge required to make enhancements to photos that would bring out the tone of the picture.

As a part of this project we try to explore image style transfer as a means of coming up with a system that can be used to automatically edit photographs based on the style of pre-existing professional photos. By doing this, we eliminate the burden of choice for novice photographers, who don't understand the correct style in which a particular photo must follow. This is the first attempt to achieve high-quality images through this methodology without the need to use extensive software.

2. Objectives

(Written by Adithya K Krishna (akk2188))
Style transfer consists of recomposing the content and style of two separated entities to create a newly generated output. In recent years, there have been several strides in applying style transfer to text and videos as well. In this project, we primarily focus on image style transfer, and we aim to achieve higher quality photos, better lighting, and exposure without the use of

photoshop or any image editing tools by employing style transfer technique which is not the common application of style transfer.

Style Transfer for Photo Editing

3. Related Work

3.1 Non-Deep Learning Methods

(Written by Adithya K Krishna (akk2188))

There have been relatively few lighter networks that attempted to achieve style transfer. Ashikhmin [1]. proposes 2 key modifications to his previous work to increase the search space and introduce candidate-specific metrics to achieve faster and better results. Effros and Freeman [2] introduced an image quilting algorithm for texture transfer achieving a photorealistic rendering

3.2 Deep Learning Methods:

(Written by Adithya K Krishna (akk2188))

This can be further classified as an improvement upon pre-trained works/ Online models and Offline models.

Offline methods include Neural Style Transfer by Gatys et al., which works on top of VGG-16 using ImageNet data by introducing style loss with a Gram matrix. This was followed by another paper by Luan et al. [3] that added the key aspect of semantic segmentation to avoid deviating from the required classes, thus giving a large improvement and more photoreal output.

There are several works that focus on single-style, multiple-style, and arbitrary-style transfer. A few interesting ones are Perceptual losses for real-time style transfer and super-resolution. By Johnson et al.[4] that uses SGD to train residual Conv nets, Chen et al. [5] that contain banks of convolutional filters, these filter banks use many filter channels to store stylish information, and Huang et al. [6] the first networks to satisfy the requirement of arbitrary style transfer by employing adaptive instance normalization which is the primary module that works towards arbitrary style transfer.

Neural Style Transfer has a large number of applications in the computer vision domain. However, it is far less explored to edit images and produce professionally edited like images from their raw forms. The previous works merely list editing images as a benefit of their modules, but not many deliberate efforts have been put in to apply neural style transfer for editing images.

There are a few research papers that work around using style transfer for specific vision applications, and there are others that build modules for general style transfer between any two types of images. For our use case, we mainly explored the second type of method because the others are highly tuned for a specific task. We use the Deep Photo Style Transfer by Fujun Luan et al. as our base method for this project, make novel changes to improve their style transfer procedure, and build an additional module that automatically selects the best style image from our curated dataset to edit the given input image. This is an end-to-end pipeline and takes as input just the input image and input segmentation files to generate the edited stylized image.

3.3 Deep Photo Style Transfer (DPST)

(Written by Tanav Shah (ts3469))

DPST is a generative algorithm. The algorithm takes two images as input, the input image, and the reference style image. The model aims to transfer the style of the reference image to the input image with photo realism while preserving its overall structure. This algorithm is built over the method proposed by Gatys et al. in their work "Image Style Transfer Using Convolutional Neural Networks."

Suppose we have a style image S and an input image I, and we want to produce an output image O. The objective function they use to minimize is given by:

$$\mathcal{L}_{\text{total}} = \sum_{\ell=1}^{L} \alpha_\ell \mathcal{L}_c^\ell + \Gamma \sum_{\ell=1}^{L} \beta_\ell \mathcal{L}_s^\ell$$

$$\text{with: } \mathcal{L}_c^\ell = \frac{1}{2N_\ell D_\ell} \sum_{ij} (F_\ell[O] - F_\ell[I])_{ij}^2$$
$$\mathcal{L}_s^\ell = \frac{1}{2N_\ell^2} \sum_{ij} (G_\ell[O] - G_\ell[S])_{ij}^2$$

Here L is the total number of convolutional layers. There are Nl number of filters in each layer, with a vectorized feature map of size Dl. The weights alpha and beta are weights for each layer, and lambda balances the amount of style to be transferred. Fl is a feature matrix, and Gl is the gram matrix defined as the inner product between the vectorized feature maps.

DPST adds a photorealism regularization term which helps to preserve the structure of the input image and produce photorealistic output images. This idea is the first and very efficient attempt at this problem. They assume that the input image is already photorealistic and then ensure that during their process, they do not lose this property. For this, they add a component to the loss term called affine loss. This term uses Matting Laplacian matrices, first introduced and used by Levin et al. in "A closed-form solution to natural image matting." This is given by:

$$\mathcal{L}_m = \sum_{c=1}^{3} V_c[O]^T \mathcal{M}_I V_c[O]$$

We skip the further mathematical details of this part in the report, which are mentioned in great detail in the paper.

The method by Gatys et al. uses a style term in the loss, but it has its limitations. This term uses a Gram matrix computed over the entire image, which encodes the exact distribution of the neural responses from the image. This makes the Gram matrix incapable of adapting to variations in the context and often leads to spillovers. To address this, DPST uses the segmentations of the input and style images to generate segmentation masks and use these as additional channels. The style loss component is then summed over all these channels. The computation of the Gram matrix still happens in the same way but on different inputs.

Finally, all these three components of the loss function are added to compute the overall loss. DPST uses a pre-trained VGG-19 model for feature extraction from the input images. The output of conv4_2 (2nd convolutional layer of the 4th block of VGG) is used as the content representation, and the outputs of conv1_1, conv2_1, conv3_1, conv4_1, and conv5_1 are used for style representations.

The output image is initialized to a random noise of the same dimension as the required output. Using SGD and Adam optimizer, the pixels of this resultant image is updated over iterations using the loss from work by Gatys. This image is saved as intermediate output. Again, taking this intermediate image as input, the same process is repeated, but this time with the new updated loss function from DPST. This final image is returned as the output.

3.4. Comparison of DPST with Other Methods

(Written by Tanav Shah (ts3469))

The older methods, such as the one from Gatys et al., work well but still cannot handle the case of spillovers as described in the previous section. Another such transfer method is “Color transfer between images” by E. Reinhard et al. They used the idea of global color transformation to affect the image colors strongly, but not have any impact on the image structure. This method, however, has limited capabilities because it cannot model spatially varying effects and, thus, cannot faithfully transfer the style of the second image. These limitations are effectively addressed by DPST.

A few more recent works, such as "Photorealistic Style Transfer via Wavelet Transforms" by Jaejun Yoo et al., attempt the style transfer problem differently. They propose their own end-to-end model, called WCT2. This model also uses VGG-19 for feature extraction but has its own architecture involving wavelet pooling, unpooling layers. Instead of the output image being generated over iterations, WCT2 employs a progressive stylization strategy to generate the output in a single pass. There are also a few variations of the WCT2 model discussed in the paper, but they do not significantly outperform the DPST method. Another paper on "Arbitrary Style Transfer with Deep Feature Reshuffle" by Shuyang Gu et al. shuffles the feature maps of the images to connect their defined global and local style losses to generate output images.

Though these methods perform slightly better than DPST, they are not very useful in our scenario. These are end-to-end pipelines and therefore do not provide an explicit option to control the amount of style to be transferred. These methods generate the output image directly using their loss function and architectures. This leads to good artistic images, but we want to control the amount of style from the professionally edited style image to be transferred to the raw input image to be edited. This is why we used DPST as our base method for this project.

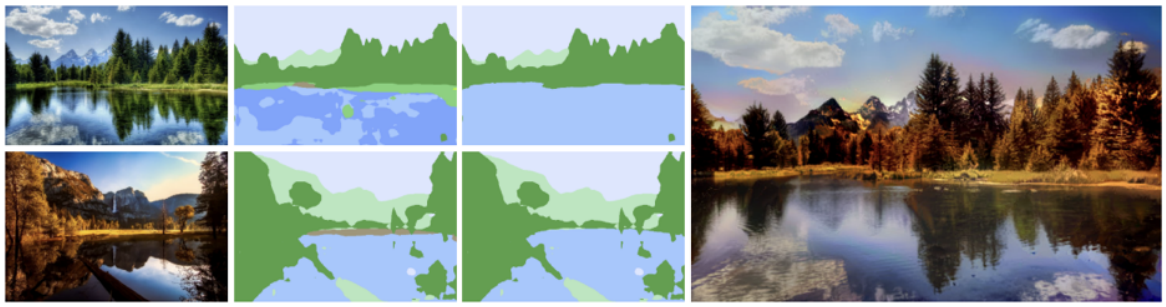
4. Dataset

(Written by Adithya K Krishna (akk2188))

The datasets typically used for this application include ImageNet[9], the world’s largest visual recognition and classification dataset; the COCO Dataset, largescale object detection, and captioning data; and the Cityscapes dataset[10], which focuses on segmentation data of urban street scenes. A few datasets specific to our application include CMP faced dataset, which includes 606 rectified images of facades from various sources, and WikiArt data, which consists of over 150 thousand painting images by 2500 artists. The images in the dataset were collected and annotated by WikiArt and were proposed in[11].

In our method, we aim to use a few photos that were taken on our mobile phones(that is, images not having a high quality) and to use images online that have specific features like good

exposure, brightness, and lighting and transfer their style to our low-quality images.



General Style Transfer Pipeline

4.1. Data Generation

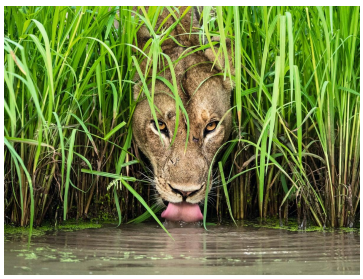
(Written by Adithya K Krishna (akk2188))

To train our model for the purpose of our implementation, the data is generated from scratch. We consider 3 main classes of images: wildlife, portraits, and landscapes, on which we apply style transfer. We employ web scraping to extract those images using Python script from scratch that uses Selenium web automation and testing library and BeautifulSoup to web scrape and download images from Google Images.

In order to obtain high-quality images for the purpose of feature extraction by the style transfer model and segmentation, the images are extracted from their original source using the python-requests library. This ensures that we get high-resolution images rather than the ones loaded on the web browser content.

We get 120 images in each class since we are using pre-trained weights for most of the layers, and only the last few layers are learned. Separate demo images are collected for the purpose of testing our performance.

The style transfer model requires an image along with the segmented mask of it; how the segmentation mask is obtained is explained below.

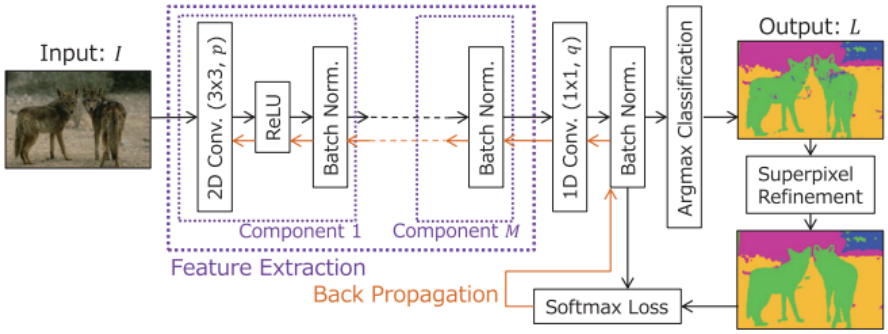




4.2. Segmentation

(Written by Adithya K Krishna (akk2188))

For any supervised task, it's generally challenging to find labeled data as it requires a lot of manual labor to annotate them, and there are very few tasks that can be automatically annotated. Especially for the task of segmentation, where we need to classify pixels based on labels which makes it difficult to label the images.

To overcome this, we implement unsupervised segmentation by using convolutional networks. In the case of unsupervised scenarios, ground truth labels of pixels are not given beforehand. So, the feature representations of the target image, along with pixel labels, are optimized to obtain the segmentation. In our approach, we optimize them such that: (1) Pixels that are spatially aligned together will have the same label, and (2) Pixels of similar feature representation need to have the same label.



The forward propagation computes a p-dimensional feature map {xn} from {vn} through M convolutional components, each of which consists of a 2D convolution, ReLU activation function, and a batch normalization function, where a batch corresponds to N pixels of a single input image. Here, we set p filters of region size 3 × 3 for all of the M components.

A response map is obtained by employing a linear classifier, which is normalized to zero mean and variance of 1.

$$\{\boldsymbol{y}_n = W_c \boldsymbol{x}_n + \boldsymbol{b}_c\}_{n=1}^{N}$$

Finally, we obtain the cluster label c_n for each pixel by selecting the dimension that has the maximum value in y_n'.
The constraint equations applied to feature similarity will result in clusters:

$$C_i = \{\boldsymbol{y}'_n \in \mathbb{R}^q \mid y'_{n,i} \geq y'_{n,j}, \ \forall j\}$$

here, C_i is the ith cluster of final responses.

In unsupervised image segmentation, one of the main limitations is the number of segments to be generated in an image. For an accurate and robust model, this value must adapt to the number of different classes in the image. The proposed strategy is to classify pixels into an arbitrary number of clusters.

If the number of clusters is too large is over-segmentation, while a very low number of clusters results in under-segmentation. Intra-axis normalization for the response map explained above can be done before assigning cluster labels based on argmax can nudge the number of clusters to a large value, thus allowing us to learn the labels.
The intra-axis normalization is given by:

$$y'_{n,i} = \frac{y_{n,i} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

4.2.1. Related Works

(Written by Adithya K Krishna (akk2188))

A few notable works in semantic segmentation include [13], [14], [15], [16], [17], [18]. These methods use a CNN as their base model and use object detectors and user input for assistance in generating the semantic segmentation labels. Another class of algorithms that aim to solve the task of semantic segmentation include weakly supervised learning models [19], [20], [21], [22]. These models use bounding boxes and image-level class labels to achieve segmentation.

Moving onto our class on methods, unsupervised learning. Deep Learning models [23], [24] have proven to be effective in texture recognition and segmentation. Backpropagation is carried out in these methods, although due to the lack of labels, by reducing the loss between soft assigned data points with an auxiliary target distribution.

In our method, we minimize the softmax loss based on the estimated clusters.

4.2.2. Methodology

(Written by Adithya K Krishna (akk2188))

When an input image is provided, the following steps are carried out. We freeze the model weights and learn the prediction clusters; then, the clusters are frozen, and the model parameters are learned. As in the general case of a neural network, the first step is the forward propagation step, where we predict the label, and the second step is the backpropagation that learns the model weight by updation using gradient descent. In addition to the forward propagation step, we do a superpixel refinement to the clusters to make the prediction better.

We want to spatially group pixels together belonging to the same class, for which we add additional constraints. By letting the number of pixels in the superpixel be maximum, we select the most frequent cluster label.

4.2.3. Training details

(Written by Adithya K Krishna (akk2188))

2 layer CNN model is trained for 30 iterations per image to achieve the segmented image in each class. We take a learning rate of 0.1. The loss saturates, and we get the image with each label colored in with different color marking each label.

5. Methodology

(Written by Rohit V Gopalakrishnan (rvg2119))

Our method is based on DPST and we add several novel modifications to it to improve it and adapt to the task of editing images. For the purpose of this project, we have divided our project into 3 major goals:

1. First, we train new classes such as ‘wildlife’, ‘scenery’, etc, and train the model to map onto a style of a class in which the input image lies. For this we need to procure a new dataset.
2. We come up with a new loss function component, to better improve the output images of the model, by yielding better loss differences.
3. Following this, we automate the pipeline for multiple classes. The automation step is further explained below in this paper.

5.1 Network Architecture

(Written by Rohit V Gopalakrishnan (rvg2119))

There have been a few research papers published that produce good results for stylizing a content image using a style image. However, these techniques have not been applied to edit photos realistically and produce results good enough to eliminate the need for manual editing. For our project, we plan to follow the architecture similar to the paper Luan et al. [5] published in CVPR 2017. They improve the style transfer model but apply it to create stylized images and not to produce good editing of a photograph. Their method works well for their use case, and we believe we can extend it to ours as well.

The core idea of the algorithm is to obtain the features of the input image and the reference image and compute the Matting Laplacian matrices (similar to the work by Levin et al.) and Gram matrices. All these attributes are then used to create the target image. The target image is initialized as a copy of the input image and iteratively moves toward the style image. The loss

consists of 3 components, two with weights that control the extent to which the target image is closer to the style or input image (style loss and content loss) and a regularization parameter.

The network architecture we use is to extract the features from the VGG-19 pre-trained model and choose the outputs of specific layers as feature extractions of the input and style images. For the style transfer in the reference paper, they use combinations of convolutional layers for this task. For computing the Gram and Matting Laplacian matrices, we follow similar procedures as previous papers. Some papers also apply segmentation to the input and style images to help the model understand better in which sections of the image the style is to be applied and to what extent.

5.2 Implementation Details and Novel Changes

(Written by Tanav Shah (ts3469))

The code from the authors of DPST was implemented in an older version of python and Tensorflow version 1 and had several incompatibility issues. We worked around these to upgrade the complete code to Tensorflow version 2.4.1 and python 3.8. Making the exact versions of TensorFlow, Keras, NumPy, pillow, python, scipy, etc., compatible with each other and getting the code base running was in itself a huge challenge which took several hours to get executable.

The DPST method uses the VGG-19 model for feature extraction, as described in the previous section. Since we know that Resnet performs better for the classification task, we tried extracting the feature vectors using Resnet. Still, the final output images could not faithfully transfer the style of the reference image. In some cases, the structure of the input image also got distorted somewhat more as compared to VGG. On reading more about this, we got to know that VGGs have inherent stability and inability to learn non-robust features of images, which makes them the best choice for feature extractor for our use case.

The code from the authors of DPST uses a pre-trained VGG model on a dataset different from imagenet. Therefore, we changed this part to use the pre-trained VGG-19 model from Tensorflow on Imagenet. This model architecture was slightly different from the ones the authors used, so we made several changes in the styling method to adapt to these. This VGG from TensorFlow is optimized for the classification task, which is significantly different from our use case. Therefore, we decided to use transfer learning, as discussed in the lectures and homework assignments, to make the VGG-19 model adapt to our use case of style transfer.

The VGG-19 model has the last two layers as fully connected layers and a 1000 class prediction layer. We modified these to be a fully-connected layer with 1024 hidden units and the final layer as a 3-class prediction layer using softmax. These three classes it predicts are from our newly curated dataset, namely, portraits, landscape, and wildlife. Apart from these last two layers, all other model layers are frozen for training, and the model is trained for five epochs on our newly curated dataset. Next, all layers are unfrozen, and the complete model is trained again on the same data. This transfer learning technique helped the VGG-19 model to get fine-tuned to our

use case of image style transfer on our data. This newly trained model is then saved so that the training does not need to be done every time we want to edit an image. With this transfer-learned model, we achieved more stable images with slightly fewer distortions on the same input and reference style images. That is why we proceeded with this model.

DPST uses three components for the loss function described in the previous section. We decided to add a new component to this loss function, which we thought about and implemented from scratch. The features extracted from the images using our new VGG-19 model are the combinations of outputs of certain model layers. These feature vectors are used to compute the loss component value. We consider the different elements of these feature vectors to be different dimensions. Therefore, flattening these feature values gives us a vector in some n-dimension space, where n is the number of feature values extracted. This is a positional vector considering the origin to have all zero values. We obtain such vectors for the features of input and style images. Next, we compute the cosine of the angle between these two vectors using the dot product. We want the features extracted to be aligned together and therefore want this angle to be as small as possible. The intuition is that since we are starting from a random noise image, we want this image to be aligned with the style image features. We did not directly use the images and used the extracted features because the previous case would result in the output image getting very, very close to the style image, which is not ideal. Since this is a component of the loss function, we use the reciprocal of cosine (sec) as the loss component value. To summarize, the function computes the norm of the two feature maps and their sum of element-wise products. The multiplication of both norms is divided by this sum to obtain the loss component value.

Combining all these novel additions to the DPST model, we achieved better results. The output images had slightly fewer distortions and were more structurally similar to the input image, still faithfully having the style of the reference image.

5.3 Novel Style Automation

(Written by Tanav Shah (ts3469) and Rohit V Gopalakrishnan (rvg2119))

As mentioned in the Objectives section, one of our main goals for this project is to come up with a method to allow the automation of choosing a suitable style to edit our source image. Previous implementations of this problem statement require the user to manually provide both the content image (i.e., input) and style image. The network then provides the transfer image as output. However, this is a tedious process to do every time, especially when the user simply wants to enhance their image and not care about the different styles that they can use. In order to achieve this, we worked on two parts of the methodology that need to be automated. This is our enhancement to the existing implementations for the given problem statement.

More important to our mentioned objective, we must come up with a system where the network itself decides which style is more suitable for the input image at hand, and apply the style transfer for that image. This has not been done before, hence we come up with our own methodology. Similar to the idea of the new loss component we add, we choose the style image

based on the cosine distance between the style and input image features. We use the transfer learned version of the VGG-19 model we created to extract the feature representations of the input image and each style image in our newly curated dataset. This feature vector is the output of the last fully-connected layer in our model, which has 1024 hidden units. We compute the cosine distance between these vectors of the input image with all the style images and select the style image which results in the least distance.

In order to find the style image best suited for the input image, we must find a well edited image that has the same physical structure as the input image. That is, we must find a similar looking image that is already enhanced in quality in order to map the style of that image more accurately onto the input image, providing us with potentially better mapping of styles. This property is quite well satisfied by our method, as shown in the demonstration section.

Below we can see an example of how our algorithm works. The input image is a scenic view of a house. All images in the style dataset are compared to the input image and we choose the image which has the highest similarity. As you can see, a better edited, more vibrant picture of a house painted white comes up. As they are similar images of scenic house views, but of different edits, this will result in better mapping of styles.

Input Image

Check through our Style Dataset for Similar image

Select Appropriate Style Image

6. Experiments

(Written by Rohit V Gopalakrishnan (rvg2119))

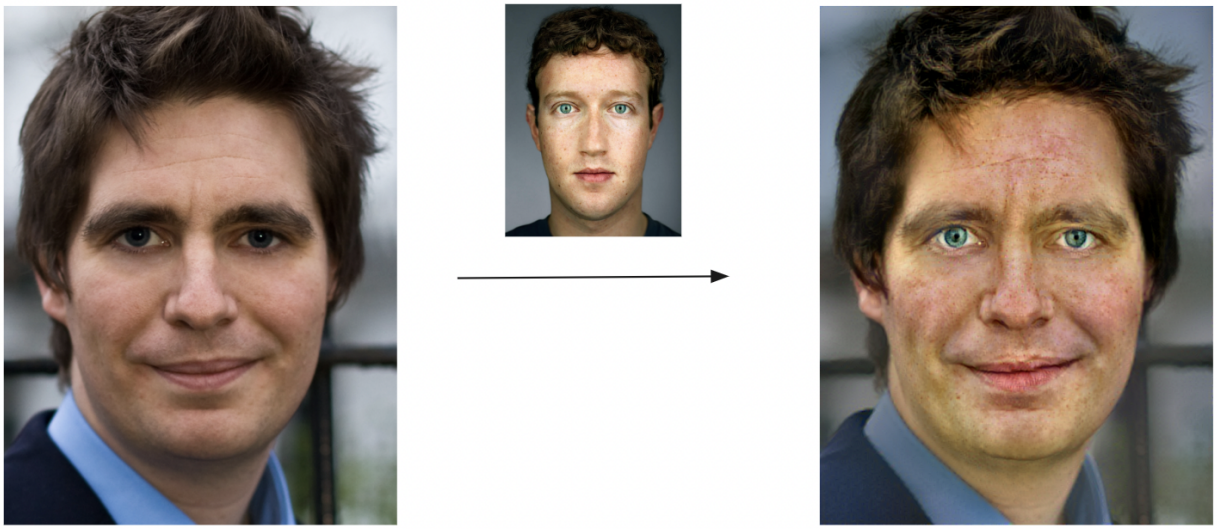
It is important to see whether the methodology we came up with holds good in solving the problem at hand. Hence, we go through some demo runs of the project to see how the style image is chosen based on the input image, and the resulting output image that it results in because of that.

6.1 Demonstration

(Written by Rohit V Gopalakrishnan (rvg2119))

In this paper we go through 5 different input images, and see the results, both in terms of the style image chosen and the output image that is generated.

In the below example, we can see the input image is a headshot of a person. In the style dataset, the head shot of Mark Zuckerberg is most similar to the input image and hence, is selected. As we can see, the output image is the face remodeled with Mark's stylistic facial features such as his blue eyes and reddish skin tone.



In the next example, we see an input image of a dog. While there isn't much apparent difference in the output image, we can make some distinctions that show it is a better edit. The contrast in the input image is high, darkening the darker pixels and highlighting the lighter pixels as such. This is fixed in the generated output image. Note that the image has been color corrected as well to match that of the style image that was chosen for this input.

In the next example, we can observe an apparent difference in the output image. The input image is that of a suburban house. The sky is over-saturated while the grass looks pale and

doesn’t show any life. The style image chosen is a similar picture of a white house, with a sky blue sky and greener grass. As it is a picture of a house, similar to the input image, the style is able to match well and we get a more vibrant output of our input image. Note the change in the color of the walls and the mailbox.

Similar to the house picture, we have a scenic view of grassland as input. The style image chosen is a similarly more vibrant image that brings out the blue of the sky and the green of the grass. We can see the changes in the output image generated. Note the subtle hints of yellow because of the flowers in the style image.

Finally we have an image of a panda. Similar to the image of the dog, we can’t see that much of a difference here. However if you look closely, you will observe that the details of the pandas face are a lot sharp while the face and body itself is smoothened out. The color correction adds a tint of red giving it a warmer hue. It is these differences that differentiate a good edit from a professional edit.

6.2 Inferences and Other Results

As evident from the above examples and previous research works on similar problems in image style transfer, results of the model can be evaluated qualitatively only. More specifically, what style image is chosen and what output image is generated is a stylistic choice, with no concrete right or wrong answer.

Previous implementations discussed the results of their model based on feedback given from a group of people. Hence, we must do the same for this problem statement as well. On review with a few of our peers, we came to the conclusion that the methodology and implementation of the additional loss component along with the choice of automating the style image choice, improved the quality of the output image, and does so better than previous methodologies.

As mentioned before, we are the first to try and automate the model to choose a style best suited for the input image, by doing so we reduce the burden on the user to have the knowledge required to make good edits for photographs. Hence, it is absolutely important that the style image is chosen appropriately for the image.

As you observed in the 5 examples above, the style image chosen at each input is very similar in content to the input image, but edited with a better more professional looking style. As they are structurally similar we get better mapping of styles and hence, we get better output images.

7. Conclusion

7.1 Conclusion

(Written by Rohit V Gopalakrishnan (rvg2119))

In conclusion, we came up with an implementation of image style transfer that is used to achieve higher quality photographic images by enhancing basic photo editing elements such as vibrance, hue, saturation, etc. In order to do this, we procured a new dataset of well edited images that we use as reference to style the input images. We came up with an additional loss component to the loss function of previous implementations to better improve the output quality of the generated image. Finally, we introduced the method of automating the stylistic usage of already edited photographs (style images).

This implementation makes it easier for beginners of editing to understand the stylistic choices that need to be made when editing photographs. It is also useful for veteran photographers who prefer editing in a particular style for the same type of photography. They can train the model on a class of photos of similar editing style, so that they can automate the stylistic choice for photographs of similar nature.

7.2 Future Work

(Written by Rohit V Gopalakrishnan (rvg2119))

For further work on this problem, we can expand our style dataset to cover a variety of classes instead of simply just 3 classes. This would account for editing styles of a lot more areas of photography. We can also conduct analytical research to discover better/new methods of improving this methodology to produce sharper more realistic output images of enhanced quality.

8. Bibliography

(Written by All)

[1] L. Zhong, N. Li and Y. Sun, "Research and Application of Image Style Transfer Method," 2022 IEEE 8th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), 2022, pp. 143-147, doi: 10.1109/BigDataSecurityHPSCIDS54978.2022.00035.

[2] X. Shi and W. Hu, "A Survey of Image Style Transfer Research," 2022 2nd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), 2022, pp. 133-137, doi: 10.1109/CEI57409.2022.9950226.

[3] Ashikhmin, N. "Fast texture transfer." IEEE computer Graphics and Applications 23.4 (2003): 38-43.

[4] Efros, Alexei A., and William T. Freeman. "Image quilting for texture synthesis and transfer." Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001.

[5] Luan, Fujun, et al. "Deep photo style transfer." Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[6] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." European conference on computer vision. Springer, Cham, 2016.

[7] Chen, Dongdong, et al. "Stylebank: An explicit representation for neural image style transfer." Proceedings of the IEEE conference on computer vision and pattern recognition, 2017.

[8] Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." Proceedings of the IEEE international conference on computer vision, 2017.

[9] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255), IEEE.

[10] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., ... & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3213-3223).

[11] Tan, W. R., Chan, C. S., Aguirre, H. E., & Tanaka, K. (2016, September). Ceci n'est pas une pipe: A deep convolutional network for fine-art paintings classification. In 2016 IEEE International Conference on Image Processing (ICIP) (pp. 3703-3707), IEEE.

[12] Penhouët, Sebastian, and Paul Sanzenbacher. "Automated deep photo style transfer." arXiv preprint arXiv:1901.03915 (2019).

[13] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," IEEE Trans. on Pattern Analysis and

Machine Intelligence, 2017.

[14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs,"

in ICLR, 2015.

[15] Shuai Zheng, Sadeep Jayasumana, Bernardino RomeraParedes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr, "Conditional random fields as recurrent neural networks," in ICCV, 2015.

[16] Joseph Tighe and Svetlana Lazebnik, "Finding things: Image parsing with regions and per-exemplar detectors," in CVPR, 2013.

[17] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik, "Simultaneous detection and segmentation," in ECCV, 2014.

[18] Jifeng Dai, Kaiming He, and Jian Sun, "Instance-aware semantic segmentation via multi-task network cascades," in CVPR, 2016.

[19] Jun Zhu, Junhua Mao, and Alan L. Yuille, "Learning from weakly supervised data by the expectation loss svm (e-svm) algorithm," in NIPS, 2014.

[20] Feng-Ju Chang, Yen-Yu Lin, and Kuang-Jui Hsu, "Multiple structured-instance learning for semantic segmentation with uncertain training data," in CVPR, 2014.

[21] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell, "Constrained convolutional neural networks for weakly supervised segmentation," in ICCV, 2015.

[22] Niloufar Pourian, Sreejith Karthikeyan, and Bangalore S. Manjunath, "Weakly supervised graph-based semantic segmentation by learning communities of image-parts," in ICCV, 2015.

[23] Mircea Cimpoi, Subhransu Maji, and Andrea Vedaldi, "Deep filter banks for texture recognition and segmentation," in CVPR, 2015.

[24] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik, "Hypercolumns for object seg