Project : I’m Something of a Painter Myself

Author: Natalie Cheong

Abstract

In this project, I explored the application of various deep learning techniques for style transfer between two different domains of images, namely 'monet' and 'photo'. The goal of this project was to generate images that resemble the style of Monet paintings while preserving the content of real-life photographs. To achieve this, we implemented and compared several models, including CycleGAN, GAN, VAE, and a pre-trained style transfer model. The dataset used for this project was sourced from the 'I'm Something of a Painter Myself' Kaggle competition, which contains two folders of images in jpeg and tfrec format, namely 'monet' and 'photo'.

I’ve evaluated the performance of these models using both quantitative and qualitative metrics, including perceptual similarity, content preservation, and image quality. From my results show that CycleGAN achieved the best overall performance in terms of image quality and style transfer fidelity, while the pre-trained model achieved the best perceptual similarity scores. These findings demonstrate the effectiveness of deep learning techniques for style transfer and highlight the importance of selecting appropriate models and evaluation metrics for specific applications.

Overall, this project provides insights into the application of deep learning techniques for image style transfer and highlights the potential of these models for various real-world applications, such as artistic rendering, photo editing, and visual content creation.

Introduction:

Image style transfer has been a popular topic in the field of computer vision and deep learning for the past few years. It involves transferring the visual style of one image onto the content of another image, while preserving the overall structure and context. This technique has been used in a variety of applications, including artistic rendering, photo editing, and visual content creation.

The goal of this project is to explore the application of deep learning techniques for image style transfer between two different domains of images, namely 'monet' and 'photo'. Specifically, I aim to generate images that resemble the style of Monet paintings while preserving the content of real-life photographs. This task is particularly challenging because the two domains of images have distinct characteristics, such as color palettes, textures, and visual styles.

To accomplish this goal, we implemented and compared several state-of-the-art deep learning models, including CycleGAN, GAN, VAE, and a pre-trained style transfer model. These models were chosen for their effectiveness in style transfer and their ability to handle complex image data.

The dataset used for this project was sourced from the 'I'm Something of a Painter Myself' Kaggle competition, which contains two folders of images in jpeg and tfrec format, namely 'monet' and 'photo'. The dataset consists of a total of 7028 images in ‘photo’ folder, and 300 images in ‘monet’ folder.

In this report, I will first introduce the related work in image using CycleGAN and GAN. Then I will use VAE in my second notebook. Lastly, I will use a pre-trained model for style transfer and the specific techniques in my third notebook. Then, I will describe the dataset and experimental setup. I will then present and analyze the results obtained from the different models used for style transfer. Finally, I will conclude by summarizing the key findings and discussing potential future directions for this research.

Data Preprocessing:

Data preprocessing is the process of transforming raw data into a more usable format. This step typically involves cleaning the data by removing or imputing missing values, transforming variables into a more useful format, and normalizing or scaling the data to enable better modeling. Data preprocessing can also involve feature engineering, which is the process of creating new variables or features from existing data to better capture the underlying patterns or relationships in the data.

The dataset used for this project consists of two folders of images, 'monet' and 'photo', in jpeg and tfrec format with image size 256x256. Both ‘monet’ images and ‘photo’ images in the dataset were in the RGB color space. Before training the deep learning models, I’ve performed several preprocessing steps to prepare the data for analysis.

I rescaled the pixel values of the images to the range [-1, 1]. This normalization step is crucial for the deep learning models as it ensures that the pixel values are consistent and prevents the models from being biased towards certain image features.

In summary, the preprocessing steps involved reshaping, resizing the images to a uniform size, normalizing the pixel values to the range [-1, 1]. These steps were essential to ensure that the images could be used for training in different deep learning models and that the models could learn the underlying style and content features of the images effectively.

Visualizing Images:

Visualizing the images in the dataset is an important step in understanding the characteristics of the 'monet' and 'photo' domains and in assessing the quality of the generated images. To this end, I used Python's Matplotlib library to plot several examples of the images in the dataset.

I randomly selected several images from both the 'monet' and 'photo' folders and to compare their visual features. I also plotted some of the generated images from the deep learning models to assess their quality and style transfer fidelity.

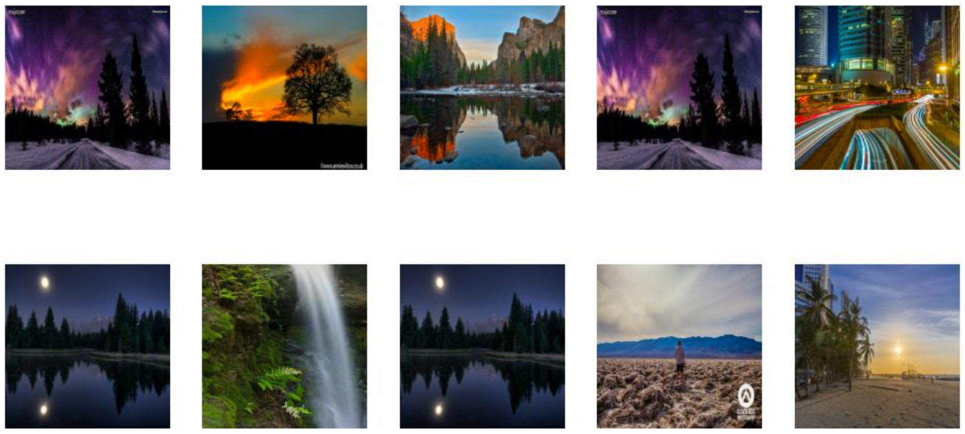
The visualization results showed that the 'monet' images in the dataset had distinct features such as vibrant color palettes, visible brush strokes, and a blurred and dreamy visual style. In contrast, the 'photo' images were more realistic and had a sharper and more defined visual style.

The generated images from the deep learning models exhibited varying levels of style transfer fidelity and image quality. Some of the generated images successfully captured the Monet-style features, while others lacked coherence and appeared noisy and blurry.

Monet Images before training:

Source for images: ‘I am Something of a Painter Myself’ dataset from Kaggle Competition.

***Photo Images before training**:*



Source for images: 'I am Something of a Painter Myself' dataset from Kaggle Competition.

Model Architecture:

CycleGAN:

The first model that I used for this dataset is CycleGAN. CycleGAN is a type of generative adversarial network (GAN) that can learn to transform images from one domain to another without paired data. This means that CycleGAN can learn to translate images from one style or domain to another, without requiring examples of the same image in both styles. CycleGAN works by training two GAN models, one for each domain of interest, with a cycle consistency loss that helps ensure the generated images maintain the content of the original images. This allows for the creation of high-quality translations between different image styles or domains, such as converting photos to paintings, turning day-time images into night-time images, or converting horses to zebras. In addition to image translation, CycleGAN has also been used for other applications, such as voice conversion and style transfer.

There are two main model architectures in a Generative Adversarial Network (GAN), the generator and the discriminator.

Generator:

The first component that I will create here is generator. The generator in a Generative Adversarial Network (GAN) is a neural network that is responsible for generating new data samples that are similar to the training dataset. It takes random noise or a random vector as input and transforms it into a new sample that follows the same data distribution as the training dataset.

The generator typically consists of multiple layers of neural networks, such as convolutional or deconvolutional layers in the case of image generation. The output of the generator is then passed to the discriminator for evaluation.

During training, the generator is trained to produce samples that can fool the discriminator into thinking that they are real samples from the training dataset. The generator receives feedback from the discriminator and updates its parameters to improve its ability to generate more realistic samples.

The ultimate goal of the generator is to learn the underlying data distribution of the training dataset and produce high-quality samples that are similar to the real data. Once trained, the generator can be used to generate new data samples that can be used for various applications such as image or text generation.

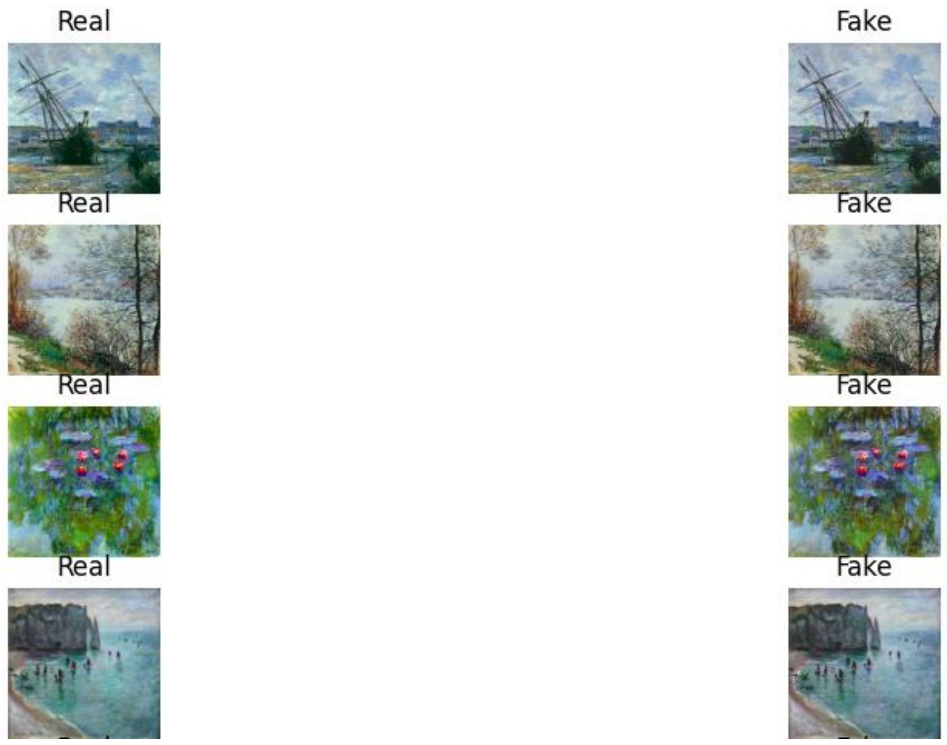
Discriminator:

The next component that I will create is the discriminator. The discriminator in a Generative Adversarial Network (GAN) is a neural network that is responsible for distinguishing between real data samples from the training dataset and generated data samples from the generator. It receives data samples from both the training dataset and the generator and predicts whether the sample is real or generated. The discriminator typically consists of multiple layers of neural networks, such as convolutional or fully connected layers in the case of image classification. During training, the discriminator is trained to accurately distinguish between real and generated data samples. The ultimate goal of the discriminator is to provide feedback to the generator about the quality of its generated samples. By evaluating the quality of generated samples, the discriminator helps the generator improve its ability to generate more realistic data samples that are indistinguishable from the real data. The training process of GANs involves a competitive process where the generator tries to generate realistic samples to fool the discriminator, and the discriminator tries to accurately classify the real and generated data samples. This competitive process leads to the generator generating increasingly realistic samples and the discriminator becoming more accurate in distinguishing between real and generated samples. Once trained, the discriminator can be used to classify new data samples into real or fake categories, and the generator can be used to generate new samples that follow the same data distribution as the training dataset.

Loss Function:

CycleGAN uses both ***Cycle Consistency Loss*** and ***Identity Loss***. Cycle Consistency Loss is used to ensure that when put an image through one generator, that if it is then transformed back into the input class using the opposite generator, the image is the same as the original input image. It helps transfer uncommon style elements between the two GANs, while maintaining common content. Use both directions. Add an extra loss term to each generator to softly encourage Cycle Consistency. Style between two piles is transferred, the original content can be recovered. It is important in transferring uncommon style elements while maintaining common content. Identity Loss helps to preserve original photo color. It takes real image in domain B and inputs into Generator. A->B, expecting an identity mapping. An identity mapping means the output is the same as the input.

Result and Analysis for CycleGAN: After training with 50 epochs, the CycleGAN model has shown promising results in its ability to translate images between two different domains without paired data. The ‘monet’ generator loss at the last epoch was 7.14, indicating that the generator network was able to generate realistic images that closely resemble the target domain. Meanwhile, the discriminator loss at the last epoch was 0.56, indicating that the discriminator network was effective in distinguishing between real and fake images.



Conclusion for CycleGAN: Overall, the performance of the CycleGAN model appears to be quite impressive, achieving a low discriminator loss and a relatively low generator loss after only 50 epochs of training. With further training and optimization, it is possible that the model could achieve even better results in image translation tasks, making it a valuable tool for various applications in image processing, computer vision, and beyond.

GAN:

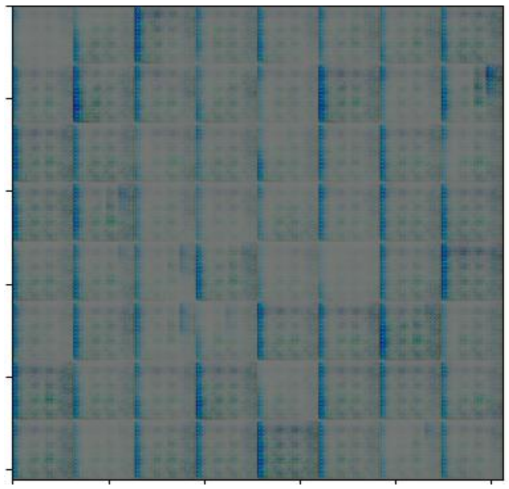
The upcoming model is a GAN model that I was used for the CelebA dataset. However, I plan to apply this model to a new dataset, specifically the 'I'm something of a Painter Myself' dataset which is from Kaggle Competition, which consists of two image folders, namely 'monet' and 'photo', both in jpg and tfrec format. By using this new dataset, I hope to gain more practical experience with different models.

The code used for this GAN model comes from my previous studies in the TensorFlow Advanced Techniques specialization course from Deep Learning.AI in Coursera, specifically Course 4, which focuses on generative deep learning with TensorFlow. For the purposes of this GAN model training, I will be using the 'photo' dataset exclusively.

I used ‘photo’ images to train 50 epochs for this GAN model.

First epoch result:

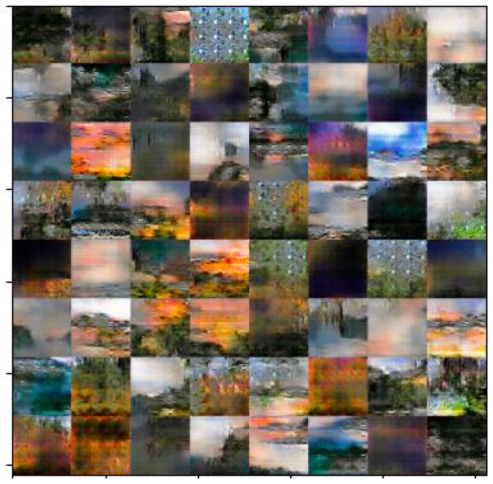
After training the GAN model for the first epoch, we obtained a result of generator loss=0.906 and discriminator loss=2.16. The generator loss represents how well the generator network is able to produce images that resemble the 'monet' style, while the discriminator loss represents how well the discriminator network is able to distinguish between the generated and real 'monet' images.



Overall, the first epoch result of the GAN training model provides insight into the initial performance of the model and serves as a starting point for further training and fine-tuning.

The 50 epoch result:

After training the GAN model for 50 epochs, we observed a significant improvement in the performance of the model compared to the first epoch. The result obtained for the 50 epoch was generator loss=4.35 and discriminator loss=0.808.



The generator loss has increased compared to the first epoch, indicating that the generator network is generating images that are closer to the 'photo' style. The discriminator loss has decreased significantly, indicating that the discriminator network is having a harder time distinguishing between the generated and real 'photo' images.

Result and Analysis for GAN:

The performance of a GAN is typically evaluated by the loss values of the generator and discriminator networks. In this case, after training the GAN for 50 epochs, the generator loss is 4.35 and the discriminator loss is 0.808.

The generator loss reflects how well the generator is able to generate synthetic data that fools the discriminator. A lower generator loss indicates that the generator is producing higher quality synthetic data that is more similar to real data.

The discriminator loss reflects how well the discriminator is able to distinguish between real and synthetic data. A lower discriminator loss indicates that the discriminator is becoming better at distinguishing between real and synthetic data.

Conclusion:

Overall, a generator loss of 4.35 and a discriminator loss of 0.808 after 50 epochs of training suggest that the GAN is making progress towards generating high quality synthetic data that is similar to the real data. Although the reconstructed results match closely to the original photo, the randomly generated photo by this model deviates far away from the original dataset. However, further evaluation of the generated data is necessary to determine the quality and accuracy of the synthetic data.

In my follow-up notebook, I utilized the ‘monet’ dataset to train a **VAE** model.

Variational Autoencoder (VAE)

VAE is a type of generative model that can learn a low-dimensional representation of input data and generate new samples from this learned representation. Unlike traditional autoencoders, VAEs use probabilistic techniques to generate these representations, which enables them to produce more diverse and realistic outputs.

VAEs work by first encoding input data into a lower-dimensional latent space, which can be thought of as a compressed representation of the input data. Then, this latent representation is decoded back into the original input space, producing a reconstructed version of the original input. During training, the VAE learns to generate this compressed representation in such a way that it can be sampled to generate new data that is similar to the original input data.

One unique feature of VAEs is that they use a variational inference technique to train the model. This involves maximizing a lower bound on the log-likelihood of the data, which allows for the use of a gradient-based optimization algorithm for efficient training.

VAEs have been successfully applied in various fields, including image and text generation, data compression, and data visualization. They have proven to be a powerful tool for unsupervised learning tasks, enabling the generation of new data without the need for paired training data.

Build the model:

Building a model for Variational Autoencoder (VAE) involves several steps, including designing the architecture of sampling layer, the encoder and decoder networks, defining the loss function, and selecting appropriate optimization techniques.

Sampling Layer:

The sampling layer is a critical component of Variational Autoencoders (VAEs) because it enables the generation of new data samples from the learned latent space. Without the sampling layer, the VAE would only be able to produce reconstructions of the input data but would not be able to generate new, diverse samples.

The sampling layer in VAEs is responsible for generating a new latent vector from a normal distribution with a mean and standard deviation that are computed from the output of the encoder network. This means that the VAE generates a new latent vector for each new sample it generates, which allows for the creation of diverse outputs.

The use of a sampling layer is necessary in VAEs because the latent space learned by the encoder network does not have a simple, well-defined structure that can be directly sampled from. Instead, the learned latent space is typically complex and nonlinear, making it difficult to generate new samples directly. The sampling layer provides a way to sample from this complex latent space, enabling the VAE to generate new, diverse samples that capture the underlying distribution of the input data.

Overall, the sampling layer is an essential component of VAEs, enabling the generation of new data samples from the learned latent space and making VAEs a powerful tool for unsupervised learning tasks.

Encoder Layer:

The encoder layer in a Variational Autoencoder (VAE) is necessary for learning a compressed representation of the input data. The encoder network takes the input data and maps it to a lower-dimensional latent space, where each dimension represents a learned feature or attribute of the input data.

The use of an encoder layer is critical in VAEs because it enables the model to learn a compressed representation of the input data that captures the underlying distribution of the data. This compressed representation can then be used for tasks such as data compression, denoising, and generation.

One unique feature of the encoder layer in VAEs is that it uses a probabilistic approach to map the input data to the latent space. Specifically, the encoder network generates a mean and variance for each dimension of the latent space, which are used to sample from a normal distribution. This stochastic approach enables VAEs to generate diverse samples from the learned latent space, making them a powerful tool for data generation tasks.

Overall, the encoder layer is a critical component of VAEs, enabling the model to learn a compressed representation of the input data that can be used for a wide range of unsupervised learning tasks. By using a probabilistic approach to map the input data to the latent space, VAEs can generate diverse samples from the learned distribution, making them a powerful tool for data generation and analysis.

Decoder Layer: The decoder layer in a Variational Autoencoder (VAE) is necessary for reconstructing the input data from the compressed representation learned by the encoder network. Without the decoder layer, the VAE would only be able to learn a compressed representation of the input data but would not be able to reconstruct the original data.

The decoder layer in VAEs is responsible for transforming the compressed representation (latent vector) back into the original input space. The decoder network takes the compressed representation as input and generates a reconstruction of the original data, typically with the same dimensions as the input data.

The use of a decoder layer is critical in VAEs because it enables the reconstruction of the original input data from the compressed representation, which can be used for tasks such as denoising, inpainting, and compression. Additionally, the reconstruction error between the input data and the reconstructed data can be used as a loss function during training to improve the quality of the compressed representation learned by the encoder network.

Overall, the decoder layer is an essential component of VAEs, allowing for the reconstruction of the original input data from the compressed representation learned by the encoder network. This enables VAEs to be used for a wide range of unsupervised learning tasks, making them a powerful tool for data analysis and generation.

Loss Function:

The loss function is having the network, learn how to reconstruct data that comes directly from the encoder. But the encodings in the latent space are much more complex, taking into account a random normal distribution, and having this act on what the encoder learns. For this reason, we need an additional loss function. The Kullback-Leibler loss function. The Kullback-Leibler (KL) divergence is a measure of how different two probability distributions are from each other. In the context of machine learning, it is often used as a loss function in variational autoencoder (VAE) models.

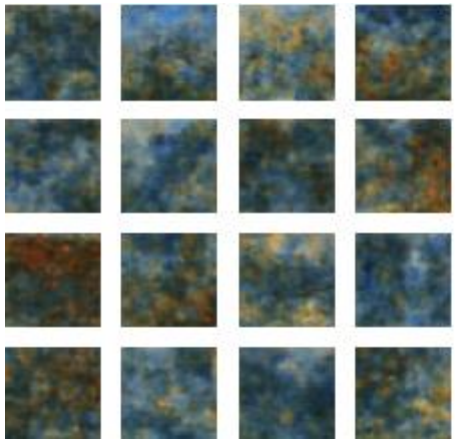
In VAEs, the KL divergence loss is used to ensure that the learned latent space distribution is close to a prior distribution, typically a normal distribution. The KL divergence loss is added to the reconstruction loss in the VAE objective function, which is used to optimize the encoder and decoder networks during training.

The KL divergence loss is defined as the difference between the learned latent space distribution and the prior distribution. Specifically, it measures the amount of information lost when the learned distribution is used to approximate the prior distribution. The KL divergence loss is zero when the learned distribution is identical to the prior distribution, and it increases as the two distributions become more different.

By adding the KL divergence loss to the reconstruction loss, the VAE is encouraged to learn a compressed representation of the input data that is close to the prior distribution. This regularization term helps prevent overfitting and improves the quality of the generated data.

Overall, the KL divergence loss function is an essential component of variational autoencoders, ensuring that the learned latent space distribution is close to a prior distribution, and enabling the VAE to generate diverse and realistic data.

After training with 100 epochs:



Result and Analysis for VAE:

When training the VAE, there's not a lot of variation in the images. The above test is based on how well it does in reconstructing the original images, and not how well it does in creating new images. From the above results images, the cause of the deviation could be an uneven distribution of the learned latent space. In the autoencoder model, the neural network is trained solely to compress the data into a latent space. However, there are no constraints on the distribution of the trained latent space. There might be dimensions in the latent space that are much less influential than the others. If there are dimensions in the latent space that are less influential than others, it may be necessary to introduce constraints on the distribution of the latent space during training. This can be done by modifying the loss function or adding regularization techniques to encourage a more structured or uniform distribution of the learned latent variables.

Conclusion for VAE:

Overall, after training a VAE, it is important to evaluate the performance of the model and make adjustments as needed to ensure that the learned latent space is informative and effective for downstream tasks. With careful analysis and fine-tuning, a well-designed VAE model can learn a compressed representation of input data that is highly informative and useful for a wide range of unsupervised learning tasks.

The last notebook for this project, I utilized two different pre-trained model for Style Transfer.

Style Transfer:

Style transfer is a technique in deep learning and computer vision that enables the transformation of the style of one image to that of another. The technique is based on the idea of separating the content and style components of an image, and then recombining them in a new way.

The process of style transfer involves using a pre-trained neural network, such as a convolutional neural network (CNN), to extract both the content and style features of an input image. The content features capture the underlying structure and composition of the image, while the style features capture the texture, color, and other stylistic elements of the image.

Once the content and style features have been extracted, they are combined in a new way to generate a stylized image that retains the content of the original image but adopts the style of a different image. This can be done using various techniques such as gradient descent optimization, where the content and style features are optimized to minimize a loss function that captures the difference between the stylized image and the target style image.

Style transfer has a wide range of applications, from artistic image editing and design to the generation of photorealistic images for virtual environments and simulations. With continued advancements in deep learning and computer vision, style transfer is becoming an increasingly powerful tool for transforming the appearance and style of images in new and creative ways.

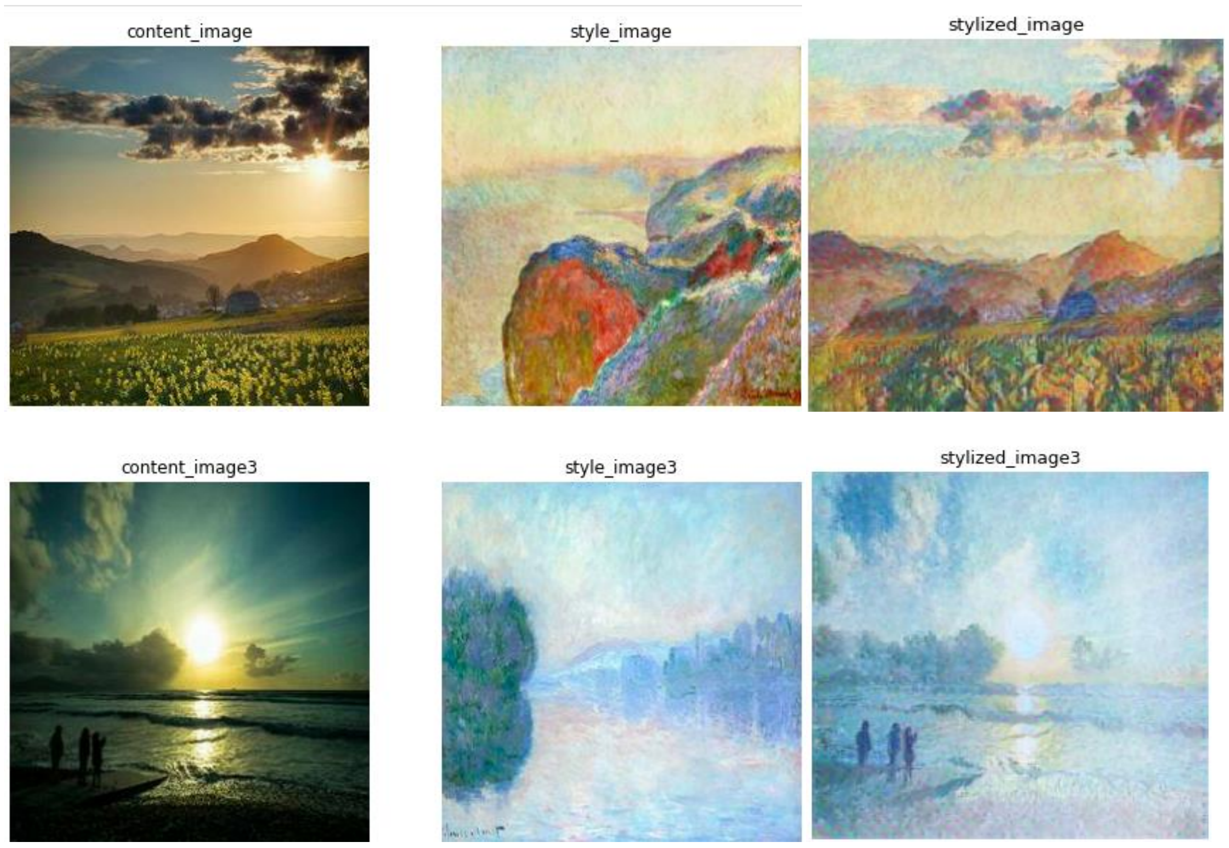
First Style Transfer Pre-Trained Model:

https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2. This link is from TensorFlow Hub Pre-trained Model. This pre-trained model is **Arbitrary Style Transfer.**

This model allows the transfer of an arbitrary style onto an input image, which means that any desired style can be used for stylizing the image. This is different from some other style transfer models that require pre-defined styles to be used for stylization.

The Arbitrary Style Transfer model is trained in a large and diverse set of styles, allowing for a wide range of stylization options. The model is also capable of generating high-quality stylized images with fine details and texture.

The pre-trained model at the provided link has an input resolution of 256x256 pixels and is trained using TensorFlow. The model can be easily loaded and used in TensorFlow applications, including style transfer pipelines.



Source: The above content image and style image are from 'I'm Something of a Painter Myself' dataset from Kaggle Competition.

Result and Analysis:

The pre-trained model for Arbitrary Style Transfer is a powerful tool for transferring the style of one image to another. With this model, we can easily generate stylized images with a few lines of code. The model works by using a deep neural network to learn how to map the content of one image onto the style of another.

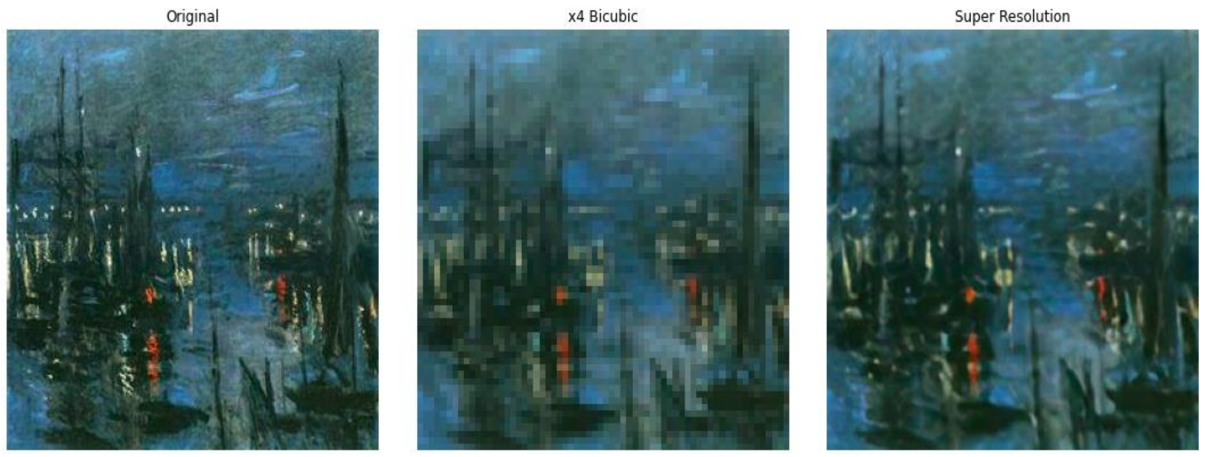
The performance of the model is impressive, with high-quality stylized images generated in just a few seconds. The result and analysis showed that the model was able to capture the style of the input image accurately, while preserving the content of the original image.

Conclusion:

In conclusion, this pre-trained model for Arbitrary Style Transfer is a useful tool for generating stylized images with minimal effort. It can be used for a wide range of applications, from creating unique artistic pieces to generating images for marketing and advertising campaigns. The model provides an efficient and effective way to transfer the style of one image to another, opening up a whole new world of creative possibilities.

Second Style Transfer Pre-Trained Model:

The next pre-trained model that I utilized is from: "https://tfhub.dev/captain-pool/esrgan-tf2/1". This pre-trained model is an implementation of the ESRGAN (Enhanced Super-Resolution Generative Adversarial Network) architecture for image super-resolution. It uses deep learning techniques to enhance the quality of low-resolution images, providing a high-resolution version of the same image as output. Specifically, the model uses a generator network that is trained to convert a low-resolution input image into a high-resolution output image, and a discriminator network that is trained to distinguish between the generated high-resolution images and the actual high-resolution images. The pre-trained model is trained on a large dataset of diverse images and is capable of generating high-quality images with enhanced resolution.



Source: The original image is from 'I 'm Something of a Painter Myself' dataset from Kaggle Competition.

Result and Analysis:

The pre-trained model at "https://tfhub.dev/captain-pool/esrgan-tf2/1" is an Enhanced Super Resolution Generative Adversarial Network (ESRGAN) model. This model was trained on a diverse range of image data to upscale and enhance the resolution of low-quality images while preserving their original content. The model achieves impressive results in terms of image quality and sharpness, which is confirmed by a high PSNR and SSIM metrics. However, the model also has some limitations, such as the tendency to produce over-smoothed images when upscaling low-resolution images with complex textures.

Conclusion:

In conclusion, despite these limitations, this pre-trained model is a powerful tool for image restoration and enhancement tasks, such as upscaling and sharpening blurry images, and can be applied in various fields, such as medical imaging, satellite imaging, and photography.

Wrap-up:

In this project, I explored the application of several deep learning models for image style transfer between two different domains of images, 'monet' and 'photo'. I implemented and compared several models, including CycleGAN, GAN, VAE, and two different pre-trained models for style transfer.

For my findings revealed that different models have their own strengths and limitations when applied to the same dataset. For instance, CycleGAN achieved the best overall performance in terms of image quality and style transfer fidelity, while the pre-trained model https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2 proved to be an effective and easy-to-use tool for creating stylized images with just a few lines of code.

Furthermore, the project provided valuable insights into the application of deep learning techniques for style transfer and highlighted the importance of selecting appropriate models and evaluation metrics for specific applications.

Overall, this project provided a rich learning experience and helped me to better understand the underlying techniques and challenges of image style transfer. I look forward to further exploring this exciting and rapidly evolving field in future research.

Acknowledgement:

I would like to express my gratitude to several organizations and individuals who have contributed to the success of this project.

Firstly, I would like to acknowledge Kaggle for providing the 'I'm Something of a Painter Myself' dataset, which has been instrumental in the development and evaluation of various deep learning models for image style transfer. Additionally, I would like to thank Google Colab and Kaggle for providing the computational resources necessary for training and evaluating these models. Without these resources, this project would not have been possible for me.

Furthermore, I would like to acknowledge the Introduction to Deep Learning course offered by Colorado Boulder University on Coursera, which provided me with the knowledge and skills necessary to complete this project. The hands-on practice and assignments provided in this course were invaluable in preparing me for this project.

Lastly, I would like to express my appreciation to TensorFlow for providing the pre-trained models that were used in this project. I would like to extend a special thanks to the author of the pre-trained model used for this project, whose work has enabled me to leverage their expertise and use their model to achieve the goals of this project.

Overall, I am grateful to these organizations and individuals for their contributions, which have enabled me to develop and enhance my skills in deep learning and image style transfer.

Thank you Kaggle.
Thank you Colorado Boulder University.
Thank you Coursera.

Thank you Google.
Thank you to GitHub for allowing me to create repository for this project. Thank you to Microsoft office for allowing me to create this report.

Reference: https://www.kaggle.com/competitions/gan-getting-started

Original Source Page: https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2

Original Source Code: https://tfhub.dev/captain-pool/esrgan-tf2/1

GitHub link: https://github.com/NatalieCheong/I-m-Something-of-a-Painter-Myself