STUDY PROJECT

MID-SEMESTER REPORT

PROFESSOR IN-CHARGE

PROF. SUREKHA BHANOT

SUBMITTED BY

ABHISHEK LALWANI 2014A3PS0257P

NIKHIL VINAY SHARMA 2014A3PS0262P

A REPORT

ON

ARTISTIC STYLE TRANSFER USING NEURAL NETWORKS

BY

ABHISHEK LALWANI 2014A3PS0257P

NIKHIL VINAY SHARMA 2014A3PS0262P

PREPARED IN PARTIAL FULFILMENT OF THE

STUDY ORIENTED PROJECT COURSE

2ND SEMESTER 2017-2018

INTRODUCTION

The project revolves around reviewing different computer vision techniques for extracting style and content from artistic images using neural networks and synthesizing new images using content from one input image and style from another input image. The style of the resulting image should be similar to the input image from which style is extracted and the content of the resulting image should be similar to the input image from which content is extracted.

We plan to use densely connected Convolutional Neural Networks in this for feature extraction because of their state-of-the-art performance on other tasks of Image Processing and Computer Vision.

The overall aim of the project is to try out various methods for achieving successful style transfer and defining and improving a universal error factor which can be used to analyze the quality of the technique used for style transfer.

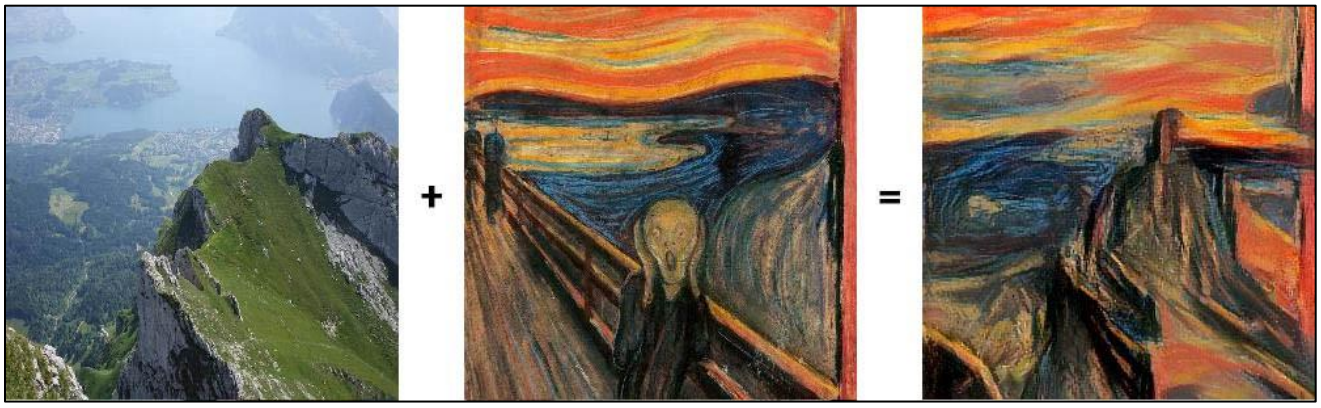
Here, in this report we study and analyze different breakthrough papers in this field of Artistic Neural Style Transfer, discussing their particular breakthrough and giving their corresponding results along the way.

The project was implemented using Keras with Tensorflow backend on python unless otherwise mentioned.

The real life applications of the process involve,

1. Instantly stylizing input images or videos for a particular style.
2. Used in photo editing applications like Prisma, Artisito and Deepart.io.
3. Used in industry leading softwares like Adobe Photoshop, Adobe Lightroom, Adobe Premiere Pro/After Effects, CorelIDRAW, DaVinci Resolve, Autodesk Maya etc.

In the end, we provide a concise conclusion on the current challenges that the present state of the art research techniques face.



From Left to Right, Content Image, then style image and then final style transferred output.

CONVOLTIONAL NEURAL NETWORKS (CNNs/ConvNets)

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer and all the properties we know about regular Neural Networks still apply.

ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Convolutional neural networks are biologically inspired variants of multilayer perceptrons, designed to emulate the behaviour of a visual cortex. These models mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images. As opposed to MLPs, CNNs have the following distinguishing features:

3D volumes of neurons: The layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. The neurons inside a layer are connected to only a small region of the layer before it, called a receptive field. Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.

Local connectivity: Following the concept of receptive fields, CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.

Shared weights: In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting the property of translation invariance.

Together, these properties allow CNNs to achieve better generalization on vision problems. Weight sharing dramatically reduces the number of free parameters learned, thus lowering the memory requirements for running the network and allowing the training of larger, more powerful networks.

LOSS FUNCTIONS

A loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function. Here we describe the popular loss functions used in for optimization:

- **Mean Squared Error**
  This measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated.
  $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
- **Cross Entropy Loss**
  Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label, i.e., a perfect model would have log loss as 0.
  $$H(p, q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y) \log (1-\hat{y})$$
- **Hinge Loss**
  The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs). For an intended output t = ±1 and a classifier score y, the hinge loss of the prediction y is defined as
  $$\ell(y) = \max(0, 1 - t \cdot y)$$
- **Absolute Loss**
  This is the sum of difference of absolute numbers of two place.

For our purposes we'll be using Mean Squared Error only.

METHODOLOGY [1]

As mentioned in the previous topic, we plan to use CNNs (Convolutional Neural Networks) for the task of extracting relevant features from our images. Normally, the image is processed by CNNs and the feature maps generated in the subsequent layers are analyzed for objects of interest. Here, as our task is of Image Synthesis, we use the reverse methodology.

1. We start with feeding a randomized image to our network of CNNs. This randomized image is going to be the image in which our target image will be synthesized.
2. While our image is going through the network, we keep comparing the content difference with the content image, and style difference with the style image.

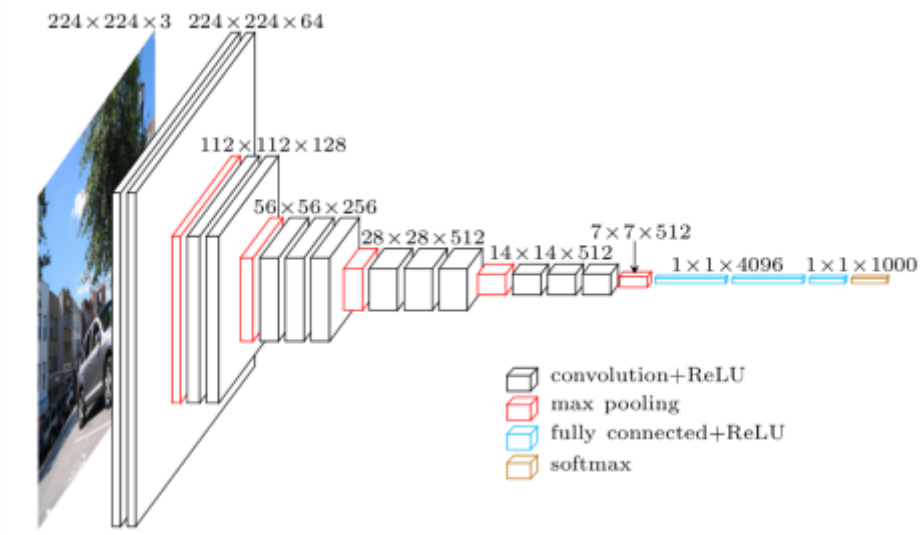
In the overall process, we define and derive an overall loss measure used for combining the content difference and the style difference of the synthesized image with that of our input images. We further apply gradient descent to minimize this error, which results into our synthesized image.

NEURAL ARCHITECTURE [2]

Normally, deep neural networks take weeks to train if we try and train them from scratch. A much better option is to use the pre-trained model which is trained to perform basic operations but can be further trained and fine-tuned for any precise task (Style transfer in our case). For this, we plan to use VGG-19, which stands for Very Deep Convolutional Networks for Large-Scale Image Recognition. It is a 19-layer deep Convolutional Neural Network which is the current State-of-the-art model for Computer Vision and Feature Extraction tasks.

Here, we only use the first 16 neural layers of the network (VGG-16) as these are the ones which are useful for Feature Extraction, the last three layers are used only for classification purposes.

As mentioned previously we feed our randomized image to this model and try and minimize the error across all the layers.



VGG-16

ANALYSIS IN VGG [2]

Once we feed our image into a convolutional neural network, 3 steps keep repeating again and again over every layer.

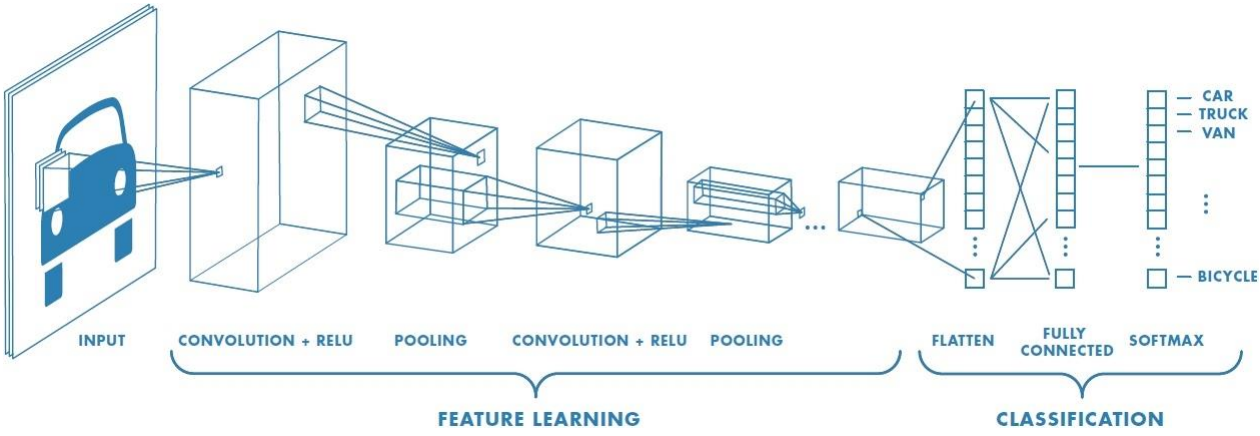
1. Convolution

2. Activation

3. Pooling

A3x3 filter slides over the entire image and convolves with the whole image. The convolved image thus generated is acted upon by the activation function. This results into what we call as the feature map. Using feature maps, we analyze our object of interest in the input image.

This is followed by pooling where the set of values is mapped to one value, which is derived from all those input values. This helps in size reduction of the generated feature maps.



Analysis of an Image using a Convolutional Neural Network

ERROR MEASURE [1]

The error measure which we plan to use will encompass the content error and the style error respect to the content image and the style image respectively. So we first separately define content error and style error and combine the two using a linear equation. The content error is defined as follows-

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

F^{l}_{ij} is the activation of the i^{th} filter at position j in layer l in the generated image.

P^{l}_{ij} is the activation of the i^{th} filter at position j in layer l in the content input image.

Similarly, the style error is defined as follows-

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

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

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

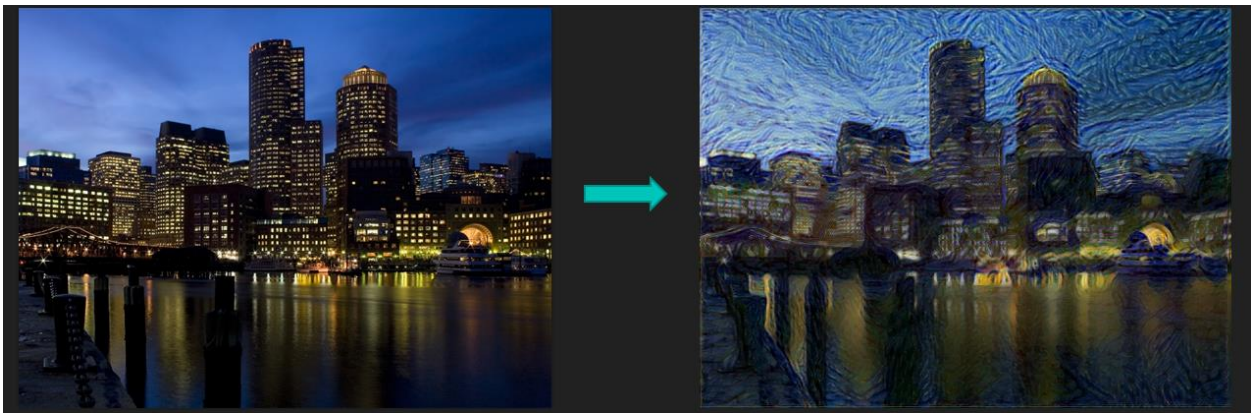
The combined error can be defined as-

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

Where ‘p’ is the photograph (content image), ‘a’ is the artwork (style image) and ‘x’ is our synthesized image.

Now, we try and minimize the error using optimization techniques, **stochastic gradient descent**.

RESULTS

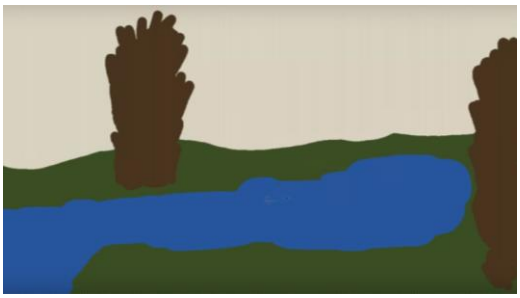


IMPROVEMENT USING SEMANTIC MEASURES [3]

Although the above method is pretty decent, the glitches generated while using the above method is pretty high. This is because the network is not able to recognize the discrete areas in a typical portrait such as hair, face, clothes and the background. It ends up merging those areas and this results into what is known as glitch art. Although this kind of glitch art also has its own place, but it is very less desirable as compared to the expected results. To improve upon this, we introduce the concept of semantic labels in the picture.

The semantic nature of an image is basically labelling each and every pixel in terms of what it represents. An example of a Semantic map is given below along with the original image.

Original Image



Semantic Map

Now, if we can somehow use the sharp boundaries and areas defined in the semantic map then we can improve upon the performance of the Style transfer model which we have developed up till now. For this, we concatenate the feature maps with a down-sampled semantic map as shown in the figure. The rest of the analysis remains the same in terms of error measure and the optimization technique used.

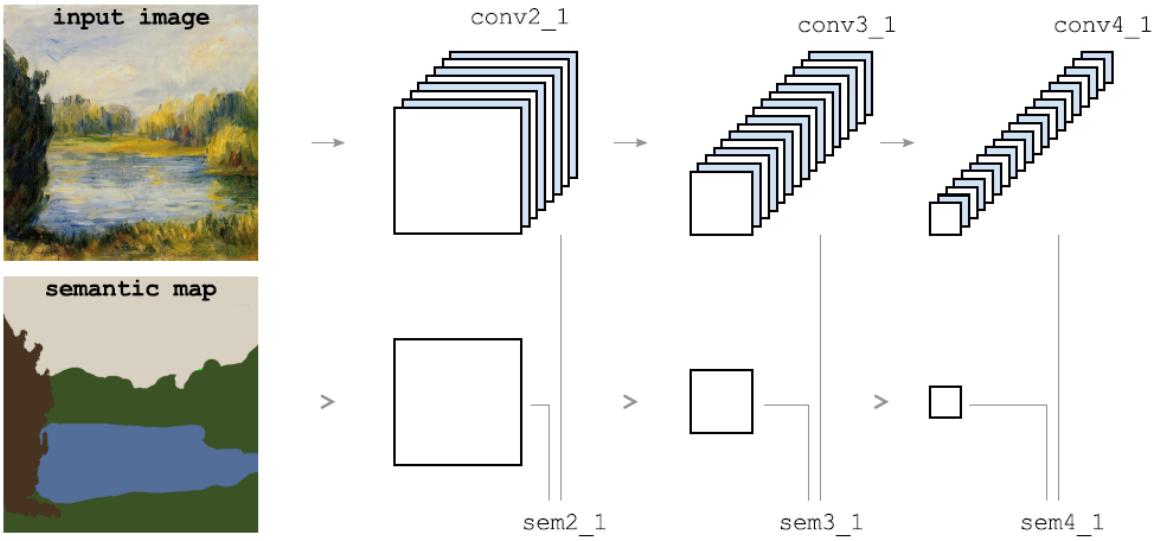
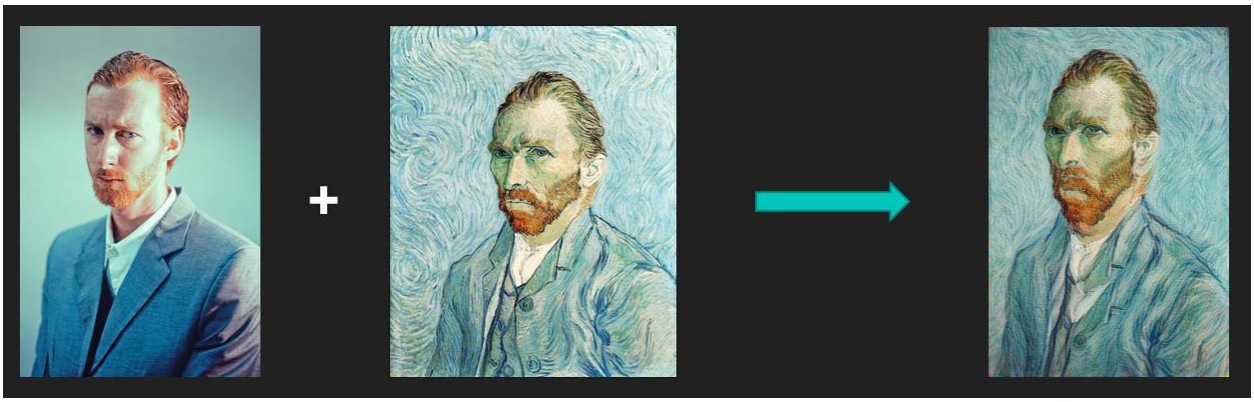
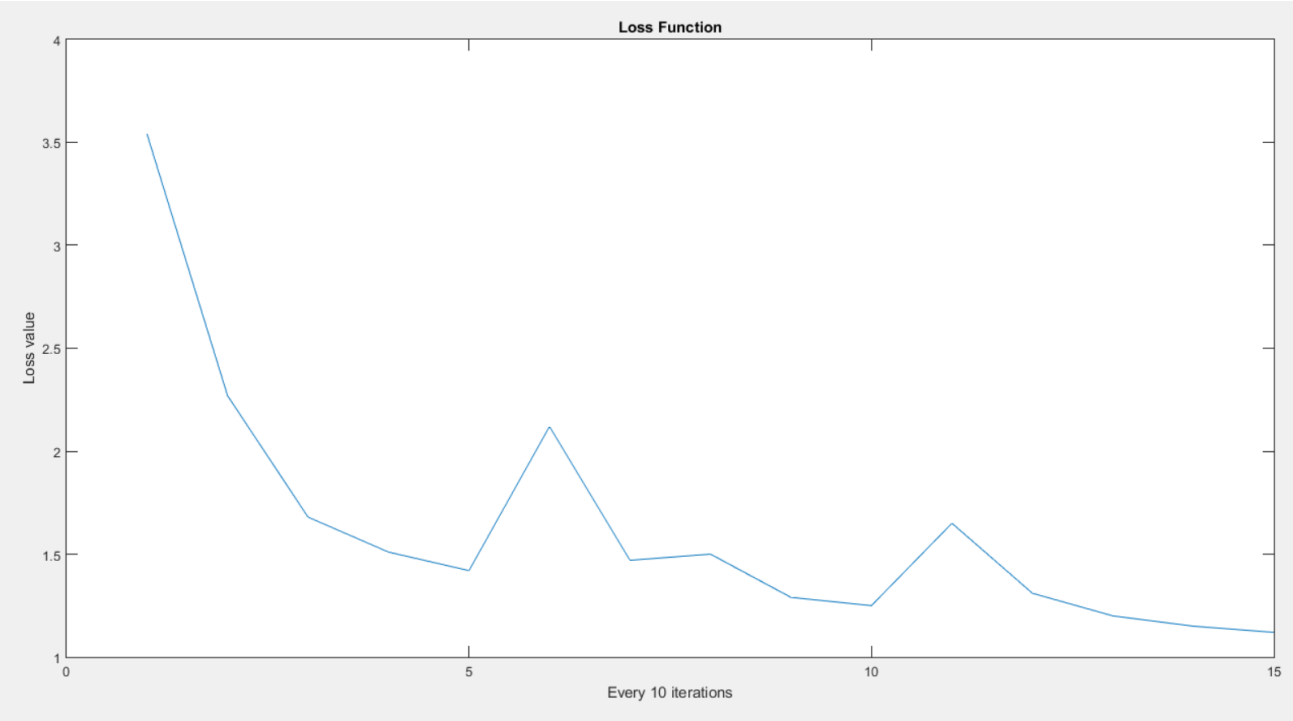


Figure 3: Our augmented CNN that uses regular filters of N channels (top), concatenated with a semantic map of M=1 channel (bottom) either output from another network capable of labeling pixels or as manual annotations.

RESULTS



GRAPH OF THE LOSS FUNCTION



IMPROVING THE NEURAL TRANSFER ALGORITHM [4]

While showing great results when transferring homogeneous and repetitive patterns, the original style representation by Gatys *et al* [1] often fails to capture more complex properties, like having separate styles of foreground and background. This leads to visual artifacts and undesirable textures appearing in unexpected regions when performing style transfer.

Further, when visually inspecting style transfer results and commenting on the quality, we use two complimentary criteria that we expect a good style transfer algorithm to meet:

1. Similar areas of the content image should be repainted in a similar way.
2. Different areas should be repainted differently.

As it turns out, it is often difficult to satisfy both simultaneously.

The paper states a lot of improvements to the older algorithm, the most useful ones are stated below.

Modifications in the original Algorithm

1. **Layer Weight Adjustment**
   A better per-layer content style weighting scheme for style and content losses are given by –

   $$w_l^s = 2^{D-d(l)}, \quad w_l^c = 2^{d(l)},$$

   Where D is the total number of layers used and d(l) is the depth of layer 'l' with respect to all other layers used. This is done to preserve more information since content is represented mostly by the shallow layers and style is represented by the deeper layers.
2. **Using More Layers**
   In [1] only five layers are used for the calculation of Gram Matrices, here we use all 16 layers for the calculation.
3. **Activation Shift**
   This is based on the fact that sparsity is detrimental to style transfer. VGG -19's normalized CNN layers output non negative activations with mean +1. For typical image outputs are also sparse in all layers, each filters have usually few non-zero activations across the spatial dimensions (matrix). This results in sparse Gram Matrices. This allows unexpected patterns to appear in regions one would typically expect to be filled with uniform background colour. Hence the equation for Gram Matrix,

   $$\mathcal{G}^l = F^l F^{l^T},$$

   Is changed to,

   $$\mathcal{G}^l = (F^l + s)\,(F^l + s)^T,$$

   Where s is the shift value added to matrices element-wise. Putting s=-1, i.e., centering the activations at 0, gives best results.

In this case, the gradient contributions are changed as follows,

$$\frac{\partial \mathcal{G}^l}{\partial F^l} = 2(F^l + s).$$

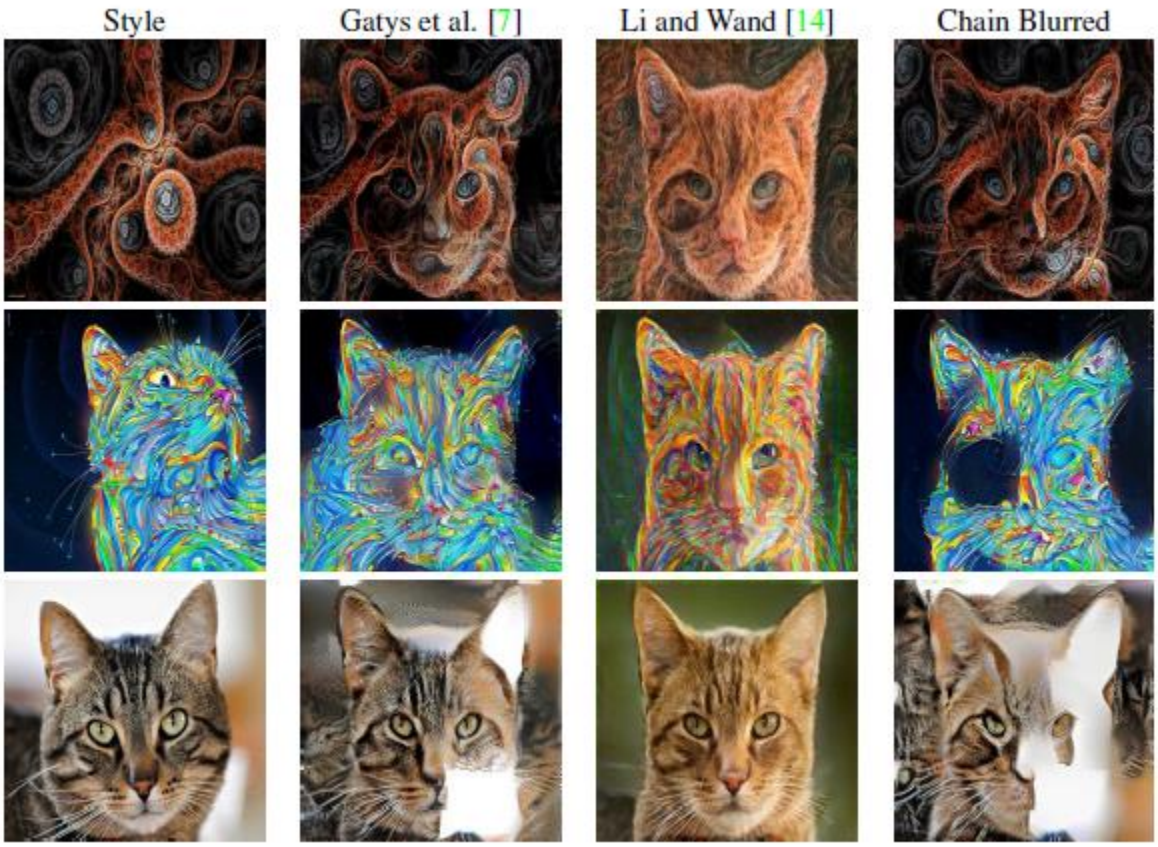
This eliminates the ambiguity of zero entries in Gram matrices and improves results while accelerating convergence across different style images and style transfer methods, i.e., faster optimization. Furthermore, this help in better distinguishing between the foreground and the background.

4. **Correlations of Features from Different Layers**
   We incorporate the use of Gram Matrices to get feature correlations belonging to possibly different layers l and k.

   $$\mathcal{G}^{lk} = F^l \left[\text{up}(F^k)\right]^T,$$

   This turned out to be not that effective in improving the process.
5. **Correlation Chain**
   We constrain correlation of high and low-level features, but in a local way, where only the correlation with immediate neighbors are considered. This approach has led to a consistent a consistent and often significant improvement in style transfer quality in most cases considered.

A comparison can be made from the following, where a picture of a cat was used as content image –



PERCEPTUAL LOSSES FOR REAL-TIME STYLE TRANSFER & SUPER RESOLUTION [5]

While showing great results whilst transferring style Gatys *et al*'s algorithm takes around 6-8 hours typically on a CPU for 512x512 image. Here is proposed a more developed method by Johnson *et al* using which one can transfer art to any image in real time. The proposed algorithm is 1000x times faster. We train feed-forward transformation networks for image transformation tasks, but rather than using per-pixel loss functions depending only on low-level pixel information, we train our networks using perceptual loss functions that depend on high-level features from a pre-trained loss network. During training, perceptual losses measure image similarities more robustly than per-pixel losses, and at test-time the transformation networks run in real-time.

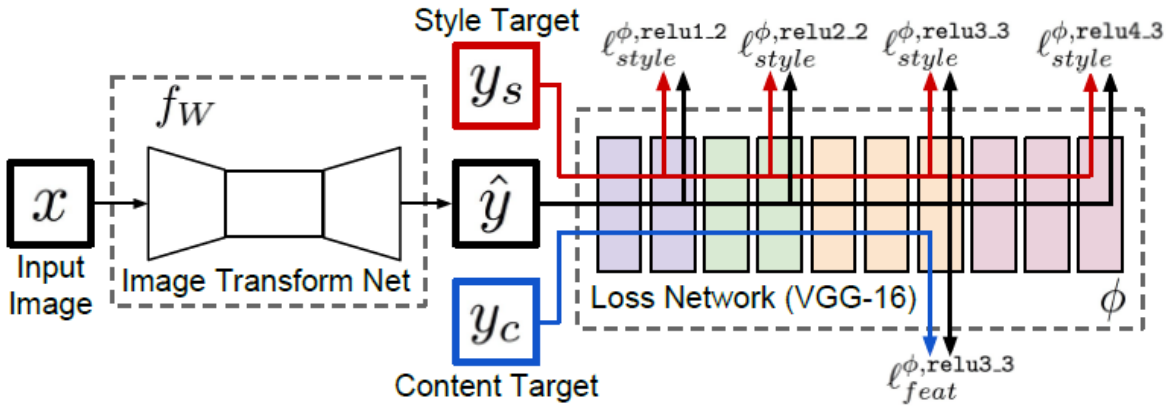


Fig. 2. System overview. We train an *image transformation network* to transform input images into output images. We use a *loss network* pretrained for image classification to define *perceptual loss functions* that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.

METHOD

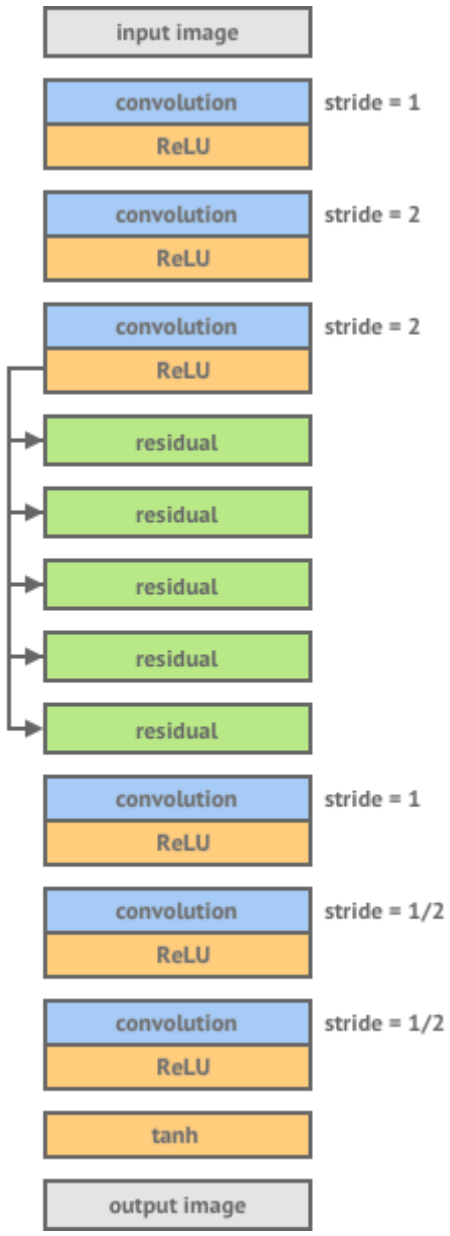
Our system consists of two components: an image transformation network f_W and a loss network that is used to define several loss functions $l_1; l_2; : : ; l_k$. The image transformation network is a deep residual convolutional neural network parameterized by weights W; it transforms input images x into output images ^y via the mapping $\hat{y} = f_W(x)$. Each loss function computes a scalar value $l_i(\hat{y}; y_i)$ measuring the difference between the output image ^y and a target image y_i. The image transformation network is trained using stochastic gradient descent to minimize a weighted combination of loss functions:

$$W^* = \arg\min_W \mathbf{E}_{x,\{y_i\}} \left[\sum_{i=1} \lambda_i \ell_i(f_W(x), y_i) \right]$$

The key insight of these methods is that convolutional neural networks pre-trained for image classification have already learned to encode the perceptual and semantic information we would like to measure in our loss functions. We therefore make use of a network (VGG-16) which has been pre-trained for image classification as a fixed loss network in order to define our loss functions.

Image Transformation Networks

1. **Inputs and Outputs.**
   For style transfer the input and output are both color images of shape 3x256x256.
2. **Downsampling and Upsampling**.
   For style transfer our networks use two stride-2 convolutions to downsample the input followed by several residual blocks and then two convolutional layers with stride 1/2 to upsample. Although the input and output have the same size, there are several benefits to networks that downsample and then upsample.
   The first is computational. After downsampling, we can therefore use a larger network for the same computational cost. The second benefit has to do with effective receptive field sizes. High-quality style transfer requires changing large parts of the image in a coherent way; therefore it is advantageous for each pixel in the output to have a large effective receptive field in the input.
3. **Residual Connections**. We use residual connections to train very deep networks for image classification. We argue that residual connections make it easy for the network to learn the identify function; this is an appealing property for image transformation networks, since in most cases the output image should share structure with the input image.



Perceptual Loss Functions

1. **Feature Reconstruction Loss**
   The feature reconstruction loss is the (squared, normalized) Euclidean distance between feature representations of content image and transformed image:

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

2. **Style Reconstruction Loss**
   Calculated in the same way as Gatys *et al,* we take the difference in Gram matrices of feature representations of transformed image and the style image.

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

And the loss function is given by,

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

TRAINING THE NETWORK

For training the network, Microsoft Coco Dataset was used (a dataset of 80,000 annotated images), which were rescaled to 256x256 and using one particular style image, style was transferred to each of these images using Gatys et al's original method. Afterwards, the image transformation network was trained on these images using the VGG-16 loss network. We trained our networks with a batch size of 4 for 40,000 iterations, giving roughly two epochs over the training data. The output images by the network are close to resembling the ones by Gatys, but is 1000x faster.

HYPERPARAMETERS USED

We used Adam with a learning rate of 1×10^{-3}. for optimization instead of using Stochastic Gradient Descent as before, since Adam is able to perform faster than and better in this case. The output images are regularized with total variation regularization with a strength of between 1 x 10-6 and 1 x 10-4, chosen via cross-validation per style target. We do not use weight decay or dropout, as the model does not overt within two epochs.

RESULTS

A LEARNED REPRESENTATION FOR ARTISTIC STYLE [6]

In this work, we investigate the construction of a single, scalable deep network that can capture the artistic style of a diversity of paintings. The paper demonstrates that such a network generalizes across a diversity of artistic styles by reducing a painting to a point in an embedding space. Importantly, this model permits a user to explore new painting styles by arbitrarily combining the styles learned from individual paintings.

The paper builds upon the technique proposed by Gatys et al, of minimizing style and content losses.

We used Tensorflow's Magenta module developed by the Google Brain Team for training and testing.

METHOD

This work stems from the intuition that many styles probably share some degree of computation, and that this sharing is thrown away by training N networks from scratch when building an N- styles style transfer system. The paper proposes to train a single conditional style transfer network *T(c, s)* for N styles. The conditional network is given both a content image and the identity of the style to apply and produces an output corresponding to that style.

This approach is referred to as *conditional instance normalization*. The goal of the procedure is transform a layer's activations **x** into a normalized activation **z** specific to painting style **s**. Building off the instance normalization technique proposed in Ulyanov et al. [9], we augment the γ and β parameters so that they're *N* x *C* matrices, where N is the number of styles being modeled and C is the number of output feature maps. Conditioning on a style is achieved as follows:

$$z = \gamma_s \left(\frac{x - \mu}{\sigma} \right) + \beta_s$$

where μ and σ are x's mean and standard deviation taken across spatial axes and γ_s and β_s are obtained by selecting the row corresponding to s in the γ and β matrices.

One added benefit of this approach is that one can stylize a single image into N painting styles with a single feed forward pass of the network with a batch size of N. In contrast, a previously mentioned single-style networks require N feed forward passes to perform N style transfers.

TRAINING THE NETWORK

The architecture used is the same as Johnson et al.[5] except for two major changes –

- **Zero-padding is replaced with mirror-padding**
  The use of mirror-padding avoids border patterns sometimes caused by zero-padding in SAME-padded convolutions.
- **Replacement of Transposed convolutions layers**
  Transposed convolutions (also sometimes called deconvolutions) are replaced with nearest-neighbor upsampling followed by a convolution. The replacement for transposed convolutions avoids checkerboard patterning.

The ImageNet (256x256x3 size) dataset is used for training using the Adam optimizer.

RESULTS

We trained the model on ten different painting styles of Monet's painting hoping to capture his impressionist style of painting. We showcase few of the outputs here.

Claude Monet, Grainstacks at Giverny; the Evening Sun

Claude Monet, Plum Trees in Blossom

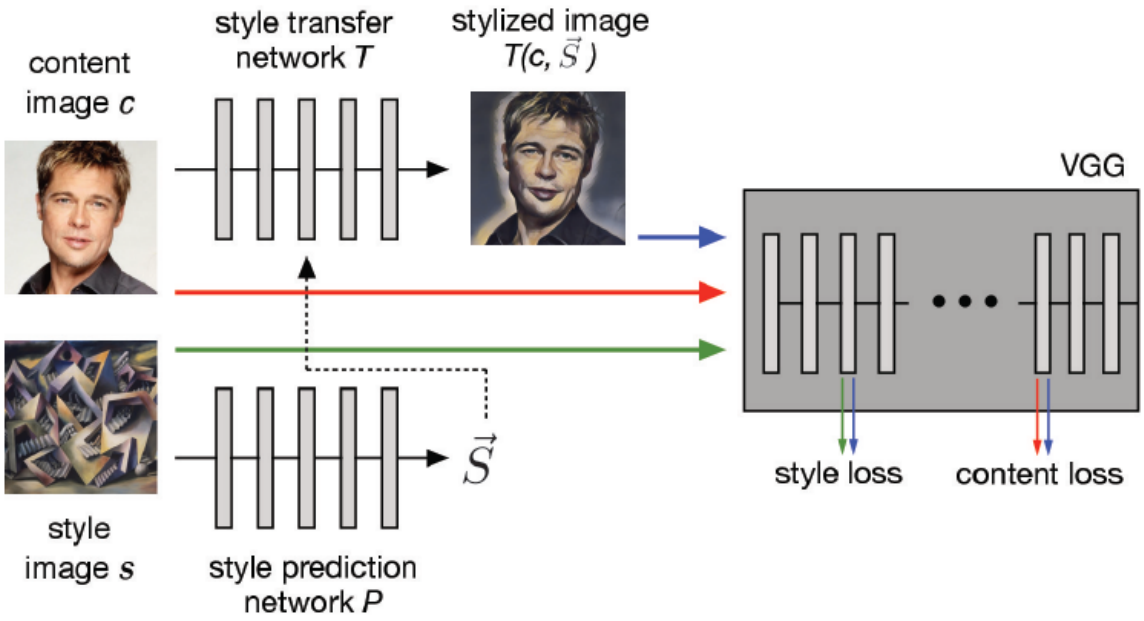
Claude Monet, Poppy Field

Claude Monet, Sunrise (Marine)

EXPLORING THE STRUCTURE OF A REAL-TIME, ARBITRARY NEURAL ARTISTIC STYLIZATION NETWORK [7]

This research paper proposes a method which combines the flexibility of the neural algorithm of artistic style with the speed of fast style transfer networks to allow real-time stylization using any content/style image pair.

METHOD



This paper builds heavily upon the work done in [6] and extends the concept of conditional instance normalization. This linear transformation mentioned in [6] is unique to each painting style s. In particular, the concatenation -

$$\vec{S} = \{\gamma_s, \beta_s\}$$

constitutes a roughly 3000-d embedding vector representing the artistic style of a painting. We denote this style transfer network as T(C, S).

We propose a simple extension in the form of a style prediction network P (.) that takes as input an arbitrary style image s and predicts the embedding vector ~S of normalization constants, as illustrated in the figure above. The crucial advantage of this approach is that the model can generalize to an unseen style image by predicting its proper style embedding at test time.

For predicting the Style Vector (Style Prediction Network), we use InceptionNet-v3 network architecture with two extra layers connected at the end and compute mean across each activation channel which returns a feature vector of dimension of approximately 750. Then we apply two fully connected layers on top of it to predict the final embedding ~S. Training of both the networks takes place simultaneously and hence takes a huge amount of time to train even on GPUs.

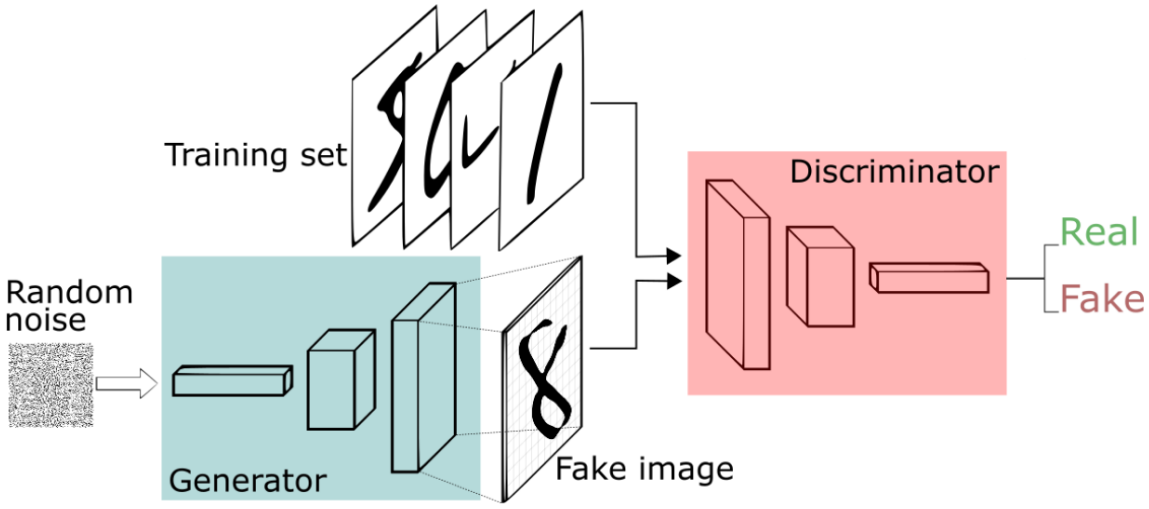
GENERATIVE ADVERSARIAL NETWORKS [10]

Generative Adversarial Networks, as the name suggests, are generative networks which are used for generating content which belongs to particular pre-defined class. For understanding the functionality of GANs, we first need to understand the terms *generative* and *discriminative* networks.

Discriminative Algorithms are basically classification algorithms which try and attach a label to input data, depending upon the features of the input data. For example, a spam detection algorithm will analyze an email and depending upon the words occurring in the email (features), it will assign a label of 'spam' or 'not-spam' depending upon the extracted features.

Generative algorithms work in the reverse direction as compared to Discriminative Algorithms. On being given a label, the generative algorithm will try and generate features which represent that particular label. In other words, they try and predict the probability of a particular feature being there in the data, given a label as input. So, they try and generate the data corresponding to a label.

Now, what happens in a typical GAN is, a generative network and a discriminative network are pitted against each other. The generative network or the *generator* tries to generate instances which the discriminative network or the *discriminator*, tries to classify as fake or real, depending upon the training set over which our discriminator is trained. As it can be seen, both the networks simultaneously try and get better at their task. The generator tries to generate better instances which are as close to the training set as possible whereas the discriminator tries to improve the classification accuracy. This simultaneous training of both the networks makes it relatively difficult to train a GAN as compared to a normal Neural Network. An example consisting of the MNIST dataset is as shown below.



Here, generator tries to generate an image of a hand-written number similar to MNIST dataset and the discriminator tries to identify that as fake or real

The loss function of a typical is defined as follows [10] -

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

This expression is for the most basic form of GAN using multilayer perceptron model as given in the paper titled “Generative Adversarial Nets” by Ian J.Goodfellow from NIPS 2014.

Here, E represents the expected value of the expression.

D represents the Discriminator network/function. In other words, it represents the probability of the input being from the original data rather than being an output of the generator network.

G represents the Generator network/function.

Using this Loss function, we train the discriminator to correctly identify a real or a fake input depending upon its probability of belonging to the original data distribution rather than being generated from the generator.

For training, we alternate between ‘k’ steps of optimizing D (Discriminator) and one step of optimizing G (Generator).

The global minimum for optimizing a GAN is at the point where the probability of the input predicted by D is equal to 0.5, or in other words the probability of the input being from the dataset and being generated by the generator is equal [10].

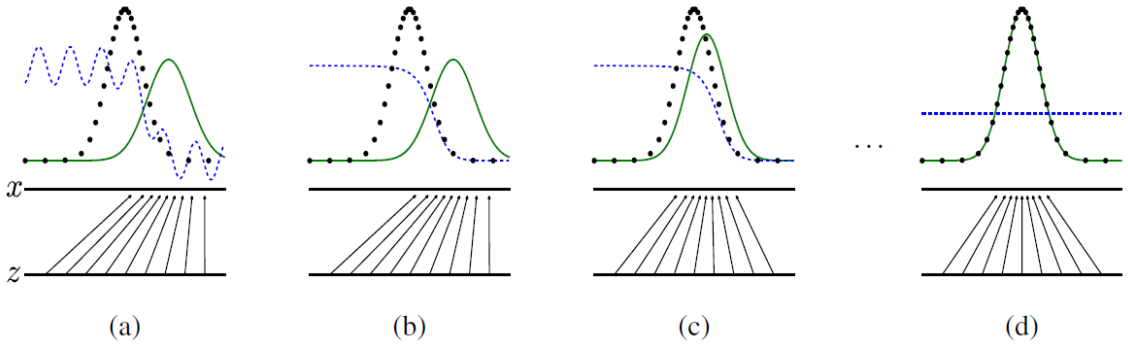
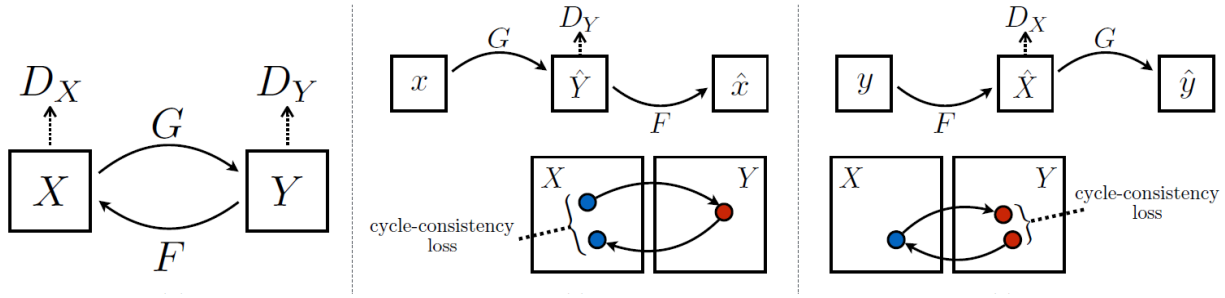


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

CYCLE GANs [11]

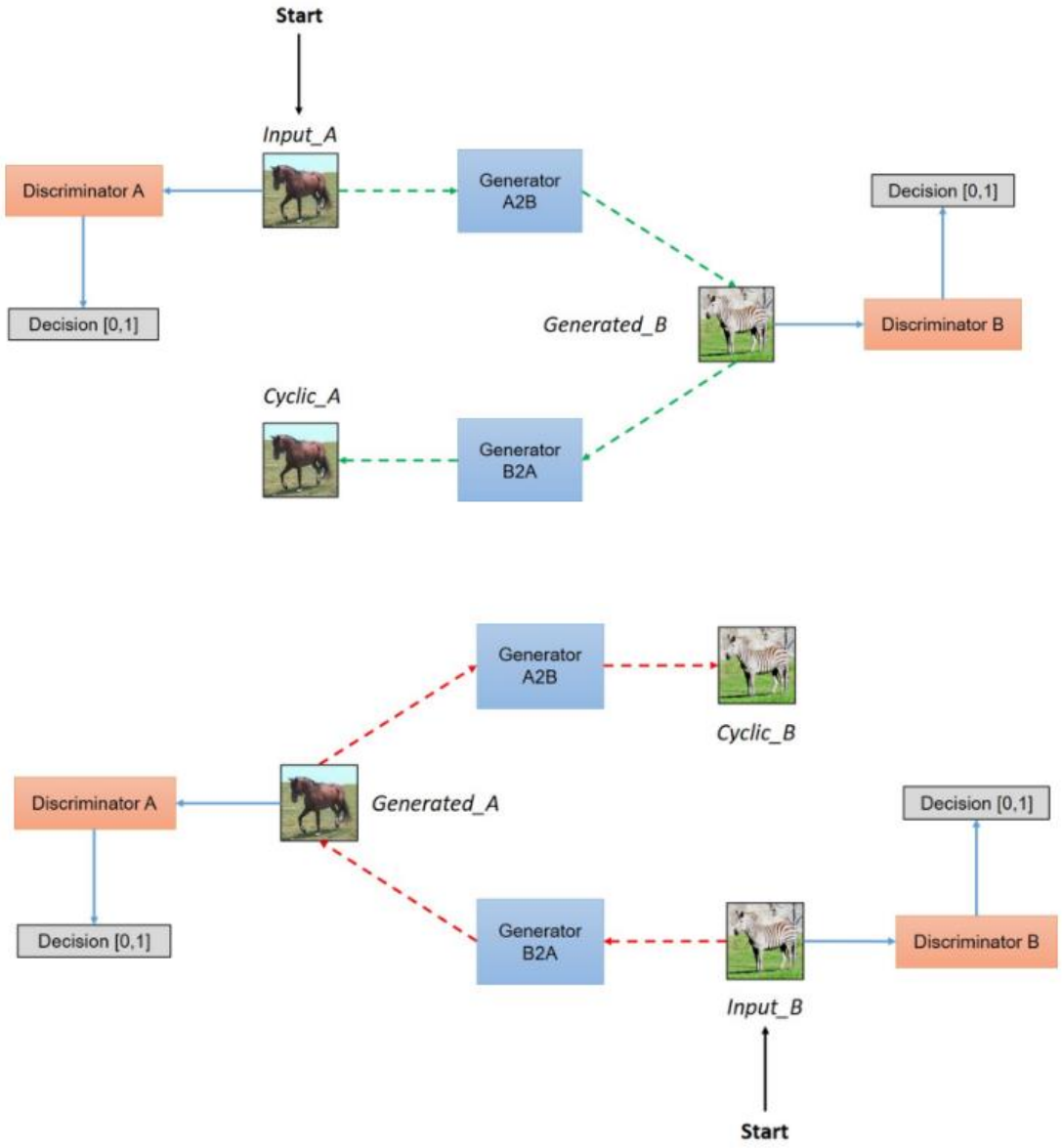
Cycle GANs are a special form of GANs which are used for learning the implicit relationship between 2 classes of datasets for which there is a lack in the quantity of supervised paired examples. They majorly focus on generating images in a different target domain as compared to the domain of the input images. The name is inspired by the introduction of a new term in the Loss function, known as 'Cyclic Loss', which basically allows us to enforce the cyclic relation of the Generator and the Discriminator network.

So, if we try and generate an image in the target domain using an input image in a particular domain, then we should be able to get the input image in our input domain using a suitable generator network and our output image in the target domain as input. This property is known as "Cycle Consistency" and we plan to use this property, along with the related knowledge of GANs to improve upon the task of Neural Style Transfer.



Here, Cycle-Consistency Loss [11] is shown visually using 2 generator networks working in opposite directions but between the same domain spaces

NETWORK ARCHITECTURE [11]



Here, as it can be seen, there are 2 generators and 2 discriminators. The 2 generators, as said before, work in the opposite direction, but, between the same 2 domain spaces. Similarly, the 2 discriminators work on images belonging to 2 different domain spaces. Thus, to optimize the whole network, we need to take 3 separate terms into account in our Loss function.

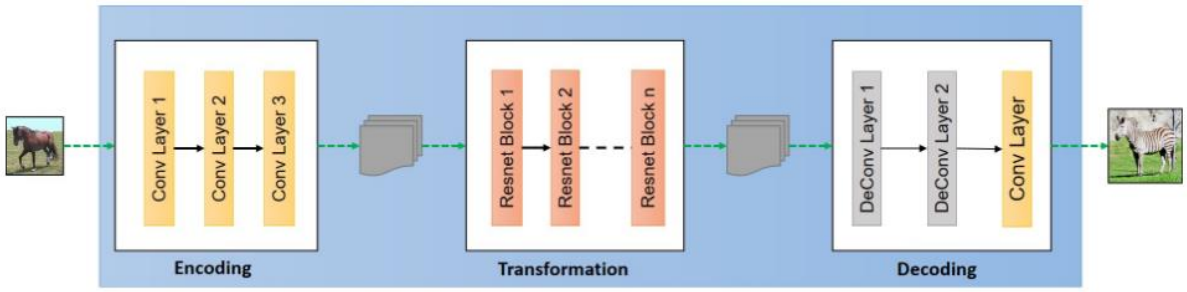
1. Loss corresponding to GAN 1 from Input Domain space to Target Domain Space
2. Loss corresponding to GAN 2 from Target Domain space to Input Domain Space
3. Cycle Inconsistency Loss

Thus, our final Loss function reduces to [11] -

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F)$$

There are some further minor changes to this function as well, such as the definition of L_{GAN} is changed from negative likelihood to least-squares for achieving stability on our model training procedure.

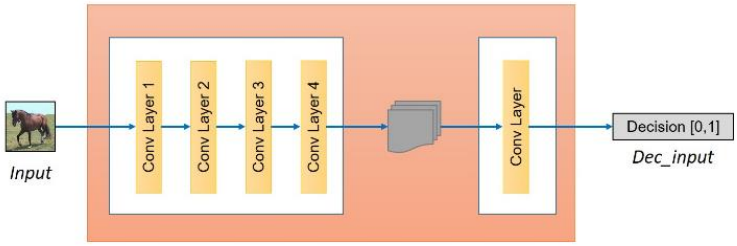
Generator Architecture [11]



The generator, as the architecture shows, is comprised of 3 parts-

1. Encoder
   The Encoder, as the name suggests, encodes the input image into a different feature space. This is done using convolution layers to extract the features of the input image which can then be transformed to target domain space and thus de-convolved to generate the output image.
2. Transformer
   The transformation of the image from the input domain space to the target domain space is done using the Resnet blocks as shown in the image. This is done to retain the input features of the image.
3. Decoder
   The last part of the generator network is the decoder, which generates the image from the transformed input feature space.

Discriminator Architecture [11]



The Discriminator Architecture, as it can be seen, basically consists of several convolution layers for extracting the information from the images. The information thus extracted is used to generate the probability of the input image belonging to a target class.

RESULTS [11]

CONCLUSION

Over the past several years, Neural Style Transfer has continued to become an inspiring research area, motivated by both scientific challenges and industrial demands with a considerable amount of research being conducted in the field. It is quite a fast-paced area, and we are looking forward to more exciting works devoted to advancing the development of this field. Although current algorithms achieve remarkable results, there are still several challenges and open issues. After thoroughly reviewing the current state of the art literature above, we summarize key challenges within this field and discuss their corresponding possible solutions.

- **Evaluation methodology**
  We believe that the lack of a standard aesthetic criterion is a major cause that prevents Neural Style Transfer from becoming a mainstream research direction like object detection and recognition. We think that the problem of a standard aesthetic criterion for neural style transfer is a generalized problem of Photographic Image Aesthetic Assessment, and one could get inspirations from related researches in this area.

- **Interpretable Neural Style Transfer**
  An important issue is the interpretability of Neural Style Transfer algorithms. Like many other CNN-based computer vision tasks, Neural Style Transfer is a black box, which makes it quite uncontrollable. Interpreting CNN feature statistics based style transfer can benefit the separation of different style attributes and address the problem of a finer control during stylization. For example, current Neural Style Transfer algorithms cannot guarantee the detailed orientations and continuities of curves in stylized results.

- **Speed, Flexibility and Quality in Neural Style Transfer**
  The most concerned challenge is probably the three-way trade-off between speed, flexibility and quality in Neural Style Transfer. Although current Arbitrary-Style-Per-Model Fast Neural Method (ASPM) algorithms successfully transfer arbitrary styles, they are not that satisfying in perceptual quality and speed. The quality of data driven ASPM quite relies on the diversity of training styles. However, one can hardly cover every style due to the great diversity of artworks. Image transformation based ASPM transfer arbitrary styles in a learning-free manner, but it is behind others in speed. One of the keys for this problem may be a better understanding of the optimization procedure in Neural Style Transfer. The choice of optimizer (e.g., Adam and L-BFGS) in Neural Style Transfer greatly influences the visual quality. We believe that a deep understanding towards optimization procedure will help understand how to find the local minima that leads to a high quality. Also, a well-studied automatic layer chosen strategy would also help improve the quality.

REFERENCES

[1] Gatys *et al*, “A Neural Algorithm of Artistic Style” [arXiv:1508.06576v2]

[2] Karen Simonyan & Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition” [arXiv:1409.1556v6]

[3] Alex J. Champandard, “Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artwork” [arXiv:1603.01768v1]

[4] Roman Novak and Yaroslav Nikulin, “Improving the Neural Algorithm of Artistic Style” [arXiv:1605.04603v1]

[5] Justin Johnson, Alexandre Alahi, Li Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution” [arXiv:1603.08155v1]

[6] Vincent Dumoulin & Jonathon Shlens & Manjunath Kudlur, “A Learned Representation of Artistic Style”, Google Brain

[7] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur et al, “Exploring the structure of a real-time, arbitrary neural artistic stylization network”, Google Deep Brain

[8] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, “Universal style transfer via feature transforms,” in Advances in Neural Information Processing Systems, 2017

[9] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. arXiv preprint [arXiv:1607.08022, 2016b]

[10] ”Generative Adversarial Nets” by Ian J. Goodfellow, Jean Pouget-Abadiey, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozairz, Aaron Courville, Yoshua Bengiox (Part of Advances in Neural Information Processing Systems 27 (NIPS 2014))

[11] “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” by Jun-Yan Zhu, Taesung Park, Phillip Isola and Alexei A. Efros [arXiv:1703.10593 [cs.CV]]