ECE 285 – MLIP – Project B
Style Transfer

Written by Inderjot Saggu. Last updated on October 18, 2019.

In this project the goal is to implement the classic technique of style transfer and one of its variants. This was first introduced in Gatys paper on "A Neural Algorithm for Artistic Style" and since then several variants especially ones that involve Generative Adversarial Networks (GANs) have become really popular. Prisma (winner of Best App of the Year 2016 both on Google Play and AppStore) probably uses the same algorithm. The authors of the original paper model the problem as follows: Given two images, we want to generate a new image that captures the style of one and content of the other. **What exactly is style and content, and how can we transfer it?** Convolution Neural Networks learn a hierarchy of feature representations. This hierarchical feature representation is used to define Content and Style reconstruction using a pre-trained VGG model:

- Content: Reconstruct the image by matching the network responses in a particular layer.
- Style: Find the correlation between different features in different layers of the CNN.

Note that additional information may be posted on Piazza by the Instructor or the TAs. This document is also subject to be updated. Most recent instructions will always prevail over the older ones. So look at the posting/updating dates and make sure to stay updated.

1 Neural Style Transfer

The first part of the project is to implement Gatys paper on Style Transfer. Given a white noise image, we will perform gradient descent on it to minimize the content and style loss respectively. How we define the loss function will determine what kind of final image we'll end up with. The approach defines content loss as follows: Given the original image $\mathbf{p}$ and the generated image $\mathbf{x}$, we define P^l and F^l as the feature response in layer l, reshaped as a 2D matrix for the original image and the generated image respectively. The content loss is then given by:

$$\mathcal{L}_{content}(\mathbf{p}, \mathbf{x}, l) = \frac{1}{2}\sum_{i,j}(F^l_{i,j} - P^l_{i,j})^2$$

For style-loss we need to compute feature correlation given by the Gram matrix, G^l where $G^l_{i,j}$ is the inner product between the vectorized feature map i and j in layer l:

$$G^l_{i,j} = \sum_k F^l_{i,k} F^l_{j,k}$$

To generate a texture that matches the style of a given image we minimize the mean-squared distance between the entries of the Gram matrix from the original image and the Gram matrix of the image to be generated. So let $\mathbf{a}$ and $\mathbf{x}$ be the original image and the generated image and A^l and G^l their respective style representations in layer l. The contribution of that layer to the total loss is then

$$\frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G^l_{i,j} - A^l_{i,j})^2$$

and the total loss is given by

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

The final loss function that we need to minimize is ($\mathbf{p}$ is the content image and $\mathbf{a}$ is the style image):

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

Try changing the $\frac{\alpha}{\beta}$ ratio and layer 'l' for determining the content of the image, observe how your results change and explain these results. Refer to this paper for more details: https://arxiv.org/pdf/1508.06576.pdf. We will now look at two variants (choose one) of Style Transfer, one that works in real-time but is data intensive and another that uses GANs.

(a) Original Image

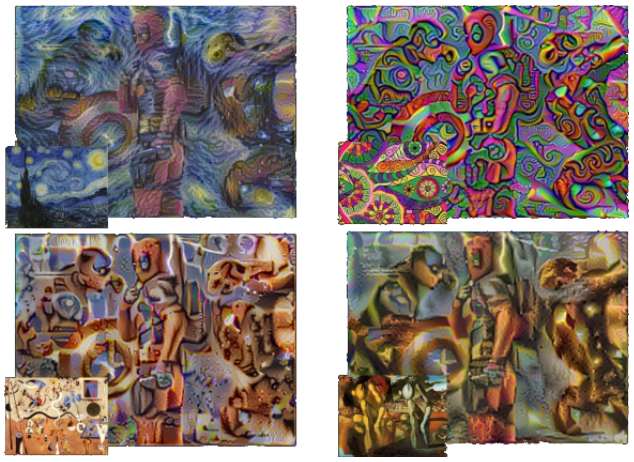
(b) Output: Style Image at bottom left

Figure 1: Neural Style Transfer, Gatys *et al.*

2 Real-Time Style Transfer

Gatys approach produce high-quality images, but is slow since inference requires solving an optimization problem. Justin Johnson *et al.* came up with a variant of style transfer that was much faster and produced similar results to the original implementation. Their approach involves training a CNN in a supervised manner, using *perceptual* loss function to measure the difference between output and ground-truth images.

2.1 Method

The architecture can be broken down into an image transformation network f_W (one that we need to train) and a pre-trained loss network ϕ. We'll be reusing the VGG network pre-trained on ImageNet dataset from the first part for ϕ. f_W on the other hand is a deep residual network that given an image x transforms it into another image $\hat{y} = f_W(\mathrm{x})$. The loss network is used to define a feature reconstruction loss l^{ϕ}_{feat} and style reconstruction loss l^{ϕ}_{style} that measure differences in content and style between images. For style transfer the content target y_c is the input image x and the output image $\hat{y}$ should combine the content of $x = y_c$ with the style of y_s.

2.2 Perceptual Loss Function

In contrast to a pixel-wise loss function, a perceptual loss function measures image similarities more robustly than per-pixel losses. For instance, given two identical images offset from each other by one pixel; despite their perceptual similarity they would be very different as measured by per-pixel losses. **How** do we define this similarity measure? We exploit the fact that a trained CNN is able to capture high-level feature representation which we can extract for defining our loss in a manner similar to the first part of the project.

- **Feature Reconstruction Loss** Let ϕ_j(x) be the activations of the j-th layer of the network ϕ when processing the image x; if j is a convolutional layer then ϕ_j(x) will be a feature map of shape $C_j \times H_j \times W_j$. The loss is then defined by (just as in the Gatys paper):

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

- **Style Reconstruction loss**: Recall the definition of gram matrix G as defined in the first part.

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

  The style reconstruction loss is then defined by the Frobenius norm of the difference between the Gram matrices of the output and the target images (note that this is the same as in the original paper):

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

2.3 Simple Loss Functions

Along with the above the final loss function also has pixel wise and total-variation loss.

- **Pixel Loss** Normalized Euclidean distance between the output image $\hat{y}$ and the target image y.
- **Total Variation Regularization** To encourage smoothness.

More details on the architecture, hyperparameter selection etc can be found in the paper https://arxiv.org/pdf/1603.08155.pdf.

2.4 Dataset

The original paper uses 80,000 images from the COCO dataset and trains one network per style target. This dataset is stored on DSMLP cluster in the directory `/datasets/COCO-2015`. It is suggested that you work with a smaller subset of images, say 10,000, initially to check whether your implementation is correct and then scale it further. You can choose any style target of your choice.

3 Image-to-Image Translation using Cycle-GANs

The goal of an image-to-image translation problem is to learn a way to translate an image in the source domain to an image in the target domain. This problem can be modeled for GANs as follows: we try to learn a mapping function G such that the distribution of G(I_{source}) is indistinguishable from the distribution of I_{target} using adversarial loss. One of the key-ideas is to constrain this problem using inverse mapping F s.t F(G(I_{source})) $\approx I_{source}$.

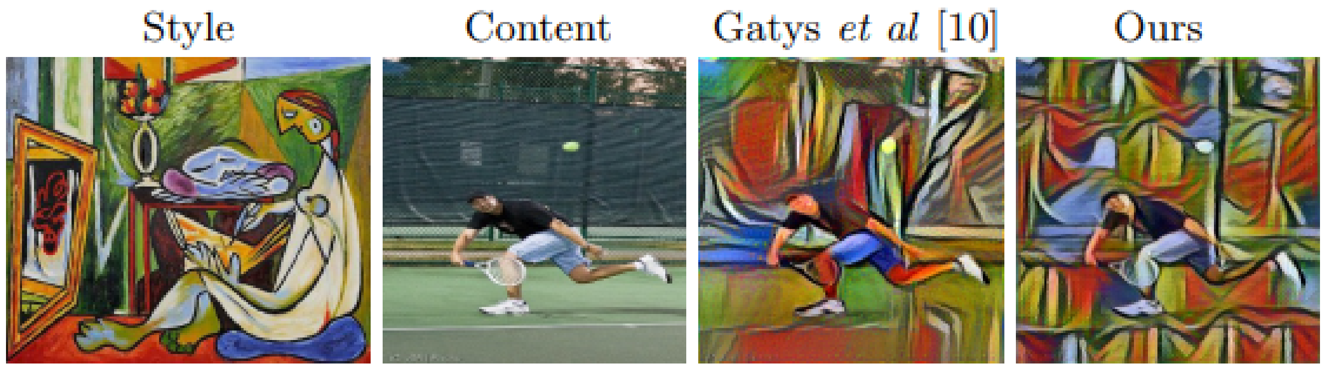

Figure 2: Style Transfer Result from Johnson et al

3.1 Method

Given samples from domain X, $\{x_i\}_{i=1}^{N}$, and samples from domain Y, $\{y_j\}_{j=1}^{M}$ we want to learn two models F and G, where G: X $\rightarrow$ Y and F: Y $\rightarrow$ X. Along with that we have two adversarial discriminators D_X, to distinguish between images x and translated images F(y), and D_Y to discriminate between y and G(x). The architecture is adopted from the first variant discussed above (Johnson et al) with minor modifications. The paper also employs two commonly used techniques to stabilize GAN training procedure.

- For $\mathcal{L}_{GAN}$,the negative log likelihood objective is replaced by a least-squares loss. This loss is more stable during training and generates higher quality results
- The discriminator is updated using a history of generated images rather than the ones produced by the latest generators. They keep an image buffer that stores the 50 previously created images.

3.2 Adversarial Loss

This can be written as:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log 1 - D_Y(G(x))]$$

where G tries to generate images G(x) that look similar to images from domain Y, while D_Y aims to distinguish between translated samples G(x) and real samples y. We also have a similar loss function that involves F and switches the role of X and Y.

3.3 Cycle-Consistency Loss

To add additional constraints to the parameter space as we try to ensure that for each image x from domain X, the image translation cycle brings x back to the original image, i.e., x $\rightarrow$ G(x) $\rightarrow$ F(G(x)) $\approx$ x. Similarly, for each image y from domain Y , G and F should also satisfy backward cycle consistency: y $\rightarrow$ F(y) $\rightarrow$ G(F(y)) $\approx$ y.

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1] + \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1]$$

The final loss function then looks like,

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

Details on architecture and techniques employed can be found in this paper https://arxiv.org/pdf/1703.10593.pdf.

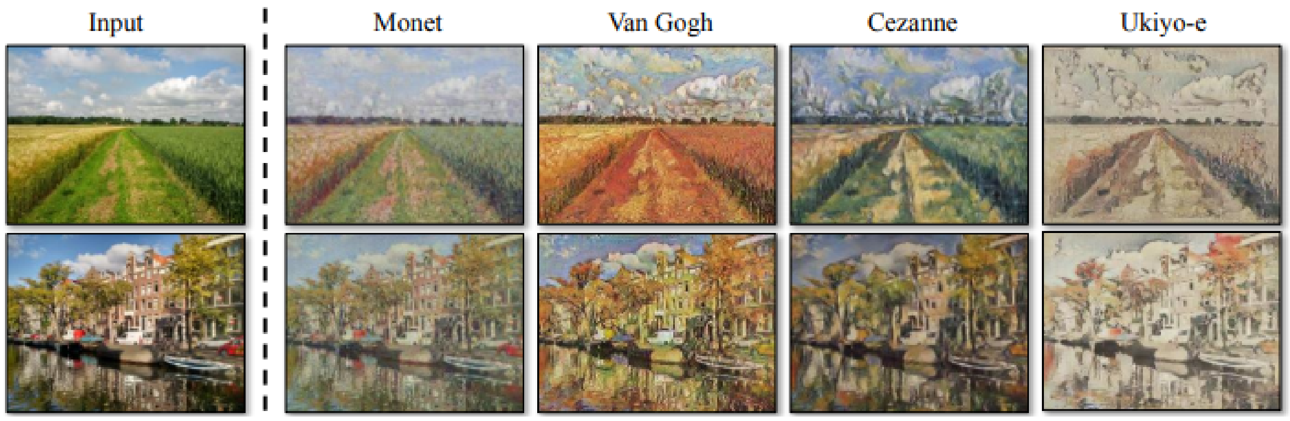

Figure 3: Cycle GAN Style Transfer

3.4 Dataset

The style images can be downloaded from WikiArt (https://www.wikiart.org/en/vincent-van-gogh)and the landscape images from FLickr (https://www.flickr.com/groups/landcape/). Both will be uploaded to the cluster.

4 Guidelines

- Implement original paper by Gatys on Neural Style Transfer
- Implement one of the two variants (Real time Style Transfer OR Image translation using Cycle GANs)
- Towards the end of the quarter, the `DSMLP` cluster will become very busy, slow at times and there might be connectivity issues. Please keep these things in mind and start early and also explore other alternatives like google co-lab (12 hours free GPU) etc.
- You are encouraged to implement classes similar to ones introduced in Assignment 3 (`nntools.py`) to structure and manage your project. Make sure you use checkpoints to save your model after every epoch so as to easily resume training in case of any issues.

5 Deliverables

You will have to provide the following

1. **A 10 page final report**:
   - 10 pages **MAX** including figures, tables and bibliography.
   - One column, font size: 10 points minimum, **PDF format**.
   - Use of Latex highly recommended (e.g., NIPS template).
   - Quality of figures matter (*Graph without caption or legend is void.)*
   - The report should contain at least the following:
     - Introduction. What is the targeted task? What are the challenges?
     - Description of the method: algorithm, architecture, equations, etc.

- Experimental setting: dataset, training parameters, validation and testing procedure (data split, evolution of loss with number of iterations etc.)
- Results: figures, tables, comparisons, successful cases and failures.
- Discussion: What did you learn? What were the difficulties? What could be improved?
- Bibliography.

2. **Link to a Git repository** (such as GitHub, BitBucket, etc) containing at least:

   - Python codes (using Python 3). You can use PyTorch, TensorFlow, Keras, etc.
   - A jupyter notebook file to rerun the training (if any),

     → We will look at it but we will probably not run this code (running time is not restricted).
   - Jupyter notebook file for demonstration,

     → We will run this on UCSD DSMLP (running time 3min max).
     This is a demo that must produce at least one illustration showing how well your model solved the target task. For example, if your task is classification, this notebook can just load one single testing image, load the learned model, display the image, and print the predicted class label. This notebook does not have to reproduce all experiments/illustrations of the report. This does not have to evaluate your model on a large testing set.
   - As many jupyter notebook file(s) for whatever experiments (optional but recommended)

     → We will probably not run these codes, but we may (running time is not restricted).
     These notebooks can be used to reproduce any of the experiments described in the report, to evaluate your model on a large testing set, etc.
   - Data: learned networks, assets, . . . (**5Gb max**)
   - **README file** describing:
     - the organization of the code (all of the above), and
     - if any packages need to be `pip` installed.
     - Example:

```
Description
===========
This is project FOO developed by team BAR composed of John Doe, ...

Requirements
============
Install package 'imageio' as follow:

        $ pip install --user imageio

Code organization
=================
demo.ipynb         --  Run a demo of our code (reproduces Figure 3 of our report)
train.ipynb        --  Run the training of our model (as described in Section 2)
attack.ipynb       --  Run the adversarial attack as described in Section 3
code/backprop.py   --  Module implementing backprop
code/visu.py       --  Module for visualizing our dataset
assets/model.dat   --  Our model trained as described in Section 4
```

6 Grading and submission

The grading policy and submission procedure will be detailed later.