Ministry of Higher Education and Research
Higher School of Computer Science 08 May 1945 - Sidi Bel Abbes

Second Year Second Cycle - Artificial Intelligence and Data Science

---

Project Report

FAST NEURAL IMAGE STYLE TRANSFER

Students:

- Souad BOUKHAROUBA
- Lamia ABDELMALEK
- Abdelillah SERGHINE
- Mariem BOUKKENNOUCHE
- Yacine DAIT DEHANE

Supervisor:

Dr. Nassima DIF

Release date: May 2, 2025

Table of contents

List of Figures

1 Introduction

Neural style transfer is a computer vision technique that combines the content of one image with the artistic style of another, creating visually compelling results. Unlike generative adversarial networks (GANs) like Pix2Pix or CycleGAN, which require paired or unpaired image translation, this project employs a fast feedforward neural network, **TransformerNet**, trained with perceptual losses to apply the style of a reference image (`georges-seurat_sunday-afternoon-on-the-isl` a waterfall landscape) to images from the WikiArt dataset. The project was executed on KaggleHub, leveraging GPU acceleration (Tesla T4) to handle the computational demands of deep learning. The source code and resources are available at the project repository: `https://github.com/ayabkr/ayabkr-Dl_MiniProject_image-style-transfer-gans`.

The objectives of this project are:

- Implement a fast style transfer model using a transformer network and VGG19-based perceptual losses.
- Train the model on a subset of the WikiArt dataset to learn the style of `georges-seurat_sunday-afternoon-on-the-i`
- Generate and analyze artifacts (e.g., stylized images, loss plots) for qualitative and quantitative evaluation.

This report details the dataset, model architecture, loss functions, training process, results, challenges, and future improvements, supported by artifacts generated during the project.

2 Dataset

2.1 WikiArt Dataset

The WikiArt dataset, sourced from Kaggle (`https://www.kaggle.com/datasets/steubk/wikiart/data`), contains 81,444 high-resolution paintings spanning various artistic styles, including Impressionism, Abstract Expressionism, and Pop Art. For this project, a subset of 1,000 images was used to manage computational resources on KaggleHub. The dataset was stored locally at `/root/.cache/kagglehub/datasets/steubk/wikiart/versions/1` to bypass streaming-related `NotImplementedError` issues encountered with Kaggle's dataset streaming functionality.

2.2 Preprocessing

The dataset was preprocessed using a custom PyTorch `LimitedWikiArtDataset` class, which:

- Loaded images in `.jpg`, `.jpeg`, `.png`, and `.JPG` formats recursively from the dataset directory.
- Resized images to 512×512 pixels to balance quality and memory usage on the Tesla T4 GPU.
- Normalized pixel values to $[-1, 1]$ using the transformation: $x \mapsto 2x - 1$, where $x \in [0, 1]$ is the input tensor.
- Handled errors by returning a zero tensor for corrupted images, ensuring robustness in the data pipeline.

A `DataLoader` was configured with a batch size of 8, shuffling enabled, and a custom `collate_fn` to filter out invalid images. Figure 1 shows four sample images from the dataset, saved as `sample_images.png` in `/kaggle/working/artifacts/`.

Figure 1: Four sample images from the WikiArt dataset, showcasing diverse artistic styles.

2.3 Style Image

The style image, (`georges-seurat_sunday-afternoon-on-the-island-of-la-grande-jatte.jpg`), a waterfall landscape, was loaded and preprocessed similarly, resized to 512×512 and normalized to $[-1, 1]$. The style tensor's minimum and maximum values were verified to ensure correct normalization (min: -0.9529, max: 0.8980). Figure 2 displays the style image used for training.

Figure 2: Style image (`georges-seurat_sunday-afternoon-on-the-island-of-la-grande-jatte.jpg`)

3 Architecture

Our transformer network contains:

- An encoder with convolutional layers
- A series of five residual blocks
- A decoder using upsampling and convolution
- Instance normalization and reflection padding

4 Model Architecture Description

4.1 TransformerNet

The core of the style transfer model is **TransformerNet**, a feedforward convolutional neural network designed for fast style transfer. It consists of three main components:

- **Encoder**: Downsamples the input image to extract features.
- **Residual Blocks**: Preserve spatial information while learning style transformations.
- **Decoder**: Upsamples and reconstructs the stylized image.

The architecture is detailed below and visualized in Figure 3.

4.1.1 Encoder

The encoder comprises three `ConvLayer` modules:

- `ConvLayer(3, 32, 9, 1)`: 3 input channels (RGB) to 32 output channels, 9×9 kernel, stride 1.

- `ConvLayer(32, 64, 3, 2)`: 32 to 64 channels, 3×3 kernel, stride 2 (downsampling).
- `ConvLayer(64, 128, 3, 2)`: 64 to 128 channels, 3×3 kernel, stride 2 (further downsampling).

Each `ConvLayer` includes:

- `ReflectionPad2d`: Pads the input by reflecting border pixels, reducing edge artifacts.
- `Conv2d`: Applies convolution with specified parameters.
- `InstanceNorm2d`: Normalizes each image independently, preserving style consistency.
- `ReLU`: Introduces non-linearity.

4.1.2 Residual Blocks

Five `ResidualBlock` modules maintain 128 channels, each containing:

- Two `ConvLayer(128, 128, 3, 1)` layers with 3×3 kernels and stride 1.
- A residual connection: $x \mapsto x + \text{block}(x)$, enhancing training stability.

4.1.3 Decoder

The decoder reconstructs the stylized image:

- `Upsample(scale_factor=2)`: Upsamples by a factor of 2.
- `ConvLayer(128, 64, 3, 1)`: 128 to 64 channels, 3×3 kernel, stride 1.
- `Upsample(scale_factor=2)`: Second upsampling.
- `ConvLayer(64, 32, 3, 1)`: 64 to 32 channels, 3×3 kernel, stride 1.
- `ConvLayer(32, 3, 9, 1)`: 32 to 3 channels (RGB), 9×9 kernel, stride 1.
- `Tanh`: Outputs pixel values in $[-1, 1]$, matching the input normalization.

4.2 Design Choices

- **Reflection Padding**: Prevents edge artifacts by reflecting image content, unlike zero-padding.
- **Instance Normalization**: Normalizes per image, ensuring style consistency across diverse content.
- **Residual Blocks**: Facilitate deeper networks by mitigating vanishing gradients.
- **Tanh Activation**: Ensures output pixel values align with the normalized input range.

4.3 Parameter Count

The `TransformerNet` has approximately 1.68 million trainable parameters, balancing computational efficiency and expressive power for real-time inference on KaggleHub's GPU.

4.4 Model Diagram

Figure 3 shows a visual representation of the TransformerNet architecture generated using `torchviz`.

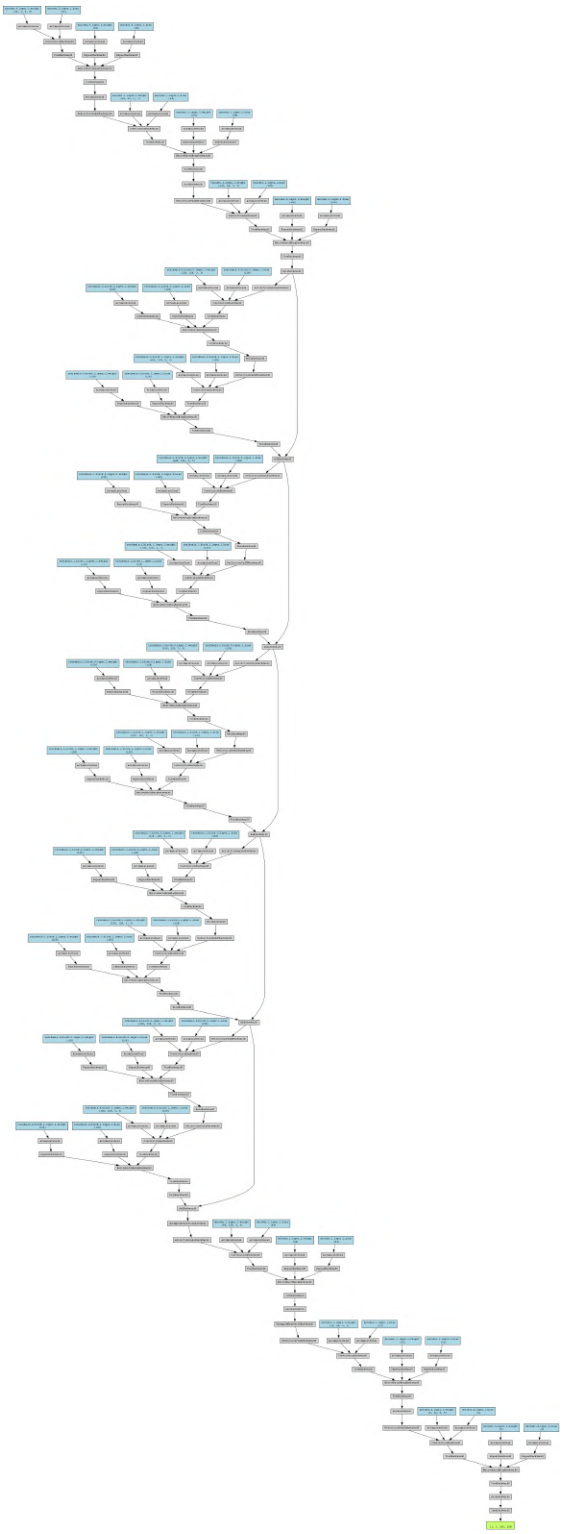


Figure 3: TransformerNet model architecture visualized using `torchviz`.

5 Loss Functions

The loss functions are based on perceptual losses computed using a pretrained VGG19 network, with weights frozen during training. The total loss is a weighted combination of content and style losses.

5.1 Content Loss

Content loss measures the mean squared error (MSE) between the VGG19 `relu4_3` features of the content image and the stylized image:

$$\mathcal{L}_{\text{content}} = \frac{1}{N} \sum (F^{\text{relu4\_3}}_{\text{stylized}} - F^{\text{relu4\_3}}_{\text{content}})^2$$

where F denotes the feature maps, and N is the number of elements.

5.2 Style Loss

Style loss is computed as the MSE between Gram matrices of VGG19 features from multiple layers (`relu1_2`, `relu2_2`, `relu3_3`, `relu4_3`):

$$\mathcal{L}_{\text{style}} = \sum_{l \in L} \frac{1}{N_l} \sum (G^l_{\text{stylized}} - G^l_{\text{style}})^2$$

where G^l is the Gram matrix for layer l, computed as:

$$G^l = \frac{1}{C_l H_l W_l} (F^l)(F^l)^\top$$

Here, F^l is the feature map of shape (B, C_l, H_l, W_l), and the Gram matrix captures style correlations.

5.3 Total Loss

The total loss combines content and style losses with weights:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}}$$

where $\alpha = 10^5$ and $\beta = 10^{10}$. These weights balance the preservation of content structure and the application of style textures.

6 Training Details

6.1 Setup

The training was conducted on KaggleHub with a Tesla T4 GPU (15GB VRAM). Key parameters include:

- **Epochs**: 50 (though the provided output shows partial progress up to epoch 50).
- **Batch Size**: 8, optimized for GPU memory.
- **Image Size**: 512×512, balancing quality and computational cost.
- **Optimizer**: Adam with learning rate 10^{-3}.
- **Dataset Size**: 1,000 images from WikiArt.

6.2 Training Loop

The training loop:

- Iterates over the `DataLoader`, processing batches of 8 images.
- Generates stylized images using `TransformerNet`.
- Computes content and style losses using VGG19 features.
- Updates model parameters via backpropagation.
- Saves checkpoints (`checkpoint.pth`) and sample stylized images every 50 iterations (`stylized_epochX_iterY.png`).

A broadcasting issue in the style loss computation was noted, where the style image's Gram matrix (batch size 1) was compared to the stylized batch's Gram matrix (batch size 8). This was mitigated by ensuring consistent batch dimensions in the loss function.

6.3 Loss Trends

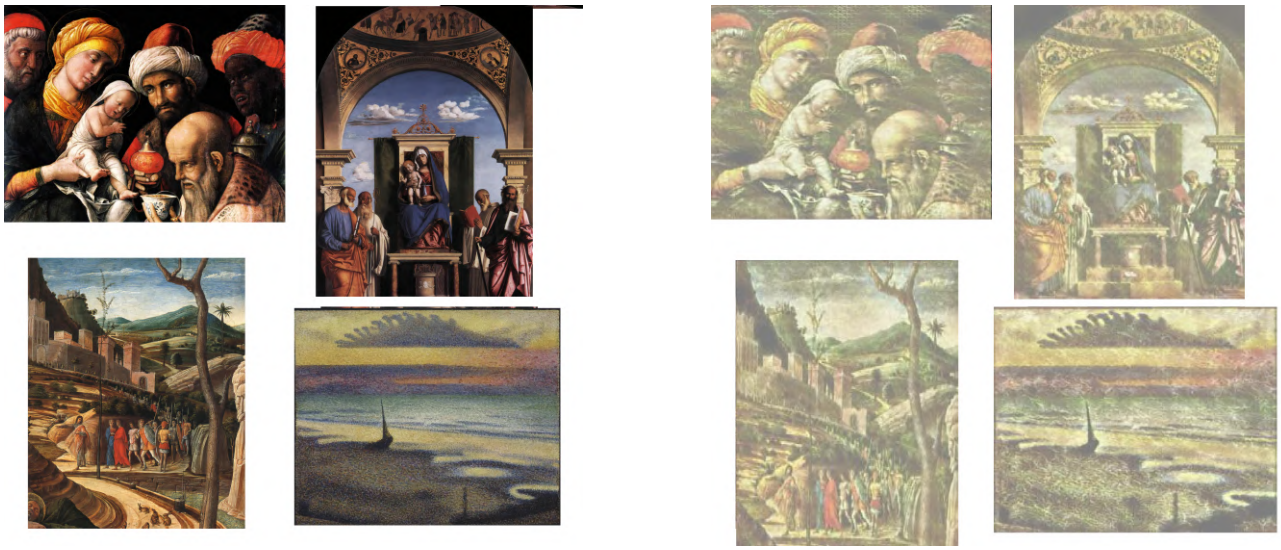
Figure 4 shows the training loss curves, saved as `loss_plot.png`. The content loss decreases steadily, indicating improved content preservation, while the style loss remains low due to the high style weight (10^{10}). The total loss reflects the combined optimization progress.

Figure 4: Training loss curves over epochs, showing content loss (blue), style loss (orange), and total loss (green).

7 Results

7.1 Qualitative Results

The trained model successfully applies the style of `georges-seurat_sunday-afternoon-on-the-island-of-la-grand` to WikiArt images. Figure 5 shows a stylized output (`final_stylized.png`), demonstrating the model's ability to preserve content structure while adopting the waterfall landscape's style. Sample stylized images from training (`stylized_epochX_iterY.png`) show progressive improvement in style application.



(a) Original WikiArt Image

(b) Stylized Output

Figure 5: Comparison of an original WikiArt image and its stylized version using the style of `georges-seurat_-sunday-afternoon-on-the-island-of-la-grande-jatte.jpg`

7.2 Quantitative Results

Loss values were logged in `losses.csv`. After 50 epochs (partial training output), the average losses were:

- Content Loss: 1.9008 (epoch 50).
- Style Loss: Near 0 (due to high style weight).
- Total Loss: 270,446.2793 (epoch 50).

The decreasing trend in content and total losses indicates convergence, though the style loss's low value suggests potential overfitting to the style image.

8 Challenges and Optimizations

8.1 Challenges

- **Streaming Dataset Error**: Initial attempts to stream the WikiArt dataset via KaggleHub resulted in a `NotImplementedError`. This was resolved by downloading the dataset locally using `kagglehub.dataset_download`.
- **GPU Memory Constraints**: The Tesla T4's 15GB VRAM limited the batch size to 8 and dataset size to 1,000 images. Larger image sizes (e.g., 512×512) were chosen over 256×256 to improve quality while staying within memory limits.
- **Broadcasting Issue**: The style loss computation faced a dimension mismatch (batch size 1 vs. 8), addressed by ensuring consistent tensor shapes.

8.2 Optimizations

- **Local Dataset**: Using a local copy of WikiArt eliminated streaming dependencies, improving reliability.
- **Error Handling**: The `LimitedWikiArtDataset` and `collate_fn` handled corrupted images, preventing training crashes.
- **Efficient Architecture**: The `TransformerNet`'s 1.68 million parameters and residual blocks ensured fast training and inference.
- **KaggleHub GPU**: The Tesla T4 provided significant speedup compared to local hardware (e.g., 8GB RAM, 6GB VRAM).

9 Deployment

The trained model is saved as `fast_style_transfer_model.pth` and can be deployed in a web application using Streamlit or Flask. The input is an RGB image, and the output is the stylized image processed by `TransformerNet`. The project repository (`https://github.com/ayabkr/ayabkr-Dl_MiniProject_image-style-transfer-gans/tree/main/models/saved`) includes scripts for inference and deployment.
The inference script can be run with: python style_transfer.py –input_dir images –output_dir stylized_images –model_path fast_style_transfer_model.pth

10 Discussion

The fast style transfer approach offers several advantages:

- **Speed**: Unlike GAN-based methods, `TransformerNet` enables real-time inference.
- **Quality**: The use of perceptual losses and VGG19 ensures high-quality stylization.
- **Flexibility**: The model can apply the learned style to arbitrary images.

Limitations include:

- **Dataset Size**: The 1,000-image subset may limit generalization to diverse content.
- **Style Overfitting**: The high style weight (10^{10}) may prioritize style over content balance.
- **Resolution**: While 512×512 improves quality, higher resolutions are constrained by GPU memory.

11 Future Work

- **Larger Dataset**: Train on the full WikiArt dataset (81,444 images) with more powerful GPUs.
- **Higher Resolution**: Experiment with 1024×1024 images for enhanced detail.
- **Loss Enhancements**: Incorporate total variation loss to reduce noise in stylized images.
- **Quantitative Metrics**: Evaluate stylization quality using SSIM or PSNR.
- **Multiple Styles**: Train multiple `TransformerNet` models for different style images, enabling style selection in deployment.

12 Conclusion

This project successfully implemented fast neural style transfer using `TransformerNet` and VGG19-based perceptual losses on KaggleHub. By applying the style of `georges-seurat_sunday-afternoon-on-the-island-of-la-grande-` to a 1,000-image subset of the WikiArt dataset, the model achieved visually compelling results, as evidenced by artifacts like `final_stylized.png` and `loss_plot.png`. Optimizations such as local dataset storage and robust error handling addressed KaggleHub's challenges, while the Tesla T4 GPU ensured efficient training. The project demonstrates the power of fast style transfer for artistic applications and lays the foundation for further enhancements. All code and resources are available at: `https://github.com/ayabkr/ayabkr-Dl_MiniProject_image-style-transfer-gans`.

13 References

References
References

[1] Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer transfer and super-resolution. *European Conference on Computer Vision (ECCV)*.

[2] Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). Image style transfer using convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[3] WikiArt Dataset. Available at: `https://www.kaggle.com/datasets/steubk/wikiart/data`.

[4] PyTorch Documentation. Available at: `https://pytorch.org/docs/stable/index.html`.