Neural Style Tranfser

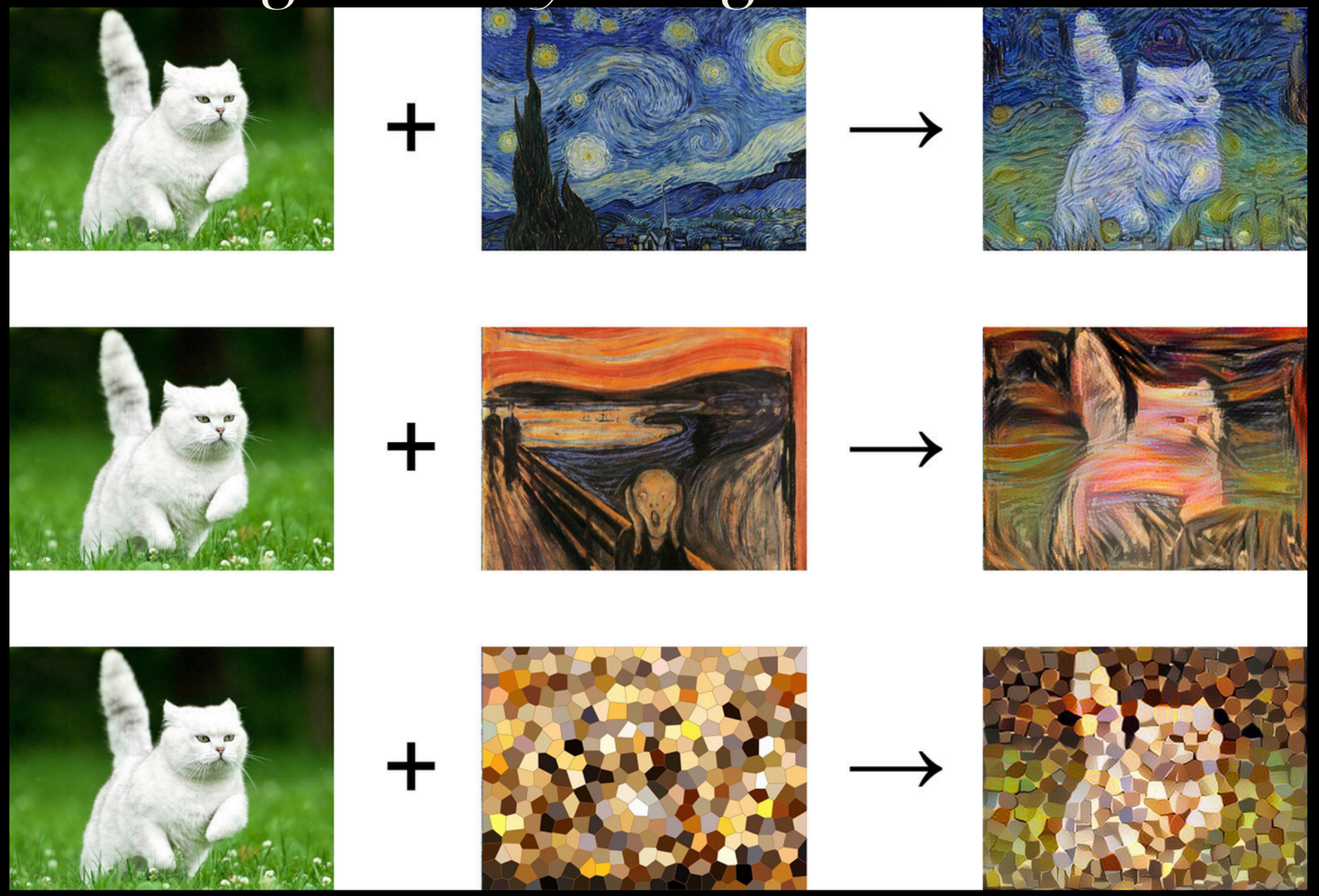
Under Supervision : Dr. shimaa yosry
presented by Hana Nabhan

Fusing Reality with Art

NST is a computer vision technique used to blend three images.

- Photo & Video Editing
- Art & Design
- Gaming & Virtual Reality (VR)
- Data Augmentation
- Interface Customization
- Film & Photography

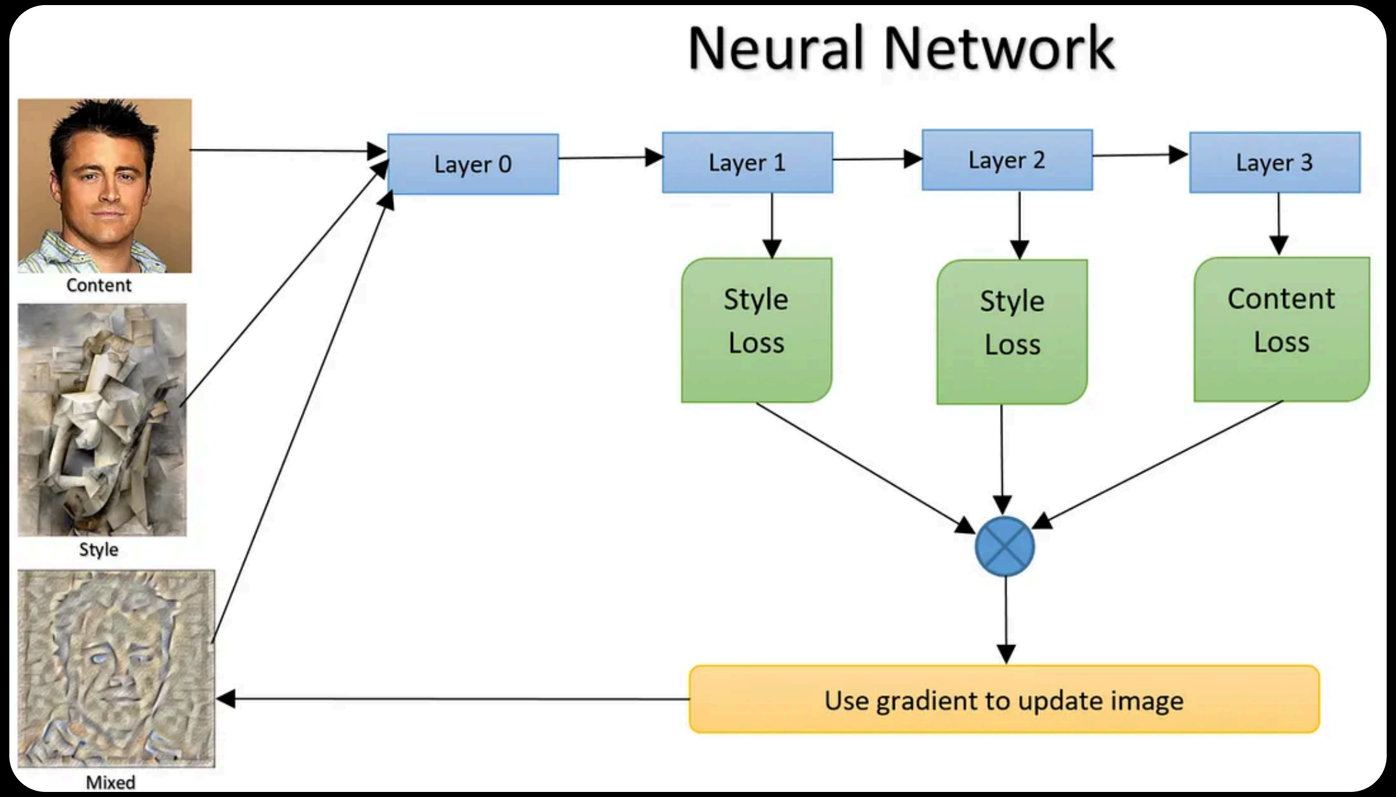
content image *style image* generated image



Seeing Like a Machine: The NST Process

Define Goals

loss function is created to measure how well preserves the original's content, adopts the new style, and remains visually coherent.



Perceive

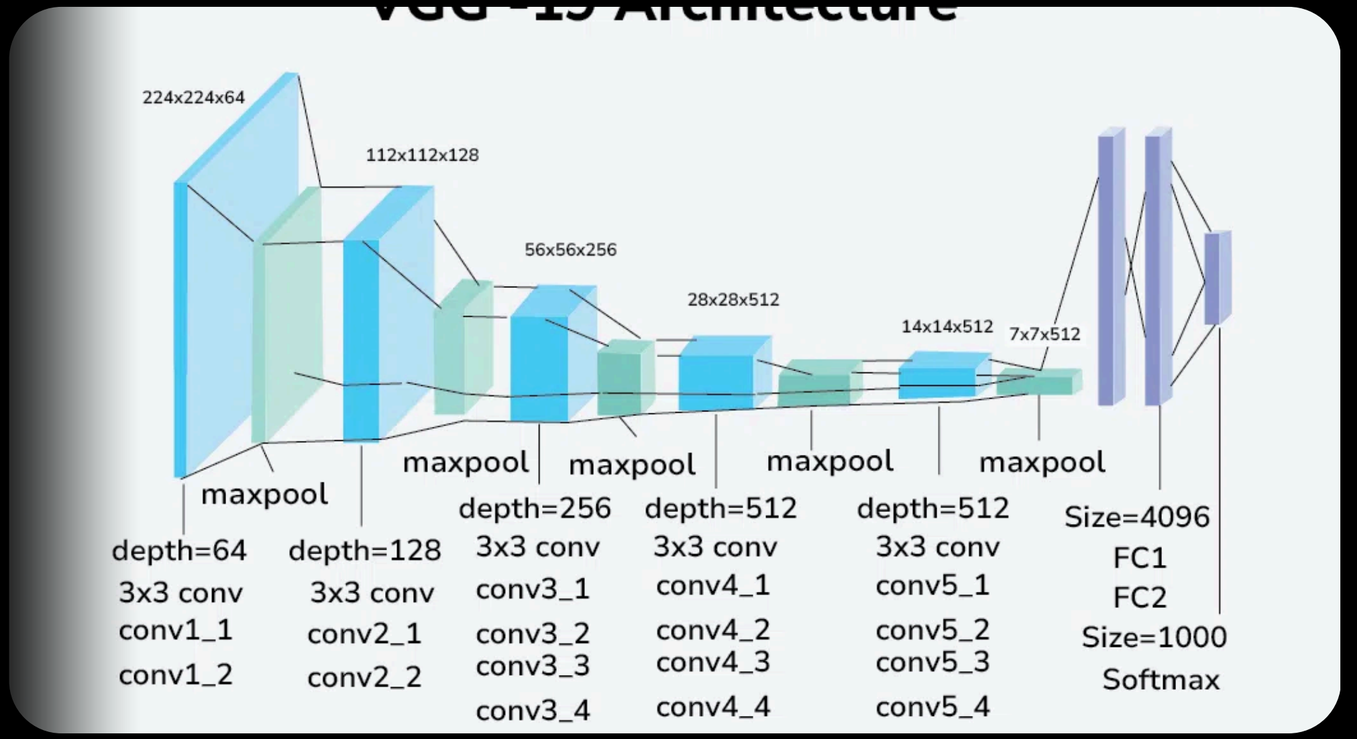
(VGG19) is used to extract feature

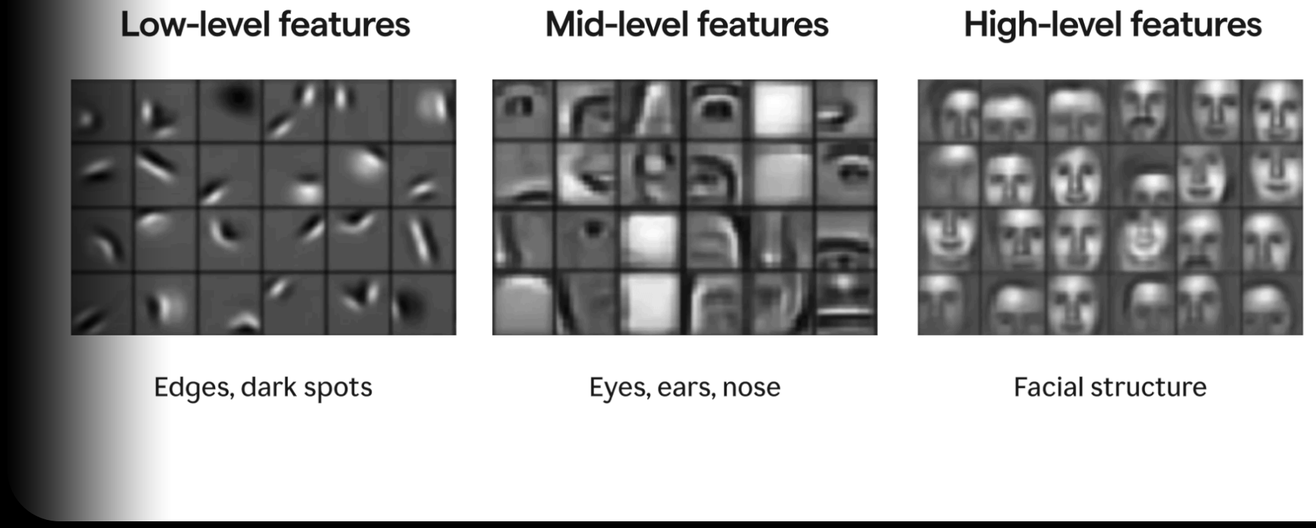
Optimize

adjust to minimize the total loss

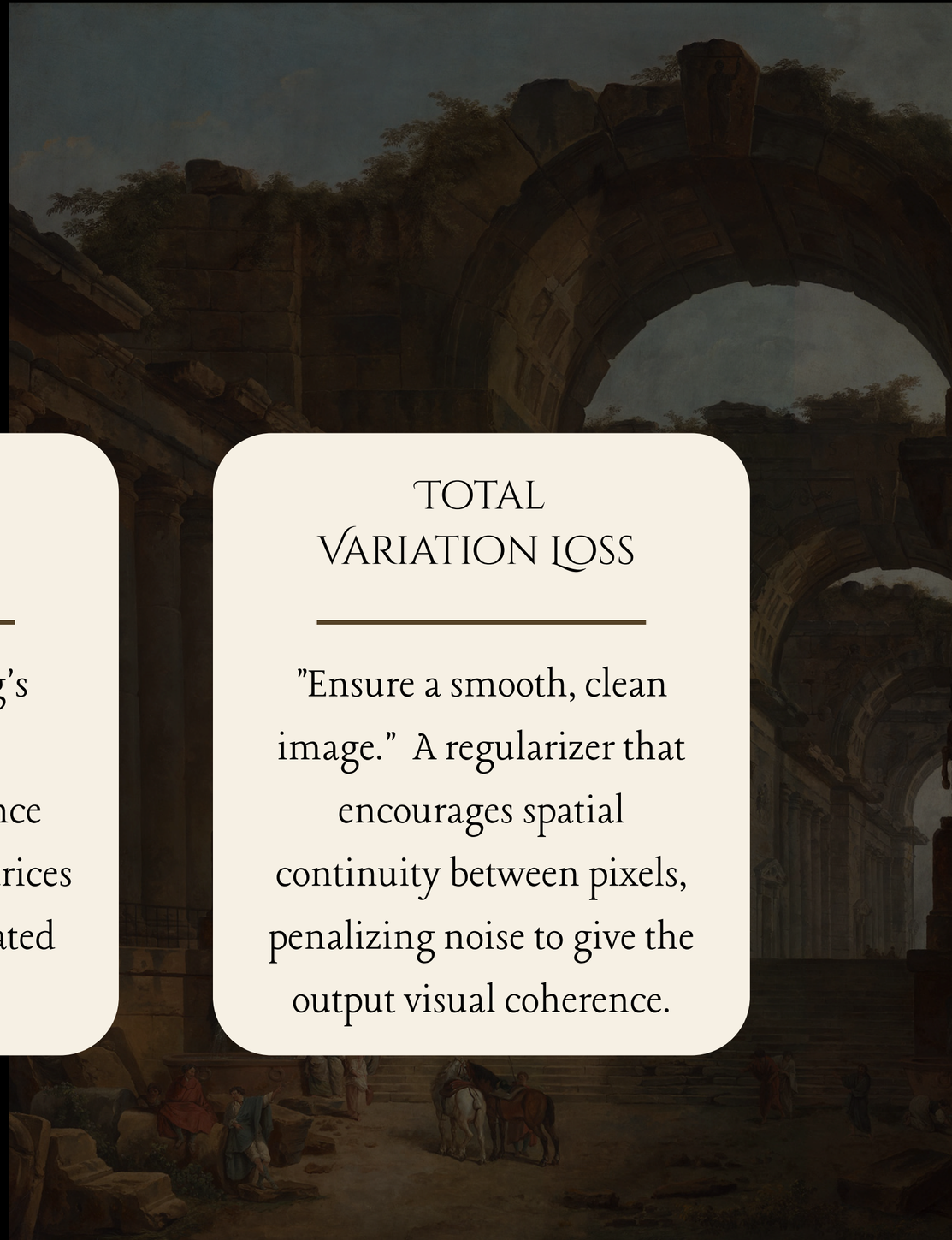
Trained on ImageNet, VGG19 has learned a rich hierarchy of visual features that can distinguish between high-level structure and low-level texture.

VGG19 Structure





The Three Goals of Optimization



CONTENT LOSS

"Keep the photo's structure."
Measures the squared-error (L2 distance) between the feature representations of the content image and the generated image.

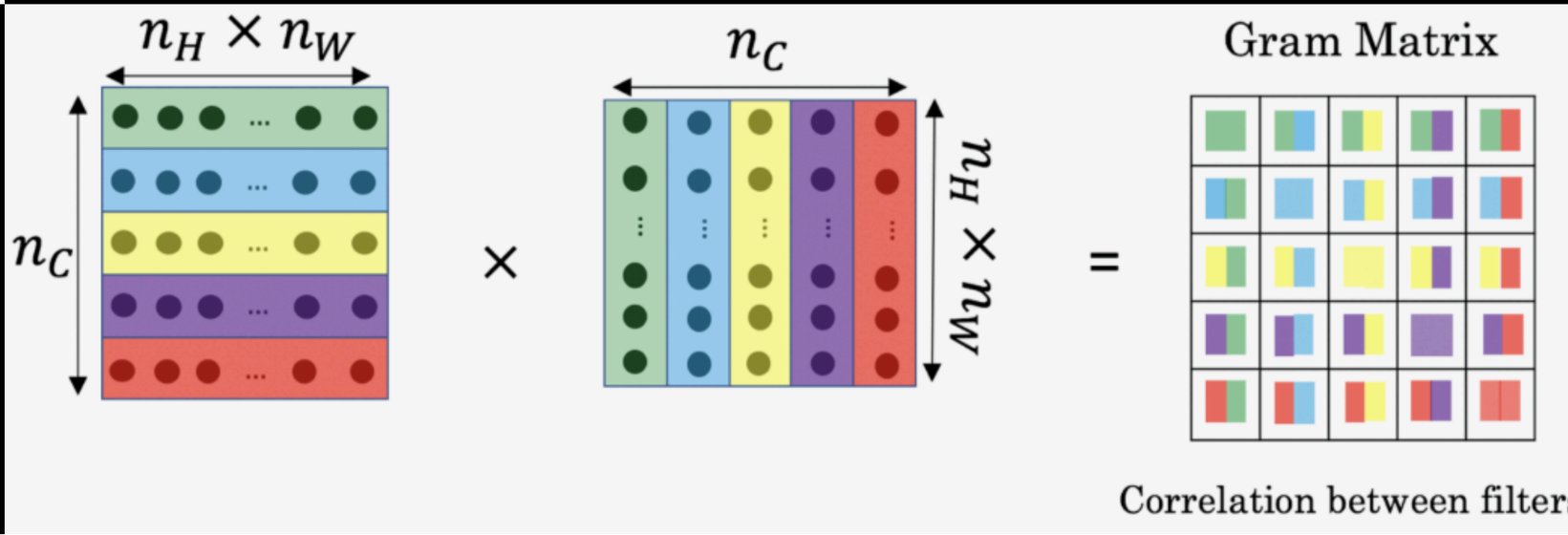
STYLE LOSS

"Mimic the painting's texture."
Measures the difference between the Gram matrices of the style and generated images.

TOTAL VARIATION LOSS

"Ensure a smooth, clean image." A regularizer that encourages spatial continuity between pixels, penalizing noise to give the output visual coherence.

The Gram Matrix



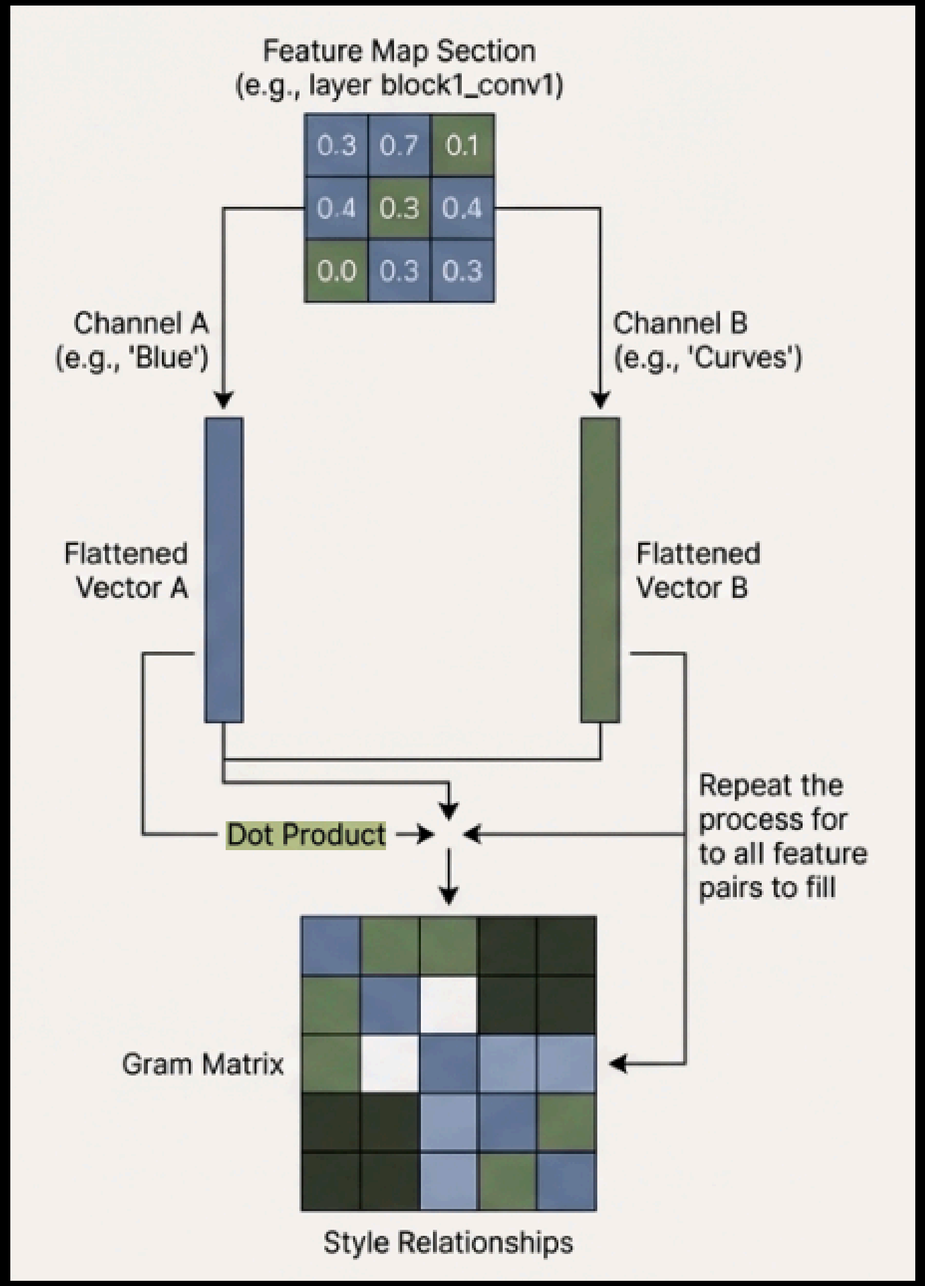
$$G^l_{ij} = \sum F^l_{ik} F^l_{jk}.$$

What it is ?

The Gram Matrix measures the correlations between different filter responses in a single layer. It captures which features tend to activate together, representing the image's texture and patterns, independent of their specific location.

The Gram Matrix

The Core Idea Dot Product "In the original painting, whenever I see 'Blue', I also see 'Curved Lines'. Does your image do that too? This relationship (Blue + Curves) is calculated using the Gram Matrix."



Loss functions

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

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

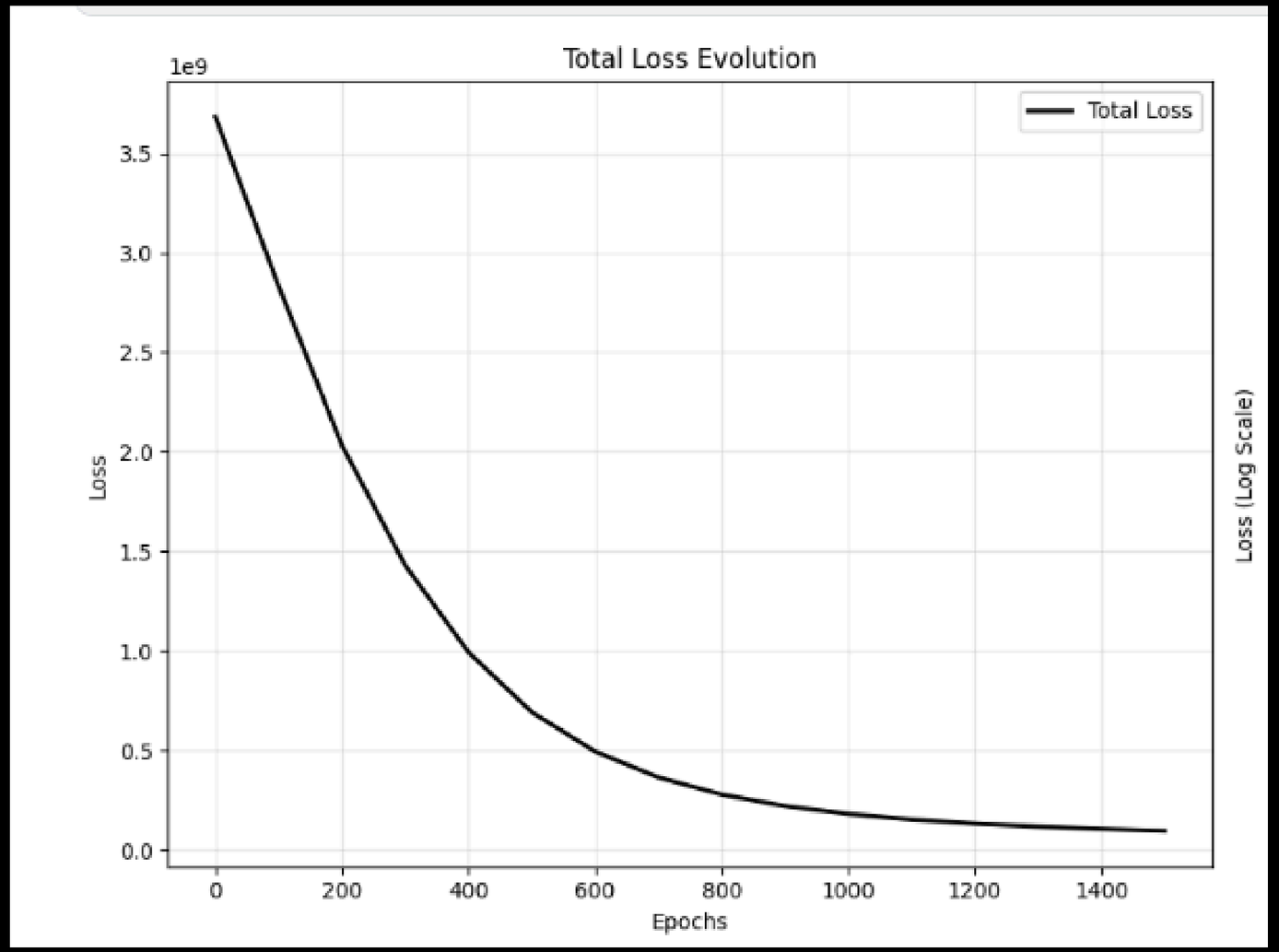
$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

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

Adam is a gradient-based optimizer commonly used in deep learning. The process uses a `@tf.function` called `train_step` that calculates the total loss and applies gradients in each step.

Hyperparameters

content_weight = 1000.0, style_weight = 0.1, total_variation_weight = 30.

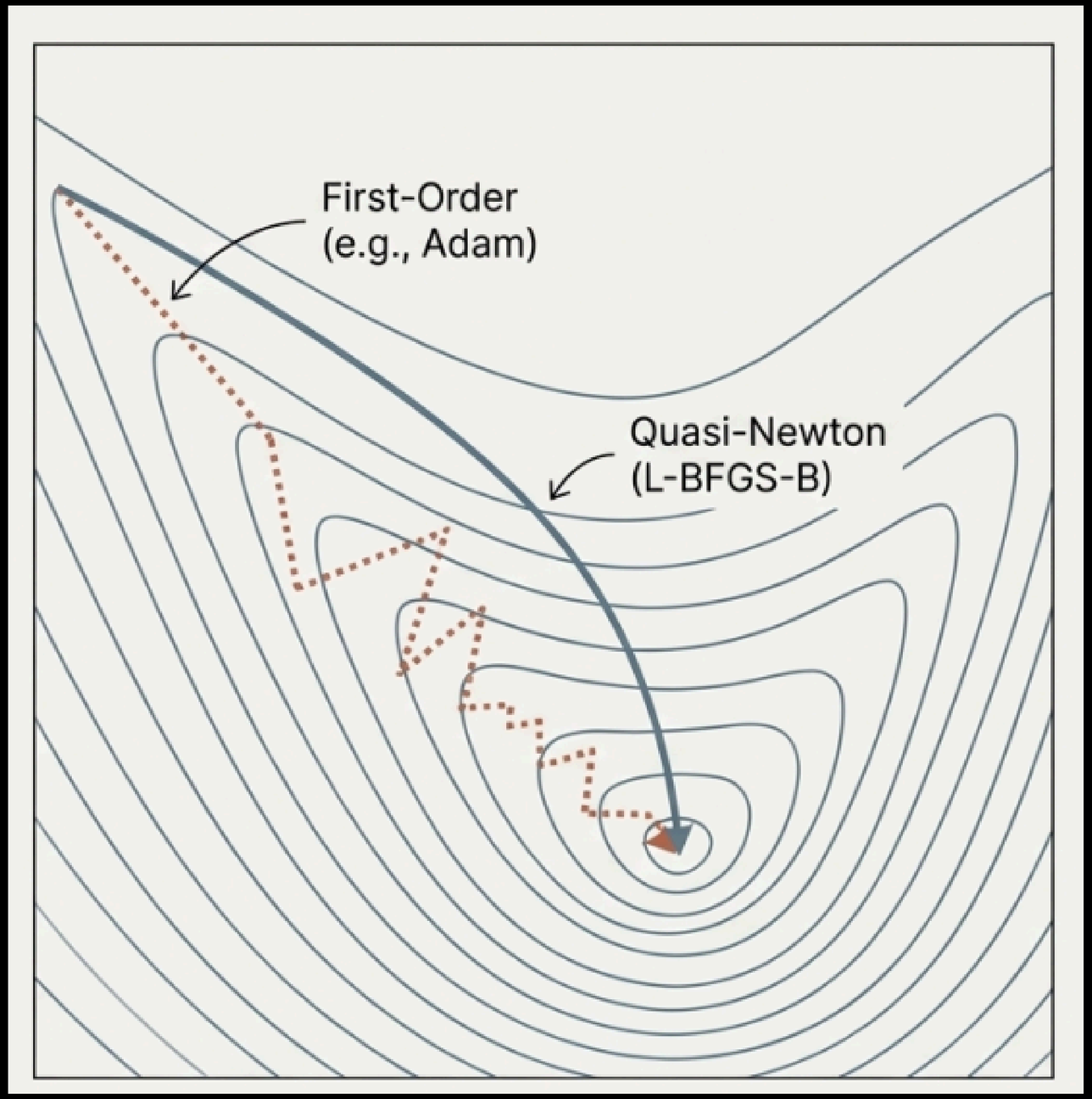


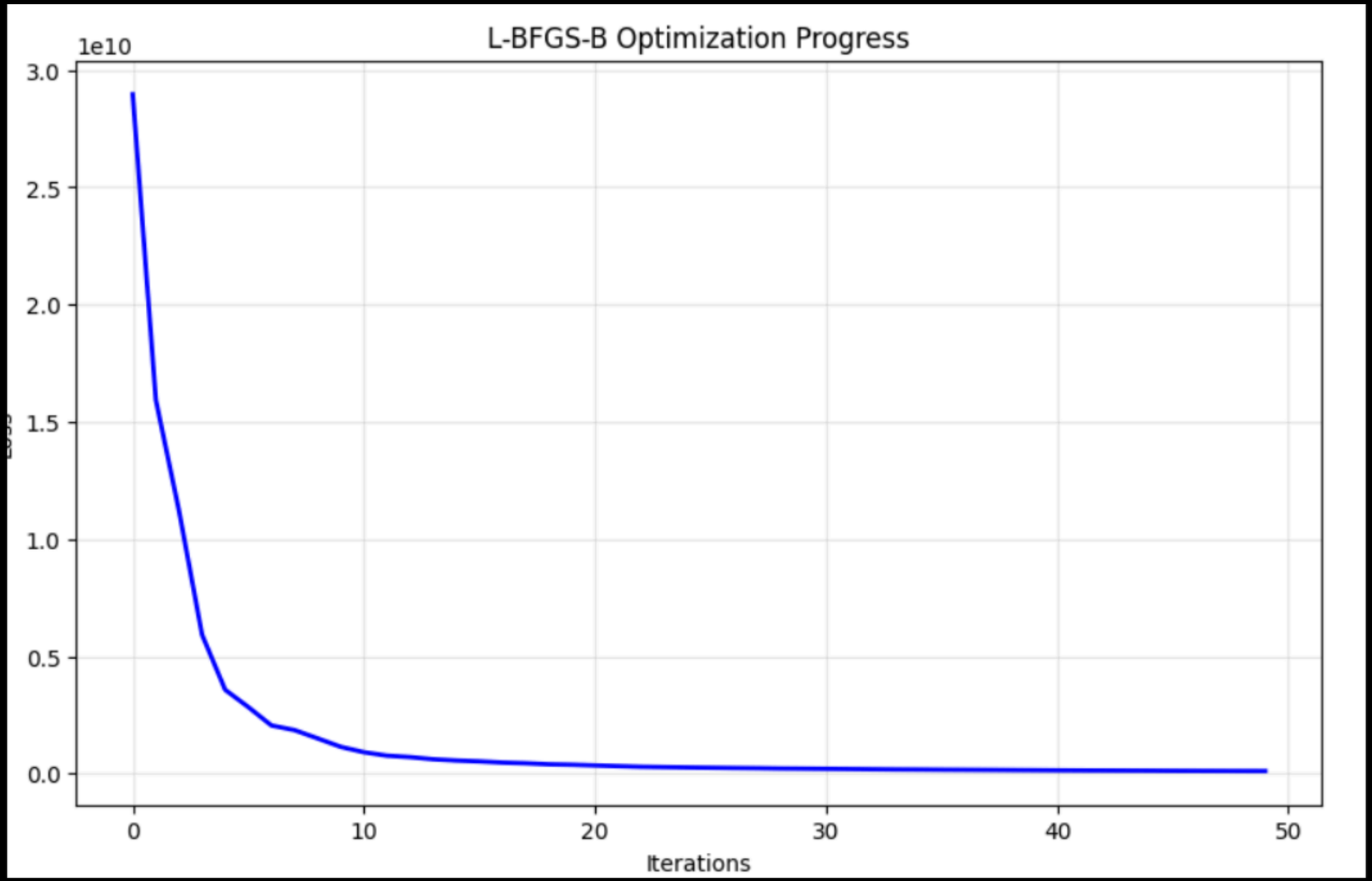
adam

Why L-BFGS-B?
It is a quasi-Newton method that can converge faster and produce sharper results.

It uses smarter mathematics by estimating the curvature of the loss landscape, not just the slope.

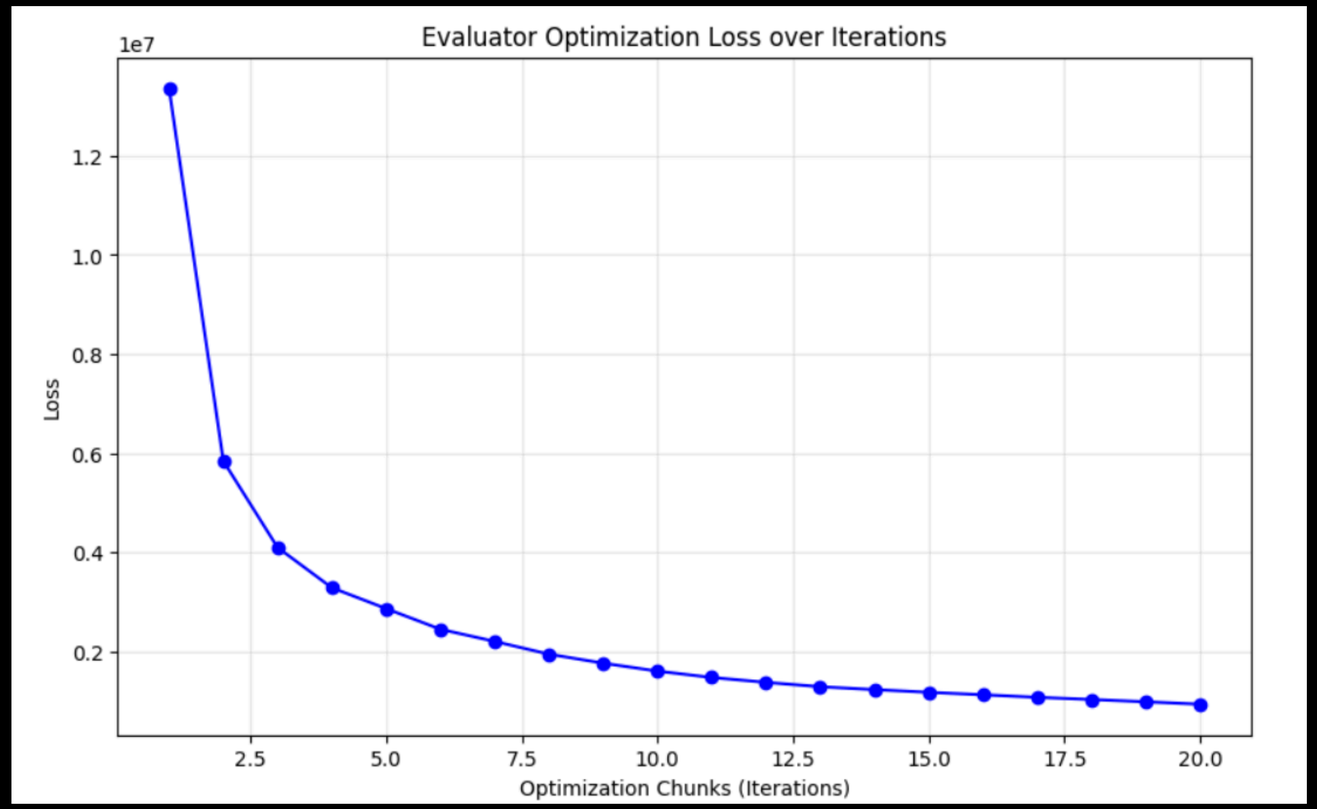
L-BFGS-B




L-BFGS-B Optimization Progress
1e10
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50
Iterations

L-BFGS-B

A New Strategy and a Hidden Cost (Evaluator)



The `scipy.optimize` function requests loss and gradients in separate calls. A naive implementation calculates the full model pass twice-once for the loss and again for the gradients, which doubles the runtime.

The Problem:
Redundant computations because the optimizer asks for loss and gradients in separate function calls. The Solution: An`Evaluator` class computes both loss and gradients in a single pass. It caches these results, so subsequent calls for either value within within the same optimization step are instantaneous.

The Solution:
An`Evaluator` class computes both loss and gradients in a single pass. It caches these results, so subsequent calls for either value within within the same optimization step are instantaneous.

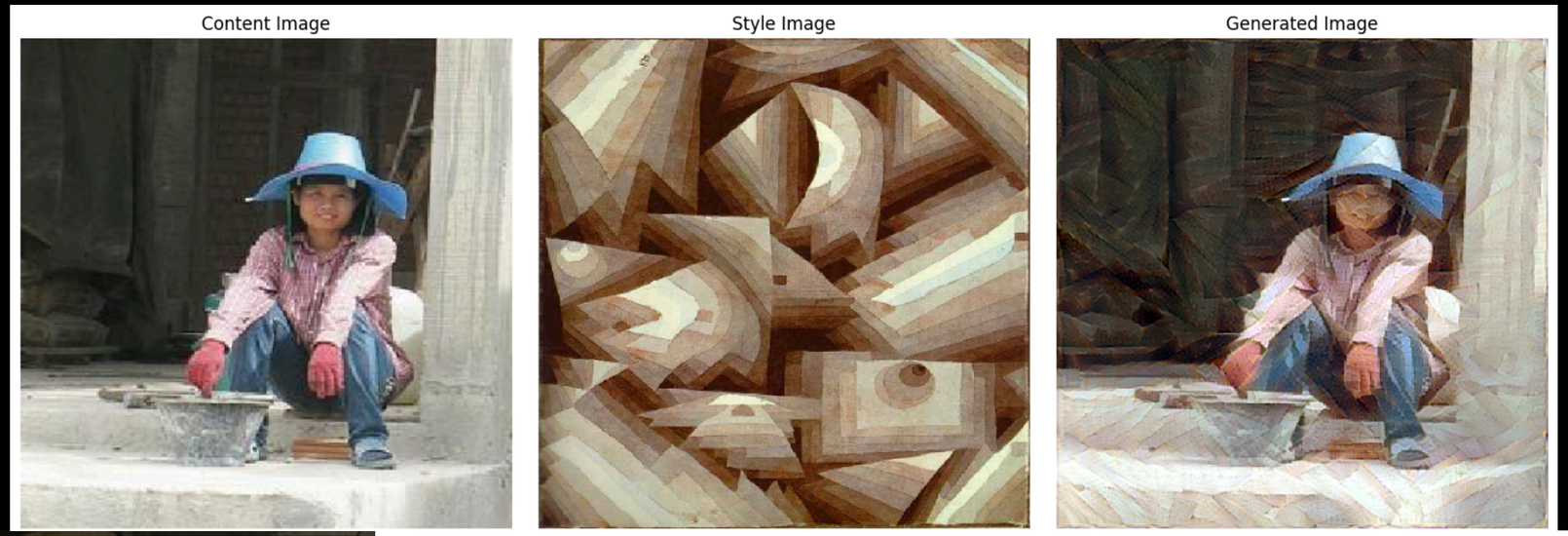
Maximum Efficiency: The Evaluator Pattern

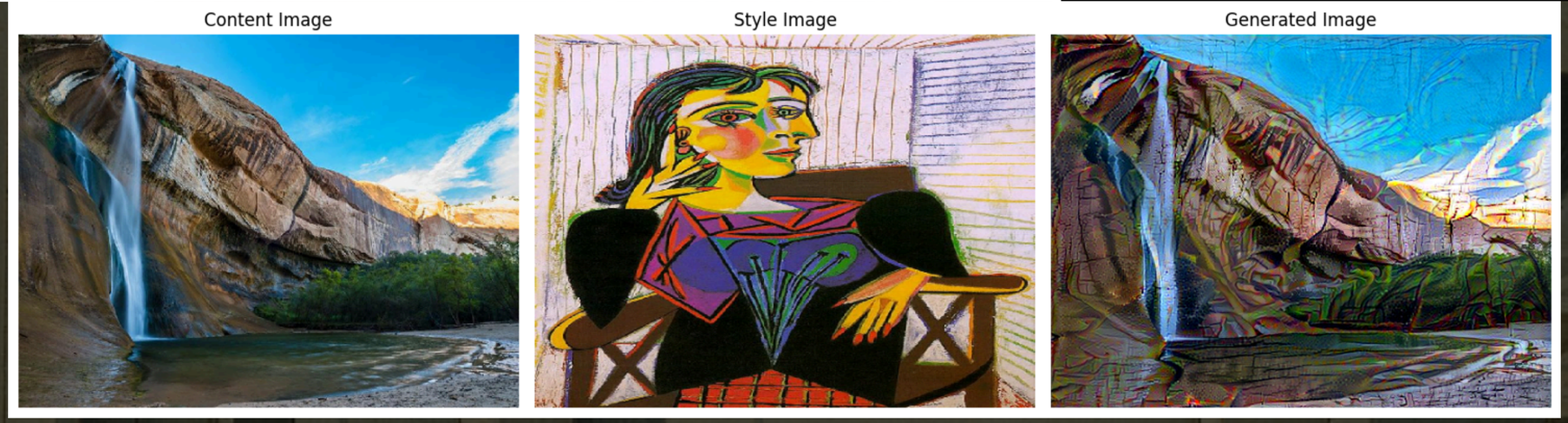
Performance Review
Adam vs. L-BFGS

L-BFGS HAS A "HEAVIER" COMPUTATIONAL COST PER STEP, BUT BECAUSE IT FINDS A MORE DIRECT PATH TO THE SOLUTION, THE TOTAL TIME TO REACH A HIGH-QUALITY RESULT IS MUCH FASTER.

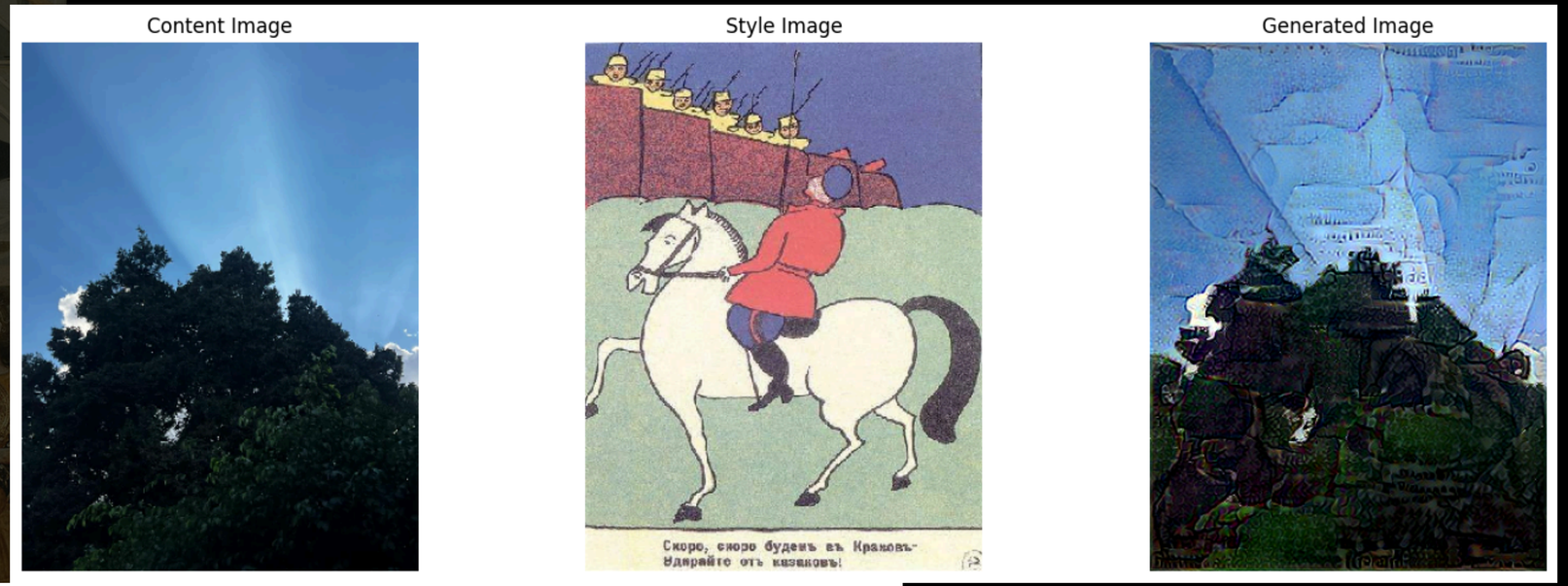
Metric	Adam	L-BFGS-B (with Evaluator)
Mechanism	First-Order (Gradient Descent)	Quasi-Newton (Approximates Curvature)
Iterations	Many	Few
Quality	Good	Often Sharper, Higher Fidelity
Implementation	Simpler, integrated into TF	More complex; requires Evaluator pattern
Best For	Robust baseline, easy to implement	Faster convergence to high quality, sharper results

The Versatility of the Algorithm





The Versatility of the Algorithm

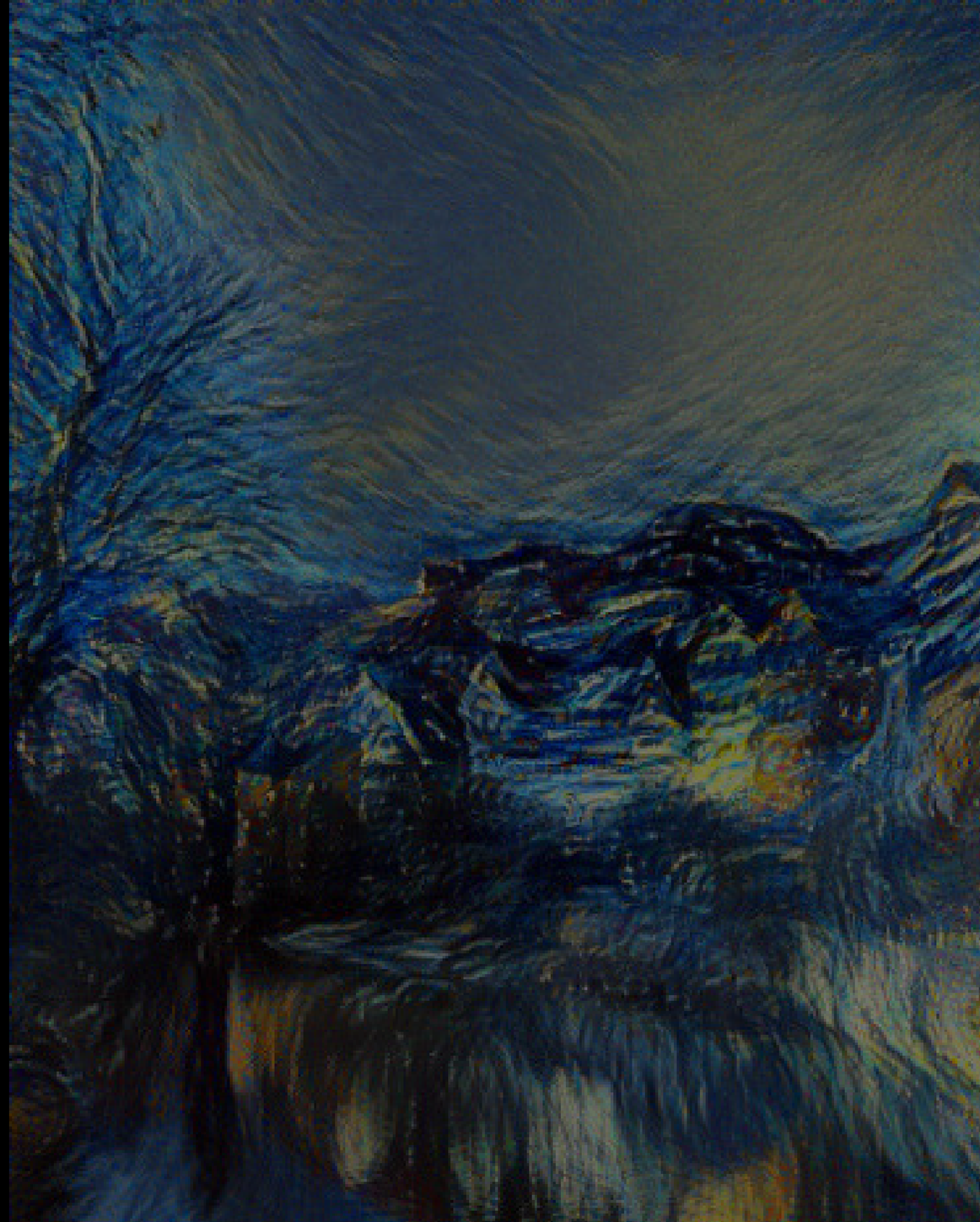


A Recap

1. hierarchies of a pre-trained CNN like VGG19. The process is an optimization problem, driven by a composite

2. loss function that balances structural preservation, stylistic mimicry, and image clarity. The choice of optimizer is critical. While Adam provides a solid

3. baseline, L-BFGS-B offers a faster path to sharper results, especially when implemented efficiently with a pattern like the Evaluator class.




Thank you
for your attention