ArtEdge

Real-Time Neural Style Transfer (NST) for Mobile

Nayeem Mohammad

Nitin Gupta

Project Overview

Neural style transfer (NST) is a computer vision technique that applies the **artistic style of one image to the content of another**.

ArtEdge develops a mobile-first solution for **real-time neural style transfer**, allowing users to transform ordinary photos into their artistic variants directly on their mobile devices.

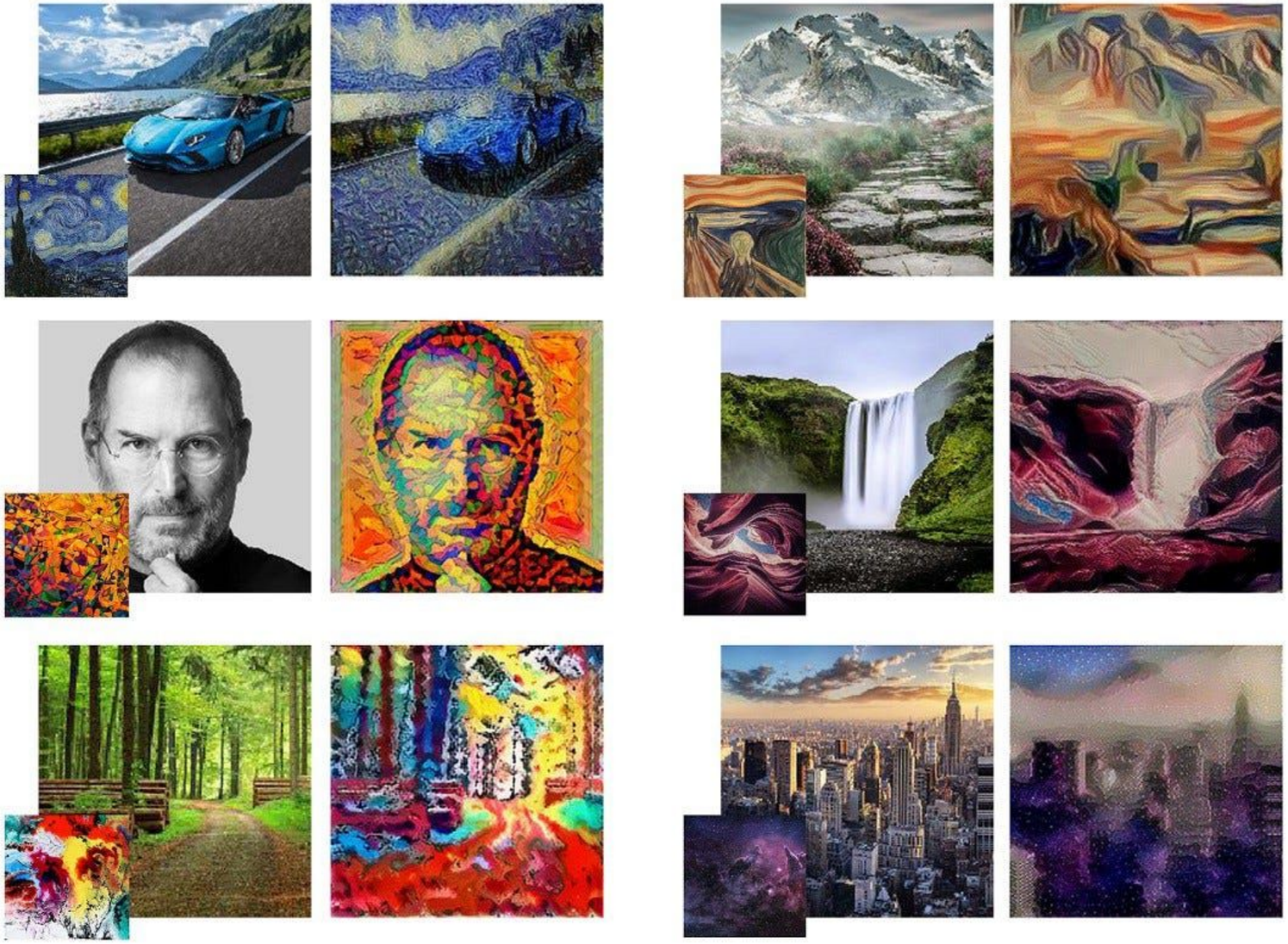


Image Source: Creatives need neural "concept" style transfer

Project Importance

Enhanced privacy by **processing sensitive images locally** rather than in the cloud

Democratization of artistic tools, making sophisticated AI art accessible to everyday users

Creating **more inclusive visual communication** technologies

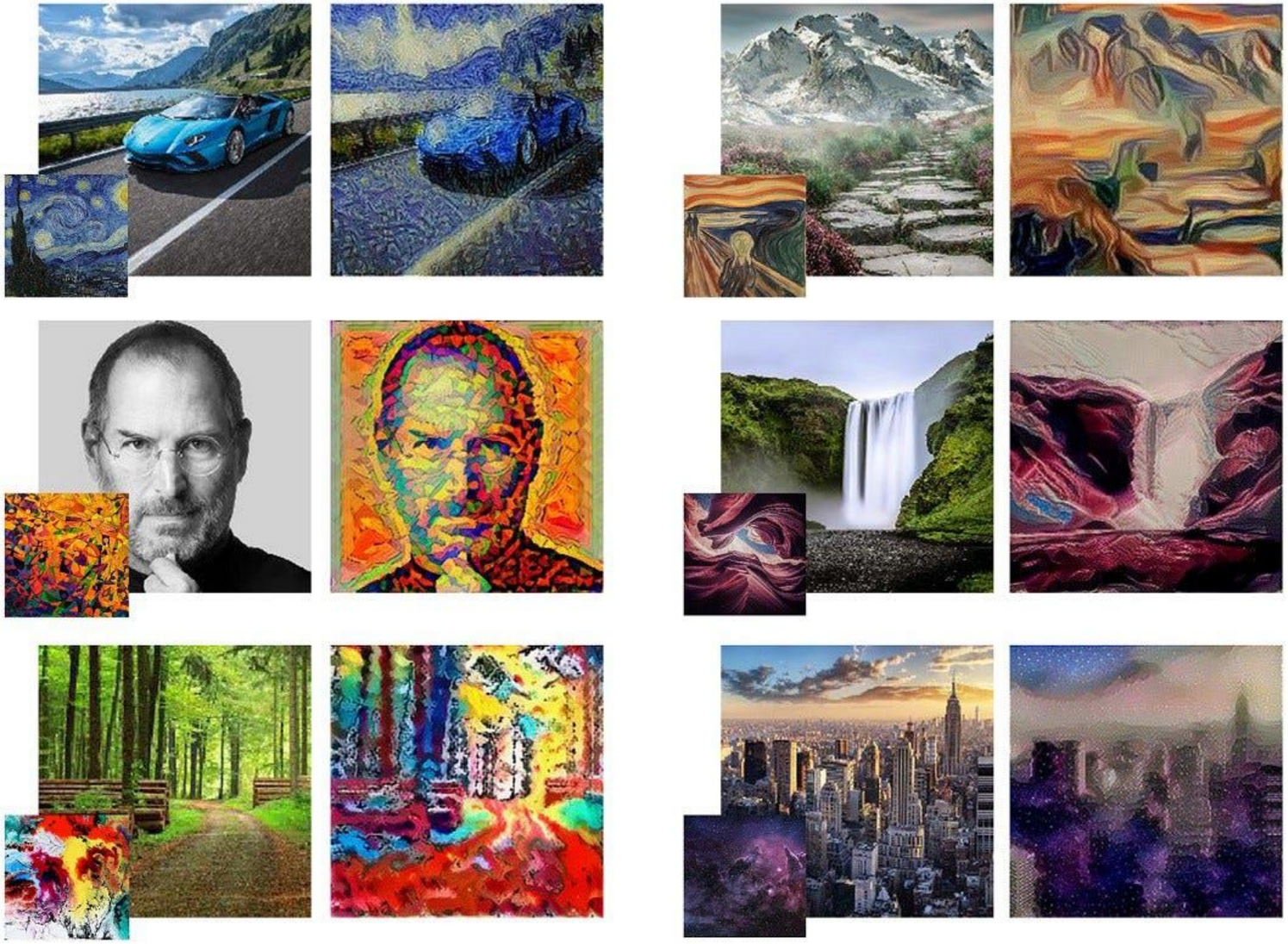
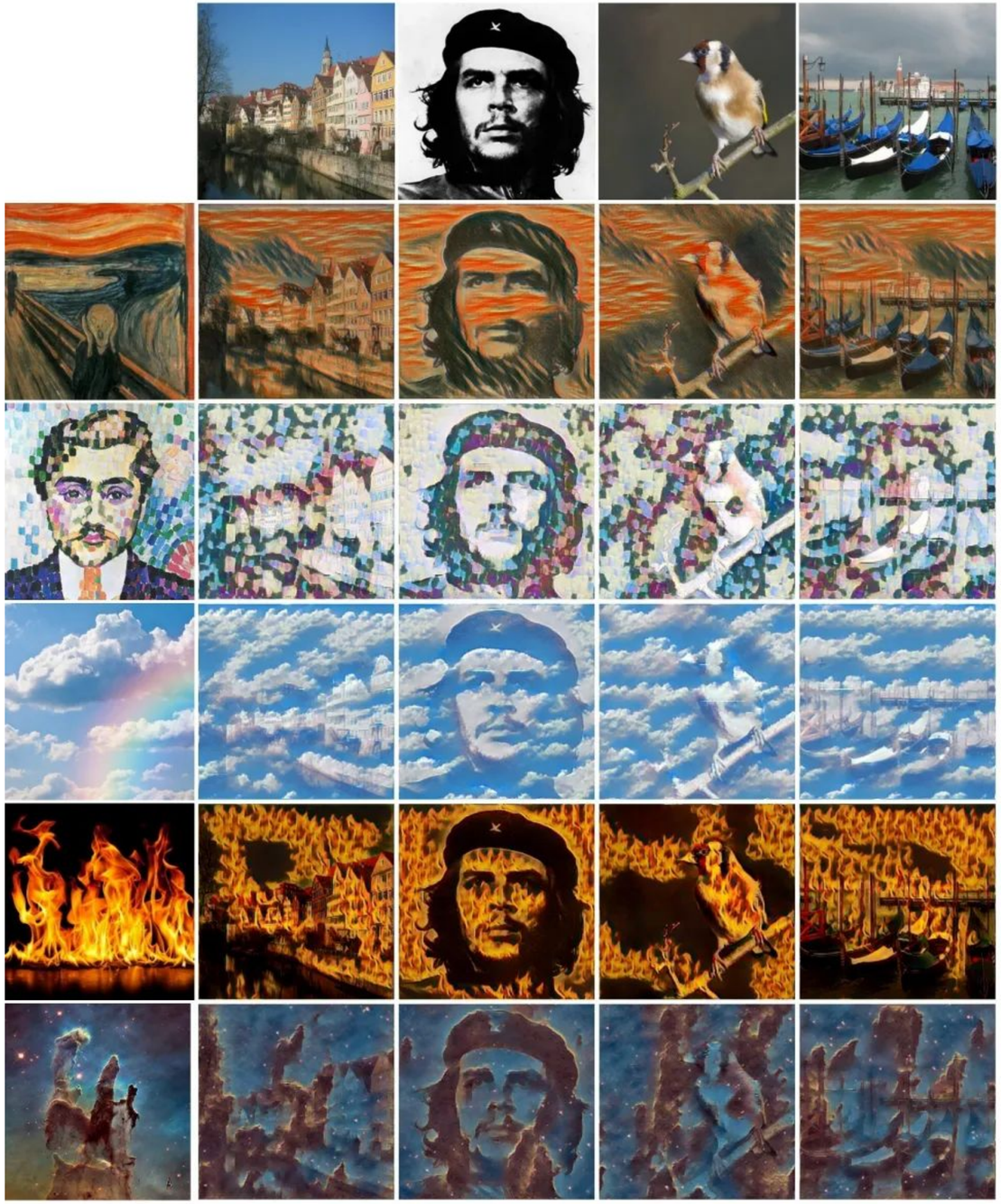


Image Source: Creatives need neural "concept" style transfer



Edge Computing Relevance

Moving computation from the cloud to the device, **reducing latency** from seconds to milliseconds

Enhancing privacy by keeping personal photos on-device

Enabling **offline functionality** for users without reliable connectivity

Developing domain-agnostic techniques, applicable to other edge AI applications

Image Source: Towards Fast Neural Style Transfer | by Connor Shorten | TDS Archive | Medium

Datasets

WikiArt: **~80,000 fine art paintings** as **2D color images** with variable resolutions (ranging from **500 to 3000 px**), featuring diverse artistic styles.

MS COCO: **~330,000 2D color images** containing **1.5 million object instances** with variable resolutions (typically around **640×480 px**), featuring diverse scenes, objects, and environments.

User-Generated Images

Custom **real-world 2D color** (RGB) photos captured on mobile devices features variable resolutions (typically **1080×1920 px**), representing diverse lighting conditions and subjects.

Image Sources: WikiArt Dataset | Papers With Code; COCO dataset; Person Holding Silver iPhone 6 | Unsplash

Literature Review

1. Original Neural Style Transfer

Challenge: Maintaining original image content while applying a completely different artistic aesthetic

Insights:

- Different CNN layers capture different visual information
- Lower layers: Texture and local patterns
- Higher layers: Semantic content and global structure
- Style is more than just color - it's about feature correlations and distributions

Source: A Neural Algorithm of Artistic Style (Gatys et al., 2016)

1. Original Neural Style Transfer

Used **pre-trained VGG-19** network

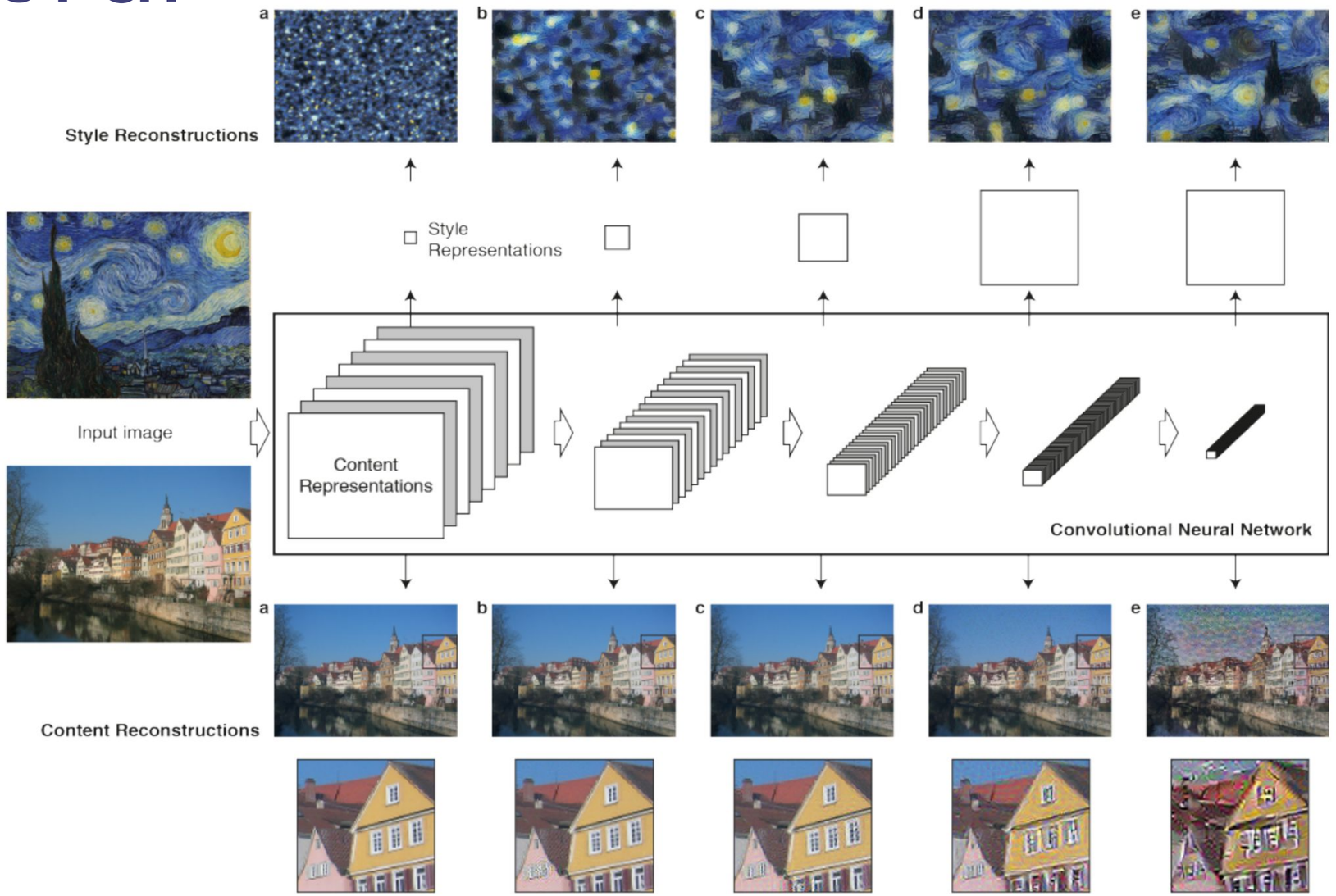
Leveraged features from multiple layers

Loss is a **weighted combination** of content and style losses

Iteratively updates input image

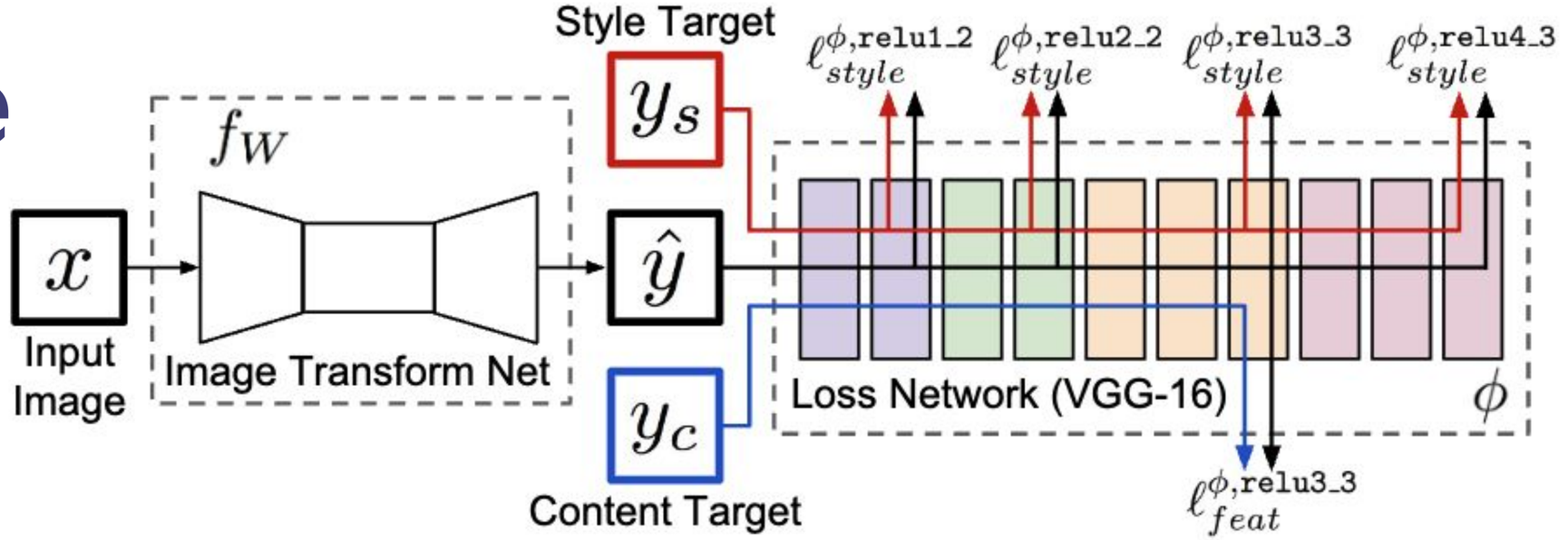
Worked across multiple artistic styles

Requires **significant computational resources**



Source: A Neural Algorithm of Artistic Style (Gatys et al., 2016)

2. Fast Style Transfer



Replaces traditional per-pixel loss with **perceptual loss** (based on high-level features from a pretrained CNN) to train feed-forward networks for Neural Style Transfer.

Training: Single-Style

(train one network per style image)

Perceptual Loss:

Feature Reconstruction Loss: Ensures content preservation using intermediate VGG features

Style Reconstruction Loss: Matches Gram matrices of features to capture textures and style

Source: (2016) [1603.08155] Perceptual Losses for Real-Time Style Transfer and Super-Resolution

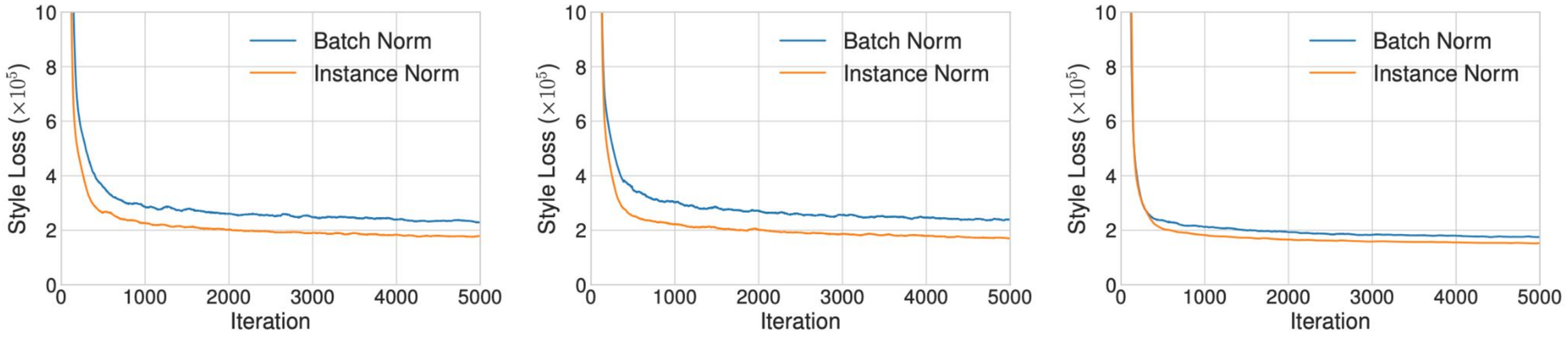
3. Improved NST: AdaIN

Batch Normalization	Instance Normalization
$\mathbf{BN}(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$	$\mathbf{IN}(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$
$\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$	$\mu_{nc}(x) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$

Combination of:

1. Flexibility of optimization-based methods
2. The speed of feed-forward approaches.

Idea: Adaptive Instance Normalization (AdaIN) simply **adjusts the mean and variance** of the content input to match those of the style input.

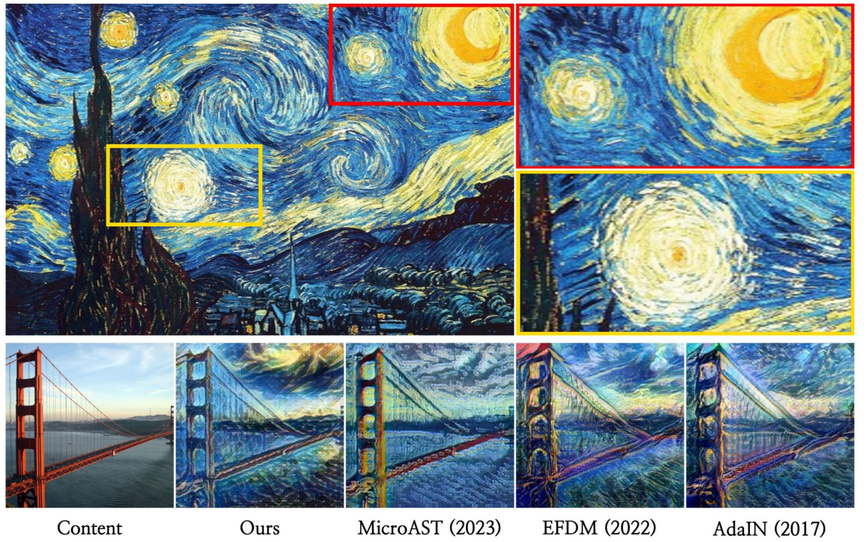


(a) Trained with original images. (b) Trained with contrast normalized images. (c) Trained with style normalized images.

Source: (2017) [1703.06868] Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

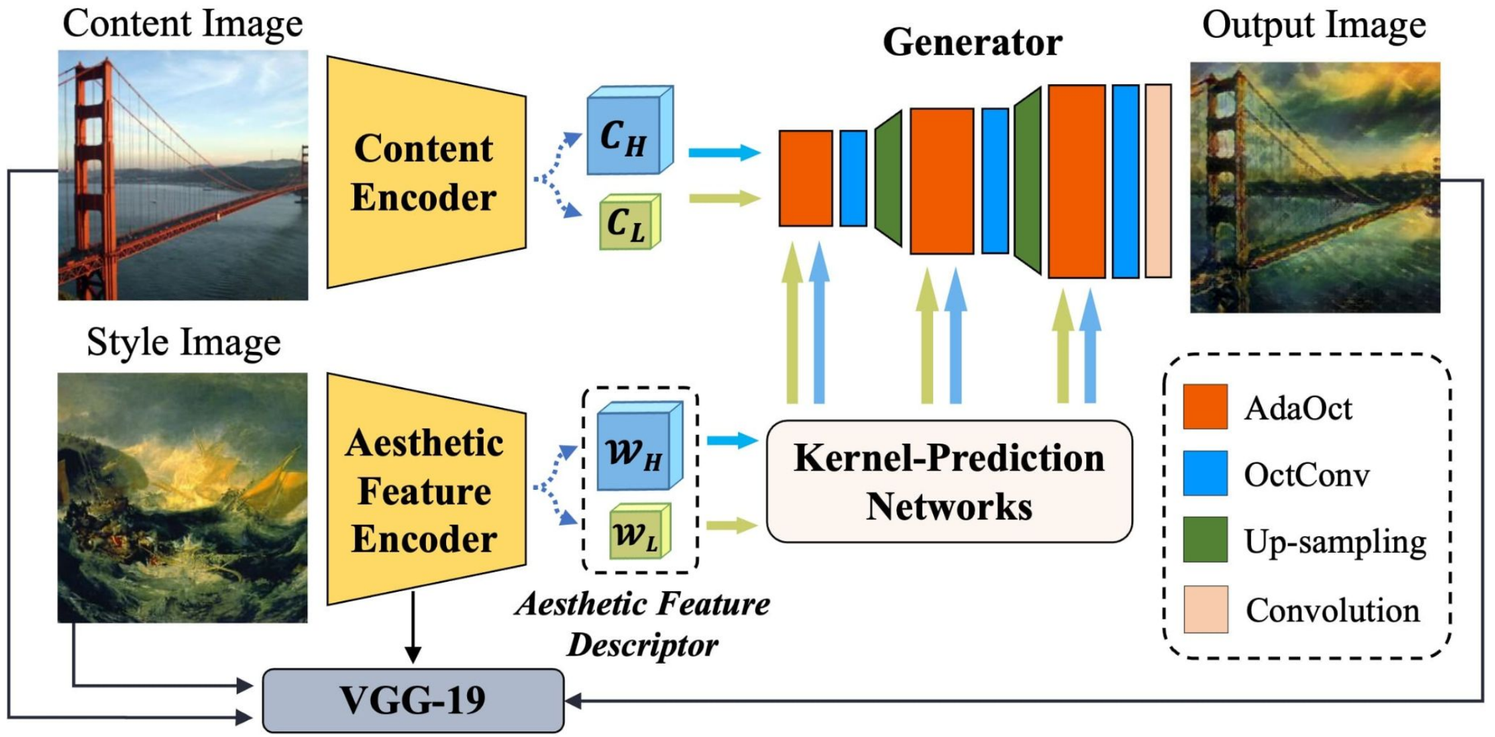
4. NST on Edge (AesFA)

Existing methods **lack spatial coherence** and **rely on heavy pre-trained models** making them inefficient for mobile/edge use.



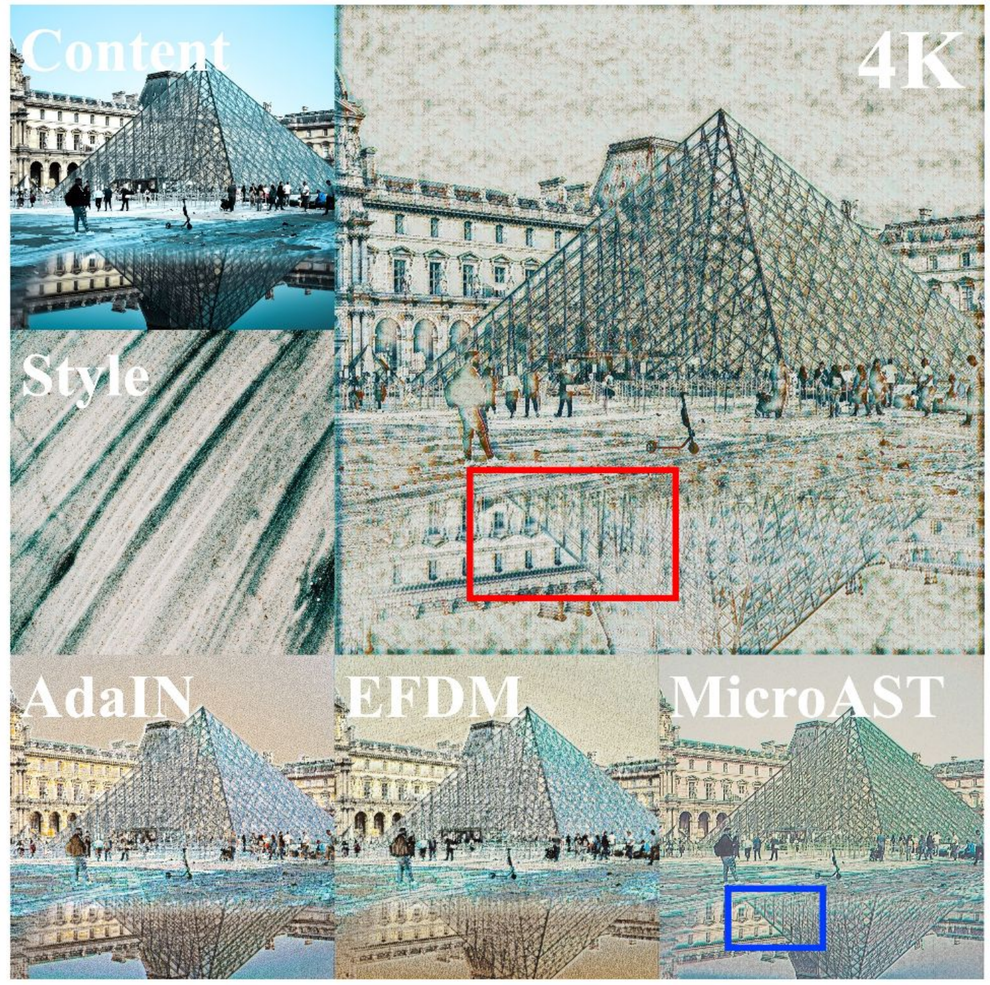
Innovations:

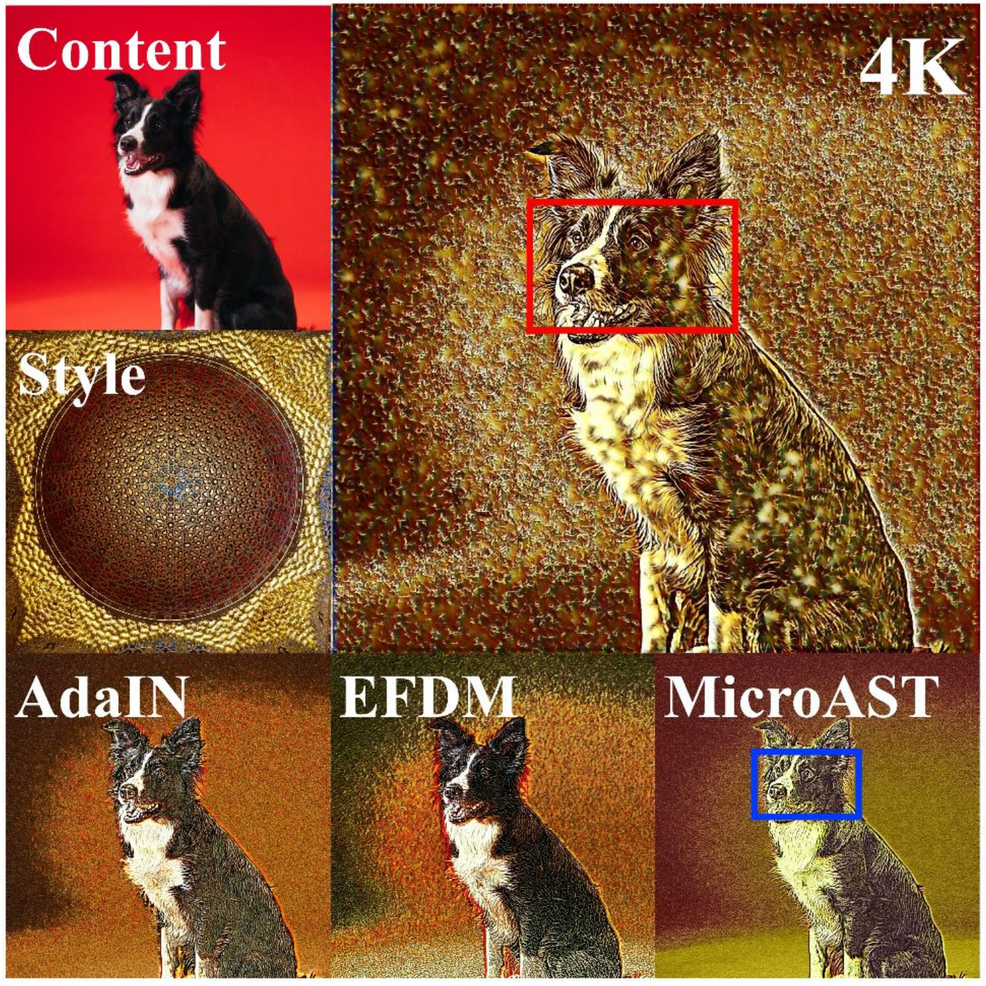
- Splits into **high-frequency** (textures/brushstrokes) and **low-frequency** (global tones) components
- Focuses on **hard negatives** for efficient training
- **No pre-trained model at inference**

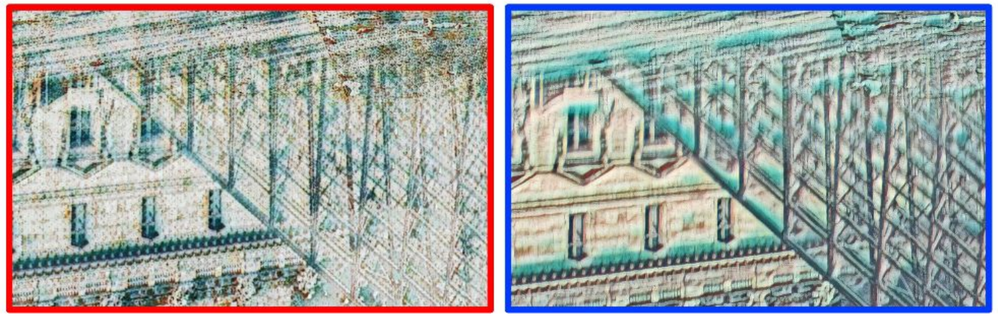


Source: (2024) [2312.05928] AesFA: An Aesthetic Feature-Aware Arbitrary Neural Style Transfer

4. NST on Edge (AesFA)

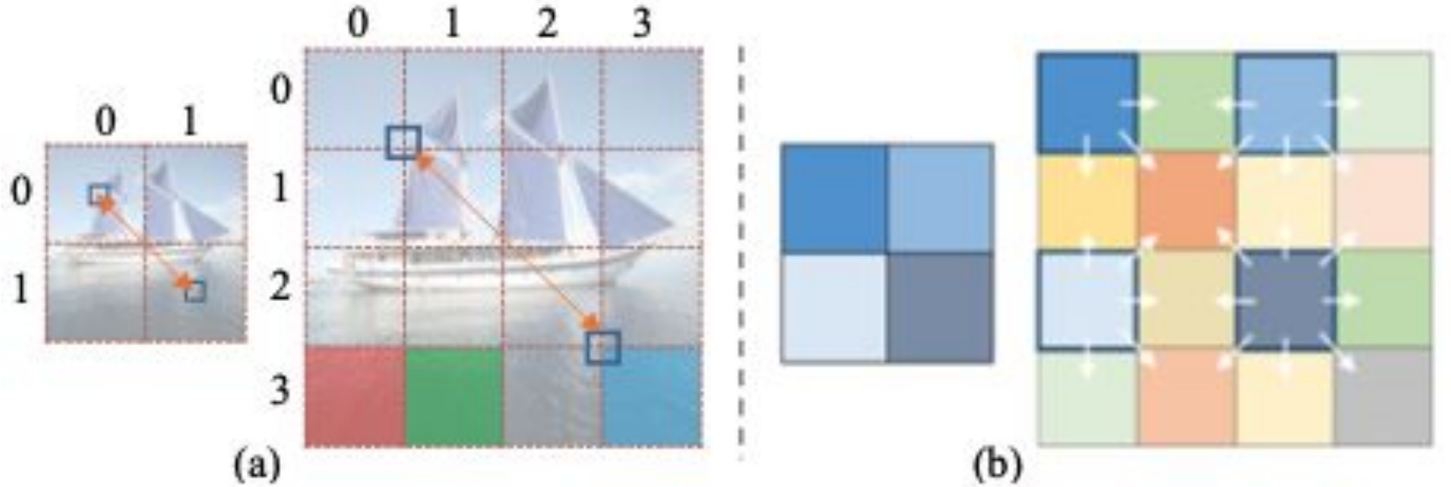






Source: (2024) [2312.05928] AesFA: An Aesthetic Feature-Aware Arbitrary Neural Style Transfe

5. NST w/ Transformers



Problem with CNNs:

- Limited receptive field → Loses global information
- Biased content representation
- Loss of fine details in deep networks

Why Transformers?

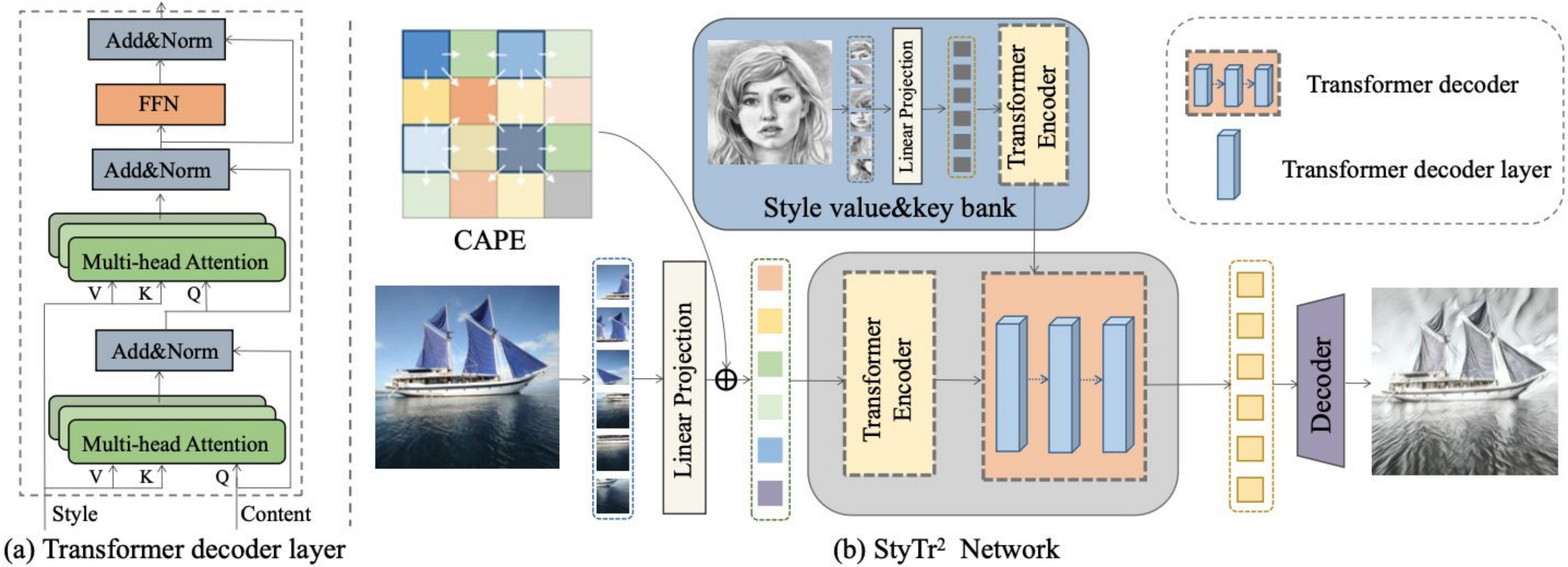
- Self-attention mechanism captures long-range dependencies
- Holistic understanding at each layer
- Strong content preservation & style adaptation

Key Contributions:

- Dual Transformer Encoders: Separate style & content feature extraction
- Transformer Decoder: Stylizes content sequence progressively
- Content-Aware Positional Encoding (CAPE): Scale-invariant and adjusts to image sizes

Source: (2021) [2105.14576] StyTr^2: Image Style Transfer with Transformers

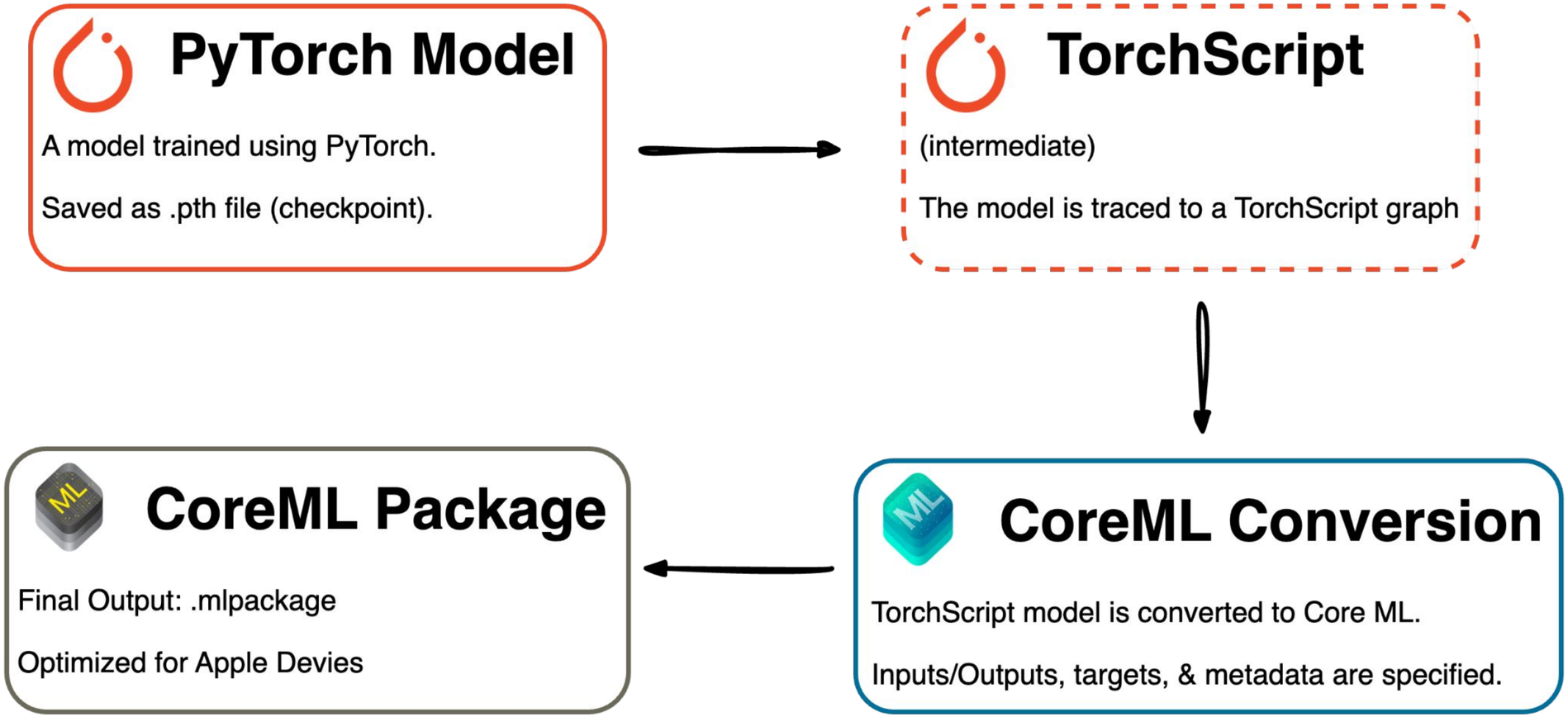
5. NST w/ Transformers



Source: (2021) [2105.14576] StyTr^2: Image Style Transfer with Transformers

Methodology

Workflow


PyTorch Model
A model trained using PyTorch.
Saved as .pth file (checkpoint).
TorchScript
(intermediate)
The model is traced to a TorchScript graph
CoreML Conversion
TorchScript model is converted to Core ML.
Inputs/Outputs, targets, & metadata are specified.
CoreML Package
Final Output: .mlpackage
Optimized for Apple Devies

Model-Specific Adaptations

MobileNet_AdaIN

Innovation: (Encoder → AdaIN → Decoder)

Uses ***MobileNetV2*** (up to layer 13), because it's lightweight and mobile-friendly, instead of VGG-based encoders for feature extraction

Implements AdaIN

Uses a separate VGG19-based **perceptual loss** network for:

Content loss: comparing relu4_1 features

Style loss: Gram matrix comparison of relu1_1 to relu3_1 layers

Training: ~4 hours, 20 epochs, 20k images,

Fast Style Transfer

It was trained in a model-per-style approach and we used pre-trained models for inference

Model-Specific Adaptations

StyTr2

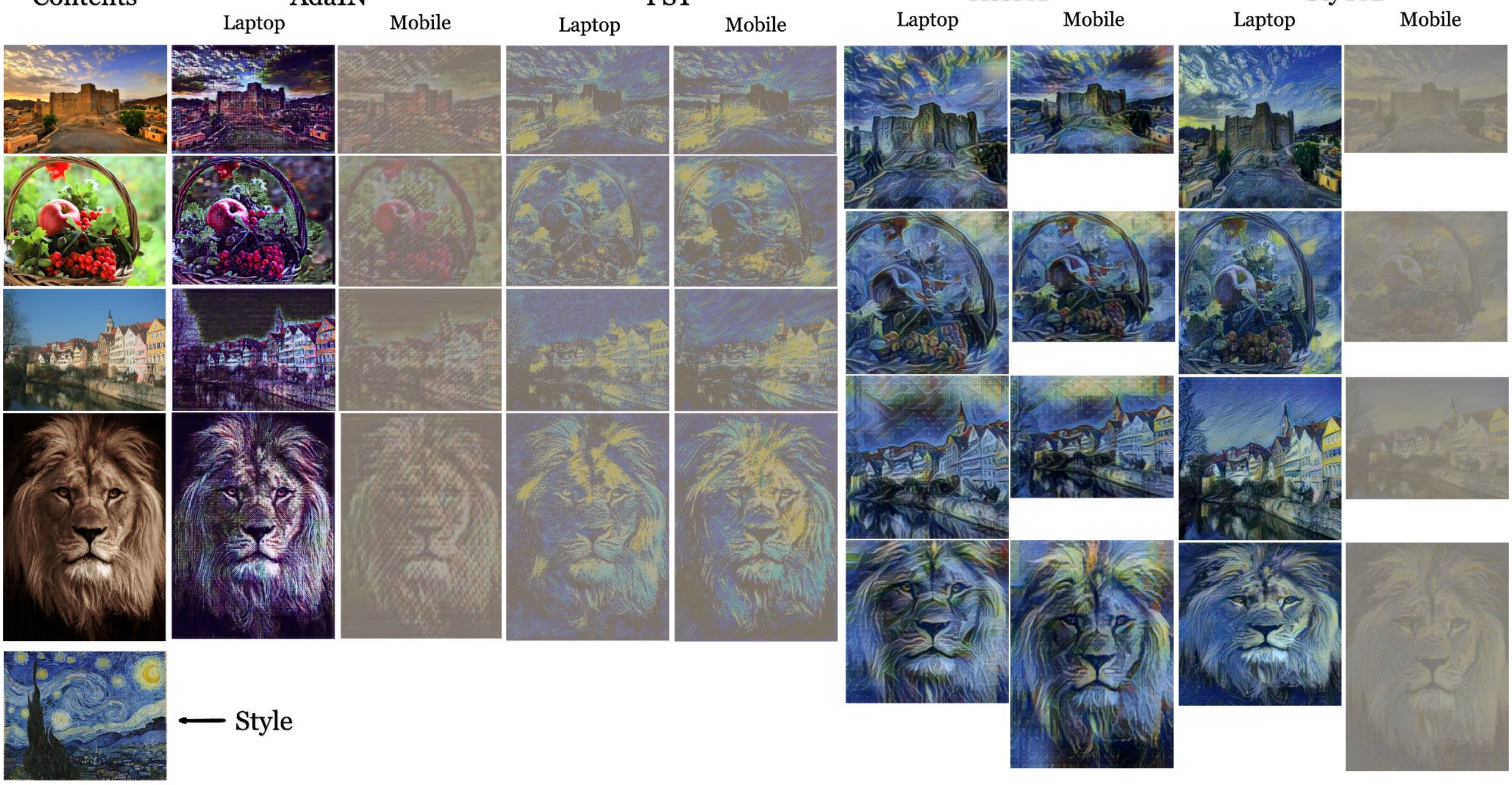
Deprecated implementations and version-specific-bugs were replaced e.g., torchvision.ops.misc package → replaced with hand-coded functions

AesFA

Mobile Optimization: Reworked key parts of the network architecture to make it compatible and efficient for running directly on mobile devices.

Model & Loss Improvements: Made internal adjustments to the model's layers and how it calculates style similarity (loss) for better overall performance and stability.

Results


Contents
AdaIN
Laptop
Mobile
FST
Laptop
Mobile
AesFA
Laptop
Mobile
StyTr2
Laptop
Mobile
Style

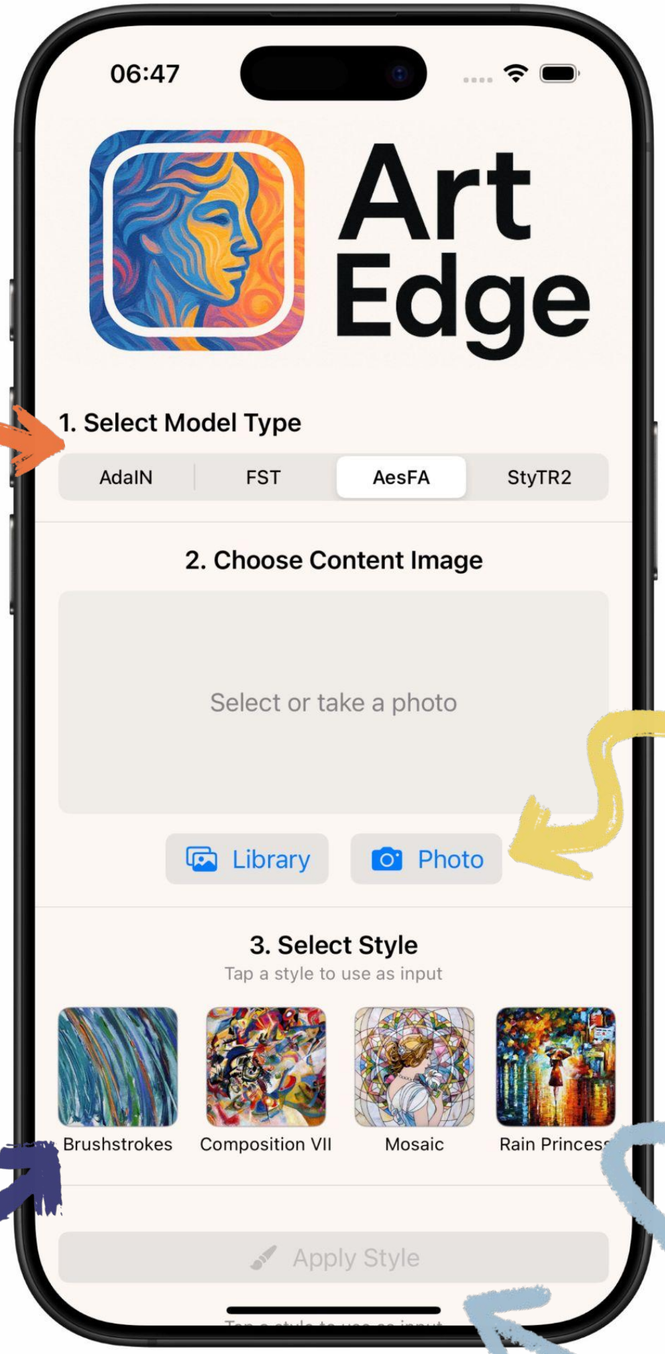
DEMO

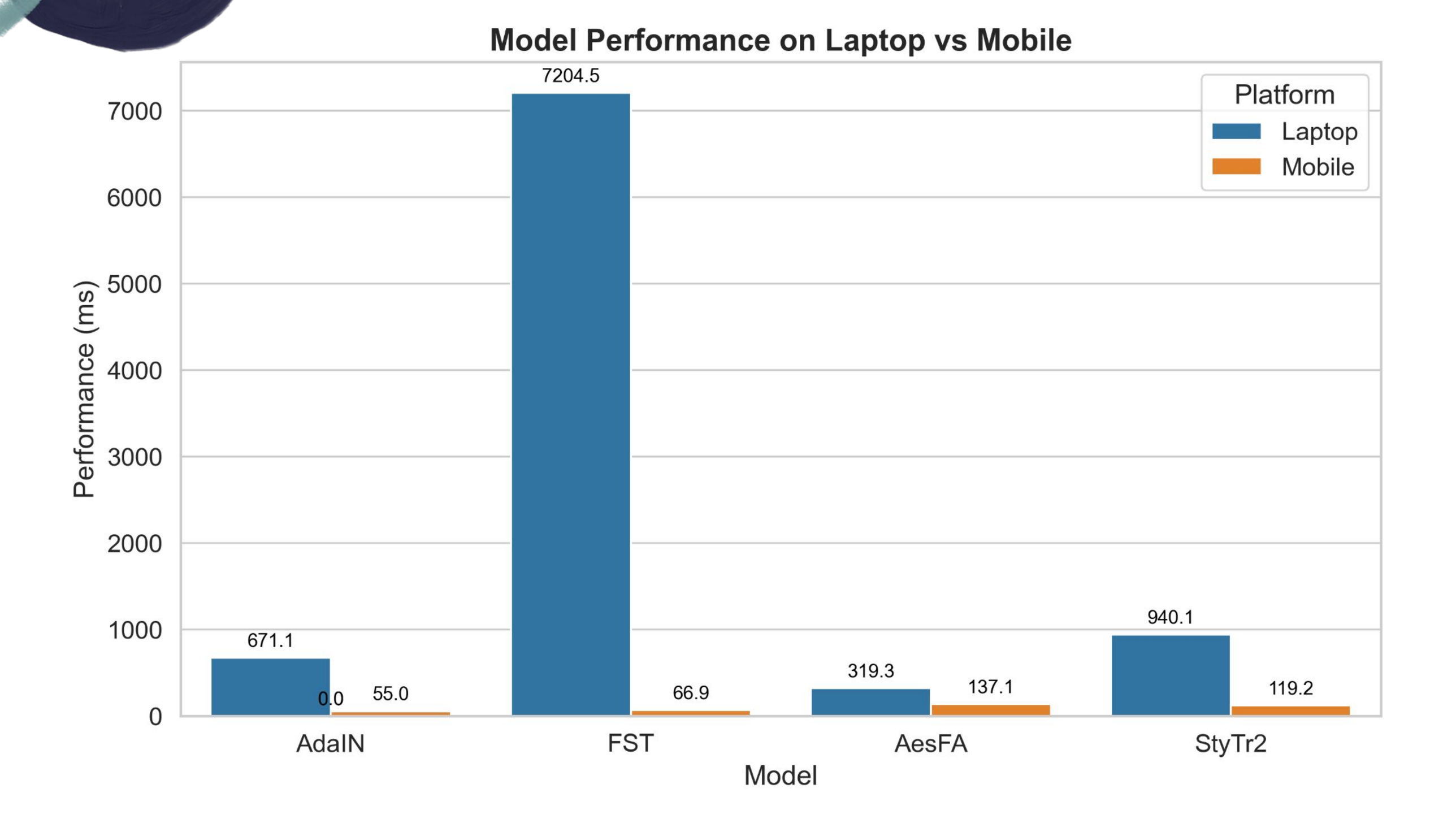
Select Different Models on Runtime

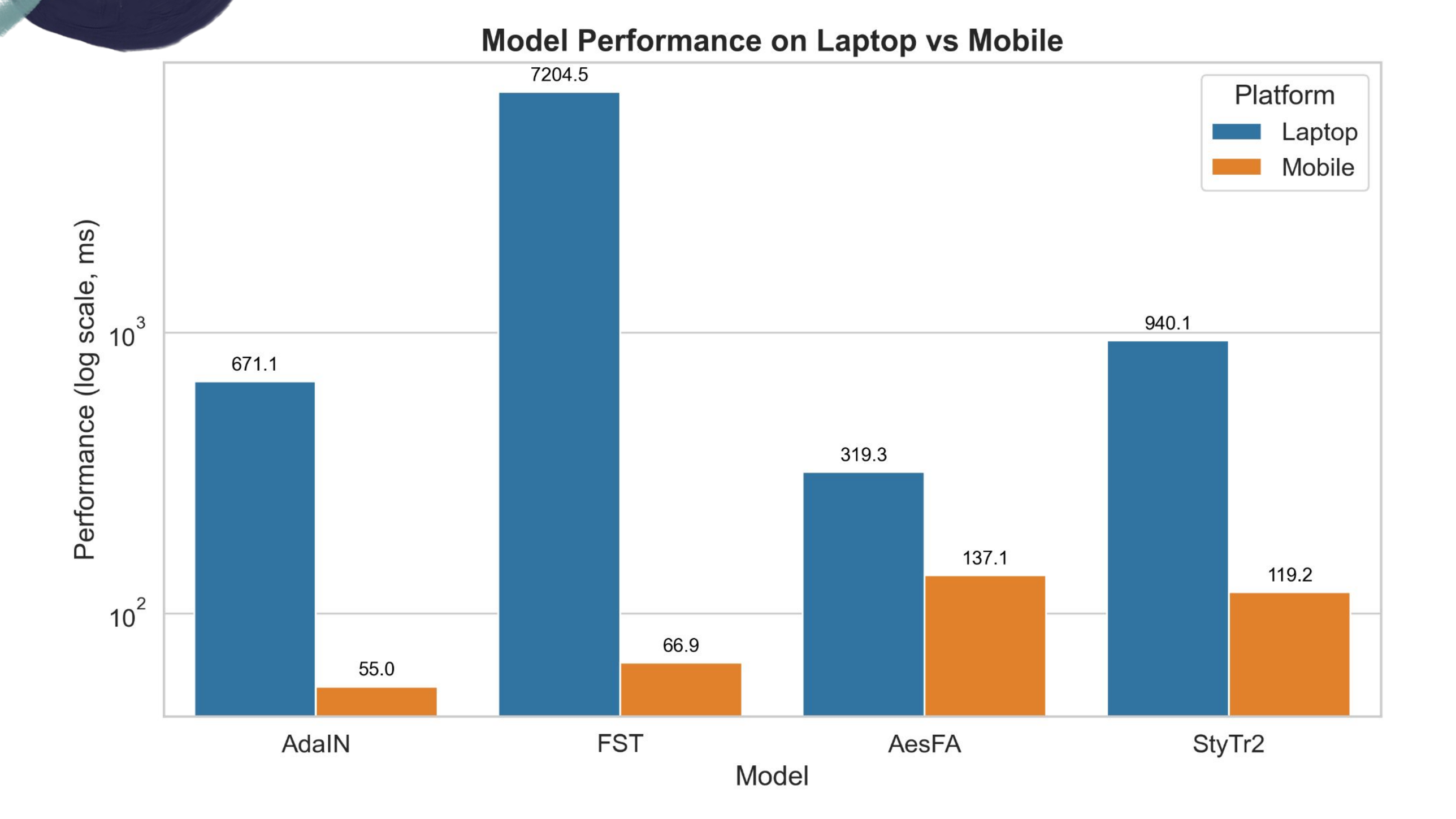
Choose From Your Phone Library or Take a Photo

Choose From Pre-Selected Styles

Apply Style Transfer!




Model Performance on Laptop vs Mobile
Platform
Laptop
Mobile
Performance (ms)
0
1000
2000
3000
4000
5000
6000
7000
671.1
0.0
55.0
7204.5
66.9
319.3
137.1
940.1
119.2
AdaIN
FST
AesFA
StyTr2
Model


Model Performance on Laptop vs Mobile
Platform
Laptop
Mobile
Performance (log scale, ms)
10^3
10^2
671.1
55.0
7204.5
66.9
319.3
137.1
940.1
119.2
AdaIN
FST
AesFA
StyTr2
Model

Conclusion

Integrated four NST models: **MobileNet AdaIN**, **Fast Style Transfer**, **AesFA**, and **StyleTr²** into a mobile deployment pipeline.

On laptop, all models produce high-quality stylized images, visually faithful to the target style.

On mobile, **Fast Style Transfer** and **AesFA** deliver decent results with good style retention and for speed AesFA is better than FST.

AesFA offers the most visually appealing output. **StyleTr²** is close but suffers from **lighting artifacts on mobile**, making results appear faded.

Future work will focus on **lighting normalization** and mobile-specific fine-tuning to enhance cross-device visual consistency.

Thank You!

SlidesCarnival
for the presentation template

Pexels
for the art photos