# Bring Van Gogh Back to Life: Style Transfer & Classification

**Introduction to Deep Learning – Final Project 2025/2026**

**Course:** 0571418201 |

## Executive Summary

We built a Van Gogh classification and style transfer pipeline using the Post-Impressionism dataset. **Part 1:** VGG-19 (94.6% accuracy, AUC 0.976) and AlexNet (92.7%, AUC 0.971) were fine-tuned to distinguish Van Gogh from other artists. **Part 2:** Neural style transfer was implemented; Optuna tuned hyperparameters ($\alpha$=0.089, $\beta$=8.79e6) to maximize P(van_gogh). Classifiers agreed on 14/20 VGG-style and 19/20 Alex-style outputs as Van Gogh-like. **Part 3:** GPU (RTX 4080) achieved ~7× speedup over CPU. **Bonus:** Grad-CAM heatmaps showed the classifier attends to brushstroke texture and expressive color—explaining both correct identifications and false positives. The project is complete and ready for submission.

## Submission Info

- **Group number:** 026
- **Team members:** Mikhail Baklanov (baklanov@mail.tau.ac.il)
- **Submission due:** 25.1.26
- **Archive name:** `Group_026_DL_Project.zip` (PDF + .ipynb + scripts)

## Part 1 Discussion

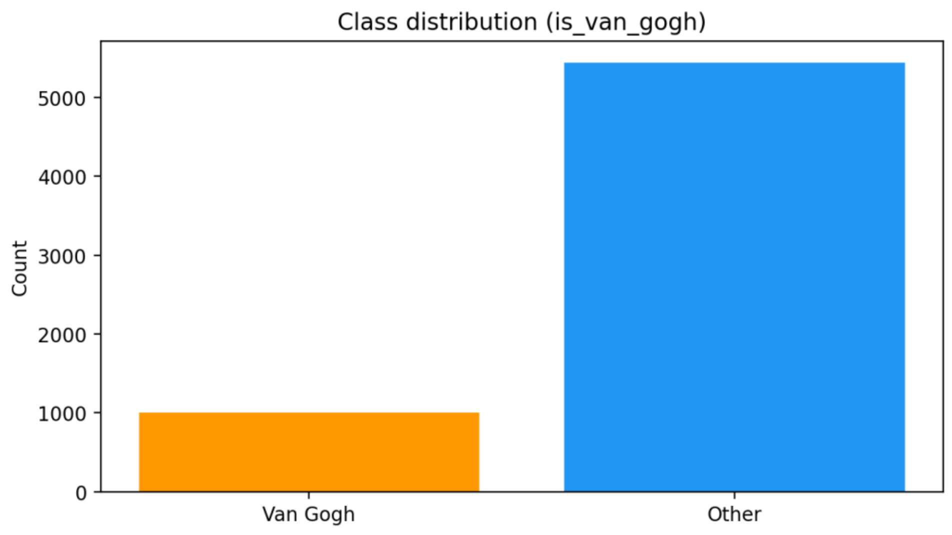
### 1. Dataset

**Overview:** We used the Post-Impressionism dataset containing paintings by Van Gogh and other Post-Impressionist artists. The dataset is available from Lab 424 (`D:\course_data`) or from Kaggle WIKIART (filtered to Post-Impressionism).

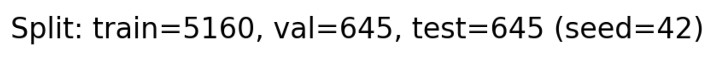
**Proportion of Van Gogh paintings:** 1005 Van Gogh out of 6450 total ≈ 15.6%.

**Labels:** Binary labels (is_van_gogh: yes/no) were created from the `classes.csv` metadata file (artist name "Vincent van Gogh") or, when unavailable, from the filename pattern `vincent-van-gogh_*.jpg`.

**Data split:** 80% train (5160), 10% validation (645), 10% test (645).



Class distribution



```
Split: train=5160, val=645, test=645 (seed=42)
```

Train/val/test split

**Discussion:** The dataset is **imbalanced** (only ~15.6% positives). Because of this, **accuracy alone can be misleading** (a naive “always Other” classifier would still score ~84%). Therefore, we report **AUC-ROC** (ranking quality) and **F1-score** (precision–recall balance) in addition to accuracy. We also analyze the confusion matrix to understand the types of errors made by each model.

## 2. Models & Transformations

**VGG-19:**

- Architecture: 19-layer convolutional network (16 conv + 3 FC) with ReLU activations
- Pre-trained on ImageNet (1.2M images, 1000 classes)
- Input size: 224×224

- Both VGG and AlexNet use the same ImageNet normalization (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

**AlexNet:**

- Architecture: 8-layer network (5 conv + 3 FC), shallower than VGG-19
- Pre-trained on ImageNet
- Input size: 224×224

**Key differences:** VGG-19 is deeper with smaller 3×3 filters, leading to richer feature hierarchies; AlexNet is shallower and faster. Both were fine-tuned by replacing the final FC layer for binary classification.

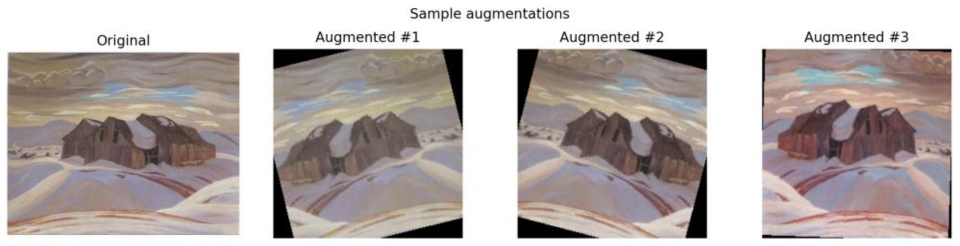
**Transformations (validation/test):**

- Resize to 256×256
- CenterCrop to 224×224
- ToTensor
- Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

**Data augmentation (training):**

- RandomResizedCrop(224, scale=(0.8, 1.0))
- RandomHorizontalFlip()
- ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.05)
- RandomRotation(15°)

These augmentations improve robustness to geometric and color variations across paintings.



Sample augmentations

## 3. Training & Tuning

**Approach:** Optuna was used for hyperparameter search over learning rate, batch size, and optimizer type. Final models were trained with the best hyperparameters for more epochs.

**Optuna search (required to report):**

- Number of trials run: 16 total (8 per model: VGG-19 and AlexNet)
- Epochs per trial: 3
- Approximate wall-clock time of search: ~97 minutes total (from W&B `_runtime` for the Optuna runs: ~51.5 min + ~45.9 min).

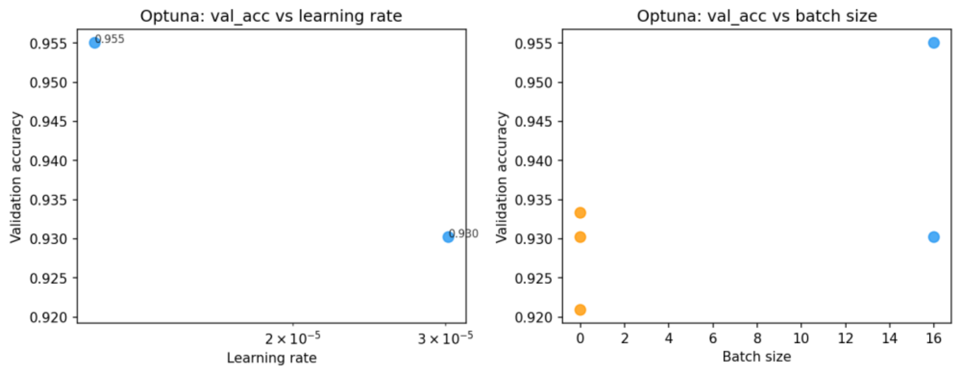
**Final hyperparameters:**

- **VGG-19:** AdamW, lr ≈ 1.18e-5, batch_size = 16 (val_acc ≈ 0.950)
- **AlexNet:** SGD, lr = 9e-5, batch_size = 16 (val_acc ≈ 0.930)

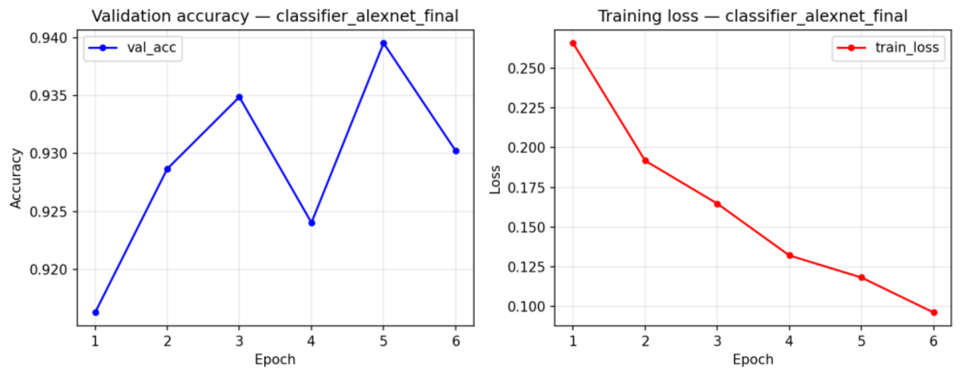
**W&B & Optuna:**

- All runs were logged to Weights & Biases (project: van-gogh-style-transfer)
- Logged: hyperparameters, validation accuracy, learning curves

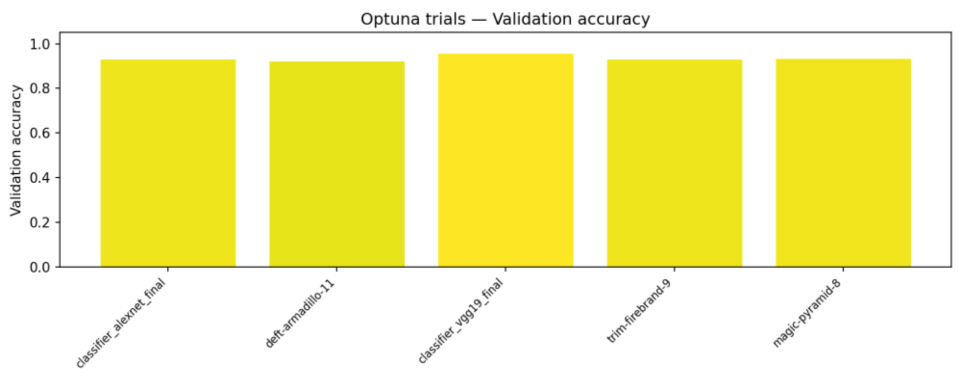
**Screenshots (generated from W&B API):**



Optuna trials: val_acc vs learning rate and batch size

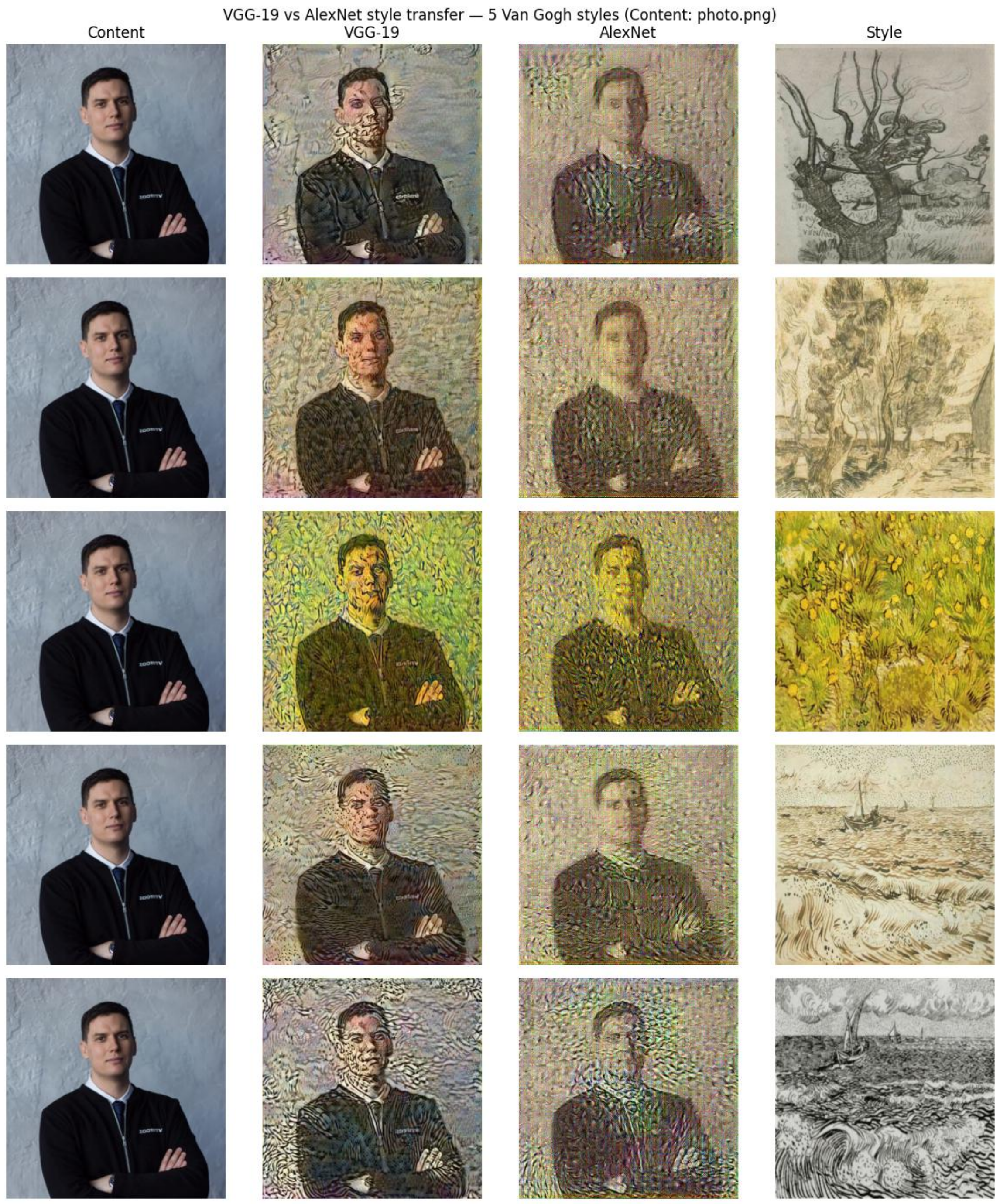


Learning curves: validation accuracy and training loss



**Optuna trials comparison**

**Discussion:** We used Optuna to systematically explore optimizers and learning rates, and W&B to compare trials. The scatter plots show how validation accuracy varies with learning rate and batch size, and the learning-curve plot highlights convergence behavior. In our runs, VGG-19 converged to higher validation accuracy and lower loss than AlexNet, consistent with the stronger final test-set performance.


VGG-19 vs AlexNet style transfer — 5 Van Gogh styles (Content: photo.png)
Content
VGG-19
AlexNet
Style

## 4. Model Comparison

| Model | Accuracy | AUC-ROC | F1-Score |
|---|---|---|---|
| --------- | ---------- | --------- | ---------- |
| VGG-19 | 0.946 | 0.976 | 0.821 |
| AlexNet | 0.927 | 0.971 | 0.761 |

**Analysis:** VGG-19 outperformed AlexNet on all metrics. The deeper architecture (19 layers vs 8) provides richer feature hierarchies for distinguishing Van Gogh's stylistic cues. AlexNet's lower F1 (0.761 vs 0.821) reflects a less favorable precision–recall balance, driven by both **more false positives** (16 vs 9) and **more false negatives** (31 vs 26) on the test split.

## 5. Prediction Analysis

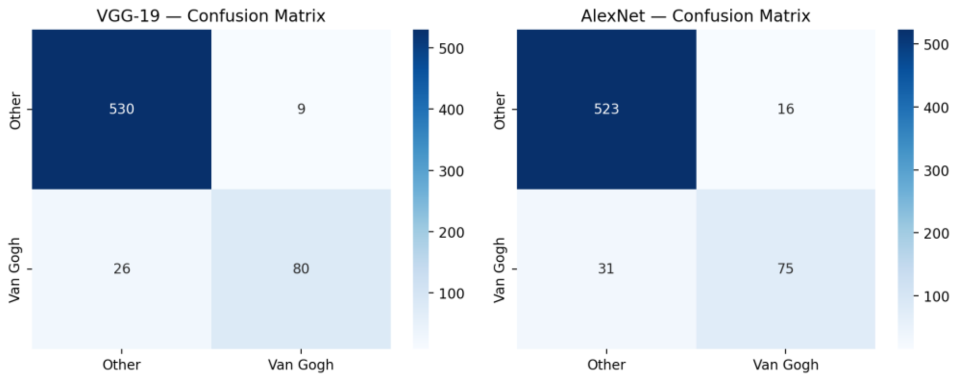
**Confusion matrices (rows = true, cols = predicted; 0 = other, 1 = van_gogh):**

VGG-19:

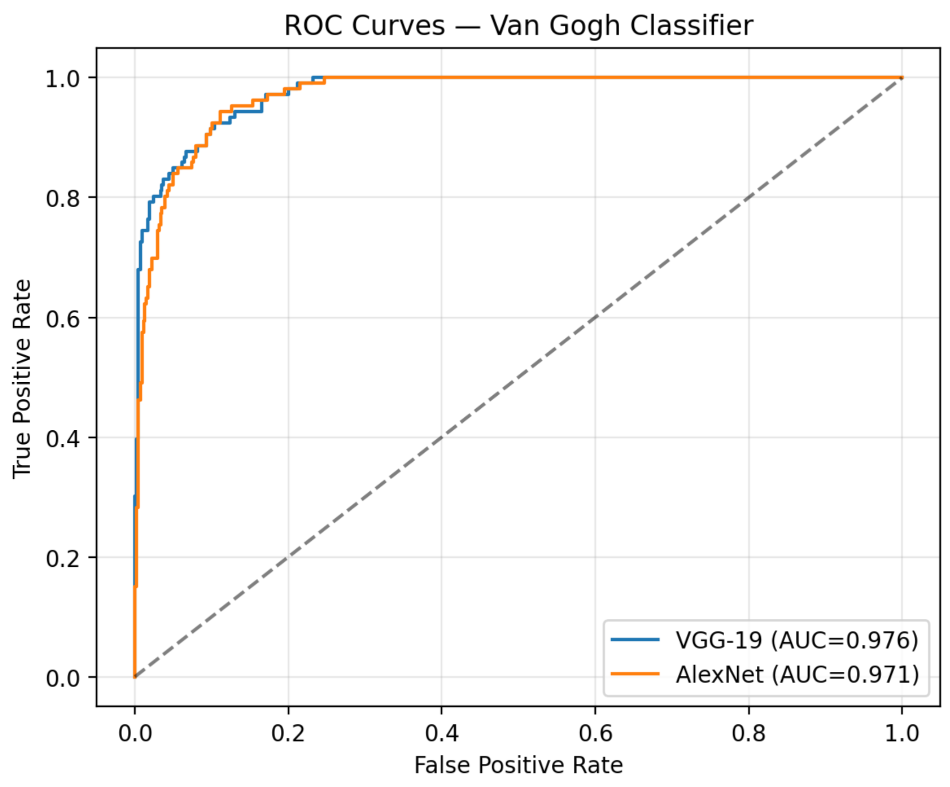
```
            Pred: Other  Van Gogh
    True Other    530       9
    True VanGogh   26      80
```

AlexNet:

```
            Pred: Other  Van Gogh
    True Other    523      16
    True VanGogh  31      75
```



Confusion matrix heatmaps



ROC curves

**Example analysis:**

- **True Positives:** Van Gogh paintings with distinctive brushwork (e.g. swirls, thick strokes) were correctly classified.
- **True Negatives:** Most Post-Impressionist works by other artists (e.g. Gauguin, Seurat) were correctly rejected.
- **False Positives:** Some non–Van Gogh paintings with similar textures (e.g. expressive brushwork, yellow tones) were misclassified.
- **False Negatives:** Van Gogh works with atypical composition or weaker stylistic cues were sometimes misclassified as other.

**Discussion:** The ROC curves show both models can separate classes well (AUC ~0.97), but the confusion matrices reveal that VGG-19 has fewer errors overall. False positives typically correspond to non–Van Gogh works with “Van Gogh-like” textures (strong brush strokes, similar color palette). False negatives can happen for Van Gogh works with less characteristic texture or different subject matter.

## Part 2 Discussion

### 1. Style Transfer Process

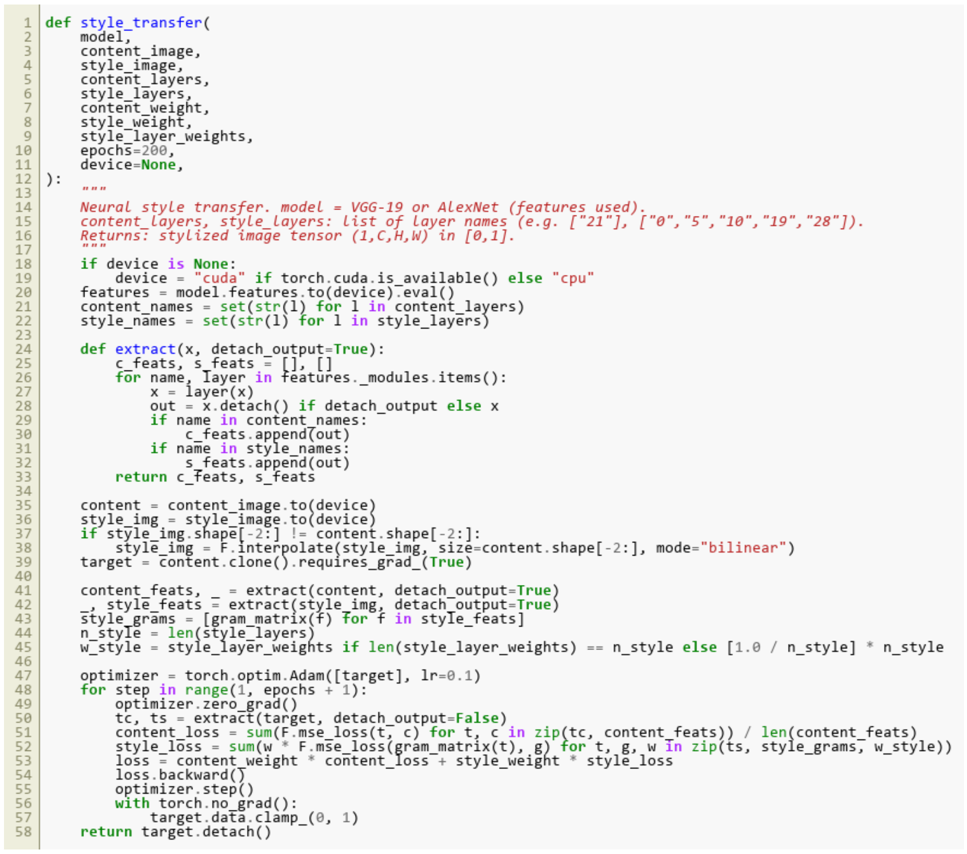
Neural style transfer (Gatys et al.) optimizes a generated image to match:

1. **Content:** feature activations from a content image at chosen layers (e.g. conv4_2)

2. **Style:** Gram matrices of feature maps from multiple style layers, capturing texture and color statistics

The total loss is L = α·L_content + β·L_style. The target image is initialized from the content image and updated via gradient descent.

**Code screenshot (from notebook, required by assignment):**



```python
def style_transfer(
    model,
    content_image,
    style_image,
    content_layers,
    style_layers,
    content_weight,
    style_weight,
    style_layer_weights,
    epochs=200,
    device=None,
):
    """
    Neural style transfer. model = VGG-19 or AlexNet (features used).
    content_layers, style_layers: list of layer names (e.g. ["21"], ["0","5","10","19","28"]).
    Returns: stylized image tensor (1,C,H,W) in [0,1].
    """
    if device is None:
        device = "cuda" if torch.cuda.is_available() else "cpu"
    features = model.features.to(device).eval()
    content_names = set(str(l) for l in content_layers)
    style_names = set(str(l) for l in style_layers)

    def extract(x, detach_output=True):
        c_feats, s_feats = [], []
        for name, layer in features._modules.items():
            x = layer(x)
            out = x.detach() if detach_output else x
            if name in content_names:
                c_feats.append(out)
            if name in style_names:
                s_feats.append(out)
        return c_feats, s_feats

    content = content_image.to(device)
    style_img = style_image.to(device)
    if style_img.shape[-2:] != content.shape[-2:]:
        style_img = F.interpolate(style_img, size=content.shape[-2:], mode="bilinear")
    target = content.clone().requires_grad_(True)

    content_feats, _ = extract(content, detach_output=True)
    _, style_feats = extract(style_img, detach_output=True)
    style_grams = [gram_matrix(f) for f in style_feats]
    n_style = len(style_layers)
    w_style = style_layer_weights if len(style_layer_weights) == n_style else [1.0 / n_style] * n_style

    optimizer = torch.optim.Adam([target], lr=0.1)
    for step in range(1, epochs + 1):
        optimizer.zero_grad()
        tc, ts = extract(target, detach_output=False)
        content_loss = sum(F.mse_loss(t, c) for t, c in zip(tc, content_feats)) / len(content_feats)
        style_loss = sum(w * F.mse_loss(gram_matrix(t), g) for t, g, w in zip(ts, style_grams, w_style))
        loss = content_weight * content_loss + style_weight * style_loss
        loss.backward()
        optimizer.step()
        with torch.no_grad():
            target.data.clamp_(0, 1)
    return target.detach()
```

style_transfer implementation

### 2. Style Loss Normalization

It is important to balance the loss from each style layer because deeper layers have more channels and produce larger Gram matrix values, which would otherwise dominate the total style loss. We use `style_layer_weights` to weight each layer equally (1/5 for 5 layers), so lowlevel and high-level textures contribute similarly. This avoids overemphasizing coarse textures at the expense of fine brushstrokes.

## 3. Hyperparameter Search (Part 2B)

**Setup:**

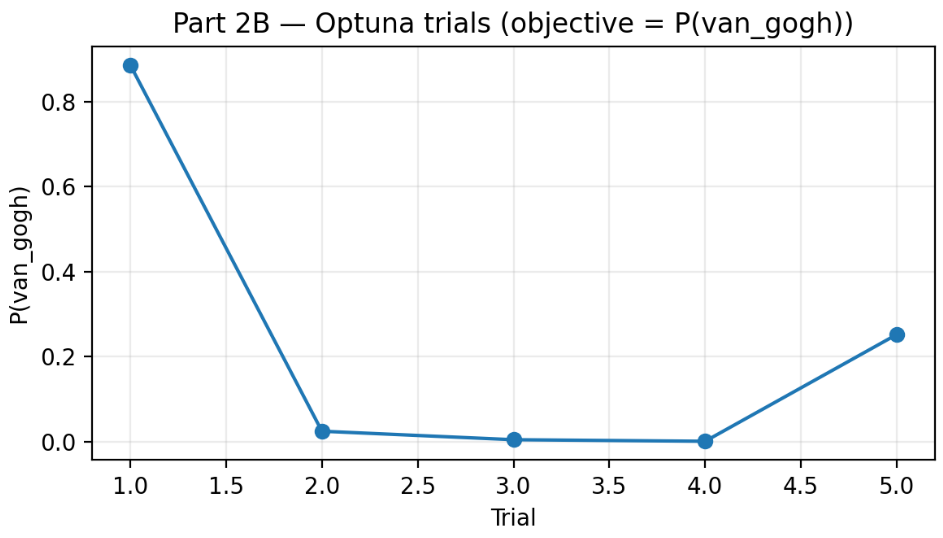
- Classifier used as judge: VGG-19
- Content image: photo.png (personal photo)
- Style image: Van Gogh painting from the dataset

**Objective:** Maximize P(van_gogh) from the Part 1 classifier on the stylized output.

**Optimal hyperparameters found:**

- content_weight (α): 0.0890
- style_weight (β): 8.79e6

**Best score:** Best \(P(\mathrm{van_gogh}) = 0.886\) on the fixed (content, style) pair (5 trials, 50 epochs per trial).



Part 2B Optuna scores

**Process:** We used Optuna to maximize the Part 1 classifier's on the stylized image. For each trial, Optuna sampled `content_weight` (α) and `style_weight` (β), ran style transfer with a fixed content/style pair and a VGG-19 feature extractor, and scored the output using the fine-tuned VGG-19 classifier from Part 1. We used fewer optimization steps per trial (50) to keep the search tractable, then used a larger number of steps (200) for Part 2C.

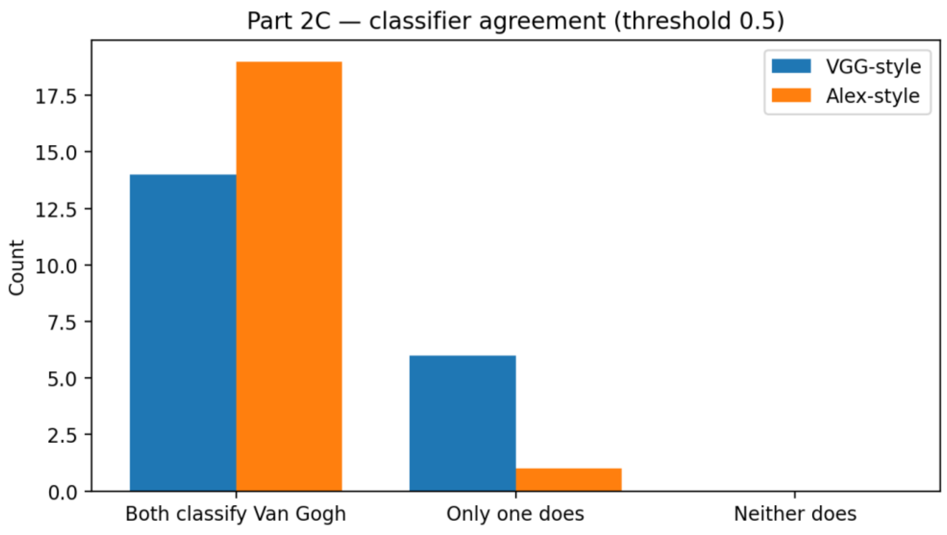
## 4. Classifier Evaluation (Part 2C)

We applied style transfer to 20 content images using 5 different Van Gogh paintings as style images. Both VGG-19 and AlexNet were used for style transfer, and both classifiers evaluated the results.

**Number of epochs per style transfer:** 200 (same for all images and models).

**Results summary:** Each content image produced two stylized outputs (VGG-style and Alex-style). Using threshold 0.5:

- **VGG-style outputs (20 images)** : both models classify as Van Gogh = 14/20, only one model = 6/20, neither = 0/20.
- **Alex-style outputs (20 images)** : both models classify as Van Gogh = 19/20, only one model = 1/20, neither = 0/20.



Part 2C agreement summary

**Examples:**

**Additional examples (different style images):**

VGG-style transfer tends to yield higher P(van_gogh) from both classifiers; AlexNet-style transfer can produce more varied results. The classifiers sometimes disagree on the same stylized image (e.g. VGG 0.99 vs AlexNet 0.06), suggesting different sensitivity to style features.

**Discussion:** Overall, the classifiers usually agree that the stylized images are Van Gogh-like (especially for Alex-style outputs). The "only one model" cases mostly happen when **AlexNet classifier** assigns a low probability while VGG-19 remains high, suggesting AlexNet is less stable on generated images (distribution shift vs real paintings).

### 5. Layer Selection

**Content layers:** VGG: [21] (conv4_2). AlexNet: [8] (conv3).

**Style layers:** VGG: [0, 5, 10, 19, 28] (conv1_1 through conv5_4). AlexNet: [0, 3, 6, 8, 10].

**Justification:** A single mid-level content layer preserves semantic structure without overfitting to pixel details. Multiple style layers at different depths capture textures from fine brushstrokes (early layers) to coarser patterns (deeper layers), giving a multi-scale style representation.

### 6. Overall Reflection

Both classifiers often assign high P(van_gogh) to stylized images, showing they recognize the transferred style as Van Gogh-like. VGG-style transfer tends to get higher scores, consistent with VGG-19's richer feature extraction. Disagreements (e.g. VGG high, AlexNet low) suggest that the two classifiers rely on different visual cues, and that stylized images may not fully match the distribution of real Van Gogh paintings.

## Part 3 Discussion

### CPU vs GPU Benchmark

**Environment (from notebook output):**

```
cuda available: True
GPU Device: NVIDIA GeForce RTX 4080
Machine name: Mikhail_PC
Initial GPU memory: 5329.1 MB allocated, 10584.0 MB reserved
```

**Benchmark:**

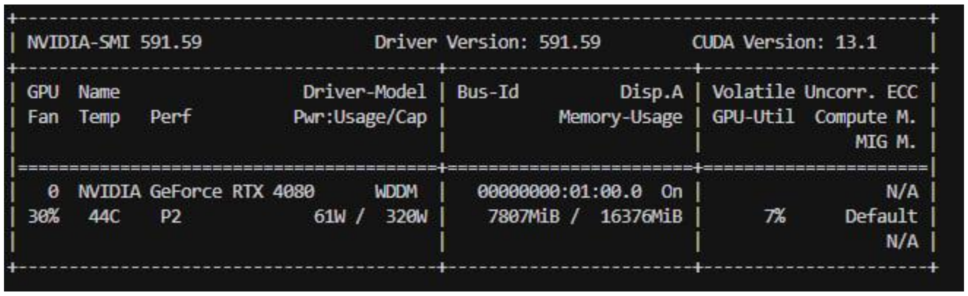
- Option chosen: 20 forward+backward iterations (mini-batches)
- Time on CPU: 106.38 s
- Time on GPU: 15.72 s
- Speedup: 6.8×

**Discussion:**

- The ~7× speedup is consistent with typical GPU acceleration for CNN training. RTX 4080's parallel compute strongly reduces iteration time.
- Potential bottlenecks: data loading with `num_workers=0` on Windows, I/O when reading images. Larger batch sizes could further improve GPU utilization.

**Reflection:** The speedup is significant but not “infinite” because training time also includes overheads: Python dispatch, memory transfers, and data loading. With larger batches and more dataloader workers (when stable), GPU utilization would typically increase further.

## nvidia-smi Screenshot



```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 591.59                 Driver Version: 591.59         CUDA Version: 13.1     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4080      WDDM  |   00000000:01:00.0  On |                  N/A |
| 30%   44C    P2             61W /  320W |    7807MiB /  16376MiB |      7%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

**nvidia-smi output during training**

**Caption:** GPU utilization (7%) and memory usage (7807 MiB / 16376 MiB) on NVIDIA GeForce RTX 4080 during model training. Driver 591.59, CUDA 13.1.

## Bonus: Grad-CAM (Model Interpretability)

*Optional – 10 pts.*

### Part A: Grad-CAM Function

**Implementation:** We implemented Grad-CAM using PyTorch forward and backward hooks on the VGG-19 classifier. The function takes:

- **Inputs:** classifier (VGG-19), image tensor (normalized for ImageNet), target class index (1 = van_gogh), target layer (default: `model.features[35]` — last ReLU after conv5_4).

**Process:**

1. Register a forward hook on the target layer to capture feature maps (activations).

2. Register a backward hook to capture gradients flowing back from the target class score.

3. Perform forward pass, then backward from the target class logit.

4. Compute weights = global average of gradients over spatial dimensions.

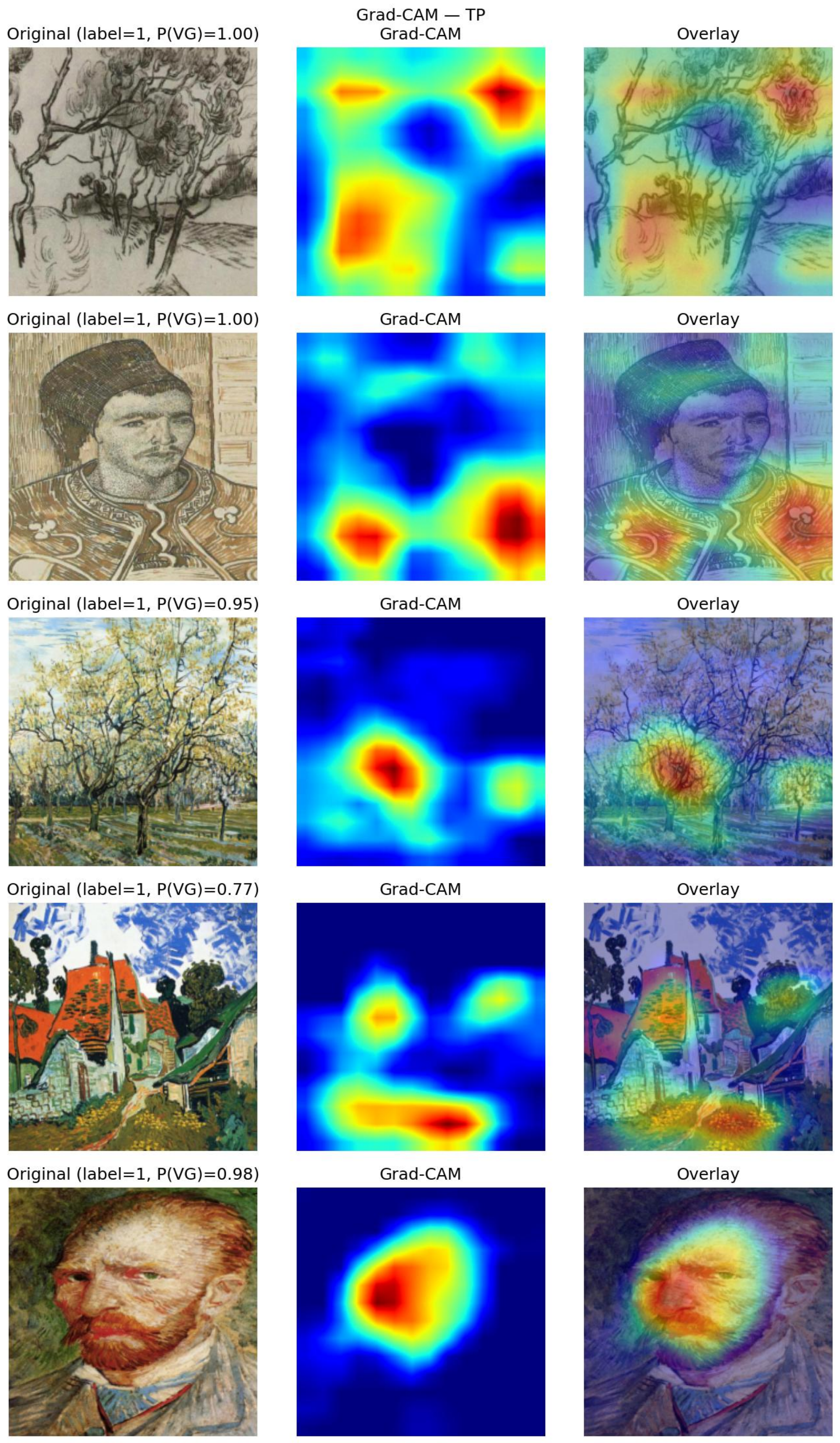
5. CAM = ReLU(Σ weights_k × activations_k) — weighted sum of feature maps.

6. Upsample CAM to input size, normalize to [0,1], overlay on image with jet colormap.

**Output format:** For each sample, we visualize three panels: (1) Original image with label and P(van_gogh); (2) Raw Grad-CAM heatmap (jet colormap, red = high activation); (3) Overlay of heatmap on the image (alpha=0.4) to show which regions the model attends to.

## Part B: Analysis

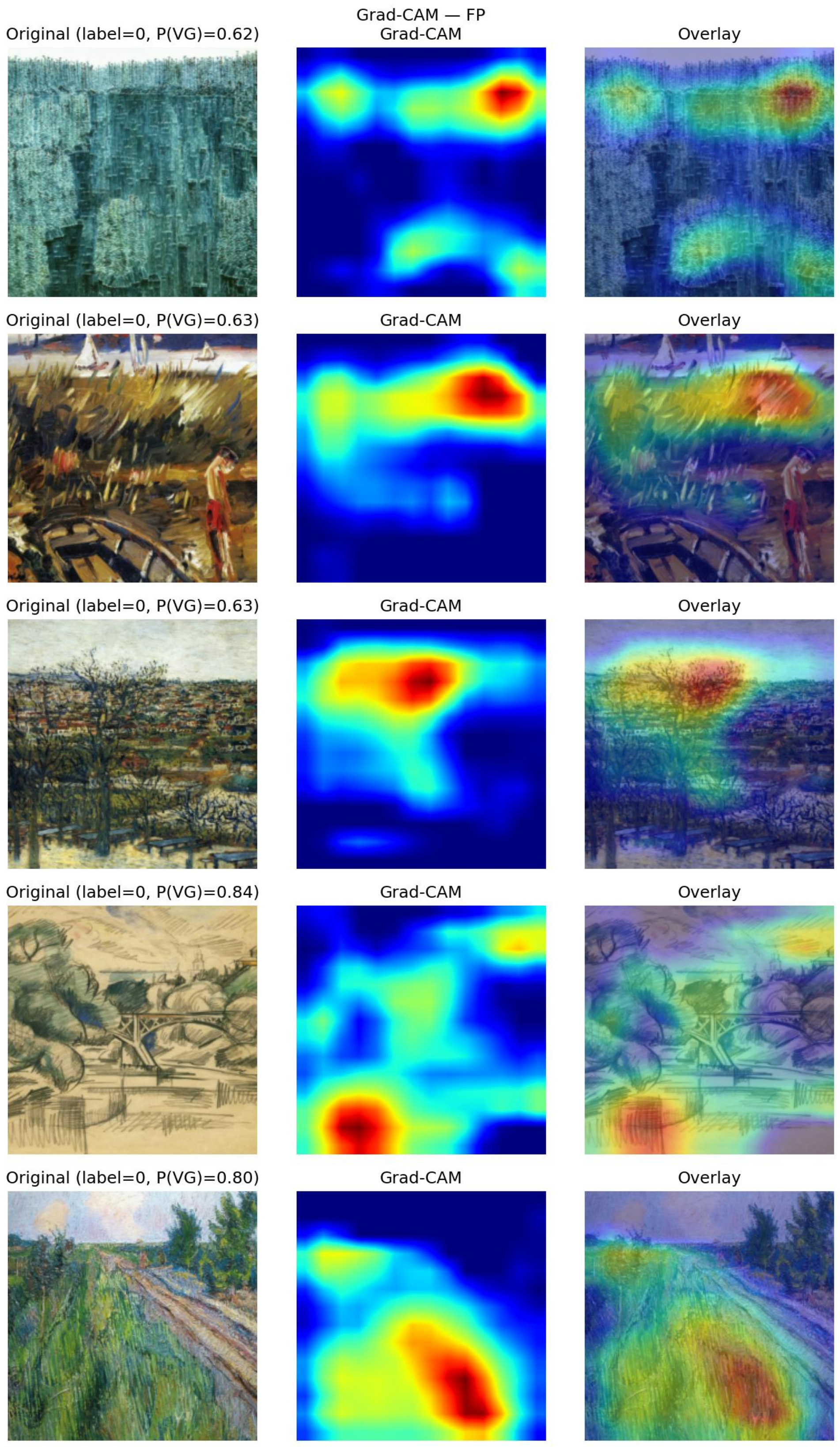
**True Positives (Van Gogh correctly classified):** The heatmaps highlight brushstrokes, swirls, and textured regions (sky, foliage, wheat fields) — the visual “signature” the model learned for Van Gogh. The classifier attends to characteristic impasto and expressive strokes rather than the full canvas uniformly. Red/yellow regions in the jet colormap indicate strong activation; the overlay clearly shows attention on textured brushwork.


Grad-CAM — TP
Original (label=1, P(VG)=1.00)
Grad-CAM
Overlay
Original (label=1, P(VG)=1.00)
Grad-CAM
Overlay
Original (label=1, P(VG)=0.95)
Grad-CAM
Overlay
Original (label=1, P(VG)=0.77)
Grad-CAM
Overlay
Original (label=1, P(VG)=0.98)
Grad-CAM
Overlay

**Grad-CAM True Positives**

*Figure: Grad-CAM heatmaps for True Positives. Each row: Original | Raw heatmap (jet) | Overlay. The model focuses on brushstroke texture and expressive regions.*

**False Positives (Other paintings misclassified as Van Gogh):** The heatmaps often show regions with similar textures — strong brushwork, bold color blocks, or swirling patterns that resemble Van Gogh's style. The model is "fooled" by paintings (e.g. Gauguin, other Post-Impressionists) that share expressive, textured brushwork.


Grad-CAM — FP
Original (label=0, P(VG)=0.62)
Grad-CAM
Overlay
Original (label=0, P(VG)=0.63)
Grad-CAM
Overlay
Original (label=0, P(VG)=0.63)
Grad-CAM
Overlay
Original (label=0, P(VG)=0.84)
Grad-CAM
Overlay
Original (label=0, P(VG)=0.80)
Grad-CAM
Overlay

**Grad-CAM False Positives**

*Figure: Grad-CAM heatmaps for False Positives. Same layout: Original | Heatmap | Overlay. Activation patterns are similar to True Positives, explaining the misclassification.*

**Conclusion:** The classifier learned a visual signature centered on **brushstroke texture and expressive color application** . It responds strongly to thick, directional strokes and vibrant patches. Grad-CAM confirms that the model's decisions are driven by texture and color statistics rather than high-level semantic content. This explains both correct Van Gogh identifications (where those cues are present) and false positives (where other artists use similar techniques).

### Part C: Improving Style Transfer (Conceptual)

**Proposed modification:** Use the Grad-CAM heatmap as a spatial weight mask. In content-critical regions (e.g. faces, high activation in the heatmap), reduce the style_loss weight so the style does not distort important content. In background regions, keep or increase style_loss weight. **Benefit:** Preserve faces and key objects while still applying strong Van Gogh style to the rest of the image.

## Appendices

### Appendix A: Metrics Table

| Model | Accuracy | AUC-ROC | F1-Score |
|---|---|---|---|
| --------- | ---------- | --------- | ---------- |
| VGG-19 | 0.946 | 0.976 | 0.821 |
| AlexNet | 0.927 | 0.971 | 0.761 |

### Appendix B: Confusion Matrices

VGG-19: [[530, 9], [26, 80]]. AlexNet: [[523, 16], [31, 75]]. (rows = true, cols = predicted; 0 = other, 1 = van_gogh)

### Appendix C: Stylized Images

See `outputs/result_vincent-van-gogh_*.png` for the 20 content images × 2 models (VGG-19 and AlexNet style transfer).

## Non-Standard Python Packages

| Package | Purpose |
|---|---|
| ------------------------- | ---------------------------------------------- |
| torch, torchvision | Models, training, transforms |
| optuna | Hyperparameter search (Part 1 & 2B) |

| | |
|---|---|
| optuna-integration[wandb] | W&B callback for Optuna |
| wandb | Experiment tracking, logging |
| scikit-learn | Metrics (AUC-ROC, F1, confusion_matrix) |
| seaborn | Confusion matrix heatmaps, plots |
| pandas | Data loading (classes.csv) |
| Pillow | Image I/O |
| matplotlib | Visualization |