# Part 1: Van Gogh Classification

---

## 1. Dataset Description

We worked with a **Post-Impressionism subset** of the WikiArt dataset. Each image is labeled by the artist and stored alongside metadata in a CSV file. The classification target was binary:

- `1` if painted by **Vincent van Gogh**
- `0` otherwise

**Class Distribution (Train):**
Van Gogh: 478
Not Van Gogh: 1522

**Class Distribution (Test):**
Van Gogh: 197
Not Van Gogh: 1046

This class imbalance highlights the importance of precision and recall-focused metrics like F1 Score and AUC-ROC.

---

## 2. Models and Architectures

We fine-tuned **two pretrained models**:

| Model | Publication | Original Dataset | Key Features |
|---|---|---|---|
| **AlexNet** | 2012 | ImageNet | Simpler, fewer parameters, 5 conv + 3 FC |
| **VGG19** | 2014 | ImageNet | Deeper, 19 layers, very strong at feature extraction |

Both models had their classifier head replaced with a single-node linear layer (`nn.Linear(num_features, 1)`) for binary classification.

---

## 3. Training Approach

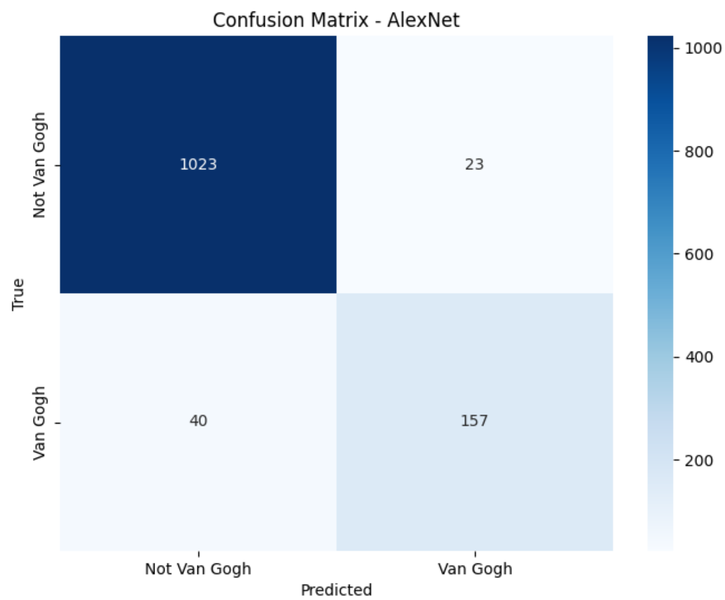
- **Data Augmentation**:
  Applied RandomCrop, Horizontal/Vertical Flip, Rotation, ColorJitter.

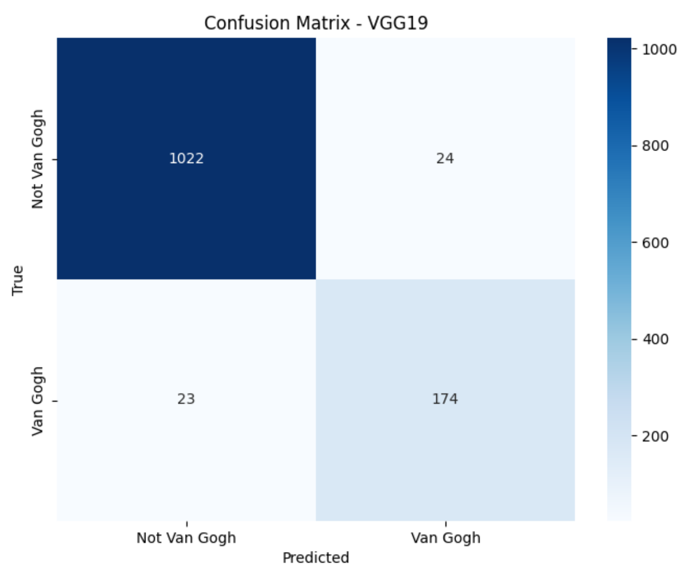
- **Optimization**:
  Optimizer: `Adam`
  Scheduler: `ReduceLROnPlateau`
  Loss: `BCEWithLogitsLoss`

- **Cross-Validation**:
  5-Fold Stratified K-Fold used to improve generalization.

- **Hyperparameter Tuning**:
  Used **Optuna** to tune:
  - Batch size
  - Learning rate
  - Weight decay
- **Experiment Tracking**:
  All key metrics (Accuracy, Loss, F1, AUC) were plotted and logged.

## 4. Evaluation Metrics and Comparison



- VGG19 significantly reduced false negatives, which is important for identifying Van Gogh paintings.

| Metric | **AlexNet** | **VGG19** |
|---|---|---|
| Best Accuracy | ~94.5% | ~97.1% |
| Best F1 Score | ~0.83 | ~0.91 |
| Best AUC-ROC | **0.9704** | **0.9833** |

**VGG19 consistently outperformed AlexNet across all metrics.**

---

## 5. Metrices Analysis

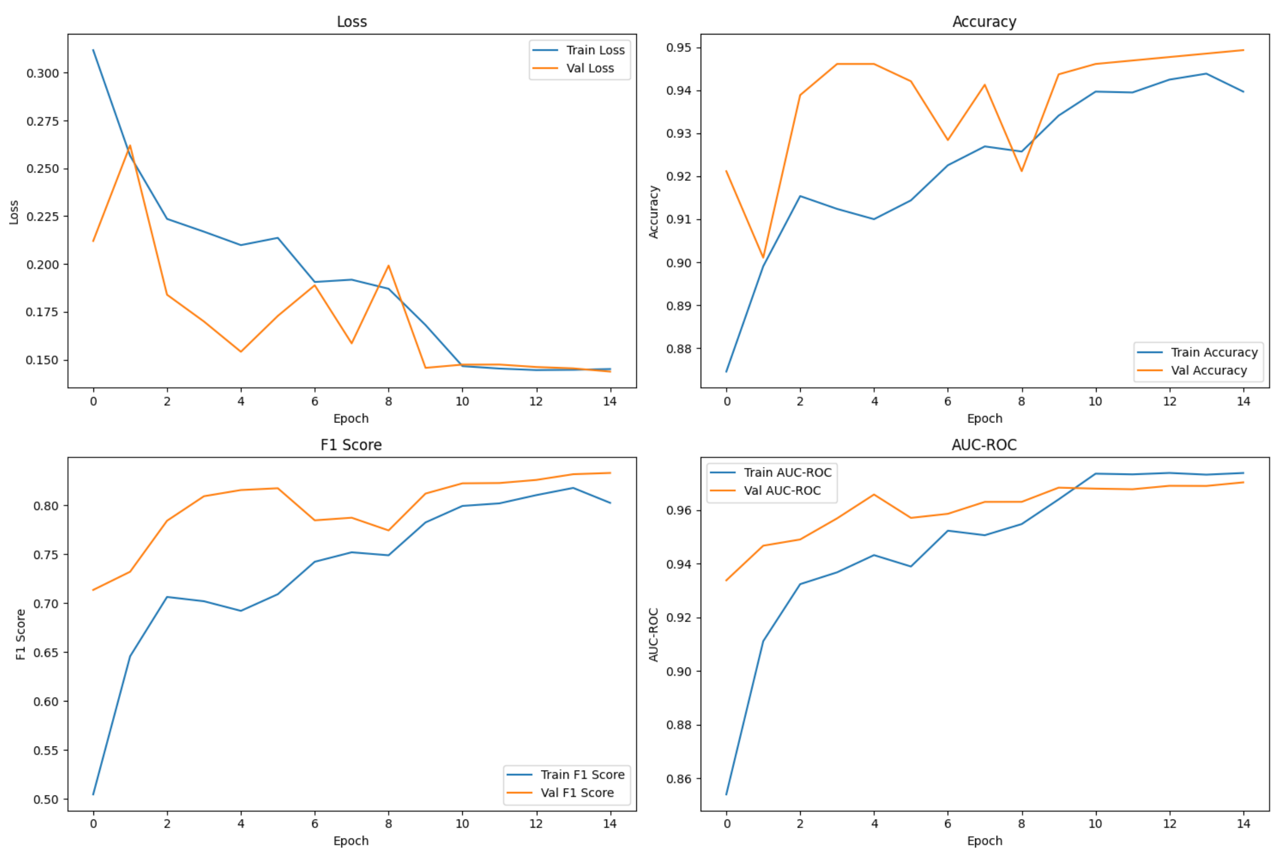
### *AlexNet*

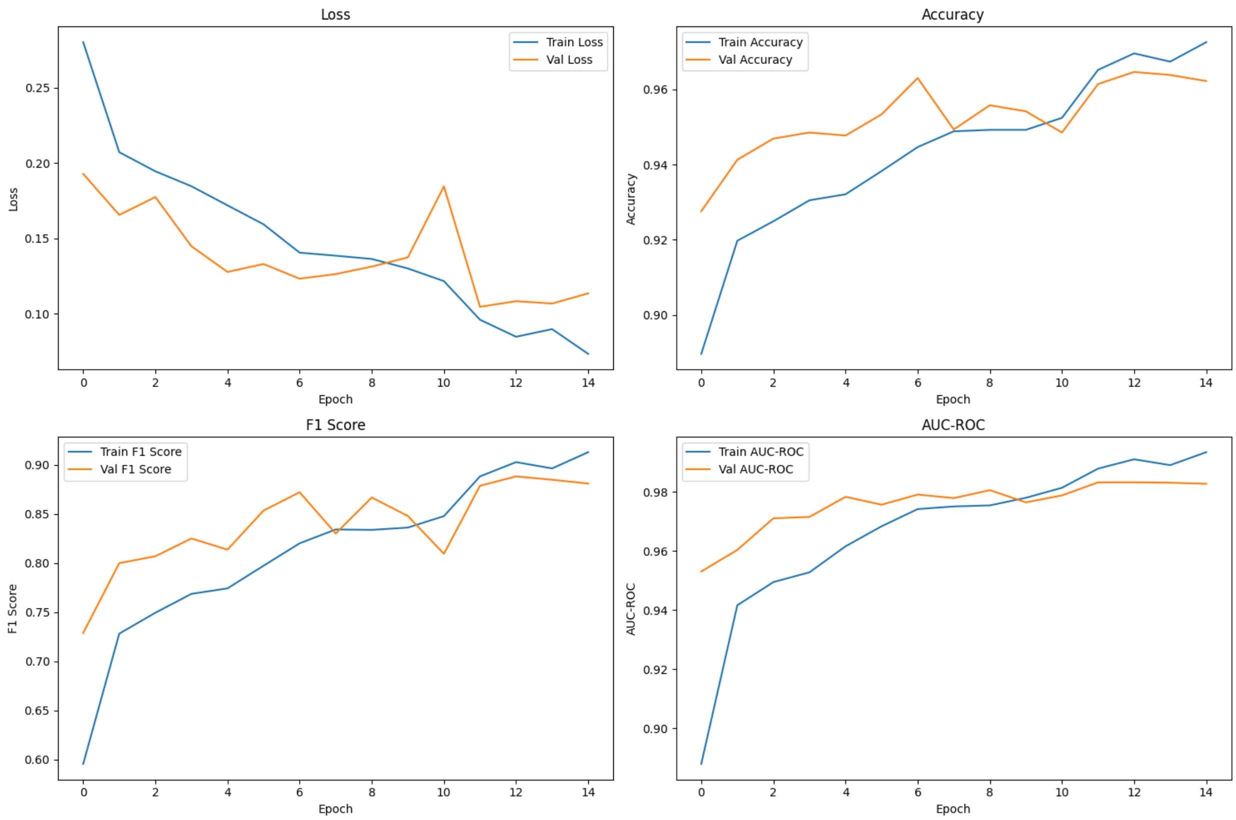
- **False Positives**: 23
- **False Negatives**: 40

### *VGG19*

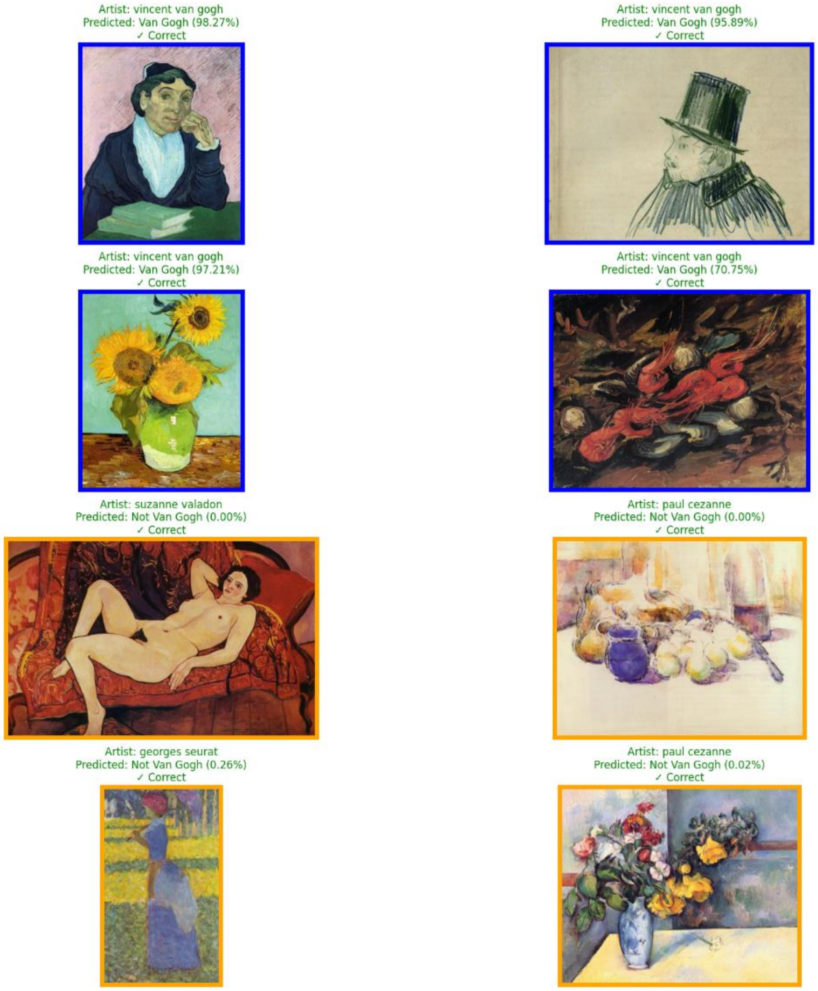
- **False Positives**: 24
- **False Negatives**: 23

**Observation**: VGG19 is more balanced, while AlexNet misses more true Van Gogh paintings (higher FN).






# 6. Model Predictions




---

## 🖌️ Part 2: Style Transfer

---

### 1. Style Transfer Function

- Built a **generic PyTorch function** compatible with both **VGG19** and **AlexNet**.
- Combines:
  - **Content Loss** from high-level layers (e.g. `conv_5`)
  - **Style Loss** using **Gram matrices** from multiple low- and mid-level layers.
- Style weights: `[1.0, 0.8, 0.6, 0.4, 0.2]`
- Style intensity (`style_weight`) = `1e6`
- Content intensity (`content_weight`) = `1`

## 2. Input Details

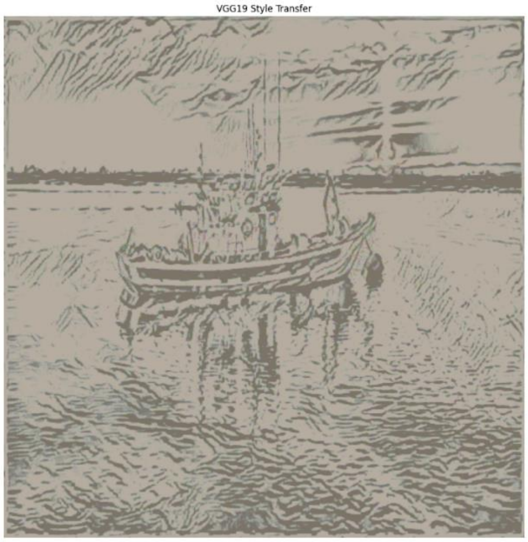
- **Content Images**: 1 personal photo
- **Style Images**: 5 Van Gogh paintings
- **Epochs**: 300 (fixed for all)
- **Output**: Stylized images from both VGG19 and AlexNet

## 3. Generated Paintings

Base Painting:

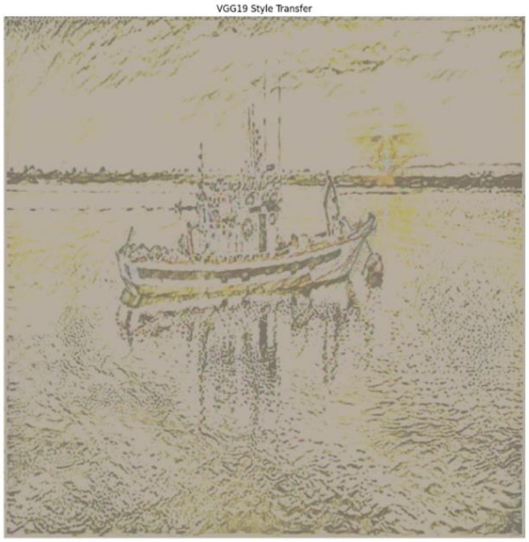
Style Transferred Paintings:

VGG19 Style Transfer



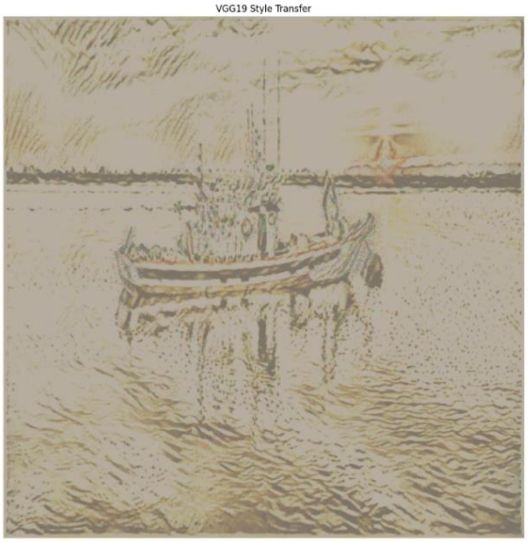
AlexNet Style Transfer

VGG19 Style Transfer

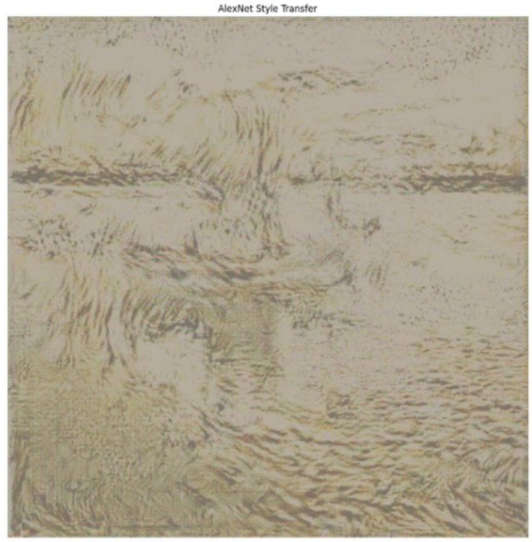


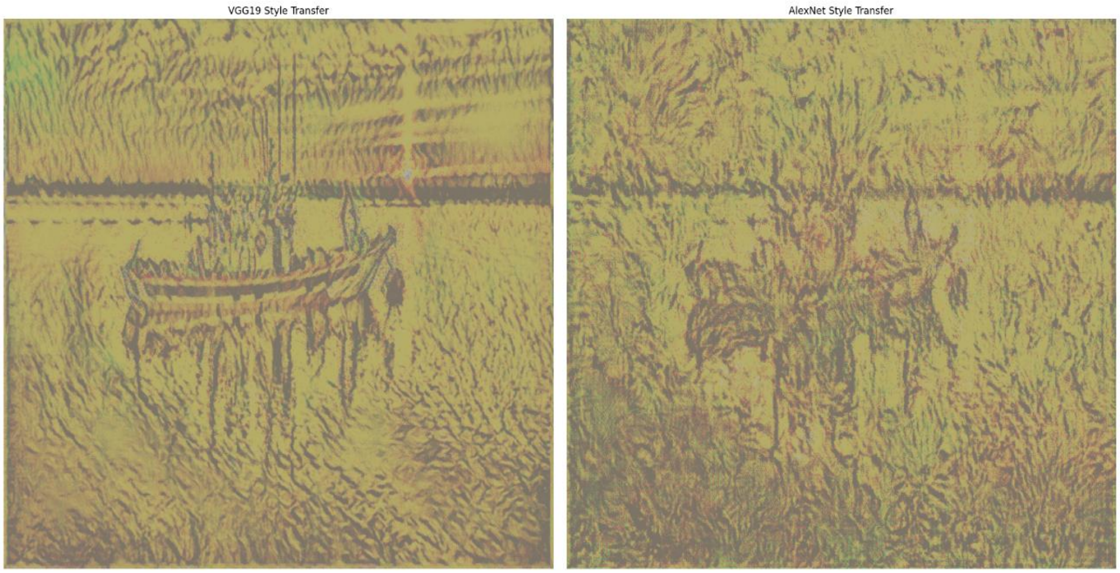
AlexNet Style Transfer

VGG19 Style Transfer



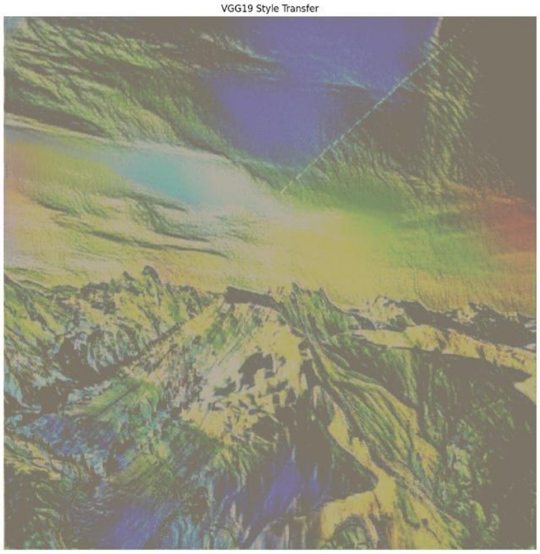
AlexNet Style Transfer



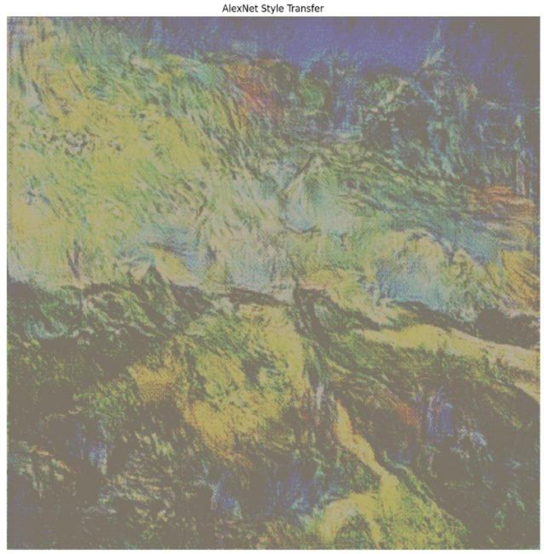



Another Example:

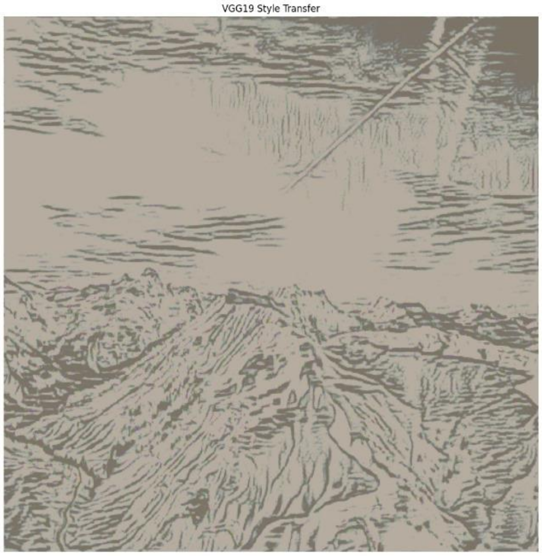
VGG19 Style Transfer



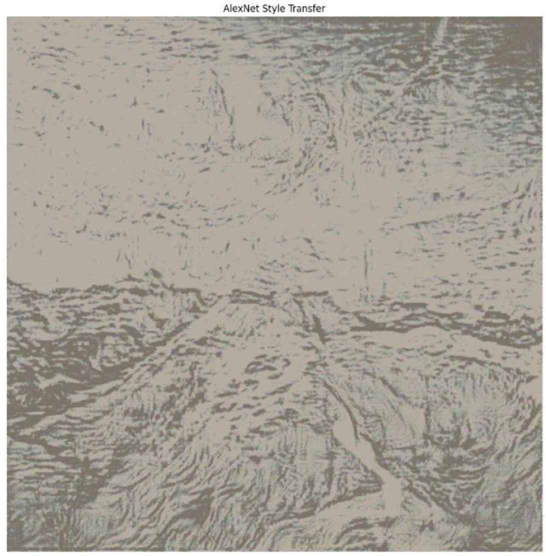
AlexNet Style Transfer



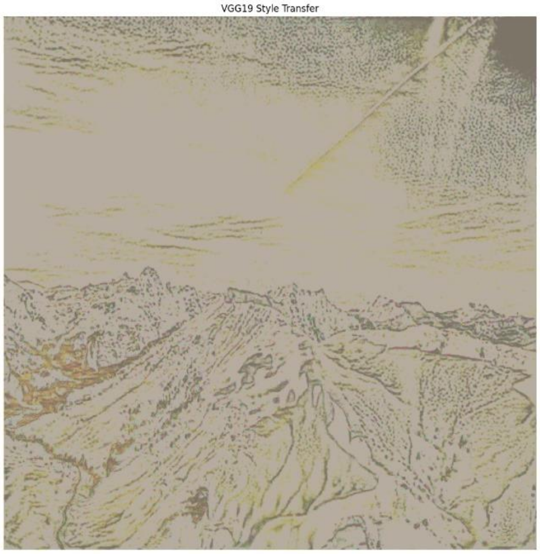
VGG19 Style Transfer



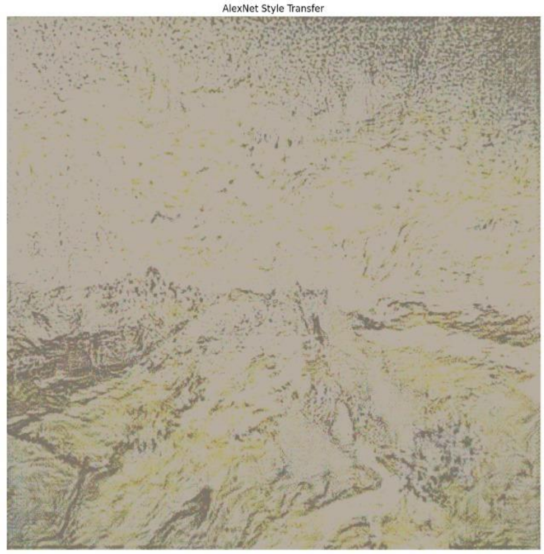
AlexNet Style Transfer

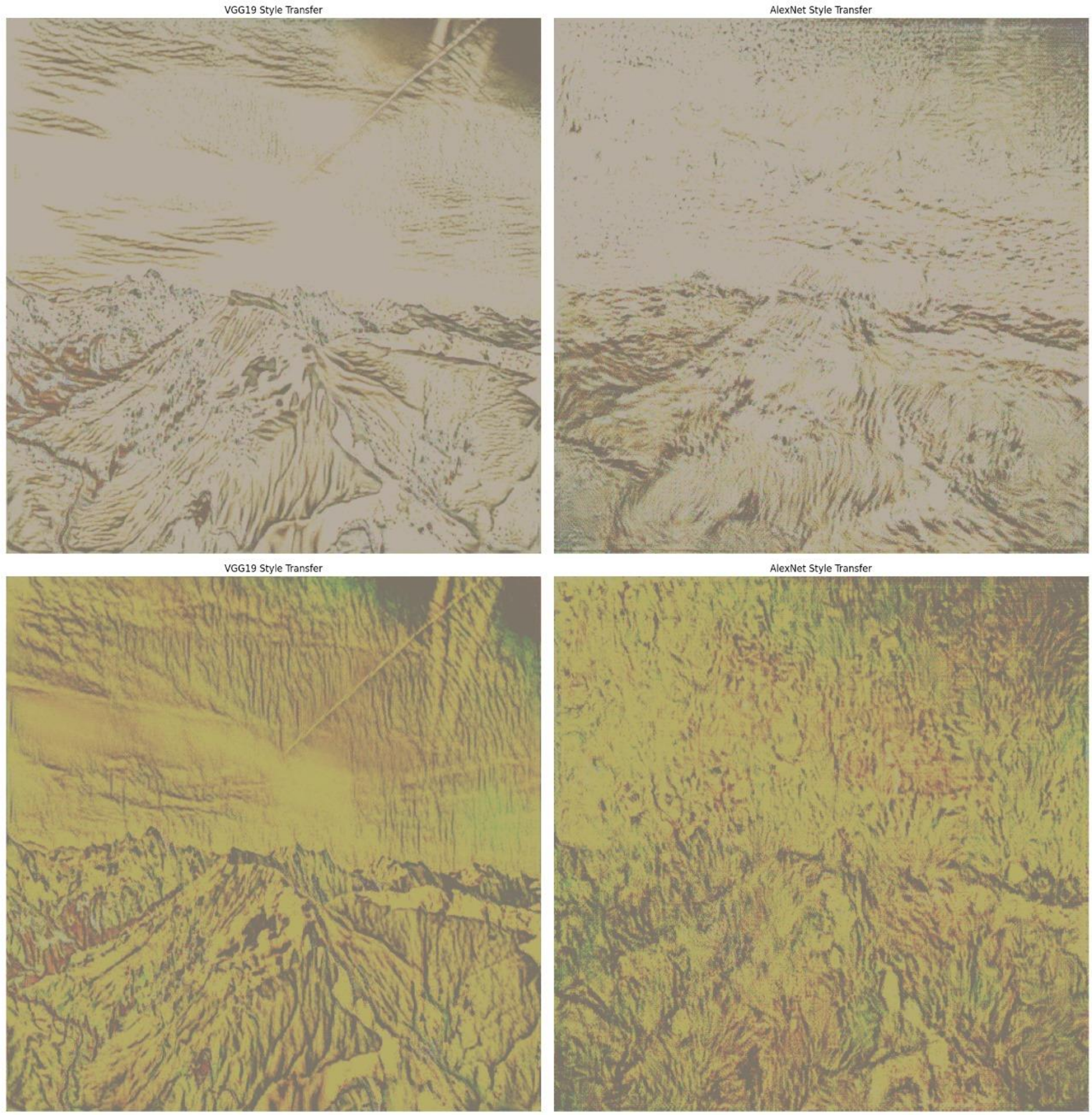


VGG19 Style Transfer



AlexNet Style Transfer






## 4. Classification-Based Evaluation

**Classification Results (% of images predicted as Van Gogh):**

| Generated By | Classified by VGG19 | Classified by AlexNet |
|---|---|---|
| **VGG19** | 60% | 80% |
| **AlexNet** | 80% | 80% |

## 5. Interpretation

- **VGG19-generated outputs** were more artistically stylized but sometimes less recognizable to the classifier as Van Gogh.
- **AlexNet-generated outputs** seemed more “conservative”, retaining more recognizable texture/patterns for the classifiers.
- **Classifier agreement** was higher on AlexNet-style results.

---

## 5. Layer Choices

| Model | Content Layer | Style Layers |
|---|---|---|
| VGG19 | `conv_5` | `conv_1` to `conv_5` |
| AlexNet | `conv_5` | `conv_1` to `conv_5` |

**Justification**:

- Shallow layers capture texture (style)
- Deeper layers preserve object layout (content)

---

## 6. Validation Through Classification

The classification models partially aligned with human perception:

- When both classifiers agree on Van Gogh style, the result is clearly artistic.
- Misalignments reveal that classifier "style recognition" doesn’t always align with human aesthetic judgment.

## Code Explanation:

### Part 01:

### 1. Imports and Setup

- **Libraries**: Uses PyTorch, torchvision, pandas, scikit-learn, Optuna (for hyperparameter tuning), and matplotlib/seaborn for visualization.
- **Paths**: Defines dataset paths (`PAINTINGS_PATH` for images, `CSV_PATH` for labels).
- **Device**: Uses GPU if available.
- **Reproducibility**: Sets random seeds for PyTorch, NumPy, and CUDA.

### 2. Custom Dataset (`WikiArtDataset`)

- **Purpose**: Loads images and labels from a CSV file, filters by subset (train/test), and checks image validity.
- **Key Features**:
    - **Label Creation**: Marks images as Van Gogh (1) if "van gogh" is in the artist's name (case-insensitive).
    - **Validation**: Filters out invalid image paths to avoid runtime errors.
    - **Transforms**: Applies data augmentation (training) or standard preprocessing (validation).

### 3. Data Augmentation/Preprocessing (`get_transforms`)

- **Training Transforms**: Random resizing, flipping, rotation, and color jitter to prevent overfitting.
- **Validation Transforms**: Fixed resizing and center cropping for consistent evaluation.
- **Normalization**: Uses ImageNet mean/std for compatibility with pre-trained models.

### 4. Model Architecture (`FineTunedModel`)

- **Base Models**: Supports VGG19 or AlexNet from `torchvision.models`.
- **Transfer Learning**: Replaces the final classification layer with a single-output neuron for binary classification (Van Gogh vs. not).
- **Output**: Raw logits (passed through sigmoid during evaluation for probabilities).

### 5. Training Loop (`train_model`)

- **Training Phase**: Computes loss, updates weights via backpropagation, and tracks metrics (loss, accuracy, F1, AUC).
- **Validation Phase**: Evaluates model performance on a held-out set.
- **Early Stopping**: Uses `ReduceLROnPlateau` to adjust learning rates if validation loss stagnates.
- **Checkpointing**: Saves the model with the best validation AUC.

### 6. K-Fold Cross-Validation (`perform_kfold`)

- **Stratified Splits**: Maintains class balance across folds using `StratifiedKFold`.

- **Training per Fold**: Trains the model on each fold and records validation AUC.
- **Result Aggregation**: Reports average AUC across all folds.

### 7. Hyperparameter Tuning (`objective`)

- **Optuna Integration**: Optimizes hyperparameters like:
  - Batch size (`8`, `16`, `32`)
  - Learning rate (log-scale between `1e-5` and `1e-3`)
  - Weight decay (L2 regularization).
- **Short Training**: Runs 5 epochs per trial to balance speed and performance.

### 8. Visualization Tools

- **Metric Plots**: Generates loss, accuracy, F1, and AUC curves (`plot_metrics`).
- **Confusion Matrix**: Visualizes true vs. predicted labels for the test set (`plot_confusion_matrix`).

### 9. Workflow (`main`)

1. **Data Loading**: Validates image paths and checks class distribution.
2. **Hyperparameter Tuning**: Runs Optuna trials for VGG19 and AlexNet.
3. **Cross-Validation**: Evaluates model stability across 5 folds.
4. **Final Training**: Trains models with optimal hyperparameters for 15 epochs.
5. **Evaluation**: Plots metrics and confusion matrices for the test set.

## Part 02:

**Key Components**

1. **Image Loading/Preprocessing**:
   - load_image: Resizes and normalizes images using ImageNet statistics (for compatibility with pre-trained models).
   - im_convert: Converts tensors back to numpy arrays for visualization.
2. **Loss Functions**:
   - **Content Loss**: Measures the MSE between intermediate feature maps of the content image and the generated image.
   - **Style Loss**: Uses **Gram matrices** to compare the correlation of feature maps between the style image and generated image. This captures texture/style rather than content.

3. **Model Setup**:
   - get_model_and_losses: Builds a modified VGG19/AlexNet model with hooks to compute content/style losses at specified layers.
   - **Content Layers**: Typically deeper layers (e.g., conv_5) to preserve content.
   - **Style Layers**: Shallow and deep layers (e.g., conv_1 to conv_5) to capture style at multiple scales.
4. **Optimization**:
   - Uses **LBFGS optimizer** to iteratively adjust the generated image to minimize the weighted sum of content and style losses.
   - Parameters like style_weight and content_weight control the trade-off between preserving content vs. style.
5. **Style Transfer Execution**:
   - neural_style_transfer: Runs the optimization loop to generate the stylized image.
   - compare_style_transfer: Compares results from VGG19 and AlexNet side-by-side.
6. **Batch Processing**:
   - batch_style_transfer: Applies style transfer to multiple content/style image pairs and saves results.
7. **Evaluation**:
   - evaluate_style_transfer: Uses the **classification models** from the first script to check if the generated images are classified as "Van Gogh style" (measuring style transfer effectiveness).

**Workflow**

1. **Input Preparation**:
   - Load content (e.g., a photo) and style images (e.g., Van Gogh's painting).
   - Preprocess images into tensors.
2. **Model Configuration**:
   - Choose a backbone (VGG19/AlexNet).
   - Define which layers to use for content/style loss and their weights.
3. **Optimization Loop**:
   - Forward pass: Compute content/style losses.

   - Backward pass: Update the generated image to minimize losses.
   - Repeat for num_epochs iterations.
4. **Result Visualization**:
   - Display/save the stylized image and compare VGG19/AlexNet outputs.