Group 9

Artist Identification & Art Style Transfer

Apitsada (Pearl) Ruknapapong, Daeun Ji, Kavya Bhat, Chi Nguyen, Shubham Kumar

Content

- Problem Objective
- Data Gathering
- Exploratory Data Analysis
- Image Preprocessing
- Models & Results

Group 9

Problem Objective

Utilizing deep learning models to

- Classify artist based on the art images
- Transfer art style from one image to another

VGG16

ResNet50

Artist Identification

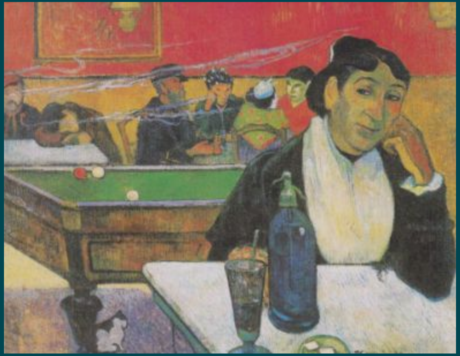
VGG19

InceptionV3

Art Style Transfer

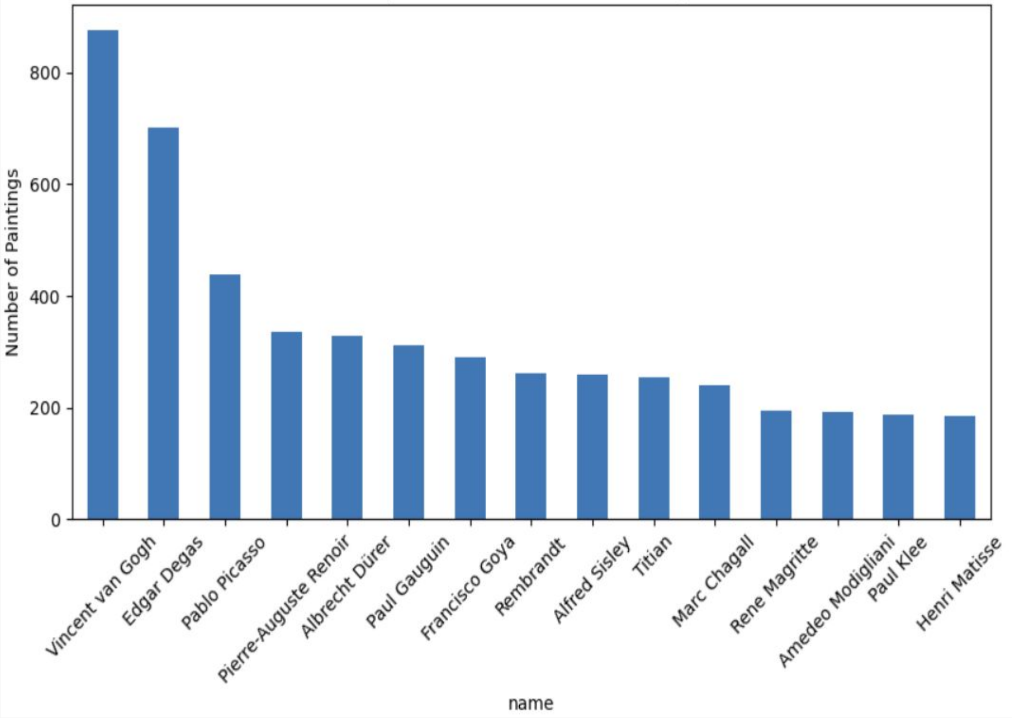
Data Gathering

Kaggle → EDA → Image Preprocessing → Model Training



Data: https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time/data

Exploratory Data Analysis



Dataset Overview:

- Total: 8,683 images
- No Missing Values

Artist Identification:

- Has 50 unique artists
- 11 artists are chosen due to number of their paintings (≥ 200)
- Train datasets: 80%
- Test datasets: 20%

Art Style Transfer:

- No traditional training dataset
- Single image style transfer

Image Preprocessing (Artist)

- Due to imbalanced in dataset, class_weight is important
- Used Keras ImageDataGenerator for data augmentation
- Resized all images to 224×224 pixels
- Normalized pixel values to the [0,1] range

```
Total images found: 8683
First 5 image filenames: ['Gustav_Klimt_113.jpg', 'Vincent_van_Gogh_388.jpg', 'Amedeo_Modigliani_24.jpg', 'Edgar_Degas_455.jpg', 'Edgar_Degas_333.jpg']
```

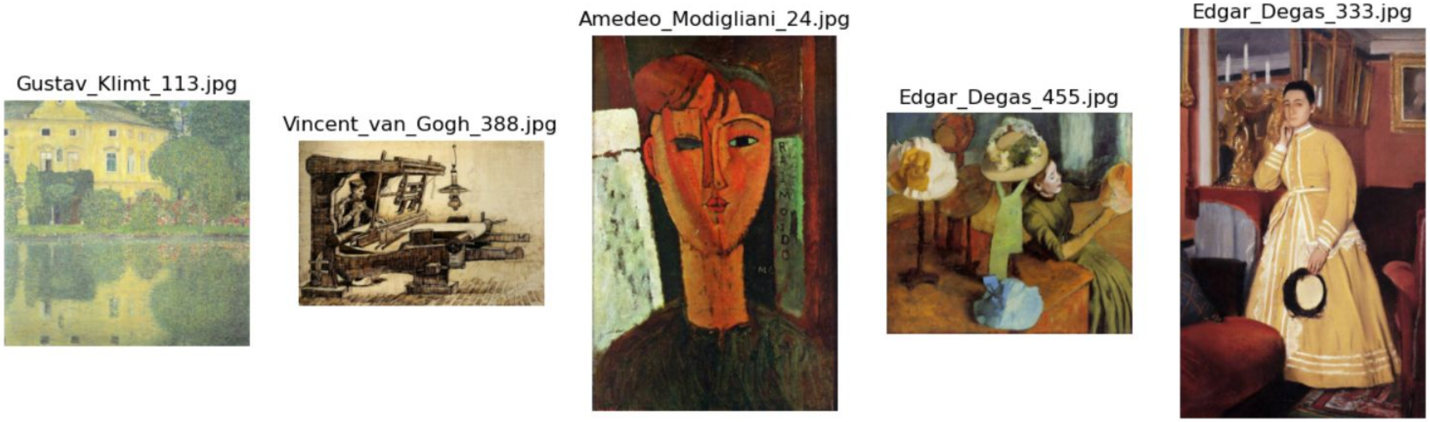


Image Preprocessing (Art Style)

- Resized images based on the aspect ratio of the original image, using the number of rows (img_nrows) and the number of columns (img_ncols).
- Normalized pixel values according to the models:
  - VGG19: normalized by subtracting the mean RGB values. This centers pixel values around 0
    - Red channel: Subtract 103.939
    - Green channel: Subtract 116.779
    - Blue channel: Subtract 123.68
  - InceptionV3: the pixel values are scaled to the range [-1, 1]

Artist Identification – VGG16

Model Architecture:

- Input Layer: 224x224 pixel images
- Convolutional Layers: Pre-tained VGG16 based, ReLU activation
- Output Layers: Number of units equal to the number of artist classes, Softmax activation

Regularization:

- Dropout Rate: 0.5 in the dense layer to reduce overfitting

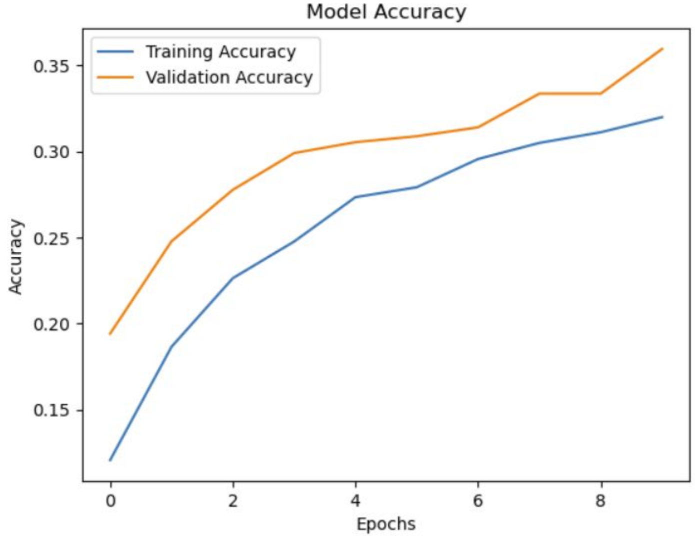
Compilation:

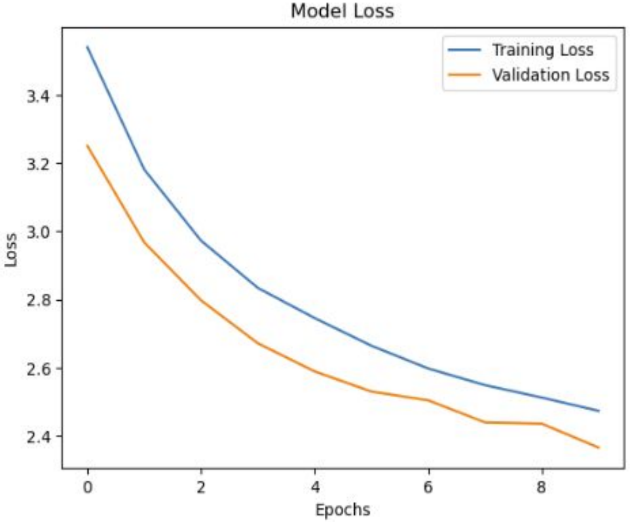
- Optimizer: Adam
- Loss Function: Sparse categorical cross-entropy
- Metrics: Accuracy

Results:

- Training Accuracy: 31%
- Validation Accuracy: 36%

Artist Identification – VGG16





Predicted Artist: Rembrandt
Actual Artist: Peter_Paul_Rubens
Prediction Probability: 33.65%

- Easy to understand and implement due to its straightforward architecture

- Requires significant computational resources due to its large number of parameters
- Risk of overfitting, especially with imbalanced datasets
- Less deep compared to ResNet50, potentially limiting its ability to capture complex features

Artist Identification – ResNet50

Model Architecture:

- Input Layer: 224x224 pixel images
- Convolutional Layers: Pre-tained ResNet50 based, ReLU activation
- Output Layers: Number of units equal to the number of artist classes, Softmax activation

Regularization:

- Dropout Rate: 0.5 in the dense layer to reduce overfitting

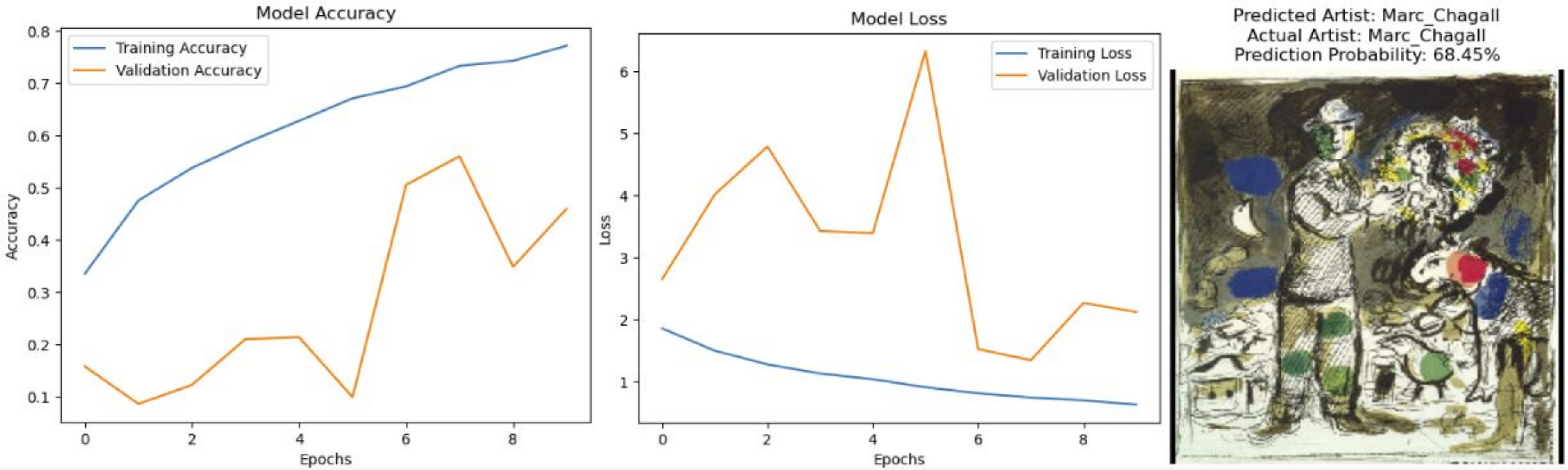
Compilation:

- Optimizer: Adam with a learning rate of 0.0001
- Loss Function: Sparse categorical cross-entropy
- Metrics: Accuracy

Results:

- Training Accuracy: 78%
- Validation Accuracy: 55%

Artist Identification – ResNet50



- Capture more complex features, leading to better performance

- More complex and computationally expensive compared to VGG16
- Longer training times due to its depth and complexity
- Risk of overfitting with imbalanced datasets

Artist Identification – Final Model

We implemented 2 phases of fine-tuning, using ResNet50 since the base model has better performance compared to VGG16:

Phase 1: Training with all layers

- Adding Classification Layers such as Dense, BatchNormalization, and Activation layers on top of the pre-trained ResNet50 model
- Using EarlyStopping and ReduceLROnPlateau callbacks to monitor training and adjust learning rates

Phase 2: Fine-tuning with selective layers freezing

- Freezing the deeper layers of the model to prevent them from being updated
- Only the shallow layers are kept trainable and fine-tuned
- Continue using callbacks to monitor and adjust the training process

Artist Identification – Final Model

Model Architecture:

- Input Layer: 224x224 pixel images
- Convolutional Layers: Pre-tained ResNet50 based, ReLU activation
- Output Layers: Number of units equal to the number of artist classes, Softmax activation

Regularization:

- BatchNormalization: Applied to dense layers to stabilize and accelerate training

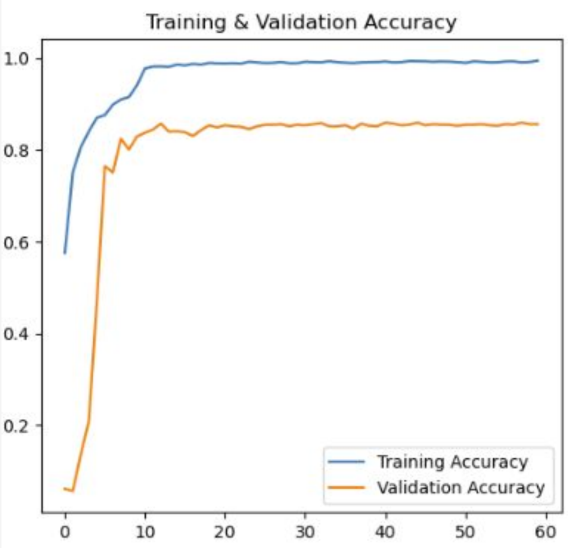
Compilation:

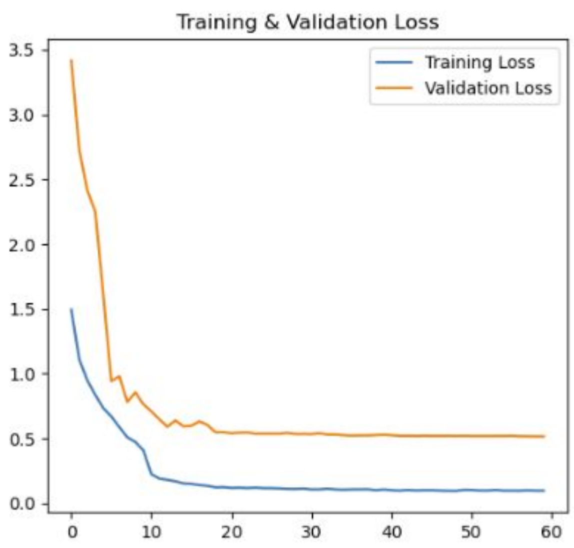
- Optimizer: Adam with a learning rate of 0.0001
- Loss Function: Categorical cross-entropy
- Metrics: Accuracy

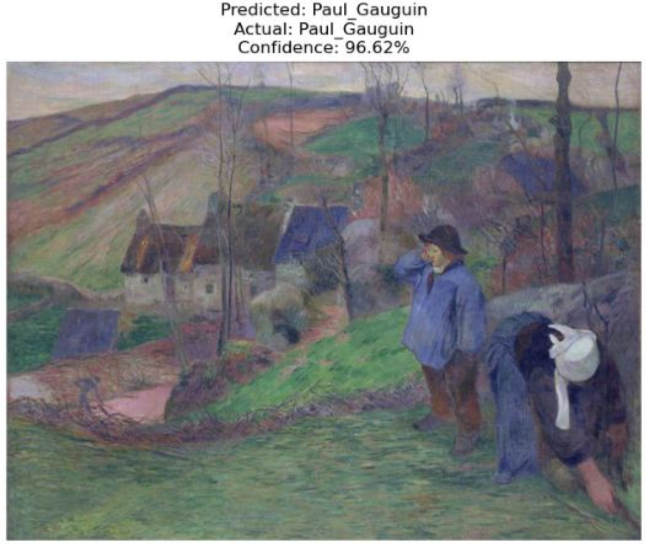
Results:

- Training Accuracy: 99%
- Validation Accuracy: 81%

Artist Identification – Final Model







Art Style Transfer - Models

- Preprocess Images: Resize and normalize the images (as mentioned in data overview)
- Extract Features: Use VGG19/ InceptionV3 to extract content and style features from the images
- Compute Losses: Calculate content and style losses using the extracted features.
    - Gram Matrix: Measures the style of an image.
    - Content Loss: Measures how much the content of the generated image differs from the base image.
    - Style Loss: Measures how much the style of the generated image differs from the style image.
- Optimize Image: Adjust the generated image to minimize the total loss

Art Style Transfer - Models

Content Loss:

- Measure how different the generated image is from the original image by comparing their feature maps at a specific layer using Mean Squared Error (MSE)

Gram Matrix:

- Capture the correlations between features by computing the dot products of the feature vectors in a given layer.

Style Loss:

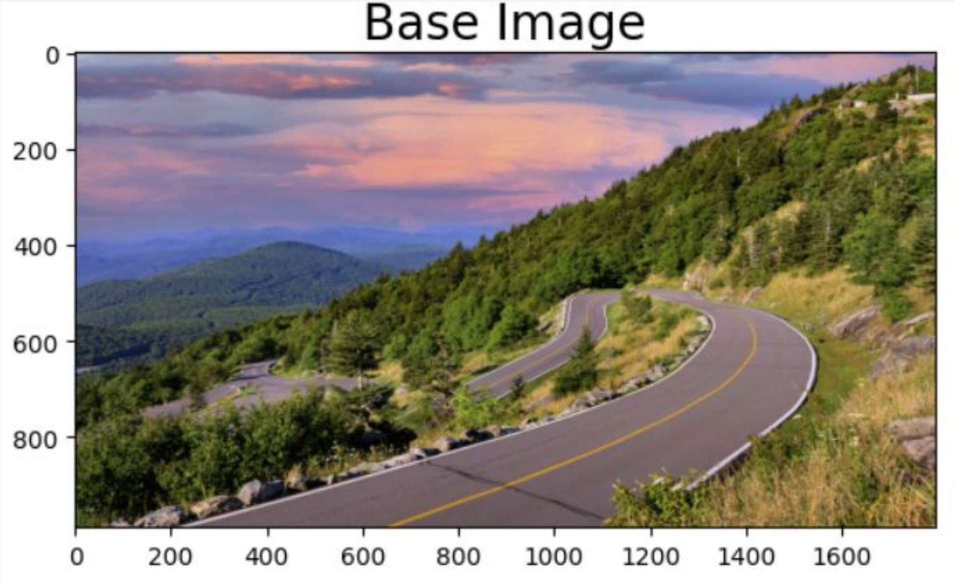
- Measure how well the generated image replicates the style of the original image by comparing the Gram matrices of their feature maps

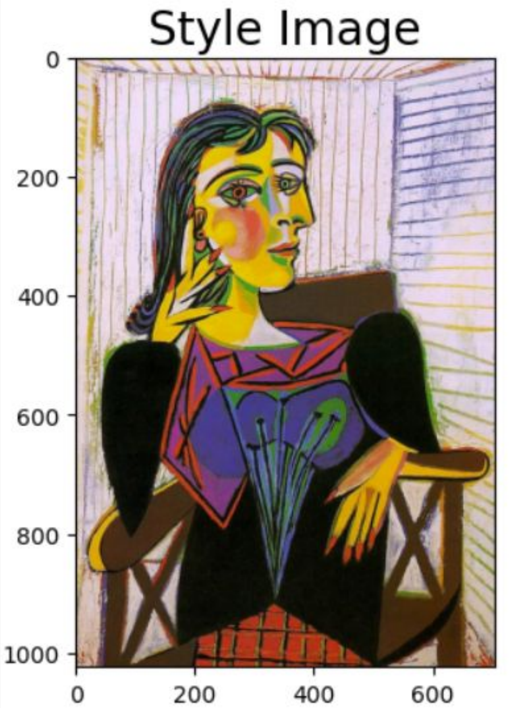
$$\mathcal{L}_{content} = \frac{1}{2}\sum_{i,j}(F^l_{ij} - P^l_{ij})^2$$

$$G_{ij} = \sum_k F_{ik}F_{jk}$$

$$\mathcal{L}_{style} = \frac{1}{2}\sum_{l=0}^{L}(G^l_{ij} - A^l_{ij})^2$$

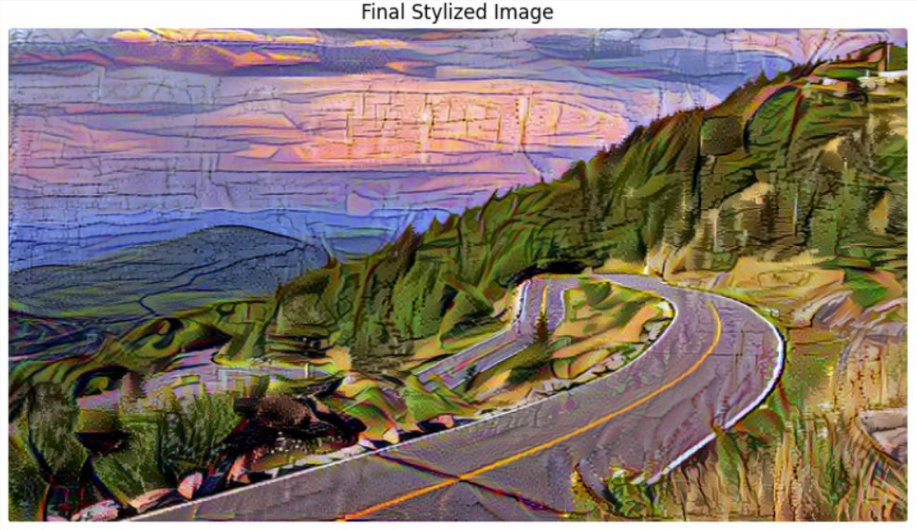
Art Style Transfer - Results





Art Style Transfer – VGG19

VGG19: Visibly more style transferred

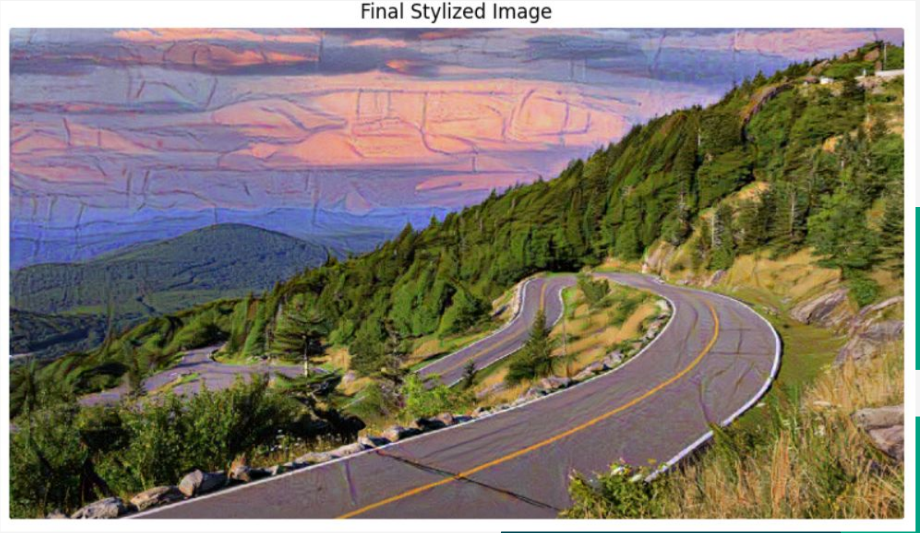


- Simplicity and ease of understanding
- Effective for tasks requiring detailed feature extraction

- Computationally intensive due to a large number of parameters
- Slow training and inference times

Art Style Transfer – InceptionV3

InceptionV3: Less visible style/ smoother line



- More efficient and faster due to fewer parameters
- Good at capturing a wide range of features at different scales

- More complex architecture, which can be harder to understand and implement

Conclusion (Artist)

- Focused on 11 artists with ≥ 200 paintings each.
- Used ResNet50 (55% accuracy) and VGG16 (36% accuracy) as based models
- Fine-tuned ResNet50
- Final ResNet50 model achieved 80% validation accuracy

Challenge

- Imbalanced data
- Computational Expensive

Conclusion (Art Style)

- Used InceptionV3 and VGG19 as based models to extract features
- Used loss functions to calculate loss and optimize style transfer
- VGG19 was able to extract and transfer more of original art style than InceptionV3

Challenge

- Computational Expensive

Thank you!

Appendix

Model Operations - Deployment

Architecture Overview:

- Collect and preprocess data
- Use a model serving framework to deploy the trained model
- Expose the model as an API endpoint using a web framework
- Distribute incoming requests across multiple instances of the model to ensure scalability and reliability
- Implement monitoring tools to track model performance and log predictions for auditing

Deployment Steps:

- Package the model and its dependencies into a Docker container
- Use Kubernetes to manage container deployment, scaling, and maintenance
- Set up CI/CD pipelines to automate the deployment process, ensuring that updates to the model are seamlessly integrated into the production environment

Model Operations - Maintenance

Regular Monitoring:

- Track accuracy

Scheduled Maintenance:

- Regularly retrain the model (monthly or quarterly) to incorporate new data and maintain accuracy
- Periodically update feature engineering processes to adapt to changes in data patterns (every 3-6 months)

Model Update Steps:

- Validate and preprocess new data, ensuring it is clean and representative
- Train the model with updated data and tuned hyperparameters
- Compare the new model's performance against the current model using validation metrics
- Deploy the updated model to the production environment, ensuring minimal downtime
- Continuously monitor the updated model's performance to ensure it meets the desired standards

Artist Identification Code

```
[ ] dataset_path = "/Users/kavyaemani/.cache/kagglehub/datasets/ikarus777/best-artworks-of-all-time/versions/1/resized/resized"

    # Create subdirectories and move images
    for file in os.listdir(dataset_path):
        if file.endswith(('.jpg', '.jpeg', '.png')):  # Ensure it's an image file
            artist_name = file.rsplit("_", 1)[0]  # Extract artist name from filename
            artist_dir = os.path.join(dataset_path, artist_name)  # Create folder per artist

            # Create artist folder if it doesn't exist
            os.makedirs(artist_dir, exist_ok=True)

            # Move image into the respective artist folder
            shutil.move(os.path.join(dataset_path, file), os.path.join(artist_dir, file))

    print("Images successfully organized into subdirectories!")
```

```
Images successfully organized into subdirectories!
```

```
[ ] # Normalize and Resize Images:Resize images to 224×224  ; Normalize pixel values to [0,1] (by dividing by 255)
    img_height, img_width = 224, 224
    batch_size = 32

    # Load dataset and split into training & validation sets
    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=batch_size
    )

    val_ds = tf.keras.preprocessing.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=batch_size
    )
    class_names = train_ds.class_names
    print("Class Names:",class_names)
```

```
Found 8683 files belonging to 51 classes.
Using 6947 files for training.
Found 8683 files belonging to 51 classes.
Using 1736 files for validation.
Class Names: ['Albrecht_DuΓòá├¬rer', 'Albrecht_Du╠êrer', 'Alfred_Sisley', 'Amedeo_Modigliani', 'Andrei_Rublev', 'Andy_
```

```
[ ] # Normalize the images (scale pixel values to [0,1])
    normalization_layer = layers.Rescaling(1./255)

    # Apply normalization to training and validation datasets
    train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
    val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))

    print("Images successfully loaded, resized, and normalized!")
```

```
[ ] data_augmentation = tf.keras.Sequential([
        layers.RandomFlip("horizontal"),  # Flip images randomly
        layers.RandomRotation(0.2),  # Rotate by 20%
        layers.RandomZoom(0.2),  # Zoom-in and out
        layers.RandomContrast(0.2)  # Slight contrast variations
    ])

    # Apply data augmentation to training dataset
    train_ds = train_ds.map(lambda x, y: (data_augmentation(x, training=True), y))

    print("Data augmentation applied to training dataset!")
```

Artist Identification Code – VGG16

```
[ ] # Load Pre-trained VGG16 model (without top classifier layers)
    base_model = VGG16(input_shape=(224, 224, 3),  # Image size (same as in preprocessing)
                       include_top=False,  # Remove final classification layers
                       weights='imagenet')

    # Freeze the base model's layers so they are not updated during training
    base_model.trainable = False

    # Build the classification model
    model = models.Sequential([
        base_model,                          # Pretrained VGG16 base
        layers.GlobalAveragePooling2D(),  # Convert feature maps to vector
        layers.Dense(128, activation='relu'),  # Fully connected layer
        layers.Dropout(0.5),                  # Dropout for regularization
        layers.Dense(len(class_names), activation='softmax')  # Output layer (Artist Classes)
    ])

    # Compile the model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Display model summary
    model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 512)	0
dense_5 (Dense)	(None, 128)	65,664
dropout_4 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 51)	6,579

Total params: 14,786,931 (56.41 MB)
Trainable params: 72,243 (282.20 KB)
Non-trainable params: 14,714,688 (56.13 MB)

Artist Identification Code – ResNet

```
[ ] # Load Pre-trained ResNet50 model (without classifier layers)
    base_model = ResNet50(input_shape=(224, 224, 3),
                          include_top=False,
                          weights='imagenet')

    # ✅ Keep shallow layers trainable to learn painting style
    for layer in base_model.layers[:50]:  # Freeze deeper layers
        layer.trainable = False
    for layer in base_model.layers[50:]:  # Fine-tune shallow layers
        layer.trainable = True

    # Build model
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(len(class_names), activation='softmax')  # Output layer
    ])

    # Compile model with lower learning rate for fine-tuning
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 9s 0us/step
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dense_2 (Dense)	(None, 256)	524,544
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 11)	2,827

```
Total params: 24,115,083 (91.99 MB)
Trainable params: 23,454,475 (89.47 MB)
Non-trainable params: 660,608 (2.52 MB)
```

Artist Identification Code – Fine-tune

Data Augmentation Using ImageDataGenerator

```
batch_size = 16
img_size = (224, 224)
input_shape = (224, 224, 3)

train_datagen = ImageDataGenerator(
    validation_split=0.2,
    rescale=1./255.,
    shear_range=5,
    horizontal_flip=True,
    vertical_flip=True
)

val_datagen = ImageDataGenerator(
    validation_split=0.2,
    rescale=1./255.
)

# Load training dataset
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical",
    subset="training",
    shuffle=True,
    classes=top_artists['name'].tolist()
)

# Load validation dataset
valid_generator = val_datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical",
    subset="validation",
    shuffle=True,
    classes=top_artists['name'].tolist()
)

# Get class labels
class_names = list(train_generator.class_indices.keys())
n_classes = len(class_names)

print("Class Labels:", class_names)
```

Build ResNet50 Model (Fine-Tuned)

```
# Load pre-trained ResNet50 model
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=input_shape)

# Make all layers trainable
for layer in base_model.layers:
    layer.trainable = True

# Add classification layers
X = base_model.output
X = Flatten()(X)

X = Dense(512, kernel_initializer="he_uniform")(X)
X = BatchNormalization()(X)
X = Activation("relu")(X)

X = Dense(16, kernel_initializer="he_uniform")(X)
X = BatchNormalization()(X)
X = Activation("relu")(X)

output = Dense(n_classes, activation="softmax")(X)

# Compile the model
model = Model(inputs=base_model.input, outputs=output)

optimizer = Adam(learning_rate=0.0001)
model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])

model.summary()
```

Artist Identification Code – Fine-tune

Train Model (Phase 1 - Train All Layers)

```
# Callbacks
early_stop = EarlyStopping(monitor="val_loss", patience=20, verbose=1, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5, verbose=1)

n_epochs = 10

history1 = model.fit(
    train_generator,
    validation_data=valid_generator,
    epochs=n_epochs,
    shuffle=True,
    verbose=1,
    callbacks=[reduce_lr],
    class_weight=class_weights
)
```

Train Model (Phase 2 - Freeze Deeper Layers & Fine-Tune)

```
# Freeze deeper ResNet layers
for layer in model.layers:
    layer.trainable = False
for layer in model.layers[:50]:  # Fine-tune only shallow layers
    layer.trainable = True

optimizer = Adam(learning_rate=0.0001)

model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])

n_epochs = 50

history2 = model.fit(
    train_generator,
    validation_data=valid_generator,
    epochs=n_epochs,
    shuffle=True,
    verbose=1,
    callbacks=[reduce_lr, early_stop],
    class_weight=class_weights
)
```

Art Style Transfer Code

VGG

```
[ ] def preprocess_image(image_path):
        from keras.applications import vgg19
        img = load_img(image_path, target_size=(img_nrows, img_ncols))
        img = img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = vgg19.preprocess_input(img)
        return img
```

```
def preprocess_image(image_path, img_nrows, img_ncols, model_name):
    """Load and preprocess the image for a given model."""
    img = load_img(image_path, target_size=(img_nrows, img_ncols))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    if model_name == 'vgg19':
        img = vgg19.preprocess_input(img)
    elif model_name == 'inceptionv3':
        img = inception_v3.preprocess_input(img)
    return img

def deprocess_image(x, img_nrows, img_ncols, model_name):
    """Revert a preprocessed tensor into a valid image."""
    x = x.reshape((img_nrows, img_ncols, 3))
    if model_name == 'vgg19':
        # VGG19 preprocess subtracts mean RGB; revert that
        x[:, :, 0] += 103.939
        x[:, :, 1] += 116.779
        x[:, :, 2] += 123.68
        # Convert from BGR to RGB
        x = x[:, :, ::-1]
    elif model_name == 'inceptionv3':
        # InceptionV3 scales input to [-1, 1]
        x = (x + 1) * 127.5
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
# ---------------------------
# Loss Functions
# ---------------------------
def gram_matrix(x):
    """Compute the Gram matrix for an image tensor (H, W, C)."""
    # Permute dimensions to (C, H, W) and then reshape to (C, H*W)
    x = tf.transpose(x, perm=[2, 0, 1])
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, features, transpose_b=True)
    return gram

def get_content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))

def get_style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    return tf.reduce_sum(tf.square(S - C))
```

```
[ ] def create_feature_extractor(model_name, img_nrows, img_ncols):
        """Return a model that outputs a list of style and content features.

           For VGG19, we use 'block5_conv2' for content and
           ['block1_conv1','block2_conv1','block3_conv1','block4_conv1','block5_conv1']
           for style. For InceptionV3 we pick some "mixed" layers.
        """
        if model_name == 'vgg19':
            content_layer = 'block5_conv2'
            style_layers = ['block1_conv1', 'block2_conv1',
                            'block3_conv1', 'block4_conv1',
                            'block5_conv1']
            base_model = vgg19.VGG19(include_top=False, weights='imagenet',
                                     input_shape=(img_nrows, img_ncols, 3))
        elif model_name == 'inceptionv3':
            content_layer = 'mixed7'
            style_layers = ['mixed0', 'mixed1', 'mixed2', 'mixed3']
            base_model = inception_v3.InceptionV3(include_top=False, weights='imagenet',
                                                  input_shape=(img_nrows, img_ncols, 3))
        else:
            raise ValueError("model_name must be either 'vgg19' or 'inceptionv3'.")
        style_outputs = [base_model.get_layer(name).output for name in style_layers]
        content_output = base_model.get_layer(content_layer).output
        model = tf.keras.Model(inputs=base_model.input,
                               outputs=style_outputs + [content_output])
        model.trainable = False
        return model, content_layer, style_layers
```

Art Style Transfer Code

```python
def run_style_transfer(base_image_path, style_image_path, model_name='vgg19',
                       iterations=10, content_weight=0.025, style_weight=1.0, img_nrows=400):
    """Execute style transfer and return the final stylized image."""
    # Determine image dimensions (maintaining aspect ratio)
    width, height = load_img(base_image_path).size
    img_ncols = int(width * img_nrows / height)

    # Load and preprocess images.
    base_image_np = preprocess_image(base_image_path, img_nrows, img_ncols, model_name)
    style_image_np = preprocess_image(style_image_path, img_nrows, img_ncols, model_name)

    # Create tensors
    base_image = tf.constant(base_image_np, dtype=tf.float32)
    style_image = tf.constant(style_image_np, dtype=tf.float32)
    # Initialize the combination image with the base image.
    combination_image = tf.Variable(base_image_np, dtype=tf.float32)

    # Create the feature extractor model.
    feature_extractor, _, _ = create_feature_extractor(model_name, img_nrows, img_ncols)

    # Wrap loss and gradients for LBFGS.
    evaluator = Evaluator(base_image, style_image, combination_image,
                          feature_extractor, content_weight, style_weight)

    # Flatten initial combination image for the optimizer.
    x_opt = combination_image.numpy().flatten()

    for i in range(iterations):
        print(f"Start of iteration {i}")
        x_opt, min_val, info = fmin_l_bfgs_b(evaluator.loss, x_opt,
                                              fprime=evaluator.grads,
                                              maxfun=20, disp=True)
        print(f"Current loss value: {min_val}")

    # Reshape and deprocess the best image.
    best_img = x_opt.reshape(combination_image.shape)
    final_img = deprocess_image(best_img, img_nrows, img_ncols, model_name)
    return final_img
```

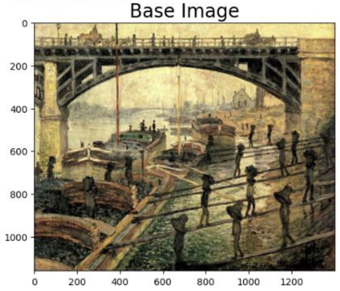
Art Style Transfer Code

```
[ ] image1 = run_style_transfer( "/kaggle/input/new-pic/Screenshot 2025-02-19 at 11.10.15PM.png","/kaggle/input/best-artworks-of-all-time/images/images/Claude_Monet/Claude_Monet_16.jpg", model_name='vgg19', iterations=1)
```

```
Start of iteration 0
Current loss value: 2.4891839870939934e+21
```

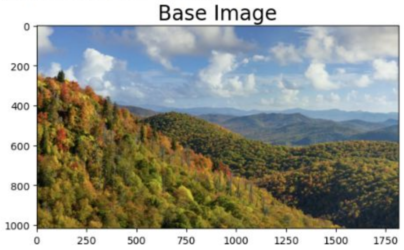
```
[ ] # plt.imshow("/kaggle/input/best-artworks-of-all-time/images/images/Claude_Monet/Claude_Monet_16.jpg")
    # plt.imshow("/kaggle/input/new-pic/Screenshot 2025-02-19 at 11.10.15PM.png")
    # plt.show()
    plt.figure()
    plt.title("Style Image",fontsize=20)
    img2 = load_img("/kaggle/input/best-artworks-of-all-time/images/images/Claude_Monet/Claude_Monet_16.jpg")
    plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7d01a6ade140>
```

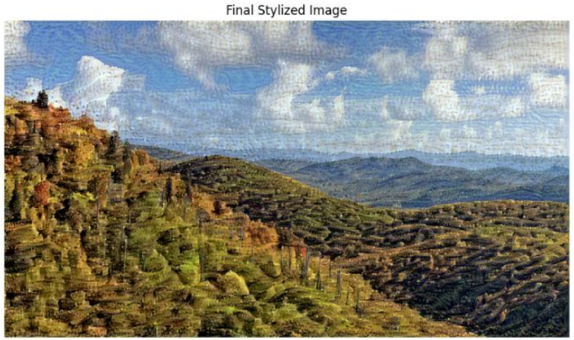



```
[ ] plt.figure()
    plt.title("Base Image",fontsize=20)
    img3 = load_img("/kaggle/input/new-pic/Screenshot 2025-02-19 at 11.10.15PM.png")
    plt.imshow(img3)
```

```
<matplotlib.image.AxesImage at 0x7d010505b640>
```




```
[ ] plt.figure(figsize=(10,10))
    plt.imshow(image1)
    plt.title("Final Stylized Image")
    plt.axis('off')
    plt.show()
```



Interesting Resources.

Art Style Transfer:

- https://www.kaggle.com/code/basu369victor/style-transfer-deep-learning-algorithm