Deep Learning Package Documentation
Neural Style Transfer with Visual Geometry Group-19 model
MSc Data Science (3rd year)

Nithin V (22pd24)
Harshan M V (22pd14)

Introduction:

This document provides detailed documentation for a neural style transfer script implemented using TensorFlow. Neural Style Transfer (NST) is a technique that takes two images – a content image and a style image – and produces an image that combines the content of the first image with the artistic style of the second image. This script uses a pre-trained VGG19 model for extracting content and style features and performs optimization to generate a stylized image.

Project Implementation:

1. Neural Style Transfer with Pre-trained VGG19 Model for images (content image +style image= output image)
2. Neural Style Transfer with Pre-trained VGG19 Model for video clips
3. Neural Style Transfer from input image to room interiors
4. Neural Style Transfer on an image by extracting audio features

Neural Style Transfer with Pre-trained VGG19 Model for images:

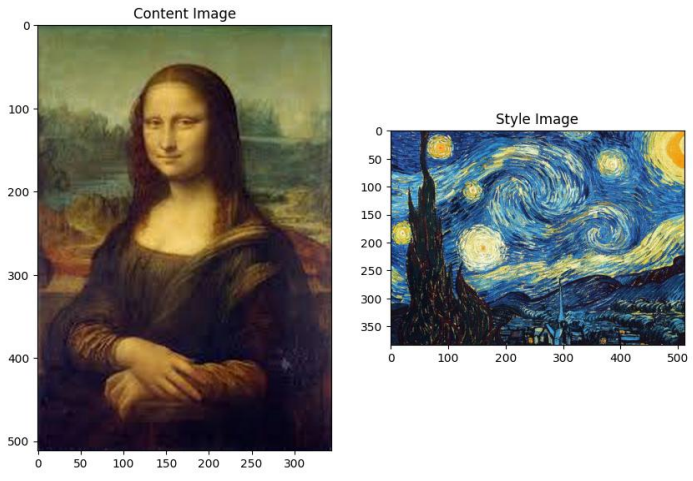


Image Loading and Preprocessing:

In this part, the code loads and preprocesses the images that will be used for style transfer. The load_file() function is responsible for opening an image file, resizing it to ensure it fits the input size requirements of the VGG19 model, and converting it into an array that can be processed by TensorFlow. The image is then expanded to add a batch dimension, as TensorFlow models require a batch of images, even if it's just one image.
The show_im() function is used to display the image in a matplotlib plot. The image is squeezed to remove any extra dimensions and shown with an optional title.

Image Preprocessing for VGG19:

The img_preprocess() function prepares the image for input into the VGG19 model. It applies the necessary preprocessing steps, such as normalization, so the model can work with the image correctly.
The deprocess_img() function is used later in the process to reverse the preprocessing applied to the image. This step is necessary to bring the image back to a viewable state (in terms of pixel values) after the style transfer process is complete.

Load and Display Images:

In this section, the content and style images are loaded using the load_file() function. These images are then displayed side by side using show_im(). This allows you to visually compare the content image and the style image before performing the style transfer.

Content and Style Layers:

- content_layers specifies layers for content features (e.g., block5_conv2).
- style_layers specifies layers for style features (e.g., block1_conv1, block2_conv1, etc.).

get_model():

- Loads the VGG19 model pre-trained on ImageNet.
- Creates a custom model extracting both style and content features.

Model Summary:

- Displays the architecture of the original VGG19 model and the custom model.

Layer (type)	Output Shape	Param #
input_layer_20 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1,792
block1_conv2 (Conv2D)	(None, None, None, 64)	36,928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73,856
block2_conv2 (Conv2D)	(None, None, None, 128)	147,584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295,168
block3_conv2 (Conv2D)	(None, None, None, 256)	590,080
block3_conv3 (Conv2D)	(None, None, None, 256)	590,080
block3_conv4 (Conv2D)	(None, None, None, 256)	590,080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1,180,160
block4_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv4 (Conv2D)	(None, None, None, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv4 (Conv2D)	(None, None, None, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

get_content_loss(noise, target):

This function computes the content loss by calculating the mean squared error between the noise image and the target content image.

gram_matrix(tensor):

This function computes the Gram matrix of a given tensor, which is useful for capturing the style of an image.
It reshapes the tensor to a 2D vector, then computes the matrix multiplication of the vector with its transpose.

get_style_loss(noise, target):

This function computes the style loss by calculating the difference between the Gram matrix of the noise image and the target style, using the mean squared error.

```python
# %%
def run_style_transfer(content_path, style_path, epochs=500, content_weight=1e3, style_weight=1e-2):
    model = get_model()
    for layer in model.layers:
        layer.trainable = False
    content_feature, style_feature = get_features(model, content_path, style_path)
    style_gram_matrix = [gram_matrix(feature) for feature in style_feature]
    noise = img_preprocess(content_path)
    noise = tf.Variable(noise, dtype=tf.float32)
    optimizer = tf.keras.optimizers.Adam(learning_rate=5.0, beta_1=0.99, epsilon=1e-1)
    best_loss, best_img = float('inf'), None
    loss_weights = (style_weight, content_weight)
    dictionary = {'model': model,
                  'loss_weights': loss_weights,
                  'image': noise,
                  'gram_style_features': style_gram_matrix,
                  'content_features': content_feature}
    norm_means = np.array([103.939, 116.779, 123.68])
    min_vals = -norm_means
    max_vals = 255 - norm_means
    imgs = []
    for i in range(epochs):
        grad, all_loss = compute_grads(dictionary)
        total_loss, style_loss, content_loss = all_loss
        optimizer.apply_gradients([(grad, noise)])
        clipped = tf.clip_by_value(noise, min_vals, max_vals)
        noise.assign(clipped)
        if total_loss < best_loss:
            best_loss = total_loss
            best_img = deprocess_img(noise.numpy())
        if i % 5 == 0:
            plot_img = noise.numpy()
            plot_img = deprocess_img(plot_img)
            imgs.append(plot_img)
            # Display the image for the current epoch
            plt.imshow(plot_img)
            plt.title(f'Epoch: {i}')
            plt.show()  # Show the image for the current epoch

            print('Epoch: {}'.format(i))
            print('Total loss: {:.4e}, '
                  'style loss: {:.4e}, '
                  'content loss: {:.4e}, '.format(total_loss, style_loss, content_loss))

    return best_img, best_loss, imgs
```

The **run_style_transfer** function performs neural style transfer by optimizing an image to match the style of a style image while retaining the content of a content image.

Parameters:

- **content_path**: Path to the content image.
- **style_path**: Path to the style image.
- **epochs**: Number of optimization steps (default 500).
- **content_weight**: Weight for content loss (default 1e3).
- **style_weight**: Weight for style loss (default 1e-2).

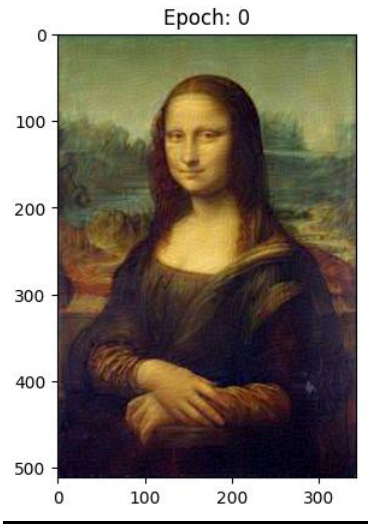
Workflow:

1. **Model Setup**: Load VGG19 model with selected content and style layers.
2. **Feature Extraction**: Extract content and style features from the images.
3. **Noise Initialization**: Start with the content image and optimize it.
4. **Optimization**: Minimize the total loss (content + style) using the Adam optimizer.
5. **Update Image**: Clip and update the image after each optimization step.
6. **Tracking Best Image**: Keep track of the best stylized image based on the lowest loss.

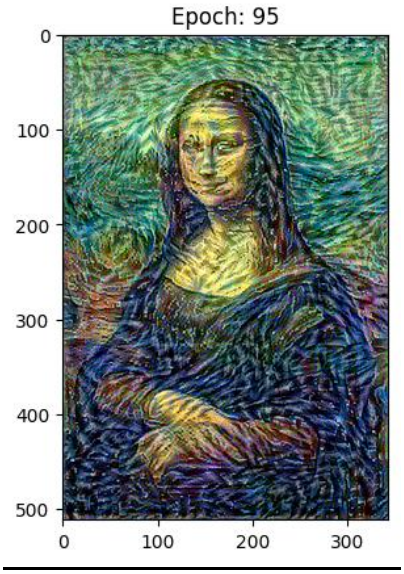
Output:

- **best_img**: Final stylized image.
- **best_loss**: Loss at the best image.
- **imgs**: List of images generated during optimization.

Initial Image (Content Image) at Epoch 0:



Final Image after 100 epochs:



Neural Style Transfer with Pre-trained VGG19 Model for video clips:

Key Steps:

1. **Extract Frames**:
    - Use OpenCV to read video frames and save them as images.
    - Directory is created to store the frames if it doesn't exist.
2. **Recreate Video**:
    - Collect the styled frames, ensure proper sorting, and use OpenCV to write these frames into a new video.

Output:

- Frames are extracted from the video and stored in the "frames" folder.
- A styled video is saved as styled_video.mp4 in the /content directory.

☑ 64 frames extracted! (from a 3 second MP4 clip)- Converted into frames. Each frame processed. Final video stored as styled_video.

☑ Styled video saved at: /content/styled_video.mp4

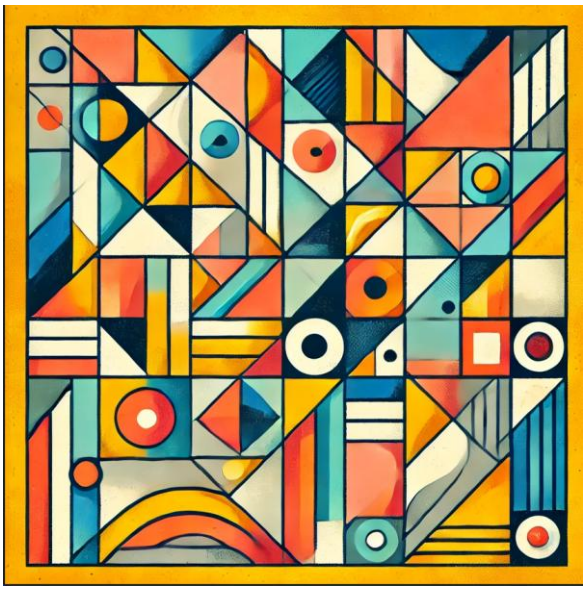
Neural Style Transfer from input image to room interiors:

In this neural style transfer (NST) implementation, the content image is a room photo, and the style image is a painting. The goal is to apply the artistic style of the painting (such as textures, colors, and brushstrokes) to the room image

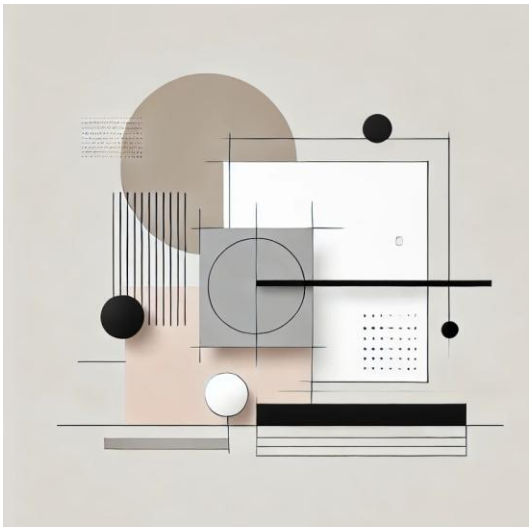
while preserving the structure and layout of the room. The VGG19 model extracts content features from the room image and style features from the painting. These features are combined through a loss function that balances content and style, resulting in a stylized room image. The room's basic structure remains intact, but it takes on the visual characteristics of the chosen painting style.

Initial room image:

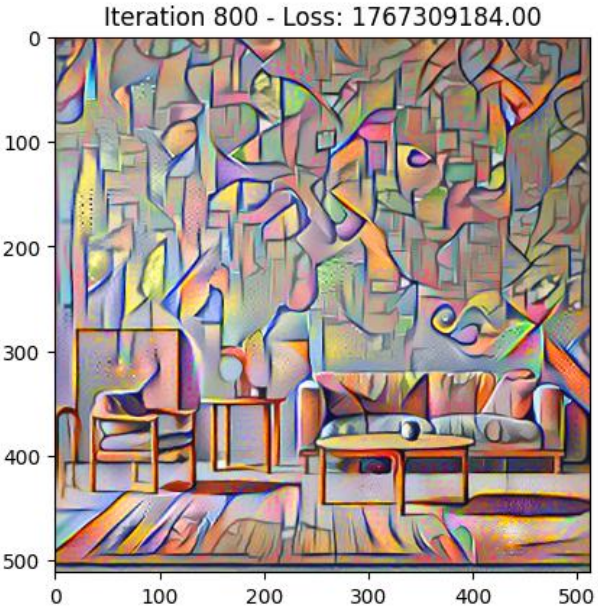
Boho Style:



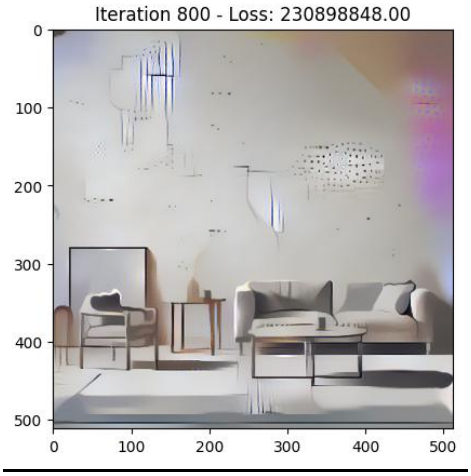
Rustic Style:



Boho Style Interior (after 800 iterations):



Rustic Style Interior (after 800 iterations):



Neural Style Transfer on an image by extracting audio features:

In this code, style transfer is done using a **room image** as the content and an **audio file** as the style source. The audio is converted into a colorful, textured image (with spectrograms, rhythm lines, tonal circles, and noise) to represent its "style." This audio-derived image replaces a traditional painting as the style reference. The VGG19 model extracts content features from the room and style features from the audio image, then blends them to produce a stylized room image that looks like it's painted using the audio's visual patterns.

Function to convert audio file into style image:

```python
# Create a more visually interesting audio-derived style image
def create_advanced_audio_style_image(audio_path):
    y, sr = librosa.load(audio_path)

    # Create multiple audio features
    mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
    mel_db = librosa.power_to_db(mel_spec, ref=np.max)
    mel_norm = (mel_db - mel_db.min()) / (mel_db.max() - mel_db.min())

    # Get rhythm/temporal features
    tempo, beats = librosa.beat.beat_track(y=y, sr=sr)
    onset_env = librosa.onset.onset_strength(y=y, sr=sr)
    onset_norm = (onset_env - onset_env.min()) / (onset_env.max() - onset_env.min())

    # Frequency features
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_norm = (chroma - chroma.min()) / (chroma.max() - chroma.min())

    # Create base image from mel spectrogram
    spec_img = (mel_norm * 255).astype(np.uint8)
    spec_img_resized = cv2.resize(spec_img, (512, 512))

    # Create a multi-colored image
    colored_img = np.zeros((512, 512, 3), dtype=np.uint8)

    # Apply different colormaps to different channels for more interesting patterns
    colored_img[:,:,0] = cv2.applyColorMap(spec_img_resized, cv2.COLORMAP_JET)[:,:,0]
    colored_img[:,:,1] = cv2.applyColorMap(spec_img_resized, cv2.COLORMAP_PLASMA)[:,:,1]
    colored_img[:,:,2] = cv2.applyColorMap(spec_img_resized, cv2.COLORMAP_VIRIDIS)[:,:,2]

    # Add some rhythmic patterns
    if len(beats) > 0:
        for i, beat in enumerate(beats):
            if i % 2 == 0:  # Every other beat
                pos = int((beat / len(y)) * 512)
                cv2.line(colored_img, (0, pos), (512, pos), (255, 255, 255),
                         thickness=2)

    # Add audio tonal features as circles
    chroma_resized = cv2.resize((chroma_norm * 255).astype(np.uint8), (12, 12))
    for i in range(12):
        for j in range(12):
            radius = int(chroma_resized[i,j] / 25) + 5
            center_x = int(i * (512/12)) + 25
            center_y = int(j * (512/12)) + 25
            cv2.circle(colored_img, (center_x, center_y), radius,
                       (255, 128 + i*10, 128 + j*10), -1)

    # Add some texture
    noise = np.random.randint(0, 100, (512, 512, 3), dtype=np.uint8)
    final_img = cv2.addWeighted(colored_img, 0.8, noise, 0.2, 0)

    return final_img
```

Mel-Spectrogram (Texture Base):

- mel_spec → Captures frequency content over time.
- Converted to dB scale, normalized, resized to 512x512 → forms base grayscale image.

Color Mapping (Visual Patterns):

- Applies 3 different colormaps (JET, PLASMA, VIRIDIS) to 3 RGB channels separately → gives vibrant, unique textures.

Beat Lines (Rhythm Info):

- Detects beats using librosa.beat_track.
- Draws horizontal lines at beat positions → shows temporal rhythm visually.

Chroma Circles (Pitch/Tonality Info):

- Uses chroma features (12 pitch classes).
- Each value becomes a **circle** with size & color based on pitch strength.

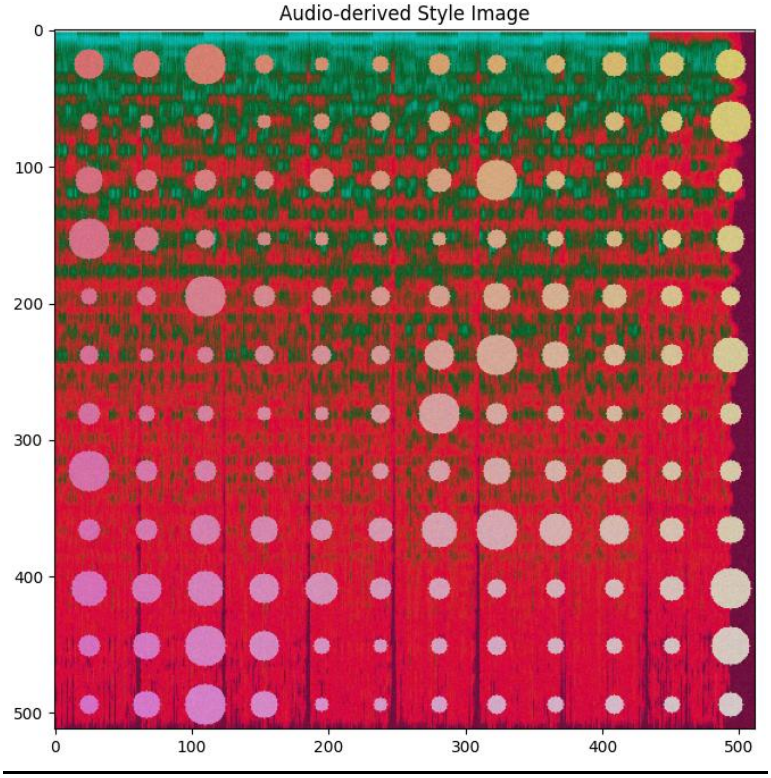
Noise Texture:

- Adds random noise blended into the image to enhance texture variation.

Audio Derived Style Image:

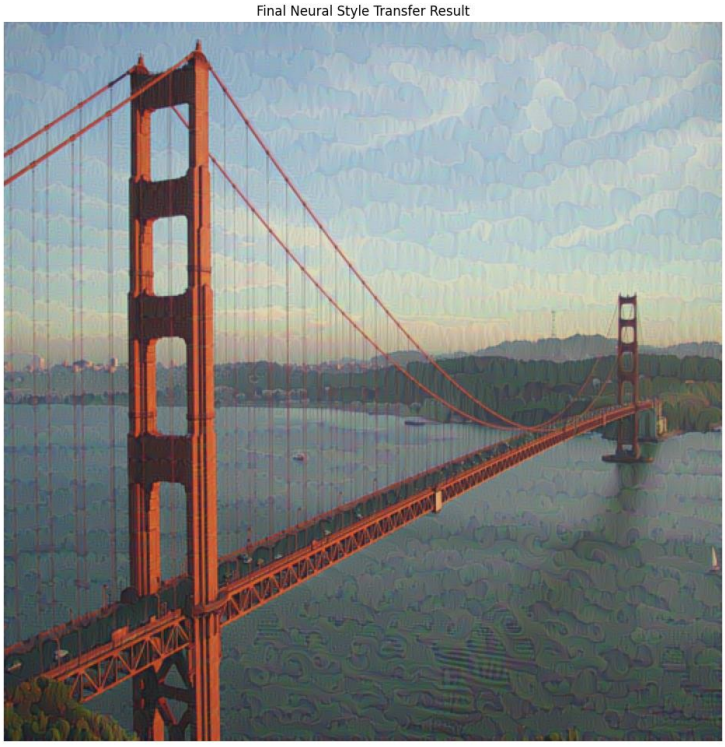
Audio: Acoustic Breeze by Bensound

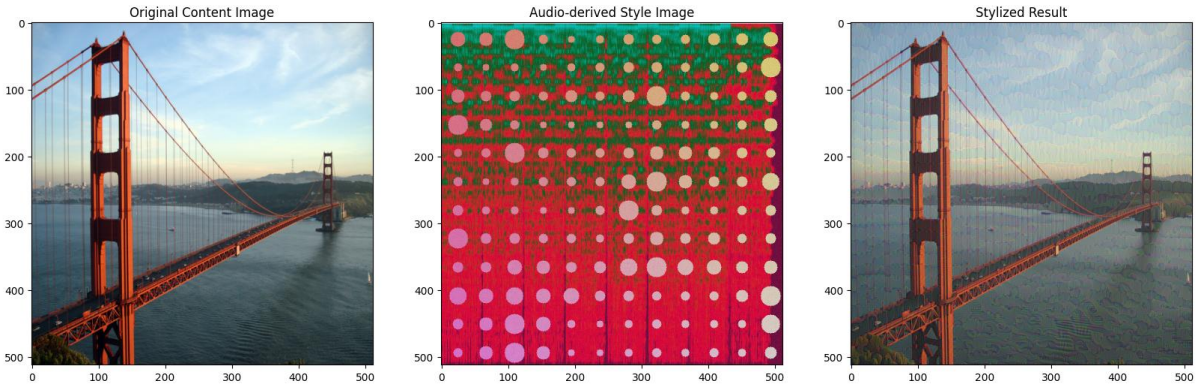
https://www.bensound.com/royalty-free-music/track/acoustic-breeze



Content Image: Golden Gate Bridge

After 500 steps:





Conclusion:

This project explored how **Neural Style Transfer (NST)** using the **VGG19 model** can be used in different ways:

1. **Image Style Transfer**
   A content image and a style image are combined to create a new artistic image.
2. **Video Style Transfer**
   The same idea is used on videos, styling each frame while keeping it smooth.
3. **Image to Room Style Transfer**
   Room images are styled using other images, which can help in interior design.
4. **Audio to Style Transfer**
   Sounds are turned into visual patterns using features like spectrograms. These are then used as style inputs in NST.

✅ **Drive Link:**
https://drive.google.com/file/d/1_z35MSuY_DL-O-l4xALmjGohuR-s161J/view?usp=sharing

✅ Github repo:
https://github.com/NithinVinukumar/DeepLearning-NST