**Artistic Style Transfer Using TensorFlow Lite**

### **1. Project Title**

**Artistic Style Transfer Using TensorFlow Lite**

This project implements an artistic style transfer model using TensorFlow Lite to apply artistic styles to content images efficiently. The implementation follows the approach provided in TensorFlow's official repository.

### **2. Dataset Collection and Preparation**

* **Dataset Source:** Online dataset - **Fast Neural Style Transfer Dataset**
  + **Content Images:** Natural scene images from the dataset.
  + **Style Images:** A curated set of paintings and artistic textures.
* **Dataset Information:**
  + Includes a diverse selection of **artistic styles** and **content images** for effective style adaptation.
* **Pretrained Models:**
  + **Style Prediction Model:** style\_predict.tflite
  + **Style Transfer Model:** style\_transform.tflite

### **3. Data Preprocessing**

* **Loading and Preprocessing Steps:**
  + Convert image to a tensor.
  + Resize images for model input (e.g., 256x256 for style, 384x384 for content).
  + Normalize pixel values to [0,1].
  + Central crop to maintain aspect ratio.
* **Feature Selection Methods:**
  + Content images are chosen based on diversity in structure and complexity.
  + Style images include different textures, colors, and artistic elements for comparison.
* **Data Augmentation:**
  + Random cropping and flipping to improve generalization.
  + Histogram equalization for consistent lighting conditions.

### **4. Pattern Identification and Justification**

* **Content Image:**
  + Provides structural information (shapes, objects, layout).
* **Style Image:**
  + Contains textures, colors, and artistic strokes.
* **Pattern Identification:**
  + The challenge is balancing content preservation with style transfer.
  + Style extraction is done using TensorFlow Lite’s style\_predict.tflite model.
  + Justification: The balance between preserving content while integrating artistic style is crucial for high-quality results.

### **5. Models Used**

* **TensorFlow Lite Models for Style Transfer**
  + **Style Prediction Model (style\_predict.tflite)**
    - Extracts the style bottleneck from the style image.
  + **Style Transfer Model (style\_transform.tflite)**
    - Applies the style bottleneck to the content image.
  + **Justification:**
    - Optimized for speed and lightweight execution on mobile devices.
    - Pretrained models reduce computation time compared to traditional NST.

### **6. Performance Metrics**

* **Content Loss** (Preserving original image structure)
* **Style Loss** (Measuring similarity to artistic style)
* **Inference Speed** (Optimized execution time using TensorFlow Lite)
* **Perceptual Quality** (Visual evaluation of results)
* **Comparison with other models:**
  + Comparing results with other Neural Style Transfer implementations (e.g., Fast Neural Style Transfer, Deep Photo Style Transfer).
  + Measuring accuracy based on content retention and artistic resemblance.

### **7. Conclusion and Next Steps**

* Successfully implemented **TensorFlow Lite-based Style Transfer**.
* The model enables real-time artistic transformations.
* Future improvements:
  + Experiment with more styles.
  + Optimize model performance for better edge-device deployment.
  + Evaluate using user preference surveys for quality assessment.
  + Extend the model to support video-based style transfer.

import IPython.display as display

import matplotlib.pyplot as plt

import matplotlib as mpl

mpl.rcParams['figure.figsize'] = (12,12)

mpl.rcParams['axes.grid'] = False

import numpy as np

import time

import functools

from skimage.metrics import structural\_similarity as ssim

# content\_path = tf.keras.utils.get\_file('my.jpg','https://www.w3schools.com/w3images/lights.jpg')

# style\_path = tf.keras.utils.get\_file('style23.jpg','https://storage.googleapis.com/khanhlvg-public.appspot.com/arbitrary-style-transfer/style23.jpg')

content\_path = '/content/drive/MyDrive/mist 4-2 sessional/cse444\_pr/PR project/content/my\_pic.jpg'

style\_path = '/content/drive/MyDrive/mist 4-2 sessional/cse444\_pr/PR project/style/candy.jpg'

style\_predict\_path = tf.keras.utils.get\_file('style\_predict.tflite', 'https://tfhub.dev/google/lite-model/magenta/arbitrary-image-stylization-v1-256/int8/prediction/1?lite-format=tflite')

style\_transform\_path = tf.keras.utils.get\_file('style\_transform.tflite', 'https://tfhub.dev/google/lite-model/magenta/arbitrary-image-stylization-v1-256/int8/transfer/1?lite-format=tflite')

# Function to load an image from a file, and add a batch dimension.

def load\_img(path\_to\_img):

img = tf.io.read\_file(path\_to\_img)

img = tf.io.decode\_image(img, channels=3)

img = tf.image.convert\_image\_dtype(img, tf.float32)

img = img[tf.newaxis, :]

return img

# Function to pre-process by resizing an central cropping it.

def preprocess\_image(image, target\_dim):

# Resize the image so that the shorter dimension becomes 256px.

shape = tf.cast(tf.shape(image)[1:-1], tf.float32)

short\_dim = min(shape)

scale = target\_dim / short\_dim

new\_shape = tf.cast(shape \* scale, tf.int32)

# print('-------')

# print(shape.numpy())

# print(scale.numpy())

# print(new\_shape.numpy())

# print('-------')

image = tf.image.resize(image, new\_shape)

# Central crop the image.

image = tf.image.resize\_with\_crop\_or\_pad(image, target\_dim, target\_dim)

return image

# Load the input images.

content\_image = load\_img(content\_path)

style\_image = load\_img(style\_path)

print('Style Image Shape:', style\_image.shape)

print('Content Image Shape:', content\_image.shape)

# Preprocess the input images.

preprocessed\_content\_image = preprocess\_image(content\_image, 384)

preprocessed\_style\_image = preprocess\_image(style\_image, 256)

print('Preprocessed Style Image Shape:', preprocessed\_style\_image.shape)

print('Preprocessed Content Image Shape:', preprocessed\_content\_image.shape)

def imshow(image, title=None):

if len(image.shape) > 3:

image = tf.squeeze(image, axis=0)

plt.imshow(image)

if title:

plt.title(title)

plt.subplot(1, 2, 1)

imshow(preprocessed\_content\_image, 'Content Image')

plt.subplot(1, 2, 2)

imshow(preprocessed\_style\_image, 'Style Image')

# Function to run style prediction on preprocessed style image.

def run\_style\_predict(preprocessed\_style\_image):

# Load the model.

interpreter = tf.lite.Interpreter(model\_path=style\_predict\_path)

# Set model input.

interpreter.allocate\_tensors()

input\_details = interpreter.get\_input\_details()

print(input\_details)

interpreter.set\_tensor(input\_details[0]["index"], preprocessed\_style\_image)

# Calculate style bottleneck.

interpreter.invoke()

style\_bottleneck = interpreter.tensor(

interpreter.get\_output\_details()[0]["index"]

)()

return style\_bottleneck

# Calculate style bottleneck for the preprocessed style image.

style\_bottleneck = run\_style\_predict(preprocessed\_style\_image)

print('Style Bottleneck Shape:', style\_bottleneck.shape)

# Run style transform on preprocessed style image with performance metrics

def run\_style\_transform(style\_bottleneck, preprocessed\_content\_image):

# Load the model.

interpreter = tf.lite.Interpreter(model\_path=style\_transform\_path)

# Set model input.

input\_details = interpreter.get\_input\_details()

# print(input\_details)

interpreter.allocate\_tensors()

# Measure execution time

start\_cpu = time.process\_time()

start\_wall = time.time()

# Set model inputs.

interpreter.set\_tensor(input\_details[0]["index"], preprocessed\_content\_image)

interpreter.set\_tensor(input\_details[1]["index"], style\_bottleneck)

interpreter.invoke()

# Transform content image.

stylized\_image = interpreter.tensor(

interpreter.get\_output\_details()[0]["index"]

)()

end\_cpu = time.process\_time()

end\_wall = time.time()

# Compute Total Loss (MSE)

mse\_loss = np.mean((preprocessed\_content\_image - stylized\_image) \*\* 2)

# Compute Structural Similarity Index (SSIM)

# Convert TensorFlow tensors to NumPy arrays before SSIM calculation

ssim\_score = ssim(

preprocessed\_content\_image.numpy().squeeze(),

stylized\_image.squeeze(),

multichannel=True,

data\_range=1.0,

win\_size=3

)

# Time Taken

cpu\_time = end\_cpu - start\_cpu

wall\_time = end\_wall - start\_wall

# Print Metrics

print(f"Total Loss (MSE): {mse\_loss:.6f}")

print(f"Structural Similarity Index (SSIM): {ssim\_score:.4f}")

print(f"CPU Time: {cpu\_time \* 1e6:.2f} µs")

print(f"Wall Time: {wall\_time \* 1e6:.2f} µs")

return stylized\_image

# # Stylize the content image using the style bottleneck.

stylized\_image = run\_style\_transform(style\_bottleneck, preprocessed\_content\_image)

# Visualize the output.

imshow(stylized\_image, 'Stylized Image')

# Import required libraries

import time

import numpy as np

import tensorflow as tf

from skimage.metrics import structural\_similarity as ssim

import matplotlib.pyplot as plt

import tensorflow\_hub as hub

# Define image paths

content\_path = '/content/drive/MyDrive/mist 4-2 sessional/cse444\_pr/PR project/content/my\_pic.jpg'

style\_path = '/content/drive/MyDrive/mist 4-2 sessional/cse444\_pr/PR project/style/feathers.jpg'

# Load content and style images

content\_image = plt.imread(content\_path)

style\_image = plt.imread(style\_path)

# Convert images to float32 numpy arrays, add batch dimension, and normalize to range [0, 1]

content\_image = content\_image.astype(np.float32)[np.newaxis, ...] / 255.

style\_image = style\_image.astype(np.float32)[np.newaxis, ...] / 255.

# Resize style image to 256x256 (recommended size for style image)

style\_image = tf.image.resize(style\_image, (256, 256))

# Load image stylization module.

hub\_module = hub.load('https://kaggle.com/models/google/arbitrary-image-stylization-v1/frameworks/TensorFlow1/variations/256/versions/1')

def run\_style\_transfer(content\_image, style\_image, hub\_module):

# Measure execution time (CPU and wall time)

start\_cpu = time.process\_time()

start\_wall = time.time()

# Stylize the content image using the style image

outputs = hub\_module(tf.constant(content\_image), tf.constant(style\_image))

# Extract the stylized image from the outputs

stylized\_image = outputs[0]

# Resize the stylized image to match the content image shape

stylized\_image\_resized = tf.image.resize(stylized\_image, content\_image.shape[1:3])

# Measure execution time (CPU and wall time)

end\_cpu = time.process\_time()

end\_wall = time.time()

# Convert TensorFlow tensors to NumPy arrays before performing operations

content\_image\_np = content\_image.squeeze()

stylized\_image\_resized\_np = stylized\_image\_resized.numpy().squeeze()

# Compute Total Loss (Mean Squared Error - MSE)

mse\_loss = np.mean((content\_image\_np - stylized\_image\_resized\_np) \*\* 2)

# Compute Structural Similarity Index (SSIM)

# Set win\_size to 3 (or any odd number smaller than the smallest image dimension)

ssim\_score = ssim(content\_image\_np, stylized\_image\_resized\_np,

multichannel=True, data\_range=1.0, win\_size=3)

# Time taken for the process

cpu\_time = (end\_cpu - start\_cpu) \* 1e6 # Convert to microseconds

wall\_time = (end\_wall - start\_wall) \* 1e6 # Convert to microseconds

# Print performance metrics

print(f"Total Loss (MSE): {mse\_loss:.6f}")

print(f"Structural Similarity Index (SSIM): {ssim\_score:.4f}")

print(f"CPU Time: {cpu\_time:.2f} µs")

print(f"Wall Time: {wall\_time:.2f} µs")

return stylized\_image\_resized

# Run style transfer and evaluate performance

stylized\_image = run\_style\_transfer(content\_image, style\_image, hub\_module)

# Display the original content, style, and stylized images

plt.figure(figsize=(12, 12))

# Content image

plt.subplot(1, 3, 1)

plt.imshow(content\_image[0])

plt.title('Content Image')

plt.axis('off')

# Style image

plt.subplot(1, 3, 2)

plt.imshow(style\_image[0])

plt.title('Style Image')

plt.axis('off')

# Stylized image

plt.subplot(1, 3, 3)

plt.imshow(stylized\_image[0])

plt.title('Stylized Image')

plt.axis('off')

plt.show()