**Project Report and Documentation**

**Artistic Style Transfer Web Application**

**Introduction**

This project aims to create a web application that performs artistic style transfer on images using a pre-trained model. The application allows users to upload a content image and a style image, then applies the artistic style to the content image and displays the result. The web application is built using Flask and TensorFlow, with the style transfer model sourced from TensorFlow Hub.

**Components**

1. Flask Application (app.py)

2. HTML Template (index.html)

**Code Overview**

**Flask Application (app.py)**

from flask import Flask, request, render\_template, send\_from\_directory, redirect, url\_for

import os

import tensorflow as tf

import tensorflow\_hub as hub

import numpy as np

import PIL.Image

app = Flask(\_\_name\_\_)

app.config['UPLOAD\_FOLDER'] = 'uploads/'

app.config['CONTENT\_FOLDER'] = os.path.join(app.config['UPLOAD\_FOLDER'], 'content')

app.config['STYLE\_FOLDER'] = os.path.join(app.config['UPLOAD\_FOLDER'], 'style')

app.config['RESULT\_FOLDER'] = 'static/images/'

os.makedirs(app.config['CONTENT\_FOLDER'], exist\_ok=True)

os.makedirs(app.config['STYLE\_FOLDER'], exist\_ok=True)

os.makedirs(app.config['RESULT\_FOLDER'], exist\_ok=True)

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

max\_dim = 512

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

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

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

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

long\_dim = max(shape)

scale = max\_dim / long\_dim

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

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

img = img[tf.newaxis, :]

return img

def tensor\_to\_image(tensor):

tensor = tensor 255

tensor = np.array(tensor, dtype=np.uint8)

if np.ndim(tensor) > 3:

assert tensor.shape[0] == 1

tensor = tensor[0]

return PIL.Image.fromarray(tensor)

def perform\_style\_transfer(content\_path, style\_path, result\_path):

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

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

hub\_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')

stylized\_image = hub\_model(tf.constant(content\_image), tf.constant(style\_image))[0]

result\_image = tensor\_to\_image(stylized\_image)

result\_image.save(result\_path)

@app.route('/', methods=['GET', 'POST'])

def index():

if request.method == 'POST':

content\_file = request.files['content\_image']

style\_file = request.files['style\_image']

if content\_file and style\_file:

content\_path = os.path.join(app.config['CONTENT\_FOLDER'], content\_file.filename)

style\_path = os.path.join(app.config['STYLE\_FOLDER'], style\_file.filename)

content\_file.save(content\_path)

style\_file.save(style\_path)

result\_path = os.path.join(app.config['RESULT\_FOLDER'], 'stylized\_image.jpg')

perform\_style\_transfer(content\_path, style\_path, result\_path)

return redirect(url\_for('result'))

return render\_template('index.html')

@app.route('/result')

def result():

return render\_template('index.html', image\_url=url\_for('static', filename='images/stylized\_image.jpg'))

@app.route('/download')

def download():

return send\_from\_directory(app.config['RESULT\_FOLDER'], 'stylized\_image.jpg', as\_attachment=True)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**HTML Template (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Style Transfer</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: 1e1e1e;

color: ffffff;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

text-align: center;

}

h1 {

margin-bottom: 20px;

}

.upload-section {

display: flex;

justify-content: center;

margin-bottom: 20px;

}

.upload-box {

border: 2px dashed ccc;

border-radius: 10px;

padding: 20px;

margin: 0 10px;

width: 300px;

height: 200px;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

background-color: 2c2c2c;

position: relative;

cursor: pointer;

overflow: hidden;

}

.upload-box.dragover {

border-color: 4CAF50;

}

.upload-box input[type="file"] {

position: absolute;

width: 100%;

height: 100%;

opacity: 0;

cursor: pointer;

}

.upload-box img {

max-width: 50px;

margin-bottom: 10px;

}

.upload-box span {

margin-top: 10px;

}

.upload-box img.preview {

max-width: 100%;

max-height: 100%;

display: none;

object-fit: cover;

border-radius: 10px;

}

.upload-box label {

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

height: 100%;

width: 100%;

}

.upload-box .icon-center {

margin-bottom: 10px;

}

.button {

background-color: 4CAF50;

border: none;

color: white;

padding: 15px 32px;

text-align: center;

text-decoration: none;

display: inline-block;

font-size: 16px;

margin-top: 20px;

cursor: pointer;

border-radius: 5px;

}

.result-section {

margin-top: 20px;

}

.result-section img {

max-width: 100%;

border-radius: 10px;

}

</style>

</head>

<body>

<div class="container">

<h1>Style Transfer</h1>

<form action="/" method="post" enctype="multipart/form-data">

<div class="upload-section">

<div class="upload-box" id="content-upload-box">

<label for="content\_image">

<img src="{{ url\_for('static', filename='images/upload\_icon.png') }}" alt="Upload Icon" class="icon-center" id="content\_icon">

<span id="content\_text">Upload original image</span>

<input type="file" id="content\_image" name="content\_image" required onchange="previewImage(this, 'content\_preview', 'content\_icon', 'content\_text')">

<img id="content\_preview" class="preview">

</label>

</div>

<div class="upload-box" id="style-upload-box">

<label for="style\_image">

<img src="{{ url\_for('static', filename='images/upload\_icon.png') }}" alt="Upload Icon" class="icon-center" id="style\_icon">

<span id="style\_text">Upload style image</span>

<input type="file" id="style\_image" name="style\_image" required onchange="previewImage(this, 'style\_preview', 'style\_icon', 'style\_text')">

<img id="style\_preview" class="preview">

</label>

</div>

</div>

<button type="submit" class="button">Transfer Style</button>

</form>

{% if image\_url %}

<div class="result-section" id="result-section">

<h2>Result</h2>

<img src="{{ image\_url }}" alt="Stylized Image">

<br>

<a href="{{ url\_for('download') }}" class="button">Download Image</a>

</div>

{% endif %}

</div>

<script>

function previewImage(input, previewId,iconId, textId) {

var preview = document.getElementById(previewId);

var icon = document.getElementById(iconId);

var text = document.getElementById(textId);

var file = input.files[0];

var reader = new FileReader();

reader.onload = function(e) {

preview.src = e.target.result;

preview.style.display = 'block';

icon.style.display = 'none';

text.style.display = 'none';

}

reader.readAsDataURL(file);

}

function handleDragOver(e) {

e.preventDefault();

e.stopPropagation();

e.target.classList.add('dragover');

}

function handleDragLeave(e) {

e.preventDefault();

e.stopPropagation();

e.target.classList.remove('dragover');

}

function handleDrop(e, inputId, previewId, iconId, textId) {

e.preventDefault();

e.stopPropagation();

e.target.classList.remove('dragover');

var input = document.getElementById(inputId);

var file = e.dataTransfer.files[0];

input.files = e.dataTransfer.files;

var reader = new FileReader();

reader.onload = function(e) {

var preview = document.getElementById(previewId);

var icon = document.getElementById(iconId);

var text = document.getElementById(textId);

preview.src = e.target.result;

preview.style.display = 'block';

icon.style.display = 'none';

text.style.display = 'none';

}

reader.readAsDataURL(file);

}

document.getElementById('content-upload-box').addEventListener('dragover', handleDragOver);

document.getElementById('content-upload-box').addEventListener('dragleave', handleDragLeave);

document.getElementById('content-upload-box').addEventListener('drop', function(e) {

handleDrop(e, 'content\_image', 'content\_preview', 'content\_icon', 'content\_text');

});

document.getElementById('style-upload-box').addEventListener('dragover', handleDragOver);

document.getElementById('style-upload-box').addEventListener('dragleave', handleDragLeave);

document.getElementById('style-upload-box').addEventListener('drop', function(e) {

handleDrop(e, 'style\_image', 'style\_preview','style\_icon', 'style\_text');

});

window.onload = function() {

var resultSection = document.getElementById('result-section');

if (resultSection && !resultSection.querySelector('img')) {

resultSection.style.display = 'none';

}

};

</script>

</body>

</html>

**Dataset**

The project does not use a specific dataset. Instead, users provide their own content and style images through the web interface. The style transfer model used is a pre-trained TensorFlow Hub model.

**Algorithm and Process**

1. Image Upload: Users upload a content image and a style image through the web interface.

2. Image Preprocessing: The uploaded images are resized and normalized.

3. Style Transfer: The pre-trained model from TensorFlow Hub performs the style transfer, combining the content of the first image with the artistic style of the second.

4. Display Result: The resulting stylized image is displayed on the web page, and users have the option to download it.

**Function Descriptions**

- load\_img(path\_to\_img): Reads and preprocesses an image from the given path.

- tensor\_to\_image(tensor): Converts a TensorFlow tensor into a PIL image.

- perform\_style\_transfer(content\_path, style\_path, result\_path): Performs style transfer using a pre-trained model and saves the result.

**Running the Application**

1. Ensure Python and Flask are installed.

2. Install TensorFlow and TensorFlow Hub.

pip install tensorflow tensorflow-hub

3. Run the Flask application:

python app.py

4. Access the application by navigating to `http://127.0.0.1:5000/` in your web browser.

**Prototype Results**

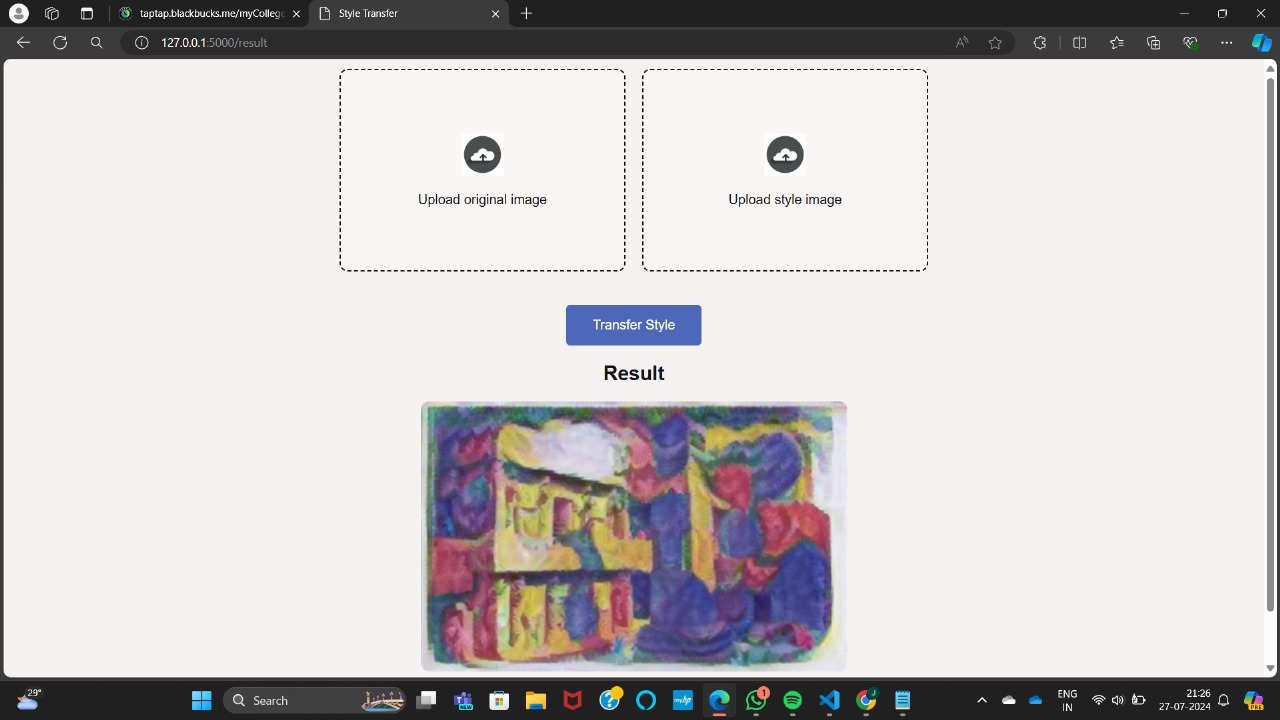
Users can see the result of the style transfer on the same page after uploading their images and submitting the form. The stylized image is displayed along with a download button.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



**Conclusion**

This project demonstrates the use of pre-trained deep learning models to perform artistic style transfer on images within a web application. It allows users to combine their own images with various artistic styles easily and interactively. The integration of TensorFlow Hub simplifies the use of complex models, making advanced image processing accessible through a simple web interface.