IMAGE STYLIZER APPLICATION REPORT

marouane NILI

[NOM DE LA SOCIETE] [Adresse de la société]

Table des matières

1 Introduction 2
2 Architecture and technology stack 3
3 Code and explanation: 4
3.1 Index.php 4
3.2 Process.php 6
3.3 Process.py 7
4 Conclusion 9

Image stylizer Php page

1 Introduction

This report describes a web application that allows users to upload an image and apply a pre-trained style transfer model from Tensorflow-Hub. The application is developed using two PHP files and a Python script. The first PHP file acts as an index page where users can select an image and submit it for processing. The second PHP file, the "process.php" file, handles the processing of the uploaded image using the Python script.

The Python script utilizes the pre-trained Tensorflow-Hub model to stylize the image and returns the result as a new image file in the directory. Finally, the updated image is displayed on the right-side of the index page using Ajax.

This report will provide a detailed overview of the architecture, technology stacks, and code involved in the development of this application.

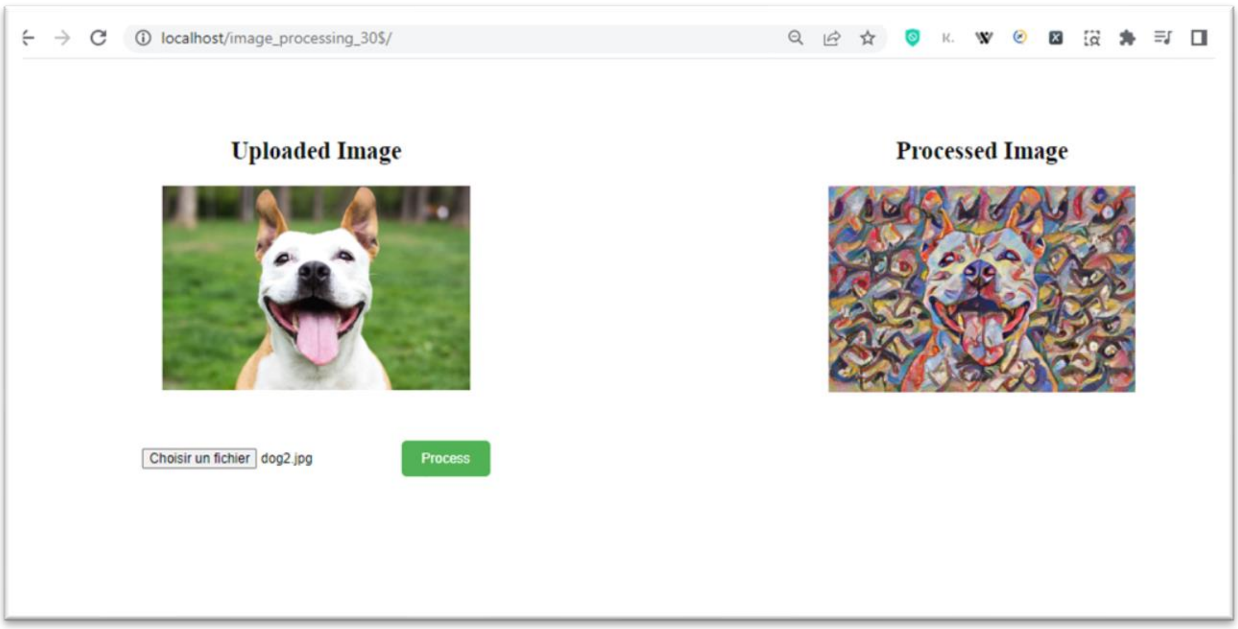


Figure 1 index.php after processing the image

2 Architecture and technology stack

The application consists of two PHP files, an index file and a process file, and one Python file. The index file contains a form that allows the user to upload an image, which is then processed by the process file.

The process file downloads the uploaded image to a local directory called images, then send the new directory of the image to the Python file, which uses a pre-trained TensorFlow-Hub model to stylize the image.

The model used in this application is 'https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2'. The output of the Python file is a stylized version of the original image, which is then stored as 'stylized-image.png' in the same directory as the PHP files.

Finally, the index file refreshes the image displayed on the right side of the page with the newly generated stylized image.

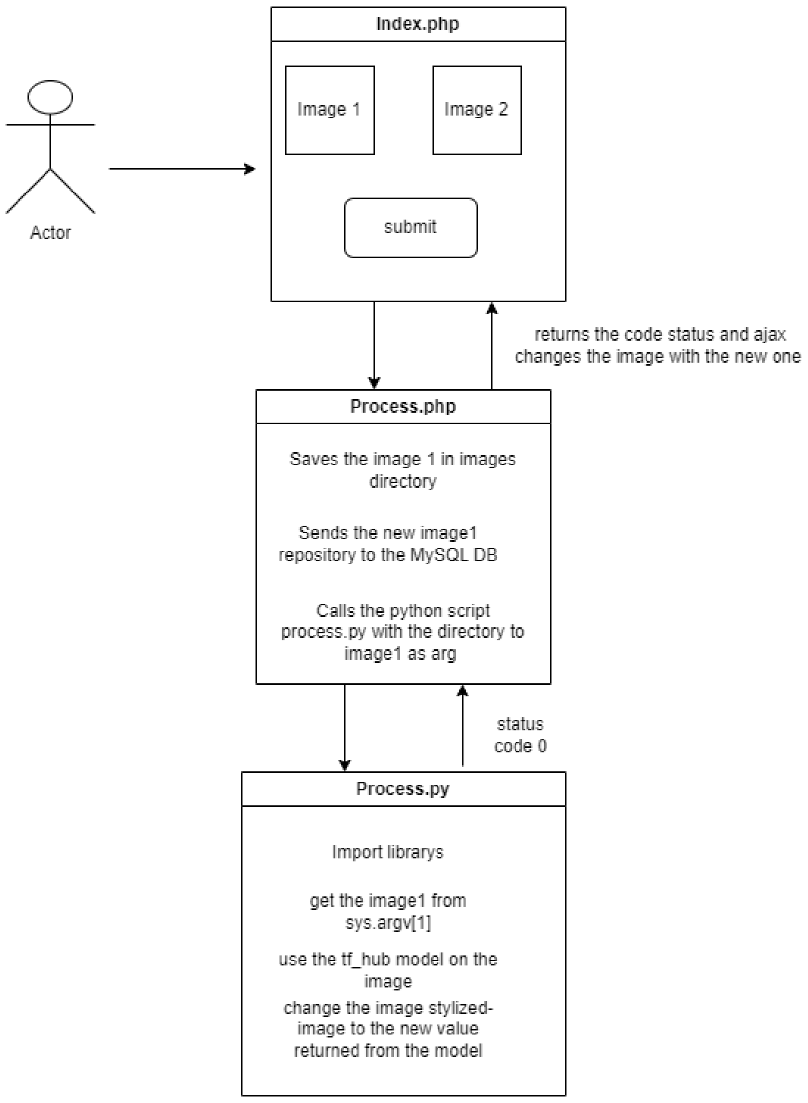
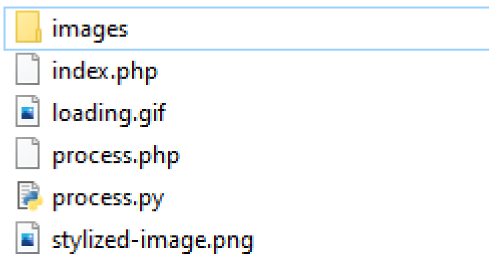

Index.php
Image 1
Image 2
submit
Actor
returns the code status and ajax changes the image with the new one
Process.php
Saves the image 1 in images directory
Sends the new image1 repository to the MySQL DB
Calls the python script process.py with the directory to image1 as arg
status code 0
Process.py
Import librarys
get the image1 from sys.argv[1]
use the tf_hub model on the image
change the image stylized-image to the new value returned from the model

Figure 2 architecture of the application



images
index.php
loading.gif
process.php
process.py
stylized-image.png

Figure 3 the files inside the folder

The technology stack used in this application includes:

Ajax for displaying the stylized-image.png in the index every when the response is returned.

PHP for the index and process files

Python for the image processing using TensorFlow-Hub

MySQL for storing the directories of the uploaded images.

3 Code and explanation:

3.1 Index.php

structure: The file contains the HTML structure of the user interface, including the form to upload an image and two image elements to display the original and stylized images.

CSS: It includes styles to style the form and images to make the page look appealing.

```
<style>
    /* styles for the spinner */
    .loading {
        display: none;
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
    }

    .left-side {
        float: left;
        width: 40%;
        text-align: center;
        padding: 50px;
    }

    .right-side {
        float: right;
        width: 40%;
        text-align: center;
        padding: 50px;
    }
    img {
        width: 60%;
        height: auto;
    }

    form {
        text-align: center;
        margin-top: 50px;
    }

    input[type="submit"] {
        padding: 10px 20px;
        background-color: #4CAF50;
        color: white;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
</style>
```

JavaScript: A small piece of JavaScript code is added to update the src attribute of the #original-image element with the uploaded image, we add new Date().getTime() to the name of the processed image so that the browser reload the new image againe and don't take it directly from cach since the old and the new image have the same name.

```
<script>
    document.querySelector("form").addEventListener("submit",
function(event) {
        event.preventDefault();
        //show spinner
        document.querySelector(".loading").style.display = "block";
        const formData = new FormData();
        formData.append("image", document.querySelector("#image-
input").files[0]);
        const xhr = new XMLHttpRequest();
        xhr.open("POST", "process.php", true);
        xhr.onload = function() {
            document.querySelector(".loading").style.display = "none";
            if (this.status === 200) {

                document.querySelector("#processed-image").src =
this.responseText+ new Date().getTime();
            } else {
                console.error("Image processing failed");
            }
        };
        xhr.send(formData);
    });

    document.querySelector("#image-input").addEventListener("change",
function() {
        const reader = new FileReader();
        reader.onload = function(event) {
            document.querySelector("#original-image").src =
event.target.result;
        };
        reader.readAsDataURL(this.files[0]);
    });
</script>
```

Form: The form is created using the form tag and it contains an input field for file upload and a submit button. The form is set to submit data to the process.php file using the POST method.

```
<form action="" method="post" enctype="multipart/form-data">
    <input type="file" name="image" id="image-input">
    <input type="submit" value="Process">
</form>
```

// action is left empty so that the page doesn’t change

3.2 Process.php

```
<?php
if ($_FILES['image']['error'] == 0) {
    $image = $_FILES['image']['tmp_name'];
    $destination = 'images/' . $_FILES['image']['name'];
    if (move_uploaded_file($image, $destination)){
    exec("python3 process.py $destination", $output, $status);}

    echo 'stylized-image.png?';
}
?>
```

Image Upload: The code checks if an image has been uploaded using the $_FILES array. If an image has been uploaded, it saves the temporary location of the image to a variable.

Calling the Python Script: The code uses the exec function to call the Python script and pass the temporary location of the image as an argument. The output of the Python script is stored in the $output variable, and the exit status is stored in the $status variable.

Displaying the Stylized Image: The code then echoes an HTML img element with its src attribute set to the stylized-image.png file.

Saving directories of images into MySQL data base:

To save directories we opened a connection with a data base called mysql_dog_images, the we added the directori called $destination to the db using mysqli_query() if the file is successfully saved in the folder called images

```
$servername = "localhost";
$username = "root";
$password = "root";
$dbname = "mysql_dogs_images";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

if ($_FILES['image']['error'] == 0) {
    $image = $_FILES['image']['tmp_name'];
    $destination = 'images/' . $_FILES['image']['name'];
    if (move_uploaded_file($image, $destination)){
        // Get the directory of the uploaded image
        $image_directory = $destination;

        // Insert the directory of the uploaded image into the database
        $sql = "INSERT INTO dogs (directori) VALUES ('$image_directory')";

        if (mysqli_query($conn, $sql)) {
            echo "Image directory inserted into the database successfully";
        } else {
            echo "Error inserting the image directory into the database: "
. mysqli_error($conn);
        }

        exec("python3 process.py $destination", $output, $status);
        echo "The image has been stylized!";
    } else {
        echo "Error uploading the image to the images directory";
    }
}

mysqli_close($conn);
```

In summary, the index.php file provides the user interface to upload an image, and the process.php file processes the uploaded image by calling the Python script and displaying the stylized image.

3.3 Process.py

This is the python code, it uses the TensorFlow and TensorFlow Hub libraries to perform image style transfer. The code does the following steps:

Imports required libraries: TensorFlow, TensorFlow Hub, Numpy, Matplotlib, and PIL:

```
import tensorflow_hub as hub

import tensorflow as tf
import sys
import matplotlib.pyplot as plt
import matplotlib as mpl


import numpy as np
import PIL.Image
mpl.rcParams['figure.figsize'] = (12, 12)
mpl.rcParams['axes.grid'] = False
```

Defines two functions: "tensor_to_image" and "load_img". The first function, "tensor_to_image", takes a tensor and returns a PIL image. The second function, "load_img", takes the path to an image and returns a TensorFlow tensor representing the image.

```
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 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
```

Loads the input content image and style image. The content image is passed in as a command-line argument, while the style image is hard-coded to be a pre-trained image.

```
content_path = sys.argv[1]
style_path =
tf.keras.utils.get_file('kandinsky5.jpg','https://storage.googleapis.com/do
wnload.tensorflow.org/example_images/Vassily_Kandinsky%2C_1913_-
_Composition_7.jpg')
```

Loads the style transfer model from TensorFlow Hub.

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

Runs the style transfer on the content and style images by calling the model with the two images as inputs.

```
content_image = load_img(content_path)
style_image = load_img(style_path)
stylized_image = hub_model(tf.constant(content_image),
tf.constant(style_image))[0]
```

Saves the stylized image to a file named "stylized-image.png".

```
file_name = 'stylized-image.png'
tensor_to_image(stylized_image).save(file_name)
print(file_name)
sys.stdout.flush()
```

the sys.stdout.flush() flushs all the printed strings into the output of the script in process.php

4 Conclusion

The application was completed successfully,

To see a demo of the application, check the video at: https://youtu.be/mDvBwdmzkjI