Project design proposal

Purpose

Given a user description of an image containing “content” and “style”, automatically find similar content and style images and apply neural style transfer (NST) to “generate” an image.

Example:

User input: “A dog in pencil sketch style”

1. Embed the user input as a vector
2. Compare this vector to the image description vectors in the postgresql database (using cosine similarity) and return the most similar **content** and **style** images.
3. Generate NST images based on content and style images.

Libraries

1. Torch
2. PostgreSQL-OCaml
3. ~~Aws-s3~~ can do this outside of OCaml
4. Rescript-MUI/ReScript
5. Dream
6. Cohttp

Module type declarations

https://github.com/jmiran15/fpse-project/tree/main/src

database.mli

```
(* This module handles interactions with the PostgreSQL database for image
data management. *)

(** Loads images with their metadata from S3 and returns them as a list. *)
val load_images_from_s3 : unit -> Image.t list

(** Generates a vector embedding for a given description string. *)
val generate_embedding : string -> Embedding.t

(** Loads images with metadata, generates embeddings, and inserts them into
```

```
the database. *)
val load_and_embed_images : unit -> unit

(** Finds the image with the most similar metadata to a given description.
*)
val find_similar_image : string -> Image.t option

module type DATABASE_CONFIG = sig
 val connection_path : string
 val username : string
 val password : string
 val port : int
end

module MakeDatabase (Config : DATABASE_CONFIG) : sig
 (** Clears all rows in the database. *)
 val clear : unit -> unit

 (** Initializes the database table. *)
 val init : unit -> unit

 (** Inserts a row into the database. *)
 val insert : Image.t -> unit

 (** Deletes a row from the database. *)
 val delete : int -> unit  (* Assuming 'int' is the type of the id. *)

 (** Updates a row in the database. *)
 val put : int -> Image.t -> unit

end
```

embedding.mli

```
(* Embedding module: Represents vector embeddings for image descriptions.
*)

(** Type representing a vector embedding. *)
type t

(** Creates an embedding from a list of floats. *)
val of_list : float list -> t
```

```
(** Converts an embedding to a list of floats. *)
val to_list : t -> float list

(** Calculates the cosine similarity between two embeddings. *)
val cosine_similarity : t -> t -> float
```

image.mli

```
(* Image module: Represents images and their associated metadata. *)

type t

(** Type representing image metadata. *)
type metadata = {
 url: string;          (* URL to the image stored in S3 *)
 author: string;       (* Author of the image *)
 description: string;  (* Description of the image *)
 embedding: Embedding.t; (* Vector embedding of the image description *)
}

(** Creates an image with the given metadata. *)
val create : metadata -> t

(** Retrieves the metadata of an image. *)
val get_metadata : t -> metadata
```

neural_style_transfer.mli

```
(* Module for performing neural style transfer on images. *)

(** Weight assigned to the style in the final image. *)
val style_weight : float

(** Learning rate for the optimization process. *)
val learning_rate : float

(** Total number of steps for the style transfer process. *)
val total_steps : int

(** Layer indexes to extract style features. *)
val style_indexes : int list
```

```
(** Layer indexes to extract content features. *)
val content_indexes : int list

(** Applies the neural style transfer algorithm on a content image using a
style image. *)
val apply_style_transfer : style_img:string -> content_img:string ->
filename:string -> unit
```

server.mli

```
(* This module defines the server routes and their corresponding
functionalities. *)

(** Route to POST "/images". Takes a description and returns a generated
image. *)
val route_generate_image : string -> Image.t
```

Implementation order

1. Implement NST and test with content/style images locally (testing torch)
2. Implement a simple backend route to return semantically similar sentences (testing postgresql-ocaml and dream)
    a. Input: sentence, Output: most semantically similar sentence in db
3. Implement seeding for content and style datasets
4. Package (combine) as command line program
5. Create ReScript-MUI frontend

Application Mock-up

https://docs.google.com/presentation/d/1ybS1zaWT0qgI14k_JIM_N-Nh1CwiI26rw1kWSxm0PYE/edit#slide=id.p

Inspiration

- https://blog.janestreet.com/deep-learning-experiments-in-ocaml/
- https://github.com/janestreet/torch/tree/master/examples/neural_transfer
- https://pytorch.org/tutorials/advanced/neural_style_tutorial.html

Flow

Pre-use:

Seed database

1. **Load** images along with metadata from the source
2. **Embed** image descriptions using OpenAI embeddings (Cohttp for HTTP requests)
3. **Upload images** to an S3 bucket
4. **Push images** with metadata and vector embeddings to the database

Usage:

1. The user enters the image description into the input box in the UI.
    a. The user can also select an “example” sentence.
    b. The user can also “regenerate” their image.
2. Embed the description with the OpenAI embeddings.
3. Query the database to find the most similar “content” image (using cosine similarity).
4. Query the database to find the most similar “style” image.
5. Run neural style transfer (NST) on content and style images and return the final image to the user.