Text to Motion Database

Design Document

Brendan Duke
Andrew Kohnen
Udip Patel
David Pitkanen
Jordan Viveiros

January 10, 2017

Contents

1 User Experience **1**
- 1.1 User Journey . . . 1
- 1.2 Home Page . . . 1
- 1.3 About . . . 1
- 1.4 Contact . . . 1
- 1.5 Sign Up . . . 1
- 1.6 Login . . . 1
- 1.7 Navigation Bar Links . . . 1
- 1.8 Text to Motion . . . 1
  - 1.8.1 Search . . . 1
  - 1.8.2 Search Results . . . 1
- 1.9 Image Pose Draw . . . 1
  - 1.9.1 Create . . . 1
  - 1.9.2 Edit/Details . . . 1
  - 1.9.3 View Uploads . . . 1

2 Database Structure **2**
- 2.1 Database Schema . . . 2
- 2.2 Table Description . . . 2

3 Module Decomposition **3**
- 3.1 Text To Motion - ASP.NET Application . . . 3
  - 3.1.1 Overview . . . 3
  - 3.1.2 Dependency Injection . . . 3
  - 3.1.3 Models . . . 3
  - 3.1.4 HomeController . . . 3
  - 3.1.5 AccountController . . . 3
  - 3.1.6 ManageController . . . 3
  - 3.1.7 ImagePoseDrawController . . . 3
  - 3.1.8 TextToMotionController . . . 4
- 3.2 Flowing Convnets - Human Pose Estimation . . . 5
  - 3.2.1 Overview . . . 5
  - 3.2.2 Shared Object File . . . 5
  - 3.2.3 Pose Estimation C Program (estimate_pose_wrapper.c) . . . 5
  - 3.2.4 Pose Estimation C++ Program (estimate_pose.cpp) . . . 6

3.3 Features In Development 10
3.3.1 Video Estimation C++ Program 10
3.3.2 Tensorflow 10
3.3.3 Standalone HTTP Server 10

4 Communication Protocol **11**

5 Development Details **12**
5.1 Languages 12
5.2 Software 12
5.3 Hardware 12

Revision History

Date	Version	Notes
January 5, 2017	0.0	File created

1 User Experience

1.1 User Journey

1.2 Home Page

1.3 About

1.4 Contact

1.5 Sign Up

1.6 Login

1.7 Navigation Bar Links

1.8 Text to Motion

1.8.1 Search

1.8.2 Search Results

1.9 Image Pose Draw

1.9.1 Create

1.9.2 Edit/Details

1.9.3 View Uploads

2 Database Structure

2.1 Database Schema

2.2 Table Description

3 Module Decomposition

3.1 Text To Motion - ASP.NET Application

3.1.1 Overview

...might be a good place to mention .net MVC. Mention of how sections will be about Models, Controllers (referencing views)...

3.1.2 Dependency Injection

...describe anything important that is in project.json...

3.1.3 Models

- **U**ser

  ...

- **...**

3.1.4 HomeController

- **F**unctionName:

  ...description of func...

  ...reference any models used...

  ...reference the view that is returned...

3.1.5 AccountController

...

3.1.6 ManageController

...

3.1.7 ImagePoseDrawController

...

3.1.8 TextToMotionController

...

3.2 Flowing Convnets - Human Pose Estimation

3.2.1 Overview

This component of the project actually renders the skeleton overlay onto an image submitted to the website.
Mapping out the joints of a person in an image requires the use of image manipulation and deep learning libraries. As of now, this proces is based on a research paper and is implemented with **Caffe** and **OpenCV** in **C++**. (*The parameters for functions given below will reference 'caffe' and 'cv' types in c++)

3.2.2 Shared Object File

The website is able to take an uploaded image and process it by using a shared object file (.so). The web app can make function calls to functions in the shared object file and pass in images as the parameters.

The C++ file **"estimate_pose.cpp"** contains all of the functions that interface with Caffe and OpenCV. The C file **"estimate_pose_wrapper.c"** is used to wrap the C++ function in C, and create the shared object file so that the C++ function can be accessed from the website.

3.2.3 Pose Estimation C Program (estimate_pose_wrapper.c)

The file **"estimate_pose_wrapper.c"** just contains 1 function. This function references a C++ function in **estimate_pose.cpp**

function: int32_t estimate_pose_wrapper(args)

- **Expected Arguments:**

  void **image*
  uint32_t **size_bytes*
  uint32_t *max_size_Bytes*

- **Returns:**

  if *image* is processed and saved, returns:
  int32_t size (describing size of file that was uploaded)

  else returns error object

- **Description:**

  this function makes a direct call to the C++ function **"estimate_pose_from_c"** in **"estimate_pose.cpp"** and simply returns the result of that C++ function call. The C++ function takes in the same args as this function

3.2.4 Pose Estimation C++ Program (estimate_pose.cpp)

This C++ Program file contains 8 functions. All of these functions take in objects from the 'openCV'(cv) and 'Caffe'(caffe) libraries as arguments

The key function in this file is **estimate_pose_from_c**, and most of the functions serve as helpers to this function

List of Functions and Descriptions (PE = Pose Estimation):

References for Objects used in C++ functions

- **References for OpenCV objects**

  Reference to cv::Mat
  Reference to cv::Point
  Reference to cv::InputArray

- **References for Caffe Objects**

  Reference to caffe::Blob
  Reference to caffe::Net

- **References for Other Objects**

  Reference to boost::shared_ptr

PE function 1: void channels_from_blob(args)

- **Expected Arguments:**

  std::vector<cv::Mat> *channels*
  boost::shared_ptr<caffe::Blob> *blob*
  int32_t *width*
  int32_t *height*

- **Returns:**

  void (saves data into *channels*)

- **Description:**

  The *blob* object contains concatenated mulit-channel data
  The *channels* object is empty to begin with
  This function converts the raw data in a Caffe blob into a 'container of channels' (vector of openCV matrices)
  The *width* and *heigth* parameters let the program know what the dimensions of the channels are in the *blob*
  The extracted information from *blobs* is saved into the *channels* vector

This function is just used as a helper for other functions

PE function 2: void copy_image_to_input_blob(args)

- **Expected Arguments:**

  caffe::Net <float> *heatmap_net*
  cv::Mat *image*

- **Returns:**

  void (saves data into *heatmap_net*)

- **Description:**

  This function converts the *image* object from OpenCV BGR format to 32-bit-floating point RGB format and copies the image to the input blob of *heatmap_net*. It lso divides the input layer of the *heatmap_net* from a multi-channel array into several single-channel arrays by calling a helper function
  image is the image that will serve as the input layer to the caffe network
  heatmap_net is the caffe network that will get its input layer filled with the RGB pixel data from *image*

  This function makes a call to **PE-cpp function 1** when it splits up the newly updated image in *heatmap_net*'s input layer into several 'input_channels'

PE function 3: void get_joints_from_network(args)

- **Expected Arguments:**

  cv::Point **joints*
  cv::Size *channel_size*
  caffe::Net<float> *heatmap_net*

- **Returns:**

  void (saves data into *joints*)

- **Description:**

  This function uses the *heatmap_net*'s "conv5_fusion" layer to get a set of joint locations for that heatmap.
  The joint locations get saved into **joints*
  The *channel_size* is used to maintain the accuracy of the position of the joints relative to the image as the image matrix is resized multiple times

  This function makes a call to **PE-cpp function 1** when it uses the *heatmap_net* to save all of the joint locations in a 'joint_channel' vector of cv::Mat objects

PE function 4: void draw_skeleton(args)

- **Expected Arguments:**

  cv::Mat *image*
  cv::Point **joints*

- **Returns:**

  void (saves image data into *image*)

- **Description:**

  This function uses the joint_locations described in **joints* to draw an upper-body skeleton on the *image* matrix passed in.
  image is the image to draw the skeleton overlay on
  **joints* contain the locations for the set of joints (wrists, elbows, shoulders and head)

PE function 5:
std::unique_ptr<caffe::Net<float>>init_pose_estimator_network(args)

- **Expected Arguments:**

  std::string *model*
  std::string *trained_weights*

- **Returns:**

  std::unique_ptr<caffe::Net<float>> heatmap_net
  pointer to an object that represents a whole caffe network

- **Description:**

  This function creates a Caffe network and copies over the trained layers from a given option for *trained_weights*
  For this application, a caffe network is initialized with the default settings:
  (*model* = 'MODEL_DEFAULT', *trained_weights* = TRAINED_WEIGHTS_DEFAULT)

PE function 6: void image_pose_overlay(args)

- **Expected Arguments:**

  caffe::Net<float> *heatmap_net*
  cv::Mat *image*

- **Returns:**

  void (saves to *image*)

- **Description:**

This function processes the *image* passed in using the *heatmap_net* to draw a skeleton on the openCV Matrix

Details on the actions taken by the function:

1. Resizes *image* to 256x256
2. calls **PE function 2** to copy the image into the input layer of the *heatmap_net*
3. after allowing the network to extract some data, declares an array object of cv::Point called *joints*
4. calls **PE function 3** to load in joint locations into *joints*
5. converts image to a format so that it can be drawn on by the program
6. calls **PE function 4** to draw the skeleton overlay on top of the *image*

PE function 7: void square_image_with_borders(args)

- **Expected Arguments:**

  cv::Mat *image_mat*

- **Returns:**

  void (saves image data to *image_mat*)

- **Description:**

  If the image's dimensions do not fit a square (length != width), this function makes it so that the image dimensions are expanded so that it fits into a square
  This is to avoid distorting the image drastically when the image is resized to 256x256

  This is just a simple helper function to pre-process the image

PE function 8: int32_t estimate_pose_from_c(args)

- **Expected Arguments:**

  void **image*
  uint32_t **size_bytes*
  unit32_t *max_size_bytes*

- **Returns:**

  if *image* is processed and saved, returns:
  int32_t size (describing size of file that was uploaded)

  else returns error object

- **Description:**

  This function is the key method in this file. This function overwrites the contents of the memory allocated for **image* so that a given image is updated to show the skeleton overlay on that image

  This function creates a new Caffe network and calls a lot of helper functions needed to process the *image*

  Details on the actions taken by the function:

  1. calls **PE function 5** to create a new caffe Network, stores new network in *heatmap_net*
  2. uses **image* and **size_bytes* to create a cv::InputArray object to represent the uploaded image
  3. creates a cv::Mat image matrix, and decodes the contents of the cv::InputArray object into the newly created matrix object
  4. calls **PE function 7** to pre-process the image matrix to reaffirm that the image is square
  5. calls **PE function 6** using *heatmap_net* and the image matrix so that the image_matrix includes the skeleton overlay
  6. converts and compresses the image_matrix into a png
  7. finally, overwrites the contents of **image* with the newly created image that has the pose estimation 'skeleton overlay'

3.3 Features In Development

3.3.1 Video Estimation C++ Program

...

3.3.2 Tensorflow

..just mention to the possibility of using it...

3.3.3 Standalone HTTP Server

...

4 Communication Protocol

...just mention to HTTP...

5 Development Details

5.1 Languages

5.2 Software

5.3 Hardware