CSE461: Design Document

Automated Quiz Evaluation System

Team Number: 12

Team Members:

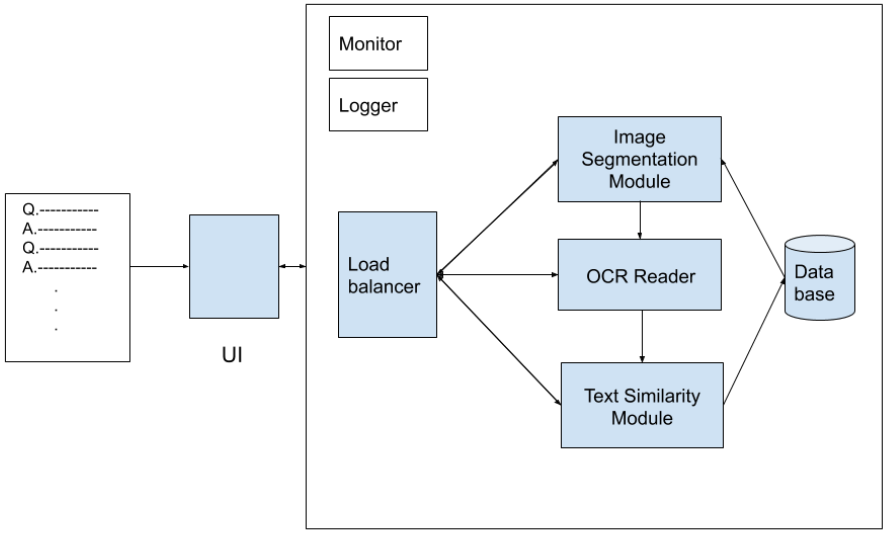
1. Zubair Abid - 20171076
2. Sravani Boinepelli - 20171050
3. Sriven Reddy Nookala - 20171081
4. C.Sai Sukrutha - 2018201054
5. Praffullitt Jain - 20171142
6. Kripa Anne Tharakan - 20171159

Link to our Project Abstract Doc can be found here:
https://github.com/sravaniboinepelli/SEProject/blob/master/ProjectAbstract.pdf

Overall Architecture

There would be two types of users: students and instructors. Instructors submit question-answer pairs (in plaintext). Students submit scans of their handwritten solutions. This solution will be fed through the system for automated processing and evaluation. To enable ease of horizontal scaling, each module shall be a containerised application with clearly defined input and output, so new containers can be fired up to handle greater load as required.

High-Level Architecture Diagram



Proposed Modules

UI

UI is the interface with which users can interact with the system and make requests. This is the only frontend exposed to the end-user, regardless of their category, and contains various options, forms, and tools usable by both categories of end-users.

As there are two types of users: Students and Instructors, different interfaces will be provided according to their type.

1. **Instructor UI**: The instructor will be provided with an interface to post questions, correct answers and check the marks of all students. They also get the option of choosing whether or not to automatically evaluate answers with the included semantic similarity module, or to manually evaluate them.
2. **Students UI**: Students UI will provide interfaces for Viewing Questions, Uploading Answers, and getting their scores. They submit to the system handwritten solutions to questions posted by the instructor that they can see on the system.

The UI is connected to all the components of the system and sends the request posted by the user to the appropriate module. It will get data from modules and provide the user with the required output.

Distributed Backend

Each module of the system described henceforth is entirely modular and containerised, and capable of being run independently. The system takes the user input and then fires up individual modules in their respective containers in order to set up the pipeline. These modules are needed for the automatic quiz correction part, and as such have no relation to other general webapp tasks.

The user (student) submits a scanned copy of their answer to the system. This scanned image is sent to the Image Segmentation Module in order to identify question-answer boundaries, and detect which question relates to which answer. Once identified, it returns the segments in a specified format - one that is readable by the OCR module coming right next in the pipeline. Taking the segmented handwritten question-answer pairs, it uses an OCR Engine like Tesseract to convert it into question-answer pairs in text format; more specifically, json. This is now

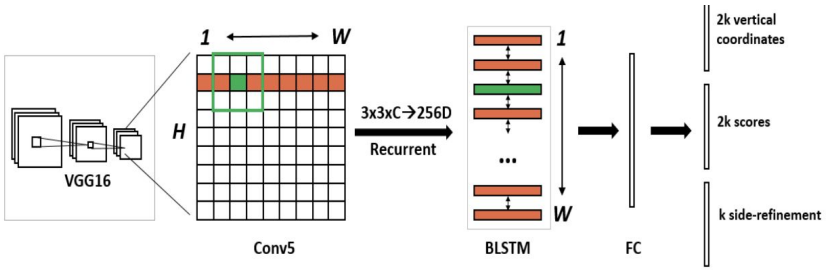
1. Either sent to the Semantic Text Similarity module for automated evaluation, or
2. Sent to the instructor's dashboard for manual evaluation,

according to the settings selected by the course instructor.

Due to the containerised nature of the modules, multiple can be fired up in parallel. This allows for horizontal scalability, and as such they are constrained only by the ability of the host machine to run however many containers it may. It can, in theory, also be extended to run a single module on a single machine as well, increasing the resources available to each.

Image Segmentation Module

Segmentation is the process of partitioning the scanned image into multiple segments (sets of pixels). The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (edge detection). Each of the pixels in a region is similar with respect to some characteristic properties, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).



CTPN Algorithm

1. Firstly the input image is passed through a pretrained VGG16 model (trained with ImageNet dataset).
2. Features output from the last convolutional maps of the VGG16 model is taken.
3. These outputs are passed through a 3×3 spatial window.
4. Then outputs after a 3×3 spatial window are passed through a 256-D bi-directional Recurrent Neural Network (RNN).
5. The recurrent output is then fed to a 512-D fully connected layer.
6. Now comes the output layer which consists of 3 different outputs, 2k vertical coordinates, 2k text/non-text scores and k side refinement values.

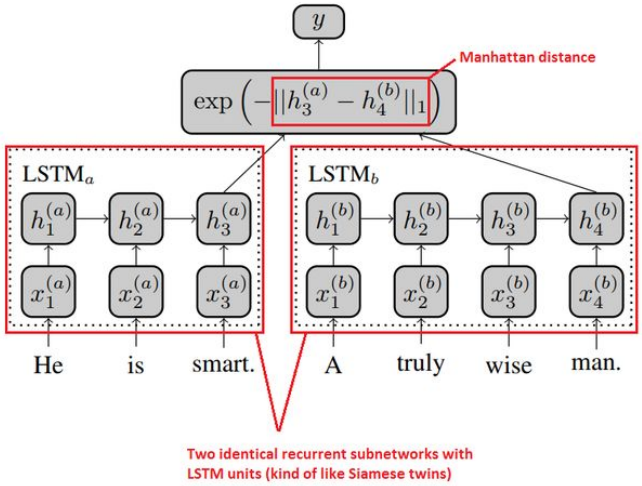
Output Layer

The first output consists of 2k vertical coordinates, where k is the number of anchor boxes. Every anchor box output contains its y coordinate for the center of the box and height of the box. These anchor boxes are fine-scale text proposals whose width is 16 pixels shown in the diagram.
The second outputs are 2k text/non-text scores. For each anchor box, the output layer also contains text/non-text scores. It includes one output for classification between foreground and background and another output is for the positive or negative anchor. The positive or negative anchor is being decided on the basis of the IOU overlap with the Ground Truth box.
The third outputs are k side-refinements. In CTPN we fix the width of fine-scale text proposal to 16 pixels but this can be problematic in some cases where some side text proposals are discarded due to low score. So in the output layer, it also predicts side refinement values for the x-axis.

Semantic Text Similarity Module

Text similarity has to determine how ‘close’ two pieces of text are both in surface closeness (lexical similarity) and meaning (semantic similarity). For instance, how similar are the

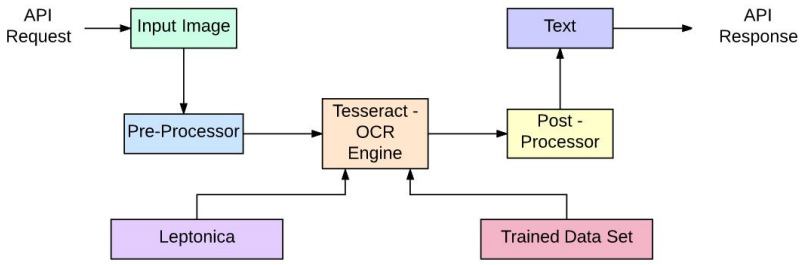
phrases “the cat ate the mouse” with “the mouse ate the cat food” by just looking at the words? A lot of work has been done in this field. We compared the current state of the art and have opted to use deep siamese LSTM networks using character embeddings for our project. Fine-tuned BERT models also performed well. Our BERT model modifies pytorch-transformers by abstracting away all the research benchmarking code for ease of real-world applicability but would need additional GPU resources. We shall experiment to see if these resources can be expended as well.
Siamese is the name of the general model architecture where the model consists of two identical subnetworks that compute some kind of representation vectors for two inputs and a distance measure is used to compute a score to estimate the similarity or difference of the inputs. In the attached figure, the LSTMa and LSTMb share parameters (weights) and have an identical structure.



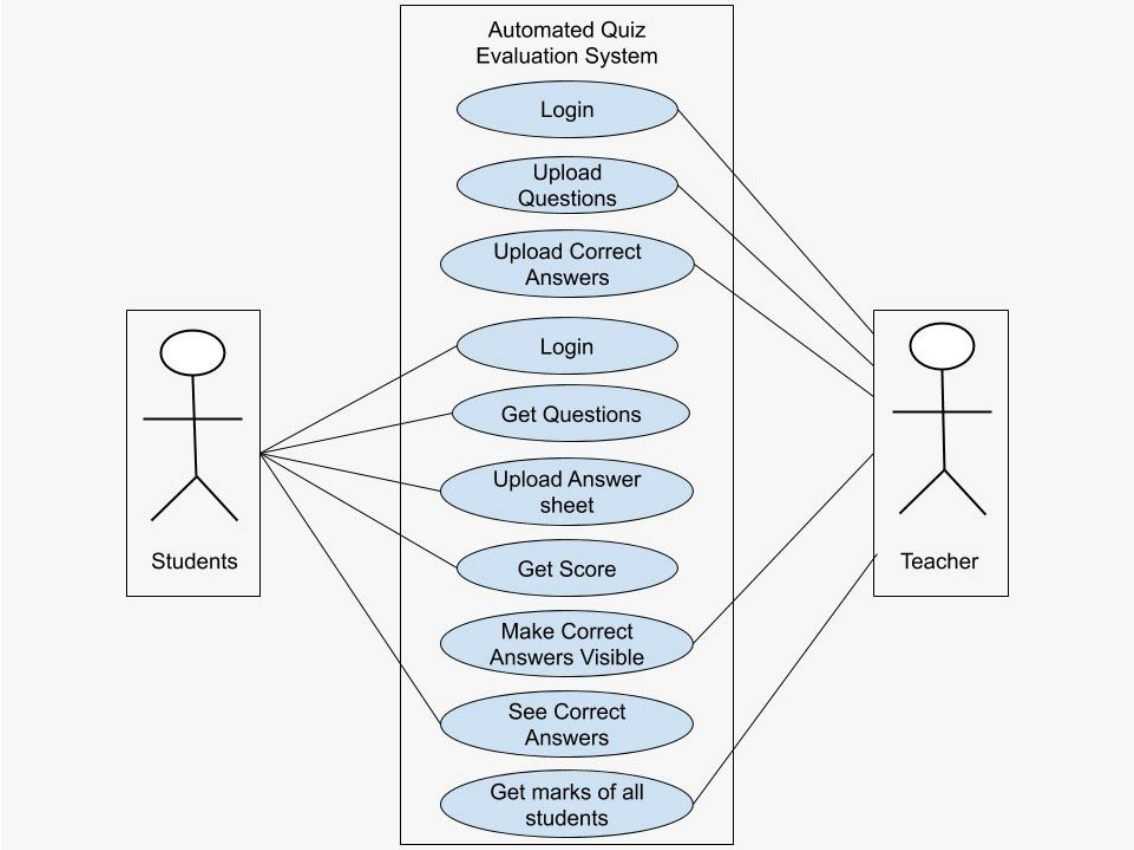
OCR Reader

Using an open source OCR (Optical Character Recognition) engine like Tesseract, we identify the characters from the segmented image. The engine is trained on various fonts to map the image pixels to actual text. For better accuracy and overall better working, we have to ensure that the image is as clear as possible and may have to perform several modifications to clean the image. The newest releases add a new neural net (LSTM) based OCR engine which is focused on line recognition, rather than legacy character pattern recognition.

OCR Process Flow



Use Case Diagram



Technologies Used

- UI
    - HTML
    - CSS
    - Javascript
- Distributed Backend
    - Flask Server
    - Docker
- Image Segmentation Module
    - CTPN implementation in Tensorflow or Keras
- OCR Reader
    - Tesseract
- Semantic Text Similarity Module:
    - numpy 1.11.0
    - tensorflow 1.2.1
    - gensim 1.0.1
    - nltk 3.2.2