Authors: Joshua Patterson & Alex Richardson

November 6, 2019

CMSI 401

SDD: Software Design Document (Architecture Section)

6.1. Introduction

This document serves as a detailed guide to understanding the functionality and design of our Software Project: *yung gan*. Our software provides a command line application to both generate music from pretrained models and to train a new model from a .wav or .mp3 music collection. Our application that will fabricate authentic-sounding hip hop beats through its unique implementation of a GAN, or Generative Adversarial Network. A vector of random noise is given as the only input to a deep network, known as the generator, where it funnels information through both recurrent and convolutional layers to create a computer generated beat in an attempt to fool another deep network, known as the discriminator. We will curate a playlist of real human-made beats from sources: Soundcloud, Youtube, and our own music. Those files will be preprocessed and stored in an SQL database along with its pertinent information. When training, those songs feed into the discriminator in batches that are combined with fake beats the generator has constructed. Over many epochs, the generator and discriminator loss should converge, indicating the training has finished. If our neural architecture is well constructed, the generator should produce beats that sound authentic to both the discriminator and to the human ear. Our system will include both a training command-line application written in Python and a generator command-line application written in Python that uses pre-trained generators included with the software to produce beats on quick command. A front-end web application is projected to be created for our command-line software. This will be done by using React.JS, CSS, and HTML. By having our command-line software run on through AWS GPU Server, an EC2 Virtual Machine Instance could be created to allow the python program to reach our front-end. Pointing Amazon API Gateway to a http endpoint on EC2 would allow this to happen.

6.1.1 System Objectives

The objectives of our system are to fulfill the task of replicating a hip-hop beat efficiently enough to our standards, and to continue to train/build on what we will have developed. As an application itself, an objective is to provide the user with a beat of their choosing (genre, sub-genre, bpm..) and to be able to interact with a front end that stores the command-line program's functionality.

6.1.2 Hardware, Software, and Human Interfaces

Hardware:

- Macbook
- Mac Computer
- Wifi
- For CPU:

  Minimum Requirements:
    - CPU: 2.4GHz dual core processor
    - RAM: 16 GB
    - Storage: 64 GB

  Suggested:
    - CPU: 2.4GHz quad core processor
    - GPU: CUA enabled NVIDIA
    - RAM: 64 GB
    - Storage: 128 GB

Software:

- Text Editor
  - Visual Studio Code Version 1.38.1
- OS
  - Mac OS Mojave Ver. 10.14.6
  - Windows OS
- Python Ver. 3.6 or above
- Keras Ver. 2.2.4
- TensorFlow Ver. 1.14
- Javascript ES6
- React.JS 16.8
- HTML5
- CSS3
- Other
  - Amazon Web Services

    - Terminal
    - Github
    - Google Docs
    - Soundcloud
    - Youtube
    - Soundcloud/Youtube Downloader

6.2 Architectural Design

The *yung gan* system is designed as a generative adversarial network, two deep learning networks minimizing loss adversarially. The networks are designed as a combination of recurrent and convolutional networks. The convolutional networks model the spectral element of music while recurrent networks model the time aspect. The discriminator runs through 2 recurrent layers, known as Long Short Term Memory, then through 8 convolutional blocks to determine song validity. The generator does essentially the reverse process by beginning with 100 random values and building that up through convolution then the LSTM layers.

6.2.1 Major Software Components

- Back-End
    - Generator
        - 2 LSTM layers
        - 10 Convolutional layers
    - Discriminator
        - 2 LSTM layers
        - 10 Convolutional layers
- Server
    - AWS GPU
        - Tensorflow Backend with CUDA
        - Keras
- Front End
    - React.JS

6.2.2 Major Software Interactions

- Back-End and Front-End Interactions
    - By having our command-line software run on through AWS GPU Server, an EC2 Virtual Machine Instance could be created to allow the python program to reach our front-end. Pointing Amazon API Gateway to a http endpoint on EC2 would allow this to happen.

- Front End
    - Front Page
        - On-Click Button
            - Downloads .mp3 file
        - Header
            - Explanation of software

6.2.3 Architectural Design Diagrams

Figure 1: Discriminator/Generator

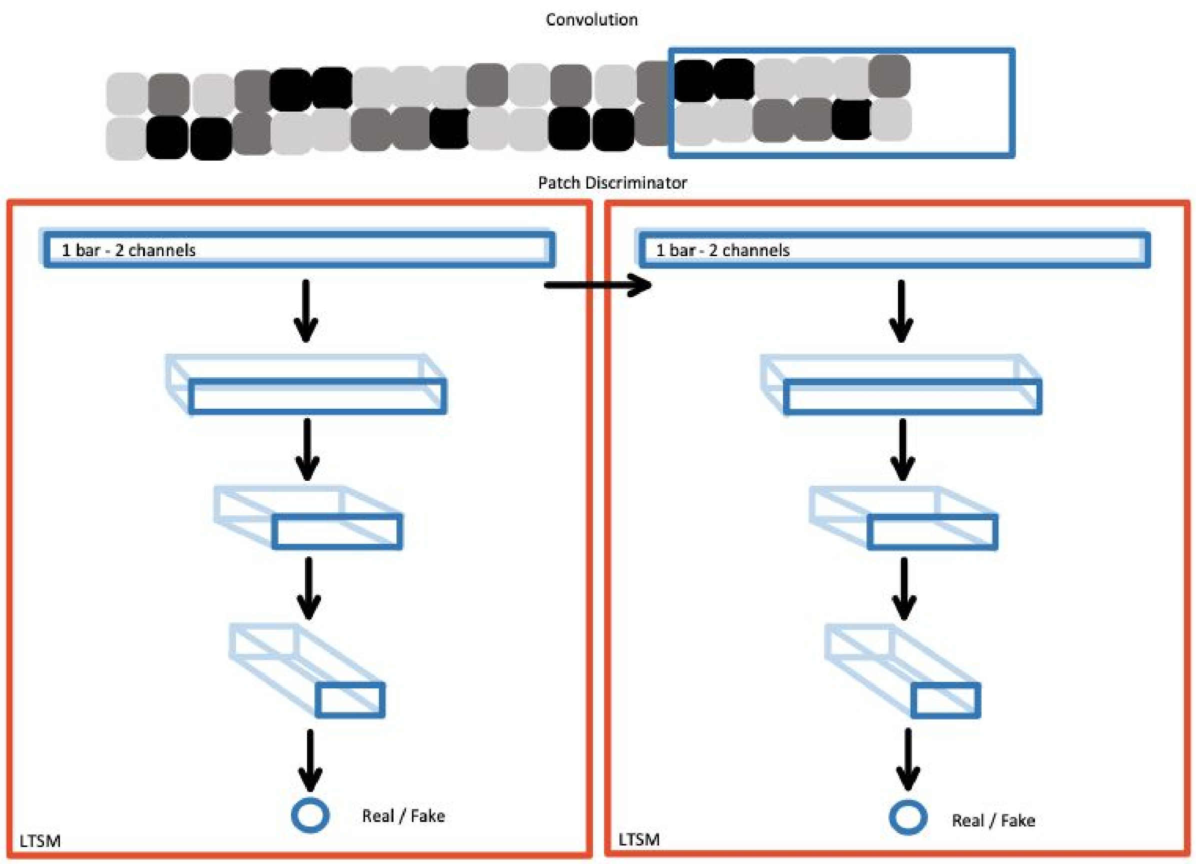


Figure 2: Scaled Down Back-End/Front-End Design

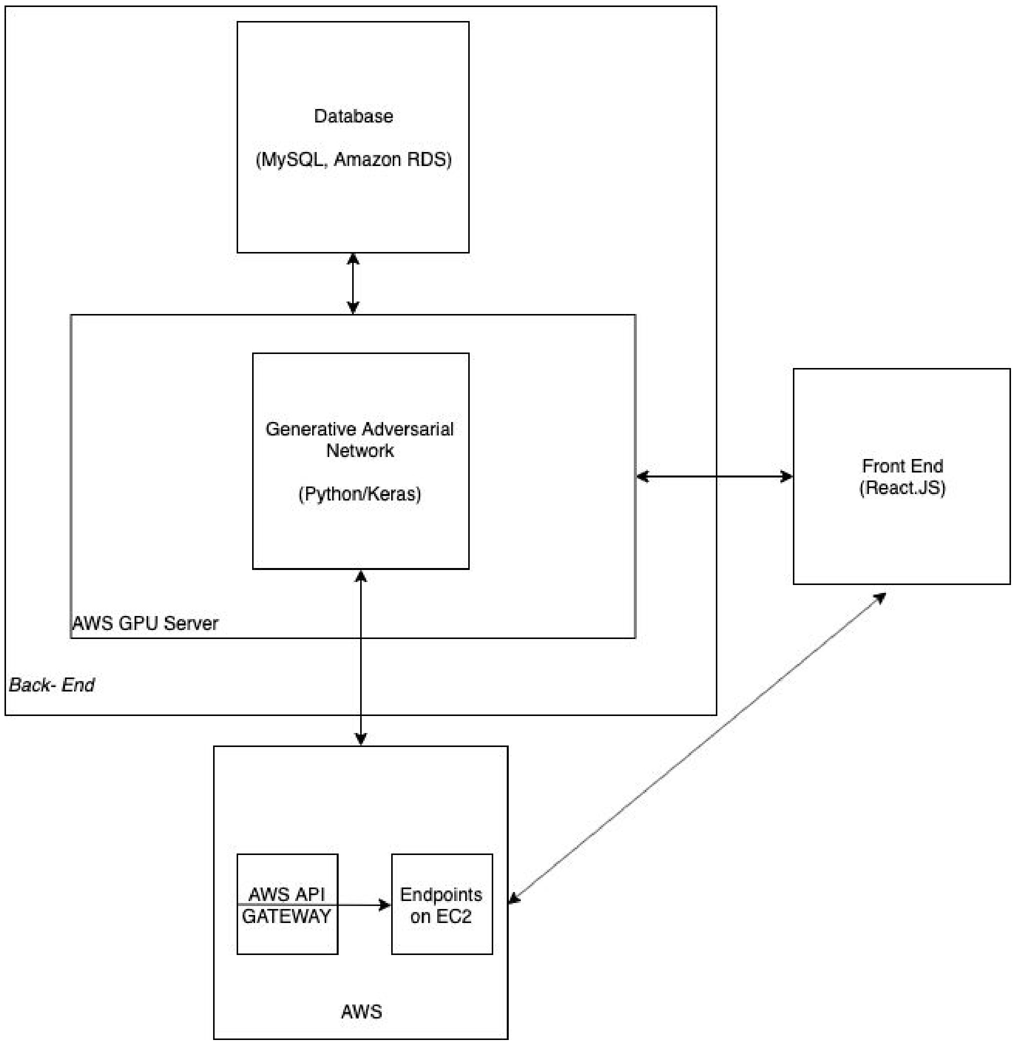
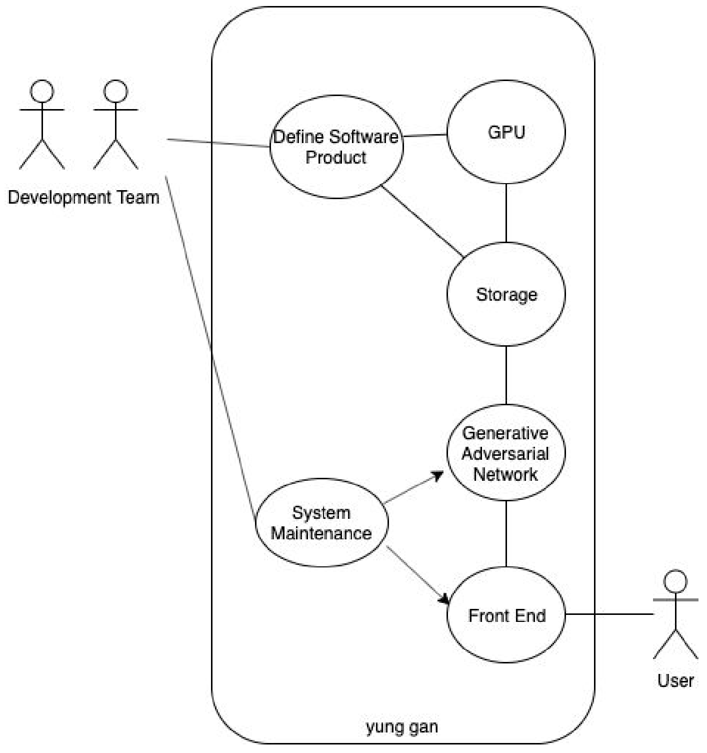


Figure 3: Use Case Diagram



6.3. CSC and CSU Descriptions

6.3.1 Class Descriptions

6.3.1.1 Detailed Class Description 1

.