|  |
| --- |
| Flat lay top view of robot deviating from group |
| Audio Analysis  Monotonous detection |
| |  |  |  | | --- | --- | --- | | Xaltius Pte. Ltd. | 11/30/21 | Smart LMS System | |

Table of Contents

[USE CASE TITLE 2](#_Toc83901001)

[USE CASE DESCRIPTION 2](#_Toc83901002)

[HOW IS IT USEFUL? 2](#_Toc83901003)

[FLOW DIAGRAM 3](#_Toc83901004)

[FLOW DIAGRAM – EXPLANATION 3](#_Toc83901005)

[TECHNICAL ARCHITECTURE 4](#_Toc83901006)

[TECHNICAL ARCHITECTURE – EXPLANATION 4](#_Toc83901007)

[DEPLOYMENT PROCESS 5](#_Toc83901008)

[TRAINING PROCESS 5](#_Toc83901009)

[INFERENCE PROCESS 5](#_Toc83901010)

[AWS COSTING 5](#_Toc83901011)

[HOW TO USE THE DEPLOYED SYSTEM? 6](#_Toc83901012)

[REFERENCES 7](#_Toc83901013)

# USE CASE TITLE

Monotonous Analysis on Trainer Speech

# USE CASE DESCRIPTION

Post-process audio clips of training sessions to provide feedback to trainers on their speech quality.

## HOW IS IT USEFUL?

This use case provides additional feedback on whether the trainer is speaking in a monotonous tone or not. As monotony in speech can induce negativity like sleepiness or boringness, if possible monotonous speech should be avoided to keep students interested and attentive during training sessions.

# FLOW DIAGRAM

Diagram

Description automatically generated

## FLOW DIAGRAM – EXPLANATION

Docker image encapsuling the execution script is built and pushed into AWS ECR for serverless deployment via AWS Lambda.

AWS Lambda executes an AWS Sagemaker Processing Job instance using inbuilt boto3 python library.

Sagemaker Processing Job will query data from RDS and update/insert data into the respective tables in RDS upon job completion.

Function is scheduled via AWS EventBridge as a Cron job of biweekly frequency.

# TECHNICAL ARCHITECTURE

Diagram

Description automatically generated

Segmentor Model

Diagram

Description automatically generated

Pipeline

## TECHNICAL ARCHITECTURE – EXPLANATION

The model architecture comprises of two stages, first was the Segmentation of the waveform to its voiced flags. Voiced flags are frames where there was speech detected. A Conv-RNN model was built upon TIMIT dataset which included audio clips of averaging 5 seconds duration with its relevant phonemic transcriptions. The goal of the CRNN model was to annotate segments within the audio clips that was voiced.

A stack of Conv1D-BatchNorm-MaxPool1D blocks were used as feature extractors from the raw waveforms that were pre-processed into 2D matrices comprising of MFCC features and its delta Δ and delta-delta Δ^2 values on top of the waveforms’ zero crossing rate and fundamental frequency (F0), at a total of 32 predictors.

To enable framewise prediction on the input sequences, time-distributed layer wrappers were used on the flatten and classification head, in addition to returning all output sequences from all the LSTM layers.

Utilizing Librosa’s YIN algorithm implementation, coupled with the output voiced flags from the CRNN model and also the raw waveforms’ zero crossing rates and root mean square values, a rule-based algorithm which took inspiration from the Two Pointers algorithm was constructed to detect periods of monotony based on the values obtained.

A picture containing shoji, building, crossword puzzle, shelf

Description automatically generated

Input matrices plot

# DEPLOYMENT PROCESS

Inference pipeline was deployed using serverless pipeline utilizing customized docker images hosted via AWS Elastic Container Registry (ECR).

Docker images were built with GPU capabilities by building on top of NVIDIA’s TensorRT docker images that includes all NVIDIA Drivers and its prerequisites.

The docker images saved in ECR are then called through AWS built-in library Boto3 using AWS Lambda functions. Outputs from the called functions queries and updates tables in AWS RDS from within the docker image itself.

## TRAINING PROCESS

The Segmentor model was trained locally.

As there will not be anymore training on the Segmentor model, there is no training pipeline required in this use case.

## INFERENCE PROCESS

The inference pipeline was built as an end-to-end structure comprising of both the polarity and phrase models. Output from the polarity model is fed directly into the phrase model to infer the sentiment phrases.

## AWS COSTING

Assuming inference API is called biweekly.

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Instance type | Hours | Costing ($ USD) |
| Sagemaker Processing Job | ml.g4dn.xlarge | 0.25 | 0.52 |
| ECR | - | - | Free tier |
| Lambda | - | - | Free tier |
| RDS | - | - | Free tier |
| Total |  |  | **0.52 per month** |

Inference API

# HOW TO USE THE DEPLOYED SYSTEM?

As this function is self-regulated, there will be no instructions for usage.

# REFERENCES

<https://www.cs.toronto.edu/~graves/ijcnn_2005.pdf>

<https://www.geeksforgeeks.org/two-pointers-technique/>

<https://towardsdatascience.com/two-pointer-approach-python-code-f3986b602640>