Keyword Search in Audio with Adversarial Examples: Final Report

G043 (s1870697, s1834237, s1837246)

Abstract

The central goal of this research is to build an end-to-end keyword recognition system that finds spoken keywords within audio files. To this end, we explore different approaches, from naive direct audio comparison to deep neural networks. The keyword and target audio are first transcribed and then compared for the deep neural network approach. In addition, we investigate the testing performance of these models on adversarial examples (examples specifically crafted to elicit a false response from the model). This research also entails the generation of ground-truth timestamps of terms in a transcript to the audio using forced alignment. During this project, we were faced with numerous resource restrictions, mainly the computational intensity of automatic speech recognition. We find that neither the naive comparison approach nor the deep learning transcription approach leads to acceptable performance. We thus conclude that specifically trained automatic speech recognition and language models are required to improve performance.

1. Introduction

We propose an exploratory approach to spoken keyword identification within audio using automatic speech recognition models with targeted adversarial examples. Given an audio snippet, the model proposed here determines the timestamps within the audio file where a keyword or keyphrase occurs.

1.1. Motivation

Most modern operating systems (GNOME, MAC, Windows, ...) contain a keyboard shortcut that allows users to search for a specific term within a document or website (gno) (mac) (win). We propose a similar functionality for audio recordings wherein the model identifies the timestamp of a spoken utterance within an audio file.

This functionality has a wide range of use cases. Many universities and other educational institutions have long started to provide materials online and offer distance learning opportunities (Cunha et al., 2020). Due to the COVID-19 pandemic, higher education institutions had to shift to online teaching and have started to provide online lectures (Rapanta et al., 2020). Quickly finding information about a relevant topic in often hour-long lectures is a vital and time-saving tool for students. Beyond the educational realm, keyword-search within audio files is an essential tool for digital forensic investigators, who often utilize keyword-search on hard disks and similar storage media (Nizam et al., 2020). The model developed here is a first step in allowing investigators to efficiently search evidence such as dash-cam footage and recorded conversations and improve the robustness of the model against tampered evidence. Further, we aim to investigate the performance of different models when tested on adversarial examples.

1.2. Research Questions and Objectives

This research aims to clarify the following research questions:

- Creating a baseline model using the mean absolute error metric for comparison
- Creating a deep recurrent neural network using Long Short Term Memory models
- Test and evaluate the models on adversarial examples

While initials goals included analysing the effects of adversarial data incorporated into the training data, this goal proved to be unfeasible due to the resource requirements of training a robust speech-to-text system.

1.3. Related Research

Over the last decade, many proprietary and open-source software tools allowing keyword search within audio have gained popularity. These tools include Panopto, Rev, 3PlayMedia and A-Seeker (Affel et al., 2020). A-Seeker uses an approach in which the audio or video file is first transcribed (Affel et al., 2020). Users then search terms in the transcription and are taken to the correct timestamp within the audio, or video (Affel et al., 2020). Processing an hour-long file using A-Seeker takes about half an hour (Affel et al., 2020). (Yu et al., 2005) propose an approach to identifying keyphrases in audio recordings which omits the transcription step. Instead, the authors propose a technique based on word and phoneme search in recognition alternatives, complimented by a supporting indexing scheme (Yu et al., 2005). (Yu et al., 2005) report an accuracy of 84%. Adversarial examples will be used to improve the proposed model. (Carlini & Wagner, 2018) have investigated audio adversarial examples for automatic speech recognition. The authors propose an approach for creating audio adversarial examples based on minimising the Connectionist Temporal Classification (CTC) loss function.

This is the first attempt at using adversarial examples to improve keyword search within audio to the authors' knowledge.

2. Data set and task

2.1. Data set

For this project, we use the TED-LIUM release 3 (TED-LIUM 3) (Hernandez et al., 2018) dataset. The Ubiqus company created TED-LIUM 3 in collaboration with LIUM (University of Le Mans, France). This dataset contains a total of 452 hours of audio recordings from TED talk conferences [1], along with aligned automatic transcripts. Each talk is provided as a `sphere` audio file with its corresponding transcript as `.stm` file. TEDLIUM is available as a torchaudio dataset, making it suitable for this project. Due to resource limitations, we use a subset of the TED-LIUM 3 dataset. This subset contains 3500 audio snippets. Each snippet consists of segmented samples from the original audio files, of 33 files in total. The data is split in an 80:10:10 ratio of train, validation, and test set. In addition, the task requires audio snippets of single keywords and keyphrases. We obtain single keywords from the English subset of the Multilingual Spoken Words dataset (MSWC) (Mazumder et al.). This dataset contains over 340,000 keywords and totals over 6,000 hours (Mazumder et al.).

For this project, we merge these two datasets. For each TED-LIUM 3 sample, we pick several keywords from the transcript. We then retrieve audio files corresponding to these keywords from MSWC. Using word alignment techniques (see Section 3.2), we then find the timestamps of these keywords within the audio. These timestamps act as the ground-truth labels.

2.2. Preprocessing

Each TED-LIUM 3 audio sample is split up into several overlapping segments using a sliding window approach (Kang & Guo, 2009). The window is the same size as the audio signal of the keyword or keyphrase, which we aim to identify within the audio sample. The TED-LIUM 3 and keyword/ keyphrase audio samples are then converted to Mel-Frequency Cepstral Coefficient (MFCC) representation (mfc, b). The MFCC representation splits audio files into 50 frames per second and maps each frame to the frequency domain (Carlini & Wagner, 2018) and thus reduces the dimensionality of audio files (see Section 3). The MFCC values are normalised by taking the mean across features.

2.3. Task

We aim to create a model that identifies keywords within audio files for this project. To this end, we investigate three approaches. In a first, naive approach, audio snippets of keywords are compared to TED-LIUM 3 data by the mean absolute difference of their normalised MFCC-values. The model identifies the keyword at the window resulting in the least mean absolute difference (w_min). The correct timestamp is computed according to Equations 1 and 2.

$$start\_time = \text{index of w\_min} / sample\_rate \quad (1)$$

$$end\_time = start\_time + keyword\_audio\_length \quad (2)$$

In a second approach, we use the pre-trained DeepSpeech speech recognition system (Han, 2014). This model uses recurrent neural networks to transcribe audio files (Han, 2014). For each phoneme in the transcript, the model also returns the timestamp at which it is uttered. However, this model alone does not achieve the desired task (finding the timestamp of a keyword within an audio file). For this purpose, we created the model pipeline depicted in Figure 1. The DeepSpeech model first transcribes the MSWC keyword and TED-LIUM 3 speech. The transcripts are then matched up by iterating over each substring of the TED-LIUM3 transcript. If we can find a substring within the TED-LIUM 3 transcript that matches the MSWC keyword transcript, this substring's start and end timestamp are returned.

Finally, we adapted a 2D Recurrent Neural Network model provided by the Keras [2] API. The details of this model are laid out in Section 4.3.

We create a total of 100 adversarial examples using an implementation provided by Carlini et al. (Carlini & Wagner, 2018). The models are then tested on the adversarial examples, allowing us to analyse the performance differences of models when confronted with such examples versus untampered data.

In addition, we created several models that we do not include in the evaluation. They are either too computationally expensive or do not function well enough. We first planned to generate adversarial examples using the ESPnet model. In this scenario, the model would be used to transcribe the audio file iteratively, adding carefully crafted noise to the file until we reach the desired transcript. The Pipeline for creating adversarial examples using ESPnet is found in Appendix 8.

Firstly, we created a model using ESPnet. ESnet is a continually updating, open-source -end-to-end speech processing toolkit with internal Kaldi integration (Watanabe et al., 2021) (esp) that has gained popularity since its inception in 2017 (Watanabe et al., 2021). We chose ESPnet due to its attractive quality of providing built-in pre-processing for audio files. This feature promised to save a significant amount of time and computational space; however, we did not anticipate the extra storage space required. Further, ESPnet provides several models out-of-the-box (esp). This feature is interesting in terms of comparing different models and evaluating their trade-offs. It was first planned to use an Encoder-Decoder architecture with hybrid CTC attention (Watanabe et al., 2017) 2. This structure uses

[1] https://www.openslr.org/51/

[2] https://keras.io/

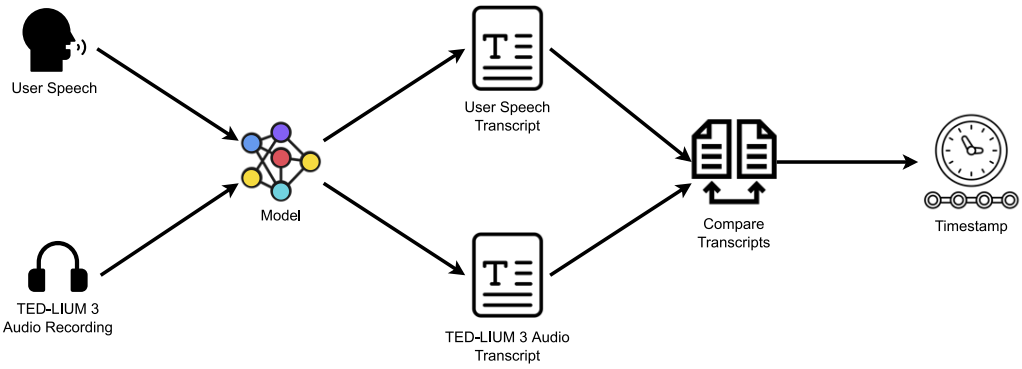


Figure 1. Pipeline for identifying the timestamp of a keyword/ keyphrase in an audio file. The DeepSpeech model transcribes the user audio (a keyword or keyphrase) and the TED-LIUM 3 audio file separately, producing a transcript and timestamps. These transcripts are then compared. Upon detecting the keyphrase in the TED-LIUM 3 transcript, output the starting and ending timestamp of the keyphrase according to the DeepSpeech model.

a convolutional neural network combined with a bidirectional recurrent neural network (specifically, containing long short-term memory models). The Listen, Attend and Spell methodology (Chan et al., 2015a) inspires this model. This model was chosen due to its attention mechanism, as the initial project goals included visualising areas of interest of the model to allow deeper analysis. Further, ESPnet provides integrated language (esp) models, which could dramatically improve model accuracy. While the creation and training of the described model were ambitious, we hypothesised that it was a realistic goal thanks to built-in multi-GPU-support (esp) and the availability of the MLP cluster.

This model proved non-viable. The standard ASR models designed for the TED-LIUM 3 dataset were too computationally intensive for the project resources. Different factors contributed to this. Firstly, a large amount of preprocessing was required. The data had to be downloaded and split into training, development, and test sets. In addition, TED-LIUM 3 provides its audio files in verb.sph+ and `.stm` formats, which are relatively uncommon and had to be parsed into `wav` format to be usable for the model. `.sph` files are between one and two minutes long on average. The transcripts of each file are broken into variable segments ranging from 50 to 300 segments. This format is a non-standard for conducting ASR. For instance, popular datasets such as the LibriSpeech ASR corpus (Panayotov et al.) and Lip Reading System (Afouras et al., 2018) contain audio files between two and five seconds, along with the corresponding transcripts. In other words, the TED-LIUM 3 audio files had to be pre-processed to match the standard file formats.

The pre-processing step is where the first issue presents itself: it produces an amount of data that is 20GB larger than the limit of a student dice machine workspace and only 10GB below the limit of the MLP cluster. The model described requires an environment of approximately 100-200GB. This estimation considers storing the pre-processed and original data and the model. Note that this issue presents itself before the training phase of the model. Unfortunately, we were unable to progress past the training stage using the ESPnet approach due to the local and google cloud virtual machines experiencing `CUDA out of memory` errors, even when reducing the complexity of the network the batch size and turning off bidirectionality. At the time, the best GPU available to us was the Nvidia 2060 RTX Turbo with 8GB of storage. While building, compiling, and running ESPnet on the MLP cluster is a solution, in theory, the necessity of packages such as ESPnet [3] and a lack of sudo privileges rendered this solution non-viable in practice.

We expect that ESPnet would have yielded the best results to find keywords within audio files. We arrived at this conclusion as the publicly available data of the model performance of ESPnet on the TED-LIUM 3 data shows a word error rate (WER) of only 9.6%. The model built with ESPnet is depicted in Figure 2. Further, we attempted to implement an end-to-end automatic speech recognition system using Pytorch [4] when the ESPnet approach proved unfeasible. To be able to maintain the original model while reducing the computational intensity, we used an open-source implementation of the LAS model structure (Chan et al., 2015b) which employed transformer models. This model employs LSTMs and provides attention features (Chan et al., 2015b). Thus, we would still be able to visualise areas of interest within the data at a later stage of the project. In addition, this model allows for a more hands-on approach, as the code is directly accessible in a smaller file structure. This allowed for more fine-tuning when the need to reduce complexity occurred. Unlike ESPnet, this model infrastructure did not provide pre-processing functionalities. Handling the pre-processing manually meant creating metadata files and segmenting the `.sph` files into smaller segments, depending on the transcript. For convenience, we converted the `.sph` files to `.wav` for this approach. While this pre-processing approach is less computationally expensive than the approach employed by ESPnet, the MFCC features had to be computed dynamically in each batch during training. This, too, is very computationally expensive,

[3] https://espnet.github.io/espnet/installation.html
[4] https://pytorch.org/

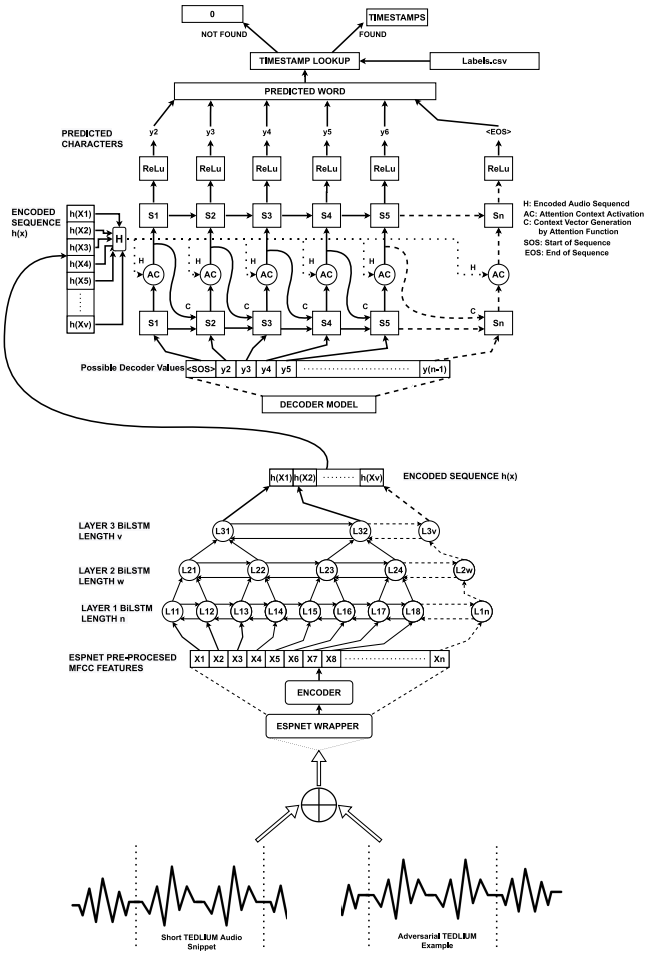


Figure 2. Ideal CTC loss Encoder/Decoder structure. Trained end to end the encoder and decoder structure allows us to input audio and output characters. Here in the model H: depicts the encoded sequence, C: acts as the context vector which stores temporal information. AC: are our attention context activation taking in a little content from the past and applying it to the present and future. SOS: and EOS: stand for start of sequence and end of sequence. The idea inspired by the LAS architecture (Chan et al., 2015b) is similar to compression in reasoning. This is where we take an audio segments, encode it into a smaller form reducing its dimensional then expand it back into character sequences. These character sequences act as buckets that output a confidence of what character is being said in a given time frame T. The full size graphic can be found in the appendix under figure 13

and the provided GPU cluster and personal machines were unable to handle the workload, even when reducing the number of training samples to 50. To combat this issue, we limited the model complexity by reducing the number of hyperparameters by 90% and reducing the model layers from five to two dimensions. While the reduction allowed us to run the models successfully, it resulted in an extremely high WER. To successfully run the model and attain an acceptable performance, we require a larger infrastructure with the possibility of using multiple GPUs. The structure of this model can be found in Appendix 7.2.

To evaluate the models, we will compare them to the ground-truth timestamps (see Section 2.1 and 3.2). A timestamp (containing start and end time) for a keyword or keyphrase is considered to be correct if it overlaps with the ground-truth timestamp. This accounts for potential inaccuracies in the ground-truth labels and reflects the fact that it is more important to find an approximate timestamp. For instance, consider the example of a lecture video: it is more important to find the section discussing a *topic* than the exact timestamp of a specific term. We further consider the time required to train a model and the time required to produce a prediction (independent of its correctness). Lastly, we consider the Word-Error-Rate (see Section 3).

3. Methodology

We will first define the main methods used in this research: CTC loss, Long Short-Term Memory models, Mel Frequency Cepstral Coefficients, and the word-error rate. After that, we will explain the techniques used for the different parts of this project, such as the model used for keyword identification, the creation of adversarial examples, and finding the "ground-truth" timestamps of keywords.

- **CTC-Loss**: Connectionist Temporal Classification (CTC) is a loss function used for training automatic speech recognition (ASR) neural networks. It solves the issue of not knowing how the characters in a transcript align to the audio (Hannun, 2017). Given an input sequence X (e.g., audio) and an output sequence Y (e.g., transcript), the CTC algorithm computes an output distribution over all Y and we infer that the Y with the maximum likelihood is likely correct (Hannun, 2017).

- **Long Short-Term Memory (LSTM) models**: For the third approach to keyword identification, we use a Long Short-Term Memory (LSTM) model. LSTMs are a type of recurrent neural network, making them suitable for learning from sequential data (Hochreiter & Schmidhuber, 1997).

- **Mel Frequency Cepstral Coefficient**: A method to extract features from audio (mfc, a). Sounds produced by humans depend on the shape of one's vocal tract (mfc, b). MFCC features represent units of sound as

such shapes (mfc, b).

- **Word-Error-Rate (WER)**: Industry standard for measuring model accuracy (mic). The WER counts the number of incorrect words identified during recognition and divides by the total number of words provided in a ground-truth transcript (mic). The WER is defined by Equation 3, where I indicates the incorrectly added words, D indicates the terms not occurring in the transcript returned by the model, and S indicates words that are substituted between reference and hypothesis (mic).

$$WER = \frac{I + D + S}{N} * 100\% \tag{3}$$

3.1. Adversarial Examples

Adversarial examples are samples created to achieve a certain (incorrect) model output (Geng & Veerapaneni, 2018). A common method is to add a small amount of carefully chosen noise to an original sample (that is, a sample that has not been altered). Adversarial examples come in two varieties: targeted or non-targeted. Targeted adversarial examples are samples that are indistinguishable from original samples for humans. Non-targeted adversarial examples, on the other hand, often appear as noise to humans, while the model classifies the sample as a particular category with high confidence. Examples of targeted and non-targeted adversarial examples can be found in Appendix 7.4.

Applied to audio files, this equates to adding carefully crafted noise. One goal of this research effort is to gauge the effects of adversarial examples on the training and performance of a deep learning model with the task of identifying keywords in audio. To obtain adversarial examples, we adapted the sample code provided by Carlini and Wagner (Carlini & Wagner, 2018) [5]. Carlini and Wagner (Carlini & Wagner, 2018) create targeted audio adversarial examples using the DeepSpeech (Han, 2014) speech-to-text system. The goal of the adversarial examples is to transcribe an audio file to a sentence that is different from what is actually heard. DeepSpeech (Han, 2014) first pre-processes audio samples and converts them to MFC format. Long Short-Term Memory models follow these steps. The task of creating adversarial examples is often formulated as an optimisation problem (Carlini & Wagner, 2018). Let x be the original audio, y the desired target transcript and $C()$ be the model. We aim to find noise δ to minimise the error $C(x') = y$, where $x' = x + \delta$ is the adversarial example. We further impose the restriction that x and x' sound similar.

3.2. Word Alignment

This section details the approach used to generate "ground-truth" labels for the dataset.

TED-LIUM 3 contains aligned automatic transcripts, dividing each audio file into audio segments or "samples" with a starting time, ending time, and the transcription of the respective audio snippet. Using MSWC, we first connect the two datasets by iterating through each audio and linking the text keywords in the transcript to the words recorded in MSWC dataset, if any. Words not occurring in MSWC are skipped. Some pre-processing and parsing is done for the transcripts to align to the MSWC words. This includes the transformation of words into numbers, differences in Unicode format handling for some punctuations, changing numericals to words (e.g., "1970s" to "nineteen-seventies"), tokenisation, and handling some edge cases (e.g., html5 becomes "HTML five").

We then attempt to generate a timestamp for each keyword in each audio segment. That is, a time interval for a keyword utterance's start and end time. This is achieved using a process called *forced alignment*. Within our implementation, this is done using the Wav2vec2 (Baevski et al., 2020) model to extract features. It is a self-supervised framework for ASR, encoding audio data using a convolutional neural network model that outputs latent speech representations and generates 'hidden' features to differentiate between the observed audio features. These are passed into a transformer network to build a contextualised representation, which the model can use to represent the complexity and traits of word usage, syntactically and semantically (Peters et al., 2018).

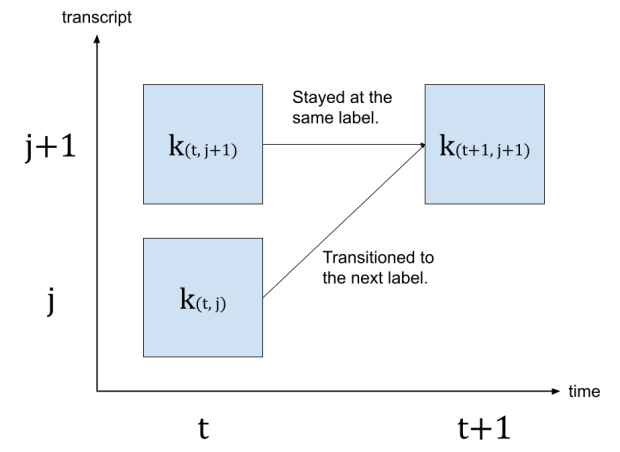


Figure 3. Wav2vec2 Transition Illustration, taken from (Baevski et al., 2020). Note that t denotes the index in the time axis, j is the index in the character axis, and k is the character label itself. The illustration indicates the transition of going from one character label to another. The probability of time step t+1 occurs depending on two cases, whether the current character label has changed to a different label in the next time step t or if it is the same as the one at time t.

For the task of force alignment, we use this model in tangent with the Connectionist Temporal Classification (CTC) segmentation algorithm. CTC is a loss function used for training ASR neural networks. However, in our context, we apply a method proposed in this paper (CTC paper) called CTC-Segmentation, which is an algorithm to align text to the audio (using the transcripts) in the occurrences of any

[5] https://github.com/carlini/audio_adversarial_examples

other unknown words (which are identified as <unk>). We reused and modified Moto Hira's implementation of the algorithm in Pytorch to achieve forced alignment for our purposes. Firstly, we generate label probabilities for each audio segment, also known as emission matrix, using the pre-trained Wav2vec2 model for Automatic Speech Recognition (ASR). Secondly, we create a trellis 2d matrix with time and label axis, which constitute the probability of labels aligned at each time step. The probability of time step t+1 occurs depending on two cases, whether the label has changed to a different label in the next time step t or if it is the same.

Finally, using the trellis matrix, we backtrace the elements generated following those with the highest probabilities until we reach the beginning of our audio segment. Note that the matrix is used for path-finding, but each segment's final probability is the frame-wise probability from the emission matrix. The characters are then merged into their original word representations. More details on the CTC algorithm to generate timestamps can be found in this article (Hannun, 2017). Figure 4 shows a demonstration of the process concatenated in the two plots. This is done for each audio segment, enabling us to train and evaluate our CTRLF model using these as labels.

4. Experiments

4.1. Baseline Experiment

This model serves as a baseline. It helps to analyse the performance gains of deep learning models compared to a naive approach that does not require any machine learning. The baseline model is set up as detailed in Section 2.3. The model does not require any training or parameter settings, as it merely uses the Mean Absolute Error to compare audio files. We used the baseline model to make predictions for 100 samples of the original TED-LIUM 3 data and the adversarial examples, respectively.

On the original TED-LIUM 3 data, the baseline model neither achieves an acceptable performance in terms of accuracy nor runtime. The model achieves an average testing accuracy of 12.5 % and an average runtime (the time it takes to make a single prediction) of 30 minutes. After evaluating the source code, we conclude that the long runtime is because each TED-LIUM 3 audio file must be split into windows of the same size as the keyword. An audio segment of only a few seconds can result in several thousand such windows. Thereafter, each window must be converted to MFCC representation. Only then can the MFCC-keyword and window be compared. This introduces considerable overhead. Further, we attribute the low accuracy to the fact that MFCC-comparison is inaccurate. Errors in the conversion of the keyword and audio file can propagate and lead to significant mean absolute errors despite the keyword matching a particular audio segment.

The model performs similarly on the adversarial data, achieving an average testing accuracy of 12.03% and average runtime of approximately 30 minutes. The adversarial examples adds minimal noise to the original sample, not changing the waveform of the original in a significant manner (for example waveforms, see Appendix 7.4), resulting in model that is not easily influenced by adversarial examples.

4.2. DeepSpeech Model (Pre-trained)

In this experiment we test a state-of-the-art ASR model. This allows us to analyse how a pre-trained model performs when incorporated into an end-to-end pipeline. We test the pre-trained DeepSpeech model [6]. Due to being pre-trained, this does not require any training on our part. Thus, we were also not required to specify any hyperparameter settings. To make predictions, we use the prediction pipeline depicted in Figure 1 and detailed in Section 2.3. This model was tested on a subset of 100 samples of the TED-LIUM 3 dataset and the set of adversarial examples, respectively.

The model achieves an accuracy of 22% on the original test set (see Table 1). The accuracy is likely low as errors in the transcripts are amplified. Thus, an error in either the keyword or the audio transcript can lead to an incorrect timestamp. Further, we must consider the possibility of the ground-truth timestamps being incorrect. Hence, correct timestamps may be considered incorrect if they do not overlap with the ground-truth. This model is faster than the baseline (see Table 1), as DeepSpeech provides optimised pre-processing (dee).

The model accuracy drops significantly when tested on adversarial examples (Table 2). This is explained by the fact that the DeepSpeech network was used to generate these examples, and hence will produce incorrect transcriptions. One may achieve improved results when training the network on the original and adversarial data.

4.3. Keras Model

We adapted the model provided by Keras [7]. We replaced the GRU layer within the internal architecture with an LSTM layer (Section 3). This model uses the CTC loss metric (Section 3) and an Adam learning rate of $1e-4$. While the model initially required 30 minutes for a single training epoch, this was reduced to 8 minutes after reducing the model complexity. The model was trained for a total of 57 epochs. Figure 5 shows the training and validation loss of this model. After an initial steep decrease in the training error, little learning happens thereafter at both training and validation time. The model may be severely underperforming due to too small of a training set (2700 audio segments), compared to the original model which used 11790 segments. In addition, the model's complexity was lowered due to the memory expense and slow training of the original model. This demonstrates that the model does not capture many patterns with the configurations and the amount of data passed. The word error rate on the test split of 300 on

[6] https://deepspeech.readthedocs.io/en/r0.9/USING.html
[7] https://keras.io/examples/audio/ctc_asr/

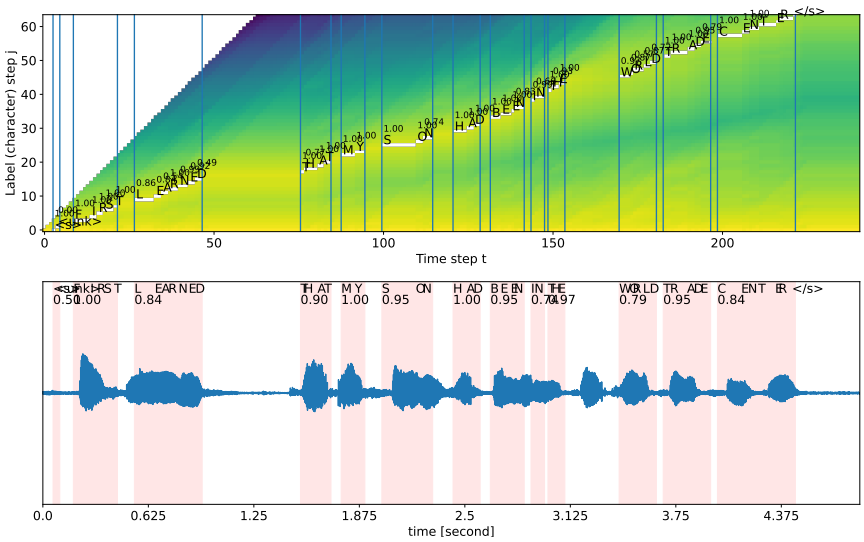


Figure 4. The figure shows the visualisation of the trellis matrix in the top plot and the audio wave signal in the bottom plot. The characters are also displayed in the trellis matrix, representing at which estimated time it was found of one audio clip. The audio plot contains the joined words, with the red background demonstrating the span of a given word. The numbers above the characters and joined words represent the confidence score of the Wav2Vec2 model.

Model	Testing Accuracy in %	Avg. Prediction Time in s
Baseline	12.50	1256.1335
DeepSpeech	22.14	5.1635

Table 1. Model performance in terms of accuracy and average prediction time on the original TED-LIUM 3 dataset.

Figure 5. The training and validation loss of the attempted Keras ASR model trained for the purposes of this project.

TEDLIUM data is 99.37% Word Error Rate, and 98.98% Word Error Rate. Because the results are extremely poor compared to DeepSpeech2 (Amodei et al., 2015), which is able to achieve 5.33 on LibriSpeech test dataset (Panayotov et al.). Due to these reasons, a deeper network and large amounts of data are necessary to achieve substantial results for performing ASR.

5. Related work

While only little research has been focused on keyword search within audio (Affel et al., 2020), the task of speech transcription has received attention over the years. As a result, there exists an abundance of models and toolkits for this purpose, such as ESPnet (esp) and DeepSpeech (Han,

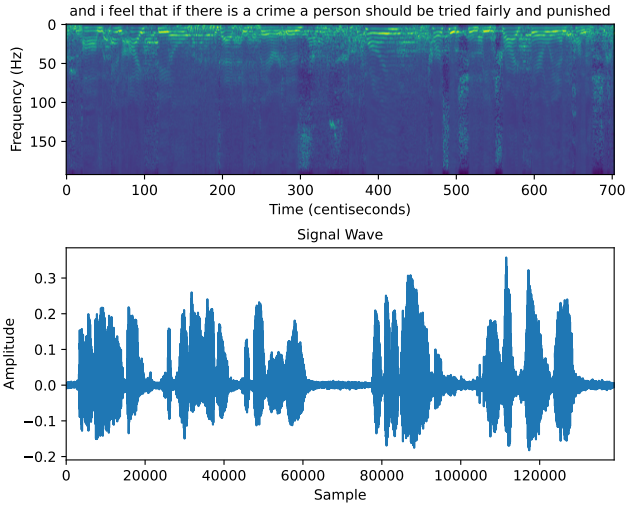


Figure 6. Example of Audio Spectogram and Audio Signal Wave of a TED audio snippet. Spectograms are used as inputs to the Keras model.

Model	Testing Accuracy in %	Avg. Prediction Time in s
Baseline	12.03	1298.7675
DeepSpeech	0.078	5.0515

Table 2. Model performance in terms of accuracy and average prediction time on the adversarial examples. Due to the major underperformance in Word Error Rate of the Keras model due to current resource restrictions, there was no merit in measuring the accuracy of timestamp matching.

2014). The most popular keyword-searching applications, such as A-Seeker (Affel et al., 2020), Panopto, Rev and 3PlayMedia. These applications rely on a transcription step similar to that used within this research and depicted in Figure 1. The A-Seeker platform too uses DeepSpeech for speech transcription (Affel et al., 2020). While the research presented here is exploratory, aiming to compare different models, the goal of A-Seeker is to provide a state-of-the-art end-to-end audio transcription platform (Affel et al., 2020). (Affel et al., 2020) report that the quality of transcription depended on factors such as the quality of the audio file. Further, they report inaccuracies caused by long pauses due to their handling of empty space (Affel et al., 2020). As we considered entire audio files, we did not encounter this issue.

Errors in the transcripts can propagate and lead to significant errors in the timestamps produced by the speech-to-text model (Yu et al., 2005). (Yu et al., 2005) thus suggest an approach omitting the transcription step. The approach is based on using lattices to search recognition alternatives (Yu et al., 2005). Lattices represent the multiple hypotheses generated from an automatic speech recognition system (Pandey et al., 2021). Keywords are then identified by their word posterior probabilities (Yu et al., 2005). This approach has proven successful with an average accuracy of 84% (Yu et al., 2005).

5.1. Future Work

As detailed in Section 2.3, running the ESPnet model was outwith the resource provisions of this project. As we expect this model to perform best, however, future work will include obtaining the necessary resources to analyse the model to identify keywords in original and adversarial audio examples. Figure 2 depicts the architecture of an optimal model, which we will implement given adequate resources. This architecture mimics the structure of Listen Attend, and Spell (Chan et al., 2015b) by using an encoder-decoder structure with attention and CTC Loss. This allows us to use temporal data through the attention mechanism to aid with character prediction. The encoder-decoder structure allows the model to be end-to-end, creating seamless transitions from audio to characters. The approach was suggested in 2017 and has been popularised ever since as a standard ASR methodology (Chan et al., 2015b). Further, we will use this model to create adversarial examples targeted to detect specific keywords and their timestamps within the audio. Figure 8 in Appendix 7.3 depicts this architecture.

The lack of ground-truth labels presented an obstacle to the model evaluation. While audio transcripts are widely available, we could not locate a dataset containing timestamps for each phoneme in the dataset. Future efforts will center around acquiring an accurately labeled dataset. However, due to the manual effort involved in this, this is outwith this project's scope.

To further analyse how a model for keyword-search within audio makes decisions, especially when faced with adversarial examples, we aim to adopt class activation maps. Traditionally, these are used on images to visualise the regions that a CNN model tends to focus on (Zhou et al., 2015). Namely, we could use GradCAM on the spectograms of the audios samples. GradCAM is a method that focuses on the flow of gradients into the final convolutional layer of a model to produce a mapping that demonstrates the critical regions in an image, (Selvaraju et al., 2016). This will allow us to study the differences between original audio clips and their adversarial counterparts and the effects of training a model on adversarial examples.

6. Conclusions

As part of this research, we examined different ASR models, from a naive baseline model to deep neural networks. In addition, we used deep learning approaches to align words to their respective timestamps within audio. Further, we created adversarial examples which were used to test the created models.

During the course of this project, we found automatic speech recognition to be highly resource intensive. To obtain even reasonable model performance, much larger resources than those currently available to us are required.

Nevertheless, we can conclude that an architecture in which the keyword and target audio are directly compared is not ideal for the purpose of identifying spoken keywords within audio, especially due to the slow runtime. The low accuracy in the ASR model also demonstrates that the model is not fitted well with the ground-truth transcriptions of TED-LIUM 3. Future experiments should include training a language model as well in order to tune Deepspeech to the sentence structure of TED-LIUM 3 transcripts. Additional GPU systems should be acquired to inhibit deeper networks such as ESPnet to improve overall accuracy. A more developed exploration into generation and use of adversarial examples would provide a significant boost to the robustness of the networks, thus improving the accuracy further.

Lastly, this research has emphasized the need for accurately labeled data containing timestamps for phonemes.

References

Welcome to DeepSpeech's documentation! — Mozilla DeepSpeech 0.9.3 documentation. URL https://deepspeech.readthedocs.io/en/r0.9/.

ESPnet: end-to-end speech processing toolkit — ESPnet 0.10.7a1 documentation. URL https://espnet.github.io/espnet/.

Keyboard Skills (GNOME 2.0 Desktop for the Solaris Operating Environment User Guide). URL https://docs.oracle.com/cd/E19754-01/806-6873/6jfpm4g35/index.html.

Mac keyboard shortcuts - Apple Support. URL https://support.apple.com/en-us/HT201236.

Practical Cryptography, a. URL http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

The dummy's guide to MFCC. Disclaimer 1 : This article is only an... | by Pratheeksha Nair | prathena | Medium, b. URL https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd.

Evaluate and improve Custom Speech accuracy - Speech service - Azure Cognitive Services | Microsoft Docs. URL https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-custom-speech-evaluate-data.

Keyboard shortcuts in Windows. URL https://support.microsoft.com/en-us/windows/keyboard-shortcuts-in-windows-dcc61a57-8ff0-cffe-9796-cb9706c75eec.

Deep Speech: Scaling up end-to-end speech recognition. 12 2014. doi: 10.48550/arxiv.1412.5567. URL https://arxiv.org/abs/1412.5567.

Affel, Harrison, Cox, Sean, and Pham, Cuong. A-Seeker: An Efficient Audio Transcription Platform; A-Seeker: An Efficient Audio Transcription Platform. *2020 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2020. doi: 10.1109/URTC51696.2020.9668862.

Afouras, Triantafyllos, Chung, Joon Son, and Zisserman, Andrew. LRS3-TED: a large-scale dataset for visual speech recognition. *CoRR*, abs/1809.00496, 2018. URL http://arxiv.org/abs/1809.00496.

Amodei, Dario, Anubhai, Rishita, Battenberg, Eric, Case, Carl, Casper, Jared, Catanzaro, Bryan, Chen, Jingdong, Chrzanowski, Mike, Coates, Adam, Diamos, Greg, Elsen, Erich, Engel, Jesse H., Fan, Linxi, Fougner, Christopher, Han, Tony, Hannun, Awni Y., Jun, Billy, LeGresley, Patrick, Lin, Libby, Narang, Sharan, Ng, Andrew Y., Ozair, Sherjil, Prenger, Ryan, Raiman, Jonathan, Satheesh, Sanjeev, Seetapun, David, Sengupta, Shubho, Wang, Yi, Wang, Zhiqian, Wang, Chong, Xiao, Bo, Yogatama, Dani, Zhan, Jun, and Zhu, Zhenyao. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015. URL http://arxiv.org/abs/1512.02595.

Baevski, Alexei, Zhou, Henry, Mohamed, Abdelrahman, and Auli, Michael. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. 6 2020. URL http://arxiv.org/abs/2006.11477.

Carlini, Nicholas and Wagner, David. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. 2018. doi: 10.1109/SPW.2018.00009. URL http://nicholas.carlini.com/code/audio.

Chan, William, Jaitly, Navdeep, Le, Quoc V, and Google Brain, Vinyals. Listen, Attend and Spell. Technical report, 2015a.

Chan, William, Jaitly, Navdeep, Le, Quoc V., and Vinyals, Oriol. Listen, attend and spell. *CoRR*, abs/1508.01211, 2015b. URL http://arxiv.org/abs/1508.01211.

Cunha, Maria Nascimento, Chuchu, Tinashe, and Maziriri, Eugine Tafadzwa. Threats, challenges, and opportunities for open universities and massive online open courses in the digital revolution. *International Journal of Emerging Technologies in Learning*, 15(12):191–204, 2020. ISSN 18630383. doi: 10.3991/ijet.v15i12.13435.

Geng, Daniel and Veerapaneni, Rishi. Tricking Neural Networks: Create your own Adversarial Examples. Technical report, 2018. URL https://ml.berkeley.edu/blog/posts/adversarial-examples/.

Hannun, Awni. Sequence Modeling with CTC. *Distill*, 2(11):e8, 11 2017. ISSN 2476-0757. doi: 10.23915/DISTILL.00008. URL https://distill.pub/2017/ctc.

Hernandez, François, Nguyen, Vincent, Ghannay, Sahar, Tomashenko, Natalia, and Estève, Yannick. TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker adaptation. 5 2018. doi: 10.1007/978-3-319-99579-3{_}21. URL https://arxiv.org/abs/1805.04699.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term Memory. *Neural Computation*, 9(8), 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Kang, Guangyu and Guo, Shize. Variable sliding window dtw speech identification algorithm. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, volume 1, pp. 304–307. IEEE, 2009.

Mazumder, Mark, Chitlangia, Sharad, Banbury, Colby, Kang, Yiping, Ciro, Juan, Mlcommons, Factored /, Achorn, Keith, Galvez, Daniel, Sabini, Mark, Mattson Google, Peter, Kanter, David, Diamos, Greg, Google, Pete Warden, Meyer, Josh, and Reddi, Vijay Janapa. Multilingual Spoken Words Corpus. URL https://mlcommons.org/en/.

Nizam, Syafiqah Hanisah Shahrol, Ab Rahman, Nurul Hidayah, and Cahyani, Niken Dwi Wahyu. Keyword Indexing And Searching Tool (KIST): A Tool to Assist the Forensics Analysis of WhatsApp Chat. *International Journal on Information and Communication Technology (IJoICT)*, 6(1):23, 6 2020. doi: 10.21108/ijoict.2020.61.481.

Panayotov, Vassil, Chen, Guoguo, Povey, Daniel, and Khudanpur, Sanjeev. LIBRISPEECH: AN ASR CORPUS BASED ON PUBLIC DOMAIN AUDIO BOOKS. URL http://www.gutenberg.org.

Pandey, Prabhat, Duarte Torres, Sergio, Orkan Bayer, Ali, Gandhe, Ankur, Amazon, Volker Leutnant, and Ai, Alexa. LATTENTION: LATTICE-ATTENTION IN ASR RESCORING. Technical report, 2021.

Peters, Matthew E., Neumann, Mark, Iyyer, Mohit, Gardner, Matt, Clark, Christopher, Lee, Kenton, and Zettlemoyer, Luke. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018. URL http://arxiv.org/abs/1802.05365.

Rapanta, Chrysi, Botturi, Luca, Goodyear, Peter, Guàrdia, Lourdes, and Koole, Marguerite. Online University Teaching During and After the Covid-19 Crisis: Refocusing Teacher Presence and Learning Activity. *Postdigital Science and Education*, pp. 923–945, 2020. doi: 10.1007/s42438-020-00155-y. URL https://doi.org/10.1007/s42438-020-00155-y.

Selvaraju, Ramprasaath R., Cogswell, Michael, Das, Abhishek, Vedantam, Ramakrishna, Parikh, Devi, and Batra, Dhruv. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 128(2):336–359, 10 2016. doi: 10.1007/s11263-019-01228-7. URL http://arxiv.org/abs/1610.02391http://dx.doi.org/10.1007/s11263-019-01228-7.

Watanabe, Shinji, Hori, Takaaki, Kim, Suyoun, Hershey, John R., and Hayashi, Tomoki. Hybrid ctc/attention architecture for end-to-end speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1240–1253, 2017. doi: 10.1109/JSTSP.2017.2763455.

Watanabe, Shinji, Boyer, Florian, Chang, Xuankai, Guo, Pengcheng, Hayashi, Tomoki, Higuchi, Yosuke, Hori, Takaaki, Huang, Wen Chin, Inaguma, Hirofumi, Kamo, Naoyuki, Karita, Shigeki, Li, Chenda, Shi, Jing, Subramanian, Aswin Shanmugam, and Zhang, Wangyou. The 2020 ESPnet update: New features, broadened applications, performance improvements, and future plans. In *2021 IEEE Data Science and Learning Workshop, DSLW 2021*, 2021. doi: 10.1109/DSLW51110.2021.9523402.

Yu, Peng, Chen, Kaijiang, Lu, Lie, and Seide, Frank. SEARCHING THE AUDIO NOTEBOOK: KEYWORD SEARCH IN RECORDED CONVERSATIONS. Technical report, 2005.

Zhou, Bolei, Khosla, Aditya, Lapedriza, Àgata, Oliva, Aude, and Torralba, Antonio. Learning deep features for discriminative localization. *CoRR*, abs/1512.04150, 2015. URL http://arxiv.org/abs/1512.04150.

7. Appendix

7.1. Word Alignment Example

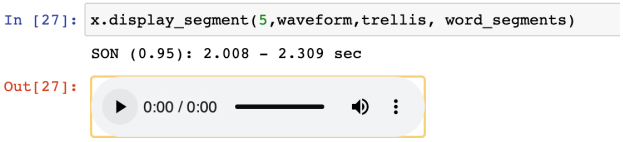


Figure 7. Example of predicted start and end timestamps of a keyword, generated by the CTC algorithm described in section 3.2. Audio contains the keyword uttered, examined via Jupyter Notebook and IPython module.

7.2. Keras ASR Model Architecture

7.3. ESPnet Adversarial Example Pipeline

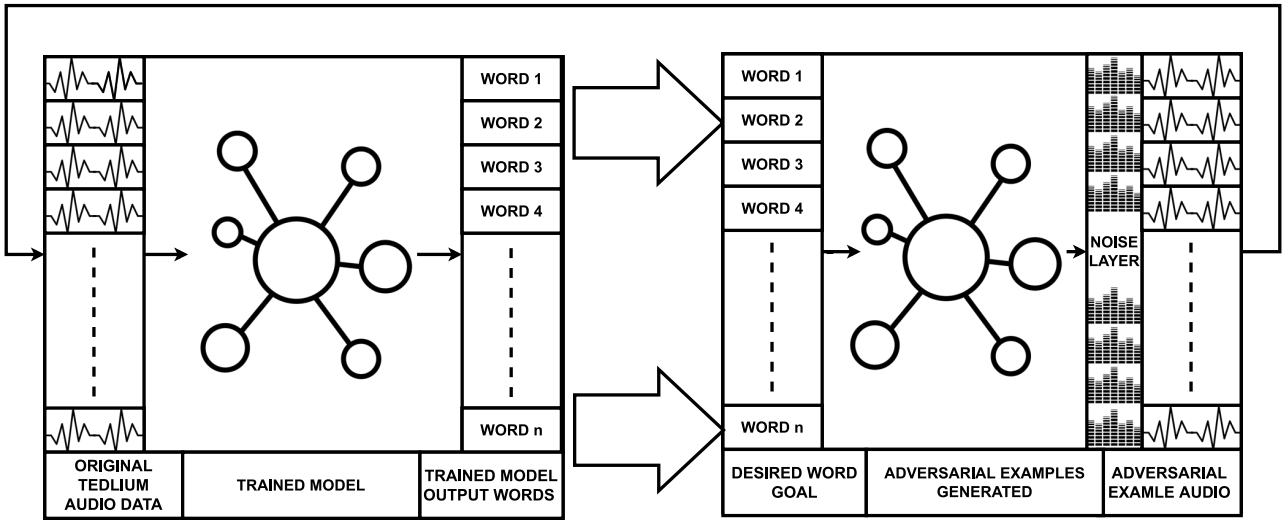


Figure 8. Pipeline for creating adversarial examples. The ESPnet model is first trained on the original data. Iteratively, noise is added to an audio file and then predicted. This cycle continues until the model predicts the desired target sentence.

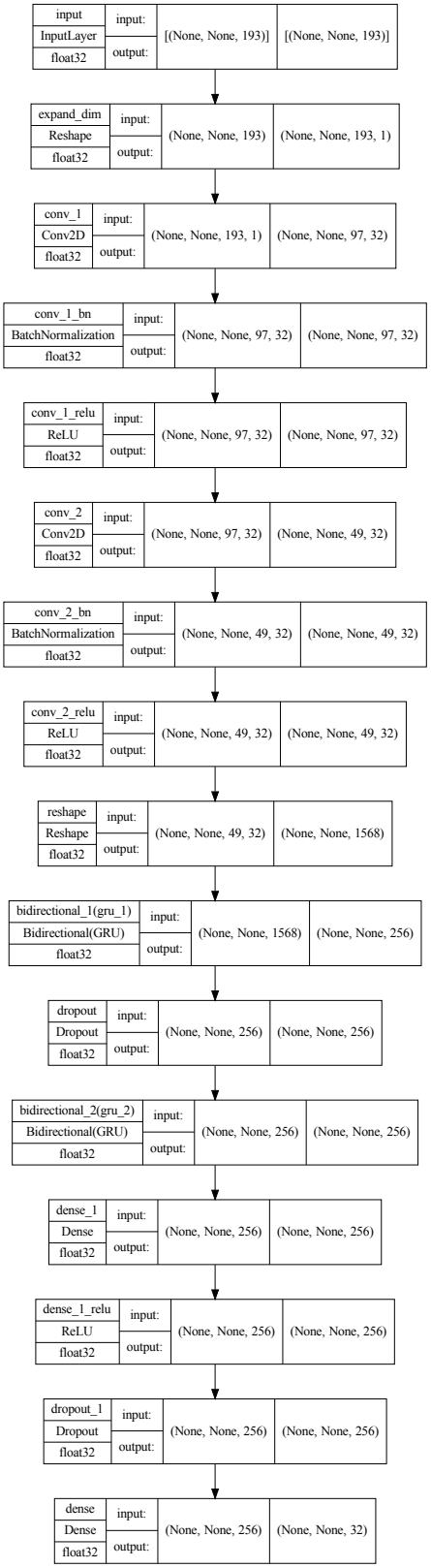


Figure 9. Keras ASR Model Architecture. The attempted model uses 2D CNN, RNN, Connectionist Temporal Classification (CTC) loss, and other regularisation techniques like Dropout and layers like Batch normalisation.

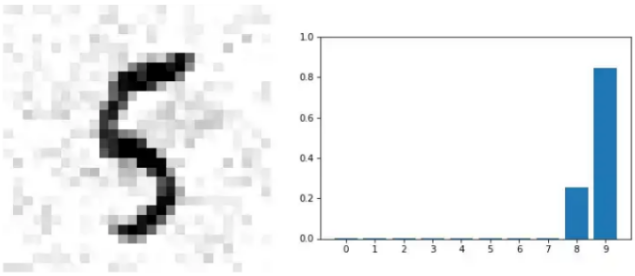


Figure 10. Taken from Geng and Veerapaneni (Geng & Veerapaneni, 2018). Example of a targeted adversarial example (here, an image) and the model's prediction.

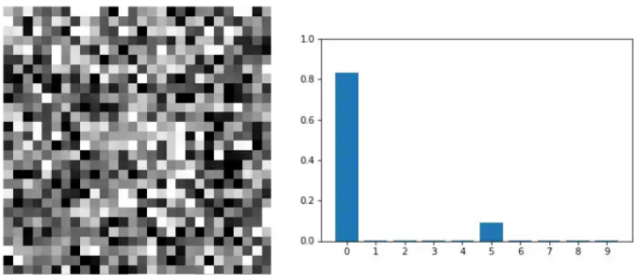
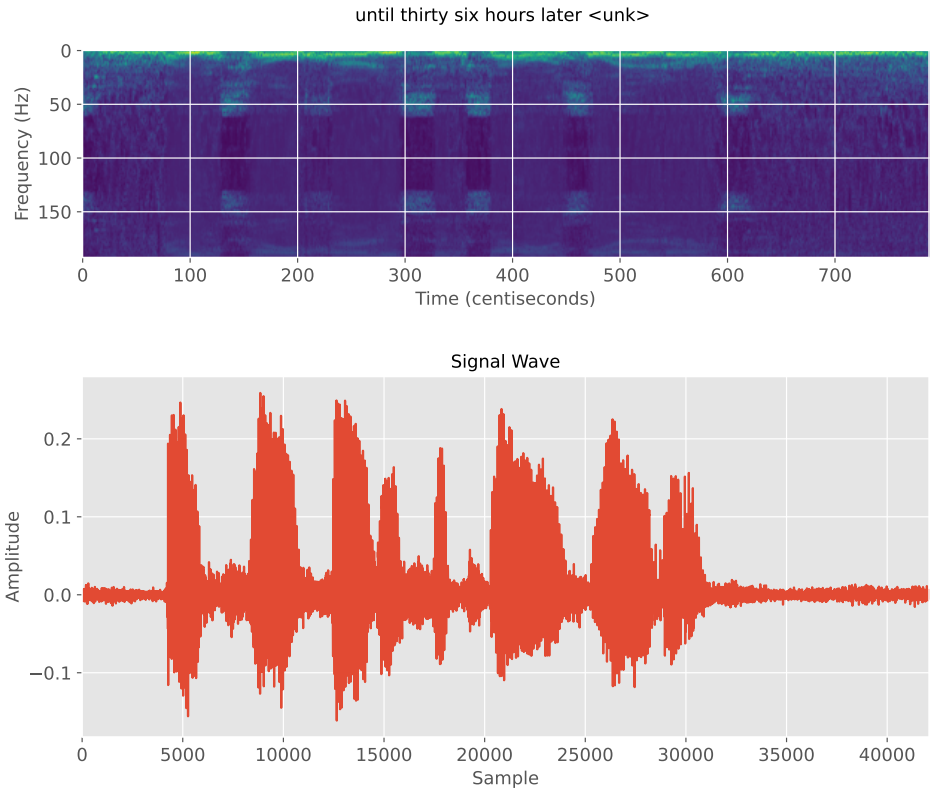
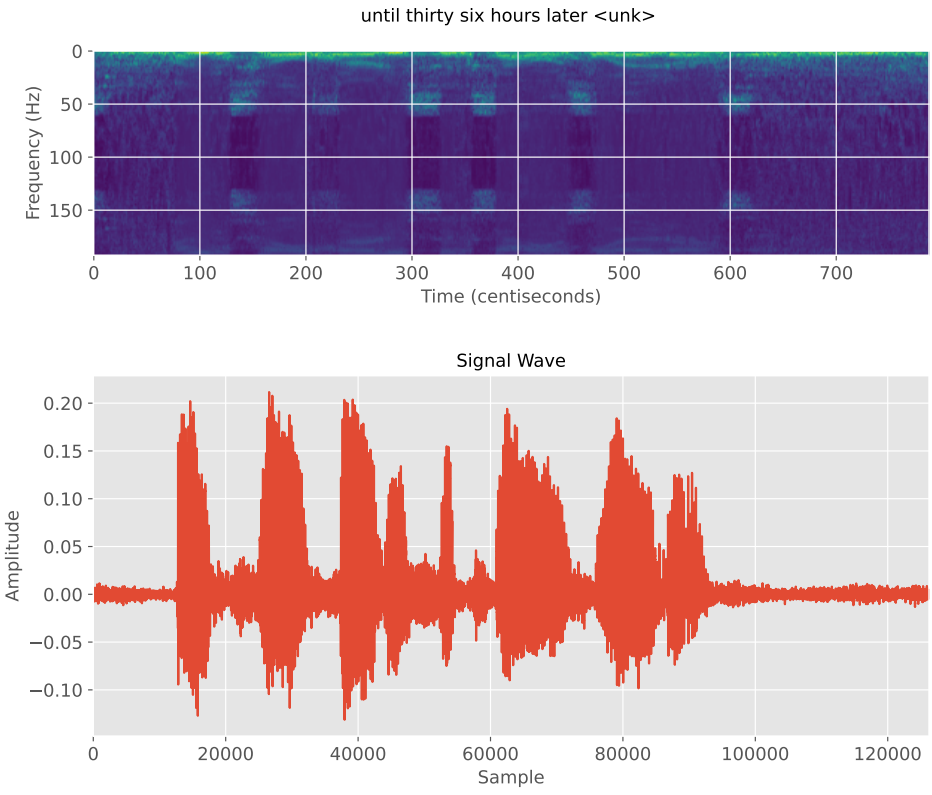


Figure 11. Taken from Geng and Veerapaneni (Geng & Veerapaneni, 2018). Example of a non-targeted adversarial example (here, an image) and the model's prediction.



(a) Spectogram(Top) and Audio wave(Bottom) of one TED audio segment.



(b) Spectogram(Top) and Audio wave(Bottom) of an adversarial example, targeted towards the same TED audio segment as the subfigure a).

Figure 12. Illustration representing the comparison between the original TED audio and its corresponding adverserial example. The illustrations look exactly similar, except that the adversarial example's audio is longer. When listening and comparing the audio clips, the words uttered in the adversarial audio is deep-pitched and slow, but they resemble to the words mentioned in the transcript.

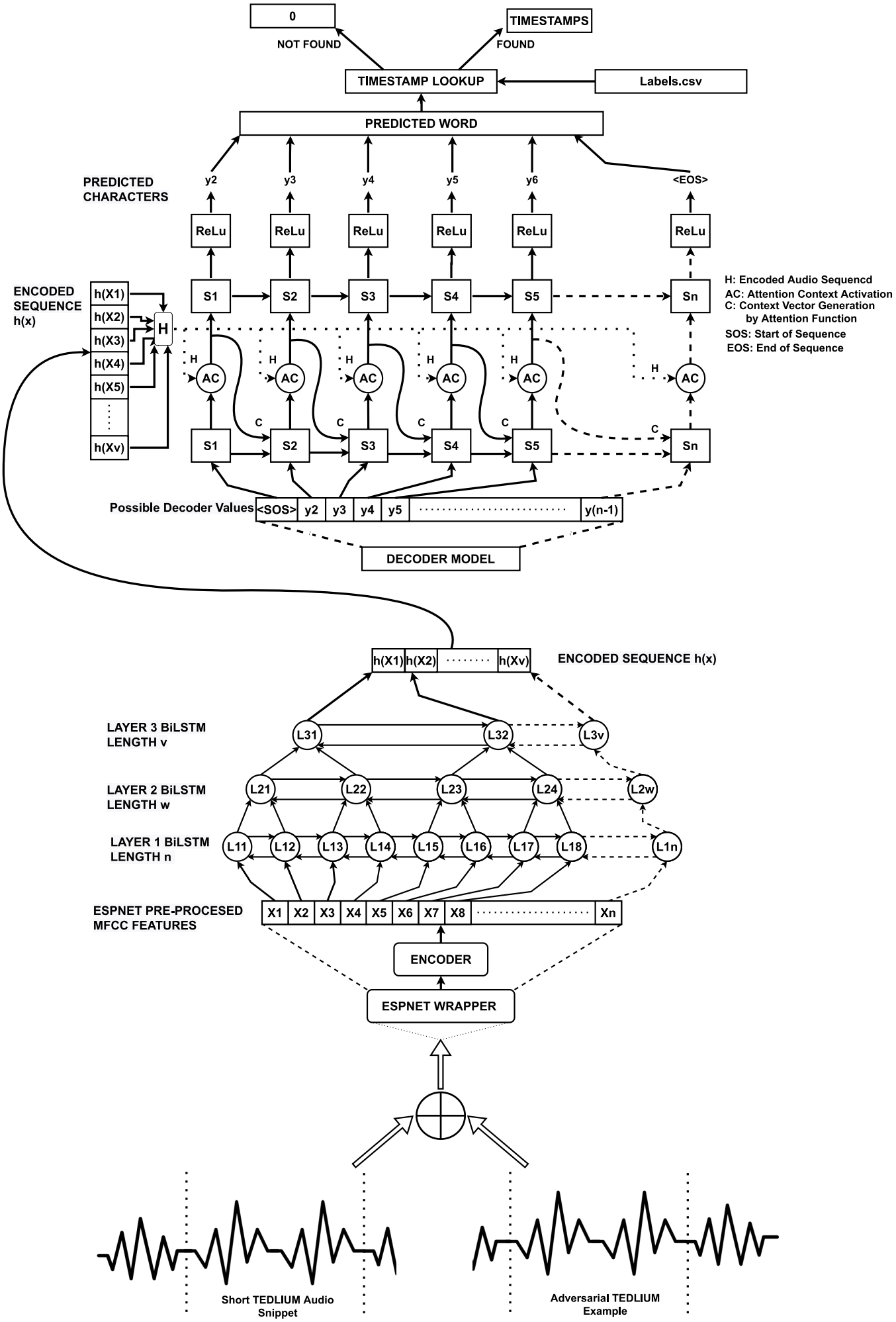


Figure 13. Full sized figure depicting the ideal model for audio keyword search using ESPNT