Deep Learning Methods for NLP & Speech Analysis

Orya Spiegel
orya.sp@gmail.com

Roi Birger
rbirger123@gmail.com

Dolev Abuhazira
dolevictory@gmail.com

Dor Azaria
dorazaria@gmail.com

Nadav Epstein
nadavepstein1@gmail.com

Or Trabelsi
ortrsa@gmail.com

Shlomo Glick
shlomog12@gmail.com

Gilad Shotland
giladsh123@gmail.com

Adi Lichy
adi.lichy@gmail.com

28 May 2022

Contents

Abstract

In recent years Machine Learning (ML) and Deep Learning (DL) have shown great results in classifying data, with its images or vectors and etc. In this project, we will show the capability's of DL on recorded speakers of different languages turning them into images and classifying them using Convolutional Neural Network (CNN). We have created five models of CNN each model has a different neural network and classification, in addition, each model was trained on a different dataset that best suit its classification. The models have been set up on a server and can be reached by using a browser.

1 Introduction

From the human voice we can understand a lot about the speaker such as the mood they speak of such as positive, neutral or negative. The age and gender of the speaker can also been notice by one voice, also we can tell the language and accent they speak in. Each one of this things can be notice by an Artificial Intelligent (AI). In this project each group create a CNN model which is able to classify the audio of the speaker to said categories:

- Gender & Age: In section 2 explain the usages of CNN model to classify gender and age. Creating two binary CNN model classifiers, the first gender which classifies the speaker as either male or female, the second age classifies the speaker as young or adult.
- Mood: In section 3 explain the usages of CNN model to classify mood. Creating a multi-class CNN model classifier, which classifies the mood of the speaker to positive, neutral or negative.
- Language & Accent: In section 4 explain the usages of CNN model to classify language and accent. Creating a multi-class CNN model classifier for the language in addition create multi-class CNN model classifier for each language to classify the accent.
- Language: In section 5 explain the usages of CNN model to classify language. Creating a multi-class CNN model classifier for the language which can classify 30 languages.

2 Gender and Age Classification from Sound (Orya Spiegel, Roi Birger)

This article was written as a final project for the 'Deep Learning methods for Language analysis and Sound' course in B.Sc. Computer Science and Mathematics in Ariel University.
The course is run by Dr. Or Anidjar.

Link to our code: https://github.com/Orya-s/DeepLearningSound

2.1 Abstract

Automatic sound analysis is becoming more relevant in the modern world. Extracting information from the speaker can serve many purposes for varied kinds of companies. For example, marketing companies can fit the right sort of products for a customer, after recognizing the age and the gender of the speaker. In addition, digital assistants like Apple's Siri are programmed to respond only to the owner of the device. This kind of technologies can improve user experience significantly and save a lot of money and resources for companies.

Keywords: **Deep Learning, CNN, Classification from Sound, Sound Classification, Machine Learning, age, gender.**

2.2 Introduction

Age and gender play important roles in social interactions. Languages reserve different grammar rules for men or women, and very often different vocabularies are used when addressing adults compared to young people. Despite the basic roles these attributes play in our day_to_day lives, the ability to automatically estimate them accurately and reliably from sound, using artificial intelligence models, is still lacking.
In our project, we used a Deep Learning model to classify age and gender from sound. Deep learning is a type of machine learning and AI that imitates the way humans gain certain types of knowledge. To use sound files for Deep Learning we used the Wav2Vec model which converts Wav files to torch vectors. After achieving the vectors, we used a Neural Network to make the predictions. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.
The network we used is Convolutional Neural Network (CNN), which we will elaborate on further when explaining the model structure.

2.3 Datasets

We started by using the Common Voice dataset (Common Voice (mozilla.org)). Common Voice is a publicly available voice dataset, powered by the voices of volunteer contributors around the world. People who want to build voice applications can use the dataset to train machine learning models.
Common Voice contains the next features: client id, path, audio, sentence, age, gender, accent, and language. We used the audio as our feature and age and gender as our labels. After preprocessing the data, we got 31,134 different attributes.
While training our models we came to the conclusion that adding more data can significantly improve our results, so we added a big part of the vox_celeb dataset.
VoxCeleb is an audio-visual dataset consisting of short clips of human speech, extracted from interview

videos uploaded to YouTube. There are two VoxCeleb datasets, we used VoxCeleb1.
VoxCeleb contains a file called Metadata, which contains the features for each speaker – ID, name, gender, nationality and whether the speaker is in the train (dev) set or the test set.

Structure of metadata-

	A	B	C	D	E
1	VoxCeleb1 ID	VGGFace1 ID	Gender	Nationality	Set
2	id10001	A.J._Buckley	m	Ireland	dev
3	id10002	A.R._Rahman	m	India	dev
4	id10003	Aamir_Khan	m	India	dev
5	id10004	Aaron_Tveit	m	USA	dev
6	id10005	Aaron_Yoo	m	USA	dev
7	id10006	Abbie_Cornish	f	Australia	dev
8	id10007	Abigail_Breslin	f	USA	dev
9	id10008	Abigail_Spencer	f	USA	dev
10	id10009	Adam_Beach	m	Canada	dev
11	id10010	Adam_Brody	m	USA	dev

Figure 1: Structure of metadata

The train and test sets are two separate files arranged in the same way. We will explain the structure of the train set as an example. The train set is a folder called vox1 dev txt, which contains a single folder called txt. In txt there are multiple folders, each one represents a different speaker by a unique ID, which connects the content of the folder with features in the metadata file. In each of the ID folders there are a few other folders, each one represents a different YouTube video, with the URL as the name of the folder. In each of these folders there are a couple of txt files, each one represents a segment of the video by frames in which the speaker is talking. Each second in the video is represented by 25 frames.
To get the audio segment, we took the first frame and the last frame in each txt file and used it to cut the right part of the video. When a speaker had a few separate parts in the video where he was speaking there were a few txt files in the folder. Taking the consecutive frames which represent a part of the video allowed us to get the audio we wanted from each speaker.

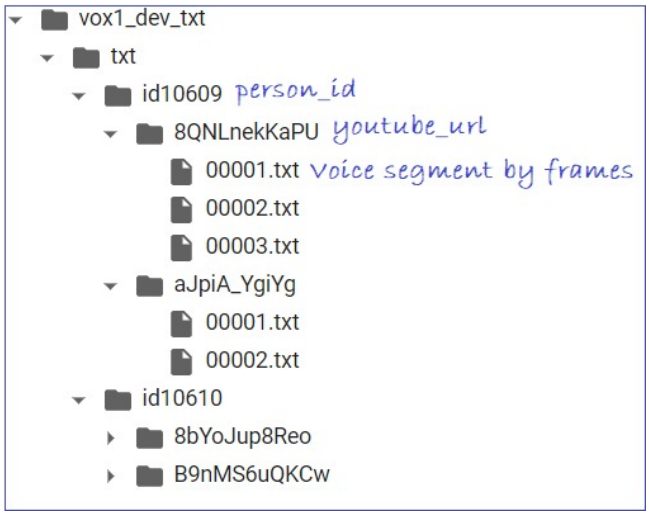


Figure 2: VoxCeleb structure

Eventually our data contains:
Gender - [101,094 Male, 54,272 Female]
Age - [3,118 Teen, 44,434 Adult]
Total sum of data - 155,366

2.4 Preprocessing

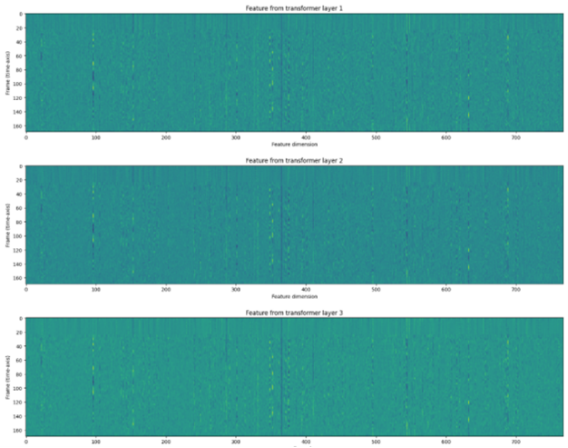
After gathering the data, we needed to represent it in the right way for it to be used as the features for our models. We started by cutting off the header of each wav audio file. The header of a wav file is 44 bytes long. To train our model we decided to use 3 seconds long wav files, so we cut each of the audio files we downloaded to 48,000 bytes, as we made sure the sampling rate of the audio is 16,000.
To prepare the wav files to be used as our features we needed to convert the wav to a different form of data, that can be used by our models to train and test, so we used Wav2Vec.

2.4.1 Wav2Vec

Torchaudio's Wav2Vec is a convolutional neural network (CNN) that takes raw audio as input and computes a general representation that can be input to a speech recognition system. The process Wav2Vec performs on the data looks like the following:

1. Extract the acoustic features from audio waveform. The returned features are a list of 12 tensors, each tensor is the output of a transformer layer.

   An example of the first 3 tensors in a plot -



2. Uniting the tensor layers to create the final tensor. Wav2Vec2 model provides method to perform the feature extraction

   and unite them in one step with the following command -

```
with torch.inference_mode():
    emission, _ = model(waveform)
```

   The emission is the tensor vector we will use to represent each wav file in our model.

Once we had the tensors for each wav, we wanted to combine each tensor with its matching labels for gender and age, and then unite all our data together. To do that we used the Pickle library.

2.4.2 Pickle

The Pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process where a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, where a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.
After combining each tensor with it’s labels in a tuple, we created a list that unites all the tuples and used pickle to serialize it. This allowed us to gather all our data online and then to only download the pickle file to train and test our model locally.

2.4.3 Load data

Once we downloaded the pickle, we wanted to arrange our data in a way that will allow us to create two separate models, one to predict the gender of the speaker and the other to predict the age of the speaker. To do that we created three lists, when loading each tuple in the pickle we added the tensor to a list which we called X_data, as it gathers the features. We added all the gender labels to a list called y_gender, and all the age labels to a list called y_age.
To make sure our model only gets numerical values we converted the labels according to these dictionaries-

```
self.genders = {"male": 0, "female": 1}
self.age = {"teen": 0, "adult": 1}
```

Then to start training and testing our models we used the train_test_split function from sklearn.model_selection library to create train, test and validation.

We split the data in the following way-
Train – 70%
Test – 20%
Validation – 10%

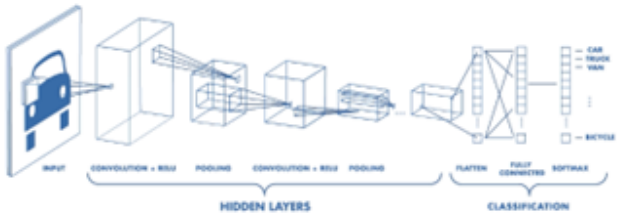
An example for splitting in the gender model-

```
X_train, X_test, y_train, y_test =
    train_test_split(np.array(X_data), np.array(y_gender), test_size=0.20)

X_train, X_val, y_train, y_val =
    train_test_split(np.array(X_train), np.array(y_train), test_size=0.15)
```

2.5 CNN Model

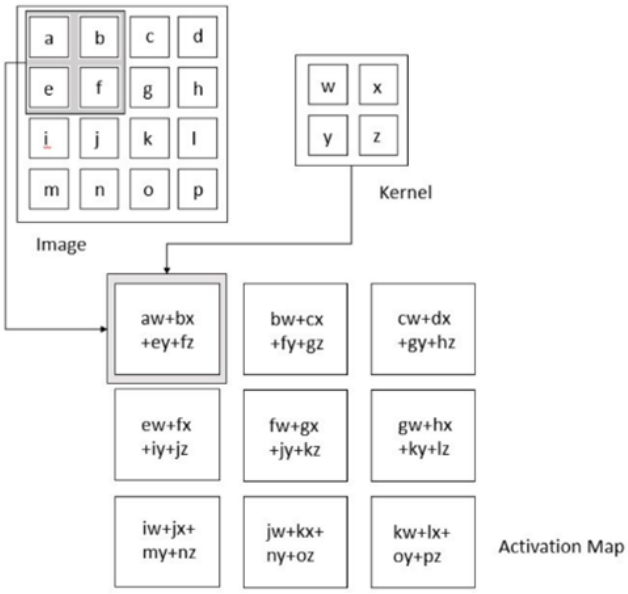
CNN is a convolutional neural network that specializes in processing data that has a grid-like topology, which is why it is commonly used for models based on images. A CNN typically contains convolutional layers, pooling layers, and a final prediction layer.

CNN example for image classification-



The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field (our data).

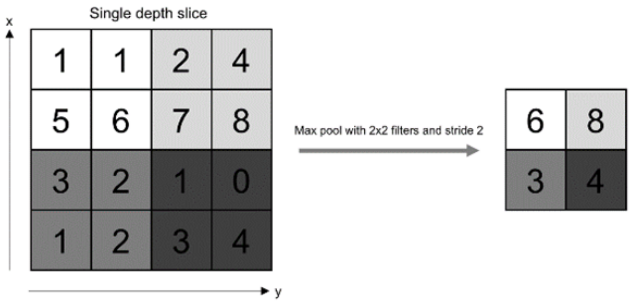
An example for how convolution works –



Using a kernel smaller than the input allows us to store fewer parameters, which not only reduces the

memory, but also improves the statistical efficiency of the model, by detecting the meaningful information. The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This too helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights.
There are several pooling functions, the most popular process is max pooling, which returns the maximum output from the neighborhood.

An example of max pooling –

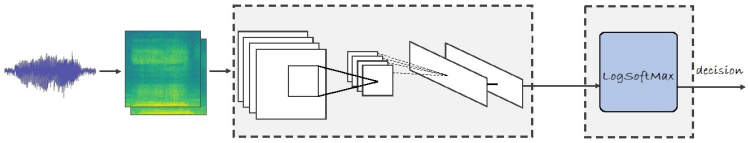


Since convolution is a linear operation and audio inputs (or images) are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map. In our model we used. The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function f(k)=max (0, k). In other words, the activation is simply threshold at zero.

Another thing we used in our model is Dropout. The Dropout layer randomly sets input units to 0 with a frequency of rate (fraction of the input units to drop) at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.
The Dropout layer only applies when training is set to True such that no values are dropped during inference.

2.6 LogSoftmax

Softmax is a mathematical function which takes a vector of K real numbers as input and converts it into a probability distribution of K probabilities, proportional to the exponential of input numbers. After applying softmax, each component will be in the interval [0,1], and the components will add up to 1, so that we can interpret these values as probabilities.
Log softmax (the log of softmax function) is advantageous over softmax for improved numerical performance. logsoftmax also has the effect of heavily penalizing the model when it fails to predict a correct class.
After training the data on our models we came to the conclusion that adding two more layers to the gender model can improve the results drastically.



2.7 Training

To train our model we take the train set and use the CNN model we created to study the data in a way that will allow us to make prediction on future data that the model didn't encounter.
We started by using the class_weight function from sklearn.utils, because in both of our models the data distribution is unbalanced. The issue with that is that the algorithm will be more biased towards predicting the majority, and the algorithm will not have enough data to learn the patterns present in the minority class. The purpose of class_weight is to penalize the misclassification made by the minority class by setting a higher-class weight and at the same time reducing weight for the majority class.

```
class_weights = class_weight.compute_class_weight('balanced',
                                                  classes=np.unique(data.y), y=data.y)
class_weights = torch.tensor(class_weights, dtype=torch.float)

criterion = torch.nn.CrossEntropyLoss(weight=class_weights)
```

The loss function we used is Cross Entropy. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Each epoch
In each epoch we started with training the data using batches. Each batch is going through the network and returns an output, which is then updating the loss for that epoch, using our cross-entropy loss function. To update the network's weights, we used the Adam optimizer, Adam is a popular algorithm in deep learning because it achieves good results fast and is suited for problems with a lot of data.
Then we moved on to validation and test, in which we used batches again (to increase the speed of iterating over the data and computing the results).

After using our network to make the prediction we updated the loss and calculated the f-score for the epoch. To be able to calculate the f-score we needed to "fix" the output from the network, as we use logsoftmax. Using a function to get the index of the maximum argument from each sample, we were able to get the prediction from the output.

```
# f1 score
f1 = f1_score(y_true, y_pred, average=None)
f1_avg = f1_score(y_true, y_pred, average='weighted')
print("f-score: ", f1, "  f-score avg: ", f1_avg, "\n")
```

The reason we calculated the f-score of each label on every epoch, is to get a better sense of the quality

of the model then we could have gotten from only calculating the accuracy. The reason for this is that in both of our models the data is divided in a very unbalanced way, and f-score is a good method for this kind of challenge because it returns the average of the precision and recall.

```
F1 = 2 * (precision * recall) / (precision + recall)
```

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, which means how good the model is at predicting a specific category.
Recall (also known as sensitivity) is the fraction of relevant instances that were retrieved, which means how many times the model was able to detect a specific category.

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}}$$

2.8 Inference

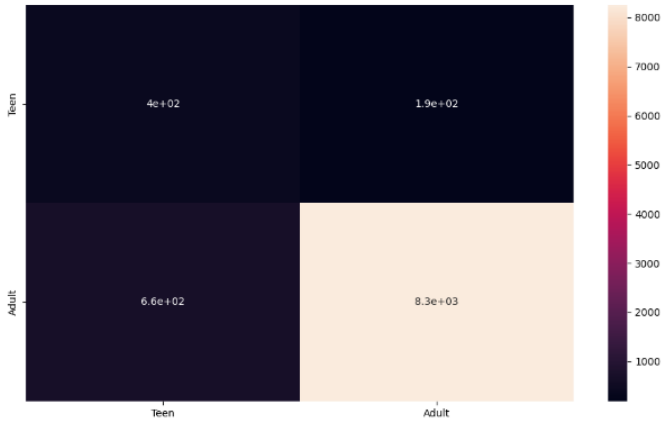
To make sure we eventually select the best models we created an inference testing. We gathered over a 100 different wav files, that the model has never seen before (not even the test set). Our goal was to make sure we choose the models that make good predictions on completely new data. For every model we tested we made sure to run over all the wav files and using the predictions we found the models that worked best.

To make sure we use our model in the right way to test the new wav files, the inference function is preprocessing the new data in the same way we preprocessed our datasets. We first load the model we want to test and make sure it's in evaluation mode (the training is set to false), then load the audio files and make sure the sampling rate is the same as it was in the training, if it isn't the same we use the resample function. Then we use Wav2Vec to convert the wav file to a tensor that represent it, and only then we can use the model to get the prediction. Here too we use the argmax function to make sure we don't return the numbers the CNN returns, but the label which has the highest probability.

```
model = torch.load("age_Binary_Model-e_11_Weights.pth")
model.eval()
waveform, sr = torchaudio.load(wav_path)

if sr != 16000:
    waveform = torchaudio.functional.resample(waveform, sr, 16000)

bundle = torchaudio.pipelines.WAV2VEC2_ASR_BASE_960H
wav_model = bundle.get_model().to(device)
embedding, _ = wav_model(waveform)

with torch.inference_mode():
    print(inf(embedding, model))
```

2.9 Results

Age – model **:**
Accuracy of the network = 91.04
f-score (weighted) average = 0.92
f-score by class = [Teen - 0.487, Adult - 0.950]



The best results for the age model were from epoch 11, with 0.0001 learning rate.

Gendermodel **:**
Accuracy of the network = 89.40
f-score (weighted) average = 0.89
f-score by class = [Male - 0.920, Female - 0.842]

The best results for the gender model were from epoch 36, with 0.0001 learning rate.

2.10 Challenges

The main challenge we had was the data. To create a good classification model, we need the data to have similar balance to the balance in the real world. We started the project with the Common Voice dataset, which caused a problem for us in both models. The number of males in this dataset is 4 times the number of females.

The other problem with this dataset was that the age label was classified to 9 groups, teens to nineties, but some of the age groups had no data in them at all and some had only a few samples in them.

We first tried to solve this problem by converting the age classification problem from a 9 classes problem to a 4 classes problem, as the first 4 classes had the most amount of audio files in them (teens to forties). But the problem was that this still left us with a very unbalanced data, and the amount of data was not big enough to make this kind of classification work well.

We also noticed that some of the models we created with these 4 classes had returned results that weren't too far from the truth, for example a person classified as 'teens' returned 'twenties' as a result, or a person classified as 'thirties' returned the result 'forties' or 'twenties'. This was better then the first version of 9 classes but still not good enough.

To deal with both of these problems we added data from a new dataset. Adding Voxceleb helped improve the result of the gender model significantly. The addition improved the balance of the data, the male were now only twice the amount of female. To try and make the results even better, we added two more layers to the gender model, as the classification was still difficult to do with this distribution of the data. Adding the layers helped increase the result of the model and got us the best results we got so far.

The problem we encountered with the new dataset was that it wasn't classified by age, so what we decided to do was to turn the four classes we had so far to a binary problem, with 'Teen' and 'Adult' classes. With this change we also needed to manually label a part of the Voxceleb dataset to increase the amount of data and add as many Teens as we could to make the classification easier. After labeling a big part of the data we had two classes for the age classification, still very unbalanced but with more data for the model to study. Making these changes increased our results significantly and got us good model.

2.11 Conclusions

This project had a lot of challenges for us, as sound analysis is new to us and required studying a lot of new methods and information. At the end we got two new models to analyze sound, which can make a great change for a lot of business or individuals. We learned how to take a neural network base and make it something completely new that has a lot of potential.
Thank you for reading!

3 A Fine-tuned Wav2Vec2.0 For Speech Emotion Recognition (Dolev Abuhazira, Dor Azaria)

3.1 Abstract

This Article written as part of the course "Deep Learning Methods for Language and Voice Analysis" by Dr. Or Anidger from Ariel University.
In this article we explore the fine-tuning on Wav2Vec2.0 [4] a pre-trained model for classifying different emotions that received by voice.
In addition, we will present the different ways we performed for handling the data, from constructing the sample space to turning them into input for the model using different libraries.

3.2 Introduction

Our emotions are an integral part of our lives, they usually reflect our opinions and our desires. It can be seen that there is a clear connection between emotions and human behavior. The ability to identify emotions provides many options in various aspects such as social media, market research, customer experience and more. As part of our work, we have fine-tuned a pre-trained model that trains on raw audio data and learns its contextual representation. With the help of fine-tuning, we can tweak this model to perform our task which is emotion recognition. In this article, we will describe the architecture of our model, and also, we will describe how we construct the dataset with elements of signal processing.

3.3 Data

3.3.1 Dataset Collection

To train our model for emotion recognition, we searched across the web for data that would represent the same classification problem. After a long search on Huggingface [3] and Kaggle, we've found the following:

- RAVDESS Dataset [6]
  Contains 1440 audio files from 24 professional actors in North American accent. Speech includes calm, happy, sad, angry, fearful, surprised, and disgusted expressions and the song contains calm, happy, sad, angry, and fearful emotions.

- TESS Dataset [5]
  Contains 2800 audio clips of 2 women expressing 7 different emotions anger, disgust, fear, happiness, pleasant surprise, sadness, and neutral.

- URDU Dataset [1]
  The dataset contains 400 audio files from 38 speakers that took part in Urdu talk shows, speech includes angry, happy, and neutral.

3.3.2 Class Distribution

Since the amount of data we collected was insufficient, we thought of merging the various classes into three classification classes to increase the accuracy of the model.
Now our model will have to classify the voice into the following three classes:

1. **Positive** - a mixture of happy and surprise.
2. **Neutral** - a mixture of neutral and calm.
3. **Negative** - a mixture of anger, fear, sad, and disgust.

After collecting the dataset and merging the classes we got the following distribution:

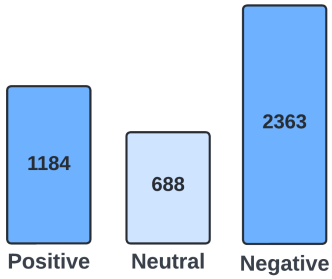


Figure 3: Class distribution

3.3.3 Data Pre-processing

The first stage of pre-processing was to collect datasets and merge all of them into one. For each audio file, we classified its label to one of the three classes (Positive, Neutral, Negative) and added it into the Panda's DataFrame by the following form: (file path, label).
For the second stage, we imported each WAV file from our DataFrame using ***touchaudio.load()***.
This ***load()*** function returns a waveform and sampling rate.
Before we're moving to the next stage, we need some theory about audio signal processing.
These audio samples are represented as time series, where the y-axis is the amplitude and the x-axis is the time series of the waveform. The amplitude's function measured the change in pressure around the receiver (a microphone).

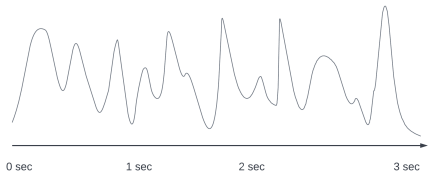


Figure 4: waveform of audio

One of the parameters of the ***touchaudio.load()*** function represents the number of frames to load. In this work, we trained the model with WAV audio files ranging from 2 to 3 seconds long and 16,000 samples per second. In signal processing, sampling is the reduction of a continuous signal into a series of discrete values.

The sample rate is the number of samples taken per second. Each sample contains amplitude (loudness) information at that point in time. A sound is made by tens of thousands of frames that are played in sequence to produce frequencies. The WAV file contains a sequence of frames and multiple channels of data. Usually, a single microphone can make one channel of sound, and a single speaker can receive one channel of sound. For our model, we sampled mono-channels audios to 16kHz. Every audio is 3 seconds long, so for every audio, the number of audio signals is:

$$Seconds \times SampleRate \times Channels = 3 \times 16,000 \times 1 = 48,000$$

A high sampling frequency results in less data loss but higher computational expense, and low sampling frequencies have higher data loss but are fast and cheap to compute:

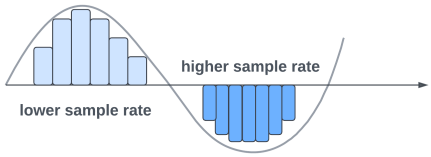


Figure 5: comparison between high and low sampling frequency

Since we fine-tune the Wav2Vec2.0 model, we must be sure that the size of the input is 48,000 samples, therefore we decided that each recording would contain exactly 48,000 samples (3 seconds long, 16k samples in a second). Since there are recordings shorter than 3 seconds, we had to fill in the missing samples by selecting random samples from the recording. This method fairly preserves the imagery of the voice.

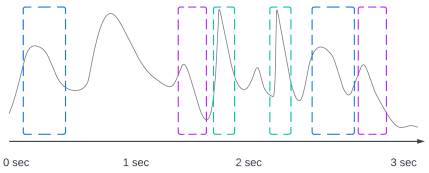


Figure 6: handling short recordings by padding samples randomly

3.3.4 Inference and fine-tuning

As mentioned above, we fine-tune a pre-trained model Wav2Vec2.0. The model was developed by researchers from Facebook.AI for the speech recognition tasks.
The architecture of the model is encoder-decoder. Encoding means converting data into a required format, and Decoding means converting a coded message into intelligible language. The encoder of the model consists of several blocks containing a temporal convolution neural network to find a representation of the raw audio data and the output of the feature encoder are fed into transformer architecture which finds the contextual representation of the feature representation. the transformers do that by the self-attention layers. the self-attention mechanism allows the inputs to interact with each other and find out to who they should pay more attention to. The model was pre-trained and fine-tuned on 960 hours of Librispeech (a collection of approximately 1,000 hours of audiobooks) on a 16k sample rate speech audio.

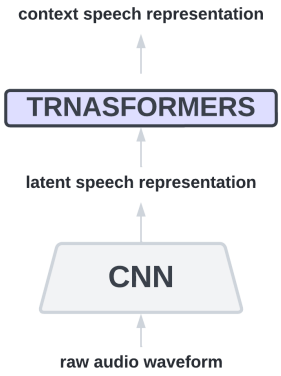


Figure 7: A very basic look at the architecture of Wav2Vec2.0

So far, we have collected 4235 raw voice data. After briefly explaining the pre-trained model, We rebuilt our data in such a way that our raw data is now contextually described by the Wav2Vec2.0 model.

Given the input to the model, the final output contains 12 vectors that describe a partial context from the context of the entire audio, for our purpose we using a matrix containing these 12 vectors.
this matrix describes the context of the audio received as input to the model.

Now, each of our samples from the raw data set will be passed as input to the Wav2Vec2.0 model and thus we will get a data set so that each example in it describes the context which is predicted by the model.

for each embedding matrix we make normalization feature scaling to bound the values between $[-1, 1]$. Let max be the greatest value and min be the lowest value from a set of features F. each feature value $1 \leq i \leq |F|$ is bounded between $[-1, 1]$ using the following formula:

$$F[i] = \frac{2\times(F[i]-max)}{(max-min)} + 1$$

Finally, our new dataset contains 4235 normalized contextual representations for our old dataset, the size of each record - tensor in the dataset will be:

[1,149,32]

3.4 Architecture

Since the output of the pre-trained model is an embedding matrix that represents the context of the raw audio data, we chose to use a convolutional neural network whose purpose is to find specific patterns that will describe the emotion in the sound data. In each convolution layer, there are neurons so each neuron is a matrix that describes the different relationships between the data obtained from the input.

Our model consists of eight convolutional layers, and four pools layers whose purpose is to reduce the size of the matrices obtained from the convolution layers, one of the reasons for using these layers is the ability to optimize the calculation method and save processing time.

In each such layer, we performed batch normalization since it contributes to the increase in model training speed. The activation function in each layer is $ReLu$ and we chose to use it because its derivative is faster to compute in the back-propagation process.

In the last layer of convolution, we have implemented a regularization method called dropout whose purpose is to prevent overfitting during learning.

Also, after the data passes through the convolution layers they are fed to a linear neural network with one hidden layer.

The output layer of the neural network contains three neurons that specify the classes to which we want to classify the data.

Since we want to know what the probability is for each of the emotions we want to classify, the activation function in the last layer is $logSoftMax$, and our loss function is $CrossEntropyLoss$.

3.5 Training

After we made an inference and saved the new data set in a way that it would contain the context of each sound file, to train the model we divided the data as follows :

Train	Validation	Test
3389	423	423

Figure 8: splitting data into train, test, and validation sets

Since the division of the classes is not uniform, we have added weights to the classes which causes the model to penalize the misclassification made by the minority classes by setting a higher class weight and at the same time reducing the weight for the majority classes.
the weights that we defined are :

$$Positive : 1.99, Neutral : 3.0, Negative : 1.0$$

We first set the model to execute about 20 Epochs, so that in each Epoch we fed the model with a batch of size 32 , and the model train on learning rate with size 0.0001.
Because we used to validate our model performance during training with the validation set, We saw that there was overfitting in the eight epoch, so we decided to stop training in a method called Early Stopping, hence the amount of epochs performed in training is seven.

3.6 Results

After training the model, the moment has come to verify whether, given new examples of the model from the test group, he will be able to classify the emotion of the voices inputs.
After performing the test, we received that the accuracy of our model stands at 70.43%. percent. And the accuracy of the model for each classes:

1. **Neutral** - 81.481%.
2. **Positive** - 72.881%.
3. **Negative** - 66.80%.

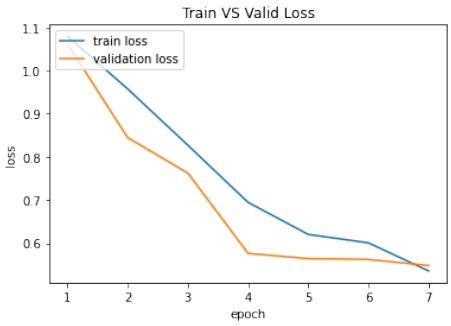


Figure 9: Early stopping method that perform on the seventh epoch

3.6.1 Performance Evaluations

To test our models, we use three different measurements to evaluate the performance: Recall , Precision, F-1 Score, and confusion matrix.

Emotion	Precision	Recall	F1-Score
Negative	0.9	0.654	0.7575
Neutral	0.427	0.785	0.5531
Positive	0.621	0.731	0.6715

Figure 10: The various measurements for model training

The measurements that we present above, are based on the confusion matrix which is a two-dimensional matrix that allows visualization of the model performance. It is a summary of the results of predictions on our classification problem task.

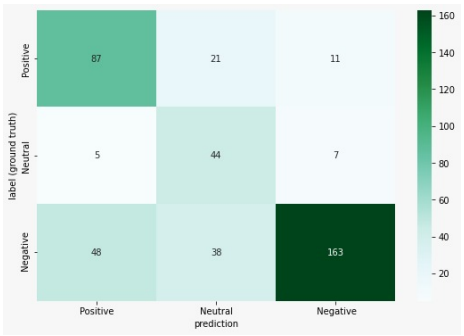


Figure 11: Confusion Matrix measurement

3.7 Conclusions

We have developed the advanced deep learning model for recognizing emotions in speech. We applied this learning model and demonstrated their results in the data set of RAVDESS, TESS and URDU, and got pretty good results regarding the fact that we had difficulty collecting a lot of data that representing the same classification problem.

In addition, during our work we came across many examples of sound files that were tagged differently from each other even though they sound pretty much the same.

We believe that if we had access to a wider and larger data set as well as a more balanced one, we would have achieved much better results.

As part of the course we learned to process raw audio and the principles behind signal processing, in addition, we learned an important principle which is Transfer learning which means taking the relevant parts of a pre-trained machine learning model and applying them to a new but similar problem.

3.8 Access to our project

All project components are at the following link :

Voice Emotion Recognition

For lite version click - here

Our project consists of various files: model architecture, training notebook, embedded data processing notebook, and Pth. file indicating the model we saved after training and the one that gave the best results for us. In addition, the main file through which you can run the program.

How to run?

1. Clone the voice emotion recognition repository
2. Enter the following command into your terminal :
   python3 Main.py

3.8.1 Required tools

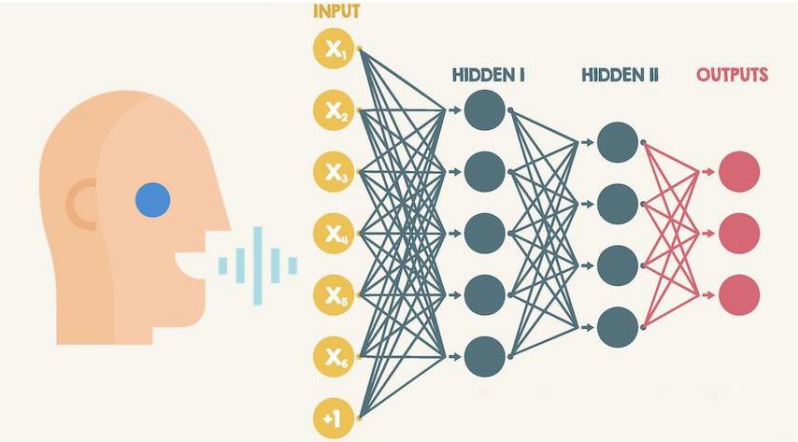
Python3, Numpy, Pandas , sounddevice, Pytorch , torchaudio , matplotlib and more [2] .

4 Language & Accent Classification (Nadav Epstein, Or Trabelsi)

This article is written following a project we worked on during our first degree in computer science
The course focuses on Deep learning techniques for voice classification.
our lecturer for this course is Dr. Or Anidjar the CEO & Chief Data Scientist @ libonea.ai.
Link to our code :Our Project Github

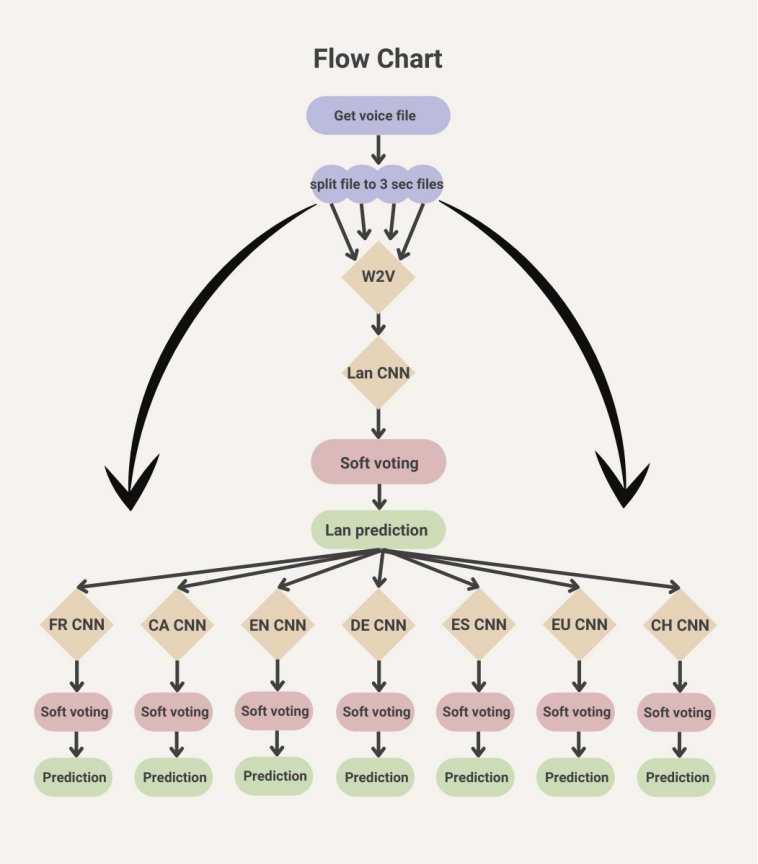
4.1 Abstract

In today's world we can learn a lot about a person just by hearing his voice. Our goal is to identify just by voice a person language and accent which can help in a variety of sections such as Marketing, Scamming, Security and etc.
In order to do that we used a model based on deep learning that classifies the language and after that the accent of that language.



4.2 Introduction

Most of the companies that deals with customer service via telephones records all the conversation's their employees had every day which means that every day they get another thousands of record hours, You're probably ask yourself "what are they doing with it?" , apparently not much the absolute majority of those recording just sitting there taking a lot of space without any use. That's where we get in by using our model (and other voice classification models) we can use that data to help to improve the customer service. How? using our model we can predict in less then a second what language the person on the other side of the phone is talking and not just that , also his accent. when we have this information we can use it to our own advantage by selecting a representative of the company that can speak the other person's language and even in the same accent, By doing that we already improved our customer service because we wont have problems of communication between the customer and the company representative and it only took one second. this is just one out of many problems our project can solve and improve. How does it work? We built out model using tagged data (Records that we already know what language and accent is heard) and we built a model using deep learning methods and trained him with the tagged data so he can learn how to tell just by voice what is the spoken language and accent. If deep learning is a new subject to you please click here before continuing.



4.3 Methodology

4.3.1 Gather Data

In order to start a project from this kind first of all we need to find tagged data in order to do that we visited Hugging Face website which is a data-set library for easily accessing and sharing data-sets and evaluation of audio tasks (also they have evaluation metrics for Natural Language Processing and computer vision).

4.3.2 Deal with Data

Once we have the data we want to split it to test and train we did it in a way that test was 20% of the data and train was 80% from the data. By doing it we made sure that after we trained our model it wont know the answers to the test data. It is important to check that we have all the languages in both test and train.

4.3.3 Build Model

After we have the data we need to build the model for the train (We will expand on our model later on in this article) basically the model will train him-self by the tagged data that we have and after we trained him long enough we can try and test the data that we kept in the test data and it will show us how good is the model.

4.3.4 Train Test and score

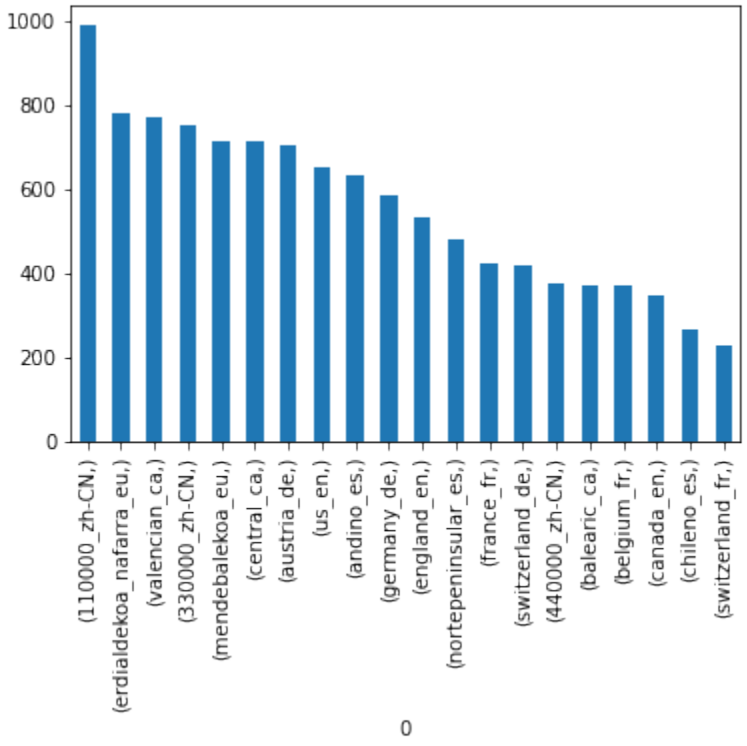
How do we know if the model had trained long enough? When we see that our percentage of the accuracy using F1 score(We'll expand about this method later on in the article) isn't going up any more or its going down we try to test the model. When we test the model we get the prediction and then we check the F1 score. Now we compare the Train F1 score with the Test F1 score and we check if there's over-fitting if there isn't we can stop train the model.

4.3.5 Using the model

After the model is finally tuned and ready and we satisfied with our score we now can use it to predict new audio files. We'll get a new audio file as a live record from microphone or as a previous recorded file. Now the file will go through a series of preparation step and after that will enter the model and we'll get the desire prediction.

4.4 Data Set

As explained before we took our data from Data Set Source. at first we wanted to predict accent from all the languages at the data-set (59 languages) but we saw that the majority of the languages didn't had tagged accent so we could only use 7 languages (French , Catalan, English, German, Espaniol, Basque and Chinese). Each of those languages has few accents which means our classification is between 20 accents. We had a problem downloading all the data-set at once because of it size so we used "Datasets" library this library provides accesses to stream many data-sets from Hugging Face which help us to get all the records one by one and we can stop at any time after we have enough data.



4.5 Pre-Processing

Before we enter the data to the model it has to go through few steps.

1. **Balancing the data manually**: Because the data set had few languages with a lot of tagged accent and the others with much less we put a higher limit so we wont have to much from one language and it will cause over-fitting.
2. **Division by Language**: Each sample was sent to her own pickle divided by language.
3. **Casting data to 16,000 sampling rate**: We needed to make the data uniform so every audio had been cast to 16,000 sampling rate.
4. **Save & load the data using pickle** : Pickle is a binary protocol for serialization we use this protocol to save and load efficiently our data/model. we need it so we can work on the data continuously and save checkpoints along the data processing and the model train.
5. **Division to Train & Test** : We decided to divide ton 20% test and 80% train
6. **Division to three seconds** : We planned that the minimum time for a record will be 3 seconds and for every additional 3 seconds our model will preform "soft voting" which will improve our prediction significantly .
7. **Wav2Vec**: Now days there is a very popular approach to train complicated models with small data-sets. Using pre-trained models that were trained on huge data-sets and find tuning them to your specific purpose. This approach called transfer learning as we transfer the knowledge from the pre-trained model. Wav2Vec is one of the current state of the art models for automatic speech recognition. Because this model works on unlabeled data it help us to create a smart matrix that represents the audio file such that audio files that are similar will represented with similar matrix's.

4.6 Our Model

Our flow composed from 8 models. the first model predicates the language and for each language we have there own model that predicates the accent. All of the models architecture are the same and were built as follow :

4.6.1 Architecture

We have six convolution layers for each of the five first layers we will preform batch normalization and relu and for must of them we will preform max-pooling after the six layer we'll preform drop-out that will help us to avoid over-fitting. the last layer is fully connected that connects to log soft max. A little bit about the method we used:

1. **Convolution layer**: Is a deep learning algorithm which Uses the convolution operation and proved to be very efficient while preforming on computer vision, NLP and voice.
2. **Batch normalization**: Normalize the value for each mini batch this technique stabilizes the learning process.
3. **Max-Pooling**: We use this method to down-sample the feature map by taking the max for every X*X sub matrix as we chose.
4. **Relu**: Activation layer which gives the value X to every X bigger then 0 and zero to every X smaller then 0.
5. **Drop out**: We use this method to prevent over fitting by dropping X percentage of the input units to zero.
6. **Log Soft-Max**: This is a common function that helps to classify for multi classes by normalize the result. This function takes the exponent of the result and divided by the sum of the exponent of all the classes. The Log-softmax help us to spread the values to larger range that improves numerical performance and gradient optimization.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$
$$LogSoftMax = \log \sigma(z_i)$$

4.6.2 Loss function

Our goal is to minimize the loss to zero(with the help of the optimizer). We use Cross Entropy loss function in order to do that. This function measures the model error by

4.6.3 Optimizer

The goal of an optimizer is to minimize the loss. We choose the Adam optimizer this optimizer is a fusion of two common optimizers SGD and RMS. This optimizer uses moving average to get to the minimum faster and increase very small steps and decrease very large steps.

4.6.4 Hyper Parameters

The main hyper parameters for the model are :

1. **Learning rate** : this parameter decides the step size the optimizer will make in each iteration in order to minimize the loss.
2. **Epochs** : number of iteration over all of the data.
3. **Batch size** : number of samples for each iteration.
4. **Class weights** : for imbalanced data multiple the weights for the low class size.

For every Class (language) we find the optimal parameters (from our attempts) and every one of them had different parameters in consideration towards the class data.

4.7 Final Prediction and Scoring

1. **Prediction**

   After the model is trained we can evaluate the model quality and predict new samples. As explained in the pre-processing section we take each audio file and split it every three seconds. Every sub sample (long 3 seconds) is sent to the model and we get the model prediction by a vector of probabilities in the length of the classes(this is the output of the softmax).The model prediction will be the argmax of this vector. How do we know which one to choose? In a perfect world all of the predictions should be the same but we know that the reality isn't perfect so we use soft-voting method, soft voting combining the probabilities of each prediction and picking the prediction with the highest total probability. We return the top 3 accent prediction because of leak of data the results of the model wasn't as we expected.

2. **Scoring** We choose to evaluate our model using F1 score and confusion matrix (as you can see in the last page)

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision+recall}}$$

4.8 Previous attempts (failures and solution's)

4.8.1 Data without filter

At first tried to use one model to predict accent (without predicting language first) we even took every language without checking whether it has enough tagged samples. After we tried to train the model we saw that we had a lot of languages which have only few tagged samples

How did we solve that problem? We deleted every accent that had less then 200 tagged samples.

4.8.2 Imbalanced data

We had few accents with huge tagged samples and few with only few hundreds of tagged samples which cause to over-fitting.

How did we solve that problem? We blocked the tagged samples that will only take 1000 samples maximum from each accent. but the data was still a little bit unbalanced so we decided to use class weights which helped us to give more weight to the classes with lower data size.

4.8.3 Too many classes

When we tried to predict the accent straight away without knowing the language first we saw that our model wasn't strong enough (because of the leak of data) and we didn't had good result.

How did we solve that problem? We decided to create a model for each language that will predict her accent and we created another model that predicates the language first. Then instead of 20+ classes we needed first to classify between 7 languages and then predict from each language her accent which is usually between 2-3 classes. that resulted a big improve on our model.

4.9 Development Environment

Because the process of training model is very rough we and our computers aren't strong enough to run train model in time. Therefor we used google colab that give us access to free gpu and faster training. google colab also have verity of options if you are a paying member such as more gpu, you can train the model while the computer is shut down and etc.

4.10 Libraries that we used

pandas , pickle, torch (torchaudio), numpy, seaborn, cv2.

4.11 Appendices

1. **classification report**:

   EN Test accuracy score: 65 %
   EN Test F1 score for 1: 73 %

   CA Test accuracy score: 58 %
   CA Test F1 score for 1: 56 %

   FR Test accuracy score: 51 %
   FR Test F1 score for 1: 64 %

   EU Test accuracy score: 75 %
   EU Test F1 score for 1: 78 %

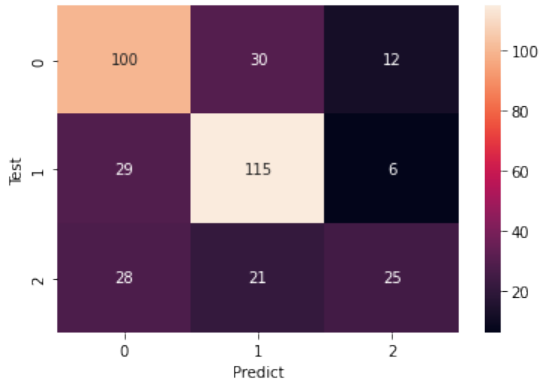
   ZH Test accuracy score: 66 %
   ZH Test F1 score for 1: 66 %

   ES Test accuracy score: 65 %
   ES Test F1 score for 1: 63 %

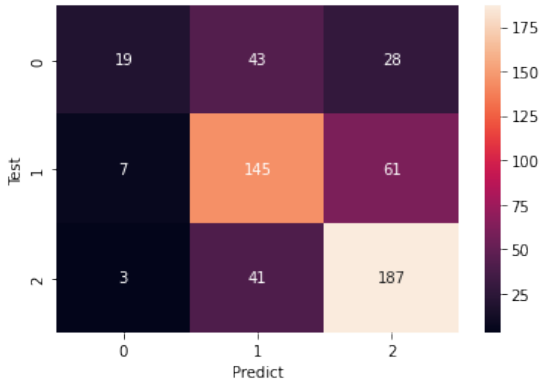
   DE Test accuracy score: 50%
   DE Test F1 score for 1: 53%

   Lan Test accuracy score: 82 %
   Lan Test F1 score for 1: 85 %

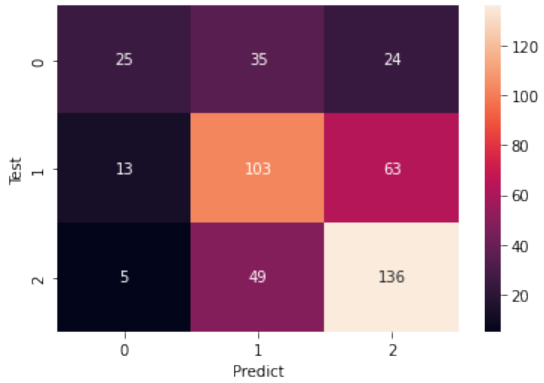
2. **Confusion matrix**:



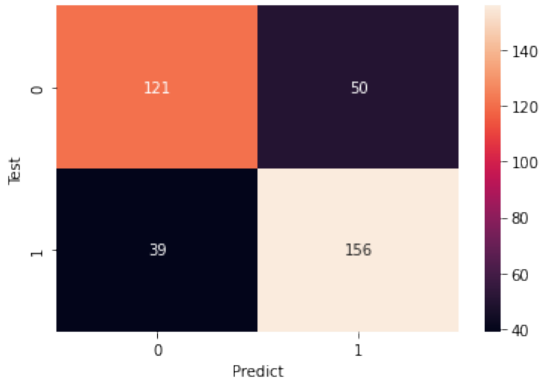
(a) EN



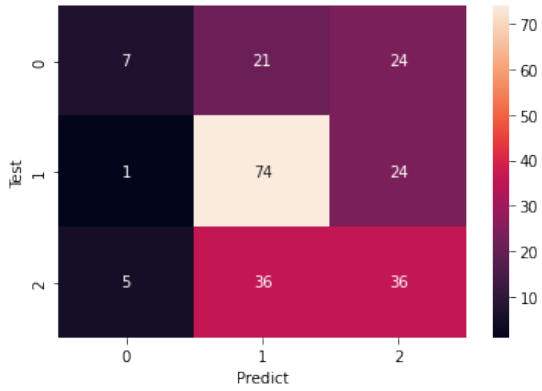
(b) CH



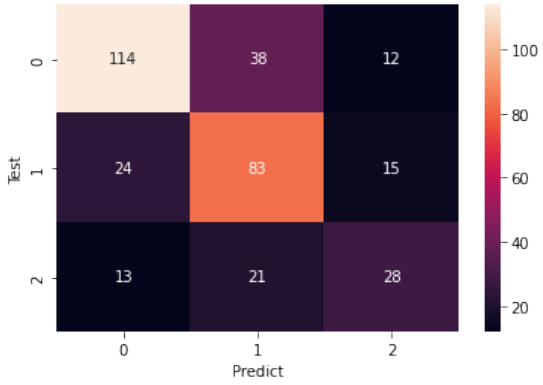
(c) CA



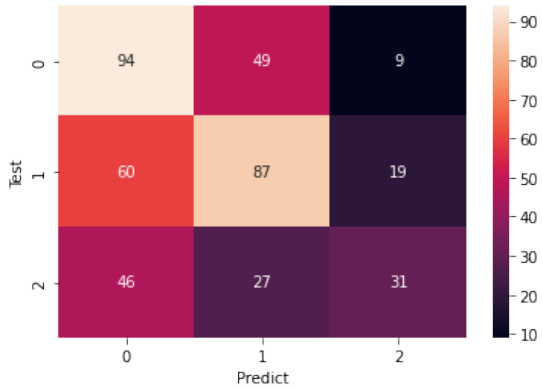
(d) EU



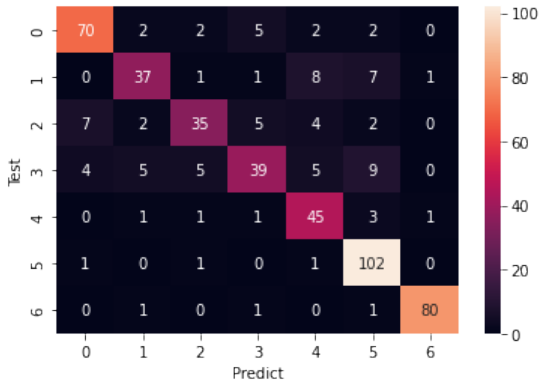
(a) FR



(b) ES



(c) DE



(d) LAN

5 Language Classification using Transfer Learning (Shlomo Glick, Gilad Shotland)

5.1 Abstract

Transformers for voice and text embedding are one of the most discussed topic in Deep Learning and Artificial Intelligence in the last 5 years. The ability of learning contextual content can be useful for many advanced tasks which looked impossible few years ago. This paper represents a project of examining the option of using trained transformer model with not so deep and complex neural network for classifying spoken language in a recording.

Keywords: **Deep Learning, Transfer Learning, Common Voice**

5.2 Work Flow

The project included few levels and steps that brought the idea to life - working model.

5.2.1 The Levels:

1. Downloading, Cleaning and Building a Data Set
2. Building a Model
3. Training - Measuring Results - Hyper Parameters Tuning Cycle
4. Getting the Final Results
5. Building an Inference Extraction Interface

5.3 Building The Data Set

There are many data sets all over the Internet that contain voice recordings of people speaking. The project main purpose was identifying the spoken language, therefore the chosen data set was the Common Voice data set which is well labeled.

Common Voice contains thousands, and in some cases tens of thousands of recordings of more than 50 languages. This volume was too big for the needs of this project, for example, English class contained more than 90 GB of recordings. This amount of data makes it hard twice - with storing the data, and additionally, if a training session with consideration of a high percentage of the data is wanted, the training will take long long time - days instead of few hours for each training session. Therefore partially downloading of the data was needed.

Partially downloading of the data was allowed only with PyTorch's API, which provides abstraction for pulling data samples from a data set, sample by sample. Configurations problems of the project contributors' local environment with automating the download, led to downloading the data through Google's Colaboratory Notebook, and saving the data that was downloaded in the environment to Google Drive.

The project aim was examining the ability of using the output of embedding Deep Learning model as an input to a pretty simple Neural Network, therefore after downloading a sample from the data set, getting it's embedding matrix was needed.
For storing the whole data after getting embedding matrices for the samples, Python's Pickle Module for packing and unpacking Python objects was chosen. Pickle provides the API for saving a pack to a pkl formatted file with calling a simple function.
Downloading the whole data using a free cloud service caused a problem of non infinite computing time, and the downloading process suffered from disconnection from server due to inactivity while downloading. This problem was solved by not saving the data to the Drive only when the whole data set was downloaded, but rather saving each window of 300 sampled and labels to a pkl file, than unpack all the files to one data set for training.

The project's model contained some fully connected layers and a decision function, therefore the flexibility of convolutions wasn't accountable for training and pulling reference from the model for changing size of the input.

Common Voice didn't provide uniform length of the samples, while some of the samples were 1 or 2 seconds length, others 4 or 5, and others more than 10, and for getting uniform length of samples on one hand, but effective length which can provide some knowledge about which language is spoken, the length of 3 seconds per training sample was chosen. Each sample that was pulled from Common Voice and was more than 3 seconds length, was sliced to 3 seconds windows, with overlapping of 1 second between each two adjacent windows, as the slicing was made by taking different parts from the WAV array that represented the recording sample.

For examining the ability and behaviour of embedding output, as mentioned as the main goal of the project, the embedding model which was chosen is Meta's WAV2Vec2, which is one of the state-of-the-art trained models that are available for using on the internet.

5.3.1 The data set building pipe line was as follows:

1. Download a sample
2. If the sample is more than 3 seconds length - slice it to 3 seconds windows
3. For each window, pull an embedding matrix from WAV2Vec2
4. For each 300 embedding matrices that were produced - save it to pkl file

5.4 The Model

5.4.1 Motivation

Embedding matrix includes the proportion between the ith speech segment and the jth speech segment in the i,j place. One way of referring to this form of data is by looking at the m x n matrix as a vector of size m

multiplied by n. Another way is by referring it as a signal that was received by the embedding component, WAV2VEC2 in this project case.

Hence, the signal processing famous operation - Convolution, which is used widely in this field for various tasks, such as blurring or finding edges in images, can be applied on embedding matrices. Convolution kernels as learnable parameters has been proved widely in the last 10 years as an effective techinque for contextual computing and efficient dimension reduction. Therefore the architecture that was chosen for the task is a Convolutional Neural Network.

5.4.2 Building The Model

At first the model was build 6 Convolutional layers and one fully connected layer, with batch normalization and max pool layers after each convolution. After few sessions of training with different hyper parameters and getting low performance from the model, we decided to expand both convolution and fully connected parts of the network to 8 convolution layers with 5 Batch Normalizations and 5 down sampling layers, 3 fully connected layers and 3 drop out layers.

5.4.3 The Model Architecture

1. 96 Convolution Kernels of 5x5 with padding value of 1
2. ReLu activation
3. Batch Norm
4. Max Pooling of 3x3 with stride of 2
5. 256 Convolution Kernels of 5x5 with padding value of 1
6. ReLu activation
7. Batch Norm
8. Max Pooling of 3x3 with stride of 1
9. 384 Convolution Kernels of 3x3 with padding value of 1
10. ReLu activation
11. Batch Norm
12. 256 Kernels of 3x3 with padding value of 1
13. ReLu activation
14. Bacth Norm
15. 256 Convolution Kernels of 3x3 with padding value of 1
16. ReLu activation
17. Batch Norm

18. Max Pooling of 2x2 with stride of 1
19. 64 Convolution Kernels of 2x2 with padding value of 1
20. ReLu activation
21. Max Pooling of 2x2 with stride of 2
22. 32 Convolution Kernels of 3x3 with stride of 1
23. Drop Out of 0.5
24. Average Pooling
25. Fully Connected layer of 1024
26. ReLu activation
27. Drop Out of 0.5
28. Fully Connected layer of 512
29. ReLu activation
30. Drop Out of 0.5
31. Softmax

5.5 Training

The data set was split to 3 parts - Train, Validation and Test, where the Validation and Train were taken from the data defined as 'Train' by Hugging Face and the Test was taken from data defined as 'Test' - this fact insures that the final results measuring was very close to reliable, although learning a distribution of data, with considering all the edge cases that can be in the distribution, without over fit the model, is a pretty hard task.
Through all the training sessions, the Train set was used for the actual training - gradient updates, and the Validation set was used for examining the results of a certain training session with certain hyper parameters. This way of examining results on the validation set and not on the test set, prevent the model of learning sub distribution with finding hyper parameters that fit to it, and enable the model to generalize the approximation of the classification function that it tries to find.

For fast training the optimization algorithm that were chosen was Adam, and the gradient updates were done with a loss computed over the data.

5.5.1 The training routine was as follows:

1. Choosing Hyper Parameters in a concluded way from the previous training session - decrease or increase learning rate or regularization factor.
2. Training
3. Examining results with the Validation set each epoch

4. Saving the model's weights each 5 epochs
5. After all epochs are done - deciding if taking the weights of one of the epochs model with other hyper parameters or training a new model from scratch

As a result of lack in local hardware for neural network training - GPU, the training session were executed on Google Collaboratory. This working environment supplied free usage of GPU, but had the big con of time limitations.

The time limitations for using Collab's GPU for training mean that slow learning is not possible - the server will get disconnected for the next 24 hours.

5.5.2 First Architecture

In this project we've trained models from with two architectures. First architecture was given by the supervising Lecturer. After few sessions we've decided to take the same architecture and add some convolution and fully connected layers for getting better results.

The Architecture:

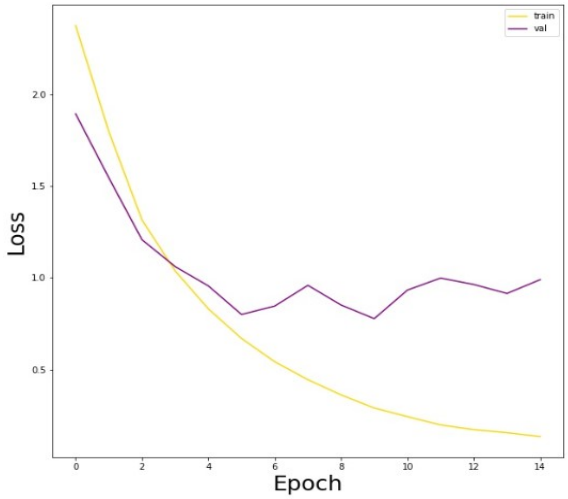
1. 96 Convolution Kernels of 7x7 with stride of 2
2. ReLu Activation
3. Batch Norm
4. Max pool of 3x3 with stride of 2
5. 256 Convolution Kernels of 5x5 with stride of 2
6. ReLu Activation
7. Batch Norm
8. Max pool of 3x3 with stride of 2
9. 384 Convolution Kernels of 3x3 with padding of 1
10. ReLu Activation
11. Batch Norm
12. 256 Convolution Kernels of 3x3 with padding of 1
13. ReLu Activation
14. Batch Norm
15. Max Pool of 5x3 with stride 3x2
16. 4096 Convolution Kernels of 9x1
17. ReLu Activation
18. Fully Connected Layer of 1024

19. Log Softmax

In the beginning the model had a big difficulty in learning, and the loss didn't show any signs of decreasing. Analysis of the data that was downloaded showed that the amounts for every language aren't equal, so there are many situations where the mini batch that was randomly picked contained many samples from group of certain languages with big amounts of data, and yet from other languages with small amount of data had no samples in the batch, therefore they almost didn't affect the gradient - and the model didn't learn how to generalize them.

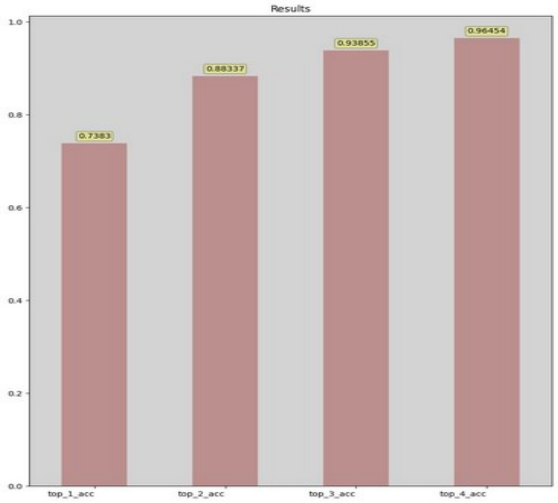
At first to overcome this problem, we added to the training's input a class weights vector, aiming to cause a bigger penalty to the gradient when the model missclassified sample from a small class. Each class got the weight 1/n where n is the number of samples from the class
For measuring results in the project and analyzing the performance, the model was tested in accuracy scale - percentage of correct classifying, in top 1 accuracy, top 3 accuracy (if the right spoken language was in the top 3 probabilities the model predicted) and top 5 accuracy.
After feeding the model with class weights, the model showed low performance due to to poor amounts of data - up to 1000 samples for each language.

The results of the model were 34.3 accuracy percent on the Validation set, when the number of languages the model was trained on was 60.
As shown in researches on Machine Learning and Statistics, as we increasing the amounts of data we can surely expect increase in the performance of the model Hence we've decided to pull more data for training sessions - up to 5000 samples for each language,and dumping languages with less than 1000 samples. That didn't gave any significant improvement in the performance, so we pulled more data, at this stage each class had between 1000 and 5000 samples for training and validation, where the model was trained on 45 languages.
The increment in the data caused also increment in the results, and the model scored accuracy of 52 percents of accuracy.
The next stage in the project was trying to get higher performance by training only on languages with more than 3000 samples, this attempt didn't make any progress, although we've trained on 29 languages. Another attempt made by using L2 Regularization in the training, which also didn't gave any improvements.
Analyzing the results showed that most of the missclassifications of the model caused by confusion by different dialects of the same language, such as different dialects in Chinese.
Therefore we've decided to unite dialects of same languages and adding convolution and fully connected layers, as described above. The model was trained on 30 languages, and gave 76 percent of top 1 accuracy, when the final model was trained in learning rate of 0.0001 and 17 epochs without L2 Regularization - we took the model from the 9th epoch.
Final Amounts of data were 179223 samples for Training Set, 25604 samples for Validation Set and 30739 for Test Set



5.6 Final Results

The final model gave on the Test set top 1 accuracy of 73 percents. For that model we wrote an inference extraction file, which giving the option of entering a recording to the model. The script will slice the audio recording into 3 seconds chunks, extract embedding matrices from Wav2Vec2 and will get for each matrix an inference from our model, and will give the three languages with the highest average score from the predictions of all the matrices, for measuring the amount of the confidence of the model's prediction.



5.6.1 The Languages which the model was trained on:

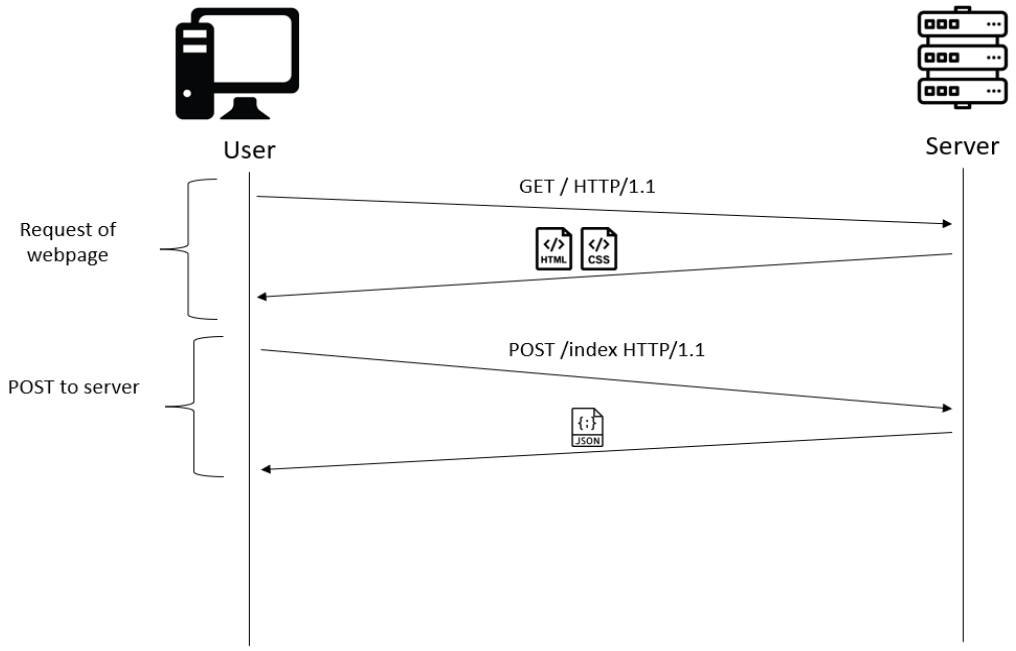
1. Estonian

2. Portuguese
3. Tatar
4. Welsh
5. Arabic
6. Catalan
7. German
8. Spanish
9. Basque
10. English
11. French
12. Esperanto
13. Italian
14. Kabyle
15. Rwanda
16. Russian
17. Chinese
18. Latvian
19. Indonesian
20. Serbain
21. Slovenian
22. Tamil
23. Romansh
24. Greek
25. Hungarian
26. Mongolian
27. Thai
28. Sakha
29. Frisian
30. Persian

Project's Github Link: https://github.com/shlomog12/Spoken_language_identification

6 Server and Website (Adi Lichy)

In this section will explain the client side which is the website side and the server-side. The models are setup on a server that allows the users to connect by browser and send or record an audio file, in addition to selecting the models which the users want to be used. the server will return the result for each model to the users with the final prediction and the probability of each class being chosen.



6.1 Server

The Server is programmed in Python using Flask which is a server library. The server is setup on a selected IP and port which then the user can connect via a browser sending a request, and the server responds by sending HTML and CSS files. The server can be run on Linux or Windows operating system.

6.1.1 Running Models

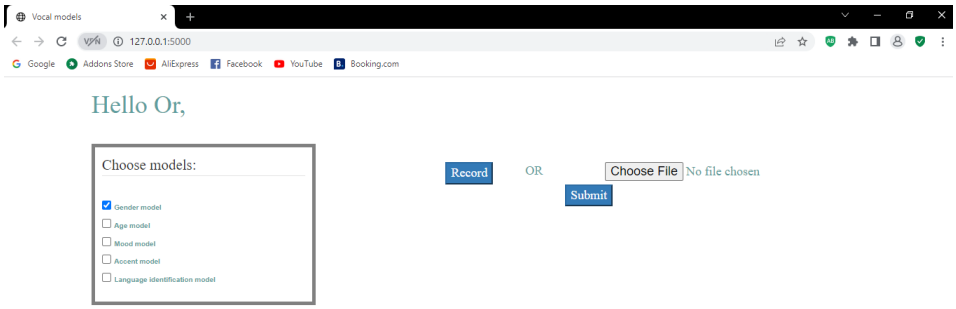
The models are loaded on the server, upon receiving a form submit from the user which contains the audio file and models selected, the server will check if the audio file is of WAV file type if not will change it to WAV using AudioSegment which is part of pydub. Also will check for the selected models to run. The audio file is read and transformed by a WAV2Vec model to a vector. The vector is sent to each model which has been selected by the user. After all the models are done and have given their predictions, the server will send the response by a JSON file to the user.

6.1.2 Inference

The server read the audio file using torchaudio and takes 1 channel with a total of 48000 sample rate which should represent 16000 sample rate per second. Hence taking 3 seconds of an audio file and using the Wav2Vec model to transform the wav to vec. and send it to the models to classify the given image.

6.2 Website

The website is built from HTML with JavaScript (JS) and CSS, The file selection and recording is by JS also the submission of the data to the server is by JS with Asynchronous JavaScript and XML (AJAX) using a fetch API, the method used is POST. Upon fetch receiving the response from the server in the form of a JSON file fetch will read the data and update the page with the predictions. For the User Interface (UI) a simple version was chosen to minimize and simplify the user experience with the website with few simple buttons and animations. The website supports Firefox, Chrome, and Edge.



6.2.1 Website Structure

The website structure is built from HTML as the body of the website creating all the elements the user sees on the page. JS is the code for the website setup the media player, record, and submission of the form. Cascading Style Sheets (CSS) with Bootstrap setup our elements position and style.

6.2.2 Submit

Upon user submission of the form, the JS will trigger and collect the audio file and selected models send fetch API which will control AJAX which creates a DataForm and send it to the server in a POST method. fetch will wait for the server response when fetch gets the response in form of JSON the JS will setup the data into the HTML page.

6.3 Summary

We have seen both server-side and client-side (as website), the user enters the website and selects models and audio file to send to the server by the website. The server holds the models and awaits requests of web page or data which contain both selected models and audio file, the server will then run each of the selected models on the audio file using WAV2VEC. After collecting all of the models predictions the server sends a response via JSON, which the website receive and extract the data and update the HTML page accordingly. The code is publicly available on GitHub `https://github.com/Lichy24/speech_models_web`.

References

[1] *URDU Dataset* .

[2] *Emotion Recognition in Greek Speech Using Wav2Vec 2.0* .

[3] *Hugging Face* .

[4] *Wav2Vec2 Base 960H*.

[5] *TESS Dataset*.

[6] *RAVDESS Dataset*.