Project documentation: Detecting spoken language from an audio sample

1. Introduction

The “Spoken Language Detection” project aimed to develop an artificial neural network model that could recognize which language is spoken in a given speech sample (e.g. English, German, Polish). Such a system could have a wide range of applications in various fields, such as real-time language recognition in devices using voice assistants for people speaking different languages living in the same household or in telephone exchanges allowing automatic language detection and appropriate routing, speech analysis or speech recognition systems.

2. Main assumptions

The main objectives of the project were:

- Finding and reading the literature on audio processing using artificial neural networks.
- Collecting audio samples in different languages.
- Processing the samples into a suitable form, alignment, augmentation, division into sets.
- Construction and training of a language recognition model.
- Evaluation and optimization of the model.

3. Methodology

3.1. Initial Familiarization

- **Literature:** In order to achieve the main objectives of the project, the project began by finding and reading articles on convolutional neural networks (CNNs) and digital signal processing (DSP) methods. All articles have been included in the bibliography.

3.2. Data Collection

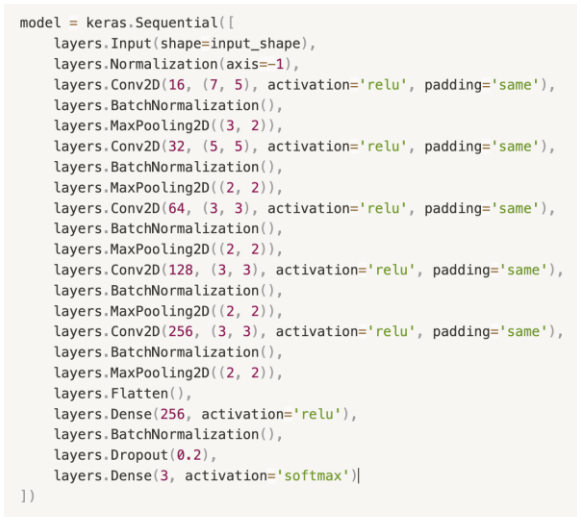
- **Data sources:** Audio data was collected from the free Mozilla Common Voice database created by the Mozilla Foundation.
- **Number of languages:** the data included samples in 3 different languages such as German, French, Polish due to limited hardware resources, however, the system itself was designed to process any number of languages without the need to change the code.

3.3. Data Processing

- **Cleaning and validated data:** The data was checked and well-described, so in order to segment the data into sets and use them to teach the model, all we had to do was use the validated.tsv files (where the information on the sound files was located) and clip_duration.tsv (where the duration of the files in milliseconds was located).
- **Segmentation:** from the set of validated files, those whose duration was greater than or equal to the required sample duration were selected. The recordings were then divided into training, validation, and test sets in 60:20:20 ratios, so that a given speaker would not be repeated in different sets, the gender ratio would be 50:50 in each set, and the number of occurrences of a single speaker in a given set would be controlled.
- **Create a data processing pipeline and feature extraction:** In the processing pipeline using tf.data.Dataset along with lazy loading, the data were resampled, properly labeled, aligned to a given length, augmented, and a spectrogram was created from them with frequencies on a mel scale and amplitude on a logarithmic scale. In the end, the model was taught on parameters nfft=2048, window=512, stdr=256 and mels=256 mainly due to hardware limitations, however, with adequate resources one can try to create a more accurate spectrogram. The batch size is 64. In addition, functions such as cache() and prefetch() were used for optimization purposes. For a sample length of 6 seconds, the input spectrogram size is (64, 375, 256, 1).

3.4. Building the Model

- **Choice of Algorithm:** Used tp Convolutional Neural Networks (CNN) architecture, since the language recognition problem was treated as a problem of image classification, which are different spectrograms.
- **Model Architecture:** The final model is as follows:



```
model = keras.Sequential([
    layers.Input(shape=input_shape),
    layers.Normalization(axis=-1),
    layers.Conv2D(16, (7, 5), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((3, 2)),
    layers.Conv2D(32, (5, 5), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(3, activation='softmax')
])
```

- **Set Size:** The model was trained on a set of 27,000 samples with 9,000 for each language in a 60:20:20 ratio.

3.5. Evaluation and Optimization

- **Collection Enlargement:** For better generalization, the training dataset can be increased. This is even advisable as tests of the model on an increasing dataset have shown.
- **More accurate spectrogram:** If one has a good enough GPU and RAM, a more accurate spectrogram can be created.
- **Model augmentation:** With enough data and resources, one can also create more layers with more filters which will certainly improve the model's ability to recognize more complex shapes. You can also try to increase the number of neurons in the Dense layer.

4. Results

- **Accuracy of the Model:** The model achieved 67% accuracy on the test and validation sets.
- **Error Analysis:** The model performed significantly well on languages from different families, e.g. It recognized languages between the Celtic, Germanic or Slavic families better than in the Slavic family.

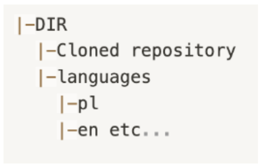
5. Conclusions

- After testing various configurations and training the model, it can be concluded that the amount of data given to train the model was too small for technical reasons. Increasing it should improve the generalization of the model.
- Future work can focus on increasing the number of recognized languages and improving accuracy by using more advanced signal processing techniques and deeper neural networks.
- Increasing the accuracy of the spectrogram is also a proposed direction of development.
- A key factor that projected the model testing results obtained was hardware limitations.

6. Running the environment:

In order to run my test environment, you must have Docker with the appropriate modules that allow you to use the GPU during learning (if you want to do so). Then follow the following steps:

- **Clone the repository:** git clone https://github.com/hubertmaka/Spoken-language-detection.git
- **Create a folder named “languages” next to the cloned repository:** mkdir languages
- **Put the folders from Mozilla Common Voice into the “languages” folder. The final folder structure should look like this:**



```
|-DIR
  |-Cloned repository
  |-languages
    |-pl
    |-en etc...
```

- **Enter the cloned repository:** cd ./Spoken-language-detection
- **Create a Docker image:** git build -t spoken-lang-detection-image [path to Dockerfile (e.g. .)].
- **Run the container:** docker run -gpus all -it -p 8888:8888 -v [DIR directory path]:/app -rm spoken-lang-detection-image bash

7. Links

- https://www.tensorflow.org/io/tutorials/audio
- https://www.tensorflow.org/api_docs/python/tf
- FuzzyGCP: A deep learning architecture for automatic spoken language identification from speech signals. Authors: Avishek Garain, Pawan Kumar Singh, Ram Sarkar
- https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns?
- https://towardsdatascience.com/spoken-language-recognition-on-mozilla-common-voice-part-i-3f5400bbbcd8
- https://towardsdatascience.com/spoken-language-recognition-on-mozilla-common-voice-part-ii-models-b32780ea1ee4
- https://towardsdatascience.com/spoken-language-recognition-on-mozilla-common-voice-audio-transformations-24d5ceaa832b
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8478554/

8. Authors

- Hubert Mąka - AGH student of ICT 3rd year