Project Group 5: ASR for clean and noisy speech

Speech Technology DT2112

Spring 2022

Agnieszka Przezdziak - agnieszka-maria.przezdziak.1133@student.uu.se

Carlo Saccardi - saccardi@kth.se

Edvard All - edvardal@kth.se

Ilias Talidi - talidi@kth.se

Romain Trost - rtrost@kth.se

Table of Contents

Introduction

A prominent problem in speech recognition is how to deal with noise, i.e. features of recorded audio orthogonal or detrimental to the recognition of the included speech. The goal of this project is to investigate a section of this problem by testing an ASR model on clean and dirty audio. A pretrained model was used with appropriate adjustments to fit the needs of the project. Recording data was augmented to add noise. By comparing the performance of models trained with noisy and clean audio in transcriptions of noisy audio, we were able to show that the model fine-tuned on noisy audios performed much better than the one fine-tuned on clean audios. Thus, the ASR system partially ignores the added noise.

Background

Acoustic mismatch between the training and testing data has been shown to degrade performance in ASR models (Huang et al. 2014). In recent years, a significant body of work has focused on improving the robustness of ASR systems to noise in recordings. Using various artificially added types of noise to the training data, Hsiao et al. (2015) showed that ASR performance can be significantly improved when evaluating data recorded in noisy and reverberant rooms. The approaches in this study included speech enhancement, neural networks and acoustic modelling. Training with large amounts of parallel clean and noisy untranscribed corpora, Mošner et al. (2019) showed a WER improvement of up to 19.6% compared to baseline. A selection method was also added to preserve k highest values and emphasise 'correct' knowledge. Deep neural networks have also been used for batch normalisation and residual learning with a focus on factor aware training (FAT) and cluster adaptive training (CAT) (Tan et al. 2018). The experiments show that the ASR was more robust when trained with these methods.

A new acoustic model has also been recently proposed (Baevski et al. 2020) - Wav2Vec2 was proposed as an improvement to existing speech recognition models. It relies less on labelled data than its predecessors and is trained to learn basic speech units while also being trained to predict the correct unit for masked parts of the recording. The basic units learned by Wav2Vec2 are 25ms long, which enables learning of high-level contextualised representations and is applicable across languages. Its main advantage is that it required very little pre-processed data: the authors showed a 8.6% WER on noisy speech, with a training set of 10 minutes of transcribed recordings and 53 000 hours of unlabeled recordings. The model has been applied to fine-tuning ASR for Turkish and English, and these attempts have also inspired this current project.

Methods

The dataset used was taken from the TIMIT dataset, a corpus containing recordings of read speech in American English, containing 5 hours of data (Garofolo et al. 1993) . The TIMIT dataset includes details such as phonetic detail, word detail, sentence type, dialect, and more; since these are not the focus of the project, they were removed during preprocessing. The fine-tuning and training was performed on an existing model, Wav2Vec2 (Baevski et al. 2020).

Data pre-processing

Python libraries supporting audio and signal processing, as well as data analysis and visualisation were used - torchaudio, librosa, pandas, numpy, and more. Wav2Vec2 transformers were included: Wav2Vec2ForCTC, Wav2Vec2Processor and Wav2Vec2FeatureExtractor. Wav2Vec2ForCTC is a model with a language modelling head on top for Connectionist TemporalClassification (CTC) (Hannun 2017). Wav2Vec2Processor combines a Wav2Vec2 feature extractor and a Wav2Vec2 CTC tokenizer into a single processor.

As a first step, the waveform was augmented with noise. The following augmentations were included from the *audiomentations* python library:

- AddGaussianNoise: adds Gaussian noise to the sample
- Gain: multiply the audio by a random amplitude factor to reduce or increase the volume
- TimeStretch: stretches the signal in time without changing the pitch

```python
def dictionary_audio(dataset, augment):
    augment = Compose([
    AddGaussianNoise(min_amplitude=0.0003, max_amplitude=0.0025, p=0.7),
    Gain(min_gain_in_db=-15.0,max_gain_in_db=5.0, p=0.5),
    TimeStretch(min_rate=0.8, max_rate=1.25, p=0.5),
    ])

    dict_data = [{}] * len(dataset)

    for i in range(len(dataset)):
        waveform, sample_rate = torchaudio.load(dataset[i]['file'])
        if augment:
            waveform = augment(samples=np.array(waveform[0]),
                               sample_rate=16000)


        new_item = {'audio': {
            'wave': np.array(waveform),
            'sample_rate': sample_rate}
            }

        for k, v in dataset[i].items():
            if k != 'audio':
                new_item[k] = v

            dict_data[i] = new_item

    return dict_data
```

Before moving to further pre-processing, a checkpoint was included to verify that the transcriptions look as expected, i.e. they resemble written text and not dialogues (this will also be used later to see a sample of the model's output).

```python
def show_random_elements(dataset, num_examples=10):
    assert num_examples <= len(dataset), "Can't pick more elements than there are in the dataset."
    picks = []
    for _ in range(num_examples):
        pick = random.randint(0, len(dataset)-1)
        while pick in picks:
            pick = random.randint(0, len(dataset)-1)
        picks.append(pick)

    df = pd.DataFrame(dataset[picks])
    display(HTML(df.to_html()))

show_random_elements(timit["train"].remove_columns(["audio", "file"]), num_examples=10)
```

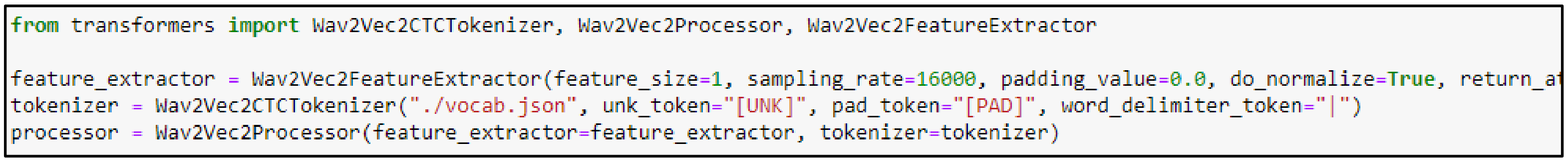
A vocabulary was also created from the transcriptions. Since in CTC modelling (which the Wav2Vec2 is an instance of) it is common to classify speech chunks into letters (Hannun 2017), the model is fine-tuned by concatenating all the transcriptions into a single string and then transforming those into a set of characters. The transcriptions were also normalised through removing punctuation and capitalization.

```python
import re
chars_to_ignore_regex = '[\,\?\.\!\-\;\:\"]'

def remove_special_characters(batch):
    batch["text"] = re.sub(chars_to_ignore_regex, '', batch["text"]).lower() + " "
    return batch

def extract_all_chars(batch):
    all_text = " ".join(batch["text"])
    vocab = list(set(all_text))
    return {"vocab": [vocab], "all_text": [all_text]}
```

In order to be able to recognise word boundaries, a special token for space (“ “) was added and marked with a character that is easier to see: | (pipe). Additionally, two other special tokens were included: “UNK” to denote tokens not encountered in the training set (to address potential issues in the testing phase) and “PAD” - a padding token whose function is to allow output with two of the same character in a row, and to differentiate between outputs that would be collapsed into one otherwise.

```python
vocab_dict["|"] = vocab_dict[" "]
del vocab_dict[" "]

vocab_dict["[UNK]"] = len(vocab_dict)
vocab_dict["[PAD]"] = len(vocab_dict)
len(vocab_dict)
```

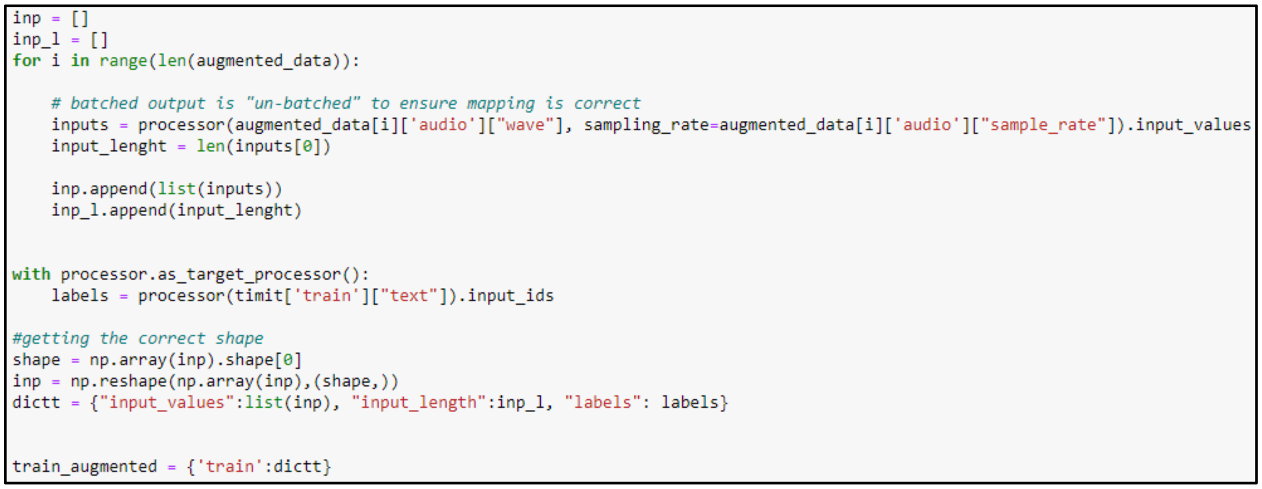
The data prepared in this way allows for the next steps: create the tokenizer and the feature extractor. The Wav2Vec2CTCTokenizer uses the vocabulary and word delimiter (|) defined before and includes the “UNK” and “PAD” tokens. The feature extractor is defined with the following parameters: feature_size (the dimension of the extracted features), sampling_rate (the sampling rate on which the model is trained), padding_value (the value that is used to fill the padding values), do_normalize (whether or not to zero-mean unit-variance normalise the input) and return_attention_mask (whether the model should make use of an attention_mask for batched inference).



```
from transformers import Wav2Vec2CTCTokenizer, Wav2Vec2Processor, Wav2Vec2FeatureExtractor

feature_extractor = Wav2Vec2FeatureExtractor(feature_size=1, sampling_rate=16000, padding_value=0.0, do_normalize=True, return_at
tokenizer = Wav2Vec2CTCTokenizer("./vocab.json", unk_token="[UNK]", pad_token="[PAD]", word_delimiter_token="|")
processor = Wav2Vec2Processor(feature_extractor=feature_extractor, tokenizer=tokenizer)
```

Finally, the dataset can be processed to the format required by the model for training. Finally, the dataset can be processed to the format required by the model for training. The audio data is first loaded and resampled. Then the input values are extracted from the audio file and normalised by the processor. Finally, the transcriptions are encoded to label IDs. The preprocessing is the same for both scenarios; depending on the scenario, the noisy data was used for both training and testing or testing only.

Preparation of the training data:



```
inp = []
inp_l = []
for i in range(len(augmented_data)):

    # batched output is "un-batched" to ensure mapping is correct
    inputs = processor(augmented_data[i]['audio']["wave"], sampling_rate=augmented_data[i]['audio']["sample_rate"]).input_values
    input_lenght = len(inputs[0])

    inp.append(list(inputs))
    inp_l.append(input_lenght)


with processor.as_target_processor():
    labels = processor(timit['train']["text"]).input_ids

#getting the correct shape
shape = np.array(inp).shape[0]
inp = np.reshape(np.array(inp),(shape,))
dictt = {"input_values":list(inp), "input_length":inp_l, "labels": labels}


train_augmented = {'train':dictt}
```

```python
inp = []
inp_l = []
for i in range(len(train)):

    # batched output is "un-batched" to ensure mapping is correct
    inputs = processor(train[i]['audio']["array"], sampling_rate=train[i]['audio']["sampling_rate"]).input_values
    input_lenght = len(inputs[0])

    inp.append(list(inputs))
    inp_l.append(input_lenght)


with processor.as_target_processor():
    labels = processor(timit['train']["text"]).input_ids

#getting the correct shape
shape = np.array(inp).shape[0]
inp = np.reshape(np.array(inp),(shape,))
dictt = {"input_values":list(inp), "input_length":inp_l, "labels": labels}


train_clean = {'train':dictt}
```

Preparation of the testing data:

```python
inp = []
inp_l = []
for i in range(len(test)):

    # batched output is "un-batched" to ensure mapping is correct
    inputs = processor(test[i]['audio']["wave"], sampling_rate=test[i]['audio']["sample_rate"]).input_values
    input_lenght = len(inputs[0])

    inp.append(list(inputs))
    inp_l.append(input_lenght)


with processor.as_target_processor():
    labels = processor(timit['test']["text"]).input_ids

#getting the correct shape
shape = np.array(inp).shape[0]
inp = np.reshape(np.array(inp),(shape,))
dict_test = {"input_values":list(inp), "input_length":inp_l, "labels": labels}

test_noisy = {'test':dict_test}
```

Training the model

The data resulting from the preprocessing steps was input to the Wav2Vec2Processor. A data collator was defined using the example provided for Huggingface's transformers (**reference**). This was done to address the fact that speech input and output are different modalities and should therefore have different padding functions applied to them.

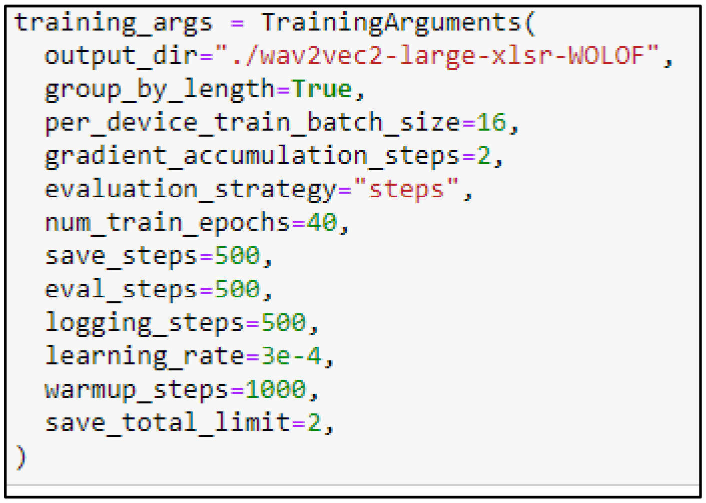
Metrics were loaded to measure the model's performance.

```
def compute_metrics(pred):
    pred_logits = pred.predictions
    pred_ids = np.argmax(pred_logits, axis=-1)

    pred.label_ids[pred.label_ids == -100] = processor.tokenizer.pad_token_id

    pred_str = processor.batch_decode(pred_ids)
    # we do not want to group tokens when computing the metrics
    label_str = processor.batch_decode(pred.label_ids, group_tokens=False)

    wer = wer_metric.compute(predictions=pred_str, references=label_str)

    return {"wer": wer}
```

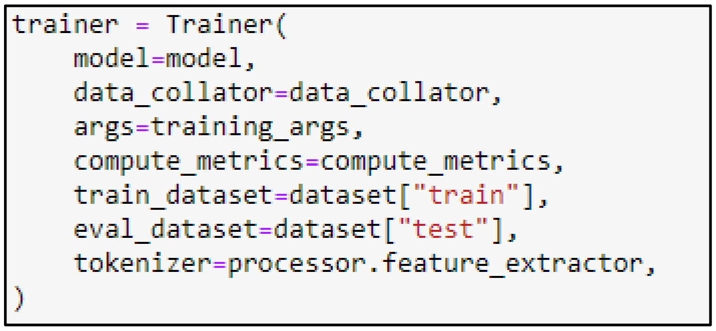
Finally, the pretrained Wav2Vec checkpoint was loaded**.**

```
model = Wav2Vec2ForCTC.from_pretrained(
    "facebook/wav2vec2-large-xlsr-53",
    attention_dropout=0.1,
    hidden_dropout=0.1,
    feat_proj_dropout=0.0,
    mask_time_prob=0.05,
    layerdrop=0.1,
    gradient_checkpointing=True,
    ctc_loss_reduction="mean",
    pad_token_id=processor.tokenizer.pad_token_id,
    vocab_size=len(processor.tokenizer)
)
```

The training parameters were imported and defined using Huggingface’s Trainer class.



```
training_args = TrainingArguments(
  output_dir="./wav2vec2-large-xlsr-WOLOF",
  group_by_length=True,
  per_device_train_batch_size=16,
  gradient_accumulation_steps=2,
  evaluation_strategy="steps",
  num_train_epochs=40,
  save_steps=500,
  eval_steps=500,
  logging_steps=500,
  learning_rate=3e-4,
  warmup_steps=1000,
  save_total_limit=2,
)
```

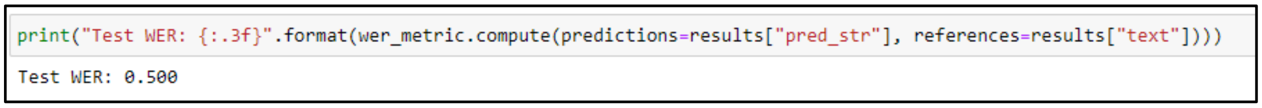


```
trainer = Trainer(
    model=model,
    data_collator=data_collator,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    tokenizer=processor.feature_extractor,
)
```

Results

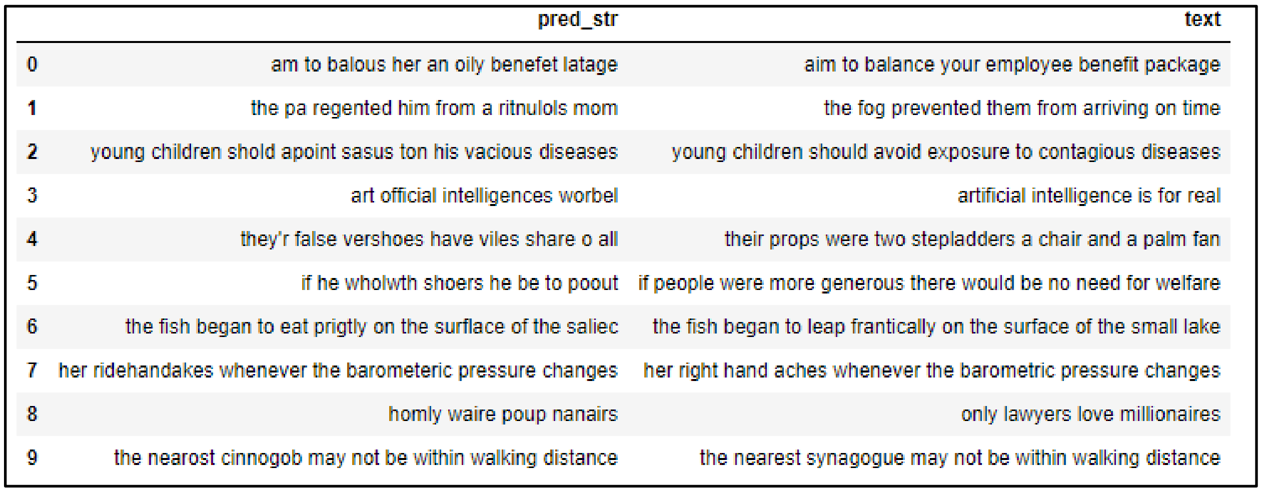
Based on the output from both models, we are now able to compare the WER achieved when the model is trained and tested on clean and noisy data respectively, as opposed to both testing and training on noisy data.

WER for the model trained on clean data:



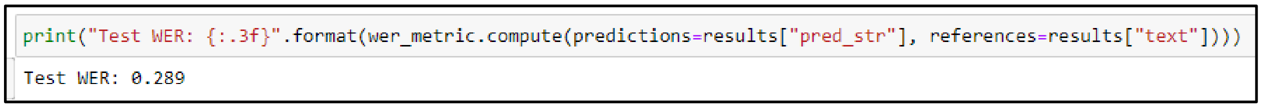
```
print("Test WER: {:.3f}".format(wer_metric.compute(predictions=results["pred_str"], references=results["text"])))
Test WER: 0.500
```

The model trained on clean data only achieved the WER of 50%. An example of the output is shown below:



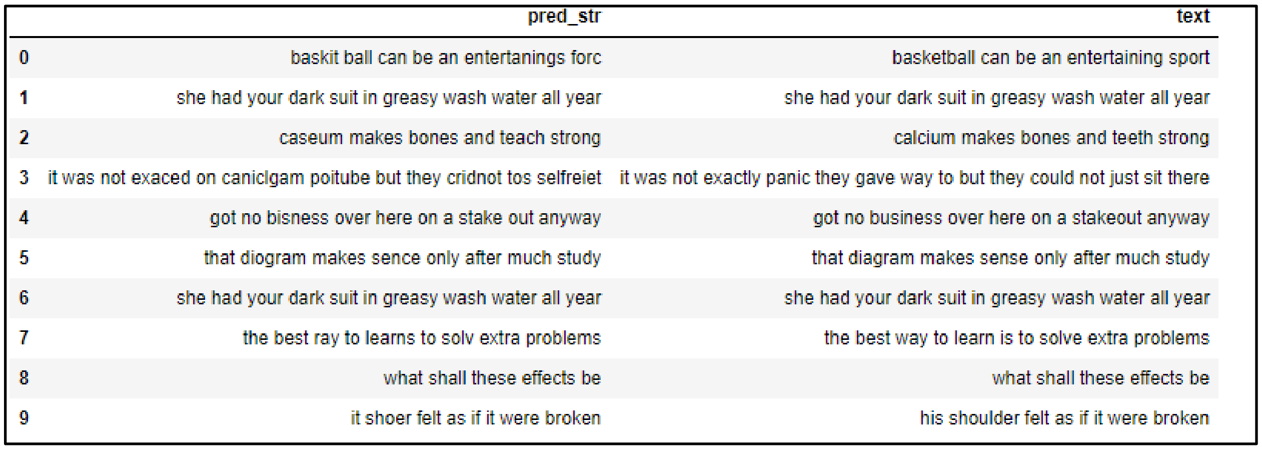
	pred_str	text
0	am to balous her an oily benefet latage	aim to balance your employee benefit package
1	the pa regented him from a ritnulols mom	the fog prevented them from arriving on time
2	young children shold apoint sasus ton his vacious diseases	young children should avoid exposure to contagious diseases
3	art official intelligences worbel	artificial intelligence is for real
4	they'r false vershoes have viles share o all	their props were two stepladders a chair and a palm fan
5	if he wholwth shoers he be to poout	if people were more generous there would be no need for welfare
6	the fish began to eat prigtly on the surflace of the saliec	the fish began to leap frantically on the surface of the small lake
7	her ridehandakes whenever the barometeric pressure changes	her right hand aches whenever the barometric pressure changes
8	homly waire poup nanairs	only lawyers love millionaires
9	the nearost cinnogob may not be within walking distance	the nearest synagogue may not be within walking distance

WER for the model trained on noisy data:



```
print("Test WER: {:.3f}".format(wer_metric.compute(predictions=results["pred_str"], references=results["text"])))
Test WER: 0.289
```

The model trained on noisy data had a WER of 28.9%, with output exemplified below:



	pred_str	text
0	baskit ball can be an entertanings forc	basketball can be an entertaining sport
1	she had your dark suit in greasy wash water all year	she had your dark suit in greasy wash water all year
2	caseum makes bones and teach strong	calcium makes bones and teeth strong
3	it was not exaced on caniclgam poitube but they cridnot tos selfreiet	it was not exactly panic they gave way to but they could not just sit there
4	got no bisness over here on a stake out anyway	got no business over here on a stakeout anyway
5	that diogram makes sence only after much study	that diagram makes sense only after much study
6	she had your dark suit in greasy wash water all year	she had your dark suit in greasy wash water all year
7	the best ray to learns to solv extra problems	the best way to learn is to solve extra problems
8	what shall these effects be	what shall these effects be
9	it shoer felt as if it were broken	his shoulder felt as if it were broken

Looking at the sample transcripts, it is easy to see the improved performance of the second model.In both cases there are spelling errors but the predicted strings in the second model resemble the original more closely overall. In the worse-performing model, there are more cases where the incorrect word is predicted (e.g. *employee* to *oily*) and more nonwords (*balous, cinnogob*), in some cases to the point where the original phrase is completely unrecognisable. Another error type concerns the word boundaries, wherein a sequence of words is glued into one (*right hand aches* into *ridehandakes*).

Conclusions

We have chosen to use a pretrained model for this project - generally, this is preferred for ASR applications. A pretrained model is trained to a point where it will perform well in general applications but not in any specific task. The model can then be fine-tuned to handle the specific task according to the application at hand. For this project, we fine-tuned it using noisy training data to make it more robust at detecting and filtering out noise from clean audio samples.

We initially intended to use the LibriSpeech dataset for this project, since it met our needs in terms of the data it contained and we were already familiar with its structure. The challenge we came across was that the dataset turned out to be too large to train within the planned window. This was part of our mitigation plan, hence we switched to the TIMIT dataset instead and completed the project successfully.

This project had a rather limited scope, and we believe that the next iteration would be to pre-train the model on a wider range of augmented data to perform in real-world scenarios, where the noise is a dynamic variable and we do not control the type of noise nor the volume. But in a realistic scenario this kind of work would need extensive resources and an extended time frame.

To sum up, we understood during this project how powerful this technology could be in the real-world environment. This kind of implementation has numerous applications in daily life, such as speech-to-text conversion which can be used in, for instance, subtitles for youtube videos, real-time voice to text notes, improvement of speech recognition applications under higher amounts of noise, and more.

References

Baevski, Alexei, et al. "wav2vec 2.0: A framework for self-supervised learning of speech representations." *Advances in Neural Information Processing Systems* 33 (2020): 12449-12460.

Hannun, Awni. "Sequence modeling with ctc." *Distill* 2.11 (2017): e8.

Hsiao, Roger, et al. "Robust speech recognition in unknown reverberant and noisy conditions." *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015.

Huang, Yan, et al. "A comparative analytic study on the gaussian mixture and context dependent deep neural network hidden markov models." *Fifteenth Annual Conference of the International Speech Communication Association*. 2014.

Garofolo, John S. "Timit acoustic phonetic continuous speech corpus." *Linguistic Data Consortium, 1993* (1993).

Mošner, Ladislav, et al. "Improving noise robustness of automatic speech recognition via parallel data and teacher-student learning." *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.

Tan, Tian, et al. "Adaptive very deep convolutional residual network for noise robust speech recognition." *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.8 (2018): 1393-1405.

Appendix

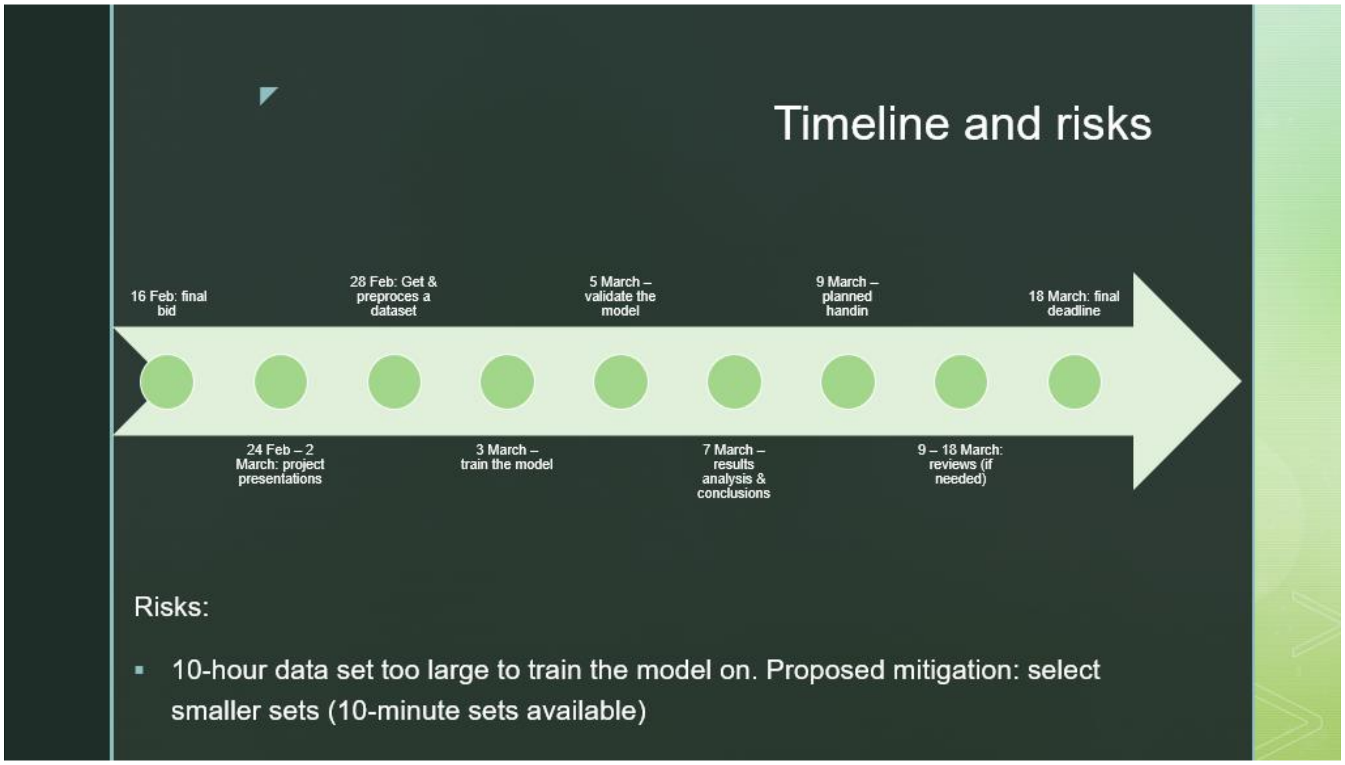
A. Full code used for the project

The code used can be found on this github page: https://github.com/iliastk/ASR-for-clean-and-noisy-speech/tree/main

B. Group members and roles

- Get & preprocess a dataset - Ilias & Carlo
- Training the model - Edvard
- Validate the model - Romain
- Write up results and provide conclusions - Agnieszka & everyone

C. Timeline and risks



Timeline and risks

16 Feb: final bid

24 Feb – 2 March: project presentations

28 Feb: Get & preproces a dataset

3 March – train the model

5 March – validate the model

7 March – results analysis & conclusions

9 March – planned handin

9 – 18 March: reviews (if needed)

18 March: final deadline

Risks:

- 10-hour data set too large to train the model on. Proposed mitigation: select smaller sets (10-minute sets available)