SPEECH-YALE (You Aren't Listening to Everything): a small FillerWordsRemoval approach

Caruso Gaetano
Politecnico di Torino
s317746@studenti.polito.it

Caruso Antonio
Politecnico di Torino
s313443@studenti.polito.it

Abstract

This paper describes a Machine Learning approach to address the problem of detecting Filler Words inside a digital audio file, so that they can be removed during the post-processing phase to make it look "speech-yale".

In order to do so, we were inspired by the popular object-detection algorithm YOLO, which actually acts on images. In parallel, our chosen approach starts by dividing the input audio signal into clips with a duration of $1s$*, assuming that each clip contains at most one filler event. A dB-Mel-Spectrogram (which reflects the human perception of sound) is then extracted and fed as input to the network, which, similarly to the YOLO algorithm, predicts both the class of the event in the clip and, if the event is present, its bounding box coordinates. The resulting task is thus similar to a classification with localization acting on images (spectrograms).*

Our idea is to make a small CNN model that is able to run even on mobile devices. Imagine receiving a vocal message and by just activating the relative option, you can filter out all the filler words with a simple touch, earing smoother and conciser recordings.

1. Introduction

There are many works in the literature on filler words removal, but all of them are either too simple [1] or too complex [10]; on the one hand, there are classification-only tasks that do not take advantage of feature sharing from convolutions and, on the other hand, approaches that make extensive use of transformers, the state-of-the-art technology. While the latter offer, without any doubt, excellent performance, sometimes using such architectures may be prohibitive due to limited computational means (e.g. CPU-only devices, smartphones, ...). In order to overcome those limits, we decided to make a trade-off between quality and performance and opt for an intermediate approach.

Despite several works apply object-detection algorithms to audio files, there is almost no one specifically thought for our exact task: examples range from general signal identification as [9] to audio segmentation and sound event detection as [8]. Another interesting work is Speech-YOLO [5], which combines the tasks of classification and regression to predict both the class of a certain word and its boundaries, using a CNN, in particular *VGG-19*; however, this model is not specifically designed for filler words detection, but rather for "regular" words.

We took inspiration from all those works and made their approaches suitable for our specific task.

2. The PodcastFillers Dataset

The reference dataset is PodcastFillers, which is specifically formulated for the Filler-Words removal task.

It contains almost 89.000 manually annotated audio events extracted from common English Podcasts; these are divided into two main branches: Fillers (Uh, Um, You Know, Other, Like) and Non-Fillers (Words, Repetitions, Breath, Laugher, Music, Agree, Noise, Overlap). From this whole representation, a consolidated label vocabulary is taken, where minor classes are removed or merged to provide the final 6 classes:

Words, Uh, Um, Breath, Laughter, Music.

For an overall of almost 77.000 total events.

We used pre-processed WAV clips:

> "Pre-processed 1-second audio clips of the annotated events, where each clip is centered on the center of the event. For annotated events longer than 1 second, we truncate them from the center into 1-second. The clips are in the same format as the pre-processed full-length podcast episodes: wav format, mono channel, 16 kHz sample rate and 32 bit depth." [11]

From the overall metadata, we kept the label of the event together with the center and duration of the occurrence, computed from the start and end timestamps of each event. We also decided to not take into account events marked as

Extra to enhance robustness, using the consolidated vocabulary. The data samples are already statically partitioned in training, test and validation subsets.

2.1. The limits of PodcastFillers Dataset

This dataset is very interesting because it represents a great effort and a step forward to make a complete and well-balanced dataset for our problem. However, as it was conceived, it has a strong drawback: there is no negative class! This can be motivated by observing the architecture of [10], which is made of a 3-stage pipeline, consisting of a VAD, ASR and a classifier on top, where the classifier has to process only data which are simultaneously interpreted as sound perceived by the VAD but a word not transcribed by the ASR. As such, the word to be processed is very likely a positive class Filler one.

Since we decided to not use transformers, we tried to deal with this lack by taking into account another dataset, too: LibriSpeech ASR corpus. It contains a large corpus of english speech, obtained from audio-books. We chose this dataset because it contains well-structured speech, read by professionals, so that we can assume that it contains no FillerWords, or, anyway, a negligible amount of them. Thus, we exploited it to generate negative class counterexamples (*Nonfiller*) clips of $1s$ to be shuffled among the other positive occurrences. The number of samples of newly generated Nonfillers was chosen such that the relative proportion, compared to the samples from positive classes already present, is similar to what can be found in common speech: 70% NonFillers and 30% Fillers. Since Nonfiller clips contain no filler event at all, the meta-values for center and delta are "don't care" and both set to 0.0.

However, there is a problem with mixing two different datasets together: the network to be trained may only learn to distinguish between the two types of data (i.e., it may learn to separate the datasets) instead of learning the characteristic features of each class, which is what really matters. In order to mitigate this limitation, we performed preprocessing on all data, in order to try to make them as homogeneous as possible.

As expected, results show that the performance is quite good if data augmentation is not performed, while it suddenly drops down when applied; this behavior suggests that the network is not able to properly generalize to new data. So, we tried other techniques to mix the two datasets.

2.2. Trials of mixing two different datasets

Rather than just splitting the entire SpeechLibri recordings into consecutive chunks of $1s$ each, we tried to make more precise cuts to create the Nonfiller counterexamples of the negative class. We needed clips of $1s$ full of common speech, without noise or filler words. This requires both VAD and ASR to properly detect these regions.

Class	**# Training**	**# Test**	**# Validation**
Breath	7,421	732	135
Laughter	5,751	579	293
Music	4,156	822	67
Nonfiller	52,058	6,172	1,319
Uh	14,722	2,598	586
Um	14,070	2,446	562
Words	18,692	2,292	735
	116,870	**15,641**	**3,697**

Table 1. Class occurrences for final dataset

ASR is computationally very expensive ($2s$ to process a single clip of $1s$), but we can avoid using it if we consider the nature of the dataset where audio-books are free, or almost free, of filler words. The VAD, instead, is 10 or 100 times less expensive than ASR, and can definitely be used to remove silence from audios.

However, this approach requires too much time (requires processing 150.000 clips) and still does not solve the problem of mixing the two different datasets, so we came up with a different solution: we only used the PodcastFillers dataset and leveraged the already existing VAD annotations to extract Nonfiller samples from full episodes.

2.3. The single adapted dataset

We built a single custom dataset entirely from PodcastFillers. In particular, positive class instances were regenerated from scratch, cropping the full episodes based on the timestamps of each sample clip and applying a random temporal shift, so that the event center does not always coincide with the clip center; the amount of shift is a random number in the interval $[-0.45, 0.45]$ and is used to place the event center in an arbitrary position inside the clip (with a margin of $0.05s$ with respect to both the start and the end of the clip). This static approach has the purpose of avoiding shifting the spectrogram on-the-fly and filling the empty part with silence or other fictitious data (e.g. noise) to preserve original sounds.

Instead, negative class samples were generated by taking clips of $1s$ directly from the full podcast episodes, by considering only intervals corresponding to speech located by VAD annotations (we consider the VAD active once it reaches 30% sound level) and having no intersections with intervals containing Filler Words.

Finally, positive and negative samples were mixed in a proportion of almost $50\%/50\%$, and the corresponding labels were built accordingly.

The total number of occurrences separately for each class and subset can be found in table 1.

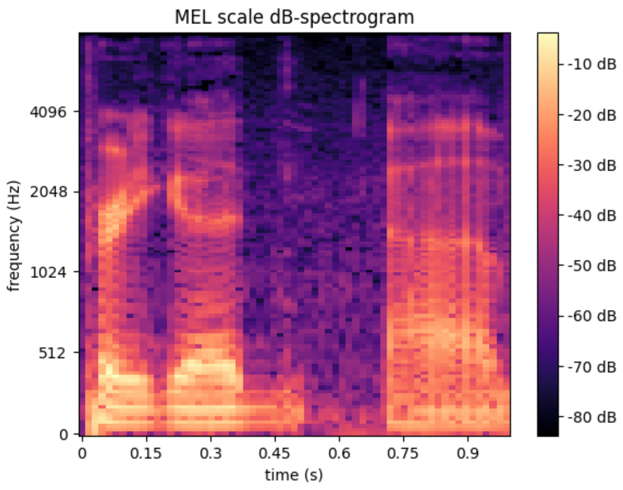

Figure 1. Spectrogram of a validation sample in 128x63 size

3. The Model

3.1. The Data loading process

Raw audio clips (*wav* format, converted to *mp3*) must pass through a transformation pipeline before they can be sent to the model as input to be processed: first, on-the-fly data augmentation during loading phase is performed (on training subset only), including time stretching (requiring adjusting the bounding box labels accordingly), volume scaling, noise addition, and pitch shifting; each transformation has a probability of 50% to be performed. This helps the network achieve better generalization capabilities for new data and reduces the chance of overfitting the model even with a limited number of training examples. After applying these possible transformations, the audio signal is converted into a dB-scale Mel-Spectrogram, using a window length of 512 samples, a hop length of half the window size and a frequency resolution of 128 Mel bins. The obtained spectrogram is then scaled to a *224 x 224* input size of the model and normalized, so that each pixel value is bounded to the interval $[0, 1]$. An example of spectrogram, in its original shape, in order to show the real scale of x-axis, is depicted in figure 1.

Data samples are loaded in mini-batches of 64 elements each.

3.2. The Network Architecture

For our purposes, different CNN architectures were considered: *ResNet-8*, *ResNet-18* [2], *ResNet-34*, and *MobileNet-v3* (large) [3]. All of them were chosen because of their relatively low complexity, which allows them to be used on low-powered devices, too. We also considered using *VGG-19* [6], but it revealed to be too complex to train with our computational means.

All models were trained from scratch and are equipped with two ”heads”: a custom classifier and regressor. The rationale behind this choice is that the classification and regression tasks are quite different, so they may require a distinct configuration of parameters and also a different network structure.

Architecture	Nr. of parameters
ResNet-8	5,353,161
ResNet-18	11,697,865
ResNet-34	21,806,025
MobileNet-v3 (large)	3,843,353
VGG-19	39,425,225

Table 2. Number of trainable parameters for each architecture

The classifier consists of 2 fully-connected layers with, respectively, 256 and 7 (nr. of distinct classes) neurons. The regressor, instead, has 3 fully-connected layers with 512, 256 and 2 (bounding box coordinates) neurons. We found that using a deeper network for the regressor led to better results, since regression may be a slightly more difficult task than classification. After each head hidden layer, we also applied dropout with 0.2 frequency on classifier and 0.3 in the regressor, in order to avoid potential overfitting.

The full number of trainable parameters is illustrated in table 2.

3.3. The Loss function

The main challenge when trying to perform two different tasks with the same model (classification and regression) is choosing a loss function which is suitable for both of them.

The natural choice for a regression task is, usually, an L2 loss (MSE). However, in this particular circumstance, we found that using a Smooth-L1 loss performs better. This is probably due to the fact that the ground truth values (center and delta for bounding boxes) belong to the $[0, 1]$ interval and the same also applies to the difference between predicted and true values. Squaring each term makes these differences even smaller, causing problems during training.

Furthermore, we observed that the performance improves if we introduce another constraint on the bounding-box coordinates, by adding another contribution to the overall loss, a ”coherence loss” term which tries to bind center and delta together by promoting the exact correspondence between the predicted and actual values for the bounding-box start and end points. This loss term is, again, represented by a Smooth-L1 function.

The classifier, instead, is trained using a classic Cross-Entropy Loss to perform multiclass classification. Moreover, noticing that each class is skewed since counting a number of occurrences different from all others, we decided to weight each class contribute by a factor that is the inverse of the class occurrences number, to penalize more mistaken predictions on rare classes and less mistakes on more frequent ones.

All terms are then added together, each one weighted by a distinct hyperparameter λ, to make the total loss function.

$$\begin{aligned}\text{Loss} = & L_{\text{classification}} \\ & + \lambda_{\text{center}} L_{\text{center}} \\ & + \lambda_{\text{delta}} L_{\text{delta}} \\ & + \lambda_{\text{coherence}} L_{\text{coherence}}\end{aligned}$$

In our tests, we found out that a good starting point for these hyper-parameters is λcenter $= 50$ and λdelta $=$ λcoherence $= 25$. These values have been chosen to bring the classification and regression contributes approximately to the same scale. Furthermore, the center contribution has been assigned a doubled value for its weight because we observed that the model had more problems in predicting the center itself accurately.

However, after that, since there are quite a number of possible lambda values, we tried to retrieve them dynamically, to be able to get their best values automatically. We started with the initialization values cited above and then let the network itself learn their optimal values during training. This approach has been discussed in [4] in various multitask learning contexts. So, we opted to use this technique in our final combined loss.

4. Evaluation metrics

Since Filler Word Removal can be considered a classification with localization task, in order to evaluate the model we considered metrics for classification only, regression only and metrics which consider both aspects simultaneously.

- Classification:
  - Accuracy
  - Precision (per class)
  - Recall (per class)
  - F1-Score (per class)
- Regression:
  - Mean Intersection-Over-Union
  - Percentage of predictions with relative error on delta within 10%
  - Mean Absolute Error (center and delta)
  - Normalized Mean Absolute Error (delta)
  - Max Absolute Error (center and delta)
- Combined:
  - Accuracy (class correctness, and IoU above 50% - positive classes, only -)

5. Experiments

5.1. Learning rate, optimizer and scheduler

We used the Adam optimizer (for both model and loss function parameters) and tried 3 different values for learning rate: 10^{-2}, 10^{-3} and 10^{-4}. Instead of directly using the optimizer, we leveraged the advantages offered by a scheduler, OneCycleLR [7], which has the purpose of automatically adjusting the value of the learning rate at different stages during training (we set its maximum to 10 times the base value), in order to avoid local minima and make the model converge faster to an optimum. Using the scheduler, the learning rate starts increasing at the beginning until it reaches the maximum allowed value and then starts gradually decreasing to reach the base value.

Using 10^{-2} causes the model to diverge during training, while 10^{-3} causes oscillations in the loss value. So we chose $lr = 10^{-4}$.

5.2. Architectures and training process

We mainly focused on two different families of architectures:

- Different versions of *ResNet*: thanks to the skip connections, they were easy to train, because of the ability of 'skipping' some layers if not needed, reducing overfitting and vanishing gradient risks.
- *MobileNet-v3* (large) model: specifically thought for mobile devices without GPU.

Both families proved to be fast for both training and inference, even on low-powered devices.

In particular, we tested *ResNet-8*, *ResNet-18* and *ResNet-34*. In all cases, we found that performance was good enough even with smaller *ResNet* versions (i.e. with a reduced number of layers), which naturally led to the selection of the best-performing (and possibly the least complex) model based on the particular computer architecture.

We also tried other architectures, such as *MobileNet-v2* and *VGG-19*. *MobileNet-v2*, despite having less parameters with respect to *ResNet* with similar complexity, was significantly slower to train. This is due to the lack of optimization for GPU architectures, this model being designed to run efficiently on CPUs, without leveraging parallelization capabilities. On the other hand, *VGG-19* was too heavy for our computational means (due to very large number of trainable parameters).

Performing fine-tuning on preexisting architectures was not a viable option, since these CNNs were initially thought for feature extraction on images, while here we work with spectrograms, and features learned on real-life objects are hardly useful on this kind of input data. Training from scratch was thus the only possible solution.

6. Results

We found that all *ResNet* models led to similar results on the same metrics, both for classification and regression, with larger models performing only marginally better. This is a clear evidence that the upper bound to performance is not determined by the model complexity, but by the data themselves: these data were manually labeled leveraging crowd-sourcing and, sometimes, labels may not be 100% accurate (the original dataset reports, for each sample, the confidence of its label).

In tables 5, 6 and 7 we reported respectively the cumulative metrics for classification, regression and combination of both aspects.

Figures 3 and 5 show the distribution of absolute errors on center and delta using, respectively, *ResNet-18* and *MobileNet-v3* networks because they are the most representative for highlighting differences.

The plots 2 and 4 report the trend of the training and validation loss for *ResNet-18* and *MobileNet-v3* models. The former converges in about 8 training epochs, while the latter, which is a simpler model, takes only 4 epochs to converge. By looking at these plots, it is clear that continuing to train *MobileNet-v3* further causes overfitting (validation loss starts to increase, despite training loss getting better), while *ResNet-18* suffers way less from this issue, thanks to its architecture with skip connections, which prevents the model from becoming too complex.

Class	Precision	Recall	F1-Score	Support
Breath	0.67	0.72	0.70	732
Laughter	0.68	0.85	0.76	579
Music	0.79	0.95	0.86	822
Nonfiller	0.97	0.88	0.92	6172
Uh	0.80	0.76	0.78	2598
Um	0.87	0.88	0.87	2446
Words	0.64	0.73	0.68	2292
Accuracy		0.83		
Macro Avg	0.77	0.82	0.79	15641
Weighted Avg	0.84	0.83	0.83	15641

Table 3. Classification Report using ResNet-18

We reported in tables 4 and 3 the confusion matrix and a classification report (for classification-only evaluation) generated using *ResNet-18*.

By observing, in particular, the F1-Score, which combines both precision and recall, it is clear that the most recognizable class is *Nonfiller*, while the most difficult class to be detected correctly by the model is *Words*, which contains clips corresponding to portions of speech with unclear words, such as cases in which two different persons speak at the same time or simply badly pronounced, unintelligible terms.

Furthermore, by performing trials on actual audio files, we noticed some limitations in the model, which may not be observed directly by looking at metrics, only:

1. "Structured" audio, i.e. actual, well-formed words (*Nonfiller*, *Uh*, *Um*) are typically better recognized by the network, both in terms of class and bounding box. This phenomenon may be due to the fact that these sounds always share the same basic patterns, independently of the particular speaker and context. On the other hand, "unstructured" audio (*Breath*, *Laughter*, *Words*) is more difficult for the model to be properly and accurately detected, since it presents a very high variability, depending on the speaker and context.

2. Some regular words are pronounced in such a way that some parts of them may recall a filler word, for example, in the word "*Um*brella", where the first part may induce the model to classify the clip as belonging to the *Um* class, instead of the correct *Nonfiller* one. Unfortunately, we found no direct way to address this issue, which is determined by an architectural limit of the model itself, and may require using more powerful approaches (e.g. those based on transformers), which, anyway, are out of scope for our purposes.

Unfortunately, comparing our result metrics with other approaches found in literature is very difficult: some works consider network architectures which are similar to the one described in this paper , such as Speech-YOLO [5] but they still use different approaches, evaluation metrics and train the model on different data, in this specific case, regular words instead of filler ones. Furthermore, for "general-purpose" audio detection, more training data are available, enabling to leverage more complex models without the risk of overfitting.

As an example, the authors of that paper obtain:

$$\text{F1-Score} = 0.807, \text{Accuracy} = 0.774, \text{IoU} = 0.843$$

Instead, the authors of [10], which use an architecture based on transformers, obtain F1-Score $= 0.942$, when using ASR + VAD + Classifier, which, however, drops to F1-Score $= 0.713$ when ASR is not used.

6.1. Models and use cases

In the end, by observing the metrics obtained and empirical trials, we found that all the tested models can be useful in different contexts:

- *MobileNet-v3* may be the best choice on mobile CPU-only devices, since it was built with those specific targets in mind;

A / P	Breath	Laughter	Music	Nonfiller	Uh	Um	Words
Breath	529	73	5	0	46	43	36
Laughter	25	495	2	5	5	3	44
Music	4	16	778	1	3	0	20
Nonfiller	14	17	153	5401	101	36	450
Uh	66	28	9	25	1972	185	313
Um	33	9	8	6	142	2155	93
Words	119	86	35	123	199	65	1665

Table 4. Confusion matrix using ResNet-18. Actual class is on row; predicted class is on column

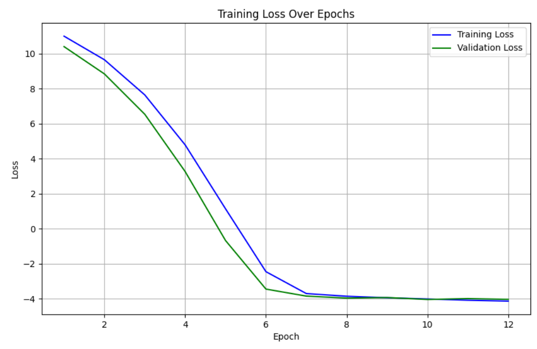

Figure 2. Training and validation loss over epochs with ResNet-18

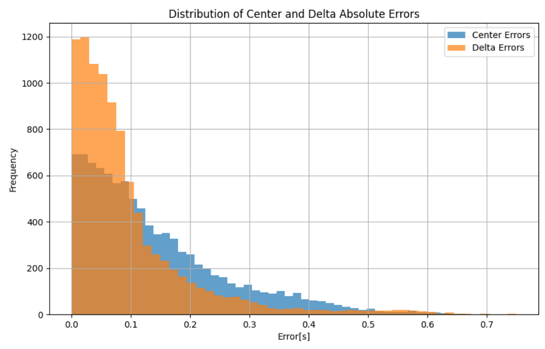

Figure 3. Distribution of absolute errors for center and delta with ResNet-18

- *ResNet-8* may be appropriate for desktop CPU-only devices, because it combines acceptable quality with reasonable performance;

- *ResNet-18* and *ResNet-34* may be the best choice for desktop devices equipped with GPUs, since, despite their larger number of parameters with respect to the smaller *ResNet-8*, they may perform better because their architecture is designed to fully leverage the parallelism offered by GPUs.

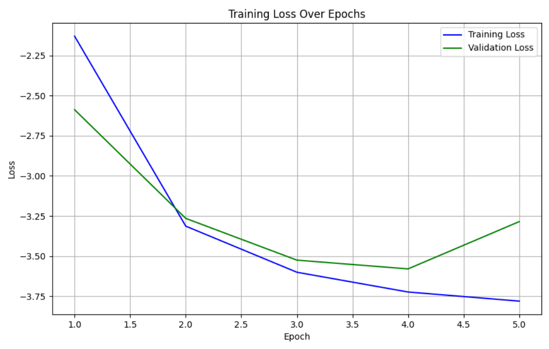

Figure 4. Training and validation loss over epochs with MobileNet-v3

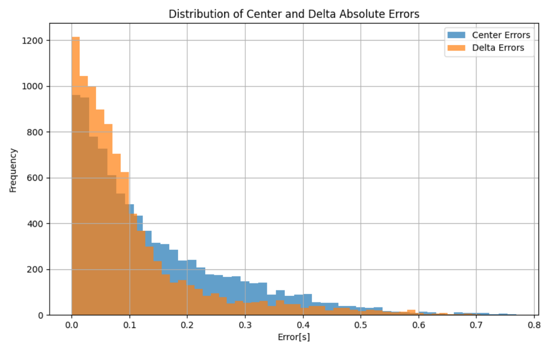

Figure 5. Distribution of absolute errors for center and delta with MobileNet-v3

7. Extra

7.1. Inference module

In order to test the model in a qualitative way, rather than simply looking at evaluation metrics, we also built an inference module that allows to take as input an audio file of arbitrary length and format and to produce, as output, another version of the same file in which both silent intervals (with sufficient length) and filler words are removed. Optional debug capabilities can be enabled, so that extra information can be visualized (e.g. start and end timestamps for each

	ResNet-8	ResNet-18	ResNet-34	MobileNet-v3
Accuracy	81%	83%	84%	77%
Weighted Avg Precision	83%	84%	84%	80%
Weighted Avg Recall	81%	83%	84%	77%
Weighted Avg F1-Score	82%	83%	84%	78%

Table 5. Classification metrics comparison

	ResNet-8	ResNet-18	ResNet-34	MobileNet-v3
Mean IoU	45.23%	46.51%	47.18%	48.10%
Perc. predicted delta within 10% relative error	26.77%	27.55%	21.46%	27.85%
MAE center	0.14s	0.14s	0.14s	0.14s
MAE delta	0.09s	0.09s	0.09s	0.10s
Normalized MAE delta	27.4%	26.84%	28.1%	29.44%
Max Absolute Error center	0.74s	0.69s	0.69s	0.77s
Max Absolute Error delta	0.73s	0.75s	0.74s	0.70s

Table 6. Regression metrics comparison

filler word occurrence) and, in case, saved to disk. When debug mode is enabled, any detected filler word is extracted as a separate audio clip and all words belonging to the same class are stored in the same folder.

The main steps performed by this module are the following:

1. *Silence removal*: audio is preprocessed to remove the parts of the signal in which power is under a certain threshold, which can be defined considering a percentage of the power min-max excursion in the full audio signal.

2. *Split in 1s clips*: audio is split in 1s-long clip, which are grouped in batches and fed as input to the trained model.

3. *Post-processing*: output information from the model (classes and bounding boxes) is used to properly remove each detected filler event occurrence. A sinusoidal fade effect is applied in order to smooth the transition in proximity of cuts in the signal.

4. *Output file saving*: the post-process signal is saved to disk as a single file.

7.2. A different approach: regression vs classification

Since the regression task is usually difficult, at first, we also considered a different approach, consisting in approximating it with an adapted binary classification task. Instead of predicting continuous coordinates for the bounding box center and delta, we divided the $1s$ clip in 10 smaller intervals, each covering $100ms$ of the clip. Each interval was assigned a binary label: 0 if that interval had no intersection with the actual bounding box, else 1. This way, the network had to produce as output 10 binary predictions for every interval (in case of positive class prediction, otherwise these values are clearly ”don’t care”).

This technique led to good results, but still inferior to those obtained by properly training a regressor, mainly because of two major drawbacks:

1. It’s not possible to predict a bounding box which exceeds the size and/or the boundaries of a clip: this is a problem when a filler event is only partially included in that clip, so the event itself may only be removed partially. The impact of this limitation may be reduced considering finer inference algorithms, involving, for example, fractional stride for the sliding window.

2. Classification only approximates regression, so there may be a lower accuracy in predicting the actual bounding box coordinates.

References

[1] ezxzeng. Um detector: detector for filler words, 2019. Accessed: 2025-05-24.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[4] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7482–7491, 2018.

	ResNet-8	ResNet-18	ResNet-34	MobileNet-v3
Accuracy	58.67%	60.90%	64%	57.62%

Table 7. Combined metrics comparison

[5] Yael Segal, Tzeviya Sylvia Fuchs, and Joseph Keshet. Speechyolo: Detection and localization of speech objects. *Proc. Interspeech 2019*, pages 4210–4214, 2019.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[7] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. *arXiv preprint arXiv:1708.07120*, 2017.

[8] Satvik Venkatesh, David Moffat, and Eduardo Reck Miranda. You only hear once: A yolo-like algorithm for audio segmentation and sound event detection. *Applied Sciences*, 12(7):3293, 2022.

[9] Luke Wood, Kevin Anderson, Peter Gerstoft, Richard Bell, Raghab Subbaraman, and Dinesh Bharadia. Deep learning object detection approaches to signal identification. *arXiv*, 2210.16173, 2022.

[10] Ge Zhu, Juan-Pablo Caceres, and Justin Salamon. Filler word detection and classification: A dataset and benchmark, 2022.

[11] Ge Zhu, Juan-Pablo Caceres, and Justin Salamon. Podcastfillers dataset, 2022. Accessed: 2025-05-24.