Towards Efficient Laughter Detection with Convolutional Neural Networks

Yuxi Dai

4th Year Project Report
Artificial Intelligence
School of Informatics
University of Edinburgh

2023

Abstract

The initial motivation of this project is to address the lack of interaction between presenters and audiences during online meetings and aims to enhance this interaction by developing a method for detecting applause and laughter. This project focuses on constructing an efficient laughter detector with convolution neural networks. By using the previous student's work [Wolter, 2022a] as a baseline, we aimed at building a CNN model that performs better in both accuracy and efficiency aspects. In the thesis, first, I solve the issues in Wolter's data preprocessing and training pipeline and retrain the baseline model to get better accuracy. Then, the report investigates different CNN architectures, including ResNet [He et al., 2016], ResNet with depthwise separable convolution[Mamalet and Garcia, 2012], MobileNetV2[Sandler et al., 2018] and EfficientNet[Tan and Le, 2019], and explores how the size of a model will influence the performance.

This report shows that the ResNet-18 with small width performs the best among all the models we evaluate, which has high accuracy and inference speed. We also discovered that MobileNetV2 and EfficientNet are only theoretically more efficient, and have low real-time speeds. Because the paper of MobileNetV2 and EfficientNet (Sandler et al. [2018], Tan and Le [2019]) use the number of calculations as the indicator for efficiency, which is an indirect index of speed.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yuxi Dai*)

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisors, Prof Philip Wadler and Dr Ondřej Klejch, for their invaluable guidance, encouragement, and support throughout my thesis.

I would also like to extend my deepest appreciation to my parents, who have encouraged me and provided me with suggestions when I face difficulties.

Table of Contents

Chapter 1

Introduction

Presentation to a large group of people on Zoom is a common situation nowadays, In this type of meeting, the default is to mute the audience, which means the speaker receives little feedback. The motivation of the project is to provide a way to enhance the interaction between the presenter and the audience during the online meeting. Since Applause and laughter are important feedback from audiences during the meeting, we want to unmute audiences when they are applauding or laughing, which is our initial goal of the project: constructing an Applause and Laughter detection method for the online meeting. However, building a robust and fast detector that can be used in industry is a difficult task. Therefore, in this project, I will focus on one part of the initial goal: constructing an efficient laughter detection method.

The previous student, Wolter [2022a], proposed a data preprocessing pipeline and build a laughter detector based on the work of Gillick et al. [2021]. He mainly investigates how the structure of data (e.g. distribution of laughter and non-laughter samples) will influence performance. I will go deeper into the laughter detection aspect and concentrate on how different architectures and sizes of CNN models will influence the performance of the laughter detector. By considering Wolter [2022a]'s model as a baseline, I will construct a CNN model with better performance. Since the final goal of the project is to apply the laughter detection model in real-time online meetings, both the performance in accuracy and speed aspects are important when evaluating the model.

The main contribution of this project are:

- Find and solve the issues in Wolter's work to improve the baseline's performance. (Chapter 3)
- Investigate how the depth and width of the model will influence the performance on accuracy and speed aspects. (Chapter 4)
- Investigate how depthwise separable convolution [Mamalet and Garcia, 2012] will influence the performance. (Chapter 4)
- Investigate the performance of the MobileNetV2[Sandler et al., 2018] and EfficientNet [Tan and Le, 2019] on laughter detection tasks. (Chapter 4)

Then we will talk introduce the structure of this thesis. The thesis is split into three main parts, which are the review of existing works on laughter event detection, the evaluation of previous work and the experiments on CNN architecture.

The thesis will start with some reviews of existing research about audio event detection, laughter detection, methods to improve the efficiency of models and some evaluation metrics (Chapter 2). The audio event detection section includes the work of Palanisamy et al. [2020] and Hershey et al. [2017]. They show that the convolution neural networks(CNN) designed for image-related tasks perform well on the audio detection tasks, which provides a solid foundation for our thesis to try some CNN architectures used in image-related fields. In the laughter detection section, Gillick et al. [2021] introduce a robust laughter detection method which uses a CNN architecture called ResNet-18. This is considered as the baseline for Wolter [2022a]'s work. Since the overall goal of our model is to embed the laughter detector into a real-time online meeting software, the inference time of the detector under limited resources is a key element we care about. Therefore we include a section about existing work on improving the efficiency of CNNs. In this section, a CNN architecture called MobileNet [Howard et al., 2017] is further explored by us in the experiment section. The last part of the review section introduces some metrics that can evaluate the performance of the laughter detector in both accuracy and speed aspects. These metrics include the confusion matrix that is used by Wolter [2022a], and some other metrics that are widely used when evaluating the performance of CNN.

In Chapter 3, we will involve the issues in Wolter [2022a]'s work and introduce how we will fix them. There are three main issues in his works, which have serious impacts on the performance of the laughter detector. The first issue appeared in the data preprocessing step, which is the laughter in non-transcribed regions. This issue was discovered by Wolter when he saw some unexpected results when he evaluate his model at the end of his project, so he has no time to solve this. The issue is that some samples labelled as 'silence' has sounds, such as human speech, in them, which lead to the detector misclassifying the true 'laughter' into 'silence'. We fix this problem by modifying the prepossessing process and regenerating the dataset and see an improvement in precision for the performance of the model.

The second issue appears in the training step, which is about manually setting a hyper-parameter: the number of epochs. The number of epochs is an important hyper-parameter for training a CNN model, which can decide how well the model will fit the training data during the training. As the number of epochs increases, the accuracy of the test set will first increase and then decrease, so a suitable epoch number can lead to better performance. Wolter manually set the number of epochs in his training process, and this strategy has several disadvantages. When the manually set hyper-parameter is too big, the model will face the problem of overfitting, which means this model will overfit to the training set and perform badly on the test set. However, if the number of epochs is set to be too small, then the model fails to learn from the training data. I solve this problem by adding a method called 'Early Stop', which let the training process automatically decide the number of epochs and when to stop the training.

The third problem is the mismatch between the training and test sets. In Wolter [2022a]'s work, the proportion of laughter and non-laughter samples are fixed, for example, if we set the distribution to 1:20, then for each audio in the dataset, every time we extract one laughter sample, we will extract 20 non-laughter samples. However, in the test set, the model is evaluated on the raw audio data directly, which does not follow the manually set distribution. We observe that the model is overfitted to the distribution of the training set, which makes the model's performance on the test set lower than we expected. Due to the time limitation, I do not explore a way to solve this problem(directly using the raw audio data as the training set is time-consuming). However, we could expect a higher accuracy when we let the distribution of the training set match the raw data. What is more, some debugging is also taken to make the preprocessing pipeline [Wolter, 2022b] actually work. Since Wolter uses a Python library which is still under development, some methods in the library are updated to a newer version and make the preprocessing pipeline fail to work.

In Chapter 4, we explore the influence of different architectures and sizes of CNN models and evaluate them from both the accuracy and speed aspects. In this chapter, we first compare the baseline (modified version of the ResNet-18 used by Wolter [2022a] and Gillick et al. [2021]) and the standard ResNet-18 architecture from the paper [He et al., 2016].

Then in the next two experiments, we find out how a CNN model's depth and width will influence the accuracy and speed performance. Depth and width are two key factors that decide the size of a model, and the base model we use in these two experiments is the ResNet. We hypothesise that the larger the model's size, the more accurate the model will be. And we also expect to see a convex accuracy curve when the model size becomes larger. The result of our experiments shows that the relation between model size and speed matches our hypothesis. However, the accuracy displays an inverse trend as we hypothesise, which first decreased and then increases, when the model size gets bigger. This result does not match the common sense. Therefore we think some mistakes appear during the training process and would like to explore this problem in future work.

The next experiment explores a method called depth separable convolution (DW) [Mamalet and Garcia, 2012], which reduces the number of calculations without a large decrease in accuracy. With a smaller number of calculations, we expect the speed of inference will increase. We apply the DW structure to three existing models in previous experiments (baseline, standard ResNet-18 we evaluate in experiment 1 and ResNet-18 with small width) and evaluate their performance. The result shows that the number of calculations and the inference speeds decrease when we combine the DW into these three models. However, the recall in the accuracy aspect decreases a lot as a trade-off.

The previous experiments are all based on the ResNet architectures. Therefore it is worth exploring some other architectures. We explore two architectures in this thesis: MobileNetV2[Sandler et al., 2018] and EfficientNet[Tan and Le, 2019]. Both two models are state-of-arts lightweight and fast CNN. Similar to what we do in the last experiment, both of these works try to use depth separable convolutions to reduce the number of calculations. Hence, we hypothesise that these two architectures will have a good performance on speed. However, the result shows that the speed of these two models is much slower than ResNet-18. This led to a discovery that a reduction in the number of calculations does not mean a faster real-time speed.

Chapter 2

Background research

2.1 Audio Event Detection

There are many fields in audio processing. Automatic Audio Segmentation, Audio Event Classification and Audio Event Detection (AED) are closely related to each other. And from a higher level, Audio Event Detection can be viewed as the combination of Automatic Audio Segmentation and Audio Event Classification, which can be seen in Figure 2.1. The Audio Event Detection will take raw audio data as input and output the events it detects.

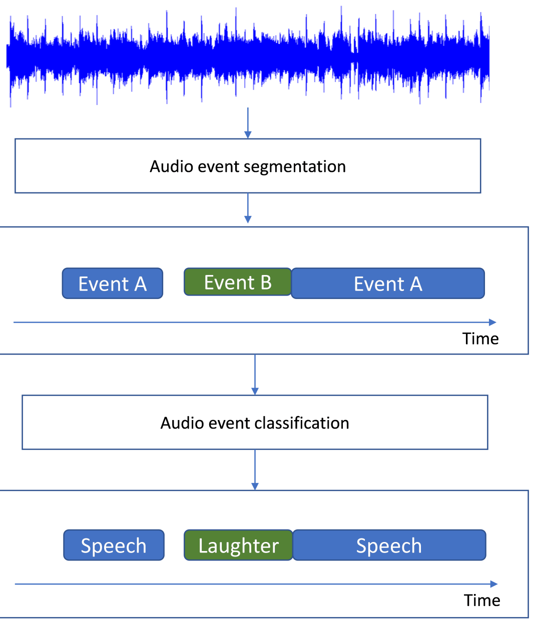


Figure 2.1: Process of audio event segmentation and classification.

We start by introducing Automatic Audio Segmentation and Audio Event Classification. The objective of Automatic Audio Segmentation is to partition audio signals into distinct segments based on their respective sound categories, such as speech, music, environmental sounds, noise, etc. The general architecture of Automatic speech processing is 1) feature extraction 2) segmentation. In the feature extraction step, the spectral features are widely utilized for representing audio signals during feature extraction. MFCCs[Davis and Mermelstein, 1980], PLP[Hermansky, 1990] are commonly used spectral features. The segmentation step involves two primary approaches: the distance-based method and the model-based method. The former method identifies changes in the audio stream and determines the segment boundaries. However, it cannot label the segment's sound category. The latter method, which involves the use of machine learning algorithms, can classify segments into specific categories, such as speech, music, or silence. The most commonly used machine learning algorithms in Automatic Audio Segmentation include Gaussian Mixture Model (GMM)[Duda et al., 1973], Support Vector Machine (SVM)[Cortes and Vapnik, 1995], Artificial Neural Networks and so on. The most recent work[Venkatesh et al., 2022] introduced a modified Yolo [Redmon et al., 2016] Convolutional Neural Network (CNN) algorithm for Audio Segmentation and Sound Detection tasks. By generating the boundary box directly on the image, the Yolo-like model constructed by Venkatesh et al. [2022] can output the start and end times of each segment.

The goal of Audio Event Classification is to assign one or more specific class labels to a given audio segments, such as speech, music, environmental sounds and noise. The architecture of Audio Event Classification is similar to Automatic Audio Segmentation, where the first step involves extracting features from the raw audio data, followed by classification algorithms applied to the extracted features. Nowadays, various types of Convolution Neural Networks (CNNs) are widely explored for audio classification tasks by researchers, owing to their success in image-related tasks like image classification and segmentation.

Several studies have evaluated the effectiveness of CNNs for audio classification and have reported promising results (Palanisamy et al. [2020] and Hershey et al. [2017]). Hershey et al. [2017] conducted a study that evaluated multiple CNN architectures for large-scale audio classification. The CNN architectures tested in the study, namely, AlexNet[Krizhevsky et al., 2017], VGG[Simonyan and Zisserman, 2014], Inception[Szegedy et al., 2015] and ResNet[He et al., 2016], were originally designed for image classification tasks. This study utilized the YouTube-100M dataset, which is a massive corpus containing 100 million YouTube videos and was constructed by the authors. The videos in the dataset were labelled based on the entire content, and each video could have multiple labels. The results indicated that all CNN architectures outperformed the fully-connected network. Among the tested architectures, ResNet-50 achieved the best performance. The authors also noted that increasing the dataset size can improve the performance of ResNet-50.

Another work that evaluated the effectiveness of CNNs for audio classification is Palanisamy et al. [2020], which hypothesizes that a pre-trained ImageNet CNN model[Deng et al., 2009] can be used as a strong baseline for the Audio Classification task. Using a pre-trained model means using a model that is previously trained on a huge amount of data, and extending this model to new tasks based on its prior knowledge. The ImageNet is a Deep CNN that is trained on large numbers of images, and is widely used as a pre-trained model in tasks such as image classification and image segmentation. While pre-trained models for audio classification tasks are typically trained on large audio datasets such as AudioSet [Gemmeke et al., 2017], the authors demonstrate that the ImageNet CNN can learn useful audio representations. The paper explores the integrated gradients for the Mel Frequency Cepstral Coefficients (MFCCs) input into the model. Figure 2.2 demonstrates that the network can detect the boundaries of high-energy regions in the audio signal, which vary depending on the type of sound. Based on these findings, the authors suggest that pre-trained ImageNet models, which excel at image boundary detection, can also perform well in audio classification tasks by distinguishing differences in the spectrum of sound.

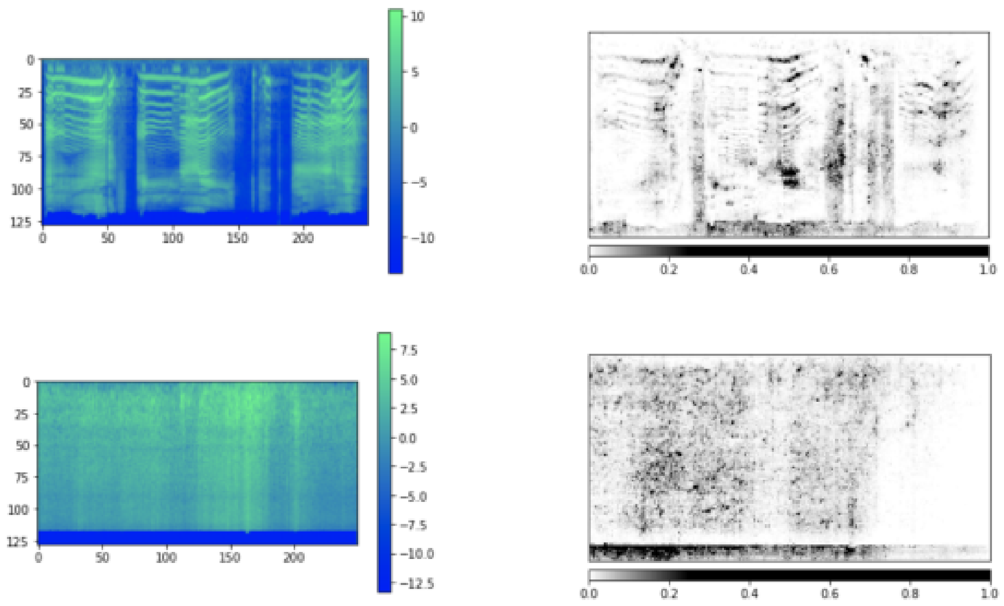


Figure 2.2: The left column displays the data(after applying feature extraction) input into the model. The right column shows the integrated gradients visualization of the input (Palanisamy et al. [2020])

Audio event detection (AED) is a field that aims to identify various types of audio signals, including speech and non-speech events within an audio stream, where laughter detection is a specific branch of AED. AED is considered a combination of audio segmentation and classification tasks, and therefore, the approach to detection is similar to that of segmentation and classification. The first step of AED involves feature extraction, with commonly used features being spectral features and prosodic features such as Mel-frequency cepstral coefficients (MFCCs) and pitch. The second step involves applying segmentation methods. In the past, methods like Support Vector Machines (SVMs), and Hidden Markov Models (HMM)[Baum and Petrie, 1966] are used. And nowadays, CNNs have been applied to AED tasks due to their ability to learn features of audio data.

In spite of the similarities between Audio Classification and Audio Event Detection, Audio Event Classification is generally considered an easier task than detection. This observation has been thoroughly examined by Phan et al. [2017], and their paper also introduces a pipeline that could reduce the false positive rate in the audio detection task. According to the authors, there are two main reasons why classification is easier than detection: 1) in the classification task, the model is usually able to capture the global context of the entire event, whereas, in the detection task, the model does not know the exact start and end times of the event and may rely on unreliable local audio features during inference. 2) the detection task need to detect a target event out of rich background sounds, while the classification task only needs to find the difference between different event.

2.2 Laughter Detection

Laughter Detection is a branch field of Audio Event Detection that focuses on identifying the unique acoustic characteristics of laughter in a given audio stream. It involves analyzing various acoustic features, such as pitch, duration, and spectral content, to distinguish between different types of laughter and other speech and non-speech sounds. Acoustic characteristics of laughter have been investigated in prior research: Provine [1996] have shown that laughter is composed of a sequence of short vowel-like sounds, each lasting about 75 milliseconds, with an interval of approximately 210 milliseconds between them. Furthermore, laughter generally has a higher pitch level than normal speech, as reported by ([Bachorowski et al., 2001]), this lead to the pitch may be a helpful factor for laughter detection. However, identifying laughter is a challenging task, as there are many different types of laughter, including voiced, unvoiced, song-like, and others [Trouvain, 2003], which exhibit high variability and are difficult to distinguish from speech.

ICSIJanin et al. [2003] is a corpus that is widely used for laughter detection, because it labelled the exact start and end time of each laughter event, unlike other corpora (e.g. YouTube-100M [Hershey et al., 2017]) that only labelled whether a segment contains laughter. While several studies have focused on laughter detection using this corpus, such as Kennedy and Ellis [2004], Truong and Leeuwen [2005], and Knox and Mirghafori [2007], each study applies different feature extraction and classification methods, as well as distinct approaches to handling the data. Both Kennedy and Ellis [2004] and Truong and Leeuwen [2005] pre-segment the data, while Knox and Mirghafori [2007] is the first paper that eliminates the need to pre-segment the data. The goal of Kennedy and Ellis [2004] is to detect the audio event when multiple people laugh together. This paper cut the dataset into non-overlapping one-second segments, and defines a segment as a laughter event when more than a certain percentage of participants are laughing. Kennedy and Ellis [2004] uses SVM as a classifier and compares different feature-extracting methods including MFCC, modulation spectrum, and spatial cues. The true positive rate reached 87% when MFCC was used for laughter detection, whereas the true positive rate refers to laughter samples that are correctly classified as laughter. The authors also discovered that the SVM classifier trained on the ICSI corpus may not perform well on other corpora. They tested the model trained on ICSI on three other corpora: CMU, NIST [John S. Garofolo], and LDC. The results show that the model performed reasonably well on CMU and NIST but no better than random guessing on the LDC corpus. This highlights the importance of testing a model on multiple corpora to ensure that it can handle a variety of situations.

In the pursuit of improving laughter detection on the ICSI corpus, Truong and Leeuwen [2005] build upon the work of Kennedy and Ellis [2004]. However, they take a different approach to defining a laughter event. Since the ICSI corpus provides labels for the exact start and end time of each laughter event, Truong and Leeuwen [2005] split the corpus into speech and laughter segments of varying lengths based on the labelled timestamps, as opposed to splitting every second into a segment as done by Kennedy and Ellis [2004]. The paper also manually filters out laughter that co-occurs with speech and unvoiced laughter. In the feature extraction step, they use PLP[Hermansky, 1990], which is a spectral feature similar to MFCCs. Additionally, the pitch is taken into account to investigate whether prosodic features can contribute to the laughter detection task. The paper employs a Gaussian Mixture Model as a classifier to train on the ICSI corpus and shows that spectral features are more effective than prosodic features for this task.

Knox and Mirghafori [2007] goes further on laughter detection on the ICSI corpus and advances the state-of-the-art in this field by proposing a more realistic approach to real-time laughter detection without the need for pre-segmentation of the data. The authors experimented with both Support Vector Machines (SVM), which were used in Kennedy and Ellis [2004], and neural networks, and found that SVM was not suitable for laughter detection without pre-segmentation due to its limitations in terms of time and storage consumption and poor resolution in detecting laughter in a fine-grained manner. The paper announces that the limitations in SVM can be solved using a neural network. Knox and Mirghafori [2007] use MFCCs as a basic feature extraction method and combine several prosodic features (fundamental frequency, local root mean squared energy, and the fundamental frequency) with it. They employed a simple neural network with one hidden layer and input the data frame by frame, with a frame size of 10 ms. For each frame, the feature of a context window with a size of 75 frames was also inputted into the model. The input data structure is illustrated in Figure 2.3.

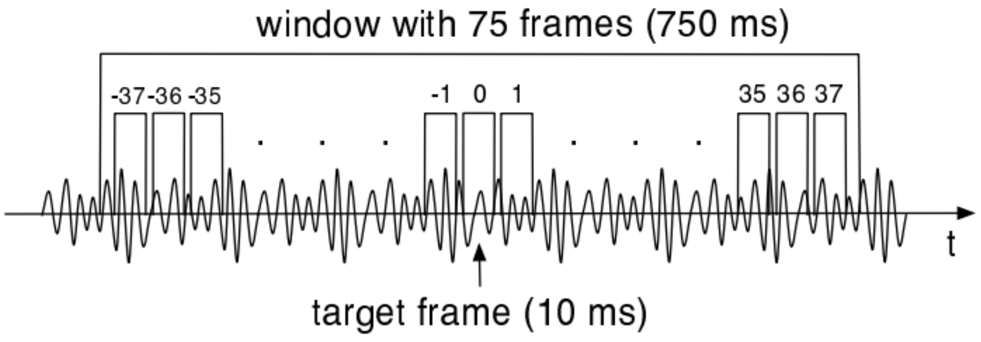


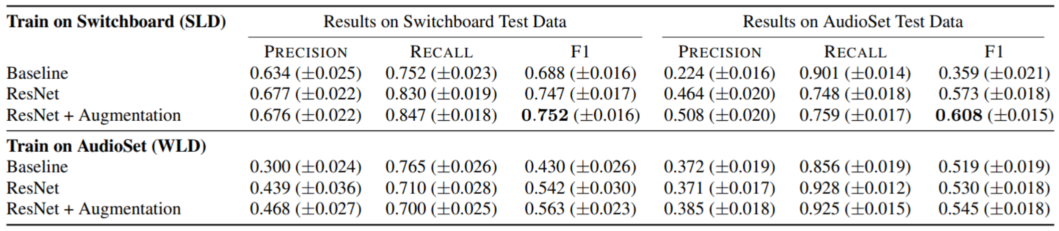
Figure 2.3: Structure of input data from Knox and Mirghafori [2007]

In further pursuit of the optimal techniques for detecting laughter in audio data, researchers Gupta et al. [2013] and Kaushik et al. [2015] explored various methods commonly used in audio event detection. Specifically, they aimed to determine which methods are most effective for detecting non-verbal cues such as laughter and fillers. Their investigation ultimately concluded that the Convolution Neural Network (CNN) is the most efficient approach for detecting these acoustic features.

Gupta et al. [2013] compare different techniques that are historically commonly used in audio event detection. Techniques that the paper discussed include Conditional Random Fields (CRFs)[Lafferty et al., 2001], Support Vector Machines (SVMs), Hidden Markov Models (HMMs), Deep Neural Networks (DNNs), Gaussian Mixture Models (GMMs) and Maximum Entropy models[Phillips et al., 2004]. They apply laughter and fillers detection on the SSPNet Vocalization Corpus[Polychroniou et al., 2014] and showed that DNNs outperformed other techniques for laughter and fillers detection.

Following this study, Kaushik et al. [2015] further compared the performance of DNNs and Convolution Neural Networks (CNNs) for laughter and fillers detection. They conducted experiments on the Switchboard [Godfrey et al., 1992] and Fisher [Cieri et al., 2004] corpora, and concludes that the CNN performs better on DNN, and any model with a simple low-pass filtering (LPF) can achieve better performance.

More recent work on laughter detection is from Gillick et al. [2021], they developed a robust state-of-the-art method for detecting laughter using CNN. They aim to find a way that can deal with laughter events in noisy environments. They hypothesize that the ResNet-18 should perform better on noisy data, because the large number of layers can make the model learn features more specific to the sound of laughter. To test their hypothesis, Gillick et al. [2021] constructed three models: the baseline model from their previous work[Ryokai et al., 2018], which is a feed-forward network, the ResNet-18 architecture that was modified for binary classification, and the ResNet-18 with data augmentation. Figure 2.4 is the result of the three models trained on Switchboard[Godfrey et al., 1992] and AudioSet [Gemmeke et al., 2017] corpus. The results indicate that the ResNet-18 architecture outperforms the other models, and that data augmentation techniques do not significantly improve the performance of ResNet-18.



Train on Switchboard (SLD)	Results on Switchboard Test Data			Results on AudioSet Test Data		
	PRECISION	RECALL	F1	PRECISION	RECALL	F1
Baseline	0.634 (±0.025)	0.752 (±0.023)	0.688 (±0.016)	0.224 (±0.016)	0.901 (±0.014)	0.359 (±0.021)
ResNet	0.677 (±0.022)	0.830 (±0.019)	0.747 (±0.017)	0.464 (±0.020)	0.748 (±0.018)	0.573 (±0.018)
ResNet + Augmentation	0.676 (±0.022)	0.847 (±0.018)	**0.752** (±0.016)	0.508 (±0.020)	0.759 (±0.017)	**0.608** (±0.015)
Train on AudioSet (WLD)						
Baseline	0.300 (±0.024)	0.765 (±0.026)	0.430 (±0.026)	0.372 (±0.019)	0.856 (±0.019)	0.519 (±0.019)
ResNet	0.439 (±0.036)	0.710 (±0.028)	0.542 (±0.030)	0.371 (±0.017)	0.928 (±0.012)	0.530 (±0.018)
ResNet + Augmentation	0.468 (±0.027)	0.700 (±0.025)	0.563 (±0.023)	0.385 (±0.018)	0.925 (±0.015)	0.545 (±0.018)

Figure 2.4: Result from Gillick et al. [2021]

2.3 Computational Complexity of Convolution Neural Network

As we also focus on improving the efficiency of the laughter detector, we will do some review on how to reduce the computational complexity of CNN.

Most of the work of laughter detection focuses on improving the accuracy, precision or recall of the detection task. However, our goal is to develop a real-time laughter detection tool for online meeting apps, which means the inference time for the model is important because we want small latency when the model is used in real life. Therefore we need to explore some ways to improve the inference time of the model.

One approach to improving the inference time of neural networks is to modify their architecture and simplify the model prior to training. In this regard, Howard et al. [2017] proposed a lightweight CNN model for mobile and embedded vision applications, called MobileNet. The authors utilized depthwise separable convolutions [Mamalet and Garcia, 2012], which convert a normal convolution layer into a depthwise convolution and a 1x1 convolution known as a pointwise convolution. This technique significantly reduces the model size and computation required. The MobileNet architecture, shown in Figure 2.5, achieves a substantial decrease in the number of trainable parameters while maintaining comparable accuracy to other network architectures such as VGG [Simonyan and Zisserman, 2014], InceptionV2 [Ioffe and Szegedy, 2015], and Squeezenet [Iandola et al., 2016].

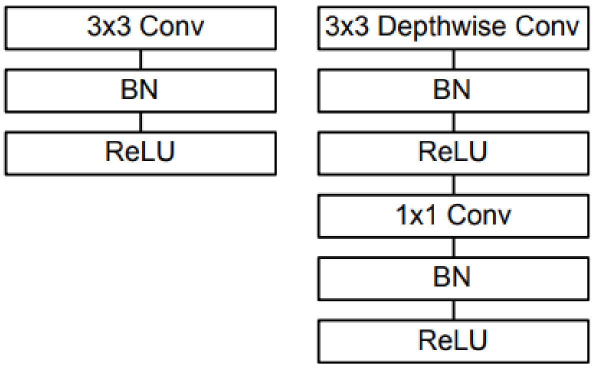


Figure 2.5: Left: Standard convolution layer with the batch norm and RELU activation. Right: Depthwise separable convolutions with the batch norm and RELU activation [Howard et al., 2017]

Another way to simplify the model before training is to simply shrink the size of the model. A smaller model will have a smaller number of parameters, which leads to a lower inference time for the same architecture. According to Tan and Le [2019], there are three factors that researchers usually modify to change the size of a model: depth, width, and resolution. They propose that balancing these factors carefully can improve performance. They develop a way to uniformly scale up the depth, width and resolution of a network by using a factor called compound coefficient. A family of models called EfficientNets is therefore created by modifying the compound coefficient.

Knowledge distillation is an alternative approach to model compression, distinct from modifying the architecture or size of models. Rather, knowledge distillation aims to compress the knowledge learned by a complex and computationally expensive model into a simpler, more lightweight model. The idea of knowledge distillation was first introduced by Hinton et al. [2015].

When the model is trained and all weights are fixed, there is still a method that can reduce the time takes for inference: quantization. Quantization is applied after the model has been trained, by converting the matrix multiplication and matrix addition in the model from float32 to int8, resulting in faster calculations. The mechanism behind it is that calculation for int8 is faster than float32 for CPU. Jacob et al. [2018] have proposed a quantization scheme for inference, which employs integer-only arithmetic. The scheme quantizes the float calculations to int calculations and then dequantizes the result back to float to achieve accuracy.

2.4 Evaluation methods

In order to comprehensively evaluate the performance of our proposed approach, it is important to consider both its accuracy and efficiency. To this end, we will introduce relevant metrics that have been utilized throughout our thesis to measure performance in both aspects.

2.4.1 Evaluate the accuracy

AUC (area under curve), precision, recall and F1 score are standard metrics for performance evaluation ren claim that AUC shows similar effectiveness for different models, while precision, recall and F1 score can tell the difference. Hence precision, recall and F1 score will be considered more in our project. To calculate these three factors, we need the following:

- False Positive (FP): Laughter samples that are not classified as laughter
- True Positive (TP): Laughter samples that are correctly classified as laughter
- False Negative (FN): Non-laughter samples that are not classified as non-laughter
- True Negative (TN): Non-laughter samples that are correctly classified as non-laughter

The formula of precision, recall, and F1 is then as the following:

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1 = 2 \times \frac{P \times R}{P+R} \text{ where } P \text{ is } Precision \text{ and } R \text{ is } Recall$$

What is more, follow by Wolter [2022a]'s work, we also split the non-laughter samples into speech, noise and silence, and use a confusion matrix to display the distribution of laughter that is wrongly classified (Figure 2.6). It is noteworthy that the confusion matrix used in our report differs from the conventional one, as our model functions as a binary classifier, only capable of determining whether a sample is a laughter event or not. Hence, information such as whether a speech sample is correctly labelled as speech remains unobserved. Therefore, as the normal confusion matrix will show the true label of samples on the x-axis and the predicted value of samples on the y-axis, our matrix will show samples that are predicted as laughter events on the y-axis at different thresholds and ignore the information such as samples predicted as speech. The threshold act as the boundary for the model to decide whether a segment can be classified as laughter, as the model outputs the probability of a sample being a laughter event. For example, the first line of the matrix represents the actual label of samples that were predicted as laughter events under the threshold of 0.2. And this lead to the fact that each line should sum up to 1.

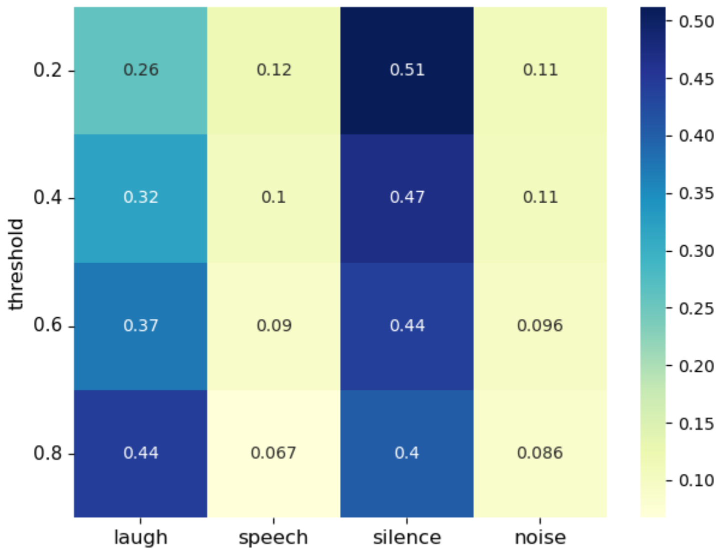


Figure 2.6: Sample confusion matrix

2.4.2 Evaluate the speed

As the performance of our model is expected to be optimized for real-time online meetings, it is crucial to assess its inference speed. To this end, we adopt two metrics in this report: floating point operations (FLOPs) and real-time factors. The FLOPs refer to the mathematical operations performed during the convolution and pooling layers. This metric has been widely used in previous works that aim to develop fast and lightweight embedded models, such as Sandler et al. [2018] and Tan and Le [2019]. In our evaluation, we calculate FLOPs using the THOP library [Lyken, 2022].

On the other hand, the real-time factor is a measure of the time required for a system to complete a task or process, relative to the duration of the task or process itself. In this task, the real-time factor of an audio data segment is calculated as:

$$RTF = \frac{time\ to\ process\ the\ segment}{duration\ of\ the\ segment}$$

In our project, we evaluate the real-time factor under both the CPU and GPU, with a greater emphasis on the CPU factor as the target platform for the laughter detector is a laptop and mobile phone. The CPU real-time factor is calculated under Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, and the GPU real-time factor is under NVIDIA GeForce

GTX 1060 6GB, both of which are available in the MLP cluster of the University of Edinburgh. In our evaluation, we calculate the real-time factor for all the models in a single remote session, with each model evaluated on 60-second audio segments for 10 iterations to obtain an average. And the number of threads we use for inference is 1. We write a data loader for evaluating the real-time factor, in which the test sample is provided to the model one by one other than in a batch. To avoid the effect of the initial slow startup, we set up a warm-up phase for both the CPU and GPU before each evaluation.

Chapter 3

Evaluate on previous work

Wolter [2022a] made progress on this project in the year 2021-2022. He developed an efficient approach for data loading and preprocessing and built a CNN model based on the ResNet-18 architecture proposed in Gillick et al. [2021]'s work. Gillick et al. [2021] trained the model on the Switchboard corpus, while Wolter [2022a] retrains the model with the ICSI corpus Janin et al. [2003]. For the purpose of establishing a baseline, his model will be considered in this project, with the aim of achieving improved accuracy and speed on the same ICSI corpus. To facilitate comparison, the same ICSI corpus and Wolter's data preprocessing pipeline [Wolter, 2022b] is employed, although some errors in this pipeline were noted by us and required rectification.

3.1 Model selection of Wolter

Gillick et al. [2021] use the Switchboard corpus [Godfrey et al., 1992] to train a CNN model based on ResNet-18 architecture He et al. [2016] to detect laughter and claim that their model is strong enough to deal with noisy environments. Wolter uses the open source repository from Gillick et al. [2021] as a reference and switches the corpus from Switchboard to ICSI [Janin et al., 2003].

3.2 ICSI corpus

The ICSI corpus has been widely used in the field of laughter detection, as demonstrated in previous studies such as Knox and Mirghafori [2007], Kennedy and Ellis [2004], and Truong and Leeuwen [2005]. This corpus is well-suited for our project, as it contains a collection of recorded meetings with detailed transcriptions that provide the start and end times of audio events. The corpus comprises 75 meetings with a total duration of 72 hours, including 3.9 hours of laughter events. The meetings were conducted in a conference room with 3-10 people present, and the audio was recorded using close-talk microphones for each participant, as well as 6 table-top microphones. Additionally, the ICSI corpus is licensed under CC BY 4.0, making it suitable for use in commercial projects.

3.3 Preprocessing

In the preprocessing step, we need to split the ICSI corpus into samples and use the feature extraction method to convert the samples from waves to feature vectors.

Wolter uses the library called Lhotse[Żelasko, 2021] to create a pipeline[Wolter, 2022b] for fast data loading. In his pipeline, the ICSI corpus is first split into 5 types of events according to the transcripts:

- laughter: segments contain only laughter
- speech: segments contain only speech
- invalid: segments contain laughter with other sounds (e.g. laughter after speech)
- noise: segments contain non-vocal voice such as footsteps and knocking
- silence: segments with no transcripts

Table 3.1 displays the duration of each event type in the ICSI corpus, from which we filter out the invalid segments for both training and testing, as they do not contain pure laughter. Notice that even though we split the corpus into 5 different types of events, our model is essentially a binary classifier, which is only able to label 'invalid', 'speech', 'noise', and 'silence' as 'non-laughter'. The different types of non-laughter are only used in the evaluation step, which helps us to analyse the distribution of miss-classified laughter events.

After segmenting the corpus, the Lhotse toolkit is employed to further divide the segments into one-second samples. Wolter fine-tunes two hyper-parameters: the number of generated laughter samples and the ratio between laughter and non-laughter samples. For each segment, a specified number of one-second laughter samples are produced, and for each laughter sample, a fixed number of one-second non-laughter samples are extracted from the segment. For the purpose of this project, we set the number of laughter samples to 5 and the ratio to 1:20. This indicates that we will generate five laughter samples and 20 non-laughter samples for each segment. This distribution is chosen based on Wolter's experiments, which show that it provides the highest precision.

Finally, we apply the feature extraction method, MFCCs, to convert the samples from raw wave data into feature vectors. These features, align with the laughter and non-laughter labels, are the input for model training.

event type	number of segments	total length
laughter	8415	3.9h
invalid	3765	3.7h
speech	98418	64.1h
noise	18224	10.9h
silence	97431	681.9h

Table 3.1: The total duration and number of segments of different events in the ICSI corpus.

3.4 Problems in Wolter's work

There are some things that need to be fixed in Wolter's pipeline. First is that some of the methods in Lhotse[Żelasko, 2021] have breaking changes, which make the pipeline built by Wolter crash. It takes some time for me to make the pipeline work with the help of the second supervisor. Second is some mistakes in preprocessing and training process that will cause a reduction in performance. These mistakes will be discussed in this section.

3.4.1 Laughter in Non-transcribed Regions

One of the major issues that affect the accuracy of the laughter detection model is the presence of sound in non-transcribed regions. As outlined in section 3.3, we classified these regions as silent and labelled them as non-laughter. However, Wolter's evaluation of the model using a confusion matrix (Figure 3.1) revealed a problem. The dark blue shading in both the first column (laugh) and the third column (silence) indicates that many true laughter events were mistakenly classified as either laughter or silence.

Upon further investigation, Wolter discovered that there were indeed sounds present in the silence segments, which had not been accounted for in the labelling process. Due to time constraints, he was unable to address this issue and left it for future work.

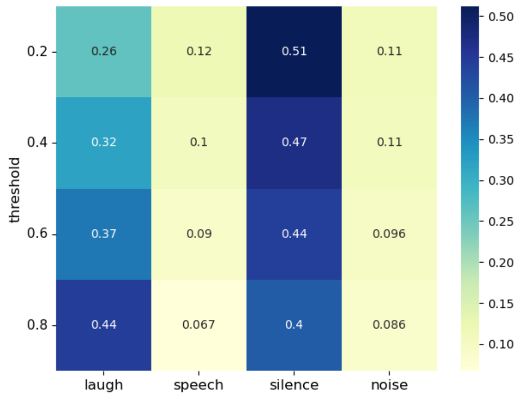


Figure 3.1: Confusion matrix for model trained on original dataset and test on original dataset

Plan of Attack:

By looking carefully into the code, I find out that the problem lies in the incorrect method of splitting the transcripts. In the ICSI corpus, each meeting has one transcript, which records the start time and end time of each sound, as well as the participants who make the sound. However, Wolter split the transcript into sub-transcripts for each participant, resulting in situations where other participants' voices were not labelled in the sub-transcripts, despite being recorded by the microphone. For example, in Figure 3.2, raw data for the close-talk microphone of participant A will record the sounds made by B since they sit in the same meeting room, but the sub-transcript of A will not label B's sound.

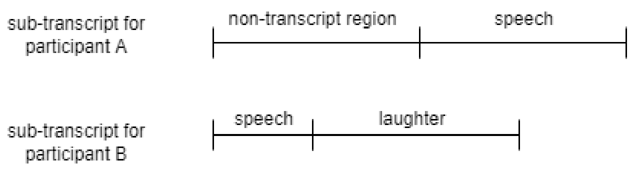

sub-transcript for participant A
non-transcript region
speech
sub-transcript for participant B
speech
laughter

Figure 3.2: An example clip of transactions for Participant A and Participant B who are in the same meeting.

To address this issue, the overlapping regions were split out from all sub-transcripts, and the segmentation was redone. The new segments were classified into 5 types of non-laughter events: invalid, speech, noise, silence, and false silence, with the latter referring to overlapping regions. Using the dataset with the new segmentation, the model trained on the old dataset was evaluated (Figure 3.3), and the resulting confusion matrix showed that the dark blue column shifted to the false silence category. This situation is reasonable since the sounds of other participants are included in the 'false silence' and the 'silence' event is now the pure silence.

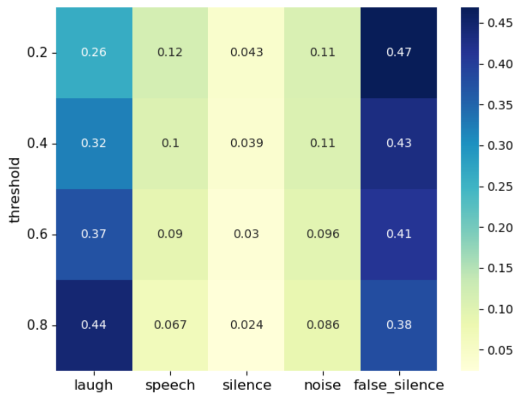


Figure 3.3: confusion matrix for the model trained on the old dataset and test on new dataset

We then treat the 'false silence' the same as the 'invalid', and completely remove it from the dataset. A new model is then trained and evaluated on the new dataset. Figure 3.4 displays the confusion matrix of the model trained on the new dataset, and we can see that only a small amount of laughter is miss-classified as silence.

What is more, some further data cleaning can be processed to remove the other participants' laughter from the 'speech' and 'noise' segments. We hypothesise that this will decrease the miss-classification in the 'speech' and 'noise' segments. However, regenerating the dataset is a time-consuming task, so we may leave it to further work.

3.4.2 Epoch setting

In the training of convolution neural networks (CNN), the number of "epochs" plays a critical role as it determines the number of iterations over the entire training dataset.

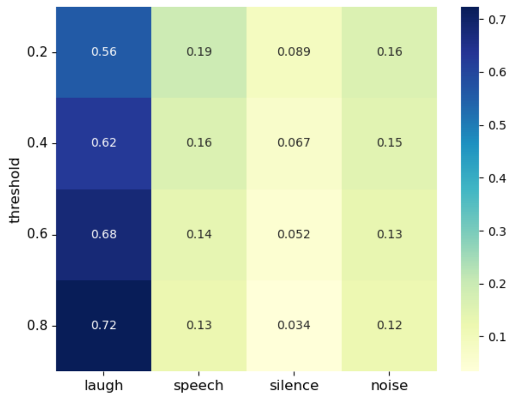


Figure 3.4: confusion matrix for model trained and tested on new dataset

With each epoch, the network has an opportunity to learn from the complete dataset multiple times. As the number of epochs increases, the model can continue to refine its parameters and improve its accuracy on the training data. However, if the number of epochs goes too big, the model will have the problem of overfitting, which means it performs well on the training data and poorly on unseen data.

Wolter set the epoch number manually in his code as a parameter, and he does not announce the epoch number he uses to train the model. Manually setting epoch numbers has many disadvantages. A large epoch number leads to a long training time and the problem of overfitting, whereas a small epoch number may prevent the model from fully learning from the data. Thus, selecting an appropriate epoch number is critical to the performance of the model, and an automated approach to determine the optimal number would be preferable.

Plan of Attack:

To decide the suitable epoch number automatically, I apply a method called 'Early Stop', which can automatically stop the training before the model starts to overfit. To use Early Stop, our dataset is split into the training set, validation set and test set. During the training process, for each epoch, we train the model on the training set, and evaluate the performance on the validation set. If the performance of validation data starts to decrease, we will consider the model begin overfitting, and stop the training process. The Early Stop has a parameter called 'patience'. 'patience' decide how many epoch the Early Stop can endure for getting a poorer performance than the historical best performance. The reason we set this parameter is that a fluctuation in performance for the validation set is normal, we need to make sure only stop the training when the decrease in the performance becomes a general trend.

3.4.2.1 Evaluate the Effect of Early Stop

To assess the impact of the Early Stop technique, we conducted an experiment in which we trained three models with varying epoch settings. These models were trained on the revised dataset discussed in Section 3.3.1 and shared the same architecture as Wolter's model. The first model, with a single epoch, was considered the baseline. The second model employed Early Stop with a ’patience’ value of 3, while the third model used Early Stop with a ’patience’ value of 7. The choice of ’patience’ value is critical, as a larger value would result in a more cautious approach towards stopping the training process.

epoch setting	precision	recall	F1
baseline (1 epoch)	1	0	0
Early Stop 3	0.56	0.51	0.53
Early Stop 7	0.62	0.48	0.54

Table 3.2: precision, recall and F1 score for model with different epoch setting

The performance of the three models is shown in Table 3.2, and we can see that the baseline model failed to learn from the data. This model has a precision of 1 and a recall of 0, which indicates that it incorrectly predicted all samples as non-laughter events. The F1 score of the model with Early Stop 3 and the model with Early Stop 7 is almost the same, which means the overall performance of these two models is similar. Therefore, we conclude that Early Stop can prevent the model from overfitting during the training process, and increasing the ’patience’ parameter leads to a more cautious approach in terminating training when overfitting occurs.

3.4.3 Distribution Mismatch Between Training and Test Data

During the training process, I noticed that the distribution of the training set, validation set and test set is different. The training and validation set have the same data distribution, which is 1:20 for laughter events and non-laughter events. While the test set uses whole meetings from the raw ICSI corpus as input, in which the laughter distribution is definitely not 1:20.

Plan of Attack:

I observe that the performance of the model on the training set and validation set is very good, in which the precision and recall both exceed 0.9, while the performance on the test set is around 0.5. I hypothesise that the low performance of the test set is due to the distribution mismatch between training and test data. To test this hypothesis, I create a small test set that uses one of the whole meetings in the training set. And find that the recall and precision of this test set are not high as it performed in the training set, which means that even if the model is trained by segments in this meeting, when the distribution of laughter event changes, the performance will decrease. CNN models can still learn from the data when the distribution of the training set and real word data is different, but we would expect the network will perform better when the distribution of the training set and the test set is the same. However, regenerating the dataset is time-consuming work, and we already trained lots of models based on the training set with a distribution of 1:20. Therefore, we decide to put it into further work.

Chapter 4

Improving Convolution Neural Networks for Laughter Detection

Wolter [2022a] follows the paper of Gillick et al. [2021], which uses a modified version of ResNet-18 as the laughter detector. Considering Wolter's model as the baseline of our project, we aim to investigate how different sizes and architectures of the laughter detector affect performance in terms of accuracy and speed.

4.1 Methodology

4.1.1 Training and Evaluation pipeline

The pipeline of the training and evaluating models is shown in Figure 4.1, in which we will train the laughter detector with different sizes and architectures during the training step. We use the ICSI corpus, consisting of 72 meetings, and follow Renals and Swietojanski [2014]'s partition scheme to divide the ICSI corpus into training, validation and test sets. Specifically, meetings Bmr021 and Bns001 are used for validation, meetings Bmr013, Bmr018, and Bmr021 for testing, and the remaining 67 meetings for training.

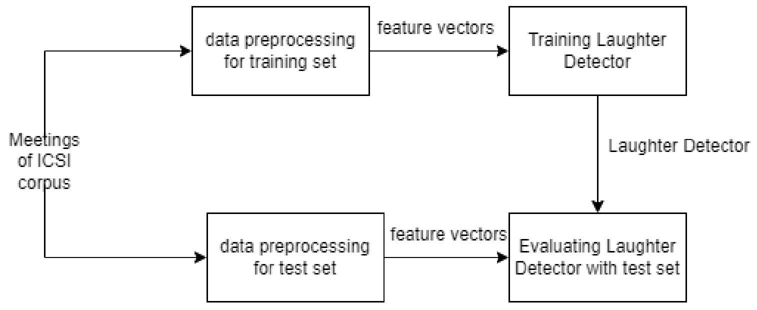

data preprocessing for training set
feature vectors
Training Laughter Detector
Meetings of ICSI corpus
Laughter Detector
data preprocessing for test set
feature vectors
Evaluating Laughter Detector with test set

Figure 4.1: pipeline for training and evaluating a model

In the data preprocessing step, we have maintained consistent training sets for each model by controlling certain parameters. Specifically, we have set the distribution of laughter samples and non-laughter samples to be 1:20, and each sample has a duration of 1 second. The non-laughter samples follow the distribution of 70 silence:20 speech:10 noise in the training set. In total, the training set will contain 258 thousand samples. We use MFCCs as feature extraction methods to convert the raw sample data to a list of feature vectors with a dimension of 44. Each 1-second sample is represented as a format of 100 feature vectors when fed into the CNN.

For the training step, we keep some of the hyper-parameter the same for different models. Specifically, we have set the Early Stop with a patience of 7 and the learning rate as 0.01 for each model. The Early Stop has been discussed in Section 3.3.2, while the learning rate is a control variable that determines the step size at which the model weights are updated during training.

4.1.2 Architecture of CNN

The CNN is a kind of deep neural network with convolution layers. The architecture of a typical CNN can be divided into several layers, each of which serves a specific purpose. Typical layers include:

- Convolution layer: This layer applies a set of filters to the input data, which takes the input to produce a set of feature maps.
- Fully connected layer: This layer connects all neurons in the previous layer to every neuron in the current layer, producing a vector of features that represents the input data.
- Dropout layer: This layer randomly sets a fraction of the input units to zero during training, which is used to prevent overfitting.
- Pooling layer: This layer reduces the size (i.e., width and height) of the input feature map.
- Output layer: This layer produces the final output of the network. Since our laughter detector is a binary classifier which outputs ’laughter’ or ’non-laughter’, the output layer we use is Sigmoid, which will output the probability of a sample being laughter.

There are two hyper-parameter that are commonly used to modify the size of CNN: the number of layers and channel size. The number of layers is the number of convolution and fully connected layers. And the channel size refers to the number of feature maps in a given layer. In our project, the input data consists of a single feature map with a size of 100x44, as previously noted in Section 4.1.1. The number of channels increases as the data passes through the convolution layers, which apply multiple filters to extract increasingly abstract features from the data. Each filter generates a new feature map, leading to a corresponding increase in the number of output feature maps.

4.2 Experiment 1: Baseline model and ResNet-18

In the first experiment, we compare a modified version of ResNet-18 used by Wolter and Gillick et al. [2021] with a standard ResNet-18 from the Torchvision library[Falbel, 2022]. Torchvison is a widely used library which implements many different CNN architectures. The implementation of ResNet-18 in Torchvison strictly follows the architecture introduced in the original ResNet paper by He et al. [2016]. We want to know the difference between the modified version of ResNet-18 and the standard ResNet-18 proposed by He et al. [2016] because Wolter and Gillick et al. [2021] did not explain how or why their model was modified.

ResNet is designed to address the problem of vanishing gradients in deep neural networks, which can make it difficult to train models with many layers. ResNet achieves this by using residual connections, which allow information from earlier layers to bypass later layers and flow directly to later layers in the network. The architecture of ResNet is based on a series of blocks, each of which contains one or more convolution layers and a shortcut connection. Figure 4.2 displays the basic structure of the block in ResNet, in which He et al. [2016] named it as a residual block. The difference between normal CNN and ResNet is the red line, which is the residual connections. The input 'x' from the first layer is directly passed to the output of the second convolution layer, and then the combination of 'x' and the output of the second convolution layer will be considered as input to the next layer. Figure 4.3 is directly extracted from the paper of He et al. [2016] and displays the difference between a normal CNN with 32 layers and a ResNet with 32 layers. We can see that there are residual connections for each two convolution layers.

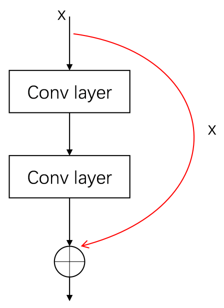


Figure 4.2: Residual block in ResNet

ResNet-18 is an architecture that consists of 8 residual blocks and has 18 layers in total. Table 4.1 shows the structure of the baseline model and the standard ResNet-18 model. The main difference is the channel size of each residual block. The channel size of the baseline is 64, 32,16,16 respectively, while the channel size for standard ResNet-18 is 64,128,256,512, which means the ResNet-18 has a larger channel size in total and is supposed to perform better than the baseline. We can also see that the baseline has one more fully connected layer than the standard ResNet-18, with two Dropout layers. However, it is hard to judge whether is modification is better than one fully connected layer.

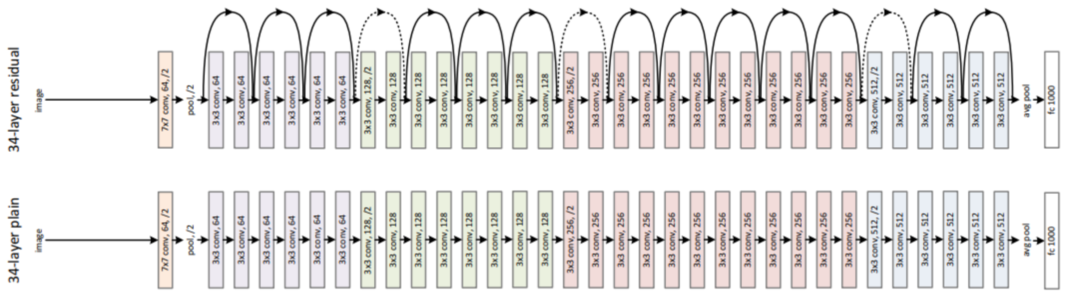


Figure 4.3: The upper one is the structure of ResNet-32 and the lower one is a normal CNN structure with 32 layers. This Figure is from He et al. [2016]

# Residual Block	baseline		standard ResNet-18	
	Operator	input size	Operator	input size
	conv 3x3v 64	1x100x44	conv 7x7 64	1x100x44
			MaxPooling 3x3, stride 2	64x50x22
x2	Residual Block 3x3 64	64x100x44	Residual Block 3x3 64	64x25x11
x2	Residual Block 3x3 32	64x100x44	Residual Block 3x3 128	64x25x11
x2	Residual Block 3x3 16	32x50x22	Residual Block 3x3 256	128x13x6
x2	Residual Block 3x3 16	16x25x11	Residual Block 3x3 512	256x7x3
	AveragePooling	16x13x6	AveragePooling	512x4x2
	Dropout 0.5			
	fully connected			
	Dropout 0.5			
	fully connected			
	Sigmoid		Sigmoid	

Table 4.1: Structure of baseline and standard ResNet-18. The three parameters in input size is channel size, feature map height and feature map width respectively.

After training the baseline and ResNet-18 models, we used the test set to evaluate their performance. Table 4.2 presents the evaluation results for both models. The first three columns show the precision, recall, and F1 score, which indicate the accuracy of the models. The parameter num column shows the size of the models, and the remaining columns show their performance in terms of speed.

Regarding accuracy, we observe that both models have similar F1 scores. However, ResNet-18 achieves higher precision than the baseline, with a 15% increase. The decrease in recall for ResNet-18 is relatively small, at 5%. Since our laughter detection task places greater emphasis on precision, which is critical for protecting user privacy, ResNet-18 is a more suitable choice than the baseline.

In terms of speed, we evaluated the models based on their FLOPs and real-time factor on both GPU and CPU. Since our goal is to develop a model that can run on a regular laptop, we were primarily concerned with the CPU real-time factor. We found that the ResNet-18 model is a bit slower than the baseline in terms of CPU real-time factor.

Table 4.2 shows that the FLOPs of ResNet-18 are much smaller than those of the baseline, indicating that ResNet-18 is expected to be faster. It is interesting to note that ResNet-18 has smaller FLOPs than the baseline, even though the number of parameters of ResNet-18 is much larger. First, we need to know what will influence the value of FLOPs and the number of parameters. The FLOPs is the number of floating-point operations in a layer, while the number of parameters calculates the learnable parameters of a model. The FLOPs include one more factor than the number of parameters, which is the size of the feature map. The first two layers of ResNet-18, which are a convolution layer and a MaxPooling layer, reduce the size of the feature map from 100x44 to 25x11, while the baseline does not change the size of the feature map. The feature map will be fed into the following residual blocks, so the size of feature map in every residual block of ResNet-18 is smaller than the baseline. Since the number of filters in ResNet-18 in each residual block is larger than those in the baseline, the number of parameters of ResNet-18 is bigger. Therefore, despite having a bigger parameter number, the overall FLOPs of ResNet-18 is less than the baseline.

	precision	recall	F1	param num	FLOPs	rtf_cpu	rtf_gpu
baseline	0.62	0.48	0.54	0.22M	22.87G	0.002707	0.000832
ResNet-18	0.77	0.43	0.55	11.7M	6.3G	0.00315	0.000647

Table 4.2: Structure of baseline and standard ResNet-18

In conclusion, the standard ResNet-18 is more suitable for the embedded laughter detection task than the baseline since it has a higher precision and F1 score. And the FLOPs of ResNet-18 is much smaller than the baseline even though the real-time factor is almost the same.

4.2.1 Issue in Evaluating the Real-Time Factor

It is worth noticing that there is an issue with analysing the real-time factor of models, which impacts the speed performance seriously. We will discuss this problem and introduce how we solve it in this section. Our evaluation of real-time factors has shown significant variability in the measured values on the same models. For example, I may get a real-time factor of 0.006 for ResNet-18 on day one and 0.003 for day two. This is due to running the project on a remotely connected cluster that provides GPU/CPU resources. For each evaluation, I will create a session and request resources. The cluster allocates different GPU/CPU resources for each session, and some CPU/GPU may be more powerful than others, leading to varying inference times for the same model in different sessions. To address this issue, we evaluated all models in one session. However, this led to two further problems.

Firstly, the time to infer the entire test set for each model was lengthy, and the cluster failed to allocate a GPU/CPU for such a long time to run all models on the entire test set. To solve this problem, we reduced the test set to 60 seconds. Secondly, we did not fully understand the resource allocation mechanism of the cluster, which meant that the cluster may allocate the GPU/CPU to other users at the same time, impacting the inference speed even when all models were run in the same session. This sharing of resources led to a variation in the real-time factor of the same model even when run twice in the same session. Therefore, when the real-time factors of the two models are close, they may not be reliable. Hence, we included FLOPs in our speed evaluation, which is the total number of floating-point operations required for a single forward pass. FLOPs is a stable constant that does not change once the model's architecture is fixed.

4.3 Experiment 2: Different Depth of ResNet

In the preceding section, we can see that the standard ResNet performs slightly better than the baseline, and this section develops further into the effects of ResNet architecture size on performance. Remember that the depth of a CNN architecture refers to the number of convolution and fully connected layers in the model. For instance, the standard ResNet-18 has 17 convolution layers and 1 fully connected layer. The previous research announces that the larger the depth, the more the model will fit into the training set. However, when a very deep CNN is used on a simplistic dataset, overfitting may occur. Therefore, determining the optimal number of layers for the laughter detector is crucial. We test ResNet architectures with depths of 10, 18, 34, and 50 (ResNet-10, ResNet-18, ResNet-34, and ResNet-50). The models with depths of 18, 34, and 50 are sourced from the Torchvision library, whereas the model with 10 layers was is conducted by myself.

The model with depths of 10, 18, and 34 has the same type of residual block, which is shown in Figure 4.2, whereas the ResNet-50 replaces the residual block with bottleneck block, a modified version of the normal residual block (Figure 4.4). The bottleneck block is utilized in ResNet-50 to decrease computational cost, since 50 layers is very deep for CNN and needs a huge amount of resources for training and inference. The bottleneck block has a similar residual connection structure to that of the standard residual block. However, instead of employing two convolution layers, the bottleneck block comprises three convolution layers. The first and the last layer of the bottleneck has the filter size of 1x1, which functions similarly to a fully connected layer. These 1x1 convolution layers are used to modify the channel size. The bottleneck block will first decrease the channel size of the input data by using the first layer of the block, then use the standard 3x3 convolution layer to the data with smaller channel size, and finally increase the channel size to match the input data coming from the side pass. Table 4.3 presents the structure of ResNet with depths of 10, 18, 34, and 50.

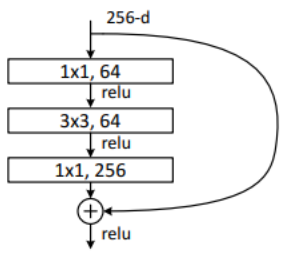


Figure 4.4: Bottleneck block in ResNet [He et al., 2016]

ResNet-10	ResNet-18	ResNet-34	ResNet-50
Conv 7x7 64			
MaxPooling 3x3, stride 2			
Residual Block x1 (64,64)	Residual Block x2 (64,64)	Residual Block x3 (64,64)	Bottleneck x3 (64,64,256)
Residual Block x1 (128,128)	Residual Block x2 (128,128)	Residual Block x4 (128,128)	Bottleneck x4 (128,128,512)
Residual Block x1 (256,256)	Residual Block x2 (256,256)	Residual Block x6 (256,256)	Bottleneck x6 (256,256,1024)
Residual Block x1 (512,512)	Residual Block x2 (512,512)	Residual Block x3 (512,512)	Bottleneck x3 (512,512,2048)
AveragePooling			
Fully Connected			
Sigmoid			

Table 4.3: Structure of ResNet with depth of 10,18,34,50

The performance of the four ResNet models with varying depths was evaluated using several metrics, as outlined in Table 4.4. In the accuracy aspect, the ResNet-18 has the highest precision, while the ResNet-34 and ResNet-50 have the highest overall F1 scores among the four models, which achieve 0.6. Notably, ResNet-50 outperformed ResNet-34 in precision. Therefore, ResNet-50 performs the best in the accuracy aspect. In the speed aspect, it is no doubt to see that the deeper the model, the slower it will be. We can see that the ResNet-34 and ResNet-50 are twice bigger CPU real-time factors than ResNet-18.

	precision	recall	F1	FLOPs	param num	rtf_cpu	rtf_gpu
ResNet-10	0.75	0.447	0.56	2.91G	4.9M	0.001563	0.000413
ResNet-18	0.77	0.43	0.55	6.3G	11.7M	0.00315	0.000647
ResNet-34	0.67	0.54	0.6	12.28G	21.28M	0.00609	0.001122
ResNet-50	0.73	0.51	0.6	14.4G	23.5M	0.006183	0.001651

Table 4.4: Evaluation of ResNet with depth of 10,18,34,50

In conclusion, the ResNet with a depth of 50 performs the best when considering the overall F1 score, while ResNet-18 performs the best when only considering the precision. Moreover, as expected, the speed of the inference process increase when the model becomes deeper.

4.4 Experiment 3: Different Width of ResNet

Both Depth and width are important factors that influence the size and performance of a model. In the previous section, we explore how the depth will influence the performance of models, and in this section, we will investigate how modifying the width of the ResNet can also affect its performance.

As we mentioned in section 4.1.2, the width refers to the number of channels of a convolution layer, which is determined by the number of filters present in the convolution layer. Figure 4.5 displays the process of how the convolution layer deals with the input feature map. In this case, there is 3 input feature map, and the convolution layer has 2 sets of filters. Each set of filters will be applied to the input feature maps get to produce an output feature map. Therefore the output channel size here is 2.

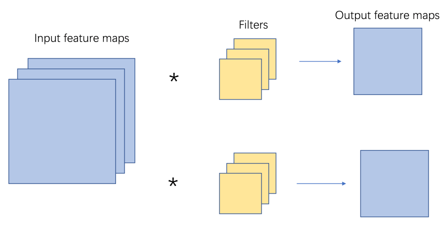


Figure 4.5: Process of how a convolution layer processes the input data, This layer has two sets of filters and the input channel size is 3.

The base model for this experiment is ResNet-18, which means we will modify its' width. The ResNet-18 has 8 residual blocks and every 2 blocks share the same width. The widths are set to 64, 128, 256 and 512 respectively for each 2 blocks, which is shown in Table 4.5. We modify the standard ResNet-18 architecture to get a ResNet-18 with a larger width and a ResNet-18 with a smaller width. The structure of the 3 versions of the ResNet-18 is shown in Table 4.5. The table displays the layer detail for each residual block. For instance, the first residual block of the three architectures is the same, which contains 2 layers with 64 filters with size of 3x3. Note that the ResNet-18 with smaller width has the same width setting as the baseline from Wolter [2022a], therefore we also include the baseline in our evaluation.

Table 4.6 presents the results of our evaluation on the modified ResNet-18 architectures with varying widths. From the table, we can see that the ResNet-18 with smaller width performs the best in accuracy, exhibiting the highest precision and F1 score. Moreover, all three ResNet models outperform the baseline in terms of the F1 score. And it is not surprising to see that the larger the width, the longer it takes to do the inference.

	precision	recall	F1	param num	FLOPs	rtf_cpu	rtf_gpu
baseline	0.62	0.48	0.54	221.22K	22.87G	0.002707	0.000832
ResNet-18-narrow	0.78	0.47	0.59	221.52K	1.56G	0.001063	0.000629
ResNet-18	0.77	0.43	0.55	11.7M	6.3G	0.00315	0.000647
ResNet-18_wider	0.728	0.492	0.587	18.12M	10.43G	0.0042	0.000656

Table 4.6: Evaluation of ResNet with three different width settings

After getting the evaluation results for ResNet architecture with different width and depth settings, we can draw a graph (Figure 4.6) with CPU real-time factor on x-axis and F1 score on y-axis. CPU real-time factor and F1 score are key factors in speed and accuracy respectively. From Figure 4.6, we can see that the F1 score first decrease and

layer	ResNet-18_narrow	ResNet-18	ResNet-18_wider
1	Conv 7x7,64 & MaxPooling		
2	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$
3			
4			
5			
6	$\begin{bmatrix}3\times3,32\\3\times3,32\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$
7			
8			
9			
10	$\begin{bmatrix}3\times3,16\\3\times3,16\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$
11			
12			
13			
14	$\begin{bmatrix}3\times3,16\\3\times3,16\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$
15			
16			
17			
18	AvgPooling & FC & Sigmoid		

Table 4.5: Architecture of ResNet with three different width setting, where FC represent Fully Connected Layer and each matrix represent residual block

then increase when the real-time factor becomes bigger. The lowest point is the baseline, and the point on the right of the baseline is the standard ResNet-18. Both the models that are smaller than standard ResNet-18(ResNet-18_narrow and ResNet-10) and models that are larger than ResNet-18(ResNet-18_wide, ResNet-32, ResNet-50) have higher F1 scores than the standard ResNet-18.However, it is expected that the curve would increase first, as the models become more complex and better fit the training data, before eventually decreasing due to overfitting. The opposite trend observed in our models suggests the occurrence of unexpected events during the training process. We guess it is caused by the incorrect setting of hyper-parameters, such as the learning rate. We will include this in Chapter 5, which is the conclusion and further work.

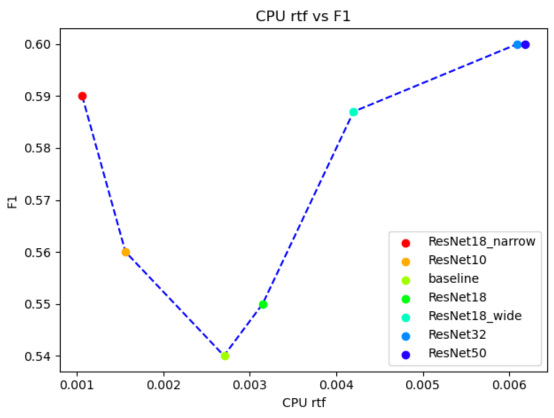


Figure 4.6: CPU real-time factor vs F1 score for ResNets with different depth and width

4.5 Experiment 4: ResNet with Depthwise Separable Convolution

The purpose of this section is to introduce the Depthwise Separable Convolution (DW) as a method to speed up the model's inference time. The idea of Depthwise Separable Convolution is proposed by Mamalet and Garcia [2012], and is used by Howard et al. [2017] to produce a state-of-art CNN architecture, MobileNet. The primary idea behind DW convolution is to replace the regular convolution layers with two sequential layers: depthwise convolution and pointwise convolution. This replacement reduces the number of parameters and calculations in the layer while maintaining accuracy.

The DW convolution can be split into two parts: depthwise convolution and pointwise convolution. Depthwise convolution applies a single filter per input channel, whereas pointwise convolution uses a simple 1x1 convolution to increase the number of channels. The normal convolution process can be viewed in Figure 4.7. This figure shows a convolution layer with the filters size of 3x3 and 64 channels is applied to the input feature maps, increasing the number of feature maps from 3 to 64. By contrast, in a DW convolution layer, the process is split into depthwise convolution (Figure 4.7) and pointwise convolution (Figure 4.9). As the input feature map has the channel size of 3, the depthwise convolution will apply 3 single 3x3 filters for the 3 input feature maps. Therefore the number of output feature maps for the depthwise convolution is the same as the number of input feature maps. Then we use the pointwise convolution to increase the number of channels. Since in this case, we want the channel size to be 64, we will apply 64 1x1x3 filters during the process. The total number of calculations and parameters in Figure 4.7 and Figure 4.9 is less than in 4.7. Therefore, by replacing the regular convolution layer with the DW convolution layer, we expect to speed up the inference process.

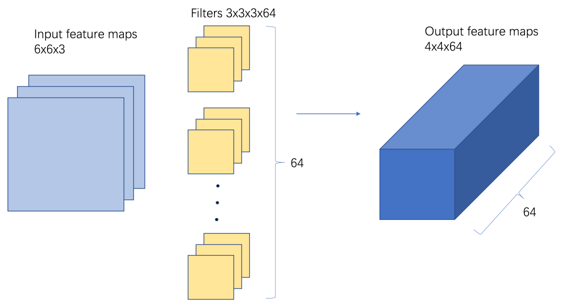


Figure 4.7: Process of a normal convolution layer to deal with the input data of size 6x6x3. The output channel for this layer is 64

We apply DW on three models: the baseline, standard ResNet-18 and ResNet-18 with a small width. Specifically, we replace all the convolution layers in the residual blocks of these models with DW layers. The evaluation is shown in Table 4.7. From the table, we can see that the FLOPs of all the models decrease efficiently after replacing the normal convolution layers with the DW layers. For the real-time factor, the CPU real-time factor of ResNet-18 is halved after using DW convolution, which is a significant improvement.

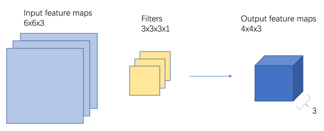


Figure 4.8: Depthwise convolution to deal with input data of size 6x6x3

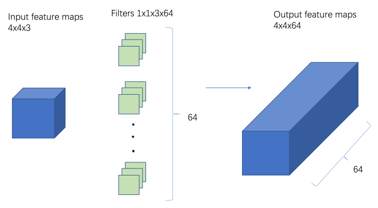


Figure 4.9: Pointwise convolution to deal with the output data from depthwise convolution. The final output channel size is 64

However, only the CPU real-time factor of baseline and ResNet-18 decreased, while the rest of the CPU and GPU real-time factors increased.

Regarding accuracy, the F1 score of the baseline and ResNet-18 with small width decreased when applying DW, while the F1 score for ResNet-18 remained almost the same. This result is not surprising since the reduction in accuracy is the trade-off of low FLOPs. However, it is noteworthy that the precision increased for all three models with DW. Considering precision is more important than recall in our project, we consider the cost of adding DW acceptable.

	model	precision	recall	F1	param num	FLOPs	rtf_cpu	rtf_gpu
models without DW	baseline	0.62	0.48	0.54	0.22M	22.87G	0.002707	0.000832
	ResNet-18	0.77	0.43	0.55	11.7M	6.3G	0.00315	0.000647
	ResNet-18_narrow	0.78	0.47	0.59	0.22M	1.56G	0.001063	0.000629
models with DW	baseline	0.773	0.322	0.455	0.0036M	3.42G	0.002049	0.000958
	ResNet-18	0.79	0.423	0.551	1.44M	0.95G	0.001433	0.000962
	ResNet-18_narrow	0.809	0.296	0.433	0.0037M	0.33G	0.001175	0.000975

Table 4.7: Evaluation of model with DW

Same as what we do in the last experiment, we plot a diagram (Figure 4.10) to visualize the relationship between CPU real-time factor and F1 score. The blue line in Figure 4.10 is exactly what we have in the last experiment, which shows the performance trend of ResNet architectures with different size settings (depth and width) without DW. The red line, on the other hand, represents the three models (ResNet-18_narrow with DW, baseline with DW, ResNet-18 with DW) we create using depthwise separable convolution. We are using the same colour to represent the model with the same depth and width settings. For example, the red dot on the blue line shows the performance of ResNet-18_narrow and the red dot on the red line displays ResNet-18_narrow with DW. From the figure we can observe that ResNet-18 with DW has a good performance, which has a huge improvement in speed, and without any loss in accuracy. However, the other two models with DW experienced a considerable decrease in the F1 score as the trade-off of the improvement in speed. Overall, the ResNet-18 with small width still has the best performance among all the models, which has a very high F1 score and the lowest CPU real-time factor.

In conclusion, applying depthwise separable convolution on the ResNet architecture can efficiently reduce the number of parameters and calculations of the model with the trade-off of decreasing in F1 score. However, the decrease in the F1 score is caused by

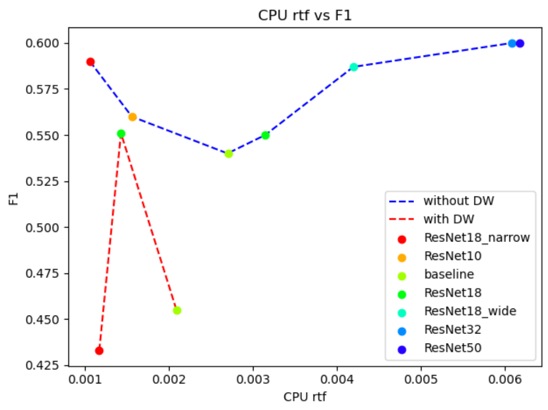


Figure 4.10: The blue line shows the relationship of CPU real-time factor and F1 score for ResNet models without applying depth separable convolution. The red line displays the models with depthwise separable convolution. The dots with the same colour means the two models have size settings.

the reduction in recall, while the precision of models with DW increases. Therefore, the DW can be a considerable method to improve the speed of our laughter detection.

4.6 Experiment 5: MobileNet and EfficientNet

Some state-of-art architectures are widely used in the industry for image-processing tasks. It is worth seeing whether these architectures are suitable for the laughter detection task. The architectures we are going to try are MobileNetV2 [Sandler et al., 2018] and EfficientNet [Tan and Le, 2019]. These architectures were selected due to their claims of high accuracy, smaller size, and faster inference times compared to other CNNs.

MobileNetV2 use the techniques of depthwise separable convolution in its architecture to achieve the goal of being a lightweight and fast CNN. The architecture of MobileNetV2 consists a group of ’inverted residual blocks’. From the name: inverted residual, we can notice that there is some similarity between the residual block used in ResNet. In fact, the inverted residual block is a modification of the bottleneck used in ResNet-50. Figure 4.11 and Figure 4.12 illustrate the difference between bottleneck and inverted residual. Notably, we can see that the input and output channel sizes of the middle layer in these two figures are different, and the type of middle layer also differs, with the bottleneck using a standard convolution layer while the inverted residual block uses a depthwise convolution layer. With the use of inverted residual blocks, the architecture of MobileNetV2, as shown in Table 4.8, comprises 17 inverted residual blocks (each with 2 PW layers and 1 DW layer), 1 standard convolution, and 2 fully connected layers, totaling 54 layers.

Subsequently, we turn our attention to the architecture of EfficientNet[Tan and Le, 2019]. Same as MobileNet, the EfficientNet also has the advantage of being a lightweight and fast CNN. However, the architecture of EfficientNet is more complex than that of MobileNetV2, as it combines the MobileNetV2 and another CNN structure, SENet

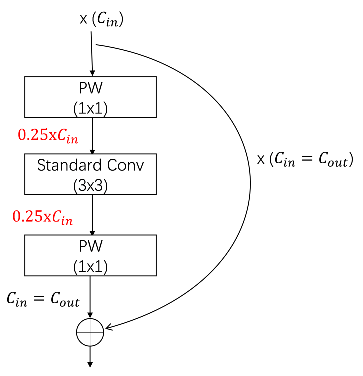


Figure 4.11: bottleneck in ResNet-50, where C_{in} is the input channel size and C_{out} is the output channel size

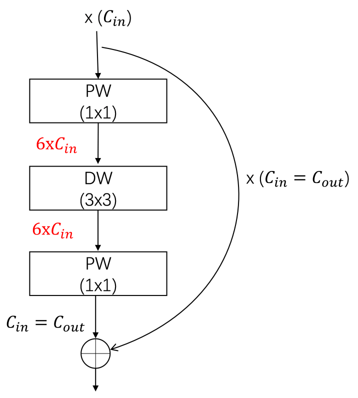


Figure 4.12: inverted residual in MobileNetV2, where C_{in} is the input channel size and C_{out} is the output channel size

# operator	Operator	output channel size
	Conv 3x3	64
x1	Inverted Residual	16
x2	Inverted Residual	24
x3	Inverted Residual	32
x4	Inverted Residual	64
x3	Inverted Residual	96
x3	Inverted Residual	160
x1	Inverted Residual	320
	FC&AvgPooling&FC	1280
	Sigmoid	

Table 4.8: Architecture of MobileNetV2, where FC represents Fully Connected Layer

[Iandola et al., 2016]. At a higher level, EfficientNet is constructed by a set of MBConv blocks, with each block uses the techniques of inverted residual from MobileNetV2 and squeeze-excitation technique from SENet. Specifically, each MBConv block can be simply viewed as an inverted residual block with a SENet structure inserted after the DW layer. Since the inverted residual block is already been discussed when we introduce the architecture of MobileNetV2, we will focus on the squeeze-excitation structure of SENet here. From Figure 4.13 we can see that the SENet first create a bypass on the output feature maps and apply a global pooling on it. This step is called 'squeeze' because the size of each feature map will be squeezed into 1x1. Then a process named 'excitation' comes after 'squeeze'. The key to 'excitation' is using two fully connected layers to learn weights from the squeezed feature maps.Finally, the original output feature maps are scaled up by the value obtained from the side pass. In the MBConv, the squeeze-excitation structure is directly concatenated after the DW layer of the inverted residual block, which means the input layer in Figure 4.13 is set to be the depthwise convolution layer.

The SENet has the advantage of increasing the accuracy with only a slight increase in computational cost, while the inverted residual is aimed at reducing the number of calculations with an acceptable loss in accuracy. Compared to MobileNet which only uses inverted residual blocks, we expect the EfficientNet to be slower but with higher accuracy than MobileNetV2 under the same size of the model.

The architecture of EfficientNet is shown in Table 4.9, which has 51 layers in total: 16 MBConv blocks, 1 standard convolution layer and 2 fully connected layers.

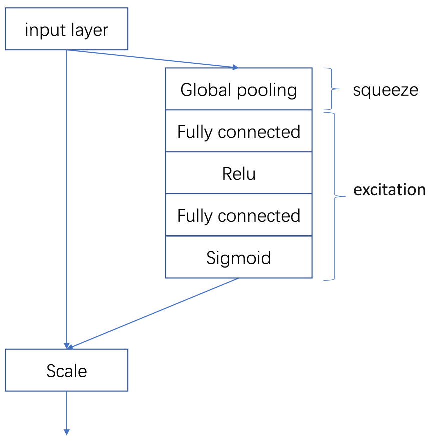


Figure 4.13: Structure of SENet

#operator	Operator	output channel size
	Conv 3x3	32
x1	MBConv	16
x2	MBConv	24
x2	MBConv	40
x3	MBConv	80
x3	MBConv	112
x4	MBConv	192
x1	MBConv	320
	FC&AvgPooling&FC	1280
	Sigmoid	1

Table 4.9: Architecture of EfficientNet, where FC represents Fully Connected layer

Both MobileNetV2 and EfficientNet adopt the standard implementation from the Torchvision library, which match the architecture used in the paper(Sandler et al. [2018] and Tan and Le [2019]). A Sigmoid layer is added to fit the binary classifier for the laughter detector. Although it would be better to control the complexity of the models at the same level, we employ the standard settings due to time constraints, but further work can be done by modifying the width and depth of the models.

During the training session of MobileNetV2, we encountered issues with the loss failing to decrease when the learning rate was initially set to 0.01. As a result, we changed the learning rate to 0.0001, which is recommended by the paper of MobileNetV2 [Sandler et al., 2018].

After the implementation of MobileNetV2 and EfficientNet, we then train these two models on our ICSI corpus and evaluate the performance. The result is shown in Table 4.10. Note that we also include the ResNet-18 in our table as a baseline. We use ResNet-18 as a comparison baseline because MobileNetV2 and EfficientNet can be considered as improved versions of ResNet-18. The inverted residual block and MBConv block of MobileNetV2 and EfficientNet, respectively, are considered as substitutions for standard convolution with the goal of reducing the number of calculations. Since MobileNetV2 has 17 inverted residual blocks and EfficientNet has 16 MBConv blocks, these blocks can be mapped to the 16 standard convolution layers in ResNet-18. Therefore, the comparison between the ResNet-18, MobileNetV2 and EfficientNet is valid.

Back to the evaluation result, Table 4.10 shows that MobileNetV2 has the lowest FLOPs value while ResNet-18 get the largest FLOPs. This completely follows our knowledge that the inverted residual block and MBConv block have a smaller number of calculations. The F1 score of MobileNetV2 and EfficientNet is slightly higher than the one in ResNet-18, while the F1 of MobileNetV2 is the highest. This is acceptable since the objective of MobileNetV2 and EfficientNet is to speed up inference with a small reduction in accuracy, and since the complexity of the three models is not fully controlled, a similar F1 score among them is desirable.

	precision	recall	F1	FLOPs	param num	rtf_cpu	rtf_gpu
ResNet-18	0.77	0.43	0.55	6.3G	11.7M	0.00315	0.000647
MobileNetV2	0.76	0.443	0.56	1.22G	2.23M	0.026296	0.001339
EfficientNet	0.758	0.44	0.557	1.57G	4.01M	0.03346	0.00222

Table 4.10: Evaluation among ResNet-18, MobileNetV2 and EfficientNet

However, our findings reveal a discrepancy in the performance of the MobileNetV2 and EfficientNet models in terms of speed, which contradicts our prior knowledge. The MobileNetV2 and EfficientNet have an extremely large real-time factor in both GPU and CPU compared with ResNet-18, while the FLOPs of MobileNetV2 and EfficientNet are much smaller than ResNet-18. We used to suppose that the higher the FLOPs, the more calculations a model need to take during inference, which leads to a longer inference time. But our result shows a contrary result. I find the answer to this unexpected situation in the paper written by Ma et al. [2018].

Ma et al. [2018] announce that the FLOPs, which are an indirect metric of speed, is only an approximation of the direct metric that we truly care about, i.e., the speed of a model. One of the most important factors that affect the speed is memory access cost (MAC), which is not included in FLOPs. Ma et al. [2018] analyze the run time of MobileNetV2 and claim that the FLOPs is not an accurate metric of speed when evaluating MobileNetV2 since there are many other factors influencing the speed. Figure 4.14 shows the run time decomposition of MobileNetV2 on GPU, where the FLOPs only calculate the convolution part of this model(the orange region). Consequently, Ma et al. [2018] come up with several practical guidelines for efficient network architecture design, and we can see that MobileNet disobey the following rules:

- Equal channel width minimizes memory access cost (MAC): The pointwise convolution in the inverted residual block of MobileNetV2 accounts for most of the complexity. Since the pointwise convolution is used to expand and shrink the channel (as we can see in Figure 4.12), the difference in channel width is very large.

- Element-wise operations are non-negligible: The depthwise convolution is considered as an element-wise operation which is frequently used in MobileNetV2.

Since both MobileNetV2 and EfficientNet include depthwise and pointwise convolution, the FLOPs failed to describe the speed of these two CNNs. And this is the reason why ResNet-18 has a smaller real-time factor than MobileNetV2 and EfficientNet.

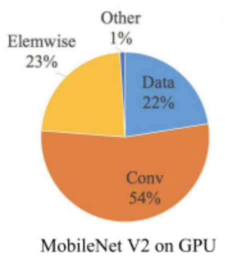


Figure 4.14: Run time decomposition of MobileNetV2 on GPU[Ma et al., 2018]

The following graph(Figure 4.15) provides a visualised version of the CPU real-time factor vs F1 score for MobileNetV2 and EfficientNet. We also include the baseline, standard ResNet-18 and ResNet-50 (which is the largest model we trained in the previous experiments) as comparisons. The size of a dot in this graph represents FLOPs, which is the number of calculations of a model. We can see that the size of dots does not increase when the CPU real-time factor becomes larger. Remember that the CPU real-time factor is calculated as:

$$RTF = \frac{time\ to\ process\ a\ segment}{duration\ of\ the\ segment}$$

, which is a direct metric of the speed of a model. In general, we can see that the F1 of EfficientNet and MobileNetV2 is higher than ResNet-18 and baseline, but is smaller than ResNet-50. However, despite having smaller FLOPs, the inference speed of these two CNN models is significantly slower than all of the ResNet models used in previous experiments.

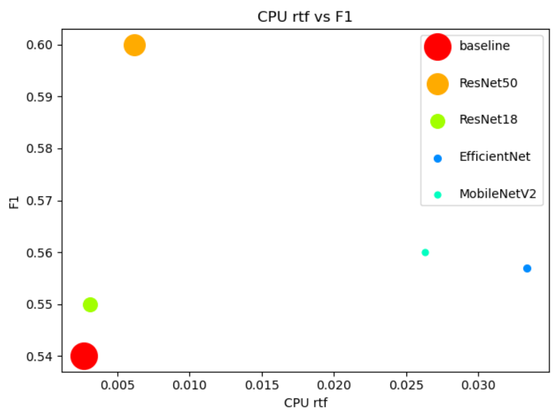


Figure 4.15: CPU real-time factor vs F1 score for MobileNetV2, EfficientNet, baseline, ResNet-18 and ResNet-50. The size of dots represent the value of FLOPs for each model.

Chapter 5

Conclusion and Future Work

In this project, we focus on constructing an efficient laughter detector with convolution neural networks with the aim of improving upon the baseline model presented in a previous study [Wolter, 2022a]. Our goal is to enhance the accuracy and speed of the laughter detector. To achieve this goal, we experiment with various CNN architectures and sizes, including ResNet, MobileNetV2, and EfficientNet, and explore the impact of altering their depths and widths on the detector's performance.

Since there are some issues in Wolter [2022a]'s work which influence the performance of the baseline, we start with solving these issues. There are three issues related to the data preprocessing pipeline and one related to the training process. The first issue is caused by the update of LhotseŻelasko [2021], which is a Python library that Wolter used to improve the efficiency of audio data preprocessing. This library is still under development, so some breaking changes appear after Wolter's project and these changes make Wolter's preprocessing pipeline crash down. The second issue in preprocessing step is the laughter in non-transcribed regions, which means some samples that are labelled as 'silence' contains sounds such as laughter. The third issue is the mismatch in distribution between training and test data, where the distribution represents the proportion of laughter and non-laughter samples. The models overfit the distribution of training data and do not perform well on the test set. The last issue appears in the training process, which is about manually setting the hyper-parameter: epoch. Wolter set the epoch number manually, while a better way is to let the training process decide what is the best epoch number to stop. Manually setting epoch numbers is time-consuming and difficult to find a suitable epoch number. In this thesis, we evaluate and solve all the issues except the mismatch of distribution, and make the performance of the baseline become better.

Then we do some experiments to explore different architectures. We mainly explore three kinds of CNN architecture: ResNet, MobileNetV2 and EfficientNet. The ResNet is chosen because it is a classic CNN architecture and is used in the baseline by Wolter [2022a]. We choose MobileNetV2 and EfficientNet because they are state-of-art lightweight and fast CNN. We further explore how the change in sizes (depth and width) of ResNet architecture will influence performance. And construct a series of models which apply the technique of depthwise separable convolution on ResNet.

Our results indicate that ResNet models with both smaller and larger sizes than the original ResNet-18 outperform the baseline model in terms of accuracy. And ResNet with smaller size improves the speed of inference. Specifically, ResNet-18-narrow, with a smaller width than the standard ResNet-18, yields the highest F1 score (0.59) and the fastest real-time factor on the CPU (0.001063).

Experiments on MobileNetV2 and EfficientNet show that the widely used metric: FLOPs failed to indicate the real speed of a model. Even though these two models successfully decrease the FLOPs, there are some other factors that also influence the speed. For example, the memory access cost. The paper from Ma et al. [2018] proposes some rules on CNN architectures for achieving faster speed, and MobileNetV2 and EfficientNet disobey two of them.

Future work may include exploring the performance on ShuffleNetMa et al. [2018], which is a state-of-art in terms of speed and accuracy trade-off and consider not only FLOPs as the metric of speed. Some other ways of speeding up a CNN can be taken into account, such as knowledge distillation and model quantization. Also, modifying the training process to make it match the distribution of the test set is also an adaptable future work. Since the F1 on the validation set(which has the same laughter distribution as the training set) is very high, we hypothesise that a model can not learn more from the training set and the low performance on the test set is caused by the lack of ability to generalization on different data distribution. The more detailed further work is introduced in the next section, which introduces the plan for the second part of the project.

5.1 Plan for MInf Project (Part 2)

The plan for part 2 of the MInf project is to continue the research on the efficiency laughter detector, and some work on the applause detector could also be taken into account. Further research on the laughter detector may include the following points:

- Keeps the distribution of the training set and the test set to be the same.
- Fine-tune the ResNets with different depths and widths and retrain them.
- Exploring the performance on ShuffleNet(Ma et al. [2018]) architecture.
- Exploring more ways to speed up the inference, such as knowledge distillation and model quantization.

The first point is mentioned in Chapter 2, in which we observe that the F1 on the validation set(which has the same laughter distribution as the training set) is very high, but the F1 on the test set is much lower. We hypothesise that a model overfits the distribution of the training set and the low performance on the test set is caused by the lack of ability to generalization on different data distributions.

We want to fine-tune the ResNets with different depths and widths because Figure 4.6 in Chapter 4 shows a trend that contrasts common sense. We think that this is caused by improper setting of some hyper-parameters, such as learning rate, which can be solved by fine-tuning the model.

In the third point, we want to explore the ShuffleNet because it is a lightweight and fast CNN architecture. And unlike MobileNetV2 and EfficientNet, which use FLOPs as an indicator of speed, Ma et al. [2018] cares about other factors that will influence the real-time inference speed of a model. And in the fourth point, more ways to speed up the inference can be tried. Knowledge distillation and model quantization are two ways which are introduced in Chapter 2. In simple words, knowledge distillation means letting a huge model learn from the training set, and then applying the knowledge the huge model learns to a much smaller model. Model quantization converts the weights in a model from floats to integers, which can speed up the calculation since the CPU calculate integers faster than floats.

Bibliography

Which model is best: Comparing methods and metrics for automatic laughter detection in a naturalistic conversational dataset

Jo-Anne Bachorowski, Moria J Smoski, and Michael J Owren. The acoustic features of human laughter. *The journal of the Acoustical Society of America*, 110(3):1581–1597, 2001.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Christopher Cieri, David Miller, and Kevin Walker. The fisher corpus: a resource for the next generations of speech-to-text. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal, May 2004. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2004/pdf/767.pdf`.

MLP cluster. Mlp cluster. URL `https://computing.help.inf.ed.ac.uk/teaching-cluster`.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

Daniel Falbel. *torchvision: Models, Datasets and Transformations for Images*, 2022. https://torchvision.mlverse.org, https://github.com/mlverse/torchvision.

Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 776–780. IEEE, 2017.

Jon Gillick, Wesley Deng, Kimiko Ryokai, and David Bamman. Robust laughter detection in noisy environments. In *Interspeech*, pages 2481–2485, 2021.

John J Godfrey, Edward C Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 1, pages 517–520. IEEE Computer Society, 1992.

Rahul Gupta, Kartik Audhkhasi, Sungbok Lee, and Shrikanth S Narayanan. Paralinguistic event detection from speech using probabilistic time-series smoothing and masking. In *Interspeech*, pages 173–177, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Hynek Hermansky. Perceptual linear predictive (plp) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

Adam Janin, Don Baron, Jane Edwards, Dan Ellis, David Gelbart, Nelson Morgan, Barbara Peskin, Thilo Pfau, Elizabeth Shriberg, Andreas Stolcke, et al. The icsi meeting corpus. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 1, pages I–I. IEEE, 2003.

Vincent M. Stanford Elham Tabassi Jonathan G. Fiscus Christophe D. Laprun Nicolas Pratz Jerome Lard John S. Garofolo, Martial Michel. Nist meeting pilot corpus speech. doi: https://doi.org/10.35111/800p-fv08.

Lakshmish Kaushik, Abhijeet Sangwan, and John HL Hansen. Laughter and filler detection in naturalistic audio. 2015.

Lyndon S Kennedy and Daniel PW Ellis. Laughter detection in meetings. 2004.

Mary Tai Knox and Nikki Mirghafori. Automatic laughter detection using neural networks. In *Interspeech*, pages 2973–2976, 2007.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

Lyken. Thop:pytorch-opcounter, 2022. URL `https://github.com/Lyken17/pytorch-OpCounter`.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

Franck Mamalet and Christophe Garcia. Simplifying convnets for fast learning. In *Artificial Neural Networks and Machine Learning–ICANN 2012: 22nd International Conference on Artificial Neural Networks, Lausanne, Switzerland, September 11-14, 2012, Proceedings, Part II 22*, pages 58–65. Springer, 2012.

Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. Rethinking cnn models for audio classification. *arXiv preprint arXiv:2007.11154*, 2020.

Huy Phan, Philipp Koch, Fabrice Katzberg, Marco Maass, Radoslaw Mazur, Ian McLoughlin, and Alfred Mertins. What makes audio event detection harder than classification? In *2017 25th European signal processing conference (EUSIPCO)*, pages 2739–2743. IEEE, 2017.

Steven J Phillips, Miroslav Dudík, and Robert E Schapire. A maximum entropy approach to species distribution modeling. In *Proceedings of the twenty-first international conference on Machine learning*, page 83, 2004.

Anna Polychroniou, Hugues Salamin, and Alessandro Vinciarelli. The sspnet-mobile corpus: Social signal processing over mobile phones. In *LREC*, pages 1492–1498, 2014.

Robert R Provine. Laughter. *American scientist*, 84(1):38–45, 1996.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

Steve Renals and Pawel Swietojanski. Neural networks for distant speech recognition. In *2014 4th joint workshop on hands-free speech communication and microphone arrays (HSCMA)*, pages 172–176. IEEE, 2014.

Kimiko Ryokai, Elena Duran Lopez, Noura Howell, Jon Gillick, and David Bamman.

Capturing, representing, and interacting with laughter. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

Jürgen Trouvain. Segmenting phonetic units in laughter. In *Proc. 15th International Conference of the Phonetic Sciences, Barcelona, Spain*, pages 2793–2796. Citeseer, 2003.

Khiet P Truong and David A van Leeuwen. Automatic detection of laughter. In *Ninth European Conference on Speech Communication and Technology*, 2005.

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.

Lasse Wolter. Honours project 2021/22 - a machine learning model for laughter detection. 2022a.

Lasse Wolter. A machine learning pipeline for laughter detection on the icsi corpus. 2022b.

Trmal Khudanpur Żelasko, Povey. a speech data representation library for the modern deep learning ecosystem, 2021. URL `https://github.com/lhotse-speech/lhotse`.

Appendix A

Appendix

GitHub repository of this project: https://github.com/linda-XI/laughter-detection/tree/main