A Study on Real-Time Automatic Speech Recognition System on Edge Devices
エッジ端末におけるリアルタイム音声認識システムに関する研究

山梨大学大学院
医工農学総合教育部

博士課程学位論文

修了年月 2024年3月

氏名　　王 宇

2023 年度 山梨大学大学院医工農学総合教育部工学専攻
博士論文公聴会及び最終審査にて発表済み
公聴会開催日：2024年2月2日
開催場所： 山梨大学大学院医工農学総合教育部工学専攻内
主催： 山梨大学

SUMMARY OF DISSERTATION

TITLE A Study on Real-Time Automatic Speech Recognition System on Edge Devices

NAME Yu Wang

Nowadays, automatic speech recognition (ASR) technology has been applied to many daily tasks, such as automatic subtitling on video websites and human-computer interaction in in-vehicle infotainment systems.

ASR technology has long been an important field within artificial intelligence. Early ASR systems used the grammatical rules of human natural language to convert speech to text. With a large amount of speech and text data being open-sourced, statistics-based ASR approaches have gradually become mainstream. Traditional statistical ASR systems first refine the speech signal into acoustic features, then use Gaussian mixture models and hidden Markov models to model the distributional and transfer probabilities of acoustic features, and use N-Gram language models to obtain the contextual probability of human natural language. Finally, these three probabilities will be combined to obtain the final recognition result.

Since the 21st century, thanks to advancements in computer hardware and the availability of open-source data, deep neural network (DNN) models have begun to excel in the fields of computer vision and natural language processing. In ASR, DNN models have gradually replaced Gaussian mixture models, hidden Markov models, and N-Gram language models as the primary models in ASR systems. ASR systems based on deep learning approaches have received considerable attention.

ASR systems are typically deployed either on a cloud server or an edge device. A mainstream ASR system comprises two primary components: a DNN model and an ASR decoder. DNN models require substantial computational resources during the forward stage, while ASR decoders generally consume significant memory. Consequently, ASR systems are often deployed on cloud servers, benefiting from advanced computational chips and ample, cost-effective memory capacity. On the cloud side, the main emphasis of ASR systems is on identifying DNN models with enhanced accuracy. Nevertheless, considering the occasional unavailability of communication networks and the need to protect users' private information from being uploaded to server databases, deploying ASR systems on edge devices becomes essential. Because this necessitates robust hardware computational power and memory capacity to efficiently run the ASR system, this is particularly challenging when deploying ASR systems on low-end devices. Therefore, the research on cloud-side ASR systems primarily focuses on developing model structures that are optimized for various devices.

Taking into account the advantages of both the device-side and cloud-side ASR systems, the workflow of a complete, deployable ASR system, which is currently popular, unfolds as follows:

1. Capture the speech signal with a microphone on the device side.

2. For more complex tasks, such as speech conversation and online navigation, the speech signal is uploaded via the communication network to a remote server. Subsequently, the cloud ASR system processes the signal to obtain recognition results, which are then relayed back to the device.

3. For simpler tasks, such as activation via a wake-up word and speech commands, or when network communication is unavailable, or user data is not allowed to be uploaded, the recognition result is obtained directly using the device-side ASR system.

This study explores optimization approaches for both cloud-side and device-side ASR systems, respectively.

For cloud-side ASR system, I propose a toolkit for developing real-time ASR systems in a cloud environment. As previously mentioned, an ASR system comprises a DNN model and an ASR decoder. Kaldi, one of the most popular ASR toolkits, offers integrated functions for building DNN models and decoders. However, since Kaldi is developed in C++, it poses challenges in debugging model structures and training strategies. Conversely, the flexibility of the Python language has led to its widespread use in deep learning frameworks like PyTorch and TensorFlow, which have fostered a proliferation of advanced neural network models. Consequently, ASR researchers are keen on finding convenient methods to integrate DNN models trained with Python-based deep learning frameworks into decoders built using Kaldi's C++ framework.

Several tools discussed in related literature offer user-friendly interfaces that facilitate the connection between DNN models and Kaldi-built decoders, enabling the creation of offline ASR systems. However, there is a scarcity of publicly available tools that simplify the development of real-time ASR systems.

I propose a comprehensive toolkit that encompasses all essential functions required to establish a complete real-time ASR pipeline. This includes recording speech signals, extracting acoustic features, transmitting over networks, processing with DNN models, and decoding. The toolkit is designed for ease of integration, allowing developers to seamlessly incorporate their custom models and decoding graphs. Additionally, it supports feature mixing and the integration of denoising models.

In my experiments, I have demonstrated that this toolkit enables the construction of highly accurate real-time ASR systems. By amalgamating multiple acoustic features and incorporating a denoising model, I have achieved significant enhancements in both the accuracy and robustness of the ASR system.

For the device-side ASR system, I propose a lightweight ASR system, which includes a novel DNN model structure and an enhanced decoding algorithm. Recently, as edge devices increasingly integrate AI applications, manufacturers have been providing their own inference frameworks for DNN models to maximize chip computational power. However,

many advanced and high-precision DNN model structures are incompatible with these frameworks. Additionally, ASR decoders typically used in cloud-based systems consume substantial computational memory, posing challenges for deployment on edge devices.

While some related works have introduced lightweight DNN model structures for device-side usage, they often overlook the compatibility with inference frameworks of lower-end edge devices, rendering them unsuitable. Furthermore, there has been limited focus on optimizing decoding algorithms in prior research.

To address these gaps, my approach encompasses several innovations. Firstly, I introduce a model structure exclusively based on convolutional neural networks, ensuring compatibility with most edge device software development kits. I also streamline the process by directly utilizing speech signals, thereby reducing CPU usage. Lastly, I have refined the prefix beam search decoding algorithm by implementing a unique pruning method using a lexicon trie and introducing a novel language model that leverages initial letters. These enhancements collectively enable the construction of a high-accuracy ASR decoder that requires less computational memory.

My experimental results demonstrate that this ASR system excels in model size, recognition accuracy, and ease of deployment, characterized by low CPU usage and a high real-time processing rate.

To summarize, my research on ASR systems for edge devices introduces new solutions for both cloud and device-side systems. On the cloud side, I have developed a toolkit to construct a real-time ASR pipeline. For the device side, I have designed a novel model structure and an ASR decoder. These advancements contribute to creating an ASR system that is lightweight, easily deployable, highly accurate, and possesses a rapid real-time processing rate. While this study has achieved significant progress, there is still potential for fine-tuning the DNN model's structure on the device side. In future work, I aim to further investigate deploying more complex models on low-end devices.

This thesis is organized as follows. In Chapter 1, I discuss the current challenges in ASR technology, review related research, and provide an outline of this study. In Chapter 2, I outline the principles of deep learning, examine the current state-of-the-art, explore classical models, and delve into some emerging technologies. In Chapter 3, I introduce deep learning-based ASR algorithms. In Chapter 4, I showcase real-time ASR technologies and propose development tools for constructing a cloud-side speech recognition system. In Chapter 5, I examine the current advancements in device-side ASR technologies and propose a deployable, lightweight ASR system, which includes a novel DNN model and an enhanced decoding algorithm. In Chapter 6, I introduce my other exploratory work: the voice activity detection. I will discuss the relevance of this work to ASR system and present some achievements in building a lightweight voice activity detection model with good environmental robustness. In Chapter 7, I summarize the findings of this study and discuss directions for future research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Nowadays, automatic speech recognition (ASR) technology is applied in many daily tasks, such as automatically subtitling videos on websites, processing remote video conferences offline, controlling smart domestic appliances in real-time using voice commands, and facilitating human-computer interaction in in-vehicle infotainment systems.

ASR technology has long been a significant field within artificial intelligence (AI). Early ASR systems utilized the grammatical rules of human natural language to convert speech to text. With the increasing availability of large amounts of speech and text data, statistics-based ASR approaches have become mainstream. Traditional statistical ASR systems initially refine the speech signal into high-dimensional acoustic features. These acoustic features contain valuable information for speech-to-text transcription. Gaussian mixture models are then used to fit the distributional probabilities of these acoustic features, hidden Markov models to fit the transition probabilities between contiguous frames, and N-Gram language models to determine the contextual probability of human natural language. Ultimately, these three probabilities are combined, and the text with the highest combined probability from all candidate word sequences is selected as the final recognition result.

Since the 21st century, with advancements in computing power and memory capacity of computer hardware, as well as the availability of large amounts of open-source data, deep neural network (DNN) [1] models have shown a strong ability to fit various data distributions in fields such as computer vision (CV) and natural language processing (NLP). In ASR, DNN models have gradually replaced Gaussian mixture models [2], hidden Markov models [3–6], and N-Gram language models [7, 8] as the backbone of ASR systems. ASR systems based on deep learning approaches have garnered considerable attention. In terms of data availability, an increasing number of high-quality speech corpora are being made public [9–11], collected for various tasks like multilingual speech-to-text transcription, text-to-speech synthesis, voiceprint recognition, noise suppression, and others.

After construction, an ASR system must be deployed to a cloud server, a host computer, or an edge device. Generally, an ASR system comprises two main components: a DNN model and an ASR decoder. The DNN model, a computing graph organized with weight matrices and operators, requires significant computation in the forward stage. Models with higher precision typically have more complex computing graph architectures. Therefore, DNN models are often deployed on cloud servers equipped with advanced computational chips to support such complex graphs and ensure good real-time performance. Besides the DNN model, ASR decoders typically require substantial memory, which cloud servers can more readily provide. As a result, cloud-based ASR systems focus on developing better DNN structures and ASR decoders to enhance accuracy and tackle more challenging tasks. Nevertheless, considering that communication networks can be unreliable and users' private information may need to remain local, deploying ASR systems on edge devices is also necessary. However, as mentioned, running an ASR system demands significant hardware computing power and memory capacity, posing a substantial challenge for device-side ASR technology, especially when installed on low-end devices. Thus, device-side speech recognition systems are more focused on developing lightweight DNN models and ASR decoders, while maintaining competitive accuracy and robustness for specific tasks.

Taking into account the advantages of both the device-side and cloud-side ASR systems, as shown in Fig.1.1, the workflow of a complete, deployable ASR system, which is currently popular, unfolds as follows:

1. Capture the speech signal with a microphone on the device side.

2. For more complex tasks, such as speech conversation and online navigation, the speech signal is uploaded via the communication network to a remote server. Subsequently, the cloud ASR system processes the signal to obtain recognition results, which are then relayed back to the device.

3. For simpler tasks, such as activation via a wake-up word and speech commands, or when network communication is unavailable, or user data is not allowed to be uploaded, the recognition result is obtained directly using the device-side ASR system.

This study explores optimization approaches for both cloud-side and device-side ASR systems, respectively.

1.2 The Motivation of This Study

As mentioned earlier, my research will focus on two main areas: cloud-side ASR systems and device-side ASR systems.

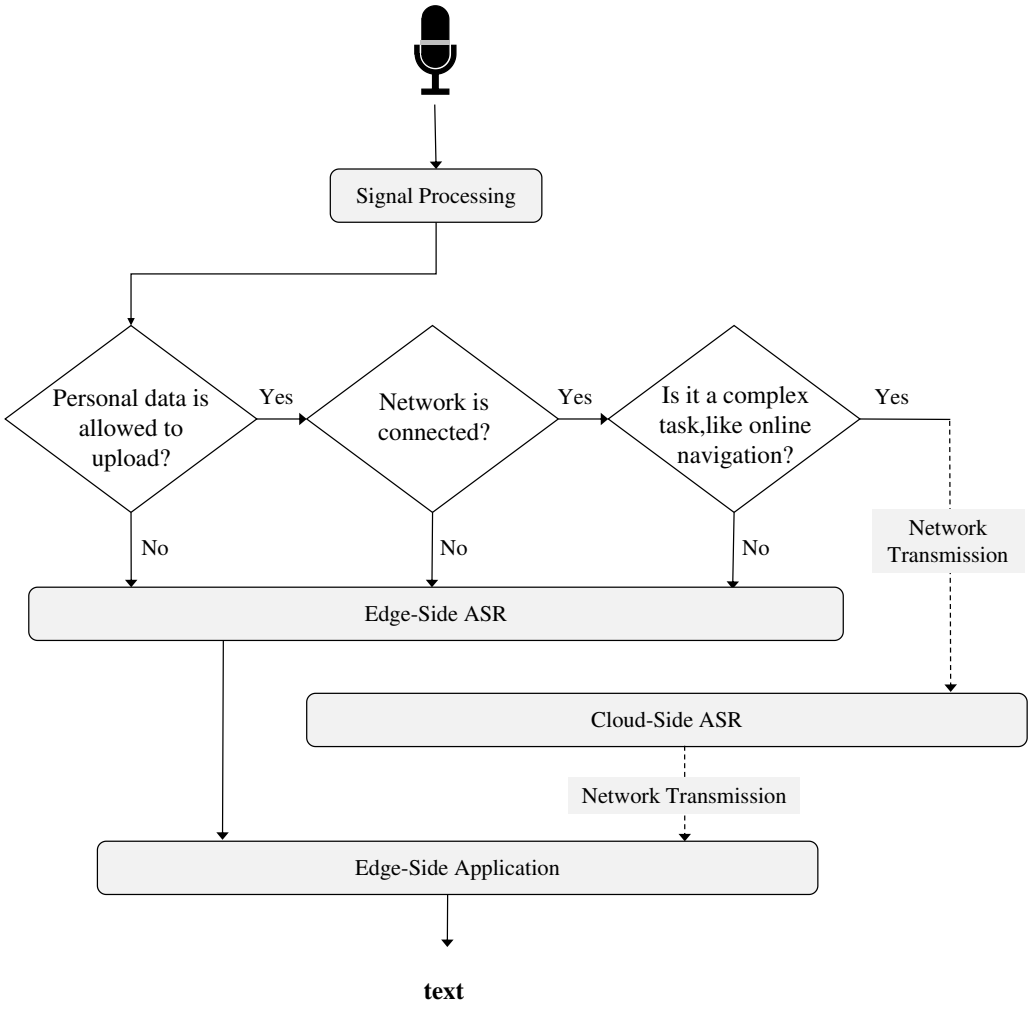


Figure 1.1: Collaborative workflow between cloud-side ASR and edge-side ASR at the task level

1.2.1 Cloud-Side Speech Recognition System

An ASR system comprises a DNN model and a decoder, with its construction relying on various development toolkits. Among them, Kaldi [12] speech recognition toolkit stands out as a popular choice. It offers comprehensive functions for DNN model training and decoder construction. For cloud-side ASR systems, the primary objective is to identify DNN models with enhanced accuracy. However, Kaldi's development in C++ restricts the flexibility in debugging both the model structure and the training process. Consequently, DNN models available in Kaldi tend to be relatively basic. In contrast, the flexibility of Python has led to the widespread adoption of Python-based deep learning frameworks like PyTorch [13] and TensorFlow [14]. These frameworks have catalyzed the development of advanced neural network models. Thus, ASR researchers are keen to find efficient methods

to integrate Python-trained DNN models with C++-based Kaldi decoders.

Several tools from related work [15–18] provide intuitive interfaces for connecting DNN models with Kaldi decoders, facilitating the creation of offline ASR systems. These tools encapsulate Kaldi in Python and offer Python interfaces for DNN model integration. However, they primarily target offline ASR system development. Real-time speech recognition, a necessity for most ASR systems, still lacks readily accessible tools for constructing real-time systems based on DNN models and Kaldi's decoder.

Therefore, this study aims to propose a new toolkit to assist researchers in developing real-time speech recognition pipelines suitable for cloud-side deployment.

1.2.2 Device-Side Speech Recognition System

On the device side, edge devices are typically limited in computational power and memory. Hence, the device-side ASR system focus is on devising models suitable for these constraints and compressing decoder sizes. Moreover, many edge device manufacturers offer their software development kits (SDKs). Recently, as AI applications have become more common in edge devices, these manufacturers have provided custom inference frameworks to optimize chip performance. Yet, many advanced, high-precision DNN models are not compatible with these frameworks. Additionally, cloud-side ASR decoders often require substantial computational memory. For instance, a decoder in my experiments in Chapter 5, built using just 5 hours of speech data, reached a size of 530 MB. These limitations in model structure and decoder size hinder the deployment of traditional ASR systems on edge devices.

While some studies have suggested lightweight DNN models for device-side use [19–21], they typically overlook the inference frameworks of lower-end devices, rendering them unsuitable. Furthermore, few have concentrated on optimizing decoding algorithms.

Therefore, this study aims to develop a DNN model structure that is not only compatible with a broader range of devices but is also lightweight to minimize memory usage. This approach is expected to facilitate easier deployment of ASR systems on device-side, particularly on low-end devices.

1.3 An Outline of This Study

For the cloud-side speech recognition system, I propose a toolkit named "ExKaldi-RT." This toolkit is a Python wrapper for the Kaldi toolkit and provides all necessary functions to build a complete, real-time ASR pipeline. These functions include recording speech signals, extracting acoustic features, transmitting data over the network, forwarding DNN models, and decoding. We have effectively encapsulated and decoupled these modules, ensuring that users can flexibly adjust real-time pipeline functionality by inserting or removing specific modules. Additionally, the toolkit supports the direct insertion of deep

learning frameworks-trained DNN models and Kaldi decoders. Furthermore, it enables feature mixing and the application of denoising models to enhance the ASR system's accuracy and robustness in noisy environments.

In our experiments, I demonstrated that using this toolkit enables the construction of a high-accuracy, real-time ASR system. By combining multiple acoustic features and integrating a denoising model, I further improved the ASR system's accuracy and robustness.

For the device-side speech recognition system, I propose a lightweight ASR system, including a novel DNN model structure and an improved decoding algorithm. I employed several approaches to address the shortcomings of related work. First, I propose a model structure comprised solely of convolutional neural networks, ensuring compatibility with the SDKs of most edge devices. Moreover, I streamlined the process by omitting the extraction of acoustic features and directly using speech signals, which reduces CPU usage. Finally, I optimized the decoding algorithm of prefix beam search by introducing a novel pruning method using a lexicon trie and a new language model based on initial letters. These optimizations enable the construction of a high-accuracy ASR decoder with reduced computational memory requirements.

In our experiments, our ASR system demonstrated excellent performance in terms of model size, recognition accuracy, easy deployment, low CPU usage, and a high real-time rate.

To summarize, my research on ASR systems for edge devices introduces new solutions for both cloud and device-side systems. On the cloud side, I have developed a toolkit to construct a real-time ASR pipeline. For the device side, I have designed a novel model structure and an ASR decoder. These advancements contribute to creating an ASR system that is lightweight, easily deployable, highly accurate, and possesses a rapid real-time processing rate. While this study has achieved significant progress, there is still potential for fine-tuning the DNN model's structure on the device side. In future work, I aim to further investigate deploying more complex models on low-end devices.

1.4 Organization of This Thesis

The remainder of this thesis is organized as follows.

In Chapter 2, I outline the principles of deep learning, examine the current state-of-the-art, explore classical models, and delve into some emerging technologies.

In Chapter 3, I introduce deep learning-based ASR algorithms.

In Chapter 4, I showcase real-time ASR technologies and propose development tools for constructing a cloud-based speech recognition system.

In Chapter 5, I examine the current advancements in device-side ASR technologies and propose a deployable, lightweight ASR system, which includes a novel DNN model and an enhanced decoding algorithm.

In Chapter 6, I introduce my other exploratory work: the voice activity detection. I will discuss the relevance of this work to ASR system and present some achievements in building a lightweight voice activity detection model with good environmental robustness.

In Chapter 7, I summarize the findings of this study and discuss directions for future research.

Chapter 2

Deep Learning

In this chapter, the principles of deep learning are introduced.

Initially, the current status of deep learning is discussed. This is followed by an introduction to the fundamental methods of deep learning, encompassing computational graphs and gradient descent. Subsequently, prevalent neural network models and their variations are outlined. The chapter concludes with a discussion on the development tools used for training and deploying neural network models, alongside key deep learning techniques pertinent to this study.

2.1 An Overview of Deep Learning

Deep learning, alternatively termed deep structured learning or hierarchical learning, constitutes a segment of machine learning methodologies. It represents a novel research direction that has garnered significant attention in recent years. The integration of deep learning into machine learning aims to align more closely with the field's foundational goal of achieving artificial intelligence (AI).

The deep learning concept emerged from research on artificial neural networks. An artificial neural network, simply referred to as a neural network, is a mathematical model designed to emulate the human brain. It features an internal structure resembling a computational graph of a perceptron, inclusive of multiple hidden layers. Neural networks synthesize increasingly abstract higher-level features by integrating lower-level features. This synthesis enables them to represent the attributes of input data, uncover underlying distributional patterns in these data, and make predictions about newly observed data.

The primary motivation behind deep learning is to develop neural network models with varied structures and capabilities, enabling them to mimic the human brain's data interpretation mechanisms. This applies to diverse data types, such as images, sounds, and texts. The goal is to empower machines to analyze and learn with human-like efficiency. A well-designed neural network model is capable of not only recognizing these data types but also interpreting and associating them meaningfully.

In recent years, the rapid development of high-speed computing chips, such as graphics processing units (GPUs), neural processing units (NPUs), and tensor processing units (TPUs), coupled with the widespread availability of big data across various fields, has significantly advanced deep learning technologies. These technologies enable machines to emulate human perception, cognition, and actions, effectively solving complex pattern recognition problems and advancing AI-related technologies. This period has seen the introduction of new model structures and training approaches, attracting more investment in exploiting deep learning's potential. To date, deep learning has demonstrated outstanding performance in data search, data mining, machine translation, speech recognition, recommendation systems, data generation, and other related fields, markedly surpassing most preceding technologies.

2.2 Computational Graph and Gradient Descent

The primary objective of deep learning is to construct a neural network model capable of performing classification or regression tasks.

$$y = f(x) \tag{2.1}$$

Eqn.2.1 illustrates the fundamental role of a neural network model, namely, its function as a mathematical entity. The symbol f denotes a neural network model, x represents the input value, and y signifies the output of the function (resulting from regression or classification). For computers, f constitutes a computational graph composed of a set of parameters and operators. Consider, for example, the execution of a simple linear equation, as shown in Eqn.2.2. This computational process can be organized in terms of data flow, as depicted in Fig.2.1. This process is referred to as a computational graph.

$$y = f(x) = k \times x + b \tag{2.2}$$

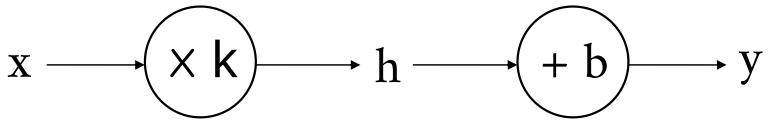


Figure 2.1: The computational graph of linear equation $y{=}k{\times}x{+}b$

Within this graph, there are two parameters: k (weight) and b (bias), along with two corresponding operators: $\times$ (multiplication) and $+$ (addition). When input data x is introduced into the graph, it flows along the designated path, interacting with each parameter, ultimately yielding the output y. The variable h denotes the intermediate

result. Although this represents a simplistic neural network model, the computational graphs utilized in research and industrial applications are typically far more complex. Distinct models feature varying computational graph structures, each comprising unique parameters and operators.

Like most mathematical models, determining the correct parameters for a neural network model involves processing input data. However, due to the high-dimensional nature of the data and the extensive number of operators in neural networks, explicitly solving these models is an insurmountable task. A viable alternative is the application of *gradient descent*. This method involves using input data x and known output $\hat{y}$ to calculate the computational error, subsequently minimizing this error by continuously adjusting the parameters until they approximate an optimal solution.

$$\theta_{i+1} = \theta_i - \eta \frac{\partial E(x, \hat{y}|\theta_i)}{\partial \theta_i} \tag{2.3}$$

Eqn.2.3 illustrates the gradient descent process. θ denotes the model's parameters. In the context of a large training dataset, we typically select a small subset of data to form a batch, which is then fed into the model. Since the model processes only a portion of the data at a time, multiple iterations are required to determine the optimal parameters. The variable i in Eqn.2.3 signifies the i-th iteration. $E(x, \hat{y}|\theta_i)$ indicates the error calculated using the training samples x, their true values $\hat{y}$, and the model parameters θ_i during the i-th iteration. The symbol η is the gradient scale, commonly referred to as the *learning rate*.

In the computational graph depicted in Fig.2.1, with known input data x, model output y, and the corresponding true value $\hat{y}$, I calculate the Euclidean distance between $\hat{y}$ and y. This distance represents the error between the prediction and ground truth, as detailed in Eqn.2.4.

$$E(x, \hat{y}|k, b) = \frac{1}{2}(y - \hat{y})^2 \tag{2.4}$$

In the forward process of this computational graph, the following results are already obtained.

$$h = x \times k, y = h + b \tag{2.5}$$

Then the gradients of parameters b and k can be computed by partial derivatives respectively.

$$f^{'}(b) = \frac{\partial E(x, \hat{y}|k, b)}{\partial b} = (y - \hat{y}) \frac{\partial y}{\partial b} = (y - \hat{y}) \tag{2.6}$$

$$f^{'}(k) = \frac{\partial E(x, \hat{y}|k, b)}{\partial k} = (y - \hat{y})\frac{\partial y}{\partial h}\frac{\partial h}{\partial k} = (y - \hat{y})x \tag{2.7}$$

After computing the gradients of k and b, these parameters can be updated using Eqn.2.3. In this procedure, I employ the Euclidean distance to quantify the error between the prediction and ground truth. This metric is also known as mean square error (MSE), a widely utilized error function in deep learning.

The emergence of gradient descent coincides with the Big Data era. Increasingly, annotated data (datasets with known inputs and corresponding outputs) are becoming publicly accessible, enabling the resolution of complex neural network model parameters. This resolution process is termed *training*, and the data employed for this purpose are referred to as *training data*.

2.3 Neural Network Models

2.3.1 Multilayer Perceptron

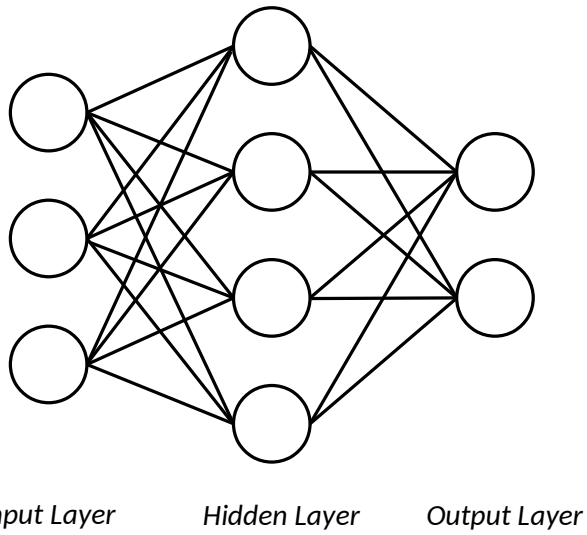


Figure 2.2: An example of multilayer perceptron model

The multilayer perceptron (MLP) [22] is a widely utilized neural network structure, comprising a stack of multiple fully connected layers, also known as *Linear* or *Feed-Forward* layers. Fig.2.2 illustrates an MLP network structure with three fully connected layers, consisting of an *input* layer, a *hidden* layer, and an *output* layer. Each neuron within a layer connects to every neuron in the subsequent layer. Beyond the *input* layer, both the *hidden* and *output* layers possess parameters: weight matrices and bias vectors. The model's input, denoted as x, is a column vector of dimension D. Reflecting the fully

connected nature, the *input* layer also contains D neuron nodes. The weight matrix for the *hidden* layer, W_h, is defined as a matrix with D rows and H columns, and the weight matrix for the *output* layer, W_o, as one with H rows and C columns. Currently, biases are not the focus. The model is mathematically represented as shown in Eqn.2.8.

$$f(x) = (x \cdot W_h) \cdot W_o \tag{2.8}$$

$\cdot$ stands for vector inner product operation. It can be seen that an MLP achieves nonlinear mapping through multiple fully connected layers. Additionally, as shown in Eqn.2.9, an activation function σ is typically applied at each layer to enhance the model's nonlinear representation.

$$f(x) = \sigma((\sigma(x \cdot W_h)) \cdot W_o) \tag{2.9}$$

The MLP accepts a column vector (a data sample) as input. As previously mentioned, to smooth the gradient descent process and enhance computational efficiency, it is common to input a batch of samples simultaneously. Consequently, the shape of the input x transforms into a matrix with N rows and D columns, where N represents the batch size of feature samples. The model's output signifies the prediction results for these N samples. When the layers of the neural network model are increased in depth, the model is referred to as a deep neural network (DNN) model. Training a DNN model presents significant challenges, which I will discuss, including the reasons for these difficulties and some proposed solutions, in the following section.

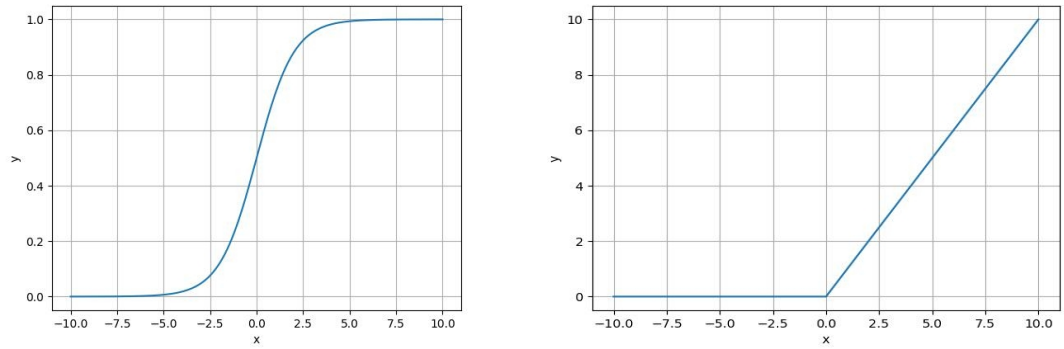
2.3.2 Activation Functions

The activation function plays a pivotal role in neural network modeling. It bolsters the network's ability to represent non-linearities, enabling it to more effectively fit diverse data distributions and enhance predictive accuracy.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

Fig.2.3 depicts two prevalent activation functions: (a) Sigmoid [23] and (b) ReLU[1] [24]. The Sigmoid function, as defined in Eqn.2.10, confines its output to the range between 0 and 1. This output is frequently utilized as a probability, a normalized scale, or a weighting factor. Despite being continuous and differentiable, thus adept at non-linear fitting, Sigmoid has notable limitations. Specifically, when input values are significantly above or below 0, their respective outputs converge closely, impeding the model's ability

[1]Rectified Linear Unit



(a) (b)

Figure 2.3: Sigmoid and ReLU activation function

to distinguish between these inputs and subsequently slowing training. Moreover, deriving the Sigmoid function, as shown in Eqn.2.11, reveals another issue.

$$\frac{\partial Sigmoid(x)}{\partial x} = Sigmoid(x)(1 - Sigmoid(x)) \tag{2.11}$$

Given that $Sigmoid(x) \in (0, 1)$, its gradient also falls between 0 and 1. Consequently, in deeper neural network layers, the gradient diminishes as one approaches the input layer. This minimal gradient is insufficient for effective parameter updates, a problem known as *gradient vanishing*.

$$ReLU(x) = max(x, 0) \tag{2.12}$$

The ReLU activation function, formulated in Eqn.2.12, operates differently. It outputs 0 for any input x less than 0, while for $x \geq 0$, it remains inactive. ReLU's continuity and differentiability, coupled with a derivative of 1 for positive x, facilitate more efficient parameter updates in deep neural networks.

Another widely used activation function is Softmax, typically employed in the final layer of classification neural networks to normalize outputs across categories. These normalized values represent the probability of each category.

$$y_i = \frac{e^{\bar{y}_i}}{\sum_{k=1}^{K} e^{\bar{y}_k}}, i \in [1, K] \tag{2.13}$$

Eqn.2.13 details the activated output for each category, where K denotes the total number of categories and $\bar{y}$ symbolizes the model's initial output.

2.3.3 Normalization Technologies

In recent years, the complexity of tasks performed by neural networks has led to an increase in the number of hidden layers. For instance, the VGG[2] [25] model, prominent in the field of CV, incorporates up to 19 layers, while the ResNet[3] [26] model can have as many as 101 layers. However, increasing network depth often causes the output data distribution of each layer to converge towards the extremes of the activation function's output range (known as the saturation interval), leading to gradient vanishing. This phenomenon hinders further training of DNN models. To address this, several normalization techniques have been developed, including *Weight Initialization* [27], *Batch Normalization* [28], and *Dropout* [29].

Batch normalization represents a significant advancement in deep learning. This technique involves two sequential linear transformations that recalibrate the layer's output distribution to align with a standard normal distribution. Consequently, the output values are positioned within ranges where the activation function is highly responsive to input variations. This sensitivity ensures that minor changes in the input can induce substantial alterations in the loss function. As a result, it not only enlarges the gradient, preventing its disappearance, but also accelerates the model's convergence.

$$\mu_B = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.14}$$

$$\sigma_B^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_B)^2 \tag{2.15}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.16}$$

$$y_i = \gamma\hat{x}_i + \beta \tag{2.17}$$

Eqn.2.14 to Eqn.2.17 demonstrate the workings of batch normalization. The term N represents the quantity of data in a batch, referred to as the *batch size*. The offset value ϵ prevents the denominator from becoming zero. Upon receiving a batch of data $[x_1, x_2, \ldots, x_N]$, the mean μ_B and standard deviation σ_B of the batch are calculated. The input data distribution is then transformed into a standard normal distribution using μ_B and σ_B. This is followed by a linear transformation employing scale factor γ and bias β.

[2]Visual Geometry Group

[3]Residual Network

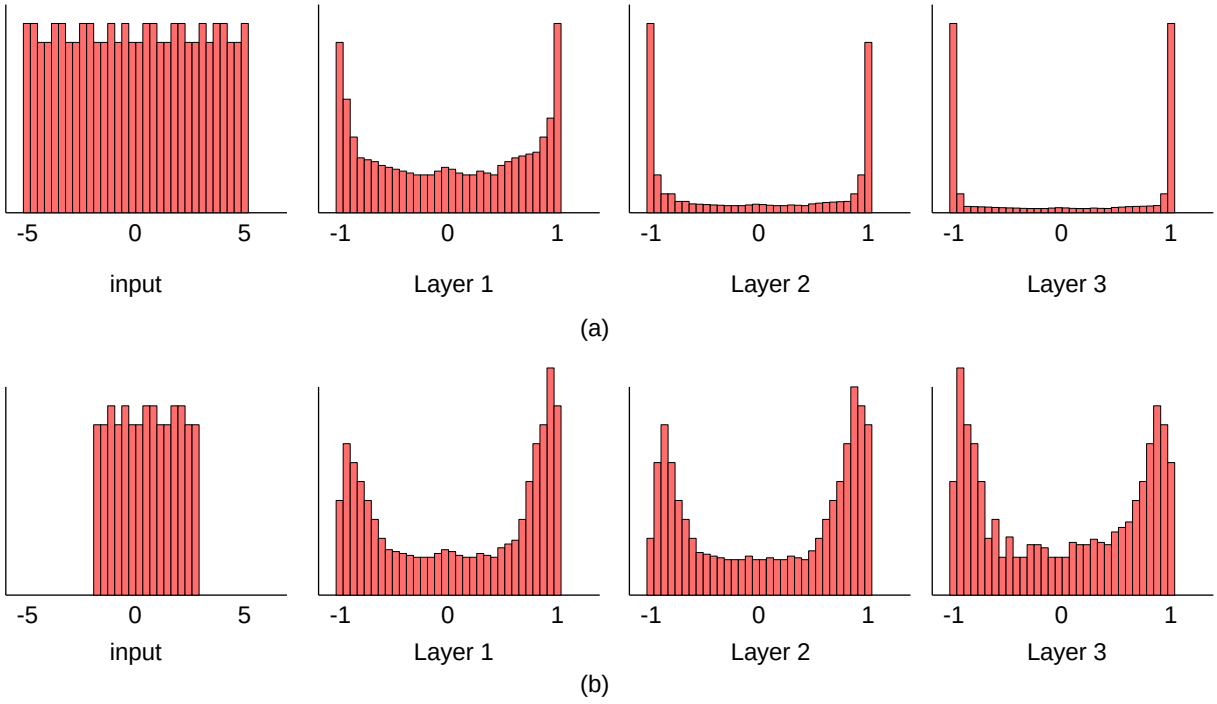


Figure 2.4: An example of output distributions using batch normalization

Fig.2.4 illustrates the output distributions at each layer both before (a) and after (b) the application of batch normalization. The output data, as seen in Fig.2.4(b), is uniformly distributed between -1 and 1.

Layer Normalization [30] is another widely-used normalization technique. While batch normalization standardizes features of the same dimension across different samples, thereby reducing the correlation between different dimensions within the same sample and accentuating the variations in the distribution of identical dimensions across different batch samples, layer normalization contrasts by standardizing the values across various dimensions within the same sample. This enhances the distributional disparities within individual samples but does not effectively establish connections between batch samples in the same feature dimension.

$$BatchNormalization : \mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j} \tag{2.18}$$

$$LayerNormalization : \mu_j = \frac{1}{D}\sum_{j=1}^{D} x_{i,j} \tag{2.19}$$

Suppose I have input data x consisting of a batch of samples. x is a matrix with N rows and D columns, where N represents the batch size and D represents the feature dimension

of a sample. For any $i \in [1, N]$ and $j \in [1, D]$, Eqn.2.18 and Eqn.2.19 demonstrate how batch normalization and layer normalization calculate their mean values, respectively.

Layer normalization offers several advantages over batch normalization:

1. The computation of layer normalization is independent of batch size, avoiding unreasonable mean and standard deviation values when the batch size is small, which could impair the effectiveness of normalization.

2. For features with temporal characteristics, such as consecutive words or speech, layer normalization is more effective at maintaining variability across different feature dimensions of each sample, aiding the model in distinguishing between samples.

Consequently, layer normalization is frequently utilized in context-dependent neural network models.

I will also discuss a classic normalization technique: *Dropout*. In deep learning tasks, the quantity of data is crucial. For complex tasks, designing a deep neural network model with many parameters is often necessary. However, a mismatch between data volume and model complexity can lead to problems:

1. Underfitting occurs when the model lacks sufficient parameters to learn the data distribution adequately.

2. Overfitting happens when a complex model is trained on limited data, leading to poor generalizability on unseen data.

To mitigate overfitting, dropout is introduced as a solution in deep learning.

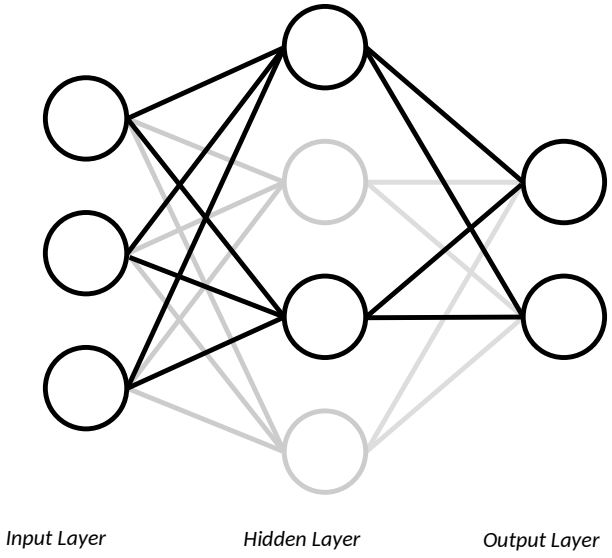


Figure 2.5: An example of dropout

Fig.2.5 illustrates an example of using dropout. Here, gray neurons indicate temporary non-functionality during computation. During training, a neuron's activation is probabilistically disabled during forward propagation by a probability p. This process enhances model generalizability by reducing reliance on specific local features and emphasizing global information. During inference, all neurons are active, and their outputs are adjusted according to the probability p. Dropout can be viewed as a form of model ensemble: each training iteration results in a different model structure due to the probability p. Essentially, it equates to training multiple simpler models simultaneously, each learning the data distribution. During prediction, the outputs of these models are aggregated, thereby improving the model's generalization capability.

2.3.4 Convolutional Neural Network and Its Variants

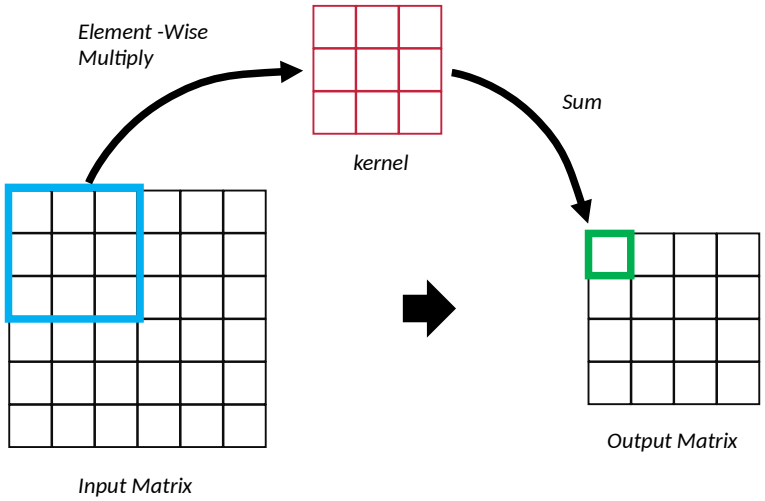


Figure 2.6: An example of convolutional neural network model

A MLP, also known as a fully connected network, includes an input layer that connects to all dimensions of the input vector. In the MNIST dataset [31], a classic entry-level dataset for deep learning, each sample is a 28×28 single-channel image featuring handwritten digits from 0 to 9. To construct an MLP model for digit recognition, one must first flatten the image into a 784-dimensional feature vector ($28 \times 28 = 784$). However, this approach is somewhat naive for most deep learning tasks. In CV, it is more common to directly input an image with larger dimensions, denoted as $W \times H \times C$, where W, H, and C represent the width, height, and number of color channels, respectively. For instance, the widely-used ImageNet dataset [32] typically features images of size 224×224 pixels with 3 color channels (RGB). Flattening such an image into a column vector for an MLP model would require an input layer with $224 \times 224 \times 3 = 150,528$ weight values, leading to a large and training-intensive model. To address this issue, the convolutional

neural network (CNN) model [33] is introduced.

As illustrated in Fig.2.6, the convolutional operation process in a CNN model is depicted. A convolutional layer consists of multiple $M \times N$ convolutional kernels, each sliding continuously over the image to perform point-to-point multiplication and summation, yielding an output matrix. CNNs significantly reduce the number of parameters required when processing large input images. Typically, through pooling operations, the matrix size is progressively reduced across layers, and the final CNN model output represents deep features extracted from the original image. These deep features, when trained in a supervised manner, encapsulate critical information for downstream tasks like image classification, object detection, and instance segmentation. CNNs are essential for managing large-scale image inputs and have become pivotal in CV and other fields where inputs can be interpreted as images, such as acoustic feature maps.

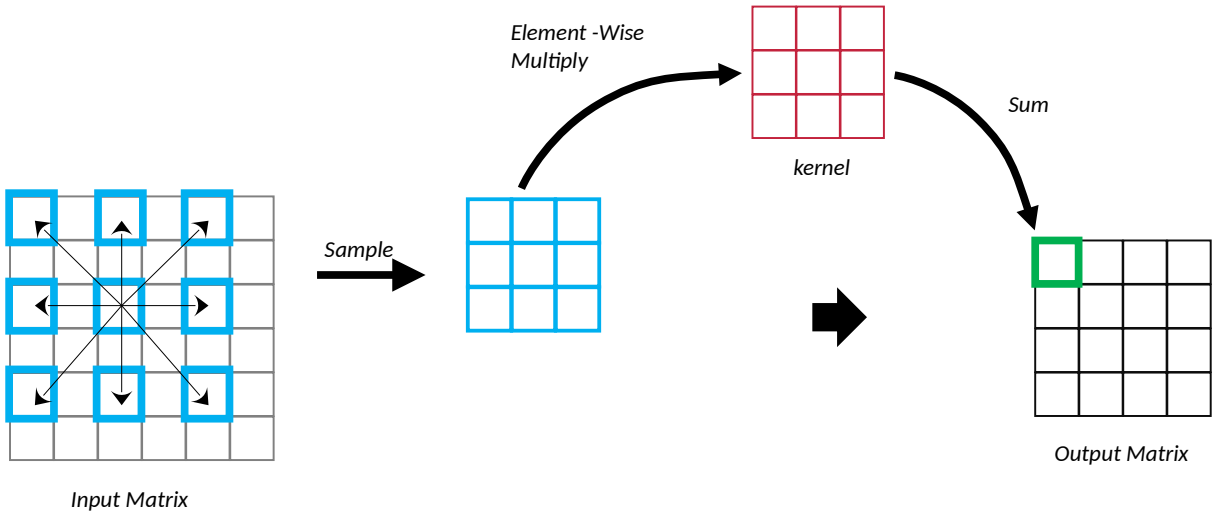


Figure 2.7: An example of dilated convolutional neural network model

Traditional CNNs can only focus on feature values within an $M \times N$ rectangular region at a time, a region I refer to as the *Receptive Field*. Increasing M and N allows the convolutional kernel to access more feature information but leads to an increase in the parameters of the neural network model. Conversely, decreasing M and N results in a smaller receptive field for the convolutional kernel, which is not conducive to capturing a wide range of correlations. One approach to address this is the *Dilated Convolution* [34] model. As shown in Fig.2.7, *Dilated Convolution* expands the receptive field and then samples sparsely and uniformly within it, rather than sampling densely from within the $M \times N$ rectangular region. This approach ensures that the convolution kernel obtains a larger receptive field without increasing the number of parameters.

However, both classic and dilated convolutions use convolution kernels of fixed rectangular shape. While this may work well for features with regular shapes, neither approach can automatically adapt to features with varying shapes. For features with rich and

complex deformations, two general approaches are used:

1. Increase the data volume or employ data augmentation to expose the convolution kernel to more varied-shaped features.

2. Manually design specific features or algorithmic modules, such as scale invariant feature transform (SIFT) [35].

These approaches, however, lack flexibility. Therefore, another CNN variant is proposed in prior research: the *Deformable Convolution* [36] model.

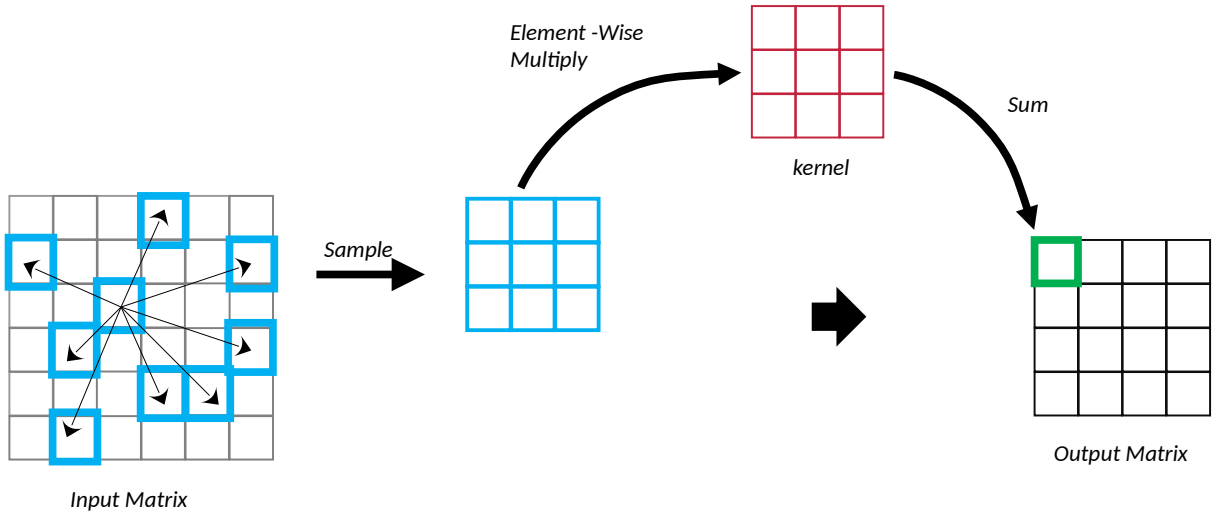


Figure 2.8: An example of deformable convolutional neural network model

Fig.2.8 shows an example of the deformable convolution model. This model predicts the offset of its sampling points from the current anchor point (the position of the convolution kernel's center point on the feature map) using an additional convolutional layer. For instance, in a traditional 3×3 convolution kernel, 9 coordinate points in the rectangular region are sampled, and the offset of these points relative to the anchor point is defined according to Eqn.2.20.

$$R = \{(-1,-1),(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)\} \tag{2.20}$$

For an anchor point p_0 on the feature map, its convolutional output is computed using Eqn.2.21.

$$y(p_0) = \sum_{p_n \in R} W(p_n) X(p_0 + p_n) \tag{2.21}$$

$W(p_n)$ denotes the weights of the convolution kernel at p_n, with X representing the feature map. In *Deformable Convolution*, the offset of the sampling position, as shown in Eqn.2.22, must be added.

$$y(p_0) = \sum_{p_n \in R} W(p_n) X(p_0 + p_n + \Delta p_n) \tag{2.22}$$

The network itself calculates Δp_n based on the current feature map, thus allowing flexible adaptation to various target shapes.

Convolutional neural networks are inherently suited for high-speed numerical operations on hardware devices. Consequently, for real-time AI tasks on the device side, such as object detection [37,38] and optical character recognition (OCR) [39] in CV, CNNs are commonly used as the backbone model and play a pivotal role.

2.3.5 Recurrent Neural Network and Its Variants

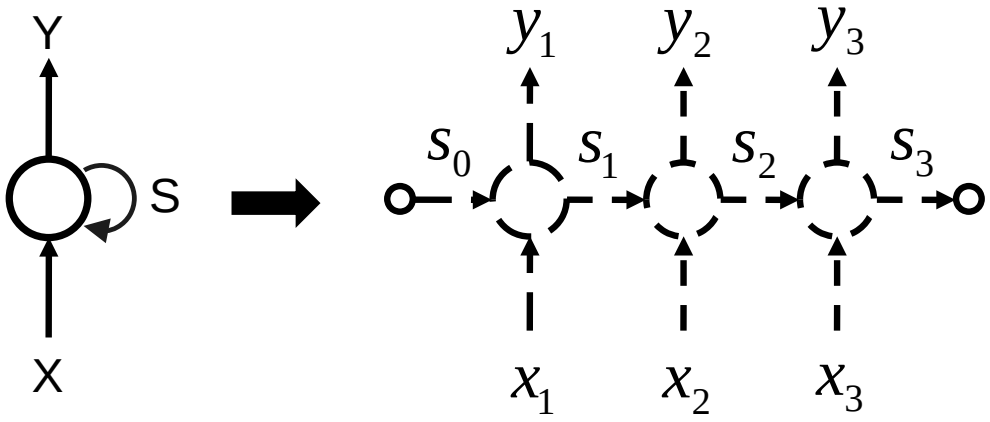


Figure 2.9: An example of recurrent neural network model

Both MLP and CNN are assumed to be context-independent, meaning that multiple samples are independent of each other. The contextual relationships between the samples are not utilized during the training and prediction stages. However, in the fields of ASR, NLP, and video action recognition, two neighboring samples in temporal order are often related. This contextual relevance becomes valuable information and plays a significant role in improving the accuracy of neural network models. Consequently, the recurrent neural network (RNN) [40], which can capitalize on this condition, excels in these tasks.

Fig.2.9 illustrates the forward propagation process in an RNN model. For each neuron, there is an additional state branch feeding back into the input stream, in addition to the inference result. Consider three consecutive temporal samples, x_1, x_2, and x_3. At a given moment i, x_i denotes the input, y_i the output, and S_i the model's state. At each

moment, x_i, along with the state S_{i-1} from the previous moment, is inputted into the RNN, resulting in output y_i and a new model state S_i. This process allows later samples in the chronological sequence to incorporate historical outputs, enhancing the model's performance. Additionally, for offline data, a bi-directional RNN is employed, where each neuron utilizes both historical and subsequent information during its forward stage.

The original RNN model is limited by short-term memory. When a sequence is sufficiently long, transferring information from earlier to later time steps becomes challenging for the RNN. Moreover, RNN models use shared weight matrices at each time step. As the length of the time sequence extends, these weight values are prone to excessively increase or decrease, leading to the vanishing or exploding of gradients during error backpropagation.

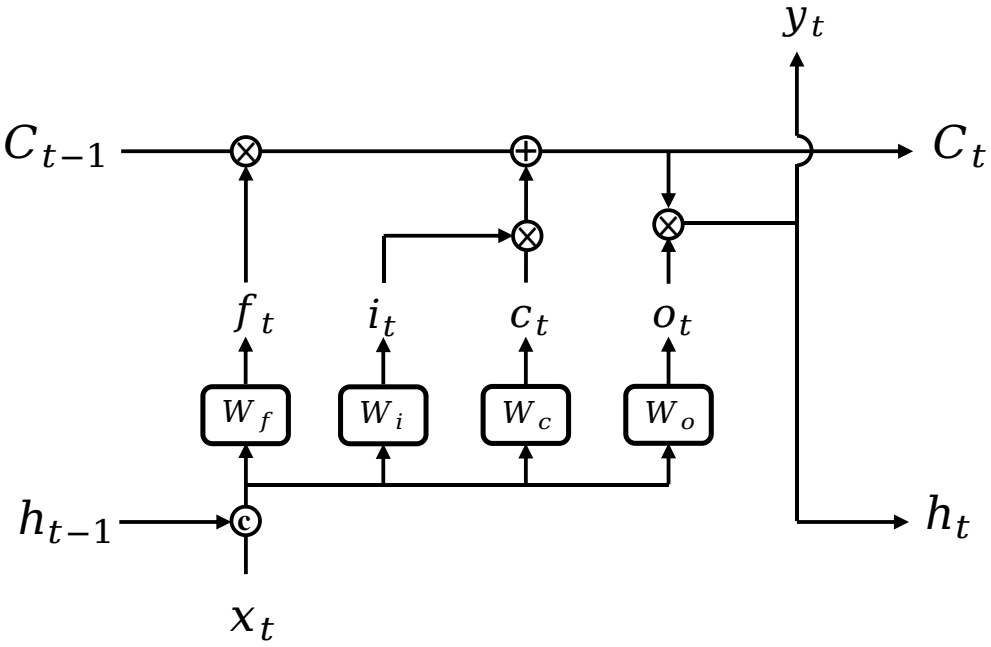


Figure 2.10: An example of long short-term memory model

The long short-term memory (LSTM) [41] model, a classic variant of the RNN model, is depicted in Fig.2.10. This figure illustrates the LSTM structure, where $\times$ denotes the multiplication operation, $+$ signifies the addition operation, and c is the concatenate operation. Equations 2.23 through 2.28 detail the LSTM model's computational process.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + B_f) \tag{2.23}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + B_i) \tag{2.24}$$

$$c_t = tanh(W_c \cdot [h_{t-1}, x_t] + B_c) \tag{2.25}$$

$$C_t = f_t \times C_{t-1} + i_t \times c_t \tag{2.26}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + B_o) \tag{2.27}$$

$$y_t = h_t = o_t \times tanh(C_t) \tag{2.28}$$

In the context of an MLP layer, W and B stand for the weight and bias, respectively. The σ symbol represents the *Sigmoid* activation function, while *tanh* refers to the *Tanh*[4] activation function. LSTM incorporates three gating mechanisms—f_t, i_t, and o_t—to regulate the retention and omission of historical data. This design enables LSTM to effectively minimize the impact of irrelevant data, adeptly manage long-term dependencies, and mitigate the issues of gradient vanishing and exploding.

2.3.6 Attention and Transformer

RNN and its variants utilize historical information and are extensively applied in NLP and ASR fields. However, they exhibit certain limitations:

1. When dealing with excessively long sequences, their capability to capture distant information is insufficient.

2. Samples appearing later in a sequence depend on the outcomes of preceding samples, prohibiting parallel computation and consequently slowing down the training and inference processes of RNN models.

The newly proposed *Attention* mechanism [8] addresses these issues, rapidly becoming a leading approach in NLP, ASR, and CV due to its robust global relationship modeling abilities. Fig.2.11 illustrates the forward propagation in a standard *Attention* layer.

$$Q = (x + PE) \cdot W_q \tag{2.29}$$

$$K = (x + PE) \cdot W_k \tag{2.30}$$

[4]Hyperbolic Tangent

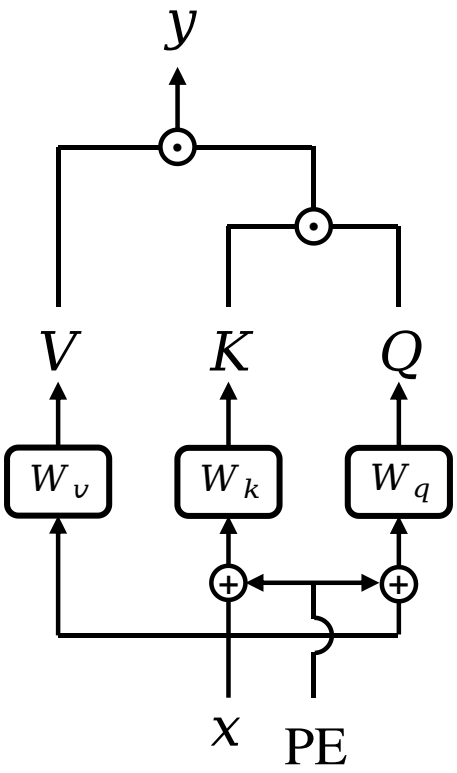


Figure 2.11: An example of standard self-attention module

$$V = x \cdot W_v \tag{2.31}$$

$$y = softmax(\frac{Q \cdot K^T}{\sqrt{d}}) \cdot V \tag{2.32}$$

In the *Attention* module, three weights, W_q, W_k, and W_v, transform the input x into Q (Query), K (Key), and V (Value), respectively. Position encoding (PE) is also integrated. Given that each sequence sample is equidistant from all others in the *Attention* module, it's crucial to incorporate positional encoding to denote the positional relationships among different sequence samples. K and V are combined with PE and undergo vector inner product calculation. Subsequently, the attention weight is derived using the *Softmax* activation function. Here, d denotes the dimension of K. This attention weight is then applied to V to generate the final output, where each new sample y_t represents the interaction result of x_t with all input samples.

Self-attention occurs when Q and K originate from the same input, while cross-attention arises when they derive from different inputs. The *Attention* module not only captures long-distance contextual relationships but also enables faster parallel compu-

tation since each sample's processing is independent of its temporal predecessors and successors.

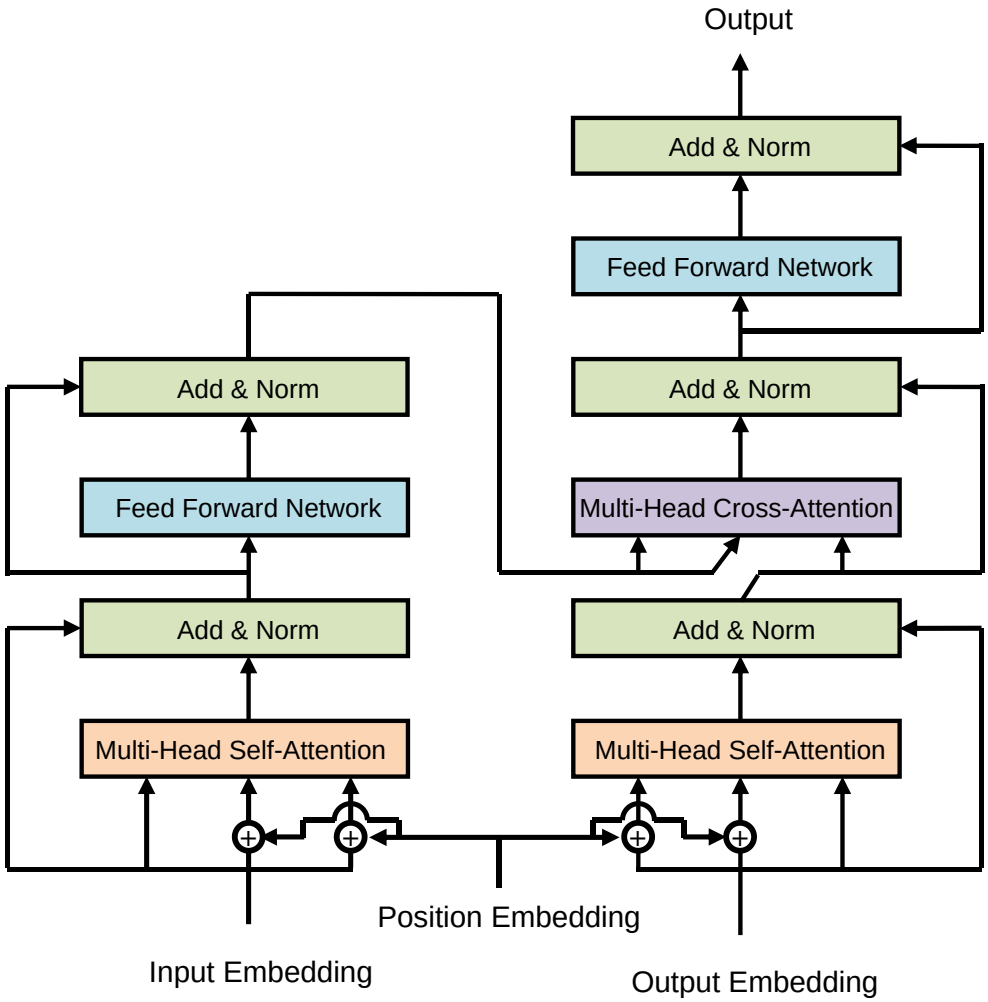


Figure 2.12: An example of classic Transformer model

Transformer [8], a DNN model, incorporates the *Attention* mechanism. As depicted in Fig.2.12, a typical *Transformer* comprises *Encoder* and *Decoder* components. The *Encoder*, which includes the self-attention module, processes Q, K, and V from identical data sources, refining the input features. The *Decoder* engages both self-attention and cross-attention modules, facilitating interactions between the outputs of the *Encoder* and the inputs of the *Decoder*. *Transformer* processes an input sequence of M samples and predicts an output sequence of N items, effectively converting a sequence of any length to another of a different length (Sequence-to-Sequence).

Variants of *Attention* exist as well. For instance, *Multiple Heads Attention* [42] enables the focus on diverse feature distributions by conducting separate attention operations on different dimensional features of the inputs. Additionally, while the standard *Attention*

computes weights for all samples in each sample to capture global information, this can be inefficient when only a fraction of the information is pertinent. *Deformable Attention* [43] addresses this by selectively concentrating on the most valuable samples at any given time, significantly enhancing computational efficiency.

2.4 Supervised Training and Self-supervised Training

Regardless of the chosen error function, the training of DNN models necessitates computing the discrepancy, defined as the difference between the model's output and the reference value. Consequently, every sample in the training dataset must possess corresponding reference values. When these reference values are actual ground truth labels, the process is termed *supervised training*. However, certain DNNs, such as generative adversarial networks (GANs) [44], do not require genuine labels. In GANs, the reference value is determined based on whether the input to the *Discriminator* is authentic or not. Similarly, RNN language models derive the reference value from the subsequent word in the input sentence. This approach is known as unsupervised training.

Typically, ground truth labels are acquired through manual annotation. As the demand for data escalates, the expense of manual labeling has become a significant bottleneck in research and development. Although unsupervised training exists, true labels remain indispensable for the majority of downstream deep learning tasks. Recently, to lessen the reliance on labeled data and enhance the versatility of DNN models, the notion of pre-trained large-scale models has garnered considerable interest.

Pre-training a large-scale model involves constructing a DNN with numerous parameters and training it using a vast quantity of unlabeled data. Subsequently, the model can be directly applied to various tasks, leveraging its robustness (Zero-Shot) [45] [46]. Alternatively, it can be fine-tuned with a minimal amount of labeled data for specific downstream tasks (Few-Shot) [47–49], or undergo knowledge distillation [50] to transfer its learned features to a smaller model. In any case, substantial volumes of unlabeled data are essential for pre-training these DNN models.

In pre-training, true labels are not utilized. This process typically involves a *Mask* technique, where certain input values are obscured, and the model's training objective becomes the prediction of these masked segments. This method is known as *self-supervised training*. It is extensively employed in large language models (LLMs), such as *BERT*[5] [51] and *GPT*[6] [45,52]. Self-supervised training is also prominently featured in CV, exemplified by *dino* [53], and in ASR, as seen in models like *Wav2Vec* [48, 49].

[5]Bidirectional Encoder Representations from Transformers

[6]Generative Pre-Trained Transformer

2.5 Development Tools and Environments

2.5.1 PyTorch Deep Learning Framework

The forward propagation, error backpropagation, and gradient descent in a DNN model involve matrix operations between the data and the model's weight matrices. To efficiently organize, manage, and optimize the extensive computing operations, utilizing various deep learning development tools, known as *deep learning frameworks*, is crucial. PyTorch [13], a popular framework, constructs dynamic computing graphs during forward propagation. This feature enables flexible model structure design and custom training process configurations. Additionally, its robust open-source community has led to PyTorch's widespread use in both academic research and industrial applications.

2.5.2 Chip and Inference Acceleration Framework

GPUs[7] are predominantly utilized for image processing and excel in matrix operations, making them a popular choice for training and inference of DNN models. To enhance the efficiency of matrix operations at the hardware level, several vendors have introduced toolkits specifically tailored for GPUs in DNN models, such as the *CUDNN* library developed by *NVIDIA*[8]. Additionally, chips specially designed for DNN models, like NPUs[9] and TPUs[10], have emerged. The advent of these hardware technologies has enabled the deployment of increasingly large DNN models in real-world applications with real-time processing capabilities. Although DNN models are typically trained using development tools such as PyTorch, significant efforts are being directed towards optimizing the model inference stage post-training, as exemplified by the *CAFFE* [54] deep learning framework. In recent years, the installation of AI-related applications in edge devices has escalated. Manufacturers such as *HISI*[11], *NOVT*[12], *RockChip*[13], and others have introduced their own chips and proprietary deep learning inference frameworks. These frameworks facilitate the deployment of DNN models on edge devices, although they may impose certain limitations on model structure.

[7] Graphics Process Units
[8] www.nvidia.com
[9] Neural Process Units
[10] Tensor Process Units
[11] www.hisilicon.com
[12] www.novtech.com
[13] www.rock-chips.com

2.6 Other Deep Learning Technologies Involved in This Study

2.6.1 Learning Rate Decay and Warm-up Policy

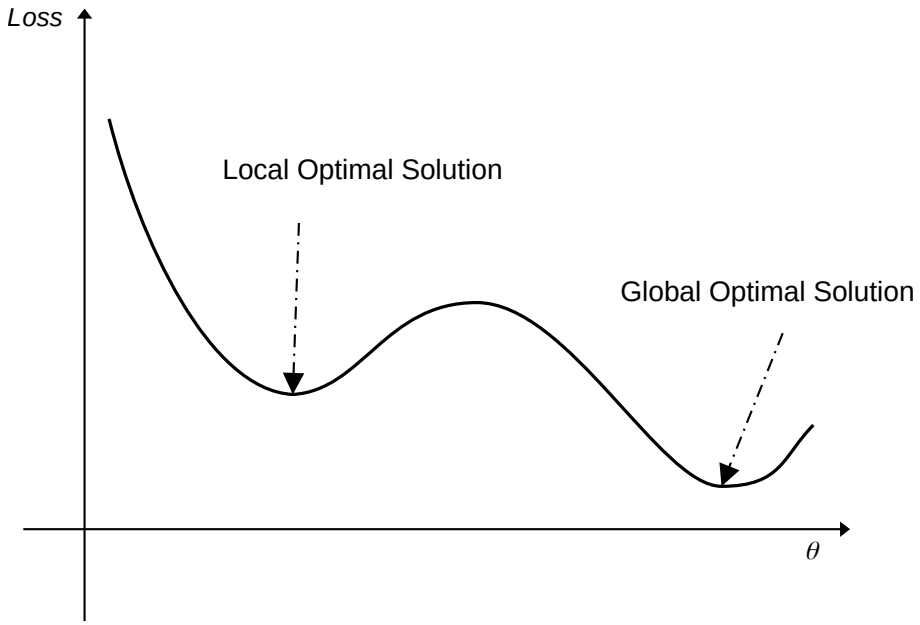


Figure 2.13: An example of locally optimal solution in gradient descent

The DNN model is trained using gradient descent, where the error progressively diminishes and eventually stabilizes. As delineated in Eqn.2.3, the model parameters are updated by subtracting a value composed of the gradient and the learning rate, denoted as η. In the initial stages of training, the model parameters are substantially distant from the final solution. Hence, a higher learning rate is preferable to expedite convergence towards the optimal solution. Conversely, in the later stages, a large learning rate may cause the model parameters to oscillate around the optimal solution without actual convergence. Thus, it is advisable for the learning rate to decrease as training progresses. However, as illustrated in Fig.2.13, the DNN model's parameter space may contain multiple local optima. A too-small learning rate during training can lead to convergence at a local optimum instead of the global optimum.

An effective solution is to implement a learning rate soft activation strategy. This approach involves temporarily increasing the learning rate to assist gradient descent in surpassing local optima after initial convergence. This method is referred to as the *warm-up policy* [55, 56].

2.6.2 Deep Residual Learning

Each layer of a CNN refines the features from the previous layer. Theoretically, deeper layers in a CNN can extract more features. However, results do not always align with this assumption.

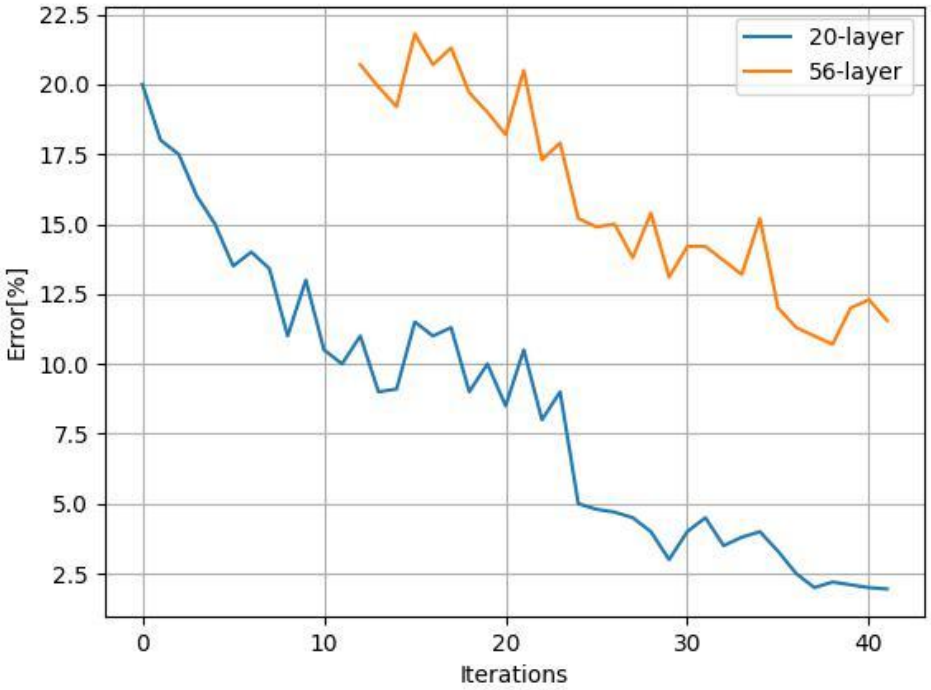


Figure 2.14: Relationship between model error and model depth

Fig.2.14 displays the experimental results of a pioneering study [26], which used different CNN layers on CIFAR-10[14] dataset. The figure presents results from the training dataset. It indicates that the error rate is lowest at 20 layers and increases at 56 layers. As mentioned earlier, this is attributed to the network's backpropagation process, which updates the gradient through the chain rule. With an increasing number of layers, the gradient tends to vanish, leading to ineffective weight adjustment in earlier layers. Consequently, networks with more layers exhibit higher training errors and poorer performance in both training and testing. While techniques such as weight initialization and batch normalization can mitigate this issue, they cannot fully resolve it. The proposed deep residual learning method [26] addresses this challenge by enabling the training of deeper neural network models.

Fig.2.15 illustrates the structure of a residual block, which includes an identity map-

[14]https://www.cs.toronto.edu/ kriz/cifar.html

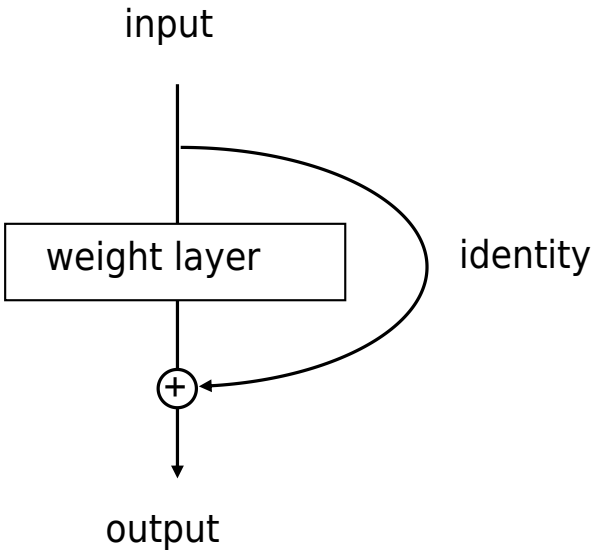


Figure 2.15: An example of residual block

ping that allows input data to flow directly into the output. During the backpropagation of error, this shortcut enables the error to be directly transmitted back to earlier network parameters without reduction, ensuring that each layer, regardless of the model's depth, receives sufficient gradient for parameter updates.

2.7 Conclusion

In this chapter, I introduced the principles of deep learning, encompassing the current state of the art, various classic neural network models, and several advanced model training methods.

Deep learning, a subset of machine learning, aims to enable machines to perceive and reason akin to humans. Central to deep learning are DNN models. A DNN consists of a directed computational graph, incorporating parameters and operators. Input data is processed through this computational graph, interacting with these parameters to yield predictive outputs.

Before deployment, the neural network model requires training using Gradient Descent to ascertain optimal parameters for a specific task. The fundamental process of gradient descent involves calculating the discrepancy between the model's predictions and the actual ground truth values. This error is then back-propagated to all model parameters using the chain rule of differentiation. Subsequently, these parameters are updated to align the predictions more closely with the ground truth values in subsequent prediction iterations.

DNN models are diverse, each suited to different tasks. For instance, the MLP model is apt for linear mapping, the CNN model excels in image processing, and the RNN is ideal for handling contextually relevant data. Among these, the Transformer model has gained prominence in fields like NLP, CV, and ASR in recent years, owing to its robust global information modeling capabilities. Moreover, the continuous proposition of deep learning technologies, such as Normalization and Pre-training, aids researchers in constructing larger and deeper DNN models.

Currently, propelled by advancements in computer hardware and the proliferation of big data, deep learning research is thriving. DNN models are increasingly being integrated into real-life applications, including smart homes and autonomous driving. Delving into the potential of deep learning within the ASR domain is a promising endeavor, with foreseeable valuable contributions.

Chapter 3

Deep Learning in Automatic Speech Recognition

In Chapter 2, I introduced deep learning and DNN models.

This chapter delves into the fundamentals of ASR systems and the significant role deep learning plays within these systems. The discussion begins with traditional ASR systems grounded in statistical models, followed by an exploration of ASR systems that leverage deep learning methods.

Initially, I discuss the extraction of acoustic features. Subsequently, I examine the Gaussian mixture model for modeling the probability distribution of acoustic features, the hidden Markov model for modeling the transition probability of these features, and the N-Gram model for predicting the probability of natural language occurrences. The narrative then shifts to describe how DNNs are increasingly supplanting these statistical models. The chapter culminates by presenting the latest advancements in end-to-end technology, showcasing ASR systems modeled entirely on deep learning principles. In this context, I also introduce various decoders used in speech recognition.

3.1 Acoustic Feature

When a person speaks, the exhaled air from their lungs becomes energized as the vocal cords close and the resultant vibrations resonate through both the nasal and oral cavities. This process gives rise to sound waves characterized by distinctive resonance peaks. During the transmission of these sound waves to the listener's ears, environmental noise can potentially interfere with the clarity of the spoken sounds.

In Fig.3.1, we illustrate the appearance of a sound wave after it has been recorded by a microphone and quantized. The sample rate is set at 8 KHz, and the bit width is 16 bits. Subfigure (a) depicts the sound waveform of the word "hello" recorded in a quiet environment, while subfigure (b) showcases a segment of white noise. Subfigure (c)

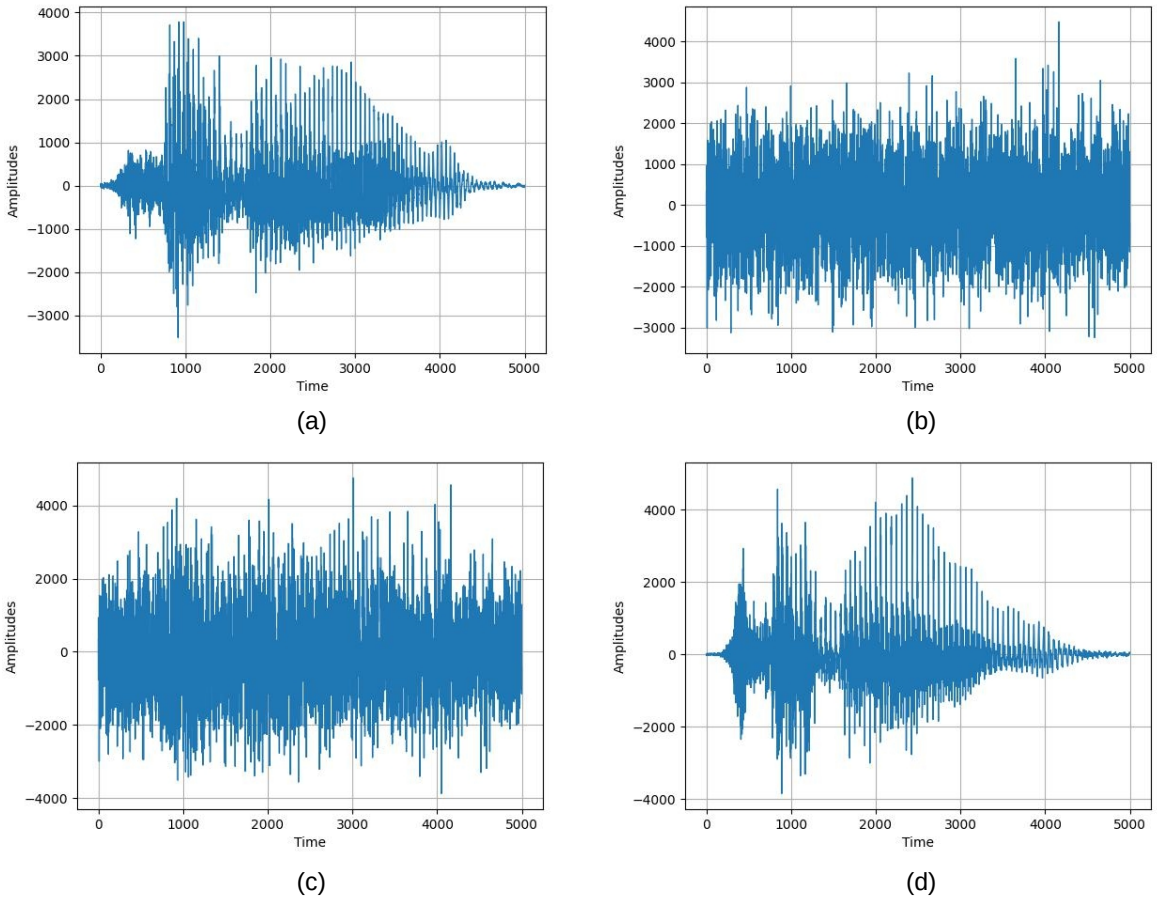


Figure 3.1: An example of speech waveforms of “hello”

represents the outcome of superimposing subfigures (a) and (b) in chronological order.

It is evident that when clean speech and background noise overlap, the speech signal undergoes significant distortion. This distortion arises from variations in muscle strength, as well as differences in the speaker’s mood, which can vary among individuals of different genders and ages and even for the same person at different times. This variability leads to substantial differences in the resulting sound waves, as illustrated in Fig.3.1(d) for the word “hello” spoken by the same individual at different times. Consequently, rather than directly utilizing raw sound waves, most typical methods rely on manually designed acoustic features.

Fig.3.2(a) shows the spectrogram obtained via discrete fourier transform (DFT) from the speech signal depicted in Fig.3.1(a). It is evident that the highlighted areas in the spectrogram correspond to the presence of spoken voice. The trajectory formed by these highlighted areas indicates the frequency changes. Fig.3.2(c) and Fig.3.2(d) display the spectrograms for the waveforms in Fig.3.1(c) and Fig.3.1(d), processed similarly. Despite the significant differences between these sound waves, their spectrograms reveal discernible

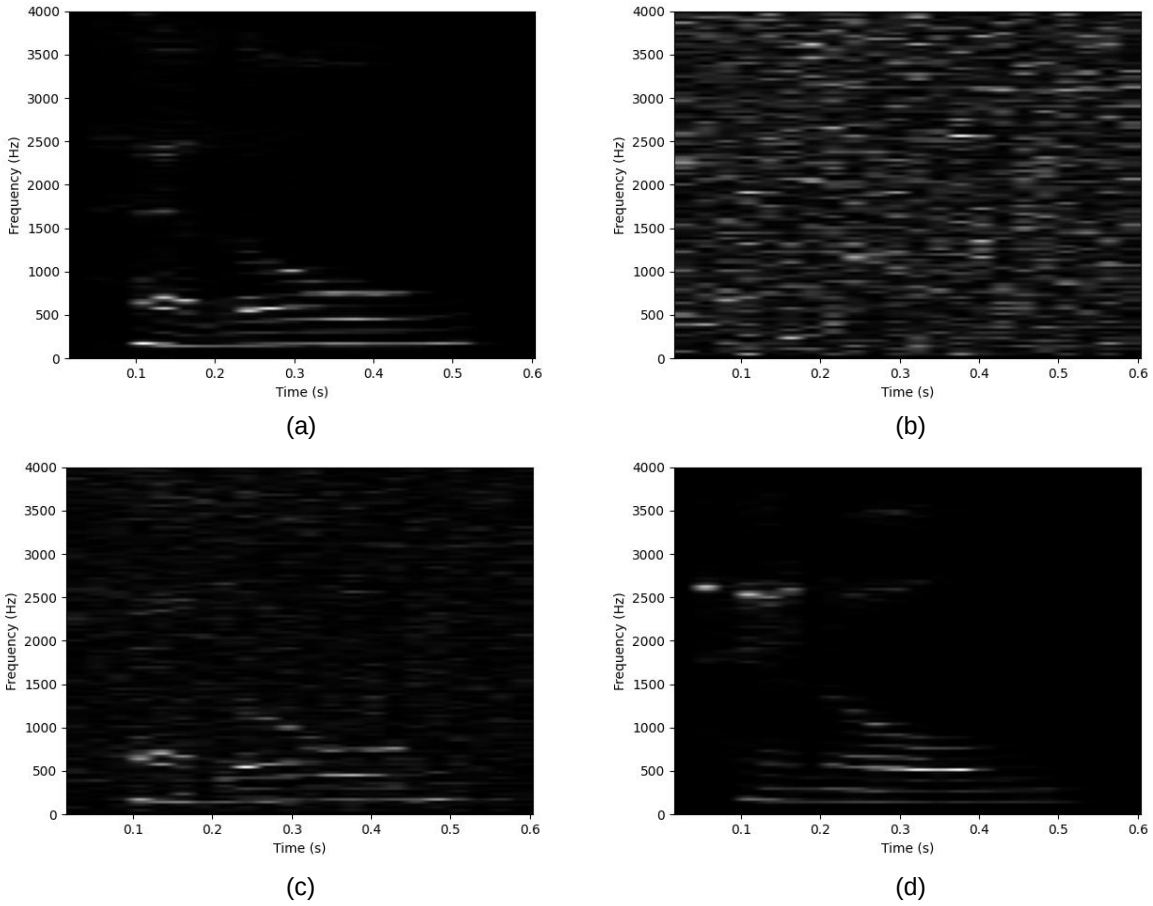


Figure 3.2: power spectrogram feature

patterns. The spectrogram, as a representation of acoustic features, provides a more straightforward model for speech sounds compared to the raw sound wave. In the realm of ASR, various other acoustic features are extensively employed. For instance, processing the spectrogram with a Mel filter bank yields an acoustic feature known as *fBank* [57]. Subsequently, applying a discrete cosine transform (DCT) to the fBank results in the *MFCC*[1] [58] feature.

3.2 GMM-HMM Acoustic Model

After extracting the acoustic features, a feature matrix X with dimensions $T \times D$ is obtained. Here, T represents the number of frames, and D represents the feature dimension. For each $t \in [1, T]$, a feature column vector $X_t = [x_1, x_2, x_3, ...x_D]$ exists. Early ASR systems utilized Gaussian mixture models (GMMs) to make predictions for each frame.

[1]Mel Frequency Cepstrum Coefficient

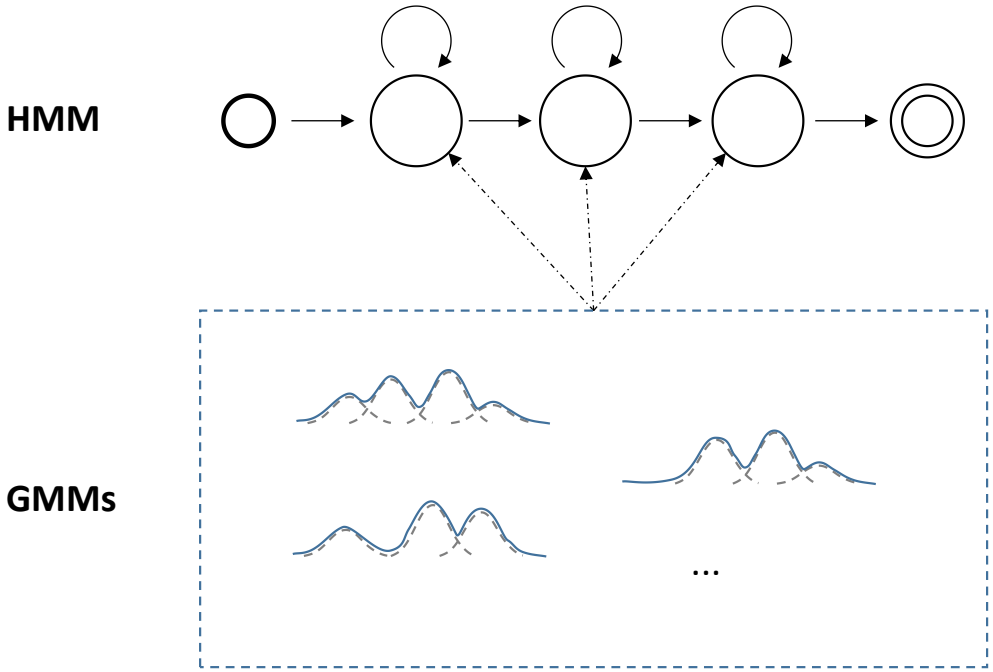


Figure 3.3: An example of GMM-HMM model

These predictions categorized the frame into a specific vocal unit, such as a phoneme, or associated it with a specific probability density function.

For example, consider the word "hello" and its set of pronunciation phonemes "*h*, *e*, *l*, *ou*". For each phoneme $p \in \{h, e, l, ou\}$, sufficient feature vectors are collected, encompassing a variety of speakers (different people, ages, genders, speaking environments, etc.), to obtain a set of feature vectors $\Phi = \{X_1, X_2, ...X_N\}$, where each $X_i \in \Phi$ represents a feature vector.

$$N_p(X_i; \mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} exp(-\frac{1}{2}(X_i - \mu)^T \Sigma^{-1}(X_i - \mu)) \qquad X_i \in \Phi \tag{3.1}$$

A Gaussian probability density distribution function is constructed using statistical methods, as in Eqn.3.1. Here, μ represents the statistical mean vector, and Σ is the covariance matrix. Typically, acoustic features are assumed to be independent in each dimension after DCT, allowing Σ to be simplified to a diagonal matrix, thereby reducing computational complexity.

$$F(X_i|p) = \sum_{k=1}^{K_p} \alpha_{p,k} \phi(X_i|\mu_{p,k}, \Sigma_{p,k}) \tag{3.2}$$

$$\hat{p} = \underset{p \in \{h,e,l,ou\}}{\arg\max} F(X_i|p) \tag{3.3}$$

Consequently, when a new feature vector X_i is presented, the observed probability of the Gaussian model for each phoneme is calculated. The phoneme $\hat{p}$ with the highest probability is then selected as the prediction for the feature vector X_i, as demonstrated in Eqn.3.2 and Eqn.3.3. In these equations, k denotes the k-th component of the GMM, $\alpha_{p,k}$ represents the weight of the k-th component, and $\phi(X_i|\mu_{p,k}, \Sigma_{p,k})$ denotes the probability value calculated using the k-th component.

Given that sounds are continuous and dynamic, multiple GMMs are typically employed for each phoneme. These GMMs are arranged sequentially in chronological order. Furthermore, GMM models for different phonemes are contextually interconnected. To address this, the hidden Markov model (HMM) has been introduced to model the transition probabilities between multiple GMMs of the same phoneme, as well as between GMMs of different phonemes. The resulting model, which integrates both GMM and HMM, is referred to as GMM-HMM [59].

Fig.3.3 illustrates a GMM-HMM model comprising three HMM nodes and several GMMs. In this model, each pair of HMM nodes (including self-transitions) is connected by a unidirectional transition arc. Each arc is assigned a weight representing the transition probability between feature vectors. For each phoneme, a GMM-HMM model is constructed. Upon receiving acoustic features, the probability of each phoneme model is computed using a *forward* algorithm. The model with the highest probability is then selected for the prediction.

Fig.3.4 illustrates the process of the *forward* algorithm. I construct a grid using a feature sequence of 7 vectors as the horizontal axis and an HMM model with 3 nodes as the vertical axis. At each black point on this grid, the probability is defined as $\Theta(i, j)$, where $i \in [1, 7]$ and $j \in [1, 3]$. Here, i is the horizontal axis coordinate representing the i-th feature vector, and j is the vertical axis coordinate representing the j-th HMM node. Each black point can be reached from the nearest neighboring black point to its left (if any) and the nearest neighboring black point to its lower left (if any). Thus, according to the Dynamic Programming strategy, the probability $\Theta(i, j)$ can be expressed as shown in Eqn.3.4.

$$\Theta(i, j|p) = (\Theta(i-1, j)\tau(j, j|p) + \Theta(i-1, j-1)\tau(j-1, j|p))F(X_i|p) \tag{3.4}$$

$\tau(a, b|p)$ represents the transition probability from HMM node a to b in the GMM-HMM model of phone p. $F(X_i|p)$ represents the Gaussian model probability of phoneme p. Therefore, $\Theta(7, 3|p)$, for this feature sequence, is the final acoustic probability given by the GMM-HMM model of phoneme p.

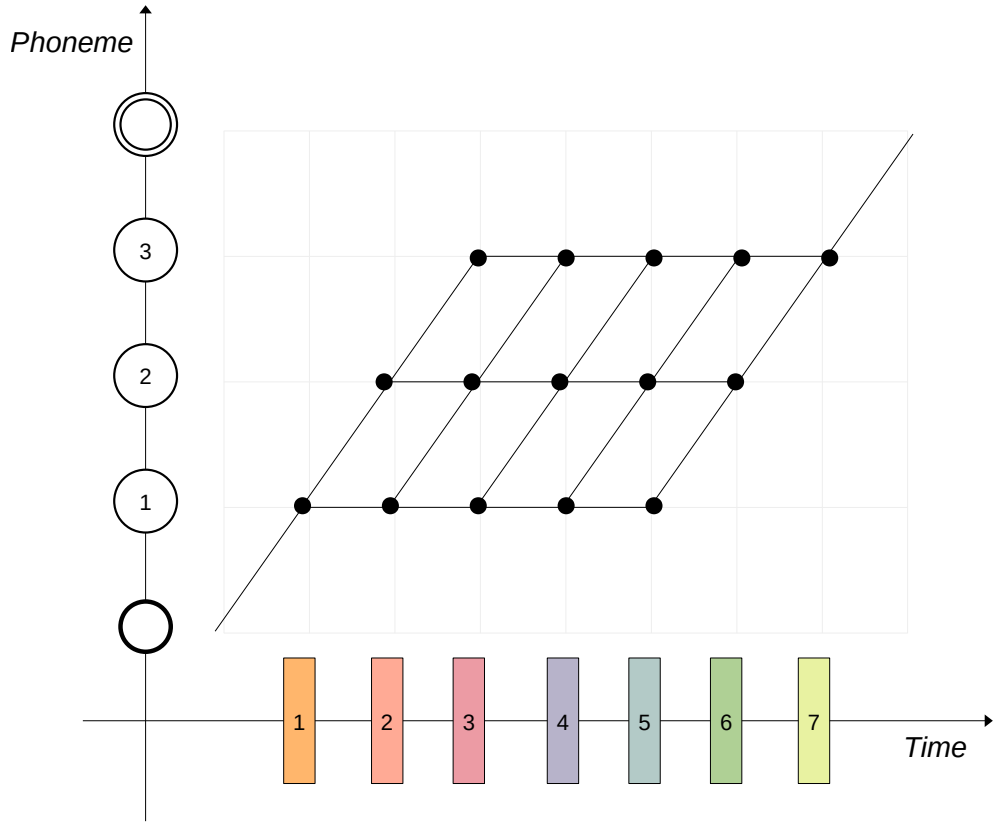


Figure 3.4: An example of forward algorithm

3.3 N-Gram Language Model

In addition to acoustic models that estimate the probability of the speech sound itself, an ASR system should also include a language model to estimate linguistic probabilities.

$$\bar{S} = \arg \max_{S \in S^*} P_{LM}(S) P_{AM}(X|S) \tag{3.5}$$

Eqn.3.5 outlines the standard paradigm for speech recognition systems. $\bar{S}$ represents the recognition result, while S^* represents the set of all possible spoken sentences. $P_{LM}(S)$ denotes the language model probability of a given utterance S, and $P_{AM}(X|S)$ represents the acoustic model probability that this utterance S is pronounced as X. In this model, it is necessary to query the probability of all word sequences. Given a possible sequence of words $S = [w_1, w_2, \cdots, w_K]$, the language model probability $P_{LM}(S)$, according to the chain rule, is presented in Eqn.3.6.

$$P_{LM}(S) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_K|w_1w_2\cdots w_{K-1}) \tag{3.6}$$

Each of these conditional probabilities is obtained by counting the frequency of occurrence of these sequences, as shown in Eqn.3.7.

$$P(w_K|w_1w_2\cdots w_{K-1}) = \frac{C(w_1w_2\cdots w_K)}{C(w_1w_2\cdots w_{K-1})} \tag{3.7}$$

In Eqn.3.7, I define $w_1w_2\cdots w_{K-1}$ as the history sequence of $w_1w_2\cdots w_K$. As the history sequence lengthens, counting becomes increasingly challenging, and the frequency of occurrence of many long sequences may even be 0. Therefore, we introduce the N-gram Markov assumption: the occurrence of the N-th word is only related to its $N-1$ preceding words.

$$P(w_K|w_{K-N+1}w_{K-N+2}..w_{K-1}) = \frac{C(w_{K-N+1}w_{K-N+2}\cdots w_{K-1}w_K)}{C(w_{K-N+1}w_{K-N+2}\cdots w_{K-1})} \tag{3.8}$$

Thus, the language model probability equation, originally presented as Eqn.3.8, can be simplified as shown in Eqn.3.9.

$$P_{LM}(S) = \prod_{k=1}^{K} P(w_k|w_{k-N+1}w_{k-N+2}\cdots w_{k-1}) \tag{3.9}$$

This language model is known as the N-Gram language model.

The N-Gram language model is a statistical approach actively used in the fields of NLP and ASR. Although DNN-based language models, such as RNN-LM [7], Transformer-LM [8] [51] [45, 52], have emerged and gained increasing attention in recent years, the N-Gram language model, known for its ease of training and querying, continues to be widely used in deployable ASR systems.

3.4 Decoding using Weighted Finite State Transducer

In the task of large-scale vocabulary continuous speech recognition (LVCSR), it's impossible to exhaustively enumerate all possible spoken utterances due to the infinite nature of the search space. Calculating the probability for each utterance and selecting the one with the highest probability as the recognition result is not feasible. To address this challenge, the use of a weighted finite state transducer (WFST) [60] as a decoding algorithm is proposed.

Fig.3.5 illustrates an example WFST graph that converts phonemes into words while simultaneously outputting the transition probability. Starting from the initial node and ending at the terminal node, each transition path represents a possible recognition result. The probability on each arc is composed of several elements: the transition probability

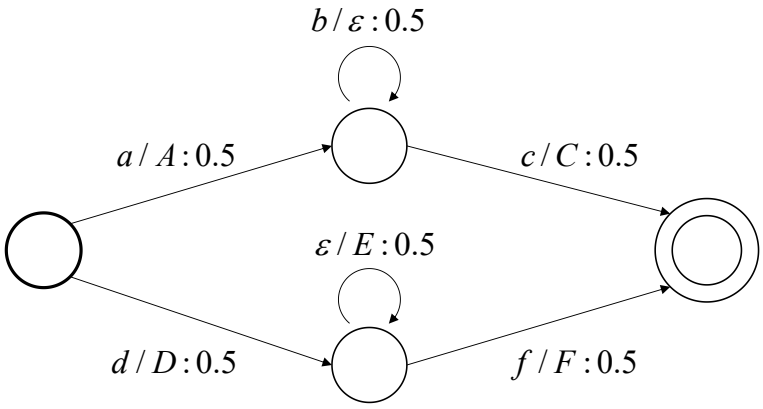


Figure 3.5: An example of weighted finite state transducer

of the HMM model, the pronunciation probability (provided by GMM or DNN), and the language model probability. In traditional ASR systems, WFST technology constructs a static search space comprising the HMM model, lexicon, context transducer, and an N-Gram language model. During decoding, a *token-passing* [61] search algorithm navigates the graph, applying pruning techniques until all acoustic feature frames are processed. The algorithm then outputs the recognition result with the highest probability.

3.5 DNN-HMM Acoustic Model

When the DNN model was introduced into the ASR system, it initially replaced GMM with DNN to model the acoustic probabilities of speech sounds. The DNN is used as a classification model to predict the specific semantic unit to which each frame of acoustic features belongs. As mentioned in Chapter 2, reference values are necessary to compute the error during the DNN's training process. However, it is challenging to label precisely which semantic unit each frame of acoustic features represents. Therefore, the training process of a DNN-HMM model proceeds as follows:

1. Train a GMM-HMM model using the expectation-maximization (EM) algorithm.
2. Use the GMM-HMM model to predict the training data, thereby generating pseudo labels.
3. Train the DNN model using these pseudo labels.

After DNN training is complete, the GMM is discarded, and the DNN is utilized to predict the distribution probabilities of semantic units.

Compared to GMM, DNN exhibits a more robust fitting capability. Typically, the accuracy of using a DNN model is higher than that of a GMM model in most ASR tasks. Widely used DNN models include the MLP, time delay neural network (TDNN) [62], and LSTM.

3.6 RNN and Transformer Language Model

The N-Gram language model struggles to model long-distance dependencies, and the memory requirements for these models expand exponentially as N increases. With the advent of RNNs, a model that naturally incorporates historical word information to predict and score subsequent occurrences, a significant advancement was made in the NLP field for language modeling. Fig.3.6 illustrates an example of an RNN language model.

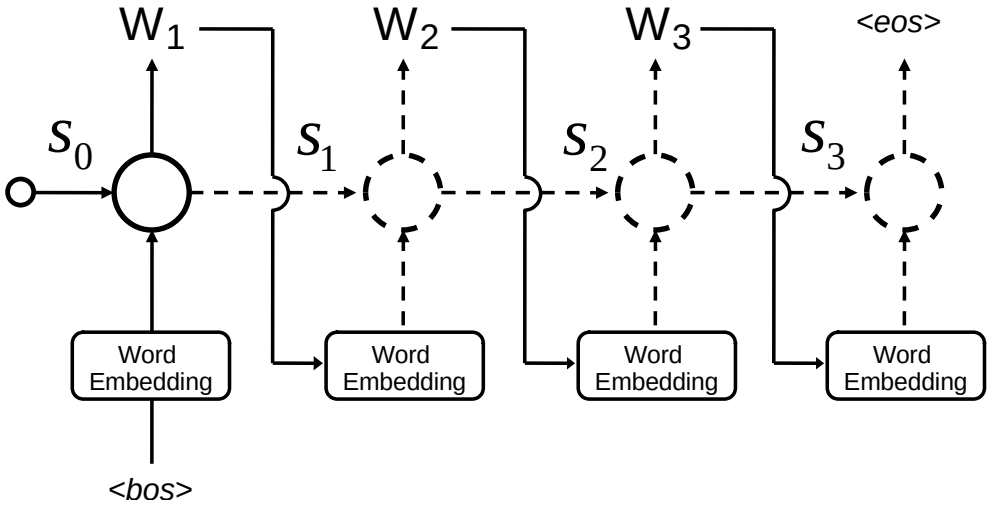


Figure 3.6: An example of RNN language model

An *WordEmbedding* module is incorporated in this model to encode all words in the vocabulary into a D-dimensional vector. As discussed in the previous chapter, neural networks necessitate numerical values as inputs. Therefore, when utilizing an RNN language model, it is essential to convert words into numerical values acceptable by the neural network. A straightforward approach is to assign a unique integer ID to each word, following the sequence of the word vocabulary list. These IDs are then transformed into one-hot encoding. For instance, if our vocabulary list comprises N words, the i-th word will have the ID i, and its one-hot encoding will be an N-dimensional column vector with all elements being 0, except for the i-th element, which is 1.

However, this one-hot encoding approach, while treating all words equally, presents several issues:

1. The data dimensions are excessively large. For natural languages, including English and Chinese, which may have vocabularies exceeding $100,000$ words, the dimensionality of such one-hot encoded vectors becomes unmanageably large, hindering efficient computation.

2. Feature sparsity is another concern. The high-dimensional vector, with only a single non-zero position, leads to a wasteful use of computational resources.

3. Inadequate information representation. One-hot vectors fail to capture the similarities between different word vectors.

A solution to these problems is the *Word Embedding* technique [63]. This approach involves training methods that assign a D-dimensional column vector to each word in the vocabulary. Typically, D is significantly smaller than N, thereby substantially reducing data dimensionality. Moreover, since word embeddings learn the latent representations of words during training, they are capable of effectively evaluating the similarity between two words. In Fig.3.6, the *WordEmbedding* module represents a table of trained word embedding vectors, enabling the retrieval of a word's embedding vector from this table.

When calculating the probability of a sentence $S = [w_1, w_2, w_3]$ using the language model, initially, w_1 is inputted into the neural network, which then outputs the probability of the next word being w_2. Subsequently, w_2 is inputted, and the probability of the next word being w_3 is computed. Compared to the N-Gram language model, the RNN language model leverages the entirety of historical word information, typically resulting in higher accuracy.

3.7 End-to-End Model

In general, the training process of a DNN-HMM-based ASR system follows these steps:

1. Extract acoustic features.
2. Train the N-Gram language model.
3. Train the GMM-HMM acoustic model.
4. Build the WFST decoding graph.
5. Predict pseudo labels.
6. Train the DNN-HMM acoustic model.

This process can be cumbersome. To simplify the ASR system, research is increasingly focusing on using DNNs to replace both GMM and HMM models, or even the entire process from feature extraction to decoding. Such systems are referred to as end-to-end (E2E) systems, with the DNN model within being termed the E2E model. Prominent E2E models are based on connectionist temporal classification (CTC) technology [6, 64] and sequence-to-sequence (S2S) technology [3–5]. As discussed in Section 3.5, DNN training necessitates a correspondence between each acoustic feature frame and its semantic units, known as *alignment*. The challenge in DNN-HMM training lies in the impossibility of directly obtaining the semantic units for each acoustic feature frame. Thus, correct alignment is essential, typically facilitated by the GMM-HMM model. CTC technology

simplifies this process by using the *forward* algorithm to compute the total probabilities of all possible alignments directly.

The training of the S2S model, similar to that of the RNN language model, involves using a sequence of historical semantic units as a prompt to predict the next semantic unit. Unlike CTC, the S2S model can output words directly, often without the need for post-processing operations like de-duplication, and integrates language model information. S2S models, including RNN-transducer [65] and transformer-transducer [66], significantly streamline the ASR system pipeline and have become a crucial area in ASR system research in recent years.

3.8 End-to-End Decoding

An E2E model that directly outputs word sequences does not require a decoder. However, since the number of words in any language is typically vast, training an E2E model is challenging. A common compromise is to model phonemes, syllables, and other subword-level semantic units. Once the model makes predictions, these subword-level semantic units are then converted into word sequences. Some pioneering studies still exploit the advantages of WFST to construct static decoding graphs [67]. These approaches involve constructing a decoder from a word-level N-Gram language model and a lexicon that converts subwords into words. This method boasts high decoding speed, accuracy, and maneuverability. Furthermore, the recognition accuracy can be enhanced by employing superior language models for the re-scoring of decoding results. Another prevalent approach is the lexicon-based beam search [68] decoding algorithm, along with its various variants. Algorithm 1 illustrates the decoding process of a basic lexicon beam search algorithm [69].

The lexicon-based beam search algorithm dynamically expands the search path during the decoding process. Although this method is slower than the WFST decoder, it offers greater scalability. For instance, it can seamlessly incorporate hotword probabilities. Additionally, it usually requires less memory than the WFST decoder. Therefore, the beam search decoder has also become a widely adopted decoding algorithm in E2E OCR systems and E2E ASR systems.

3.9 Conclusion

In this chapter, I introduce both the traditional statistical ASR system and the deep learning-based ASR system.

In the field of AI, ASR serves as a prevalent mode of perception and interaction. After a speaker vocalizes, the sound is captured by a machine and transformed into numerical data, which a computer can then process. Certain downstream tasks, such as

Algorithm 1: lexicon beam search

Input: probability matrix M with T frames and D dimensions, beam size β
Output: words

```
// beam is object whose properties contain at least last char id lastcid,
  probability p, and the tracker to a lexicon trie node ptr;
lastBeams = {};
put an initial beam into lastBeams;
for t to T do
    newBeams = {};
    for pbeam In lastBeams do
        set variable cids to all char ids which ptr of pbeam can arrive to on trie;
        for cid In cids do
            nbeam = pbeam.clone();
            put cid into nbeam;
            update probability p of nbeam;
            put nbeam in newBeams;
        end
    end
    sort newBeams by probability p of each beam;
    clear lastBeams;
    select top β beams from newBeams and put them into lastBeams;
end
return words of best beam in lastBeams;
```

keyword search (KWS) [70], can be performed to understand the voice directly without full recognition. However, converting sounds into text facilitates easier recording of evidence and natural speech understanding. The process of converting speech to text is, in a narrow sense, ASR.

Early methods relied on the grammatical rules of natural human language, limiting ASR systems to processing utterances predefined by linguists' speaking rules. Subsequently, models based on statistical methods, including GMM, HMM, and the N-Gram language model, emerged in ASR, enabling systems to handle large-scale, continuous, and naturally spoken utterances. The basic process of the statistical method involves:

1. Extracting acoustic features from the sound.

2. Modeling the distributional and contextual probabilities of these features using GMM and HMM.

3. Dynamically listing all possible text sequences using the N-Gram language model.

4. For each text sequence, querying the corresponding GMM and HMM model based on the dictionary, and computing the likelihood of observing the acoustic features in these models.

5. Selecting the text sequence with the highest probability as the prediction result.

The integration of DNN models into speech recognition has gradually supplanted these statistical models, exhibiting superior accuracy and deployability. Such ASR models are termed Hybrid models. DNN models, in contrast, abandon the modeling paradigm of statistical language models, directly processing acoustic features or sounds to produce text sequences. These ASR models are referred to as E2E models. E2E models significantly streamline ASR system modeling and, due to their close association with deep learning, enable the incorporation of advanced deep learning features, such as speech-visual multimodality. The exceptional performance of ASR systems based on deep learning has inspired the motivation for this research. In this study, I investigate both Hybrid and E2E models.

Chapter 4

A Novel Real-Time Automatic Speech Recognition Toolkit

In Chapter 3, I introduced the construction of deep learning-based ASR systems, including both Hybrid and E2E systems.

In this chapter, I will delve into the pipeline of real-time ASR systems and propose a novel toolkit for building real-time Hybrid ASR systems on cloud server. Firstly, I present the current state-of-the-art in real-time ASR technology. Following this, I discuss the Kaldi speech recognition toolkit, which is highly relevant to this study. Subsequently, I provide a concise overview of the proposed ExKaldi-RT toolkit.Afterward, I detail the implementation of the proposed system. Finally, I demonstrate, through experimental evidence, the accuracy and real-time performance of this proposed toolkit.

4.1 An Overview Real-Time Automatic Speech Recognition Technology

4.1.1 Real-Time Automatic Speech Recognition Pipeline

ASR system is deployed in two ways: offline ASR system and online ASR system. The latter is also referred to as a real-time ASR system or streaming ASR system. In this thesis, it is termed the "real-time ASR system." Offline ASR systems typically process a complete utterance of speech with known starting and ending points. For example, the speech transcription tools used in meetings are usually offline ASR systems. An offline ASR system is easier to implement than a real-time ASR system because the recording environment is more controlled, and environmental noise is better regulated. However, most ASR tasks are expected to deliver recognition results promptly after the speaker has finished speaking. This is the case with speech assistants on smartphones and speech control systems for home appliances. In these scenarios, recording distances and

environmental noise often vary, making implementation more challenging.

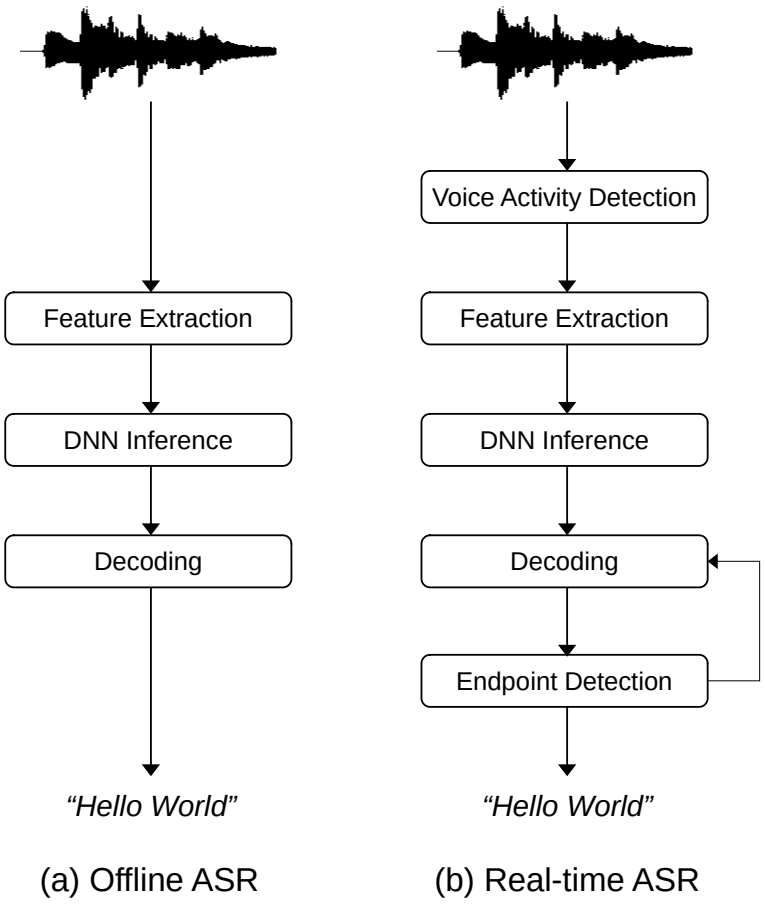


Figure 4.1: Differences between offline and real-time ASR system

Fig.4.1 illustrates the differences between an offline ASR pipeline and a real-time ASR pipeline. In contrast to the offline ASR system, the real-time ASR system incorporates an additional endpoint detection module to identify the start and end points of spoken voice. Endpoint detection can be accomplished using voice activity detection (VAD) [71] technology or by analyzing the decoding results of the recognition process. The presence or absence of an endpoint often leads to variations in the behavior of the DNN model and the ASR decoder. Unlike offline ASR systems, real-time ASR systems struggle to access global contextual information and lack additional post-processing features to enhance accuracy, such as re-scoring with an RNN language model [72]. Although these limitations can lead to reduced accuracy, real-time ASR systems offer timely results, significantly enhancing the interactivity of ASR technology. Consequently, research in real-time ASR systems has garnered widespread attention, focusing on speech signal processing, streaming modeling, and online decoding.

4.1.2 Neural Network Models for Real-Time Recognition

The primary distinction between real-time and offline ASR systems lies in their data processing capabilities. Real-time ASR can only access historical data up to the current moment, limiting its contextual information. The MLP model processes only the acoustic features of a single frame and does not utilize contextual information, making it suitable for both offline and real-time ASR systems. However, as discussed in the previous chapter, to enhance accuracy, more advanced models like TDNN, LSTM, and Transformer models leverage the contextual relationships of consecutive speech frames. Consequently, when training a DNN for real-time ASR, it's necessary to modify the internal structure of these models. For instance, the LSTM can be restricted to process input strictly in chronological order. An alternative is to employ endpoint detection technologies to identify the start and end points of speech, effectively converting the recognition process to mimic an offline ASR system without model structure restrictions. The former method offers immediate recognition results and handles long utterances efficiently, making it more suitable for real-time applications. The latter, while typically more accurate, relies heavily on endpoint detection technologies.

4.2 Kaldi Speech Recognition Toolkit

Kaldi [12] stands as a highly popular toolkit for ASR systems development. This open-source platform integrates various functionalities essential for ASR system development. These include extracting and refining acoustic features, training GMM-HMM acoustic models, training N-Gram language models, constructing WFST-based decoders, and performing post-processing on decoding outcomes, among others. Initially, Kaldi predominantly facilitated the creation of statistical-based ASR systems. With the advent of DNN in the ASR domain, Kaldi expanded its capabilities to include training DNN models and developing Hybrid ASR systems. Moreover, Kaldi enables the construction of real-time ASR pipelines.

Kaldi is primarily coded in C++. Post-compilation, developers utilize the resultant commands via a command line interface to conduct all operations necessary for ASR system training. Additionally, Kaldi incorporates a range of scripts in Perl, Python, and Shell to facilitate user tasks during ASR system training. The following list demonstrates Shell scripts used to construct a sample ASR system employing Kaldi's compiled tools.

```
1
2 local/prepare_data.sh waves_yesno
3 local/prepare_dict.sh
4 utils/prepare_lang.sh --position-dependent-phones false data/local/dict "<SIL>" data/
    local/lang data/lang
5 local/prepare_lm.sh
6
7 # Feature extraction
8 for x in train_yesno test_yesno; do
```

```
  steps/make_mfcc.sh --nj 1 data/$x exp/make_mfcc/$x mfcc
  steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x mfcc
  utils/fix_data_dir.sh data/$x
done

# Mono training
steps/train_mono.sh --nj 1 --cmd "$train_cmd" \
  --totgauss 400 \
  data/train_yesno data/lang exp/mono0a

# Graph compilation
utils/mkgraph.sh data/lang_test_tg exp/mono0a exp/mono0a/graph_tgpr

# Decoding
steps/decode.sh --nj 1 --cmd "$decode_cmd" \
    exp/mono0a/graph_tgpr data/test_yesno exp/mono0a/decode_test_yesno

for x in exp/*/decode*; do [ -d $x ] && grep WER $x/wer_* | utils/best_wer.sh; done
```

In this segment, *local/prepare_lm.sh* refers to the script designated for language model training. *steps/make_mfcc.sh* is employed for acoustic feature extraction, *steps/train_mono.sh* for training the GMM-HMM model, *utils/mkgraph.sh* for constructing the WFST decoder, and *steps/decode.sh* for the decoding process.

The following list is a part of the *steps/make_mfcc.sh* script.

```

utils/split_scp.pl $data/segments $split_segments || exit 1;
rm $logdir/.error 2>/dev/null

$cmd JOB=1:$nj $logdir/make_mfcc_${name}.JOB.log \
  extract-segments scp,p:$scp $logdir/segments.JOB ark:- \| \
  compute-mfcc-feats $vtln_opts $write_utt2dur_opt --verbose=2 \
    --config=$mfcc_config ark:- ark:- \| \
  copy-feats --compress=$compress $write_num_frames_opt ark:- \
    ark,scp:$mfccdir/raw_mfcc_$name.JOB.ark,$mfccdir/raw_mfcc_$name.JOB.scp \
    || exit 1;
```

Within this code, *compute-mfcc-feats* serves as the command-line tool designated for feature extraction. It interfaces with other tools via a pipeline symbol, enabling sequential data processing.

Kaldi offers comprehensive functionality, but since it is based on the C++ language, this inevitably leads to the following two issues:

1. Kaldi is not easy to use.

2. Kaldi does not easily embed deep learning frameworks-trained DNN models.

As outlined above, Kaldi is designed for researchers with a solid background in the ASR field and programming skills. Its lack of entry-level instructional documentation tends to deter deep learning researchers.

As discussed in Chapters 2 and 3, deep learning demonstrates significant potential in ASR. This potential is largely due to extensive research and development efforts, supported by open-source deep learning frameworks such as TensorFlow and PyTorch. DNN

models are particularly adept at fitting the distributions of acoustic features, and they have achieved impressive performance in various ASR tasks. Although Kaldi provides tools for training DNN models, the models and training methods it offers are relatively basic. Additionally, the use of C++ in Kaldi does not facilitate flexible programming. In contrast, mainstream deep learning frameworks enable the construction of highly accurate DNN models, which can further enhance ASR system performance. Despite this, Kaldi's capabilities in other aspects of the ASR pipeline remain intriguing to researchers. Consequently, several tools have been proposed in related studies, such as PyKaldi [15,16] and PyTorch-Kaldi [17], which aim to integrate Kaldi with these advanced deep learning frameworks. In my prior work, I proposed the ExKaldi [18] ASR toolkit, another Python-based Kaldi wrapper. ExKaldi allows for the seamless integration of DNN models, trained using deep learning frameworks, into Kaldi's ASR pipeline, significantly simplifying the programming process.

4.3 Motivation for This Study

In the past few years, ASR has reached the level of practicality with a lot of widely-used products, including the Google speech-to-text service, Amazon Alexa, and Apple Siri. Most of these applications are real-time ASR systems. As mentioned in Section 4.2, relevant developing tools already exist for building Hybrid ASR systems using the Python language. A key feature of these tools is providing an interface to compute acoustic probabilities with a DNN acoustic model trained by a deep learning framework and then decode with Kaldi's decoding program. Howerever, These Kaldi wrappers do not have complete tools for real-time recognition. Therefore, a toolkit that can easily build an real-time ASR system is expected.

4.4 An Overview of ExKaldi-RT Toolkit

I propose a new open-source toolkit named "ExKaldi-RT" (real-time ASR extension toolkit of Kaldi). In Chapter 3, I provided a detailed introduction to the composition of Hybrid systems and End-to-End systems. ExKaldi-RT primarily targets building Hybrid systems. In recent years, despite End-to-End systems performing well in some tasks, their recognition results heavily rely on the predictions of DNN models. The semantic units output by these models are of relatively large granularity (typically words or syllables), and they require learning both the probabilities of the acoustic model and the language model simultaneously. Therefore, End-to-End system desires the model to be sufficiently large and the training data to be abundant to ensure high confidence in the model's output. Hence, when the model size is limited or when there is insufficient data, leading to poor robustness of the DNN model, the system's recognition accuracy will sig-

nificantly decrease. However, in Hybrid systems, the acoustic model consists of DNN and HMM, where the DNN model is solely responsible for predicting the distribution probabilities of acoustic features, with the semantic units' granularity being smaller (typically phonemes or the probability distribution functions of GMM). Besides the DNN-HMM acoustic model, a WFST decoder containing additional probabilities from dictionaries, language models, and contextual speech states is also used. As the result, the HMM model and these additional probabilities can help correct some of the DNN model's erroneous predictions. Therefore, Hybrid systems have better fault tolerance to the outputs of DNN models, enabling the model to maintain good robustness even in scenarios with small model size and limited data. In addition, the self-transition structure present in HMM model makes Hybrid systems more capable of handling variations in the temporal dimension. For example, related work [73] on speech recognition for the elderly has found that due to the slower speech of older adults, Hybrid systems also exhibit better accuracy compared to End-to-End systems. In summary, Hybrid systems have also garnered considerable attention from researchers.

Kaldi possesses the full capabilities for building Hybrid systems. I have packaged Kaldi's feature extractor and WFST decoder. Unlike the above tools [15–18] mainly developed for offline ASR, ExKaldi-RT aims to build an real-time ASR environment to, hopefully, enable users to apply their original recognition models and evaluate them under actual environments. While similar real-time ASR tools [74] are available by combining Kaldi's online feature pipeline and stream management tools such as GStreamer, ExKaldi-RT implements online ASR with only Python language and is based on a DNN acoustic model trained with the deep learning frameworks. Users can utilize the rich model architectures of any deep learning framework supported by the Python language, such as PyTorch [13], TensorFlow [14], ONNX [75], CAFFE [54], without the need for additional compilation, quantization, or other operations. In the section on example code, I will demonstrate several examples of how to easily install these deep learning framework models in ExKaldi-RT's pipeline.

ExKaldi-RT provides integrated tools to construct online ASR pipelines, including recording real-time audio stream, doing VAD, transmitting data when using a remote connection, computing online feature, estimating probability, and decoding on the fly. Most of these jobs can be customized, which means that users can design their original algorithms for signal processing, feature extraction, socket packet compression, and N-best rescoring. Our Kaldi-based ASR toolkit is provided as open source, it can foster more research and development in the direction of real-time ASR systems.

I performed the benchmark experiments on the minimum LibriSpeech public dataset. The experiments showed that our ExKaldi-RT toolkit could build the online ASR system with an ideal ASR performance in real-time. The contributions of our research are as follows:

1. It provides complete capabilities for constructing a real-time speech recognition

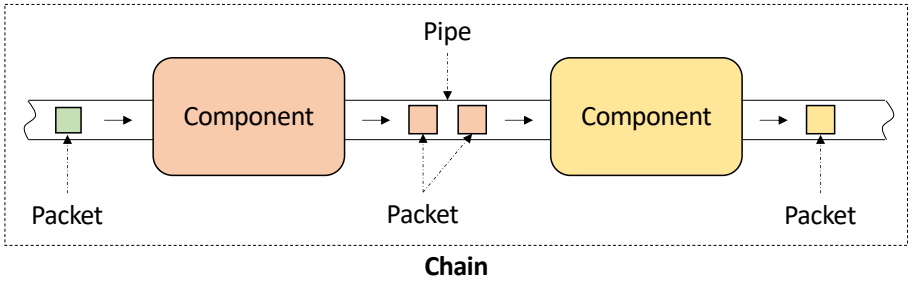


Figure 4.2: An ASR chain by connecting components in ExKaldi-RT

pipeline based on Hybrid systems with high accuracy and real-time performance.

2. It is developed in Python and supports the direct integration of DNN models trained for deep learning networks as acoustic models, making deployment of models easier.

3. It offers several advanced tricks to further enhance the accuracy and robustness in noisy environments of a real-time ASR system, such as fusing acoustic features, incorporating noise reduction models, etc.

4. It is open source.

In fact, since ExKaldi-RT was developed for speech recognition systems utilizing DNN model + WFST decoder, as a bonus scene, it also supports the development of End-to-End systems that use the same architecture, such as [67].

4.5 Design of ExKaldi-RT

4.5.1 Architecture

Fig.4.2 shows the dataflow in the ASR chain composed of components and pipes in ExKaldi-RT. Data flows through the pipes are continuously processed. Basically the ASR pipeline is organized in the following pieces:

- *Packet*: which carries data, including digitized audio signal, acoustic feature, acoustic probability, decoding result and special flag such as the endpoint used to the truncate stream.

- *Component*: which processes packets and generates new data, such as a feature extractor and a decoder.

- *Pipe*: which caches and transfers packets, and exchanges state information between two components.

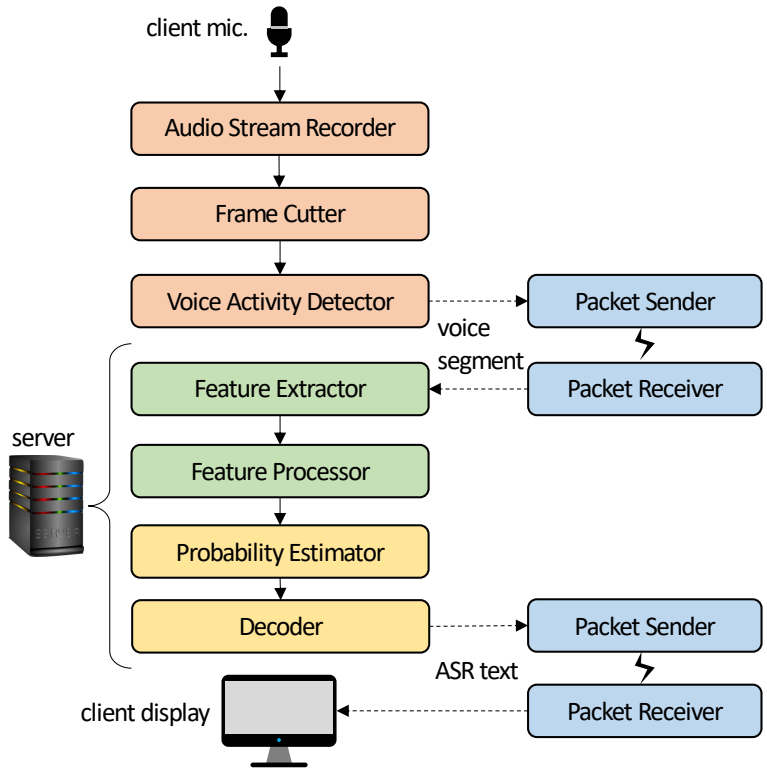


Figure 4.3: An online ASR pipeline built with ExKaldi-RT.

- *Chain*: which is a container and provides high-level interface to link and manage the ASR pipeline.

In general, a component gets packets from the input side of a pipe and appends the processed results into the output side of the pipe. All components in a chain perform the above actions simultaneously. A component monitors and modifies the state of the pipe. For example, once an error occurs, it propagates the error information forward and backward through the pipes.

Fig.4.3 shows a classic pipeline for online ASR built with ExKaldi-RT. Although this is a client-server mode, the pipeline can work in a series connection mode in ONE computer. In the current version, I do not parallelly connect components to handle multiple tasks. Instead, I adopt parallel threads or processes in a single component. This makes it easier to maintain consistency in the time of data processed by different tasks.

In this pipeline, *Audio Stream Recorder* collects audio stream from a microphone connected to the client and *Frame Cutter* cuts the stream into frames with a sliding window. *Voice Activity Detector* filters out silent audio. The components *Feature Extractor* and *Feature Processor* are used to compute online acoutic features. If using a remote connection, *Packet Sender* and *Packet Receiver* transmit data packets between the two host computers. *Probability Estimator* predicts the observed probability of acoustic features and *Decoder* outputs the ASR results to the interface of applications.

4.5.2 Voice Activity Detection

VAD plays a remarkable role in real-time ASR. I remove long silent audio in order to reduce unnecessary calculations. The VAD strategy used in ExKaldi-RT is executed as following steps:

1. Detect a continuous silence sequence,
2. For the part shorter than a threshold, keep it, and for the part longer, the threshold discards it,
3. Append an endpoint mark to truncate stream, or do not.

I support VAD with both the numerical audio stream data-based and cut frame data-based approaches. This means that in addition to raw waveforms, users can input some acoustic features to implement more complex VAD algorithms with a deep learning approach. The VAD function based on the Google WebRTC VAD module is defaultly available in ExKaldi-RT.

4.5.3 Online Feature Extraction

I have wrapped some tools to implement the details of acoustic feature extraction, such as computing fast fourier transform (FFT) and Mel filter bank (fBank). To achieve the same accuracy as Kaldi, some functions are developed with C++. For instance, I get the float floor value in the C++ environment, although it may be different in Python. I have designed several online feature extractors to compute spectrogram features, fBank features, MFCC features, and their combinations based on these tools. Besides, ExKaldi-RT supports customizing feature extraction steps that allow users to apply novel technologies. In our experiments, I show an example of speech separation using magnitude spectrum when extracting MFCC features. Classic feature transformation technologies: appending differential, splicing the context features, cepstral mean and variance normalization (CMVN) [76], and linear discriminant analysis with maximum likelihood linear transform (LDA+MLLT) [77] are available in ExKaldi-RT online feature extraction.

4.5.4 Remote Transmission

ExKaldi-RT can work on a client-server architecture. In this case, ExKaldi-RT provides tools to transmit packets between the client and the server. A recommended way is to collect audio streams and do VAD on the edge client. Therefore, unnecessary silent audio will not be transmitted to the feature extractor on the server. When sending packets to remote host, the sender will add verification information to the packets' header and resend them if the receiver fails to verify the data. Verification information includes the size of packets, an endpoint mark, data type and others. In the current version, I prepare

a simple function to encode data packets without compressing, but users can still apply their original encoding algorithms to improve transmission efficiency.

4.5.5 Online Decoding

ExKaldi-RT wraps the *LatticeFasterDecoder* function of Kaldi with C++ code and implements the real-time decoding based on WFST. Instead of computing acoustic probability with the *NNET* [78] model in Kaldi and other related toolkits, the decoder accepts probabilities predicted by the acoustic probability estimator. Users can embed their original DNN-based acoustic model trained with a deep learning framework. DNN models that invoke context or history, such as TDNN and LSTM, are also available. Besides endpoint detection provided by Kaldi, I also recognize the endpoint flag marked by the previous components, such as the voice activity detector. If an endpoint is determined, the decoder will output N-best results. It is possible to rescore N-best list with an RNN-based language model further to cherry-pick the best hypothesis.

4.6 Example Code

The following list shows an Python example code for building a typical online ASR pipeline with the ExKaldi-RT toolkit, using only 15 lines of code. The version of Python is python==3.8.16. This script can quickly deploy the series of components to complete an ASR task by performing audio recording, computing MFCC feature, predicting acoustic probability, and decoding. A container named *chain*, which link the components each other, can easily drive the pipeline. This code would dynamically display debug information and recognition results on the standard output. These results are still accessible to external interface applications.

```
1
2  #Read stream from microphone and cut frames.
3  reader=StreamRecorder()
4  cutter=ElementFrameCutter()
5
6  #Compute Online MFCC features.
7  extractor=MfccExtractor()
8  processor=FeatureProcessor(FrameSlideCMVNormalizer())
9
10 #Predict probabilities with original DNN function.
11 extimator=AcousticEstimator()
12 extimator.acoustic_function=Your_Function
13
14 #Decode the probability with WFST graph.
15 decoder=WfstDecoder(symbolTable="words.txt",
16                     silencePhones="1:2:3:4:5",
17                     frameShiftSec=0.01,
18                     tmodel="final.mdl",
19                     graph="HCLG.fst")
```

```
20
21 #Link these components.
22 chain = Chain()
23 chain.add(reader)
24 chain.add(cutter)
25 chain.add(extractor)
26 chain.add(processor)
27 chain.add(acousticmodel)
28 chain.add(decoder)
29
30 #Run and display the results dynamically.
31 dynamic_run(chain)
```

In the pipeline of online ASR, matrix operations are frequently executed. In order to improve the calculation speed, most components adopt the batch computing. Although the batch computing increases latency, it can comprehensively improve real-time performance. In some components, I further process batch data with multiple CPU threads, which is currently the bottleneck hindering the real-time factor, especially when using multiple neural networks.

In the example code above, *YOUR_FUNCTION* represents the inference function of the DNN model used to calculate acoustic probabilities. In this function, you can use any deep learning framework and original model to compute the model's output. Below, I have provided an example of inference using the classic PyTorch and ONNX [75] frameworks. PyTorch is currently one of the most popular deep learning frameworks in the world. ONNX is a unified format for neural network models, and models trained with deep learning frameworks such as PyTorch and TensorFlow can generally be converted to this format. Therefore, ONNX is often used for model inference.

The sample code for PyTorch is as follows. The version of PyTorch is torch==1.12.1.

```
1
2 import torch
3 from .my_models import MLP
4
5 # Initialize the model
6 model = MLP(input_dim=128)
7 checkpoint_state = torch.load("mlp.pt")
8 model.load_state_dict(checkpoint_state, strict=True)
9 model.eval()
10 model.cuda()
11
12 # Forward model
13 def Your_Function(feats,*args,**kwargs):
14     feats = torch.from_numpy(feats).float().cuda()
15     with torch.no_grad():
16         probs = model(feats).cpu().numpy()
17     return probs
```

The sample code for ONNX is as follows. The version of ONNX is onnxruntime-gpu==1.9.0.

```
1
```

```
import onnxruntime as ort

# Initialize the model
model = ort.InferenceSession("mlp.onnx", providers=["CUDAExecutionProvider"])

# Forward model
def Your_Function(feats,*args,**kwargs):
    feats = feats.astype("float32")
    probs = model.run(["probs"], {"feats":feats})[0]
    return probs
```

4.7 Experiments

The experiments show that the DNN-based acoustic model using ExKaldi-RT performs as well as the Kaldi's original model. Besides, external modules can be easily implemented in online ASR pipeline built with ExKaldi-RT. To demonstration this, therefore, a simple implementation and evaluation of external VAD functions and the speech separation approach is also presented.

4.7.1 Experimental Setup

Our experiments used the minimum LibriSpeech [9] corpus, which contains 5 hours clean data for training and 2 hours clean data for evaluation. When evaluating the performance of online recognition, I simulated the process of real-time audio recording by continuously reading the data stream from the files. The standard training recipe (including the DNN model training) can be found in Kaldi's examples. I used the alignments, lexicons and decoding graph generated after the *tri3b* stage of standard recipe. In the experiments of speech separation, I used NOISEX-92 [79] backguroud noise dataset to synthesize the experimental data. All these dataset are publicly available. I adopted TensorFlow (version 2.4.0) as the deep learning engine to train the DNN-based model in the experiments. The machine configuration was as follows: CPU was Core-i7 6950X 3.0GHz, memory was 128 GB, GPU was GeForce GTX-1080Ti, and OS was Ubuntu 18.04.

4.7.2 DNN Acoustic Model and Online CMVN

I firstly trained a widely-used fully-connected DNN acoustic model and embedded them into the pipeline. The *word error rate* (WER) obtained on offline and online ASR environment respectively is shown in Table 4.1. In order to provide a baseline, I quote the results of Kaldi's chain model which incorporated additional i-vector feature. Although I only used MFCC feature and maximum likelihood criterion to train the DNN model, thanks to the more flexible modeling approaches provided by the deep learning framework, I achieved the ASR performance similar to Kaldi's baseline. After applying this model to the online ASR system, there is a small increase of 0.16 points in WER. This

Table 4.1: WERs [%] and RTF of offline and online ASR systems with the DNN acoustic model

	offline	**online**	
	WER	**WER**	**RTF**
baseline	18.58	18.49	—
DNN+Constant CMVN	19.81	19.97	0.57
DNN+Slide CMVN	—	22.66	0.60
DNN+Slide CMVN w/ pre-stats	—	20.16	0.64

may be caused by few loss of precision when exchange float value between Python and C++ codes.

Table 4.1 also detailed the performance of different CMVN approaches available in ExKaldi-RT. The CMVN statistics of this experiments are collected by per-speaker. Constant CMVN is an extreme case that statistics are collected in advance and fixed when computing online feature. The more common case is to accumulate statistics and apply CMVN within a sliding window. I set sliding window size as an empirical value: 6 seconds. The results show that if the statistics collected in advance are filled in the first 6 seconds window, the accuracy loss of online ASR could be minimized (compared with constant CMVN, only 0.35 points decreased in this experiments). Moreover, the real-time factor (RTF) shown in Table 4.1 reveals that ExKaldi-RT could achieve the real-time performance.

4.7.3 Acoustic Feature

The experiments consider different acoustic features, i.e., 13 dims. MFCC feature, 24 dims. fBank feature, impoved 40 dims. MFCC feature with LDA+MLLT transformation, as well as their maxture feature. Table 4.2 compares the WERs and RTFs of the online ASR systems using these features. The first three static features are processed by combining with Δ+$\Delta\Delta$ and splicing with ten frames of context. The mixture feature is composed of 39 dim. MFCC (13 static+Δ+$\Delta\Delta$), 24 dims. fBank, 40 dims. LDA+MLLT and then spliced with three frames of context. ExKaldi-RT supports in handling multiple feature combinations. The mixed features achieved the best WER (18.56%) at the sufficiently acceptable RTF in this set of experiments.

4.7.4 Voice Activity Detection

I then show the results regarding the VAD functions. In order to compare the impact of VAD on short speech and long speech, I divided the test dataset into *short* and *long* subsets by checking whether the duration is longer than ten seconds. Table 4.3 shows WER and RTF of different VAD methods on these two subsets. Mark T in Table 4.3

Table 4.2: WERs [%] and RTF using various acoustic features

	WER	**RTF**
MFCC	19.97	0.57
fBank	19.65	0.54
LDA+MLLT	19.35	0.67
Mixture	18.56	0.75

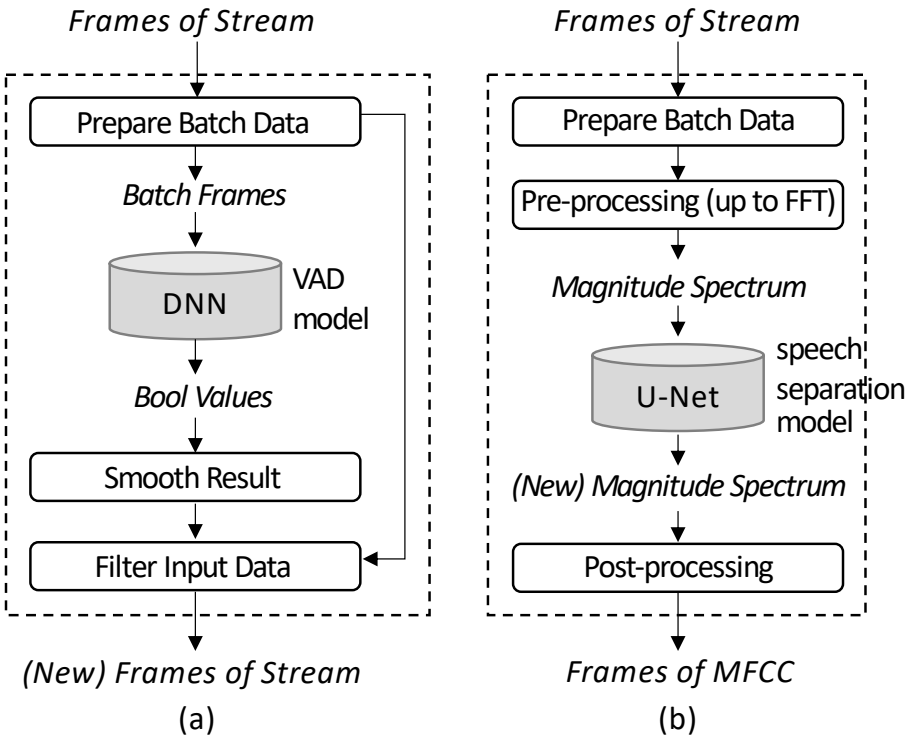


Figure 4.4: Embedded deep learning models of (a) VAD and (b) speech separation in the ASR pipeline.

means truncating speech stream. I trained a VAD model with an evaluation accuracy of 94% using Kaldi's pre-trained alignments. The DNN model was embedded in *Voice Activity Detector* as shown in Fig.4.4 (a). When using the WebRTC, the RTF is really decreased by excluding some silent audio. The time increases when using the DNN-based VAD; however, it shows that the VAD has better detection performance, which positively impacts WER, than the general VAD approach based on signal processing theory. Even on *short* subsets, the accuracy of speech recognition is improved by 0.35 points. This experiment shows that ExKaldi-RT supports VAD models to improve real-time performance and even ASR accuracy.

The test data in the experiments were cut out for each utterance in advance, therefore, it is difficult to have a great improvement in the performance of real-time recognition. In a

Table 4.3: WERs [%] and RTF using the VAD comporments

	Short		**Long**	
	WER	**RTF**	**WER**	**RTF**
w/o VAD	20.44	0.67	19.13	0.57
WebRTC VAD w/ T	23.23	0.59	22.29	0.52
WebRTC VAD w/o T	21.17	0.60	20.05	0.53
DNN VAD w/o T	20.09	0.71	19.52	0.59

Table 4.4: WERs [%] and RTF using the speech separation model

	WER	**RTF**
clean	22.66	0.60
clean+noise	47.58	0.67
clean+noise w/ separation	38.09	0.86

real environment, it can be inferred that a good DNN-based VAD model is able to reduce WER and RTF. In this set of experiments, I mainly focus on verifying that ExKaldi-RT can easily realize external VAD functions in the pipeline.

4.7.5 Speech Separation

Speech separation with a deep learning approach has also received much interest in recent years. This set of experiments embedded a deep learning speech separation model into the feature extractor. In this thesis, I build a U-Net [80] speech separation model and also show that it is easily installed in the ASR pipeline. The U-Net model was embedded in *Feature Extractor* as shown in Fig.4.4 (b). I first calculate the magnitude spectrogram from the input speech signal and then input it into the speech separation model to generate a new magnitude spectrogram with the noise removed. I simply configure a signal-to-noise ratio (SNR) of 15 dB to synthesize clean data and noise to generate a training and testing dataset. Table 4.4 shows the experimental results by the ASR pipeline. I did not collect the CMVN statistics for new generated data in advance, therefore, I compare the results only using sliding CMVN. When the acoustic model trained on the clean dataset is installed in a noisy environment, the WER drops sharply. However, by introducing the speech separation model, the WER has been improved with a 9.49 points reduction. I show that the NN-based separation model can be easily used in the online ASR pipeline built by ExKaldi-RT and can improve the ASR performance in a more realistic noise environment.

4.8 Conclusion

In this chapter, I introduced the pipeline of real-time ASR systems and propose a new toolkit ExKaldi-RT for building read-time Hybrid ASR systems.

Kaldi is one of the most popular development toolkit in the field of ASR. It provides a interacted tool chain for building a HyBird ASR system. However, it cannot directly use DNN models trained by deep learning frameworks. Some pioneering researches have embedded DNN models into the Kaldi's ASR system by wrapping Kaldi tools in Python language. However, these tools can only build offline speech recognition systems, but do not provide the ability to build real-time ASR pipelines.

A new toolkit, ExKaldi-RT, is proposed in this study. ExKaldi-RT is also a wrapper for the Kaldi tool. It is characterized by:

1. It is developed using Python language and easy to embed DNN models for deep learning frameworks.

2. It provides a complete tools for building a real-time ASR pipeline, including recording from microphone, network transmission, feature extraction, DNN prediction, decoding, etc.

3. It can flexibly fuse multiple acoustic features to improve the accuracy of ASR system.

4. It can install denoising DNN models to improve the robustness of ASR system.

In the experiment, I showed that ExKaldi-RT can achieve competitive accuracy and real-time performance.

Chapter 5

An Improved End-to-End ASR Model and Decoder on Embedded Devices

In Chapter 4, I introduced real-time ASR technology and proposed the ExKaldi-RT toolkit for developing Hybrid ASR systems deployed on cloud side.

This chapter outlines the methodology for deploying a real-time ASR system on the device side, discussing the challenges and proposing a new lightweight E2E ASR model and decoder. I begin by clarifying the target equipment for this study and describing the background and prior research related to this study, as well as outlining its objectives. Then, I provide a concise summary of the methodology and key findings. Subsequently, I detail the improved E2E model structure and its training strategy. After that, I discuss the optimization efforts for the decoder. Finally, I present the configuration and results of experiments conducted to validate the effectiveness of this study.

5.1 Edge Devices With AI Applications

With the development of computer hardware, more and more edge devices have started to be equipped with high-performance computing chips and memory, such as smartphones and micro-computers. In recent years, the rise of deep learning has driven these edge devices to provide efficient environments for AI, including NPU modules that can accelerate neural network computations. Some system-on-chip (SoC) devices, such as surveillance cameras and smart-home appliance controllers, have also begun to incorporate NPUs with considerable computing power and memory capacity to run neural network models for tasks like object detection and voice command recognition. However, unlike high-end edge devices like smartphones, the development of these low-end hardware devices is usually only for one or a few specific tasks, for example, an electronic door lock that

only needs to carry a face recognition model. Therefore, to control development costs, these devices are equipped with smaller memory, weaker NPU computing power, and restrictions on the DNN model structures that can be used.

The target of this research is these low-end edge devices. Thus, if designing a lightweight ASR system for these devices, the hardware limitations to consider are:

1. The system must has a low memory usage. The operating memory of these low-end edge devices is usually 2 or 4 GB, which includes the space for the system and other software operations. Therefore, it is generally desired that a ASR system occupies less than 500 MB of memory during operation (including the model, decoder and cache during computation).

2. The neural network model must have a low computational requirement. The NPU power of these low-end devices is usually around 1 TOPS[1], so the model structure should be optimized by reducing the number of parameters (expected to be within 50 M), avoiding dense computations, and fusing operators. For example, if a BN layer is placed after a CNN layer, the CNN+BN can be fused into a new CNN layer by linear transformation during model inference.

3. The model structure is limited. These low-end devices usually have their own neural network inference frameworks, which strictly limit the neural network layer structures they can support. Therefore, when designing general neural network model structures, it is necessary to avoid using some custom model structures.

Typically, after int8 quantization, models of a size and structure comparable to ResNet-50 can still run well on most low-end devices. These limitations will play an important role as references in our design of the model structure and in the evaluation of model performance.

5.2 The Motivation of This Study

Recently, ASR systems are widely used in various embedded equipment, such as smart home devices or in-vehicle infotainment systems. ASR systems benefit from some cutting-edge technologies, including DNN, WFST, and self-supervised learning.

When running ASR systems, large amounts of memory and computing power are normally required. Current ASR systems require thousands of hours of transcribed speech to achieve an acceptable level of performance. Using Kaldi and the Mini-Librispeech [9] dataset, which has only 5 hours of speech transcription, the memory occupied in the hard disk for building a WFST decoding graph is about 530 MB. On the other hand, most ASR systems for industrial products require low-latency real-time recognition of

[1]Tera Operations Per Second

speech. That is, the corresponding time of the ASR algorithm needs to be completed within one second. Therefore, an ASR system is typically deployed on a cloud server and communicates with edge devices through a wireless network. However, performing speech recognition on embedded devices locally is also an important requirement when the network is unavailable or when the user's personal data is not allowed to be uploaded.

When installing an ASR system on an embedded device, less computation and lower memory occupation are expected. Compared with the traditional modeling approach of the ASR system (which makes a decoding graph from a HMM-based acoustic model (AM), a pronunciation lexicon transducer and a statistical language model (LM)), E2E model (which only has a single end-to-end trained neural network model) is favored because of its smaller model size and state-of-the-art accuracy. Regarding the E2E modeling framework, there are a range of excellent papers on the topic of this framework and how it can be used to address the challenges of implementing speech recognition on embedded devices [19–21, 81–84].

In [20], a combination of CTC-based E2E AM and RNN-based LM and beam-search decoding was used for speech recognition in mobile and embedded devices. In this model, the number of parameters in the final E2E model was about 15 M. StreamE2E also achieved faster computation and better recognition accuracy on mobile devices by optimizing the structure of the RNN transducer (RNN-T) [65] model and using text-to-speech (TTS)-based speech augmentation.

Thus, speech recognition has been achieved on edge devices, which are devices that have many available resources. However, two major problems must be solved in order to drive speech recognition on many general-purpose edge devices. First, on most embedded devices, model architecture is limited, depending on the SDKs provided by the hardware manufacturers. The Android platform is widely used on edge platforms, and there are some excellent open-source frameworks like TensorRT [85] and NCNN [86] to help developers deploy their models. However, there are still many platforms and embedded devices active in industrial products that are not compatible with these open-source frameworks. In addition, many edge devices do not utilize advanced neural network structures like LSTM and transformer. Therefore, many of the network structures proposed in previous work may be hard to install. Second, differing from the decoding algorithms using WFST, the beam search family decoders do not build a large static decoding graph, but dynamically expand the search path during decoding. Therefore, they are also mainstream decoding algorithms for E2E models on mobile devices. However, generally speaking, the performance of beam search decoders is a matter of contention, and related research is less concerned with the optimization of beam search algorithms for ASR decoding. The problems discussed above are the focus of this thesis.

Therefore, my goal is to propose a smaller, faster ASR system with stronger SDK compatibility and high accuracy. I hope that the number of parameters of the neural network model is within 1 M, the memory size of the whole ASR system is less than 10

M, the single core occupancy on an ARM-32 architecture chip, which is often used on a low-end embedded device, is not more than 20%, and the char error rate (CER) on the open source dataset Librispeech is less than 10%. Furthermore, I hope that the CER on a specific domain can reach less than 5%. This setting can ensure that my ASR system can be used on most mobile products for custom ASR tasks and provide a good user experience.

5.3 An Overview of Proposed Lightweight Model and Decoder

In this chapter, I propose the following improvements to satisfy the above requirements in order to run highly accurate and memory-saving speech recognition on edge devices. First, CNN have a natural computational advantage on GPUs or NPUs and are compatible with most SDKs. Therefore, regarding the model structure, in this study, I propose to mainly use CNN layers in order for the model to be deployed on edge devices as easily as possible. To compensate for the shortcomings of the CNN layer in capturing information over a long distance, I fuse features of different scales. Moreover, instead of manual speech features such as MFCC or fBanks, I adopt a CNN feature extraction module to compute acoustic representations from audio waveforms directly. With this model, all inference processes are assigned to the dedicated GPUs or NPUs. Second, in the training stage of the model, I use a self-supervised learning approach based on wav2vec2.0 [49]. This strategy ensures that my tiny model can be converged and, in the case of the small amount of real data I can collect, plays an important role in improving accuracy. In my experiments, my E2E model performs competitively compared to some popular lightweight models, in terms of not just the model size but also the error rate. Finally, in terms of decoding, I propose a decoder using an improved prefix beam search to handle CTC probability output. I separate acoustic pruning for different prefixes, which prevents the right path from being excluded by mistake. In addition, instead of a word piece LM, which always takes up a large amount of memory, I design a joint method of sub-word LM and initialism LM to capture the context information within and between words, respectively.

My proposed model had about 0.79 M parameters. Using the greedy decoder, the CER on Librispeech was 13.57%. Although this accuracy rate is still far from my goal, it is better than that of some popular lightweight models, including a ResNet-18 [26] model and MobileNetv3 [87] model. Then the model was transferred to my specific task. Using the improved prefix beam search [88] decoding algorithm, the error rate on my test dataset was 4.86%, slightly higher than the result 3.99% of the widely used WFST decoder, my ASR system only have four thousandths of the memory compared to said WFST decoder. My improvement approaches yielded good results experimentally.

The contribution of this chapter is to show that it is possible to realize an ASR model

with high ASR performance that works in real-time and with little memory on edge devices. This ASR model could be realized without using a transformer model, which provides high ASR accuracy, but with a model mainly based on convolutional layers, utilizing pre-training based on wav2vec2.0, learning rate adjustment, and a prefix beam search algorithm with an initialism LM.

5.4 Design of Lightweight End-to-End ASR Model

5.4.1 An Outline of Model Architecture

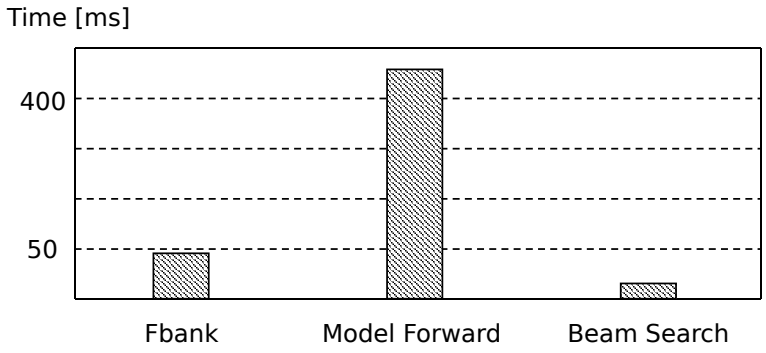


Figure 5.1: The power consumption of an E2E ASR system on an embedded device

In recent years, deep learning models have shown amazing potential in vision-related tasks and are widely deployed on embedded devices, such as ResNet [26] series, YOLO [37, 38] series, MobileNet [87, 89] series, and so on. A CNN-based model has been proven to have better compatibility with most mobile devices and has also been introduced into ASR tasks because of its a small mount of parameters and fast inferencec [90, 91]. Fig.5.1 shows an example of the CPU inference time of a dummy E2E ASR system on an low-end embedded device with a 4-core Cortex-A53 1GHz CPU and 2G of RAM. The acoustic feature is 296×128-dimensional fBanks and the model is built using ResNet-18 backbone. When running with CPU only, the ASR algorithm takes up a non-negligible amount of computational resources, not only during model forwarding but also during the extraction of acoustic representations. As a result, I design my model architecture according to the following criteria:

- mainly using CNN layers in order to reduce parameters and ensure that the model can be deployed to most embedded devices,
- being able to transfer the entire model to the GPU or NPU in order to speed up the inference process,

as well as reducing the number of parameters of the model.

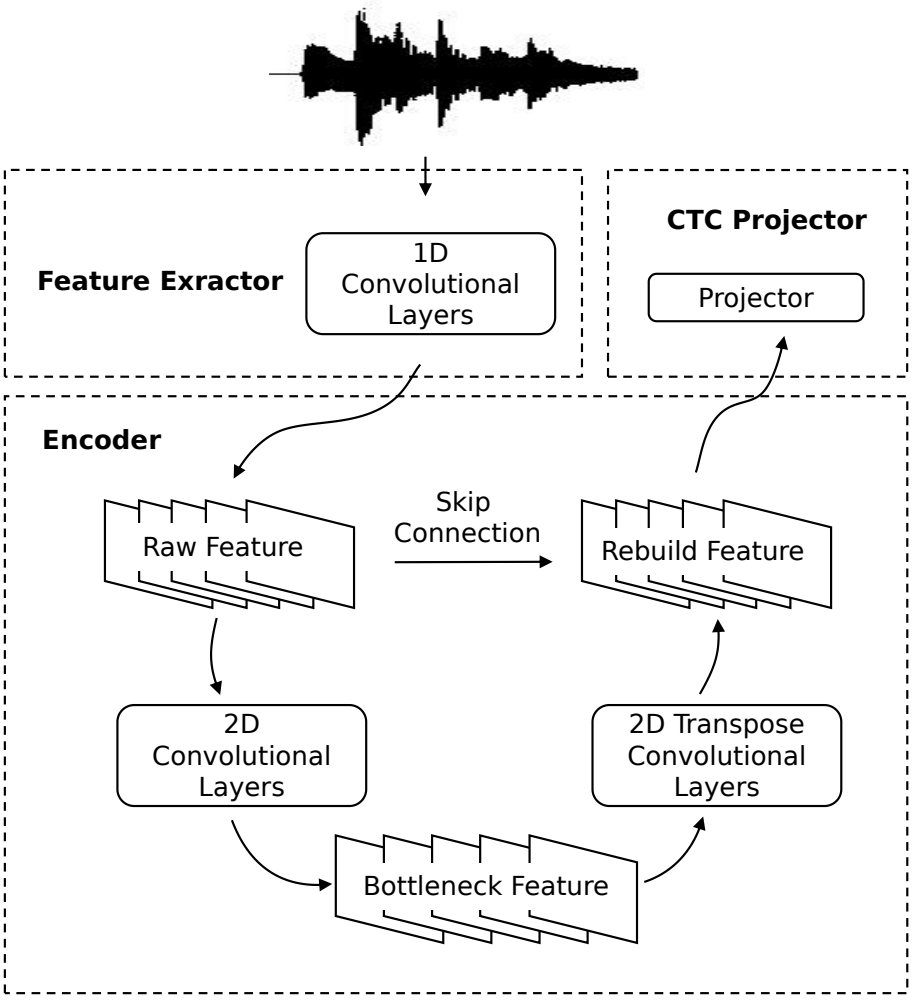


Figure 5.2: The basic architecture of my E2E model

Fig.5.2 illustrates the basic modules of my E2E model. The model consists of three components: the convolutional feature extractor "*Feature Extractor*," the feature reconstruction module "*Encoder*," and the projection head "*CTC Projector*." *Feature Extractor* module extracts raw acoustic representations from 1-dimensional audio waveforms. In *Encoder*, with reference to feature pyramid network (FPN) [92] and U-Net [93], I use multiple CNN layers to embed the raw acoustic representations to different scales and skip connections to fuse these features in order to capture local and long-distance linguistic relations in a context of a certain length. As shown in Fig.5.2, there are three acoustic features generated in my model: *Raw Feature*, *Bottleneck Feature* and *Rebuild Feature*. These features can be used for different tasks. For the ASR task, I select *Rebuild Feature*, which contains more sophisticated semantic information, and finally input it to the *CTC projector*. In my experiment, I trained various *Feature Extractor*s and *Encoder*s, which varied in performance in terms of model size and accuracy. Section details the proposed

model structures.

5.4.2 Self-supervised Training

In the training stage of the E2E ASR model, it is necessary to train a feature extractor to extract acoustic representations from waveforms. For this purpose, I adopt the recently proposed concept of wav2vec2.0. The original wav2vec2.0 is an elaborate self-supervised pre-training framework that can perform powerful feature extraction for speech recognition. Therefore, I believe that using this framework will make the training of the feature extractor consisting of convolutional layers that I use in this study more robust. It trains an ASR model through two steps: *pre-training* and *fine-tuning*. In the *pre-training* stage, the fully convolutional networks are used to extract the acoustic representations from raw waveforms, and then the acoustic representations are sent to the multi-layer transformer network to reconstruct the feature map. At the same time, the acoustic representations are processed using masking and product quantization to generate learning targets. Then, by minimizing the contrastive loss and diversity loss between the reconstructed feature map and the targets, the feature extractor can obtain the ability to represent acoustic information, and the encoder can understand this information. Throughout the entire *pre-training* stage, speech transcripts are not required. After *pre-training*, the parameters of the feature extractor are fixed and a projection layer is appended behind the Transformer encoder, and then the parameters of these two modules are fine-tuned by calculating the CTC loss with the speech transcripts. This step requires labeled data.

For the E2E ASR model to work on edge devices, the original wav2vec2.0 is difficult to deploy on embedded devices because it uses Transformer, which is not supported by many SDKs, but I can apply its training approach to other layer structures such as my proposed model. I train my E2E speech recognition model in a similar way to wav2vec2.0 in order to achieve higher accuracy with the small amount of speech data collected on the device, as well as to help training convergence in a model with as small a number of parameters as possible. In the feature extractor, I use fewer layers than the feature extraction module of wav2vec2.0, as well as replace the traditional two-dimensional convolutional layer with the customized MobileNetv3 block where one-dimensional convolutional layer is used, thus reducing the number of parameters. As mentioned in Section 5.4.1, because CNN is mainly selected, I have designed a completely different model structure from the transformer encoder of wav2vec2.0. Through these operations, my model structure has been greatly compressed.

Compared to the ASR system deployed in the cloud, the device-side ASR system is usually used for specific tasks. Therefore, developers pay more attention to maximize the accuracy in a specific domain under the limited model architecture than to its universality. However, the corpus of a specific domain is often very difficult to collect, for example due to commercial rights. With the help of wav2vec2.0, I can achieve high accuracy even with a

small amount of domain data through self-supervised training. This is exactly what I need. However, compared to wav2vec2.0, the model structure I propose is smaller, so the upper limit of the accuracy it can achieve is theoretically lower. Therefore, instead of directly using a small amount of domain data to fine-tune the model, I first use labeled public data for fine-tuning, and then use the domain data for transfer learning. In summary, differing from original self-supervised learning strategy of wav2vec2.0, I train the model with an extra step: *transfer-learning*. I first use large-scale unlabeled data to pre-train *Feature Extractor* and *Encoder* and fine-tune the *Encoder*. Then, a *transfer-learning* [94] process is performed using the fine-tuned model with a small amount of target domain speech data.

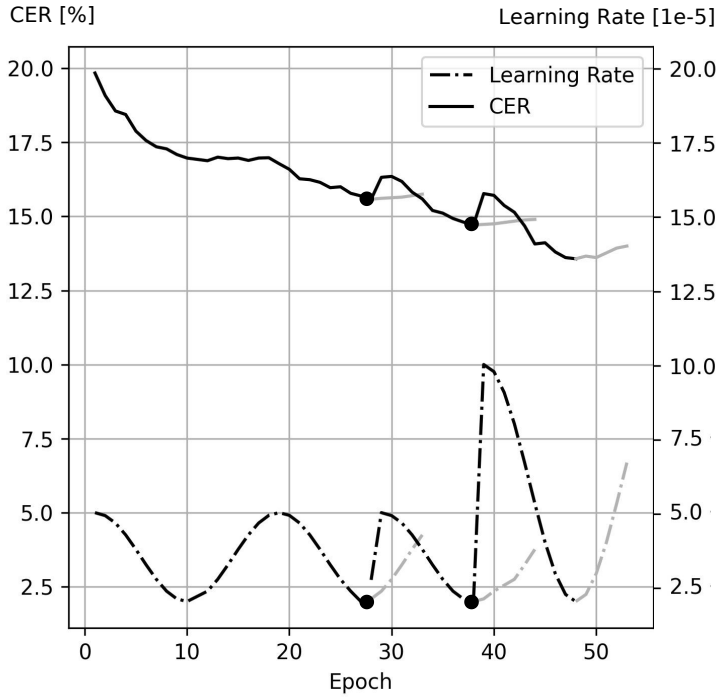


Figure 5.3: Changes in learning rate and CER after using the learning rate incentive strategy

When conducting the *fine-tuning* and *transfer-learning* steps, I change the learning rate in order to obtain the ASR model with a better character error rate (CER). I use a classic cosine annealing scheduler [56], but once the parameters of the model have converged, I find the best model parameter that has been saved in advance and restart the training cycle with a motivated initial learning rate. Because the cosine annealing scheduler constantly scales the learning rate during the training process, the initial learning rate can be used again. After the model converges, values of the learning rate that are less than the current initial learning rate are likely to no longer get better results. Therefore,

in the next cycle, I allow the learning rate to be magnified to a greater value than the current initial value. The mechanism used in my experiment is shown in Eqn.5.1. n is the number of cycles, and γ_0 is the initial learning rate of the first training loop.

$$\gamma_n = 2^{n-1} \times \gamma_0 \quad (n > 0) \tag{5.1}$$

Fig.5.3 shows the change in learning rate and CER after using this trick during the *fine-tuning* step in my experiment. I stimulated the learning rate to different initial values on two occasions (see the black dots), and each time, I got better results than before. Fig.5.3 indicates that the learning rate incentive method has a certain positive effect on model training.

In my experiment, I used an early stop strategy, which makes the training loop stop if the CER does not decrease in a certain time. This trick may lead to the early end of training before a new round of learning rate would be increased to the initial value. To avoid this situation, I amplified the learning rate to the same value as before and achieved a better result. Therefore, the early stop strategy may cause the model to miss the opportunity to get better in the new learning round.

5.4.3 Decoding with Prefix Beam Search

The WFST decoder is a popular algorithm in both traditional ASR systems and novel E2E systems. In my experiment, I used a token-lexicon-grammar (TLG) [67] decoding graph and obtained amazing accuracy. However, because it constructs from word piece LM, such a WFST decoding graph always has a large size. Therefore, I prefer the beam search algorithm and propose several improvements that could be made to the algorithm. A prefix beam search algorithm is widely used to decode CTC probability on OCR [88] and ASR because of its flexibility and excellent precision, and I use this algorithm in my system. In the process of merging prefix beams, I dynamically search words in the lexicon trie tree. However, differing from most beam search algorithms, I do the acoustic pruning after a prefix is decided. I list the lexicon-based pruning strategy in Algorithm 2.

I define an object *beam* to carry a tracker responsible for searching forward in the lexicon trie and finding the path whose chars can be combined into a complete word. Therefore, instead of using a common pruning candidate list containing all char ids, I use a respective list for each beam by searching the next destinations of its tracker. This small change can reserve as many search paths as possible under the same beam size and avoid pruning the correct word, especially when the probability of a path is not high enough.

Another improvement in my work is the initialism of LM. The ASR system on embedded devices generally builds a grammar graph to recognize limited commands. However, for large vocabulary continuous speech recognition (LVCSR), I need to leverage the power of statistical LMs. If a word piece LM is used, the score is accumulated when and only

Algorithm 2: Prefix beam search

Input: probability matrix M with T frames and D dimensions, blank index *bid*, blank probability threshold α, beam size β

Output: words

```
// beam is object whose properties contain at least last char id lastcid,
   probability of blank branch pb and non-bank branche pnb, and the tracker to a
   lexicon trie node ptr;
lastBeams = {};
put an initial beam into lastBeams;
for t to T do
    if M[t, bid] > α then
        skip frame t;
    end
    newBeams = {};
    for pbeam In lastBeams do
        topK = {};
        set variable cids to all char ids which ptr of pbeam can arrive to on trie;
        for cid In cids do
            put cid into topK;
        end
        put bid into topK;
        put lastcid of pbeam into topK if it is not appeared in topK;
        sort topK by the probability of each cid in topK at frame t, then keep top
          β candidates;
        for cid In topK do
            set variable nbeam to a new beam or an existed beam in newBeams
              according to what the cid and lastcid of beam are;
            update lastcid, pb, pnb and ptr of nbeam;
        end
    end
    sort newBeams by the sum of pb and pnb of each beam;
    clear lastBeams;
    select top β beams from newBeams and put them into lastBeams;
end
return words of best beam in lastBeams;
```

when a word is outputted. The right path may have been incorrectly pruned before the word was synthesized. LM using sub-word units is one solution, but it is difficult to capture the contextual relationship between words. I use two methods to address this

Table 5.1: Various language modeling units of an example phrase "Automatic Speech Recognition."

word	AUTOMATIC‖SPEECH‖RECOGNITION
sub-word	A‖U‖T‖O‖M‖A‖T‖I‖C‖ S‖P‖E‖E‖C‖H‖ R‖E‖C‖O‖G‖N‖I‖T‖I‖O‖N
PD sub-word	A_‖U‖T‖O‖M‖A‖T‖I‖C‖ S_‖P‖E‖E‖C‖H‖ R_‖E‖C‖O‖G‖N‖I‖T‖I‖O‖N
initialism	A‖S‖R

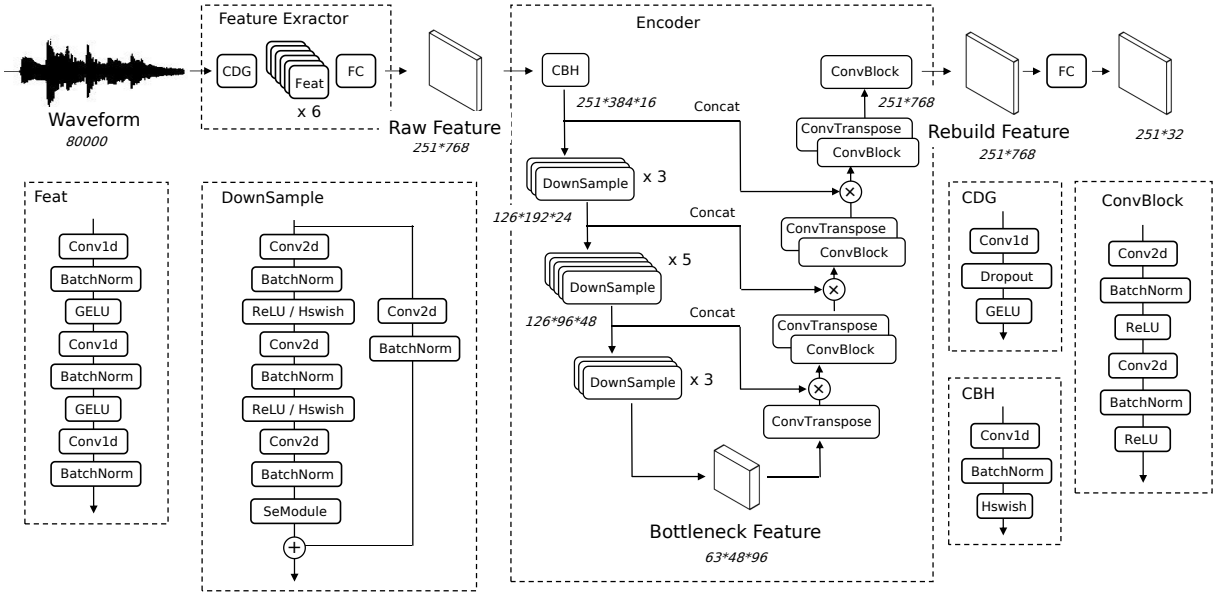


Figure 5.4: The *LW-extractor + LW-encoder* model architecture

problem, as shown in Table 5.1. First, I mark the position for the initial sub-word with a specific tail symbol "_" in order that the LM can recognize the beginning of a new word, so that the implicit contextual information between words can be inferred during decoding. I call this method the position dependent (PD) sub-word LM. The second is called initialism LM. Just as the initial abbreviation of a phrase can represent the semantics of the phrase, I propose a new language modeling approach: modeling for the initialism. Because only the initial sub-words are used, this LM can be very compact in size and easily added to the search path as well as the sub-word model. The final score of a search path can be calculated as follows:

$$P_{beam} = a \cdot P_{CTC} + b \cdot P_{cLM} + c \cdot P_{iLM} \tag{5.2}$$

where a, b, c are the weights of CTC probability P_{CTC}, sub-word LM score P_{cLM} and initialism LM score P_{iLM}, respectively. Eqn.5.3, 5.4 and 5.5 give the computing methods of these three probabilities. All probabilities are scaled to the sub-word level by length regularization. In these equations, P_b is the score of the blank path; P_{nb} is the score of the no-blank path; $P(cUNK)$ is the score of the unknown symbol of char LM; $p(iUNK)$ is the score of unknown symbol of initialism LM; N_{char} is the number of decoded chars; N_{word} is the number of decoded words; $P(S_c)$ is the score of n-gram char LM; and $P(S_i)$ is the score of n-gram initialism LM.

Parameter a can simply be set to 1.0, leaving only b and c to be control. These two values can be fine-tuned according to the decoding results. Generally, I can find the best values by parameter search. In my experiment, I used the search range of $b \in [0.05, 1.0]$, $c \in [0.05, 1.0]$.

$$P_{CTC} = (P_b + P_{nb})/(N_{frame}) \tag{5.3}$$

$$P_{cLM} = \begin{cases} P(cUNK) & \text{if } N_{char} = 0, \\ P(S_c)/(N_{char} + 1) & \text{if } N_{char} > 0. \end{cases} \tag{5.4}$$

$$P_{iLM} = \begin{cases} P(iUNK) & \text{if } N_{word} = 0, \\ P(S_i)/(N_{word} + 1) & \text{if } N_{word} > 0. \end{cases} \tag{5.5}$$

5.5 Implementation of the E2E ASR Model

Fig.5.4 illustrates the details of the implementation of my lightweight E2E ASR model. I name the feature extractor *LW-extractor* (lightweight feature extractor) and the encoder *LW-encoder* (lightweight encoder). In the *CDG* block in *Feature Extractor*, I use a large kernel-size CNN layer to filter out unimportant signals in the waveform. In the next *Feat block*, I use 3 layers of CNN with a small kernel size to refine the acoustic features. There are three skip connections in *Encoder*. The *DownSample* block is a duplicate of the MobileNetv3 basic block but with different arguments. In FPN architecture, the encoder yields features of multiple scales. In my model, I only select the outermost output, which maintains the same size as the input feature, so it is more convenient to calculate wav2vec2.0 loss at the *pre-training* stage. Furthermore, more semantic information is available at this output.

5.6 Experiments

5.6.1 Dataset

As mentioned above, I need to use a relatively large amount (more than 100 hours) of public data for pre-training and fine-tuning my model, and for experimental comparison

with previous work and other models. Then, I need to use a small amount (less than 100 hours) of domain data for transfer learning to simulate the deployment of my ASR system.

During the *pre-training* and *fine-tuning* training steps, I used the Librispeech speech dataset, which is an open-source corpus including 960 hours of training data of spoken English with a sampling rate of 16 kHz. In the *transfer-learning* step, I used a domain dataset containing about 32 hours of spoken English. This is a spoken English corpus collected for the purpose of developing an ASR system for police equipment. The speakers were adult North American police officers, and the recording environments were on the street and in the office, so the audio includes some street noise and office background noises. It was collected in the near field by Streamax Technology staff. The distance between the lips and the microphone was approximately 30 to 50 cm. The sampling rate was 16 kHz. This dataset consists of some control commands for the embedded equipment, such as:

Start recording

and some conversations during street patrols and rests in the office, such as:

Get out of the car and put your hands up now.

The Streamax dataset was split into two parts. The training dataset included 30 hours data (including 12,446 utterances). I used 2 hours data (including 844 utterances) named *Streamax* as the test dataset.

5.6.2 Evaluation Measures

Some evaluation measures were used to evaluate the performances of the ASR models in my experiments. I used CER to evaluate the accuracy of the ASR models, the number of parameters to evaluate the size of the model, and the memory occupation in the hard disk of model files to evaluate the size of the decoding graph. Finally, to evaluate the speech processing time, I used Inference Time and CPU occupancy to evaluate the performance of the ASR models when the models were deployed in an embedded device. Lower values of these factors indicate higher performance.

5.6.3 Details of Training Conditions

The *Adam* [95] optimizer (an improved gradient descent algorithm) was used throughout my experiment. In the *pre-training* step, I adopted an initial learning rate of 0.0005 and trained the model with 64 NVIDIA TITAN RTX GPUs. The learning rate scheduler is a polynomial decay policy. Next, to obtain good initial parameters to speed up the convergence, I did a speculative job. I first conducted the supervised CTC training with a configuration that was provided by the wav2vec2.0 framework: all 960 hours of the labeled

Librispeech training dataset, an initial learning rate 0.0001, and a tri-stage learning rate scheduler. This trick enabled us to get an initial model with a CER 20% on a *test-clean* dataset. Then, in the *fine-tuning* step, I used a single GPU with batch size 4. I cropped utterances whose duration was out of the range of 0.5 s to 30 s. As introduced before, I used an initial learning rate 0.00005, a cosine annealing scheduler, and the learning rate incentive policy mentioned in Eqn.5.1.

5.6.4 Evaluation of E2E Model

My model takes 1-dimensional waveform as input. Both *pre-training* and *fine-tuning* are trained on all 960 hours of training data of the Librispeech corpus. For better comparison, I first trained the wav2vec2.0-small model in my training environment. The final model parameter was 94.4 M, and the CER on the Librispeech test-clean dataset was 2.89%. Then, I trained all models in this experiment using the same greedy search decoder used with the reference model.

Let us now compare the customized models of ResNet-18 and MobileNetv3-small backbone, both classical models used on embedded devices. Both models use 128-dimensional fBank features with Int8 quantization. All models compute CTC error with letter level. I tried two feature extractors: a multiple-layer full convolutional feature extractor named *CNN-extractor*, which has the same structure as wav2vec2.0 but without group normalization and layer normalization, and my proposed feature extractor, *LW-extractor*. The FPN feature encoding module tried a multiple-layer residual CNN named *Res-encoder* and my *LW-encoder*. Table 5.2 summarizes the number of parameters and the CERs of various models using the greedy search decoder. In this experiment, compared to the *ResNet-18* model, the *MobileNetv3* model converged more easily and achieved better accuracy with only 0.54 M parameters. I have shown that the MobileNetv3 network using only the CNN backbone has excellent potential for deployment on mobile devices.

Then, I tried a combination of *CNN-extractor* and *Res-encoder* and achieved the CER 6.64% with 39.87 M parameters, which is acceptable compared with previous work. I then tested the pair *CNN-extractor* + *LW-encoder*, which reduced the number of parameters to 5.19 M while CER increased by 4.75%. Finally, the proposed *LW-extractor* + *LW-encoder* model achieved a relatively good performance that best met my expectations in my experiment, with a number of only 0.79 M parameters and a CER of 13.57%. Note that the parameter includes the entire model: feature extractor, encoder, and projector. Finally, I used the domain dataset to implement the *transfer-learning* step of my proposed model and obtained a CER 8.52% on the *Streamax* test dataset. Although I did not reach my goal of reducing the CER below 5% on the domain dataset, I were able to reduce the CER by 11.52% compared to the previous model. This shows that transfer training can significantly improve the ASR system's ability to recognize speech in a specific domain and ensure its accuracy after deployment. The next section describes experiments in

Table 5.2: Number of parameters [M] and CERs [%] of various models using greedy search

	params	**Librispeech test-clean**	**Streamax** (conversation)
wav2v2c2.0-small	94.4 M	2.89	–
ResNet-18	3.61 M	57.3	–
MobileNetv3	0.54 M	15.45	–
CNN-ext. + Res-enc.	39.87 M	6.64	–
CNN-ext. + LW-enc.	5.19 M	11.39	–
LW-ext. + LW-enc.	0.79 M	13.57	20.04
Transfer Learning	–	–	8.52

which the decoder was optimized to reduce the CER further.

5.6.5 Comparison of Decoding Methods

In this experiment, I trained language models using all transcripts of my domain training dataset. Table5.3 lists the memory size of the graph file in the hard disk and the CER on *Streamax* test dataset using the various decoders.

I first decoded using a WFST decoder. A letter-level T, a word-to-letters L, and a 3-gram word-level G compose the TLG decoding graph by a sequence of operations of a finite state transducer. The process is shown in Eq. 5.6.

$$TLG = T \circ min(det(L \circ G)) \tag{5.6}$$

The symbol $\circ$ represents the "compose" operation, *det* is "determine," and *min* is "minimize." The second line of table 5.3 shows . As far as accuracy is concerned, WFST decoder achieved good performance, with the best CER 3.99% in my experiments. However, as I had predicted, the graph size of 662 MB was too large to deploy on embedded devices.

I then evaluated the improved prefix beam search algorithm. Because the beam search decoder dynamically expands the search path, the lexicon and the original LM are the main parts of the decoding graph. Therefore, in this experiment, I have considered the file size of the N-Gram LM and word-to-letter lexicon. All LM were trained with the KenLM [96] toolkit and compressed to binary format. The lexicon used in my experiment contained $89,123$ words with a file size of 0.77 MB. I compared the results using 4-gram char-level LM and 4-gram PD char-level LM. Using PD LM achieves worse accuracy. This indicates that when lacking text data, the PD method is not enough to model the inter-word and between-words relationship at the same time. I then superimposed the prefix-dependent lexicon trie pruning trick on 4-gram char LM to further improve the accuracy; CER dropped by 0.11% when I did this. Finally, I added the 4-gram initialism

Table 5.3: Graph file size [MB] and CERs[%] on *Streamax* dataset using various decoders

	Graph size	**CER**
greedy search	–	8.52
WFST decoding	662 MB	3.99
prefix beam search		
+ 4-gram PD character LM	1.59 MB	7.79
+ 4-gram character LM	1.47 MB	5.13
+ lexicon trie pruning	1.47 + 0.77 MB	5.02
+ 4-gram initialism LM	1.47 + 0.77 + 0.57 MB	4.86
flashlight lexicon decoder		
+ 3-gram word LM	143.95 + 0.77 MB	10.76

LM. After combining the char-level LM, lexicon pruning, and initialism LM, I obtained the best result (CER 4.86%) of my experiment. Although it was 0.87% higher than the CER using the WFST decoder, the total size of the decoding graph was 2.81 MB and only four-thousandths of the former. Finally, I compared the lexicon decoder in the fairsq's library used in wav2vec2.0. I built a 3-gram word LM. Regardless of the file size of the LM or CER, my decoder performed better than it in this task.

5.6.6 Evaluation on Embedded Device

I also deployed my ASR system on an embedded device developed by Streamax[2] Technology Co., Ltd. This device is configured with a 4-core Cortex-A53 1 GHz CPU and 2 GB of RAM, an NPU with 1 TOPS of arithmetic power. As shown in Table 5.4, my model achieved significantly reduced CPU usage and inference time. Note that the test was conducted when only system applications were running on the device, and sufficient resources were available to ensure the running condition of my ASR algorithm. The model inputs 3 seconds of speech at a 16 KHz sampling rate. When using the ResNet-18 model, acoustic features are extracted from the speech using a window size of 32 ms and a slide of 10 ms to compute fBank, with a feature dimension of 128. The inference time in the following table includes feature computation time, model inference time, and decoding time. If the model running on the NPU, it uses int8 for parameter quantization. The improvement in CPU usage and inference time was due to the small size of the model and the delegation of all inference processing to the NPU, which allowed us to take full advantage of the hardware performance of the embedded device.

[2]www.streamax.com

Table 5.4: CPU usage and Inference Time when my ASR system ran on the embedded device

	CPU	**Inference Time**	
	w/o NPU	w/o NPU	w/ NPU
ResNet-18	25%	690 ms	72 ms
LW-ext. + LW-enc. (proposed)	16%	230 ms	14 ms

5.7 Conclusion

In this chapter, I presented a lightweight E2E ASR model that is easy to deploy for low-resource embedded devices.

The two main contributions of my model are as follows:

- the ASR model mainly uses convolution layers, which enables it to be supported by most SDKs,
- the ASR model size is relatively small and consumes low levels of resources while still guaranteeing good accuracy and RTF.

My model consists of three modules: the feature extractor, the encoder, and the projector. The feature extractor can extract acoustic representations from speech waveforms using multiple convolutional layers with a small kernel size. In the encoder, I adopt an FPN architecture to fuse hidden features to make up for the shortcomings of the CNN in capturing long-distance context information. In the training stage of the model, to achieve the best performance, I optimize the learning rate decay strategy to squeeze the convergence ability.

In the decoding stage, I propose an improved method for the prefix beam search algorithm: prefix-based lexicon trie pruning and the initialism LM. This allowed us to build a decoder with competitive accuracy using only a few memory resources. The proposed system has demonstrated that ASR technologies could be effectively implemented in more practical developments.

In this chapter, I present a lightweight E2E ASR model designed for efficient deployment on low-resource embedded devices. The model's two primary contributions are:

1. Utilization of convolution layers predominantly, facilitating compatibility with most SDKs.
2. A compact model size that requires minimal resources, while maintaining high accuracy and a favorable real-time performance.

The model comprises three modules: the feature extractor, the encoder, and the projector. The feature extractor employs multiple convolutional layers with small kernel sizes to extract acoustic representations from speech waveforms. In the encoder, I incorporate a FPN module to amalgamate hidden features, compensating for the limitations of CNNs in capturing long-range contextual information. During the model's training phase, I refine the learning rate decay strategy to enhance convergence.

For the decoding stage, I introduce an enhanced version of the prefix beam search algorithm, incorporating prefix-based lexicon trie pruning and an initialism-based Language Model. This approach enables the construction of a decoder that achieves competitive accuracy with minimal memory usage.

In our experiment, I demonstrated the effectiveness of this study. The proposed system demonstrates the effective implementation of ASR technologies in practical applications.

Chapter 6

Voice Activity Detection using Convolution Neural Network

In Chapter 5, I introduced a lightweight End-to-End model and an improved ASR decoder deployed on edge-side. This chapter will discuss some of my exploratory work on the voice activity detection (VAD) function. I will first introduce the role of VAD and some related work, then move on to discuss my experiments and conclusions.

6.1 Introduction

In real-time ASR systems, VAD plays a crucial role. When the device-side ASR system, which also includes real-time voice wake-up system and keyword spotting system, is in operation, it needs to continuously listen to external sounds. However, most of these sounds are silence or environmental noise. This part of the sound signal, being of no value, will continue to participate in the subsequent process. If it is necessary to use a cloud-side ASR model to process this voice, then these valueless data will be transmitted to cloud servers, causing a significant waste of data traffic. For example, one second of sound with a sampling rate of 16 KHz and a quantization width of 16 bits will result in approximately 32 KB of data traffic loss without any compression algorithms. Moreover, directly processing this silence and noise sound by either cloud-side or device-side ASR systems will also result in a waste of computational resources. To deal with this dilemma, the VAD algorithms, by detecting the start and end points of speech intervals and retaining only the sound segments with speech while discarding silence or noise-only signals, significantly reduce the resource consumption of ASR systems. On the other hand, if VAD is used, the DNN model and decoder of the ASR system can behave differently. For example, once the start and end points of a speech interval are determined, real-time speech recognition can be converted into an offline speech recognition approach, as shown in Fig.6.1. Typically, offline ASR systems have better accuracy because at any moment of the speech, the

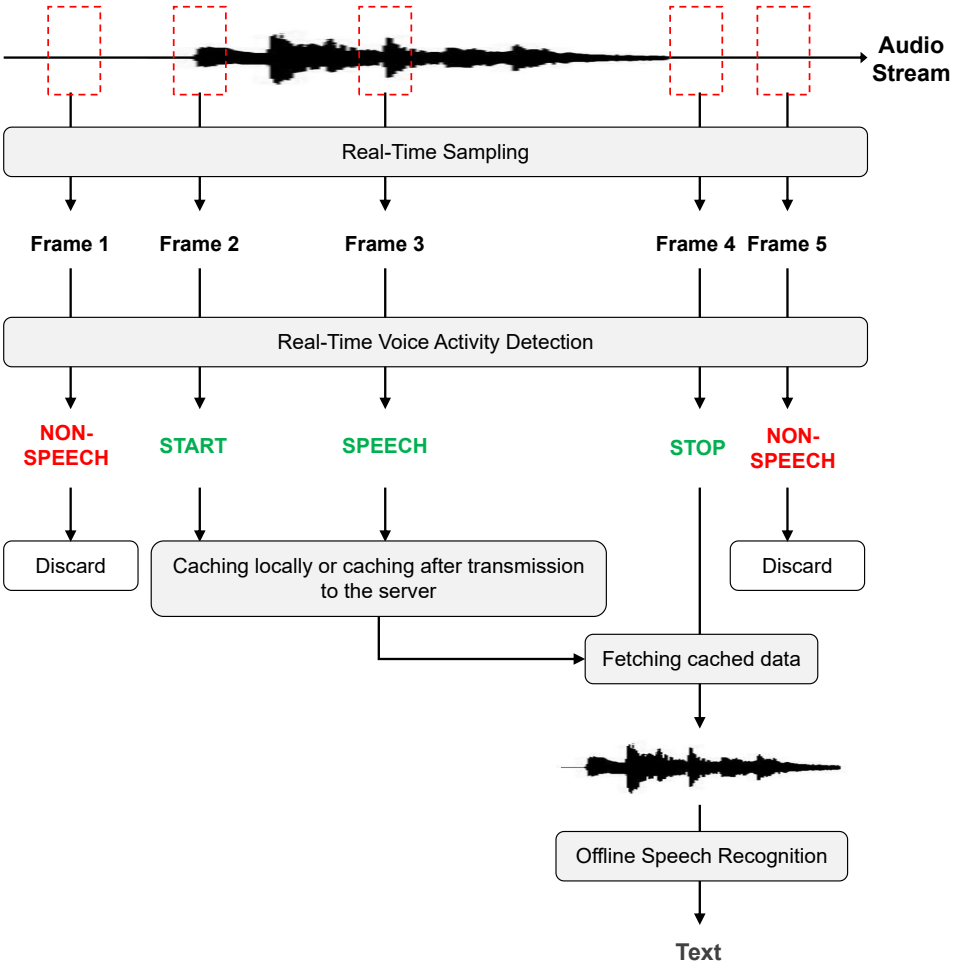


Figure 6.1: Using VAD to convert real-time recognition to offline recognition

inference can see the global context information.

WebrtcVAD [97] is one of the most popular VAD tools. It is a statistical method based on the Gaussian model, known for its fast speed, low latency, and easy deployment. However, WebrtcVAD's robustness in abnormal noise environments is poor. With the rise of deep learning in the field of speech, some previous studies have used DNN models to detect voice activity [98, 99]. These works have demonstrated that, compared to traditional statistical methods, VAD methods based on DNNs have better robustness in noisy environments, thanks to the neural network model's stronger capability to fit diverse noises. At the same time, using raw waveforms directly for speech detection yields better results than manually designed acoustic features, such as fBank and MFCC, partly because raw waveforms retain more distinctive features of noise. Although VAD methods based on DNN models have achieved good accuracy and robustness in these tasks, compared to statistical methods, neural networks generally have a larger resource overhead. Moreover, most related works use model structures like RNNs, which, as I described in

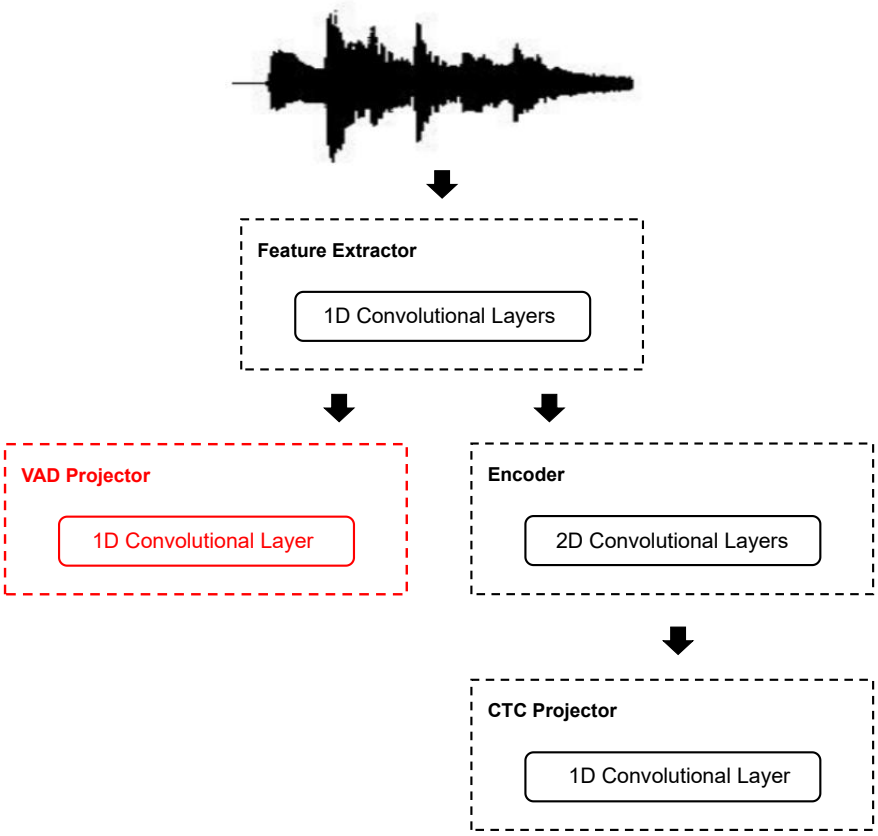


Figure 6.2: Multitasking network modeling for voice activity detection and speech recognition

Chapter 5, are not very compatible with edge devices. Therefore, a VAD model that is accurate, robust to environmental noise, lightweight, and deployable is in demand.

In this chapter, I explore the construction methods of a lightweight VAD model using convolutional neural networks. In Chapter 5, I proposed a lightweight ASR model containing a feature extractor that processes waveforms into acoustic features, consisting only of tiny one-dimensional convolutional structures. Therefore, As shown in Fig.6.2, I directly added a classification layer after this feature extractor to predict whether a speech frame is noise or background, forming a multi-task network for activity detection and speech recognition. In this model, the VAD projector uses ASR model's feature extractor directly. Although this operation coupled the VAD model with the ASR model to some extent, it brought only a small increase in parameters to the entire speech recognition system. Additionally, by incorporating noise augmentation to train the VAD projector, the robustness of this voice detection in noisy environments was further enhanced.

6.2 Experiments on Weight Sharing Voice Activity Detection Model

6.2.1 Models

To achieve weight sharing with the ASR model, I directly used the same structure for the feature extractor as the one in Chapter 5, which contains 19 layers of one-dimensional convolutional layers. A layer of one-dimensional convolution was also used in the *VAD Projector*. Therefore, I named this model *Lightweight-20*. In the experiments, I first compared the following schemes without using the weights of the ASR model's feature extractor:

1. Lightweight-20-scratch: Directly train the feature extractor and VAD projector from scratch.

2. Lightweight-11-scratch: Reduce the number of convolutional layers in the feature extractor to 10, then train from scratch.

3. Lightweight-20-finetune: Load the weights of the ASR feature extractor as pre-trained weights, and then fine-tune the entire model.

In the above models, since VAD does not use the weights of ASR, VAD and ASR are two independent task models, and the additional parameters of the speech system include the entire VAD model's feature extractor and projector. I then compared the scheme of sharing the weights of the ASR feature extractor:

1. Lightweight-20-freeze: Load the weights of the ASR feature extractor, freeze the feature extractor, and only train the VAD projector.

2. Lightweight-20-noise: Based on Lightweight-20-freeze, use noise augmentation to train the VAD projector.

In these shared-weight models, VAD and ASR constitute a multi-task model, where the only additional parameters in the speech system comes from the weights of the VAD projector.

6.2.2 Datasets

I used Librispeech as the training dataset, consistent with my training of ASR model of Chapter 5. In a speech utterance of Librispeech, the annotation in only provides the sequence of words without giving the exact time each word was spoken. Therefore, to get the speaking time for each word, I used an automatic labeling method ad shown in Fig.6.3. I first used a state-of-the-art (SOTA) offline End-to-End ASR model, wav2vec2.0,

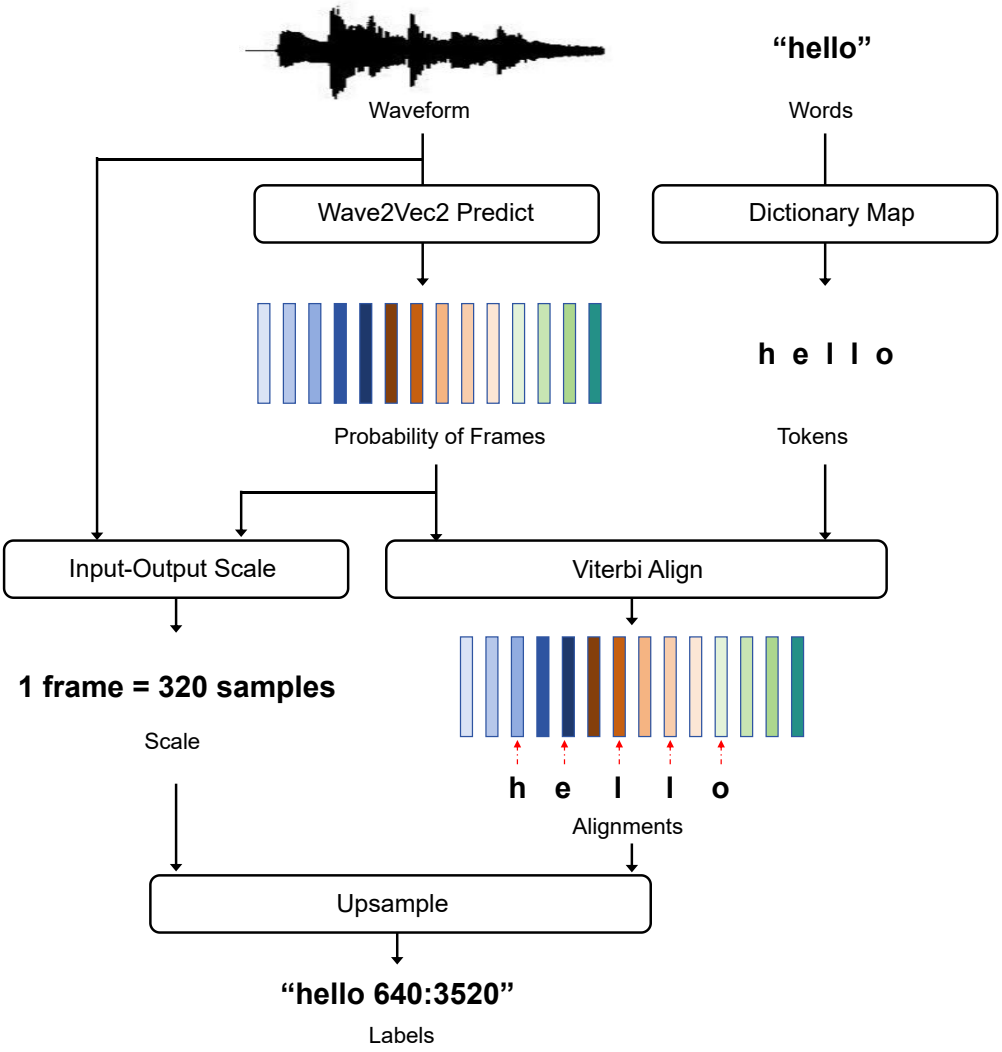


Figure 6.3: Automatic labeling process for training data

to calculate the probabilities for each speech utterance. Since the End-to-End model of wav2vec2.0 is trained with the CTC loss function, which uses the forward-backward algorithm for error calculation, it allows us to use the Viterbi algorithm to align the probabilities with the sequence of words. Through alignment, I obtained the speaking time for each word. The process of this alignment operation is shown in Figure A. Because ASR models are not absolutely accurate, the final alignment information contains a certain degree of error. The word error rate of the wav2vec2.0 model on the Librispeech test set is below 5%, so I can infer that the confidence level of the pseudo labels in the training set constructed by my method is above 95%, which is sufficient to help me train a VAD model with good accuracy. For testing, I used the entire TIMIT [100] dataset as the test set because TIMIT provides alignment information at both the phoneme and word levels for each sentence.

Table 6.1: The performance of various CNN-based VAD models

	Params	**Accuracy**	**Time/200ms**
WebrtcVAD	–	0.8145	< 1 ms
ResNet-18*	1072.26 K	0.8613	13 ms
MobileNetV3-S*	493.12 K	0.8811	25 ms
Lightweight-20	116.74 K	0.8564	7 ms
Lightweight-11	59.62 K	0.8226	4 ms
Lightweight-20-finetune	116.74 K	0.8418	7 ms
Lightweight-20-freeze	116.74/0.19 K	0.8465	7 ms
Lightweight-20-noise	116.74/0.19 K	0.8516	7 ms

After obtaining the alignment for the training and test sets, the alignment information can be converted into labels usable for deep learning based on the scaling factor of the input-output length ratio of the neural network model. These labels are used to calculate loss and accuracy with the network's output. The model outputs two categories: speech and non-speech. Considering the imbalance in the training data, especially the insufficient number of noise samples, I used focal loss.

During the noise experiment, I used the Noise-92 [79] noise dataset. This library contains 15 types of noise, including white noise and factory noise. I evenly split each type of noise, using half for training and the other half for testing. During training, noise was randomly added to the LibriSpeech data (with a 0.5 probability of adding or not adding noise, randomly selecting a type of noise, and a random signal-to-noise (SNR) ratio within the range of 0 dB to 40 dB). For testing, noise was added to all test data in a fixed order and SNR to ensure consistency across multiple tests.

6.2.3 Comparison of VAD methods

I first evaluated the number of parameters of different models, their F1-scores on the test dataset, and the inference time for every two hundred milliseconds of speech. The experimental environment is a Linux PC with Core i5-10400F CPU and NVIDIA GeForce GTX 1650 GPU. As shown in Table 6.1, models based on DNNs demonstrated higher accuracy than WebrtcVAD. Our test dataset included multiple speakers and scenarios, showing that DNN models have a better ability to fit diverse data distribution, but at the cost of reduced inference speed.

In the Table6.1, *ResNet-18** and *MobileNetV3-S** are models which have the same structure as the classic ResNet-18 and MobileNetV3 respectively, but use smaller number of channels. In our experiments, the MobileNetV3-S* model achieved the highest accuracy, mainly due to the positive effects of operations such as channel attention, but these operations also resulted in the slowest inference speed. My model had significant advan-

tages in both numbers of parameter and inference speed compared to MobileNetV3-S*. Without sharing the feature extractor weights with the ASR model, the Lightweight-20-scratch model achieved an accuracy of 0.8564/0.8811 = 97.2% of the MobileNet-S model, but with only 116.74/493.12 = 23.7% of its parameter size and 25/7 = 3.6 times of its inference speed. If the feature extractor weights were shared with the ASR system, after noise-augmented training, our model's accuracy was almost identical to that when not sharing weights, but at this point, the entire speech recognition system only added 0.19 K parameters for the VAD projector, which is far less than the 116.74 K parameters added when not sharing weights.

6.2.4 Robustness on Noisy Environment

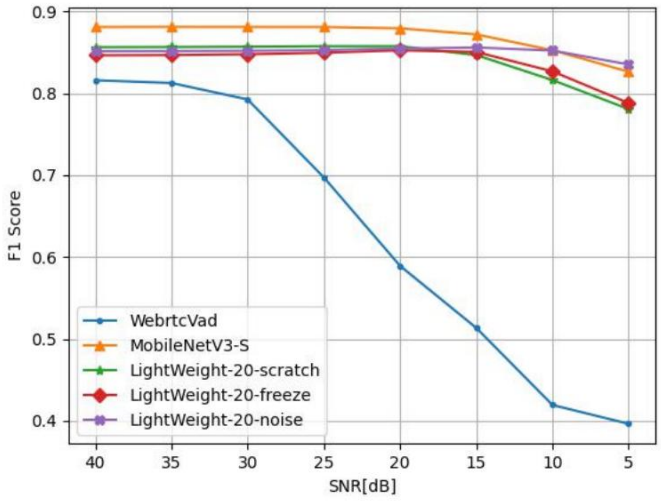


Figure 6.4: Robustness of each VAD model under noise conditions

I evaluated the robustness of VAD model under different noise intensity conditions. In Fig.6.4, the horizontal axis represents the SNR, where a lower value indicates stronger noise. When the SNR is around 40 dB, the environmental noise is minimal, and at this point, the accuracy of various models is similar to the results presented in Table6.1. However, as the noise intensity gradually increases, the accuracy of WebrtcVAD sharply declines, becoming almost non-functional. In contrast, all DNN-based VAD models demonstrated strong robustness. Even in very strong noise conditions (with an SNR of only 5 dB), the proposed Lightweight-20-noise model still maintained a high accuracy, surpassing all other models, thanks to noise augmentation training.

6.3 Conclusion

In this chapter, I explored the construction of a lightweight, noise-robust VAD model that exclusively utilizes a CNN structure.

In the ASR model proposed in Chapter5, I added a VAD projector after the feature extractor of the ASR model for detecting speech sounds. At this point, the VAD model and the ASR model form a multi-task network. Although this approach couples the VAD model with the ASR model, reducing its flexibility, this VAD model only introduced a very small increase in the number of parameters. In our experiments, through noise augmentation training, we achieved a VAD model with optimal overall performance: although its accuracy was only 97% of the best-performing model, MobileNetV3-S*, it only added 0.19 K parameters to the entire ASR system and offered a faster inference speed. Moreover, in strong noise environments, my model also demonstrated the best robustness compared to WebrtcVAD and several other DNN models mentioned in the experiments.

In future work, we hope to further evaluate the differences between my VAD model and other VAD models from previous studies, as well as their performance when deployed on edge devices.

Chapter 7

Summary

7.1 The Contributions of This Study

In this thesis, I present the benefits of ASR systems for edge devices, their current state of development, and the challenges they encounter. Typically, ASR systems deploy highly accurate DNN models and ASR decoders in cloud servers. However, in scenarios where network connectivity is unavailable or user data privacy must be safeguarded, running ASR on the edge device becomes essential. Consequently, the traditional device-side ASR system has evolved into a dual structure: the cloud-side ASR system and the device-side ASR system. When network access is available or for complex speech tasks such as chatbots and online navigation, sound signals are uploaded to and processed by the cloud-side ASR system. Conversely, for scenarios where data uploading is not feasible or for simpler speech tasks like voice wake-up, the device-side ASR system is activated.

The challenges faced by speech recognition systems include:

1. To improve accuracy, new models are constantly proposed, yet integrating these models into the streaming pipeline is challenging;
2. Edge devices often encounter variable recording distances and environmental noises;
3. These devices typically have limited computing power and memory;
4. Collecting labeled data poses significant difficulties;
5. The architecture of models that can be supported on edge devices is restricted.

Therefore, the objective of this thesis is to develop a system:

1. That exhibits high robustness to environmental noises;
2. That maintains high accuracy even with a limited amount of data;
3. That can be easily deployed both on cloud servers and edge devices.

In Chapters 4 and 5, I propose our solutions for the cloud-based ASR system and the device-side ASR system, respectively.

An ASR system comprises a DNN model and an ASR decoder. Kaldi, one of the most popular ASR toolkits, offers integrated functions for building DNN models and decoders. However, since Kaldi is developed in C++, it poses challenges in debugging model structures and training strategies. Conversely, the flexibility of the Python language has led to its widespread use in deep learning frameworks like PyTorch and TensorFlow, which have fostered a proliferation of advanced neural network models. Consequently, ASR researchers are keen on finding convenient methods to integrate DNN models trained with Python-based deep learning frameworks into decoders built using Kaldi's C++ framework.

In Chapter 4, I described the ExKaldi-RT toolkit, a novel initiative aimed at developing an online ASR system for cloud servers. ExKaldi-RT serves as a wrapper for the Kaldi toolkit and is characterized by its distinctive features:

1. It is developed in Python, facilitating the easy integration of DNN models from various deep learning frameworks.

2. It offers comprehensive tools for constructing a real-time ASR pipeline. This includes capabilities such as microphone recording, network transmission, feature extraction, DNN prediction, decoding, and more.

3. It allows for the flexible fusion of multiple acoustic features, enhancing the accuracy of the ASR system.

4. It supports the installation of denoising DNN models, thereby improving the robustness of the ASR system in diverse environments.

ExKaldi-RT has demonstrated the capability to achieve state-of-the-art results by leveraging advanced ASR technologies. In my experiments, the ASR system developed using the ExKaldi-RT toolkit exhibited competitive accuracy and real-time performance. Furthermore, I demonstrated that the integration of fusion features and denoising models through ExKaldi-RT significantly improves the accuracy and robustness of ASR systems, particularly in noisy environments.

On the device side, edge devices are typically limited in computational power and memory. Hence, the device-side ASR system focus is on devising models suitable for these constraints and compressing decoder sizes. Moreover, many edge device manufacturers offer their SDKs. Recently, as AI applications have become more common in edge devices, these manufacturers have provided custom inference frameworks to optimize chip performance. Yet, many advanced, high-precision DNN models are not compatible with these frameworks. Additionally, cloud-side ASR decoders often require substantial computational memory. For instance, a decoder in my experiments in Chapter 5, built using

just 5 hours of speech data, reached a size of 530 MB. These limitations in model structure and decoder size hinder the deployment of traditional ASR systems on edge devices.

In Chapter 5, I presented a lightweight E2E ASR model, designed for easy deployment on low-resource embedded devices. The two primary contributions of our model are:

1. The ASR model primarily utilizes convolutional layers, enhancing its compatibility with most SDKs for edge devices.

2. The model is compact in size, ensuring low resource usage while maintaining high accuracy and favorable real-time performance.

Our model comprises three modules: the feature extractor, the encoder, and the projector. The feature extractor can extract acoustic representations from speech waveforms using multiple convolutional layers with small kernel sizes. In the encoder, I adopt a feature FPN architecture to fuse hidden features, compensating for the CNN's limitations in capturing long-distance contextual information. During the training stage, to achieve optimal performance, I fine-tune the learning rate decay strategy to enhance the model's convergence capability. In the decoding stage, I propose an improved approach to the prefix beam search algorithm, which includes prefix-based lexicon trie pruning and the initialism language model. This allows us to construct a decoder with competitive accuracy while using minimal memory resources. In our experiment, I demonstrated the effectiveness of this study. The proposed system shows the effective implementation of ASR technologies in practical applications.

In Chapter 6, As an extension of the ASR system on the device side, I explored the construction of a lightweight, noise-robust VAD model that exclusively utilizes a CNN structure. I added a VAD projector after the feature extractor of the End-to-End ASR model proposed in 5 for detecting speech sounds. At this point, the VAD model and the ASR model form a multi-task network. Although this approach couples the VAD model with the ASR model, reducing its flexibility, this VAD model only introduced a very small increase in the number of parameters. In our experiments, through noise augmentation training, we achieved a VAD model with optimal overall performance: although its accuracy was only 97% of the best-performing model, MobileNetV3-S*, it only added 0.19 K parameters to the entire ASR system and offered a faster inference speed. Moreover, in strong noise environments, my VAD model also demonstrated the best robustness compared to WebrtcVAD and several other DNN models mentioned in the experiments.

In recent years, ASR systems have begun to be deployed on an increasing number of edge devices. Today, some high-end edge devices, such as smartphones and micro-server, can have CPU clock speeds up to 3.2 GHz, RAM up to 16 GB, and NPU computational power up to 15 TOPS. However, some low-end edge devices, such as surveillance cameras, smart home appliance controllers, vehicle interactive terminals, voice-operated controllers for factories, voice command-triggered mini recorders, and multilingual real-time translation pens, might have CPU clock speeds of less than 1 GHz, less than 2 GB of RAM, and

NPU computational power of less than 2 TOPS. Nonetheless, there is a demand to deploy AI functions, including speech recognition systems, on these low-end edge devices. In this study, I propose a real-time ASR system that is divided into cloud-side and edge-side systems. The cloud-side system has fewer restrictions on the model's structure, computation, and memory, allowing for the use of more accurate speech recognition models. However, in some cases, such as when the network is unavailable, the edge-side system must be used for speech recognition. The lightweight model I propose for deployment on edge-side systems uses common CNN convolutional structures, with model parameters of less than 1 M and the entire system occupying less than 10 MB of memory. This ensures that my model can be more easily deployed on these low-end devices and run efficiently. Additionally, my model can achieve a error rate of less than 5% on specific tasks with small amounts of domain data, making it capable of handling common speech recognition tasks on these low-end devices, such as voice wake-up, speech command controlling, and keyword spotting.

7.2 Future Work

In future work, the exploration of methods for deploying high-precision models to edge devices will be intensified. Planned optimization strategies for the lightweight E2E ASR model include a combination of the newly developed *LW-extractor* and a wav2vec2.0-based transformer encoder to enhance the pre-training of the feature extractor. Post-training, the *LW-extractor* can be coupled with the proposed *LW-encoder* for efficient encoder pre-training, potentially improving feature extractor performance. Additionally, employing the pre-trained feature extractor and a transformer encoder from wav2vec2.0 as teacher models in knowledge distillation [50] could leverage the advantages of a self-supervised training strategy, enhancing state-of-the-art performance. Additionally, I have continued my exploratory work on VAD, such as evaluating the differences between it and the SOTA VAD model and testing its performance after deployment.

References

[1] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 2015.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29:82–97, 2012.

[3] Pengcheng Guo, Florian Boyer, Xuankai Chang, Tomoki Hayashi, Yosuke Higuchi, Hirofumi Inaguma, Naoyuki Kamo, Chenda Li, Daniel Garcia-Romero, Jiatong Shi, Jing Shi, Shinji Watanabe, Kun Wei, Wangyou Zhang, and Yuekai Zhang. Recent developments on ESPnet toolkit boosted by conformer. In *Proceedings of the the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5874–5878, 2021.

[4] Suyoun Kim, Takaaki Hori, and Shinji Watanabe. Joint CTC-attention based end-to-end speech recognition using multi-task learning. In *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4835–4839, 2017.

[5] Dario Amodei and et al. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pages 173–182, 2016.

[6] Binbin Zhang, Di Wu, Zhendong Peng, Xingchen Song, Zhuoyuan Yao, Hang Lv, Lei Xie, Chao Yang, Fuping Pan, and Jianwei Niu. WeNet 2.0: More Productive End-to-End Speech Recognition Toolkit. In *Proceedings of INTERSPEECH 2022*, pages 1661–1665, 2022.

[7] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of INTERSPEECH 2010*, volume 2, pages 1045–1048, 2010.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st Advances in Neural Information Processing Systems (NeurIPS 2017)*, pages 1–11, 2017.

[9] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[10] Kikuo Maekawa. Corpus of spontaneous japanese: Its design and evaluation. *Proceedings of Spontaneous Speech Processing and Recognition (SSPR)*, 01 2003.

[11] Binbin Zhang, Hang Lv, Pengcheng Guo, Qijie Shao, Chao Yang, Lei Xie, Xin Xu, Hui Bu, Xiaoyu Chen, Chenchen Zeng, Di Wu, and Zhendong Peng. WENETSPEECH: A 10000+ hours multi-domain mandarin corpus for speech recognition. In *Proceedings of the 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6182–6186, 2022.

[12] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Vesel. The kaldi speech recognition toolkit. *Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8024–8035, 2019.

[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[15] Dogan Can, Victor R. Martinez, Pavlos Papadopoulos, and Shrikanth S. Narayanan. Pykaldi: A python wrapper for kaldi. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[16] Liang Lu, Xiong Xiao, Zhuo Chen, and Yifan Gong. Pykaldi2: Yet another speech toolkit based on kaldi and pytorch. *ArXiv*, abs/1907.05955, 2019.

[17] M. Ravanelli, T. Parcollet, and Y. Bengio. The pytorch-kaldi speech recognition toolkit. In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[18] Yu Wang, Chee Siang Leow, Hiromitsu Nishizaki, Akio Kobayashi, and Takehito Utsuro. Exkaldi: A python-based extension tool of kaldi. In *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pages 929–932, 2020.

[19] Yuekai Zhang, Sining Sun, and Long Ma. Tiny Transducer: A highly-efficient speech recognition model on edge devices. In *Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6024–6028, 2021.

[20] Yanzhang He and et al. Streaming end-to-end speech recognition for mobile devices. In *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6381–6385, 2019.

[21] Tara N. Sainath and et al. A streaming on-device end-to-end model surpassing server-side conventional model quality and latency. In *Proceeedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6059–6063, 2020.

[22] Marius-Constantin Popescu, Valentina Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *Wseas Transactions on Circuits and Systems*, 8, 07 2009.

[23] Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *ArXiv*, abs/1711.04735, 2017.

[24] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[25] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[27] Wadii Boulila, Maha Driss, Eman Alshanqiti, Mohamed Al-Sarem, Faisal Saeed, and Moez Krichen. Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives. In *Advances on Smart and Soft Computing*, pages 477–484, 2022.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, arXiv:1502.03167, 2015.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[30] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[31] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29:141–142, 2012.

[32] 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 (CVPR)*, pages 248–255, 2009.

[33] Anirudha Ghosh, A. Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. 2020.

[34] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016.

[35] Wilhelm Burger and Mark J. Burge. *Scale-Invariant Feature Transform (SIFT)*, pages 609–664. 2016.

[36] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017.

[37] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, pages 1–15, 2022.

[38] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO Series in 2021. *arXiv preprint arXiv:2107.08430*, pages 1–7, 2021.

[39] Shi Baoguang, Bai Xiang, and Yao Cong. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *ArXiv*, arXiv:1507.05717, 2015.

[40] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649, 2013.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[42] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate. *ArXiv*, abs/2006.16362, 2020.

[43] Zhuofan Xia, Xuran Pan, Shiji Song, Li Erran Li, and Gao Huang. Vision transformer with deformable attention. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4784–4793, 2022.

[44] J. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[45] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[46] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[47] Alexei Baevski, Steffen Schneider, and Michael Auli. vq-wav2vec: Self-supervised learning of discrete speech representations. In *Proceedings of International Conference on Learning Representations (ICLR)*, pages 1–12, 2020.

[48] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised Pre-Training for Speech Recognition. In *Proceedings of INTERSPEECH 2019*, pages 3465–3469, 2019.

[49] A. Baevski, H. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, pages 1–12, 2020.

[50] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, 2019.

[52] Gokul Yenduri, Manju Ramalingam, G. ChemmalarSelvi, Y Supriya, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, G DeeptiRaj, Rutvij H. Jhaveri, B. Prabadevi, Weizheng Wang, Athanasios V. Vasilakos, and Thippa Reddy Gadekallu. Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *ArXiv*, abs/2305.10435, 2023.

[53] Maxime Oquab, Timoth'ee Darcet, Théo Moutakanni, Huy Q. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russ Howes, Po-Yao (Bernie) Huang, Shang-Wen Li, Ishan Misra, Michael G. Rabbat, Vasu Sharma, Gabriel Synnaeve, Huijiao Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *ArXiv*, abs/2304.07193, 2023.

[54] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[55] Yimin Ding. The impact of learning rate decay and periodical learning rate restart on artificial neural network. pages 6–14, 01 2021.

[56] Zhao Liu. Super convergence cosine annealing with warm-up learning rate. In *CAIBDA 2022; 2nd International Conference on Artificial Intelligence, Big Data and Algorithms*, pages 1–7, 2022.

[57] Richard Mushi and Yo-Ping Huang. Assessment of mel-filter bank features on sound classifications using deep convolutional neural network. pages 334–339, 08 2021.

[58] Siwat Suksri and Thaweesak Yingthawornsuk. Speech recognition using mfcc. 2012.

[59] Dan Su, Xihong Wu, and Lei Xu. Gmm-hmm acoustic model training by a two level procedure with gaussian components determined by automatic model selection. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4890–4893, 2010.

[60] T Hori and A Nakamura. *Speech Recognition Algorithms Using Weighted Finite-State Transducers*. Springer Cham, 2013.

[61] SystemsS, Jatin, YoungN, H., Russellj, S H., and ThorntonCambridge. Token passing : a simple conceptual model for connectedspeech recognition. 1989.

[62] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Proceedings of INTERSPEECH 2015*, 2015.

[63] S. Selva Birunda and R. Kanniga Devi. A review on word embedding techniques for text classification. In *Innovative Data Communication Technologies and Application*, pages 267–281, 2021.

[64] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 369–376, 2006.

[65] Mohammadreza Ghodsi, Xiaofeng Liu, James Apfel, Rodrigo Cabrera, and Eugene Weinstein. Rnn-transducer with stateless prediction network. In *Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7049–7053, 2020.

[66] JQian Zhang, Han Lu, Hasim Sak, Anshuman Tripathi, Erik McDermott, Stephen Koo, and Shankar Kumar. Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss. *arXiv preprint arXiv:2002.02562*, 2020.

[67] Yajie Miao, Mohammad Gowayyed, and Florian Metze. EESEN: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *Proceedings of the 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 167–174, 2015.

[68] Harald Scheidl, Stefan Fiel, and Robert Sablatnig. Word beam search: A connectionist temporal classification decoding algorithm. In *Proceedings of the 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 253–258, 2018.

[69] Myle Ott and et al. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 48–53, 2019.

[70] V.Z. Këpuska and T.B. Klein. A novel wake-up-word speech recognition system, wake-up-word recognition task, technology and evaluation. *Nonlinear Analysis: Theory, Methods and Applications*, 71(12):2772–2789, 2009.

[71] Shilpa Sharma, Punam Rattan, and Anurag Sharma. Recent developments, challenges, and future scope of voice activity detection schemes—a review. In *Infor-*

mation and Communication Technology for Competitive Strategies (ICTCS 2020), pages 457–464, 2021.

[72] X. Liu, Y. Wang, X. Chen, M. J. F. Gales, and P. C. Woodland. Efficient lattice rescoring using recurrent neural network language models. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4908–4912, 2014.

[73] Meiko Fukuda, Ryota Nishimura, Hiromitsu Nishizaki, Koharu Horii, Yurie Iribe, Kazumasa Yamamoto, and Norihide Kitaoka. A new speech corpus of super-elderly japanese for acoustic modeling. *Computer Speech & Language*, 77:101424, 2023.

[74] Chee Siang Leow, Tomoaki Hayakawa, Hiromitsu Nishizaki, and Norihide Kitaoka. Development of a low-latency and real-time automatic speech recognition system. In *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, pages 925–928, 2020.

[75] onnx. Open neural network exchange. *https://github.com/onnx/onnx*, Accessed 10 Feb 2024.

[76] N. Vishnu Prasad and S. Umesh. Improved cepstral mean and variance normalization using bayesian framework. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 156–161, 2013.

[77] N.K. Goel and R.A. Gopinath. Multiple linear transforms. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 481–484 vol.1, 2001.

[78] Karel Veselý, Arnab Ghoshal, Lukáš Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Proceedings of INTERSPEECH 2013*, pages 2345–2349, 2013.

[79] Andrew Varga and Herman J. M. Steeneken. Assessment for automatic speech recognition: Ii. noisex-92: A database and an experiment to study the effect of additive noise on speech recognition systems. *Speech Commun.*, 12:247–251, 1993.

[80] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *ArXiv*, arXiv:1806.03185, 2018.

[81] Jinhwan Park, Yoonho Boo, Iksoo Choi, Sungho Shin, and Wonyong Sung. Fully neural network based speech recognition on mobile and embedded devices. In *Proceedings of the 32nd Advances in Neural Information Processing Systems (NeurIPS 2018)*, pages 1–11, 2018.

[82] Shiliang Zhang, Ming Lei, Zhijie Yan, and Lirong Dai. Deep-FSMN for large vocabulary continuous speech recognition. In *Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5869–5873, 2018.

[83] Ding Zhao, Tara N. Sainath, David Rybach, Pat Rondon, Deepti Bhatia, Bo Li, and Ruoming Pang. Shallow-Fusion End-to-End Contextual Biasing. In *Proceedings of INTERSPEECH 2019*, pages 1418–1422, 2019.

[84] Khe Chai Sim and et al. Personalization of end-to-end speech recognition on mobile devices for named entities. In *Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 23–30, 2019.

[85] NVIDIA. Tensorrt open source software. `https://github.com/NVIDIA/TensorRT`, referred on 4th/12/2022.

[86] Tencent. ncnn. `https://github.com/Tencent/ncnn`, referred on 4th/12/2022.

[87] A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le. Searching for MobileNetV3. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[88] Awni Hannun. Sequence modeling with ctc. *Distill*, 2017. https://distill.pub/2017/ctc.

[89] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[90] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented Transformer for Speech Recognition. In *Proceedings of INTERSPEECH 2020*, pages 5036–5040, 2020.

[91] Vitaliy Liptchinsky, Gabriel Synnaeve, and Ronan Collobert. Letter-based speech recognition with gated convnets. *arXiv preprint arXiv:1712.09444*, pages 1–13, 2017.

[92] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[93] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015)*, pages 234–241, 2015.

[94] Hamza Kheddar, Yassine Himeur, Somaya Al-Máadeed, Abbes Amira, and Fayçal Bensaali. Deep transfer learning for automatic speech recognition: Towards better generalization. *ArXiv*, abs/2304.14535, 2023.

[95] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 12 2014.

[96] Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, 2011.

[97] Google. Webrtc. *https://webrtc.org/*, Accessed 5 Feb 2024.

[98] Jens Heitkaemper, Joerg Schmalenstroeer, and Reinhold Haeb-Umbach. Statistical and neural network based speech activity detection in non-stationary acoustic environments. In *Interspeech*, 2020.

[99] Sina Alisamir, Fabien Ringeval, and Franc cois Portet. Cross-domain voice activity detection with self-supervised representations. *arXiv.org*, abs/2209.11061, 2022.

[100] J. Garofolo, Lori Lamel, W. Fisher, Jonathan Fiscus, D. Pallett, N. Dahlgren, and V. Zue. Timit acoustic-phonetic continuous speech corpus. *Linguistic Data Consortium*, 11 1992.

Acknowledgements

I have traveled a long way and endured much hardship. Now, at the prime age of 28, I finally present this doctoral dissertation to you. Over twenty years of pursuing education, a journey filled with trials and tribulations, every scene seems as if it were just yesterday.

I was born in a rural area of Sichuan Province in China, into a poor family. During my childhood, my parents worked away from home, and I often stayed with relatives and entered school at an early age. At that time, the transportation conditions were very poor, and I often had to walk dozens of miles through mountain roads to school before dawn. One by one, my playmates dropped out of school, but I persisted and continued my education until I was admitted to a university.

My parents have had a significant influence on me. My father loved to study in his youth, but he had to drop out of school because his family was too poor to afford the tuition fees, which became a lifelong regret for him. Therefore, he worked hard with the hope that he would be able to afford my education when I wanted to study. Although my mother never received formal education, she did her best to teach me and wanted me to be well-read. Under their guidance, I grew up longing to explore the world beyond the mountains, always believing that education could change my destiny.

In 2011, I was admitted to Qinghai University in China. At university, I studied industrial design, a field I love for its creativity, as it allows me to freely express my ideas through design. However, I realized that design is a form of creativity with limitations. I had endless ideas but couldn't easily bring them to life. Later, through the school's public foundation courses, I was introduced to the C language. I became interested in this equally creative field, but unfortunately, I was no longer eligible to change my major and thus missed the opportunity to deeply explore the charm of computer science. Yet, this experience planted a restless seed in my heart.

I graduated from university in 2015. I gave up the opportunity for postgraduate recommendation and chose to seek employment instead because I was doubting whether I truly wanted to become a designer. I wanted to find the answer through practical experience. I went to work in Shanghai alone, a city filled with hope and opportunities. There, I met many friends who were bravely experimenting in their life's journey, just to do what they loved. This encouraged me: if there is something you want to do, you should go for it, as time waits for no one.

I initially started with product design and later shifted to interaction design related to software, where I encountered voice user interfaces. This played a significant role in my decision to switch to the field of voice recognition after going to Japan for further studies. During my two years working in Shanghai, I was greatly influenced by Japanese design

culture. In my spare time, I began learning Japanese and exploring Japanese cinema. It wasn't until I heard the line 'I can feel your heart' in a movie that I finally decided to follow my heart and no longer be lost in uncertainty. After telling my parents about my decision to study in Japan, they understood and supported me without hesitation, which relieved me of any worries. Looking back today, I am constantly grateful to have such open-minded parents.

In early 2017, I quit my job and began a new life at a Japanese language school in Tokyo. My days were filled with Japanese language classes in the morning, part-time work in the afternoon, and studying my major in the evening, all while preparing for Japanese, English, and various university entrance exams. I often spent weekends at the National Library researching papers and saved money by living frugally to afford many books. Due to financial pressures, I struggled to focus on my studies; it was an extremely arduous process. When deciding whether to continue with my familiar field of design or venture into the almost unknown realm of computer science, I hesitated for a long time. Eventually, I decided to take a chance, knowing that I came to Japan to change my life. I chose speech recognition as my field of study. But with limited study time and no experienced seniors to consult, I had to learn on my own. Countless times, I dreamt of those nights spent studying under the lamp, feeling lost and helpless yet unyieldingly resilient. In my first year in Japan, I faced two failures in graduate school entrance exams. Just when I was losing hope, I met Professor Nishizaki.

At the end of April 2018, I visited Professor Nishizaki for the first time. I had a very enjoyable conversation with him. He recognized my learning experience and abilities and was willing to accept me as a graduate student. At that moment, it felt like I had walked out of an abyss and saw the long-missed sunlight. Over the next three years, Professor Nishizaki greatly assisted me in both my studies and my life. With his help, I truly became a programmer and entered the highly creative fields of voice recognition and deep learning.

In 2021, I successfully graduated with a master's degree. With the understanding and support of Professor Nishizaki, I returned to my home country to work and continued to pursue my doctoral degree on the job at University of Yamanashi. Thanks to the learning experiences in the lab, I began to engage in the research and development of artificial intelligence, fully leveraging my passion and talent. Professor Nishizaki often used his free time to remotely guide my research work and help publish the results, which greatly contributed to my meeting the requirements for my doctorate. During these years plagued by COVID-19, I managed to complete my master's and doctoral studies, which I now find even more precious. I express my deep gratitude to Professor Nishizaki for his extensive care and help over the past six years.

I am deeply grateful to Professor Hiromitsu Nishizaki of the Graduate School of Comprehensive Research, Engineering Domain, University of Yamanashi, for his tremendous support over the past six years. He has repeatedly offered valuable suggestions regarding my research and studies, and as the chief examiner of this dissertation, his guidance has been invaluable.

I would also like to express my gratitude to Professor Yoshimi Suzuki, Professor Fu-

miyo Fukumoto, Professor Ryutaro Obuchi, Associate Professor Masahiro Toyoura of the Engineering Domain, and Professor Kazuho Ito of the Life and Environmental Sciences Domain of the Graduate School of Comprehensive Research, University of Yamanashi, for their guidance as co-examiners of my doctoral dissertation.

I received advice from various professors as a researcher in the field of speech and language for this dissertation. I would like to express my deepest thanks to Professor Takehito Utsuro of the Intelligent Functional Engineering Domain, Systems and Information Science, Tsukuba University, and Associate Professor Akio Kobayashi of the Department of Intelligent Information Science, Faculty of Information Technology, Yamato University, for their advice on the content and direction of my research.

In addition, I am grateful for the enriching research life spent with the seniors, peers, and juniors in my laboratory, sharing both hardships and joys, and for their valuable discussions on research. In particular, I would like to express my deep gratitude to senior Guo Rui for her encouragement when I joined the lab and to senior Leow Chee Siang for his immense support in my studies and life over these six years.

There are unlucky aspects in my life; I was born into a poor family, which made my journey in education very difficult. But I am also very fortunate because I have met such great parents, and many good teachers and helpful friends. They have truly allowed me to change my own destiny through hard work.

Finally, heartfelt thanks are extended to all the teachers who participated in the review of this thesis.

謝辞

私は長い旅をし、多くの困難に耐えてきたが、28歳という絶好の年齢となった今、この博士論文を皆さんに差し上げる。20年以上にわたる勉強の旅は波瀾万丈で、あらゆる場面が昨日のことのように思える。

私は中国四川省の田舎の貧しい家庭に生まれた。幼い頃、両親が遠方で働いていたため、私は親戚の家に滞在することが多く、早くから学校に入学した。当時はバスがなかったので、夜が明ける前に何十キロも山道を歩いて学校に通わなければならないこともよくあった。仲間は次々と学校を退学していったが、私は大学に入学するまで粘り続けた。

両親は私に大きな影響を与えている。父は昔勉強が大好きでしたが、家が貧しかったので学費が払えず、学校を中退し、それが生涯の後悔になった。それで彼は、私が勉強したいときに学費を払えるように、一生懸命働いた。母は学校に行ったことはなかったが、私に知識のある人間になれるように一生懸命教えてくれた。彼らの教育を受けて、私は山の外の世界への憧れでいっぱいで、学ぶことが自分の運命を変えることができると常に信じてきた。

2011年に私は中国の青海大学に入学した。大学では工業デザインを学んでいったが、デザインを通じて自分のアイデアを自由に表現できるこのクリエイティブな分野が好きになった。しかし、デザインは創造性が制限されており、無限のアイデアはあるものの、それを簡単に実行することはできないことがわかった。その後、学校の公開基礎講座を通じてC言語に触れ、これも創造性の強いものに興味を持った。しかし、残念ながらその時点では専攻変更の条件を満たしていなかった。そのため、コンピュータサイエンスの魅力を深く感じる暇はなかったが、それは私の心に落ち着きのない種を植え付けた。

2015年に青海大学を卒業した。私は大学院の入学資格を諦めて就職することを選んだ。というのも、当時は本当にデザインが好きなのかという疑問があり、実践を通して答えを見つけたいと思っていたからだった。希望とチャンスに満ちている上海へ行った。そこでは、自分の好きなことに全力で取り組んでいる多くの友人に出会い、「やりたいことがあるなら、やるべきだ。時間は待ってくれない」と勇気づけられた。

最初はプロダクトデザインに携わり、その後ソフトウェア関連のインタラクションデザインに携わり、音声インターフェースに触れるようになった。これは、私が私が日本に留学しるときに音声認識の分野に参入する決意をしたことと重要な関係があった。上海で過ごした2年間、世界の広さを感じた。日本のデザインに大きな影響を受け、暇な時には日本語を勉強したり、日本の文化に触れるようになった。

ある日、「あなたの心が感じられる」というセリフを聞き、ようやく自分の心に従って一歩を踏み出そうと決心した。日本に留学することを両親に伝えた後、両親は私を理解し、サポートしてくれたので、不安はなくなった。今思い返すと、このような寛容な両親に恵まれたことにいつも感謝している。

2017年の初めに仕事を辞め、東京の日本語学校に留学し始めた。午前中は日本語の授業を受け、午後はアルバイト、夜は専門知識を学び、様々な試験の準備をしていた。週末には論文を読みに国立図書館によく行った。経済的困難により学業に集中できず、とても苦しい時期であった。自分の得意なデザイン専攻か、ほとんど馴染みのないコンピュータ専攻のどちらかを選び続けたとき、長い間迷ったが、人生を変えるために日本に来たのだと思い、最終的に挑戦してみることにした。私は方向性として音声認識を選択したが、しかし、勉強時間も少なく、参考になる先輩もいなかったので独学で学ぶしかなかった。今に至るまで、何度も夢の中でその年、夜の灯りの下で勉強していた自分自身に出会っている。彼は迷いながらも強い意志を持っていった。来日1年目で大学院受験に2度失敗してしまった。心が折れそうになった時、西崎先生に出会った。

2018年4月末、私は初めて西崎先生を訪問しました。先生との会話は非常に楽しく、私の学習経験と能力を認めてくださり、研究生として受け入れてくださった。その瞬間、私は深い淵から抜け出し、久しぶりに太陽の光を見たような気がした。それからの3年間、西崎先生には私の勉強や生活において大変お世話になった。先生の助けで、私は本格的にプログラマーになり、音声認識とディープラーニングといった非常にクリエイティブな分野に参入した。

2021年、無事修士号を取得した。西崎先生のご理解とご支援を得て、私は中国に帰国して仕事をし、山梨大学で現職博士号取得に向けて勉強を続ける。研究室での勉強のおかげで、私は人工知能の研究開発に携わるようになり、自分の情熱と才能を最大限に発揮することができた。西崎先生はよく自分の休み時間を利用して、私の研究をリモートで指導し、結果を発表し、最終的に博士卒業の要件を満たしてくださった。これはとても助かった。新型コロナウイルス感染症が猛威を振っていたここ数年、無事に修士課程、博士課程を修了することができ、そのことが一層尊いと感じた。

山梨大学大学院総合研究部工学域教授西崎先生には、日頃から大変お世話になり、私の研究と学習について何度も貴重な助言をいただき、また本論文の主査としてご指導いただき、心より感謝申し上げる。

山梨大学大学院総合研究部工学域教授鈴木良弥先生、山梨大学大学院総合研究部工学域教授福本文代先生、山梨大学大学院総合研究部工学域教授大渕竜太郎先生、山梨大学大学院総合研究部工学域准教授豊浦正広先生、山梨大学大学院総合研究部生命環境学域教授伊藤一帆先生には、博士論文の副査としてご指導いただき感謝の意を表する。

本論文の研究に対して音声・言語系の研究者として様々な先生方にご助言をい

た。筑波大学システム情報系知能機能工学域教授宇津呂武仁先生、大和大学情報学部知能情報学域教授小林彰夫先生には、研究内容や研究方針に関してご助言いただき深謝の意を表する。

また、研究室生活において、研究室の先輩・同期・後輩方には、研究室における生活において、共に苦労や楽しみを共に充実した研究生活を過ごせたこと、そして研究に関して討論をしていただき感謝の意を表する。特に、研究室に入るときにたくさん励ましてくれた郭瑞先輩と、この6年間勉強や生活において多大な援助をしてくれたLeow Chee Siang先輩に深く感謝の意を表したいと思う。

私の人生には不運があり、貧しい家庭に生まれたため、勉強の道は非常に大変であった。しかし、私はとても幸運である。なぜなら、このような素晴らしい親と多くの良い教師や友人に会ったから。彼らは私に努力を通して私の運命を本当に変えさせた。現在、私はこの20余年の知識を求める旅に私の最終的な答えを提出しようとする。それは満点ではないが、十分に優雅だと思っている。

最後に、本論文を査読していただきました先生方に心より感謝申し上げる。

The Relationship between The Published Literature and This Thesis

The following published papers are relevant to Chapter 4, "A Novel Real-Time Automatic Speech Recognition Toolkit."

1. Yu Wang, Chee Siang Leow, Akio Kobayashi, Takehito Utsuro, Hiromitsu Nishizaki, "ExKaldi-RT: A Real-Time Automatic Speech Recognition Extension Toolkit of Kaldi," Proceedings of the 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE 2021), pp.346-350, 2021.10.

2. Yu Wang, Chee Siang Leow, Hiromitsu Nishizaki, Akio Kobayashi, and Takehito Utsuro, "ExKaldi: A Python-Based Extension Tool of Kaldi," Proceedings of the 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE 2020), pp.470-473, 2020.10.

3. Yu Wang, Hiromitsu Nishizaki , Akio Kobayashi , Takehito Utsuro, Chee Siang Leow, "Development and Evaluation of Kaldi Extension Tools with Python," 情報処理学会研究報告，音声言語情報処理, 2019-SLP-130(5), pp.1-5, 2019.12.

4. レオ チーシャン，王 宇，小林彰夫，宇津呂武仁，西崎博光，"Kaldiベースのストリーミング音声認識システムの開発"，日本音響学会2021年秋季研究発表会講演論文集, 1-3Q-4, pp.1033-1036, 2021.9.

The following published papers are relevant to Chapter 5, "An Improved End-to-End ASR Model and Decoder on Embedded Devices."

1. Yu Wang, Hiromitsu Nishizaki, "A Lightweight End-to-End Speech Recognition System on Embedded Devices," IEICE Transaction on Information & Systems, Vol.E106-D, No.7, pp.1230-1239, 2023. 7.

Extramural Presentations

Journal Papers

1. Yu Wang, Hiromitsu Nishizaki, "A Lightweight End-to-End Speech Recognition System on Embedded Devices," IEICE Transaction on Information & Systems, Vol.E106-D, No.7, pp.1230-1239, 2023. 7.

International Conference Presentations (Peer-reviewed)

1. Yu Wang, Chee Siang Leow, Akio Kobayashi, Takehito Utsuro, Hiromitsu Nishizaki, "ExKaldi-RT: A Real-Time Automatic Speech Recognition Extension Toolkit of Kaldi," Proceedings of the 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE 2021), pp.346–350, 2021.10.

2. Yu Wang, Chee Siang Leow, Hiromitsu Nishizaki, Akio Kobayashi, and Takehito Utsuro, "ExKaldi: A Python-Based Extension Tool of Kaldi," Proceedings of the 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE 2020), pp.470-473, 2020.10.

Domestic Presentations (Non-refereed)

1. Yu Wang, Hiromitsu Nishizaki , Akio Kobayashi , Takehito Utsuro, Chee Siang Leow, "Development and Evaluation of Kaldi Extension Tools with Python," 情報処理学会研究報告，音声言語情報処理, 2019-SLP-130(5), pp.1-5, 2019.12.

2. レオ チーシャン，王 宇，小林彰夫，宇津呂武仁，西崎博光，"Kaldiベースのストリーミング音声認識システムの開発"，日本音響学会2021年秋季研究発表会講演論文集，1-3Q-4, pp.1033-1036, 2021.9.