Knowledge Distillation for End-to-End ASR in Resource-Constrained Environments

Paul Martin

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2023

Abstract

This dissertation explores knowledge distillation techniques for end-to-end Automatic Speech Recognition (ASR) models in resource-constrained environments. The aim is to reduce the size and inference time while maintaining a high-quality transcription. Experiments are conducted on Wav2vec 2.0 to determine a good initialisation strategy for shallower student models of the same architecture that learn to mimic the larger teacher model. It is found that copying the teacher's middle layers to the student yields the best Word Error Rate (WER) after subsequent distilling. This is validated for different student sizes.

Further, an effort is made to distil Wav2vec 2.0 to several smaller convolutional ASR models (CNNs) with faster CPU inference. As two CNNs have a $4\times$ shorter output length than the teacher, various subsampling methods are proposed and evaluated to determine a target for each student output frame. An alignment algorithm is determined that allows for the distillation to shorter-output students with comparable effectiveness as distilling to a student of the same output length as the teacher.

However, distilling from Wav2vec 2.0 to each CNN-based model does not improve their WER compared to fine-tuning using Connectionist Temporal Classification (CTC). Finally, the trade-offs between achieving a low WER and fast inference on CPUs and GPUs are analysed. The CNN models ContextNet and Citrinet identified as reasonable candidates for resource-constrained environments where only a CPU is available, as they achieve the best accuracy and fastest CPU inference. On devices with a GPU, using a 6-layer Wav2vec 2.0 distilled from a well-copied initialisation is preferable, which maintains a low WER with fast GPU inference.

Research Ethics Approval

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

Declaration

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

(*Paul Martin*)

Acknowledgements

I want to sincerely thank my supervisor Hao Tang for his exceptional guidance and supervision throughout, for being an oracle to all my questions and for inspiring many of my approaches and experiments. I am truly grateful for all the assistance he has given me.

Thank you to my family and friends for their support and encouragement. A special mention goes to my flatmate's dog, Wuwu for the occasional playful distraction.

Thank you to Julia for all her love and support.

Contents

Chapter 1

Introduction

Automatic Speech Recognition (ASR) plays a crucial role in digital speech processing, facilitating human-computer interaction by converting spoken language into written text, which is a primary mode of communication among humans. Since the first spoken digit recognizer was developed by Davis et al. in 1952, speech recognition technology has come a long way. For many years, Hidden Markov Model (HMM)-based systems and later hybrid HMM-deep neural network models dominated the field (Wang et al., 2019). However, recent advancements in hardware and the development of new neural architectures have led to significant improvements in the capabilities of end-to-end models. While most speech recognition systems can be labelled as 'end-to-end' since they turn speech signals (waveforms) into text, these comparatively newer models stand out because they use a single deep neural network that handles all parts of the recognition process, from one 'end' to the other.

1.1 Motivation

While servers and cloud platforms with high-end GPUs have the compute to run real-time inference tasks on large networks with hundreds of millions of parameters, many consumer-facing speech recognition systems are used on smartphones and desktop computers for dictation and nowadays also commonly on smart speakers such as Google Home or Amazon Alexa, which tend to have higher resource constraints. These could utilise remote servers to offload the recognition and further processing, but that creates latency, scalability issues, additional operating costs for the company offering the service and most importantly privacy concerns.

1.2 Contributions

In an effort towards high-quality on-device speech recognition in resource-constrained environments, this dissertation will focus on compressing existing end-to-end ASR models with the goal of decreasing model size and inference time on CPUs and GPUs while maximising accuracy.

At the time of writing, the dominant architecture in Automatic Speech Recognition is the self-attention-based Transformer architecture (Roger et al., 2022; Vaswani et al., 2017), which excels in capturing both short and long-range dependencies in sequential data. I am therefore using Wav2vec 2.0 (Baevski et al., 2020) as a representative Transformer-based model. The compression method of choice is Knowledge Distillation (Bucilă et al., 2006) in which a smaller student network is trained on the output of the larger teacher network.

Studies of Knowledge Distillation (KD) typically initialise the student either randomly, by copying the teacher's first (or last) layers, or by copying alternating layers. However, to my knowledge, there is no comprehensive study on which initialisation is most effective. As the layers of a neural network tend to encode different steps of the prediction pipeline (Pasad et al., 2022), I hypothesise and subsequently show how carefully choosing which layers to copy from the teacher has a non-negligible impact on how the student learns.

Another aspect of Knowledge Distillation I investigate is how we can effectively distil from the dominant Transformer architectures to smaller Convolutional (CNN) architectures which employ local filters to identify short-range dependencies. While this may first appear to be a significant constriction from transformers, most attention maps are nearly diagonal, attending only to local patterns. There have been several recent successes in small variants of CNN architectures performing comparably to small transformer models (e.g. Wav2vec 2.0-BASE with 95M parameters) (Li et al., 2019; Kriman et al., 2019; Han et al., 2020; Majumdar et al., 2021), but they have yet to achieve the accuracy of larger models, such as Wav2vec 2.0-LARGE (317M parameters). In this dissertation, I distil Wav2vec 2.0-BASE (instead of LARGE due to resource constraints) to various CNN architectures. As some of the students have a shorter output length than the teacher, I overcome this problem by proposing a **novel alignment method for distilling to shorter-output students**, allowing for more flexible cross-architecture knowledge distillation in ASR and possibly other fields.

Finally, I compare the performance of well-initialised student models derived from the Wav2vec 2.0 teacher model with the performance of CNN students for low-resource environments, finding that CNNs are indeed an alternative worth considering.

1.3 Structure

The rest of this dissertation is structured as follows. In Chapter 2 the relevant background on speech signals, the related model architectures, and knowledge distillation is introduced. Related work is discussed. Chapter 3 introduces and evaluates initialisation strategies for Wav2vec 2.0 students as one compression approach. Chapters 4 and 5 propose and evaluate subsampling methods for distilling to the shorter-output CNN students. The approaches of distilling to the same architecture and distilling to CNNs are compared using the models' parameter counts and inference times. Chapter 6 concludes this project by reviewing the main contributions, pointing out the limitations and suggesting future directions.

Chapter 2

Background

In this chapter, I will introduce the background required for following along with this dissertation. I will begin with the notion of waveforms and spectrograms as digital representations of speech, to then explore how different types of neural networks can perform automatic speech recognition to convert these speech signals to text. I will further provide a background on how Knowledge Distillation can be used to compress large models and finally argue why it is of interest to investigate distilling Transformer-based end-to-end models to Convolutional Neural Networks for Automatic Speech Recognition.

2.1 Digital representations of speech

As sound and thus speech is fundamentally a series of fluctuations in the air pressure picked up by a microphone, their digital representation is a plot of the air displacement (pressure) by time, as seen in Fig. 2.1. However, since we cannot record the change in air pressure as a continuous function, we instead sample it at discrete time steps. This is known as the sample rate and is for speech data often set at 16kHz, so 16 thousand samples per second. Wav2vec 2.0 uses waveforms as its input (Baevski et al., 2020).

Another widely used representation is the log-mel-spectrogram (also found in Fig. 2.1). As the name suggests it is a modification of the spectrogram. The spectrogram is a representation of the waveform frequencies as time progresses. It is computed by applying a Discrete Fourier Transform on a sliding window of length T_{window} and stride T_{stride}. The amplitudes of the extracted frequencies (oftentimes on a log scale) are plotted across time. As we humans have a more fine-grained auditory perception of lower frequencies than higher frequencies, Stevens et al. (1937) proposed the Mel-scale. Applying this scale to the spectrogram by averaging small overlapping groups in the low frequencies and larger overlapping groups in the higher frequencies, we get the Mel-spectrogram. Let the total number of groups be denoted by $n_{filters}$. The log-mel-spectrogram has the amplitudes (y-axis) scaled logarithmically. The three CNN students that I am using all take log-mel-spectrograms with $T_{window} = 25$ms, $T_{stride} = 10$ms, and $n_{filters} = 80$ as their input (Kriman et al., 2019; Han et al., 2020; Majumdar et al., 2021).

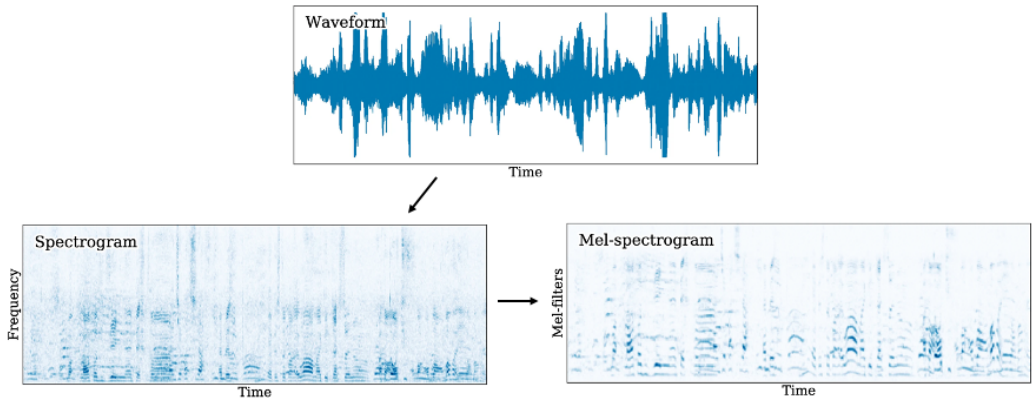


Figure 2.1: Example of a waveform, its corresponding spectrogram and mel-spectrogram as digitalisations of sound. The waveform shows the air compression over time, the spectrogram shows the amplitude of each frequency composing the signal as time progresses, and the mel-spectrogram averages the spectrogram's frequencies according to the mel-scale. The amplitude is shown by the darkness at each frequency-time or filter-time pixel. The arrows display the order in which each representation is derived. Diagram adapted from Benito et al. (2019).

2.2 Wav2vec 2.0

Of the models used in this dissertation, I begin by introducing Wav2vec 2.0, as a representative self-supervised Transformer model. The details of self-supervision and Transformers are described below.

The Wav2vec 2.0 architecture consists of two main components: the feature extractor and the Transformer stack. The feature extractor converts waveform batches into latent speech representation vectors $\boldsymbol{z}_1, ..., \boldsymbol{z}_N$ with a total stride of 20ms between each batch and a window size of 25ms.

The latent speech representations are then fed into a series of Transformer Layers. Transformers were originally introduced by Vaswani et al. (2017) for machine translation, but have found their place in many other areas of deep learning, including speech recognition. While the originally proposed architecture uses a stack of six encoder-type self-attention layers and a stack of six similar self-attention layers for the decoder, networks in ASR tend to use a stack of 12 encoder-type Transformer layers and replace the decoder with an RNN or, in the case of Wav2vec 2.0, a linear layer. Thus, I only describe the workings of the Transformer encoder layer here and any references to a Transformer layer henceforth address the encoder.

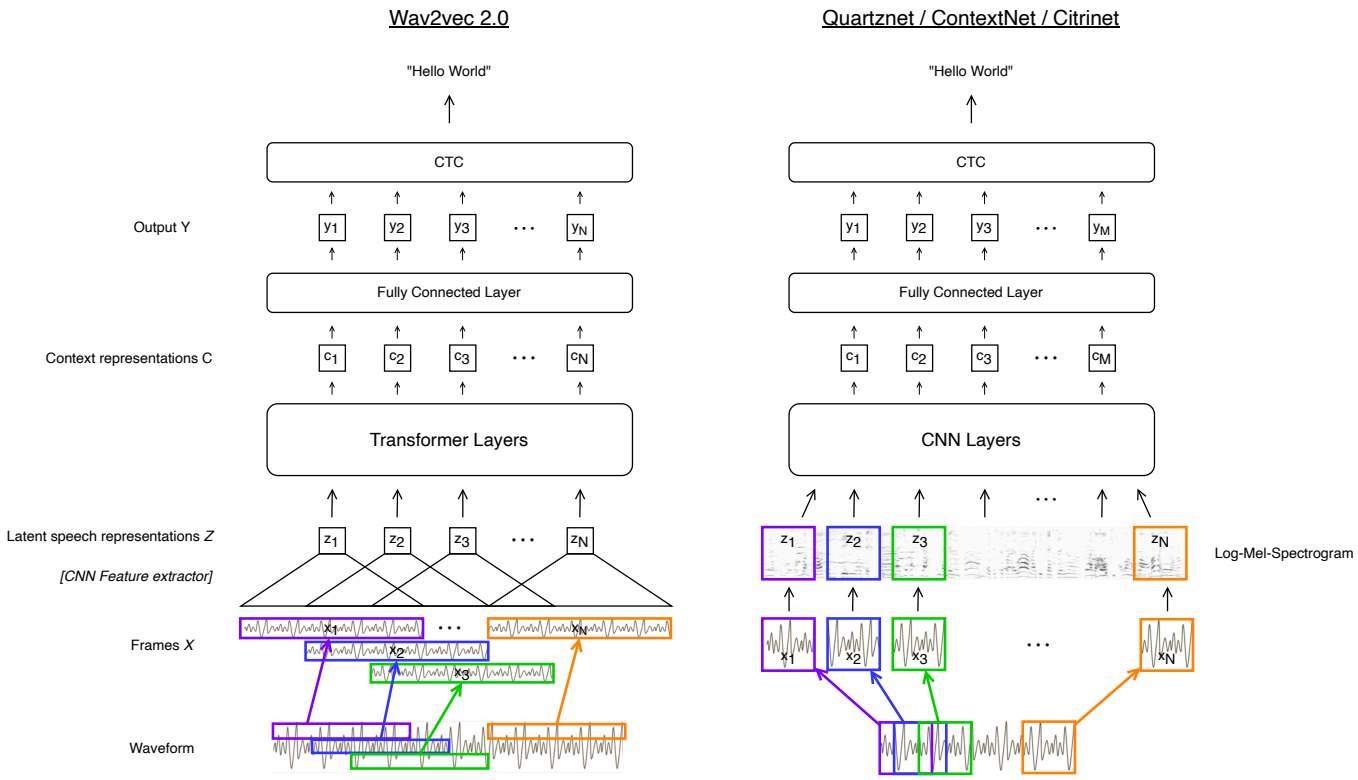


Figure 2.2: Diagram showing the abstract architectures of Wav2vec 2.0 on the left and the CNN-based Quartznet, ContextNet and Citrinet. Wav2vec 2.0 starts by splitting the waveform into overlapping frames (bottom). These are fed into a CNN feature extractor that extracts important features from the waveform data, as a replacement for the traditional Mel-frequency Cepstrum. The resulting latent speech representations can now be processed by a series of Transformer layers. These have been abstracted away, but are detailed in Section 2.2. After passing through the main processing step, Wav2vec 2.0 uses a fully connected layer for post-processing, to reduce the context representations to values representing the vocabulary. Finally, a linear layer and decoding algorithm (see Section 2.4) are used to join the outputs together and produce the final string. The Covolutional networks on the right use a log-mel-spectrogram instead of a CNN feature extractor for their latent representations $\boldsymbol{z}_1, ..., \boldsymbol{z}_N$. They also use a series of CNN layers, which are detailed in their respective sections (2.3).

Transformer

The Transformer's main task is to augment each element in a sequence of input vectors with its context of one or more other elements from the same sequence. In text processing this can, for example, be a verb's subject and object, or a pronoun's antecedent (e.g. "*John* ate *his* sandwich"). In speech recognition, examples of the acoustic context are the pitch, or the previous and next phones (basic unit of speech). To incorporate these factors for each element, the transformer learns to select which other input elements to include (or *pay attention to*).

The Transformer architecture consists of three fundamental building blocks: a positional

encoding, self-attention unit, and non-linearity in the form of a feed-forward network.

Self-attention We start with the latent speech representations $\boldsymbol{z}_1, ..., \boldsymbol{z}_N$ produced by Wav2vec 2.0's feature extractor. To determine where to point our attention head (i.e. which element to use as context) for state $\boldsymbol{z}_i$, we perform a search query $\boldsymbol{q}$ on a set of key and value pairs $(\boldsymbol{k}_j, \boldsymbol{v}_j)$ derived from the state vectors. The query is formulated as

$$\boldsymbol{q}_i = \boldsymbol{z}_i W_Q$$

and the key-value pair for each state $\boldsymbol{h}_j$ can be generated as

$$\boldsymbol{k}_j = \boldsymbol{z}_j W_K, \quad \boldsymbol{v}_j = \boldsymbol{z}_j W_V$$

where W_Q, W_K and W_V are learnable weight matrices.

Performing the search is done by first computing the attention weight w_j for each key as

$$w_j = \boldsymbol{q}_i^\top \boldsymbol{k}_j,$$

then normalising the weights using a softmax and finally taking the weighted sum over all values

$$\boldsymbol{c}_i = \sum_{j=1}^{N} \alpha_j \boldsymbol{v}_j \quad \text{where} \quad \alpha_j = \frac{\exp(w_j)}{\sum_{l=1}^{N} \exp(w_l)}.$$

In full,

$$\boldsymbol{c}_i = \sum_{j=1}^{N} \frac{\exp(W_Q^\top \boldsymbol{z}_i^\top \boldsymbol{z}_j W_K)}{\sum_{l=1}^{N} \exp(W_Q^\top \boldsymbol{z}_i^\top \boldsymbol{z}_l W_K)} \boldsymbol{h}_j W_V \tag{2.1}$$

Instead of having just a single self-attention selector, Transformer layers tend to combine multiple such self-attention units into a multi-head attention, taking their weighted average using learned weights. The advantage of this is that multiple different positions in the sequence can be attended to with varying importance. A typical number of heads is 8.

Positional encoding Using only the multi-head self-attention mechanism, we quickly run into the issue that after augmenting a hidden vector $\boldsymbol{z}_i$ with its previous and subsequent vectors $\boldsymbol{z}_{i-1}$ and $\boldsymbol{z}_{i+1}$, the resulting $\boldsymbol{c}_i$ is indestinguishable from a $\boldsymbol{c}'_i$ where the context is flipped such that $\boldsymbol{z}'_i = \boldsymbol{z}_{i+1}$ and $\boldsymbol{z}'_{i+1} = \boldsymbol{z}_i$. Any temporal information in the context is therefore lost. To solve this, we can add a positional encoding vector to each input vector $\boldsymbol{z}_i$ before feeding it into the transformer. Vaswani et al. use the sinusoidal function

$$PE_{(i,2j)} = \sin(i/10000^{2j/d_{model}})$$
$$PE_{(i,2j+1)} = \cos(i/10000^{2j/d_{model}})$$

where j specifies the location in the position encoding vector for the ith input vector $\boldsymbol{z}_i$ and d_{model} is dimension of the layer's output vector. Other approaches exist.

Non-linearity The final Transformer building block is a non-lineary in the form of two position-wise feed-forward layers with a ReLU activation in between, operating independently on each context vector $\boldsymbol{c}_i$.

Self-supervised learning

With knowledge of how the Wav2vec 2.0 architecture works, we can now look at how it is trained using self-supervised learning (SSL). Self-supervision is a learning paradigm in which a model learns patterns in the data without the need for explicitly annotated data. Instead, the model is taught to solve a pretext task such as reconstructing parts of the input sequence that have been masked. This avoids the need for large amounts of task-specific data that is usually prepared in a laborious job usually done by humans, if it is available at all. Once the model has learnt to solve the pretext task, it can be fine-tuned on a specific task. Self-supervised models often perform better than models trained only on the downstream task and require significantly less annotated data.

For instance, the model proposed by Han et al. (2020) is trained on 970 hours of labeled speech data for ASR using a supervised approach, while the SSL-trained Wav2vec 2.0 is first trained on 53k hours of unlabeled data and subsequently on 100h of labeled data for ASR and achieves a comparable accuracy.

I will elaborate a little further on how Wav2vec 2.0 works, as it is a key part of this project. A visualisation of its self-supervision stage can be found in Fig. 2.3. As detailed in Section 2.2, the model begins by extracting latent speech representations Z using its feature extractor, where each representation vector encodes a 25ms window with a 20ms stride. These are then passed into a chain of Transformers. The key difference between supervised training on the downstream task and self-supervised training is that in the downstream task all vectors from Z get passed into the Transformer block to be transcribed while in Wav2vec 2.0's pretext task some latent speech representations get masked before being fed into the Transformer block. The model should then predict the quantization of each masked vector.

Once the pre-training step is complete, the quantised representations are removed and the model is fine-tuned on a labeled speech dataset together with the CTC loss described in Section 2.4.

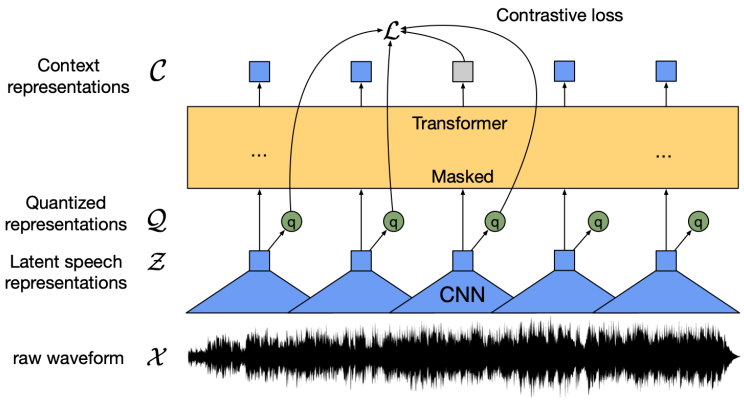


Figure 2.3: Visualisation of the SSL pre-training stage of Wav2vec 2.0, taken from the original publication (Baevski et al., 2020). It shows how the waveform is fed into a CNN encoder to generate latent speech representations. They are passed into a chain of Transformers with some latent representations masked. The model's objective in this stage is to predict the quantised representations of the masked sections. The function to compute the quantised representations is trained in parallel to the prediction, as part of the self-supervision stage.

2.3 CNN models

Besides Wav2vec 2.0, I am also using three different CNN-based models. Namely, QuartzNet (Kriman et al., 2019), ContextNet (Han et al., 2020), and Citrinet (Majumdar et al., 2021). I will detail the relevant aspects of their architectures their differences and why I chose them.

Convolutional Layers First, a primer on Convolutional (or CNN) layers. They work by convolving a 1D or 2D kernel (also referred to as filter) over their input. Which neighbouring frames are incorporated to what extent depends on the structure of the filter, which is learnt during the training phase. The CNN layers used in the following architectures utilize a larger kernel size and dilation (expands the kernel by skipping intermittent pixels) to capture long-range dependencies in the input audio data.

General architecture

The three models I am introducing in this section are based on the same convolution-only Jasper architecture proposed by Li et al. (2019), which itself was inspired by Wav2Letter (Collobert et al., 2016).

As visualised in Fig. 2.2, the model has no CNN feature extractor and instead relies on a pre-computed Log-Mel-Spectrogram. Its architecture consists of a stack of B so-called *Jasper blocks*, which themselves consists of R *sub-blocks*. Each sub-block has a 1D convolutional layer, followed by batch normalisation (Ioffe and Szegedy, 2015), and a ReLU activation function (Nair and G. E. Hinton, 2010). Each Jasper block is followed by a residual connection (He et al., 2015).

As in Wav2vec 2.0 the output of the central CNN block gets passed through a fully connected layer that adapts each context representation $\boldsymbol{c}_i$ to the vocabulary size. Jasper, as well as the derived QuartzNet and Citrinet are trained using a CTC loss.

QuartzNet

With the largest proposed version of Jasper having 333M parameters (54 convolutional layers) and the smallest version having 201M parameters (34 convolutional layers), it is a rather large model compared to the 95M parameters of Wav2vec 2.0-BASE. QuartzNet has therefore been developed by Kriman et al. (2019) as a light-weight alternative to Jasper, having only 20M parameters. The authors achieve this significant reduction by using the deptwise seperable convolutions introduced by Howard et al. (2017) instead of standard convolutions, though Kriman et al. rename them to *time-channel separable convolutions*. The key improvement here is that while standard convolutions perform the channelwise and spatial computation at the same time, the separable convolution first applies a convolutional filter to each channel independently to then, in a second step, apply 1x1 pointwise convolutions to create a linear combination of the channels. This change reduces the number of parameters and makes the computation faaster without significantly sacrificing performance.

ContextNet

ContextNet (Han et al., 2020) is remotely based on QuartzNet, but improves upon it by adding *squeeze-and-excite* (SE) units from Hu et al. (2020) (see Fig. 2.4). The SE unit adds a wider context by first average-pooling (Lecun et al., 1998) the incoming feature map's channels to pass the resulting vector through a bottleneck formed by two fully connected layers with ReLU and sigmoid activation functions, where the it is first *squeezed* by a factor of 8, to subsequently be expanded (*excited*) back to its original length. The resulting vector is treated as a list of scaling factors that is applied to each channel in the original feature map using pointwise multiplication. This method learns to add a wider context by emphasising some parts of the (processed) utterance and suppressing others.

Importantly, ContextNet has a stride of 2 in four of its convolutional layers, resulting the number of frames $\boldsymbol{y}_1, ..., \boldsymbol{y}_M$ output by the CNN encoder to be 4x shorter than the number of input frames $\boldsymbol{c}_1, ..., \boldsymbol{c}_T$. This is in contrast to Wav2vec 2.0 and QuartzNet keeping the number of frames constant and will become very relevant when distilling from Wav2vec 2.0 to ContextNet.

The original implementation uses an RNN-T decoder A. Graves, 2012, but for consistency with the other models I am replacing it with a CTC decoder here.

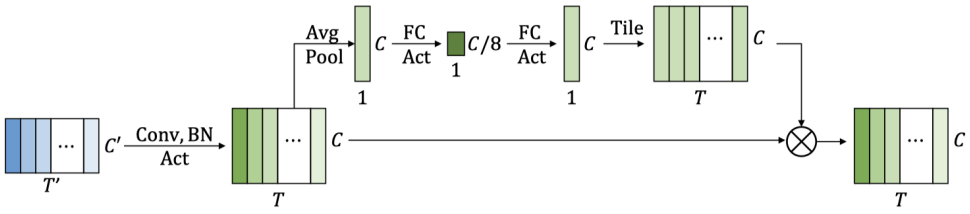


Figure 2.4: Visualisation of the squeeze-and-excite unit (in green) taken from Han et al. (2020). A set of T context vectors is input from the left. The vectors are averaged to a single vector, compressed ($8\times$) and subsequently expanded using feed-forward layers. It is then treated as a mask to be pointwise multiplied with each of the input context vectors. The result is output.

Citrinet

Citrinet's architecture (Majumdar et al., 2021) is very similar to that of ContextNet with small differences in the kernel size and its use of a ReLU activation function instead of SiLU (Hendrycks and Gimpel, 2016) used by ContextNet. I am including it in this dissertation to have another comparable squeeze-and-excite-based model with a slightly different parameter setting, and that was intended to be used with a CTC loss.

2.4 Training for ASR using Connectionist Temporal Classification

Now that all relevant architectures are introduced, I will elaborate on how they are trained.

Training and Fine-tuning

Wav2vec 2.0 was proposed has self-supervised learning, as detailed previously, at it's core and is subsequently fine-tuned (adapted to a specific task, typically using a smaller dataset) for ASR using a supervised training schedule with a CTC loss, as explained below.

QuartzNet, ContextNet and Citrinet do not use a self-supervision stage and are directly trained on supervised ASR. QuartzNet and Citrinet are trained on a CTC loss, but ContextNet is trained using an RNN-T decoder A. Graves, 2012, which I change for a CTC decoder for consistency in distilling. This training process, however, requires significant computing power (e.g. QuartzNet training took 5 days on eight NVIDIA Tesla V100 GPUs), and is not something I can easily replicate with the available hardware. Therefore I am using the pre-trained QuartzNet 15x5, ContextNet 256 and Citrinet 256 models published by NVIDIA (2022; 2021a; 2021b) and fine-tuning them on my own dataset . For more details, refer to Section 5.1.1.

Connnectionist Temporal Classification

Connnectionist Temporal Classification (CTC) loss (Alex Graves et al., 2006) is the training objective used in Wav2vec 2.0, QuartzNet, Citrinet and my adaptation of ContextNet. Its advantage over, for example, cross-entropy for speech is that while in cross-entropy loss each of the network's output frames is compared (aligned) to a single target frame, CTC computes a likelihood for the output by summing up the probabilities of all possible alignments. For instance, of the three outputs

`EXAMPLE*******`, `***EXAMPLE****`, `**EXAAAMM**PLE`

where * represents a `<PAD>` token, only the first (and possibly the second) might align well in a default cross-entropy setting. A modified alignment of frames to a so-called *forced alignment* by pre-segmenting the training speech data can get around this. In contrast, the CTC approach of considering all possible alignments is much more flexible and has a smaller impact on the structure of the network, as force-aligned models tend to have an HMM component to model the long-range sequential structure Alex Graves et al., 2006.

As seen in the rightmost example above, subsequent occurrences of the same character are reduced to a single character (e.g. `AAAMM` becomes `AM`). Therefore, to allow repeating characters, such as in `HELLO`, a 'blank' token must be output between each repeating character. While the blank token may be distinct from the pad token, the implementations of Wav2vec 2.0 (Facebook-Research, 2022), QuartzNet (NVIDIA, 2022) and Citrinet (NVIDIA, 2021a) use the same token as padding and blank.

It is also important to note that CTC makes an independence assumption, which assumes that each output label in the output sequence is conditionally independent of the others given the input sequence (waveform or Log-Mel-Spectrogram). While this assumption makes the loss computation more tractable, it may limit its language modelling capabilities. For instance, a model with a CTC decoder may confuse the spellings of "to", "too" and "two" in the spoken utterance "There are two things to do". This can be improved by introducing an additional language model, though I am not using any for my experiments, as they are unimportant for my proposed methods and add unneccesary complexity.

2.5 Knowledge Distillation

As touched upon in the introduction, the problem with current state-of-the-art models (Roger et al., 2022), such as Wav2vec 2.0 is their size. With multiple million or billion of parameters they not only consume vast amounts of memory but also have a rather slow inference. Originally introduced by Bucilă et al. (2006) and advanced by G. Hinton et al. (2015), Knowledge Distillation (KD) aims to solve this issue by compressing a large model (the teacher) to a smaller one (the student), retaining as much of the teacher's accuracy as possible. In contrast to training the smaller model from scratch, in Knowledge Distillation we teach the student to copy the teacher's distribution over all possible outputs, instead of attempting to attain a 'perfect' distribution of 1 for the ground truth and 0 for all other options.

To perform the distillation, we feed the original dataset that was used to train the large model (or a similar one) into the teacher. In parallel, we feed the same data into the student network. The student's loss is calculated by summing the Kullback-Leiber divergences (eq. 2.2) between each student output frame $\boldsymbol{y}_1^S, ..., \boldsymbol{y}_N^S$ and its corresponding teacher frame from $\boldsymbol{y}_1^T, ..., \boldsymbol{y}_N^T$, as shown in equation 2.4.

The Kullback-Leiber (KL) divergence between the prediction and target output vectors $\boldsymbol{y}$ and $\boldsymbol{y}^*$ is given by

$$D_{KL}(\boldsymbol{y}^* \| \boldsymbol{y}) = \sum_{j=1}^{\dim(\boldsymbol{y})} y_j^* \log\left(\frac{y_j^*}{y_j}\right) \tag{2.2}$$

$$= H(\boldsymbol{y}^*, \boldsymbol{y}) - H(\boldsymbol{y}^*) \tag{2.3}$$

where the subscripts y_j and y_j^* are the jth element of the respective vectors.

Summing the KL-divergences of all student-teacher frame pairs gives the loss for a single datapoint.

$$\mathscr{L}_{KD} = \sum_{i=1}^{N} D_{KL}(\boldsymbol{y}_i^T \| \boldsymbol{y}_i^S) \tag{2.4}$$

We can also write the KL-divergence as shown in eq. 2.3, where $H(\boldsymbol{y}, \boldsymbol{y}^*)$ is the cross-entropy between the prediction frame $\boldsymbol{y}$ and the target $\boldsymbol{y}^*$, and $H(\boldsymbol{y}^*)$ is the target frame's entropy. Since the teacher is frozen (not being trained), it will always output the same frames for a given input. Hence, $H(\boldsymbol{y}^*)$ is a constant and will be nullified when taking the derivative of $\mathscr{L}_{KD}$ during backpropagation. I am mentioning this for the reader's reference, as some papers write Knowledge Distillation in terms of cross-entropy instead of KL-divergence, as the training outcome is the same.

As an extension to computing the $\mathscr{L}_{KD}$ of the output frames, some newer publications (Chang et al., 2022; Lee et al., 2022) go a step further and also match select hidden states produced by the layers.

2.6 Criticism of Related Work

This section introduces and discusses the prior work related to the focus of my dissertation, thereby motivating the relevance of my project in the field.

Initialising students Despite various publications having studied the distillation of ASR models, and specifically Wav2vec 2.0, only Peng et al. (2021) appear to remotely explore the impact that the student's initialisation has on the accuracy after distillation. Other works may use the initialisations of previous papers, despite their selection being unjustified (Yang et al., 2023; Fu et al., 2023). Even the exploration done by Peng et al. is limited to comparing two initialisations of which one is unjustified. However, they nonetheless show that the choice of initialisation matters significantly. To improve on their work, I investigate the initialisation problem further.

Transformer to CNN distillation While Knowledge Distillation has been shown to work effectively in compressing large models to smaller ones of the same architecture, there is very little work done on cross-architecture KD. This area and especially the distillation of Transformer-based models to CNNs is of particular interest, as both have shown very promising results in ASR (Baevski et al., 2020; Hsu et al., 2021; Gulati et al., 2020; Li et al., 2019; Kriman et al., 2019; Han et al., 2020; Majumdar et al., 2021). While Transformers can still boast with marginally better Word Error Rates (WER) (See LibriSpeech ranking on Papers with Code, 2022), CNNs are slowly catching up. The advantage of CNNs is their significantly faster inference time and lower parameter count (See Table 1 in Chia et al., 2019).

While individual publications on distilling Transformers to CNNs for image classification (Liu et al., 2022) and text classification (Chia et al., 2019) exist, showing it can be done, there has so far been no work on this in speech processing. This is a shame, as distilling to CNNs appears to be a good way to further compress the large Transformer models besides reducing their width or depth.

Thus, I investigate the effectiveness of distilling a Transformer-based SSL model to a CNN in order to reduce both size and inference time significantly while retaining a low WER. To this end I investigate the impact of different student architectures and compare the results to training a CNN from scratch to form a conclusion about the relevance of the Transformer-to-CNN framework in ASR. Importantly, I propose a new method of alignment for distilling a CTC-based model to students with fewer output frames than the teacher.

Chapter 3

Preliminary experiments on student initialisation

In this chapter, I explore whether a Wav2vec 2.0 student with 2, 6 and 10 transformer layers can learn better when initialised with a selection of parameters from its larger 12-layer Wav2vec 2.0 teacher. I start by motivating the investigation, stating the hypotheses and providing a brief overview of the experimental setup and finally discuss the results.

3.1 Prior work

While other publications (Yang et al., 2023; Fu et al., 2023) have worked on distilling Wav2vec 2.0 for ASR, Peng et al. (2021) are the only ones, to the best of my knowledge, that compare how a student's initialisation impacts its achieved performance. Specifically, they compare initialising by copying alternating layers with copying the last layers. The alternating strategy is adopted from DistilBERT (Sanh et al., 2020), but no justification for using the last layers is given. Chang et al. (2022) and Fu et al. (2023) propose to distil ASR models not only based on the student's and teacher's outputs but also by comparing their hidden representations after specific layers. In particular, Chang et al. (2022) distil a similar network to Wav2vec 2.0 to a 2-layer network of the same architecture. They first initialise with the teacher's first two layers and then finetune the student to predict the teacher's hidden representations after the latter's 4th and 8th transformer layer as well as the final output. The authors reason that each of these layers encodes important information about specific tasks that they were interested in (4th layer for speaker identification, 8th layer for ASR and 12th layer for keyword spotting and intent classification).

3.2 Hypotheses on good layer selection

Thus, I make the following hypotheses about which teacher layers may be favourable for initialising the student:

Hypothesis 1. *A fine-tuned layer knows how to process its previous layer's outputs, so*

copying ***subsequent layers may avoid re-learning how to process each other's inputs****. This may be extended to favour any pair of layers that are close.*

Hypothesis 2. *Far-apart layers tend to encode different information, so copying* ***distant layers would transfer more information****.*

Hypothesis 3. ***Middle layers will be more important to copy****, as Pasad et al. (2022) find that 'the central layers encode the most contextual information' in Wav2vec 2.0.*

Hypothesis 4. *Sequential layers encode sequential processing steps. Thus, the* ***order of layers in the teacher should remain the same*** *when copied to the student. If teacher layer L_i comes before layer L_j, their order shall not be reversed to L_j, L_i in the student.*

Hypothesis 5. ***Larger students depend less on the layer choice****, as a wider range of layers is copied from the teacher, conveying most information.*

While hypotheses 1 and 2 appear contradictory, it may be important to strike a balance between them. The layers should be close enough to easily adapt to their previous layer's output, but far enough to still encode sufficiently different processing steps.

3.3 Experiments

To test hypotheses 1–4 I conceive a set of experiments, where a 12-layer Wav2vec 2.0-BASE is first fine-tuned on an ASR task from the English Wall Street Journal (WSJ) dataset (Paul and Baker, 1992). A 2-layer Wav2vec 2.0 student is initialised by copying a specific combination of layers from the teacher (the feature extractor is also copied) and subsequently fine-tuned by distilling the teacher's output frames $\boldsymbol{y}_1^T, ..., \boldsymbol{y}_N^T$ (see Fig. 2.2) to the student's output frames $\boldsymbol{y}_1^S, ..., \boldsymbol{y}_N^S$ using the KD-loss from equation 2.4. The loss is averaged for each batch. The performance is measured using the Word Error Rate (WER) metric, which ranges from 0 to 100%, with lower values being preferable.

The initialisation combinations used are, as pairs of teacher layers: (1,2), (6,7), (11,12), (4,9), (1,12), and (8,5). A random student initialisation is also trained as a baseline. These are chosen to test the hypotheses through the following comparisons:

I) Comparing the students with initialisations (6,7), (4,9), and (1,12) will indicate whether it is better to use consecutive teacher layers (hyp. 1) or distant layers (hyp. 2), and whether close but non-adjacent layers are acceptable, or may even be favourable. If hypothesis 1 holds, the (6,7) student should attain a low WER while both other students attain a high WER. If hypothesis 2 holds, student (1,12) should be better.

II) Comparing the students with initialisations (1,2), (6,7) and (11,12) will test hypothesis 3. If it holds, the student with (6,7) should attain a lower WER than the other two. All pairs consist of consecutive layers to not interfere with testing hypotheses 1&2

III) Comparing students (6,7) and (5,8) to their respective inverses (7,6) and (8,5), the students with inverted initialisations should do significantly worse if hypothesis 4

holds.

These comparisons each form a small sample set, so more experiments are run as shown in Figure 3.1.

In addition to these experiments, I also test whether the effect of changing the initialisation configuration has a lower impact on larger students (hyp. 5) by distilling the above teacher to 6-layer and 10-layer Wav2vec 2.0 students in the configurations from Table 3.1.

Experimental Setup The self-supervised teacher Wav2vec 2.0-BASE model published on Hugging Face (Facebook, 2021) is fine-tuned for ASR on the WSJ dataset using a Sentencepiece tokeniser (Kudo and Richardson, 2018). I have split the official training data into custom train and dev sets with a 98.64/1.36 split at the speaker boundary, such that no individual is in both sets. The model was fine-tuned over 50 epochs with a batch size of 4, an Adam optimiser (Kingma and Ba, 2017) and an exponentially decaying learning rate schedule, starting at 2.6×10^{-5} and decaying by 10% every epoch. A grid search for the learning rate is in Appendix A. The fine-tuned Wav2vec 2.0-BASE achieves a WER of 2.1% on the dev-set.

The 2-layer Wav2vec 2.0 students were distilled with the same settings as the teacher and used the same WSJ train set.

In adaptation to the memory limitations of my GPU (NVIDIA GeForce RTX 2080 Ti), the 6-layer and 10-layer students are trained using a batch size of 2. As these are larger models and take longer to train, I have kept the number of epochs at 20 to allow the loss to plateau.

For further details on the experimental setup, refer to chapter 5.

3.4 Results and Discussion

The Word Error Rates attained by each 2-layer student after the distillation are shown in Fig. 3.1. Each cell is a student that was initialised by copying two layers from the teacher, where the x and y-axes specify the teacher's layer that was copied to the student's first and second layers respectively. The cell's colour represents the WER at which the student converged. For instance, the bottom right cell shows that the student initialised with teacher layers (11,12) converges to a dev-WER of 39.6%.

Referring to the comparisons outlined in the previous section (3.3), the following conclusions can be drawn

I) For the hypothesis that consecutive layers are favourable over distant layers (hyp. 1) to hold, we would expect student (6,7) to achieve a significantly lower WER than any students initialised with non-consecutive layers. This is not the case. While the far-apart initialisation (1,12) with a WER of 39.6% is comparable to the random initialisation (WER of 39.8%) and indeed much worse than the 29.3% achieved by the consecutive initialisation. However, student (4,9) also attains a better-than-random WER of 32.5% and student (5,8) even performs slightly better

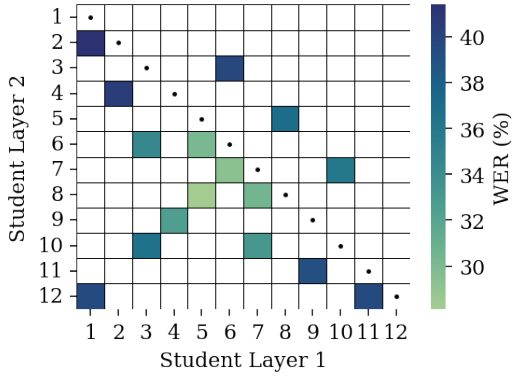


Figure 3.1: Heatmap comparing initialisation strategies for a 2-layer Wav2vec 2.0. Each cell is a student that is initialised by copying the layers on the x and y-axis from the teacher to the student's first and second layers. In the left triangle, the order of layers is preserved from the teacher. In the right triangle, the order is flipped. The colour shows the WER after knowledge distillation. Lighter is better. The student with initialisation (5,8) does best while the students in the corners do worst. Training parameters are described in Section 3.3.

(28.1%) than the student with adjacent layers, despite them having copied layers that have 1 and 2 layers between them in the teacher network. This indicates that factors other than proximity may be more important when choosing which layers to copy.

II) To check whether middle layers are favourable to copy (hyp. 3), we can see from Fig. 3.1 that the students in the grid's centre attain a WER that is approximately 10%-points less than the WER of students near the corners. In particular, the student initialised with central layers (6,7) performs significantly better than the students initialised with the first two layers (1,2) and the last layers (11,12). Both non-central students are comparable to the randomly initialised student. Thus, the evidence supports hypothesis 3.

III) In Fig. 3.1 we can see that students (6,3), (8,5), and (10,7), which were initialised by reversing the order of layers from the teacher, attain WERs of 36.0–39.7%. These are significantly worse than the WERs of students (3,6), (5,8), and (7,10) that copied the same layers in their original order. Interestingly, some of the reversed students are marginally better than the randomly initialised student. This may be worth investigating further.

Thus, I have gathered evidence to support my hypotheses that it is the most effective to copy the central layers (hyp. 3) and that the order in which the layers are copied should not be reversed (hyp. 4). The tests for hypotheses 1 and 2 are inconclusive, as the layers copied to student (4,9) were originally three layers apart, but the student still does much better than a randomly initialised student.

To test these hypotheses on students with more than two layers and test whether the importance of layer selection declines for larger models (hyp. 5), I distil the teacher to 6-layer and 10-layer models, as shown in Table 3.1. For both model sizes, the random

initialisation does much worse than the students initialised from the teacher. This is to be expected, as the randomly initialised models cannot benefit from the initialisation found by the self-supervised pre-training of Wav2vec 2.0.

Considering only the initialisations by copying, the 6-layer student sees its WER reduced by approximately 40% and 60% when using the middle layers instead of the last or first layers respectively. For completeness, I also include the results from initialising using every second layer (even or odd layers), as proposed by Sanh et al. (2020) and used for Wav2vec 2.0 by Peng et al. (2021). Using alternating layers is better than initialising using the first or last layers, but not as good as using the middle layers.

When distilling to the 10-layer student, the importance of selecting good teacher layers diminishes almost entirely. Surprisingly, copying the middle layers does worse than using the first or last. However, I suppose this may be because choosing layers is now more about which layers to remove than which to keep, as 10/12 layers are copied. The difference between the distilled models is negligible, though, with only a 7% improvement from using the middle layers to the last layers.

Summarising the differences in initialisations between model sizes, the patterns found in the investigation of the 2-layer student appear to also hold for the 6-layer student. However, they appear not to hold for the 10-layer student, though the small differences in WERs among the 10-layer students may be due to randomness in the distillation pipeline. Appendix Fig. B.1c shows this similarity between the students across epochs. Overall, the impact of choosing which layers to copy from the teacher on the WER of the distilled student appears to decrease as the model size increases (hyp. 5). In contrast, copying any selection of layers from the teacher instead of using a random initialisation becomes more important.

Table 3.1: Comparison of initialisation policies for a 6-layer and 10-layer Wav2vec 2.0 student. The policy column describes which layers are copied from the teacher to initialise each student. 'Even' means layers 2,4,...,12 are chosen. 'Odd' are layers 1,3,...,11. Since there are only six even or odd layers, this policy is only available for the 6-layer student. The quoted WER is measured on the development set after knowledge distillation.

Initialisation policy	Student WER (%)	
	6-layer	10-layer
First	13.0	2.8
Middle	**4.7**	2.9
Last	8.1	**2.7**
Even	6.5	-
Odd	6.7	-
Random	23.9	19.7

3.5 Summary

In this chapter, I investigated the impact of initialising a Wav2vec 2.0 student model by copying different combinations of teacher layers with the aim of identifying and understanding the properties of a good layer selection. Five hypotheses on the relative importance of different layers were outlined and tested on 2-layer, 6-layer and 10-layer students.

The results show that copying central layers from the teacher improves the student's performance after distillation, supporting hypothesis 3. Evidence supports the hypothesis that maintaining the original order of layers when copying leads to better accuracy (hyp. 4). However, the results for hypotheses 1 and 2 were inconclusive, as other factors may play a more significant role in determining the effectiveness of an initialisation strategy.

Extending these hypotheses from the 2-layer students to larger models, I found that the importance of layer selection decreases. This especially holds when distilling to a 10-layer student, which showed no diminishing performance differences across initialisations. This supports hypothesis 5 and suggests that larger models may be more robust to variations in initialisation strategies.

The best initialisations found for each layer size are (5,8), middle layers and last layers, achieving WERs of 21.1%, 4.7% and 2.7% on the development set.

In summary, this chapter provided valuable insights into the impact that initialisation strategies have on the distillation of Wav2vec 2.0 models. These results are used in Section 5.6 where they are evaluated against the efforts of distilling to CNNs in the context of resource-constrained environments.

Finally, I hope that these findings can contribute to future research on knowledge distillation towards more efficient and accurate ASR systems.

Chapter 4

Distilling to shorter-output models

In this chapter, I propose a novel method for distilling CTC-based ASR models to students with a shorter output length. Specifically, this approach is designed to distil Wav2Vec 2.0 into ContextNet and Citrinet, as these models each employ four subsampling layers in their encoders, resulting in a $4\times$ shorter output sequence than Wav2Vec 2.0. The method aims to enable efficient knowledge transfer from the teacher model (Wav2Vec 2.0) to the student models (ContextNet and Citrinet) while maintaining a low WER, despite the difference in output lengths.

I begin by discussing prior work on knowledge distillation of CTC-based ASR models to shorter-output students, providing a motivation for why this problem should be addressed, followed by a more formal problem statement. Next, I will present a trivial solution before delving into more sophisticated approaches, including token pooling, discounted pooling, and CNNs for learnable down-sampling. Finally, a Viterbi-based alignment algorithm and its variations are introduced, for which a detailed evaluation is available in the experiments chapter (ch. 5). This structure aims to guide the reader through the variety of down-sampling approaches for CTC-based models, offering a comprehensive understanding of the proposed algorithm for distilling to models with shorter output lengths.

4.1 Prior work

This section on prior work is brief, as, to the best of my knowledge, thus far only one publication has distilled CTC-based neural speech recognition systems to shorter-output students. Lee et al. (2022) have compressed Wav2vec 2.0 by downsampling the feature extractor's output (which serves as the transformer encoder's input) using a CNN layer. This allowed the transformer layers to be reduced to a shorter sequence length, thereby removing parameters. Since this thinner model has a shorter output length, Lee et al. use a deconvolutional layer (Zeiler et al., 2010) to extend the output sequence to the original Wav2vec 2.0 length. While they show that a deconvolutional layer indeed works acceptably well for distilling to shorter-output models, their method requires the head to stay attached, thereby modifying the student. A general solution to the problem, however, should be possible without extending the student and increasing its size and

worsening its inference time.

Sharing a similar objective to mine, Yoon et al. (2021) work towards enabling cross-architecture knowledge distillation for a more flexible compression to more efficient architectures. They do this for the distillation between CTC-based convolutional and recurrent neural networks . Although their approach appears to be effective, it has a notable drawback: the output sequences of the teacher and student must be of equal length.

4.2 Problem statement

Thus far, I have only briefly mentioned the challenge of knowledge distillation from a CTC-based teacher to a student model that outputs fewer tokens without elaborating on the reason. The issue arises specifically because the KD-Loss (see eq. 2.4) individually compares each student frame to a corresponding teacher frame, aiming to make the former as similar as possible to the latter. The loss is computed by summing over the KL-divergence between each frame pair.

When the teacher and the student have an equal number of frames, it is straightforward to align the frames: the first student frame matches the first teacher frame, the second matches the second, etc. However, when the teacher outputs N frames and the student outputs $M < N$ frames, we have to explicitly decide which frames to pair. Concretely, for each student output frame $\boldsymbol{y}_i^S$ with $i = 1, ..., M$, how do we determine the target $\boldsymbol{y}_i^*$ from the teacher, in such a way that the student model learns the most optimal output and achieves the lowest possible Word Error Rate?

In this dissertation, I focus on the case where $M \approx N/4$ (the ratio between Wav2vec 2.0 and ContextNet / Citrinet). Nevertheless, the proposed approaches should be applicable to other teacher-student output ratios as well, including situations where the ratio between the output lengths is not fixed.

4.3 Trivial solution: Choosing the closest frame

If we assume that the output frames linearly map onto the input frames, then we can determine the input timeframe associated with each output frame. Following this notion, an output frame should produce the label that approximately represents the phone from its corresponding input frame. As a result, the output frames corresponding to identical input frames in various models should be similar. Therefore, to identify a teacher frame that matches the output frame of the shorter students, we can simply determine the time points linked to the outputs of each model and match the student and each student frames with the closest teacher frame.

However, as shown in Fig. 4.1, the output from a CTC model tends to be very spikey, meaning that most output frames contain a `<PAD>` or `<BLANK>` token with only a few frames associating a high probability to a non-pad token (label). Further, Fig. 4.1 also shows that an emitted label is often not emitted in the middle of its corresponding sound, but can be emitted when the speaker starts or stops pronouncing the phone.

When $M \approx N/4$, four teacher frames are output for every student frame, so choosing the closest teacher frame is equivalent to sampling every fourth frame. Let us make the slightly simplified assumption that only a single label is emitted for each corresponding sound, which is randomly located within the time when the speaker utters the sound. The average character takes an English speaker 60 ms to read out (Trauzettel-Klosinski and Dietz, 2012), which corresponds to 4.5 Wav2vec 2.0 frames with a 25 ms window and 10 ms. Thus, when sampling one in four (M/N) frames, where every group of four frames correspond to one average spoken character, and a label is output once in each such group, the probability of sampling every label is given by $p = (M/N)^M$. In the WSJ dataset, an utterance gets split into an average of $N = 783$ frames. Thus, for the distillation from Wav2vec 2.0 to ContextNet or Citrinet, where $M \approx N/4$, the probability of sampling every label from the teacher is $p = (1/4)^{783/4} = 1.4 \times 10^{-118} \approx 0$. Even if we consider it acceptable to only extract at least half of all labels output by the teacher, $p = (1/4)^{783/8} = 1.2 \times 10^{-59}$ per utterance.

This poses a problem because the student learns to mimic the extracted output, which will consist mostly of frames that assign a high probability to <PAD> tokens (pad-frames). Relaxing the above assumptions does not improve the calculated odds enough to make this a viable alignment method.

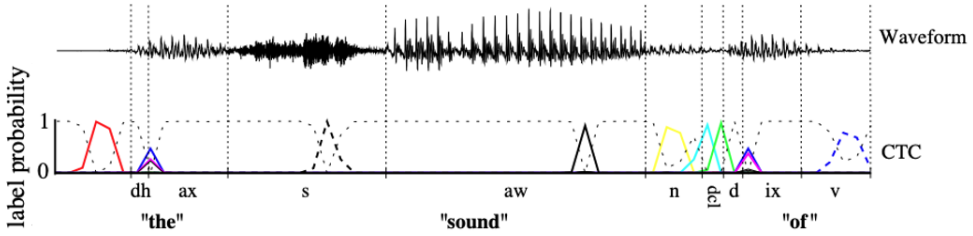


Figure 4.1: Exemplary visualisation of the spike times of a model trained on a CTC loss. The different coloured lines show the probability that a specific token is output at a given time. The faint dotted line shows the probability of outputting a <PAD> at each time step. The phonemes and words heard at each time interval are written underneath. The Figure has been adapted from Alex Graves et al. (2006).

4.4 Max pooling: Choosing the best of four frames

As we have determined above that we can split the output into groups of 4, where each group approximately corresponds to one spoken character, we can simplify the problem from choosing the best M frames to choosing the best frame for each group of 4. *Max pooling* uses the most confident frame, where the probability associated with a single non-pad token is the highest.

This method is expected to greatly outperform the previous trivial method, as the probability of choosing a frame for each group that outputs the corresponding character's label increases to 1. Thus, the probability of extracting at least one frame for each label increases to $p = 1^{783/4} = 1$.

The improvement to $p = 1$, however, is under the assumption that any set of 4 consecutive frames contains one or more sounds that together correspond to at most a single character. It is quickly obvious that this is not the case. For instance, when reading the word `choice` the character pairs `ch` and `ce` each only produce a single sound.

When only the single frame with the highest confidence is chosen, all other frames within the group get discarded, even if they may predict other important labels. For the utterance `choice`, two adjacent frames within the same group may predict `c` and subsequently `h`, but only one of them gets extracted. If this pattern is consistent across multiple utterances, the student may learn incorrect sound-character correspondences. In the above example, it could learn that the `ch` sound corresponds to the single character `h`.

4.5 Discounted pooling: Choosing a bit of everything

To solve this issue of forgotten frames, I introduce *average pooling* where not only a single frame is chosen, the average probability distribution over tokens is calculated across all frames within a group. Averaging allows us to include information about all the outputs of the teacher instead of only a sample.

However, if a group contains three pad-frames and only one frame that associates a high probability to a non-pad token, the averaged probability distribution will be biased towards the `<PAD>` . Thus, the student is taught that the sound corresponds to a pad-token with a low probability of being something else. Just like max pooling improved on the trivial method, we can improve on average pooling by averaging only over frames predominantly predicting non-pad tokens.

As a generalisation of this, I propose *discounted pooling*, where all frames in a group are averaged, but the weight of pad-frames is reduced by a discount factor $f_{discount}$.

Thus, each target frame $\boldsymbol{y}_i^*$, for which the KL-divergence $D_{KL}(\boldsymbol{y}_i^* \| \boldsymbol{y}_i^S)$ is calculated, is given by

$$\boldsymbol{y}_i^* = \frac{1}{\| \sum_j d(\boldsymbol{y}_j^T) \|_1} \sum_{j=4(i-1)}^{4i} d(\boldsymbol{y}_j^T) \tag{4.1}$$

where the discounting function is

$$d(\boldsymbol{y}) = \begin{cases} \boldsymbol{y} / f_{discount} & \text{if argmax}(\boldsymbol{y}) = \texttt{<PAD>} \\ \boldsymbol{y} & \text{otherwise} \end{cases}$$

This is a generalisation of average pooling, as setting $f_{discount} = 1$ is equivalent to basic averaging while taking the limit as $f_{discount} \to \infty$ is equivalent to ignoring all frames that predict a `<PAD>` . Any other discount factor $1 < f_{discount} < \infty$ includes the pad-frames but puts a greater emphasis on non-pad-frames. This may be the best choice, as some pad-frames likely still contain relevant information. The ratio of pad to non-pad frames

would also be encoded. A summary of interpretations of the discount factor is shown in Table 4.1.

Table 4.1: Short summary of interpretations of different discount factors. A more detailed explanation is found in Section 4.5.

$f_{discount}$	Interpretation
1	Average pooling
$1 < f < \infty$	Discounting <PAD>
∞	Ignoring <PAD>

While discounted pooling is a substantial improvement over choosing the closest teacher frame, it still has difficulties with repeated characters. In CTC, a repeated character, such as LL in HELLO is produced, by outputting L, followed by one or more <PAD>s , followed by another L. The pad-tokens in the middle are important, as the CTC decoder reduces any frames that repeat the same character to a single instance of that character. If the frame sequence outputting L**L (* representing <PAD>) is grouped and pooled, it is indistinguishable from the sequence LL**. Thus, the student may in some instances learn to produce only a single character whereas the teacher knows to output two. The following methods all deal with this issue.

4.6 Dynamic: Squeeze to fit

As the problem of merging characters appears to be inherent to the pooling mechanism, the new *dynamic* method departs from this approach. It instead replicates a greedy CTC decoder incrementally until the resulting sequence reaches the student's output length.

1. Remove pad-frames that are next to other pad-frames and non-pad-frames that are next to frames outputting the same token. Choose the frames to remove such that all groups (where all frames output the same token) have a similar $n_{removed} : n_{original}$ ratio.

   Remove frames from each group such that Removed frames from the longest sequences (outputting the same token) first.

2. Remove pad-frames whose neighbours are different (e.g. E*L in H*E*L*L*O).

This maximally reduces the number of frames while still decoding to the same string.

If the sequence is still longer than the student's output:

3. Remove all remaining pad-frames. These should be those whose two neighbours produce the same token.

4. Remove all remaining tokens.

In particular, this method focuses on retaining the <PAD> between frames that output the same token to teach the student that a sound corresponds to a repeated token in the decoded string instead of only one. By ensuring that the $n_{removed} : n_{original}$ ratio

stays roughly equal between the groups, we try to minimise time warping introduced by removing frames at irregular intervals.

4.7 CNN: Learning to choose

An alternative approach, which was also used by Lee et al. (2022), is to bridge the difference in output lengths with a CNN layer. While the authors use a deconvolutional layer that increases the student's output length to match the teacher, I use a convolutional head on the teacher to match the student. I do this to keep the student's architecture constant for comparability with other distillation methods, but I expect the effectiveness to be similar.

The CNN head is attached to the final teacher layer and uses a stride of 4. The kernel size is set to 4 for comparison with the pooling methods. The teacher is fine-tuned with a CTC loss. Using the downsampling CNN, the new teacher's output now has the same length as that of the student and we can proceed with standard knowledge distillation.

4.8 Aligning: Letting the student choose

While all of the above methods only consider the teacher when choosing the frames, the primary goal is to improve the student by copying the teacher. As our student is pre-trained and already correctly predicts some tokens, it may be useful to take advantage of the student and let it inform our decisions on how to align frames.

4.8.1 Alignment algorithm

The alignment process works by computing a similarity matrix $A \in \mathbb{R}^{M \times N}$ where each element is the dot product similarity between the ith student frame and jth teacher frame: $a_{ij} = \boldsymbol{y}_i^S \cdot \boldsymbol{y}_j^T$. In matrix notation this is

$$A = S\,T^{\mathsf{T}}$$

where $S = \left[\boldsymbol{y}_1^S, ..., \boldsymbol{y}_M^S\right]$ and $T = \left[\boldsymbol{y}_1^T, ..., \boldsymbol{y}_N^T\right]$.

To determine the optimal alignment, we find a path through A (from a_{11} to a_{MN}) that passes through the elements with the highest similarity. An example is shown in Fig. 4.2.

4.8.2 Making aligning tractable

In terms of computational cost, this alignment method is significantly more expensive than the other mechanisms described above. While calculating the similarity matrix A is a simple multiplication that can be offloaded to the GPU, finding the optimal (maximum similarity) path is a typical dynamic programming algorithm, which has a time complexity of $O(M \times N)$. In comparison, the trivial, pooling and dynamic algorithms have a complexity of $O(N)$.

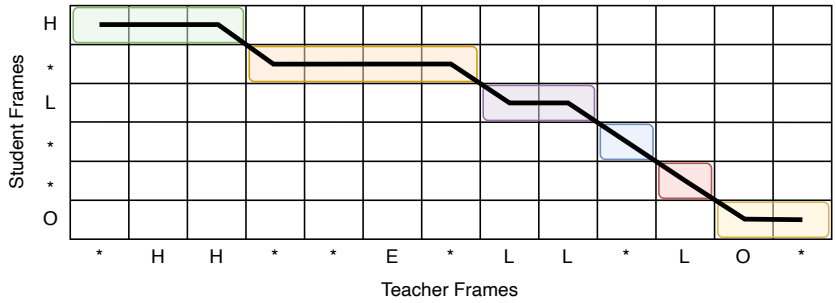


Figure 4.2: Diagram showing an exemplary maximum similarity path between the teacher frames on the bottom and the student frames on the left. The teacher output correctly decodes to the string “HELLO”, while the student incorrectly outputs “HLO” and is being trained using the alignment method described in Section 4.8. Each cell is a pairing between a teacher and a student frame and is filled with the dot-product similarity between the frames. The black line shows the best path from the first frame of the student and teacher to eaches last frame while passing through cells where the corresponding frames are maximally similar. The coloured cells are the groups of teacher frames determined by the alignment to be matched to a single student frame.

To improve upon this, we can use the observation that movement through the matrix has to be either to the right or diagonally to the bottom right see Fig. 4.2). We cannot move vertically, as this could mean that a student frame has no target frame. Therefore, a cell’s value (as per the Viterbi algorithm) is only dependent on the cells to its left. By incrementally calculating the values of each column from the left, we can parallelise computing the value of each cell in a column, as they are only dependent on the previous column which has already been computed. This can further be offloaded to the GPU to reduce the effective time complexity to $O(N)$. The training time (20 epochs, batch size 4, on an NVIDIA GeForce RTX 2080 Ti) is reduced from approximately 60 days to 40 hours.

4.8.3 Using the alignment

Once we have computed an alignment between the student and teacher output, this splits the teacher frames into M groups of varying length, as seen in Fig. 4.2. We can apply the pooling methods from above to compress these groups down to a single target frame. As before, average pooling on groups that predominantly consist of pad-frames, will likely cause the group’s label(s) to be outweighed. The average of the second group (orange) in Figure 4.2 will have a high probability associated with `<PAD>` and a low probability for `E`. This is undesirable, as we want to teach the student to output an `E` at the corresponding frame.

As a characteristic of the alignment process is that each group tends to only contain a single label (bad alignments may have more) and some pad-frames, this suggests that discounted or max pooling may lend themselves well for compressing these groups. An evaluation of the pooling methods can be found in Section 5.4.6.

4.8.4 Ignoring pad-frames (again)

Since the alignment algorithm finds the path through the matrix that passes through the maximum number of similar frames and both the student and teacher tend to output mostly <PAD> tokens, the algorithm may occasionally choose to not align a student and teacher frame with the same label in favour of maximising the number of aligned pad-frames.

To counter this undesirable side-effect, we can again ignore <PAD>s . However, as pad-frames may still associate a relevant probability to other tokens, simply setting their similarity to 0 is likely not optimal. The new similarity matrix $A' = S_{-pad} T_{-pad}^{\mathsf{T}}$ is computed from the S and T matrices that have the row corresponding to the <PAD> token removed.

A more detailed analysis and the impact of using the non-pad alignment instead of the default alignment are presented in section 5.4.5.

4.9 Summary

This chapter addresses the main challenge with distilling knowledge to models with shorter output lengths: how to effectively align and compress the teacher's longer output to match the student's shorter output without losing crucial information. Overcoming this challenge is an important step towards enabling knowledge transfer between ASR models with different output lengths while maintaining a low word error rate.

In this chapter, I began with a brief review of prior work, provided motivation for tackling this problem, and formalised the problem statement. Subsequently, I explored various approaches to aligning the teacher's output with the student's, ranging from the trivial solution of selecting the closest frame to more advanced mechanisms such as different types of pooling, dynamic frame removal while preserving the output structure, and employing a CNN for learnable down-sampling. Further, I introduced several variations of a Viterbi-based alignment algorithm that considers both the teacher and student output during alignment. Due to the algorithm's higher time complexity compared to the other methods, I implemented parallelisation to reduce the effective complexity from quadratic to linear as a function of the audio length.

The methods proposed here are explored further in the following chapter, where their effectiveness and limitations are evaluated.

Chapter 5

Experiments on distilling to smaller models

To compress Wav2vec 2.0 for automatic speech recognition into more efficient models with fewer parameters and faster inference time, the preceding two chapters have focused on shrinking the Wav2vec 2.0 architecture as well as establishing the foundation for distilling to smaller CNN-based networks. In particular, I have introduced an initialisation approach for students derived from Wav2vec 2.0, by improving on previous efforts. Furthermore, I have suggested a collection of algorithms to effectively subsample the teacher output to match the student's output length to enable distilling to the shorter-output Citirnet and ContexNet models. These CNNs are of particular interest, as they have recently been shown to achieve similar word error rates to transformer models such as Wav2vec 2.0-BASE. I hope to further improve the CNNs' performance through knowledge distillation from the mentioned transformer.

This chapter details the experiments that were run to evaluate the subsampling algorithms proposed in the previous chapter and compares the results from distilling Wav2vec 2.0 to CNNs to distilling Wav2vec 2.0 to shallower versions of itself. Further, the architectures' performance is compared and the distillation approaches are evaluated in the context of end-to-end ASR in resource-constrained environments.

The chapter begins by describing the experimental setup that was introduced in Section 3.3 in more detail and presenting the baselines. Following that, the first distillation attempt from Wav2vec 2.0 to Quartznet is made and the methods for distilling to shorter-output models are evaluated using Citrinet and ContextNet. Particular focus is on the Alignment distillation algorithm that incorporates the student into subsampling the teacher's output. In a round of final experiments, each architecture's number of parameters as well as CPU and GPU inference times are measured to be evaluated in the concluding chapter.

5.1 Experimental setup

5.1.1 Dataset: WSJ

As previously mentioned in Section 3.3, the dataset used in this set of experiments is the Wall Street Journal (WSJ) speech dataset (Paul and Baker, 1992). The *WSJ0* and *WSJ1* releases have been merged to give a total of 81.5 hours of newspaper articles read by 283 individuals from the United States in the joint training set. The releases' testing sets are `eval92` with 8 speakers reading a total of 42 minutes and `dev93` with 10 speakers reading a little over 1 hour of articles.

Since no development set is provided, I split the joint training set into a new pair of training and dev-set. Specifically, as the networks are supposed to be both speaker-independent and able to recognise and transcribe previously unseen words, the utterances in the `eval92` and `dev93` test sets are from different news articles to the training set and from different speakers. This should also be the case for the new dev set. However, as I have found no split such that no speakers or extracts overlap with the training set, I have settled on splitting along the speaker boundary but keeping overlapping transcripts. This causes the number of unseen, out-of-vocabulary words to be much lower in the dev set than in the test sets and will likely lead to the models performing slightly better on the dev set. Nonetheless, when taking the variability into account and acknowledging that the dev-set will not recognise overfitting on transcripts very well, this split is better than not using a development set at all.

Table 5.1: Statistics on the internal training and development sets and official testing sets from the Wall Street Journal speech recognition dataset. Details on how the training and development sets are built are found in Section 5.1.1.

Dataset	Time	Speakers	Utterances	Out of vocabulary	Overlapping transcripts
Training	80h 28min	279	36,908	-	-
Dev	1h 1min	4	508	0.6%	75%
`eval92`	42min	8	333	12.1%	0%
`dev93`	1h 5min	10	503	14.3%	0%

5.1.2 Implementation

The implementation of the experiments uses PyTorch (Paszke et al., 2019) with a highly modified version of the Hugging Face Trainer (Wolf et al., 2020). In particular, I have added an extension for knowledge distillation and custom logging. As some of the models use the NVIDIA NeMo toolkit, I have also added functionality to train models built from this toolkit to reduce variability between the models and improve their comparability.

The dataset loader is also a custom implementation that aims to reduce the load on the file server of the compute cluster that I ran the experiments on.

The pre-trained Wav2vec 2.0-BASE model used in the experiments is taken from the official release on Hugging Face (Facebook, 2021). The pre-trained QuartzNet, Citrinet and ContextNet models are taken from NVIDIA's catalog[1]. As their published implementation of ContextNet uses an RNN-T decoder (A. Graves, 2012), I replace the decoder with a linear layer and use a CTC loss. This maintains comparability between CTC-based models and avoids the results being influenced by the decoder.

Throughout the research process, I planned to include the model by Li et al. (2019) that my CNNs are based on and have manually implemented many parts of it, as open-source versions of it had many errors. However, since the training of the model from a random initialisation would have taken significantly too long for this dissertation and I could only find a pre-trained release of the largest version with 333M parameters (compared to the 95M of Wav2vec 2.0), it is not included here.

5.1.3 Training parameters

As in Chapter 3, the teacher and baseline is a Wav2vec 2.0-BASE model that is fine-tuned for 50 epochs with a batch size of 4, an Adam optimiser (Kingma and Ba, 2017). A grid search (see Appendix A) has determined an exponentially decaying learning rate schedule, starting at 2.6×10^{-5} and decaying by 10% every epoch.

The QuartzNet, Citrinet and ContextNet models are fine-tuned or distilled for 20 epochs, as that is when their validation loss tends to converge. Grid searches for each model have determined their learning rates to be 1.4×10^{-5}, 3×10^{-5}, and 7×10^{-6} respectively. For details see Appendix A. A learning rate schedule exponentially decaying by 10% every epoch is used.

While the examples in the previous chapter assumed that each output token is equivalent to a single character or `<PAD>` , the output length of Citrinet and ContextNet is marginally too short to produce the correct transcription character by character. Therefore, both models use a Sentencepiece tokeniser (Kudo and Richardson, 2018) as a variation of Byte Pair Encoding (Sennrich et al., 2016). For consistency during distillation, a single tokeniser is used across all models. The one trained alongside Citrinet is chosen, as that avoids retraining the pre-trained model while ContextNet needs to be fine-tuned on the new decoder anyways.

No Language Model is used, as that is beyond the focus of this series of experiments and this dissertation.

5.2 Baselines

A set of baselines are implemented as comparisons to the distilled models. They are fine-tuned from their pre-trained initialisation as described above. Their word error rates on the development set are shown in Table 5.2. As expected, the transformer-based Wav2vec 2.0 indeed attains the lowest WER. We can, however, also see that the other

[1] https://catalog.ngc.nvidia.com/

models are also reaching similarly low error rates, with Citrinet being the next-lowest and QuartzNet the highest.

Wav2vec 2.0 having the lowest dev-WER justifies the further investigation of distilling to the other models to improve their performance.

Table 5.2: Word error rates achieved by fine-tuning pre-trained versions of the table's models. WER is calculated on the development set. These WERs are taken as baselines against which the distillation efforts are evaluated. Training parameters are defined in Section 5.1.3.

Model	Dev WER (%)
Wav2vec 2.0	**2.0**
QuartzNet	4.9
Citrinet	2.8
ContextNet	4.1

5.3 Distilling to QuartzNet

With the baselines introduced, I begin with the easiest CNN to distil to: QuartzNet. While the fine-tuned baseline performs worst on the dev-set (Table 5.2), its advantage over the other two CNN-based models is that its output is the same length as that of Wav2vec 2.0, making the distillation process trivial. Thus, to calculate the KD-Loss, we can compare each student frame to the corresponding teacher frame and sum the KL-Divergences. However, as seen in Table 5.3 the baseline with a WER of 4.7% significantly outperforms the distilled model with 9.1%.

In the following sections on subsampling methods for Citrinet and ContextNet, I attempt to achieve a WER equal to or lower than that of the distilled QuartzNet. As the Citrinet and ContextNet baselines attain a slightly lower WER than QuartzNet, achieving a comparable word error rate through knowledge distillation will mean that distilling by subsampling is not much worse than distilling to the same output length.

Table 5.3: Comparison of the baseline WER for QuartzNet from Table 5.2 with the QuartzNet model trained using knowledge distillation from Wav2vec 2.0. As the student and teacher have the same output length, no subsampling is required. The word error rates are calculated on the `eval92` and `dev93` sets in addition to the development set, as the models are not tuned any further.

Model	dev	`eval92`	`dev93`
QuartzNet Baseline	4.9	9.9	11.7
QuartzNet Distilled	9.1	15.5	18.3

5.4 Distilling to ContextNet & Citrinet by subsampling

The main emphasis in this chapter is on evaluating the subsampling algorithms proposed in Chapter 4 to enable distilling to the shorter output-length models ContextNet and Citrinet. The word error rates attained by distilling from the fine-tuned Wav2vec 2.0 to the two models using different subsampling methods are shown in Table 5.4. In this section I go through the results, analysing the performance of each algorithm. I specifically focus on evaluating variations of the alignment mechanism to determine the best approach.

Table 5.4: Dev-WER of distilling Citrinet and ContextNet using various subsampling methods. The alignment algorithm for ContextNet first fine-tunes the student before distilling to it, as this is found to be the best approach (see Table 5.6). The baselines (see Section 5.2) are added for comparison.

Subsampling method	Citrinet	ContextNet
Fine-tuned baseline	2.8	4.1
Closest frame	41.1	96.6
Max pooling	27.1	25.6
Average pooling	99.0	100
Discounted pooling (50)	28.4	22.4
Dynamic	75.3	56.8
CNN head	13.1	13.9
Alignment (w/o <PAD>)	**9.7**	**7.4**

5.4.1 Closest frame distillation

In Section 4.3, where the trivial subsampling method was introduced, I hypothesised that it will sample mostly frames predicting <PAD> tokens and discard the important frames that predict non-pad labels. The student would therefore learn to only output pad-frames, as that is the most predictive of the extracted teacher sample. Looking at the WER of ContextNet using the closest frame method in Table 5.4, the 96.6% indicates that the model indeed learns to make predictions that are far from the ground truth. Looking at the prediction output of the model, it produces primarily <PAD> tokens with the occasional token, that seemingly coincidentally matches the ground truth.

While the WER for Citrinet is much worse than the fine-tuned baseline, it is far from predicting only <PAD>s . Looking at the training loss and dev-WER progression in Figure 5.1 we can clearly see that both are increasing and the model will also approach predicting almost exclusively pad-frames.

5.4.2 Distilling by pooling

As max pooling is proposed as an improvement over the trivial approach, we can confirm that it does indeed perform better than the previous method. The word error rates converge (see Fig. 5.2) at 27.1% and 25.6% for Citrinet and ContextNet respectively, as

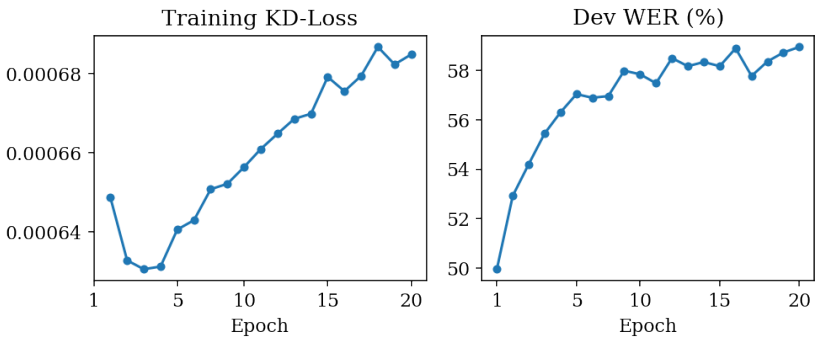


Figure 5.1: Training KD-Loss and Dev-WER measured during the knowledge distillation from Wav2vec 2.0 to Citrinet using the closest frame subsampling algorithm.

stated in Table 5.4. Interestingly, the dev-loss increases slightly while the training loss and dev-WER decrease. Training is stopped as the dev-WER converges.

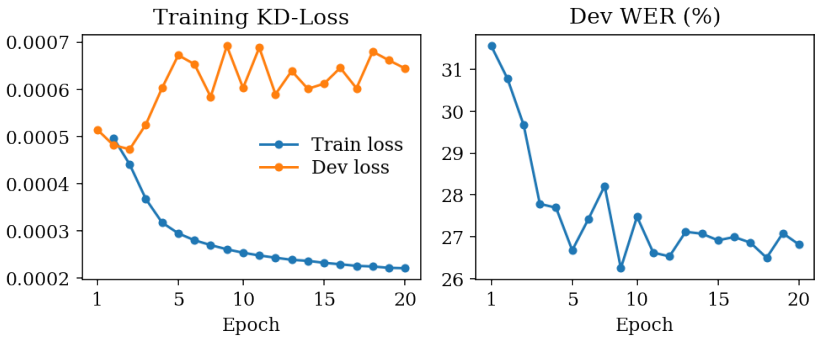


Figure 5.2: KD-Loss (of training and dev-set) and dev-WER are measured during the knowledge distillation from Wav2vec 2.0 to Citrinet using the max pooling subsampling algorithm.

To further improve upon max pooling, average pooling and discounted pooling include all frames in a group instead of only the frame with the greatest confidence (see Sections 4.4 and 4.5). As expected, average pooling (implemented as discounted pooling with $f_{discount} = 1$) converges both models to a WER of 100% by learning to predict only `<PAD>` tokens. The discounted pooling method, however, does improve on max pooling for ContextNet, though it is slightly worse for Citrinet. The used $f_{discount}$ of 50 has been determined by trying several discount factors between 1 and 100. Overall, it can be argued that the accuracy difference between max pooling and discounted pooling is negligible.

5.4.3 Dynamic distillation

The dynamic method is introduced to solve the problem that the pooling methods can neglect labels in specific situations (see Section 4.6). The method handles these cases by splitting the teacher's output into unequal groups and removing frames that do not contribute to the CTC output while trying to maintain the frames' temporal position relative to the input mostly constant. The WERs of 75.3% and 56.8% attained by

distilling to Citrinet and ContextNet using this method (see Table 5.4) indicate other factors may be more important when choosing which teacher frames to sample. The pooling methods' better performance indicates that the time warping introduced by removing frames at irregular intervals, which also vary across utterances, significantly harms the student's accuracy.

5.4.4 Distilling through a CNN head

As a slight variation on the method used by Lee et al. (2022), I experiment with attaching a learnable CNN head with a stride and kernel size of 4 onto the teacher. The CNN is fine-tuned on top of the frozen Wav2vec 2.0 teacher.

Comparing the word error rates attained by distilling through the CNN to those achieved by the pooling methods, we can see that the WER is roughly halved from 27.1% to 13.1% for Citrinet and from 25.6% to 13.9% for ContextNet. This makes the CNN head so far the best approach for distilling to shorter output-length models. However, no subsampling methods have so far come close to the fine-tuned baseline. subsampling has also not yet achieved a WER equal to or lower than that of the distilled QuartzNet, implying the problem of distilling to a shorter output length has not yet been overcome.

5.4.5 Aligning with and without padding

Subsampling by aligning makes use of the pre-trained student's output to guide the sampling of the teacher. As detailed in Section 4.8.1, a similarity matrix is computed between the student and teacher frames through which we find a path that passes through the most similar frames. This warps the time in accordance with the student instead of being solely reliant on the teacher, as in the dynamic approach (see Section 5.4.3).

In Section 4.8.4 the issue is raised that comparing the similarity between all tokens may cause the algorithm to prefer a path that primarily aligns pad-frames instead the important, label outputting frames. An example of this is shown in Figure 5.3a. To fix the issue, the similarity matrix calculation can be adjusted to ignore any similarities between `<PAD>` tokens (see Section 4.8.4). Applying the adjustment yields the alignment in Figure 5.3b. We can see that all similarities between `<PAD>`s get ignored and the path now passes through the desired label tokens.

In text form, this fixes the alignment from

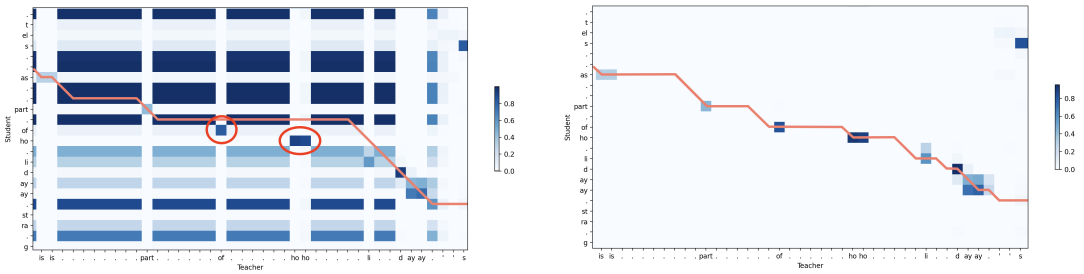
```
Teacher: .telel.sis..partof. li.. dayay'str ateg
Student: .tels ..as..part. ofho.lidayay.stra.  g
```

where, for example, the teacher's `of` aligns with a student's `<PAD>` (represented by a dot) and the student's `of` is aligned with a teacher's `<PAD>`. The improved version is

```
Teacher: .tels..is..part.ofho.lidayay'str ateg
Student: .tels..as..part.ofho.lidayay.stra.  g
```

. Spaces are added to visually align tokens that encode strings of different lengths.

The effectiveness of this change is evaluated in Table 5.5.



(a) Including <PAD> (b) Excluding <PAD>

Figure 5.3: Example of a misalignment incurred by maximising the similarity between any token pair (Figure a). Ignoring the <PAD> token when calculating frame similarities (Figure b) fixes the issue. Teacher and student outputs are on the x and y-axes respectively. A field's darkness indicates the similarity between the frame pair. The optimal path is shown in orange. The falsely non-aligned frames are encircled in red in Figure a. A fine-tuned Wav2vec 2.0 and pre-trained Citrinet are aligned.

5.4.6 Pooling of aligned groups

Since the alignment algorithm so far only matches each student frame with one or more teacher frames, the determined groups have to be reduced to a single frame. Here the same pooling methods as in Section 5.4.2 can be applied. The methods are evaluated together with the choice of computing the similarity matrix with or without <PAD> tokens, as the two mechanisms appear to be interdependent, as shown in Table 5.5.

We can see that computing the similarity matrix without <PAD>s generally performs better. However, average pooling seemingly works best when the similarity is calculated with <PAD>s , as this causes labels and pad-frames to be grouped independently (see Fig. 5.3a). This does not happen when <PAD>s are excluded from the similarity matrix, as seen in Fig. 5.3b.

Concluding the results from Table 5.5, performing max pooling on the groups determined by finding the best path through the similarity matrix computed without the <PAD> tokens gives the lowest WER on Citrinet and, by extension, on ContextNet.

Table 5.5: WERs achieved on the dev-set by distilling from Wav2vec 2.0 to Citrinet using the alignment subsampling method. The groups of teacher frames identified by the alignment are pooled using the compared methods. Two alignment variants that either include or exclude <PAD> tokens in the similarity computation (see Fig. 5.3) are also evaluated.

Pooling method	Alignment	
	with <PAD>	w/o <PAD>
Max pooling	100	**9.7**
Average pooling	**48.8**	100
Discounted pooling ($f_{discount} = 50$)	70.3	13.8

5.4.7 Aligning to ContextNet

Since Citrinet has been pre-trained for ASR, it lends itself well for distilling using the alignment algorithm. However, as the decoder of ContextNet has been switched for a randomly initialised linear layer, the model's output is unusable for alignment, as seen in Table 5.6. To find a good initialisation for alignment distillation, I pre-train ContextNet. However, as using the fine-tuned baseline from Section 5.2 defeats the purpose of this experimentation on knowledge distillation since we could just keep the baseline, I also fine-tunine ContextNet for only 1 epoch and alternatively use the result of discounted pooling. I choose discounted pooling over the CNN head method, as the latter would require an extra effort of fine-tuning the subsampling head.

To avoid a too high learning rate for the pre-trained ContextNet, I also run the experiments with the learning rate of Citrinet (7×10^{-6}).

As expected, Table 5.6 shows that the baseline initialisation achieves the best WER of 7.4%. The other two initialisations may also be used with their WERs of 11.9% and 12.2%, as they do not require prior fine-tuning.

Table 5.6: WERs achieved on the dev-set by distilling Wav2vec 2.0 to different initialisations of ContextNet. Different learning rates are tried to account for the model being pre-trained instead of having a randomly initialised docoder.

Initialisation	Learning Rate	
	3×10^{-5}	7×10^{-6}
Randomly init. decoder	100	100
Fine-tuned baseline	**10.5**	**7.4**
Fine-tuned for 1 Epoch	12.8	11.9
Discounted pooling 50	12.2	13.6

5.4.8 Summary of subsampling mechanisms

To conclude the section on distilling to ContextNet and Citrinet by subsampling from Wav2vec 2.0, I have shown in Table 5.4 that the proposed algorithms and in particular alignment are reasonable approaches for distilling to shorter-output models. Specifically, the alignment algorithm using max pooling and ignoring `<PAD>`s attains a WER that is comparable to the WER of QuartzNet on the development set. This is significant as it shows that by subsampling we can distil to shorter-output models similarly well as by using default knowledge distillation to a model with the same output length as the teacher. Looking ahead at Table 5.7, the feasibility of subsampling is confirmed by the WERs on the `eval92` and `dev93` sets.

The methods explored in this dissertation and in this series of experiments enable the effective distillation to shorter-output models and lay the foundation for future experiments in this area.

5.5 Conclusion on distilling Wav2vec 2.0 to CNNs

From the above experiments and the results on the `eval92` and `dev93` sets in Table 5.7 we can clearly see that while subsampling for knowledge distillation is effective, they do not beat the word error rates of the fine-tuned baselines for QuartzNet, Citrinet and ContextNet. While the `eval92` and `dev93` WERs of the distilled ContextNet somewhat approach the baseline's WER, most baseline error rates are approximately half of the distilled versions' rates. Thus, knowledge distillation using a frame-level KL-Divergence on the output of Wav2vec 2.0 for fine-tuning pre-trained QuartzNet, ContextNet and Citrinet, and presumably other CNN-based ASR systems, does not improve on the student's word error rates compared to fine-tuning them using a CTC-loss.

5.6 Application to resource-constrained environments

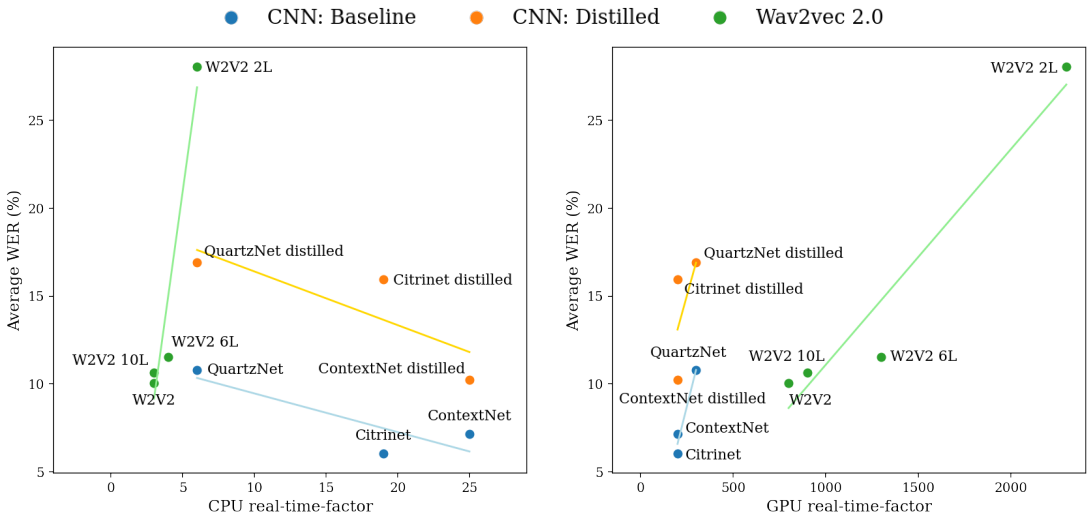
In order to evaluate the use of CNN-based ASR models in resource-constrained environments in contrast to distilling Wav2vec 2.0 to a smaller version of itself, this section compares the architectures' efficiencies. In particular, the parameter count, the inference time on one CPU core as well as the GPU inference time are considered. The trade-offs between accuracy and performance are shown in Table 5.7 and Figure 5.4. The inference times are quoted as a real-time-factor (RTF). An RTF of 10 indicates that the model transcribes an utterance $10\times$ faster than real-time.

Analysing the approximate trends for each family of models as shown in Figures 5.4a and 5.4b, we can observe differences between the trade-offs on a CPU and GPU. While the trend across CNN models is surprisingly in the direction of improving both CPU inference time and accuracy, the trend is reversed on the GPU, where the larger QuartzNet is worse but runs faster than the smaller Citrinet and ContextNet. The distilled models mirror each of these trends, although with a higher WER than the baselines. The trend when compressing Wav2vec 2.0 to fewer layers is similar for both processing units. Distilling down to 6 layers has little impact on the accuracy but improves the CPU and GPU real-time-factor by 33% and 63% respectively. The 2-layer model's accuracy, however, is greatly impacted by the compression and the improvement in performance is likely not worth it.

To minimise the impact on a host device's computational resources while maximising the transcription accuracy, it is important to consider the device's properties when deciding on the model. While edge devices with an inbuilt GPU, such as the NVIDIA Jetson product family, would benefit more from using a lightweight model that is highly performant on GPUs, such as the 6-layer Wav2vec 2.0, a CPU-only device such as a Raspberry Pi would benefit more from using a Citrinet or ContextNet, as they achieve the lowest WER on the `eval92` and `dev93` sets and have the best performance on the tested CPU. ContextNet has a lower parameter count and faster CPU inference, while Citrinet is not significantly slower despite attaining a better WER. The performance on other processing units may vary slightly.

Table 5.7: WERs achieved by the best models from each set of experiments, evaluated on the development, `eval92` and `dev93` sets. The error rates are compared against the parameter count, CPU and GPU inference speeds for each model. The inference speeds are real-time-factors (RTF), where an RTF of 10 means that the model transcribes the audio $10\times$ faster than real-time. The distilled models use the best initialisations (for Wav2vec 2.0) and subsampling methods (for ContextNet and Citrinet) determined in Sections 3.4 and 5.4 respectively.

	WER (%)					
Model	dev	`eval92`	`dev93`	# Parameters	CPU inference	GPU inference
Wav2vec 2.0	**2.0**	9.5	10.6	95.2M	3	800
Wav2vec 2.0 (10L)	2.5	9.9	11.4	81.0M	3	900
Wav2vec 2.0 (6L)	3.8	10.5	12.6	52.6M	4	1300
Wav2vec 2.0 (2L)	21.1	25.6	30.5	24.3M	6	**2300**
QuartzNet	4.9	9.9	11.7	19.9M	6	300
QuartzNet distilled	9.1	15.5	18.3			
ContextNet	4.1	6.4	7.9	**9.8M**	**25**	200
ContextNet distilled	7.4	8.9	11.6			
Citrinet	2.8	**5.1**	**7.0**	10.3M	19	200
Citrinet distilled	9.7	13.6	18.3			



(a) CPU performance – WER trade-off

(b) GPU performance – WER trade-off

Figure 5.4: Visualisation of the trade-off between the models' WERs and their inference time on a CPU (Figure a) and GPU (Figure b). The WER on the y-axis is the mean between each model's rates on `eval92` and `dev93`. The models are coloured by their families. Baseline and distilled CNNs are distinguished, as the distilled models achieve a consistently higher WER than the baseline. The trend for each family of models is approximated by a linear trendline in that family's colour. A model is preferred to be in the bottom right corner. The top left corner of each plot should be avoided.

Chapter 6

Conclusion

In this project, I explore how knowledge distillation can be used to reduce the size and inference time of existing end-to-end models for Automatic Speech Recognition, as an effort towards high-quality on-device transcription in resource-constrained environments. Experiments are carried out to test hypotheses on finding a good initialisation for students that are derived from the teacher, where Wav2vec 2.0 (Baevski et al., 2020) is used as a representative model. Further experiments on distilling from Wav2vec 2.0 to various CNN-based ASR architectures are conducted. The following contributions are made.

1. It has been identified that by copying the teacher's middle layers to the reduced student as its initialisation, we can attain the best WER after knowledge distillation. This has been shown to hold for 2-layer and 6-layer students of Wav2vec 2.0 with a reduction of the error rate by up to 50% when using the optimal initialisation. As the number of student layers is further increased to 10, the impact of choosing which layers to copy is diminished, as long as layers are copied.

2. A range of subsampling methods have been proposed and evaluated for knowledge distillation to students with shorter output lengths. By aligning the teacher frames to the student output using a non-pad similarity matrix and reducing the teacher frames using max pooling we can distil to shorter output length models equally well as to models of the same output length as the teacher. This is useful outside of this dissertation for more flexible cross-architecture knowledge distillation of CTC-based models including for the exploration of thinner architectures with shorter output lengths.

3. The effectiveness of knowledge distillation from Wav2vec 2.0 (as a representative transformer) to a variety of CNNs with and without subsampling has been evaluated, finding that it does not achieve better accuracy than fine-tuning using CTC.

4. The trade-off between achieving a low WER and a fast inference on CPUs and GPUs is evaluated, finding that CNNs such as ContextNet or Citrinet are well suited for resource-constrained environments, as they attain the best accuracy and the fastest inference on CPUs among the tested models. In contrast, a 6-

layer Wav2vec 2.0 distilled from a well-copied initialisation achieves a fast GPU inference time while maintaining a low WER.

6.1 Limitations

As this project was limited in time and resources, I have incurred the following limitations.

1. I have shown for Wav2vec 2.0 that a student can be initialised well by copying the middle layers from the teacher. Whether this holds for other architectures and other speech processing tasks as well as other deep learning fields remains open.
2. As a limitation of the WSJ dataset, my development set had a significant vocabulary and transcript overlap with the training set, impacting its generalisation to not only the `eval92` and `dev93` sets but also across other datasets. This is most importantly seen in Table 5.7 where Wav2vec 2.0 achieves a lower WER than ContextNet and Citrinet on the development set, justifying the distillation from the former to the latter. However, testing on `eval92` and `dev93` reveals that in fact ContextNet and Citrinet attain a better accuracy.

6.2 Future work

As an extension of this dissertation, the following work can be done in the future.

1. An extension of the initialisation strategy can be explored for different model architectures in ASR, across other speech processing tasks as well as other deep learning fields. A generalised initialisation recommendation for knowledge distillation students of the same architecture as the teacher would a useful contribution.
2. The subsampling algorithm can be extended to a *supersampling* that enables knowledge distillation to longer-output models. This may be useful for distilling from thinner architectures or from architectures with more downsampling layers. The proposed algorithms can be used as a starting point.
3. The experiments on initialisation have found that by copying the teacher's layers, some information is transferred to the student, as opposed to a random initialisation. By flipping the knowledge transfer such that a small model is first trained and subsequently some layers copied to initialise a larger network we may be able to train a large neural network faster while possibly sacrificing accuracy.

Bibliography

Baevski, Alexei et al. (Oct. 2020). *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. arXiv:2006.11477 [cs, eess]. DOI: 10.48550/arXiv.2006.11477. URL: http://arxiv.org/abs/2006.11477 (visited on 10/22/2022).

Benito, Diego de et al. (June 2019). "Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset". In: *EURASIP Journal on Audio, Speech, and Music Processing* 2019. DOI: 10.1186/s13636-019-0152-1.

Bucilă, Cristian, Rich Caruana, and Alexandru Niculescu-Mizil (2006). "Model Compression". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '06. event-place: Philadelphia, PA, USA. New York, NY, USA: Association for Computing Machinery, pp. 535–541. ISBN: 1-59593-339-5. DOI: 10.1145/1150402.1150464. URL: https://doi.org/10.1145/1150402.1150464.

Chang, Heng-Jui, Shu-wen Yang, and Hung-yi Lee (Apr. 2022). *DistilHuBERT: Speech Representation Learning by Layer-wise Distillation of Hidden-unit BERT*. arXiv:2110.01900 [cs, eess]. DOI: 10.48550/arXiv.2110.01900. URL: http://arxiv.org/abs/2110.01900 (visited on 10/24/2022).

Chia, Yew Ken, Sam Witteveen, and Martin Andrews (Sept. 2019). *Transformer to CNN: Label-scarce distillation for efficient text classification*. arXiv:1909.03508 [cs, stat]. DOI: 10.48550/arXiv.1909.03508. URL: http://arxiv.org/abs/1909.03508 (visited on 10/24/2022).

Collobert, Ronan, Christian Puhrsch, and Gabriel Synnaeve (Sept. 2016). *Wav2Letter: an End-to-End ConvNet-based Speech Recognition System*. arXiv:1609.03193 [cs]. DOI: 10.48550/arXiv.1609.03193. (Visited on 03/24/2023).

Davis, K. H., R. Biddulph, and S. Balashek (Nov. 1952). "Automatic Recognition of Spoken Digits". en. In: *The Journal of the Acoustical Society of America* 24.6, pp. 637–642. ISSN: 0001-4966. DOI: 10.1121/1.1906946. URL: http://asa.scitation.org/doi/10.1121/1.1906946 (visited on 03/22/2023).

Facebook (Jan. 2021). *facebook/wav2vec2-base-960h · Hugging Face*. URL: https://huggingface.co/facebook/wav2vec2-base-960h (visited on 04/13/2023).

Facebook-Research (Dec. 2022). *fairseq/examples/wav2vec at main · facebookresearch/fairseq*. en. URL: https://github.com/facebookresearch/fairseq (visited on 03/25/2023).

Fu, Yanzhe et al. (Mar. 2023). *DistillW2V2: A Small and Streaming Wav2vec 2.0 Based ASR Model*. arXiv:2303.09278 [cs, eess]. URL: http://arxiv.org/abs/2303.09278 (visited on 03/29/2023).

Graves, A. (Nov. 2012). "Sequence Transduction with Recurrent Neural Networks". In: *ArXiv*. (Visited on 03/25/2023).

Graves, Alex et al. (2006). "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. event-place: Pittsburgh, Pennsylvania, USA. New York, NY, USA: Association for Computing Machinery, pp. 369–376. ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

Gulati, Anmol et al. (May 2020). *Conformer: Convolution-augmented Transformer for Speech Recognition*. arXiv:2005.08100 [cs, eess]. DOI: 10.48550/arXiv.2005.08100. URL: http://arxiv.org/abs/2005.08100 (visited on 10/24/2022).

Han, Wei et al. (May 2020). *ContextNet: Improving Convolutional Neural Networks for Automatic Speech Recognition with Global Context*. arXiv:2005.03191 [cs, eess]. DOI: 10.48550/arXiv.2005.03191. URL: http://arxiv.org/abs/2005.03191 (visited on 10/23/2022).

He, Kaiming et al. (Dec. 2015). *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs]. URL: http://arxiv.org/abs/1512.03385 (visited on 03/24/2023).

Hendrycks, Dan and Kevin Gimpel (June 2016). "Gaussian Error Linear Units (GELUs)". In: *arXiv: Learning*. (Visited on 03/23/2023).

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (Mar. 2015). *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531 [cs, stat]. DOI: 10.48550/arXiv.1503.02531. URL: http://arxiv.org/abs/1503.02531 (visited on 10/24/2022).

Howard, Andrew G. et al. (Apr. 2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv:1704.04861 [cs]. URL: http://arxiv.org/abs/1704.04861 (visited on 03/24/2023).

Hsu, Wei-Ning et al. (June 2021). *HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units*. arXiv:2106.07447 [cs, eess]. DOI: 10.48550/arXiv.2106.07447. URL: http://arxiv.org/abs/2106.07447 (visited on 10/22/2022).

Hu, Jie et al. (Aug. 2020). "Squeeze-and-Excitation Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.8, pp. 2011–2023. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2019.2913372. URL: https://ieeexplore.ieee.org/document/8701503/ (visited on 03/24/2023).

Ioffe, Sergey and Christian Szegedy (Mar. 2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167 [cs]. URL: http://arxiv.org/abs/1502.03167 (visited on 03/24/2023).

Kingma, Diederik P. and Jimmy Ba (Jan. 2017). *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 04/13/2023).

Kriman, Samuel et al. (Oct. 2019). *QuartzNet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions*. arXiv:1910.10261 [eess]. URL: http://arxiv.org/abs/1910.10261 (visited on 03/22/2023).

Kudo, Taku and John Richardson (Aug. 2018). *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. arXiv:1808.06226 [cs]. URL: http://arxiv.org/abs/1808.06226 (visited on 03/31/2023).

Lecun, Y. et al. (Nov. 1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791. URL: http://ieeexplore.ieee.org/document/726791/ (visited on 03/24/2023).

Lee, Yeonghyeon et al. (July 2022). *FitHuBERT: Going Thinner and Deeper for Knowledge Distillation of Speech Self-Supervised Learning*. arXiv:2207.00555 [cs, eess]. DOI: 10.48550/arXiv.2207.00555. URL: http://arxiv.org/abs/2207.00555 (visited on 10/24/2022).

Li, Jason et al. (Aug. 2019). *Jasper: An End-to-End Convolutional Neural Acoustic Model*. arXiv:1904.03288 [cs, eess]. DOI: 10.48550/arXiv.1904.03288. URL: http://arxiv.org/abs/1904.03288 (visited on 10/23/2022).

Liu, Yufan et al. (July 2022). *Cross-Architecture Knowledge Distillation*. URL: http://arxiv.org/abs/2207.05273 (visited on 10/17/2022).

Majumdar, Somshubra et al. (Apr. 2021). *Citrinet: Closing the Gap between Non-Autoregressive and Autoregressive End-to-End Models for Automatic Speech Recognition*. arXiv:2104.01721 [eess]. URL: http://arxiv.org/abs/2104.01721 (visited on 03/22/2023).

Nair, Vinod and Geoffrey E. Hinton (June 2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: (visited on 03/24/2023).

NVIDIA (June 2021a). *STT En Citrinet 256 — NVIDIA NGC*. en. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/stt_en_citrinet_256 (visited on 03/25/2023).

– (Sept. 2021b). *STT En ContextNet 256 MLS — NVIDIA NGC*. en. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/stt_en_contextnet_256_mls (visited on 03/25/2023).

– (Mar. 2022). *STT En Quartznet15x5 — NVIDIA NGC*. en. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/nemo/models/stt_en_quartznet15x5 (visited on 03/25/2023).

Papers with Code (2022). *LibriSpeech test-clean Benchmark (Speech Recognition)*. en. URL: https://paperswithcode.com/sota/speech-recognition-on-librispeech-test-clean (visited on 10/24/2022).

Pasad, Ankita, Ju-Chieh Chou, and Karen Livescu (Dec. 2022). *Layer-wise Analysis of a Self-supervised Speech Representation Model*. arXiv:2107.04734 [cs, eess]. URL: http://arxiv.org/abs/2107.04734 (visited on 03/29/2023).

Paszke, Adam et al. (Dec. 2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv:1912.01703 [cs, stat]. URL: http://arxiv.org/abs/1912.01703 (visited on 04/13/2023).

Paul, Douglas B. and Janet M. Baker (1992). "The design for the wall street journal-based CSR corpus". en. In: *Proceedings of the workshop on Speech and Natural Language - HLT '91*. Harriman, New York: Association for Computational Linguistics, p. 357. ISBN: 9781558602724. DOI: 10.3115/1075527.1075614. URL:

http://portal.acm.org/citation.cfm?doid=1075527.1075614 (visited on 04/13/2023).

Peng, Zilun et al. (2021). "Shrinking Bigfoot: Reducing wav2vec 2.0 footprint". en. In: *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*. Virtual: Association for Computational Linguistics, pp. 134–141. DOI: 10.18653/v1/2021.sustainlp-1.14. URL: https://aclanthology.org/2021.sustainlp-1.14 (visited on 03/29/2023).

Roger, Vincent, Jérôme Farinas, and Julien Pinquier (Aug. 2022). "Deep neural networks for automatic speech processing: a survey from large corpora to limited data". en. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2022.1, p. 19. ISSN: 1687-4722. DOI: 10.1186/s13636-022-00251-w. URL: https://asmp-eurasipjournals.springeropen.com/articles/10.1186/s13636-022-00251-w (visited on 04/13/2023).

Sanh, Victor et al. (Feb. 2020). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv:1910.01108 [cs]. URL: http://arxiv.org/abs/1910.01108 (visited on 03/29/2023).

Sennrich, Rico, Barry Haddow, and Alexandra Birch (June 2016). *Neural Machine Translation of Rare Words with Subword Units*. arXiv:1508.07909 [cs]. URL: http://arxiv.org/abs/1508.07909 (visited on 03/31/2023).

Stevens, S. S., J. Volkmann, and E. B. Newman (Jan. 1937). "A Scale for the Measurement of the Psychological Magnitude Pitch". en. In: *The Journal of the Acoustical Society of America* 8.3, pp. 185–190. ISSN: 0001-4966. DOI: 10.1121/1.1915893. URL: http://asa.scitation.org/doi/10.1121/1.1915893 (visited on 03/22/2023).

Trauzettel-Klosinski, Susanne and Klaus Dietz (Aug. 2012). "Standardized Assessment of Reading Performance: The New International Reading Speed Texts IReST". en. In: *Investigative Opthalmology & Visual Science* 53.9, p. 5452. ISSN: 1552-5783. DOI: 10.1167/iovs.11-8284. URL: http://iovs.arvojournals.org/article.aspx?doi=10.1167/iovs.11-8284 (visited on 04/06/2023).

Vaswani, Ashish et al. (Dec. 2017). *Attention Is All You Need*. arXiv:1706.03762 [cs]. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 10/23/2022).

Wang, Dong, Xiaodong Wang, and Shaohe Lv (Aug. 2019). "An Overview of End-to-End Automatic Speech Recognition". en. In: *Symmetry* 11.8, p. 1018. ISSN: 2073-8994. DOI: 10.3390/sym11081018. URL: https://www.mdpi.com/2073-8994/11/8/1018 (visited on 04/13/2023).

Wolf, Thomas et al. (July 2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv:1910.03771 [cs]. URL: http://arxiv.org/abs/1910.03771 (visited on 04/13/2023).

Yang, Xiaoyu et al. (Mar. 2023). *Knowledge Distillation from Multiple Foundation Models for End-to-End Speech Recognition*. arXiv:2303.10917 [cs, eess]. URL: http://arxiv.org/abs/2303.10917 (visited on 03/29/2023).

Yoon, Ji Won et al. (2021). "TutorNet: Towards Flexible Knowledge Distillation for End-to-End Speech Recognition". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29. arXiv:2008.00671 [eess], pp. 1626–1638. ISSN: 2329-9290,

2329-9304. DOI: 10.1109/TASLP.2021.3071662. URL: http://arxiv.org/abs/2008.00671 (visited on 10/24/2022).

Zeiler, Matthew D. et al. (June 2010). "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919, pp. 2528–2535. DOI: 10.1109/CVPR.2010.5539957.

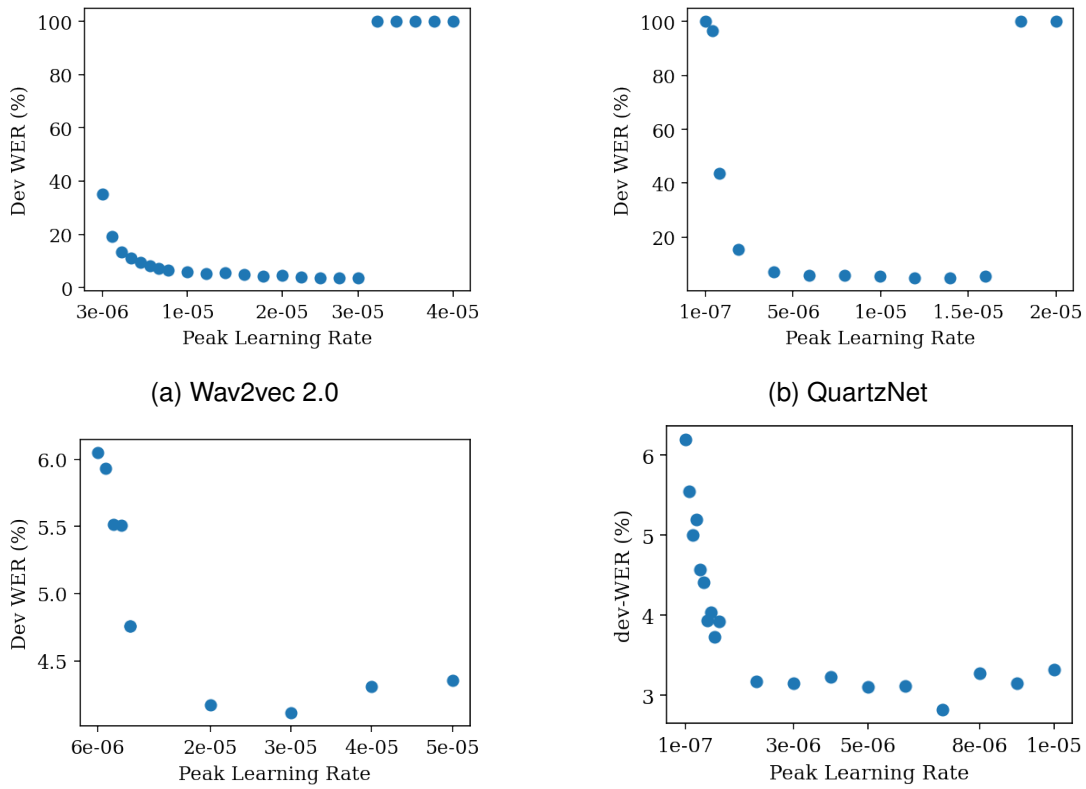
Appendix A

Learning Rate Grid Searches

The plots in Figure A.1 show the results for grid searches on the learning rate of the model architectures used. The training parameters are described in Section 5.1.3. Each model is fine-tuned on the WSJ training set. The best learning rates found are shown in Table A.1.

Table A.1: Best learning rates (LR) found in the grid searches from Figure A.1.

Model	Best LR
Wav2vec 2.0	2.6×10^{-5}
QuartzNet	1.4×10^{-5}
ContextNet	3×10^{-5}
Citrinet	7×10^{-6}



(a) Wav2vec 2.0

(b) QuartzNet

(c) ContextNet

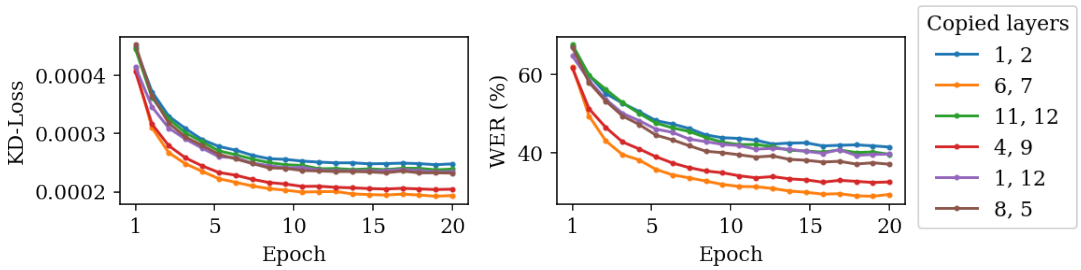
(d) Citrinet

Figure A.1: Grid searches for determining the best starting learning rate for various models. The learning rate is exponentially decaying by 10% per epoch. Fine-tuning was done for 20 epochs.

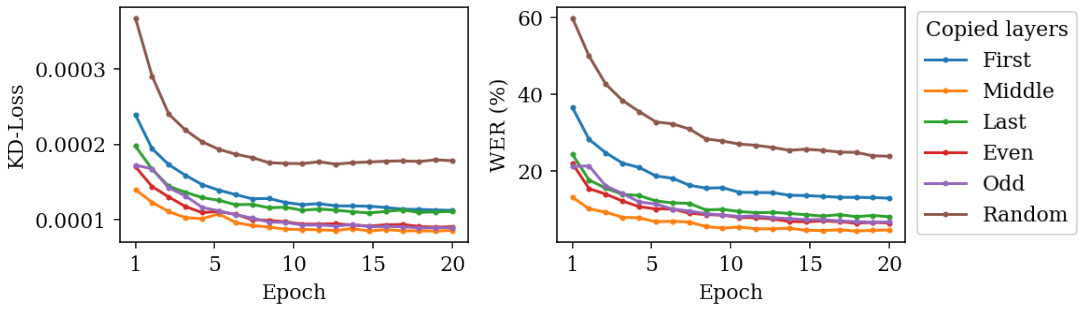
Appendix B

Dev-Loss and Dev-WER for student initialisations

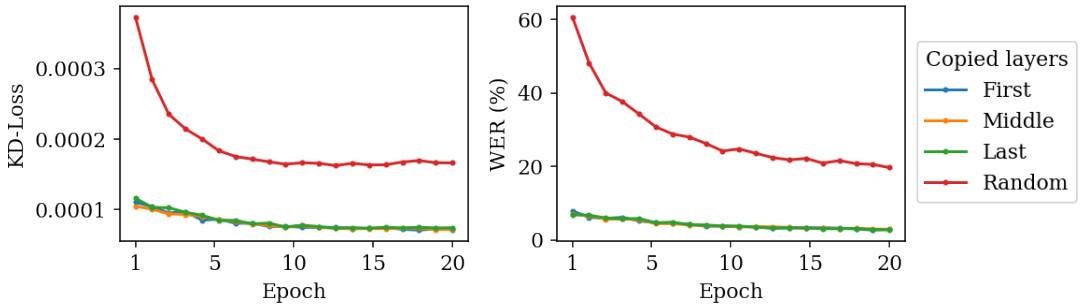
Figure B.1 shows the knowledge distillation loss (see eq. 2.4) and word error rate on the development set across epochs for various student initialisations. The students are initialised according to the experiments outlined in Section 3.3 and trained using knowledge distillation from a fine-tuned Wav2vec 2.0. The training is done in accordance with the parameters outlined in Section 5.1.3.



(a) 2-layer students



(b) 6-layer students



(c) 10-layer students

Figure B.1: Wav2vec 2.0 models with different numbers of transformer layers, trained using knowledge distillation from a fine-tuned Wav2vec 2.0. The students are initialised by copying layers from the teacher according to the experiments outlined in Section 3.3. The plots show the knowledge distillation loss and WER on the development set across epochs for each student.