

Extremely Lightweight Vocoders for On-device Speech Synthesis

Tianren Gao

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-69

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-69.html>

May 13, 2021



Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to sincerely thank Bichen Wu and Daniel Rothchild for their continuous advice on the project. They enabled me to grow both in research and non-technical areas. I would also like to thank Bohan Zhai and Flora Xue for collaborating on this work to navigate through different ideas and project stages. And I appreciate Prof. Kurt Keutzer and Prof. Joseph Gonzalez for their continuous support and invaluable insights into my project. I'll be forever grateful for having the opportunity to work with all of them.

Extremely Lightweight Vocoders for On-device Speech Synthesis
by Tianren Gao

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

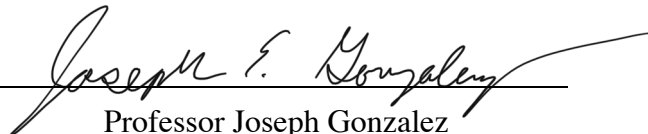


Professor Kurt Keutzer
Research Advisor

May 7, 2021

(Date)

* * * * *



Professor Joseph Gonzalez
Second Reader

May 11th, 2021

(Date)

Extremely Lightweight Vocoders for On-device Speech Synthesis

by

Tianren Gao

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Kurt Keutzer, Chair

Spring 2021

Extremely Lightweight Vocoders for On-device Speech Synthesis

Copyright 2021
by
Tianren Gao

Abstract

Extremely Lightweight Vocoders for On-device Speech Synthesis

by

Tianren Gao

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Kurt Keutzer, Chair

As edge device applications begin to increasingly interact with users through speech, efficient automatic speech synthesis is becoming increasingly important. Typical text-to-speech pipelines include a vocoder, which translates intermediate audio representations into raw audio waveforms. Most existing vocoders are difficult to parallelize since each generated sample is conditioned on previous samples. Flow-based feed-forward models, for example WaveGlow, is an alternative to these auto-regressive models [17]. However, while WaveGlow can be easily parallelized, the model is too expensive for real-time speech synthesis on the edge. This work presents SqueezeWave, an extremely lightweight vocoder that can generate audio of similar quality to WaveGlow with 61x - 214x fewer MACs.

Contents

Contents	i
List of Figures	ii
List of Tables	iii
1 Introduction	1
1.1 From the Cloud to the Edge	1
1.2 Efficient Speech Synthesis	2
2 Related Works	3
3 Preliminaries	4
3.1 Flow-based Generative Models	4
3.2 Architecture of WaveGlow	5
4 Methodology	8
4.1 Redesign Bijections in Flow	8
4.2 Depthwise Separable Convolutions	10
4.3 Other Improvements	11
5 Experiments	12
5.1 Experimental Setup	12
5.2 Audio Quality and Efficiency Analysis	12
6 Conclusion	15
Bibliography	16

List of Figures

3.1	Architecture of WaveGlow.	5
3.2	Architecture of WN in WaveGlow.	7
4.1	Architecture of the WN function in SqueezeWave.	9
4.2	Normal convolutions vs. depthwise separable convolutions. Depthwise separable convolutions can be seen as a decomposed convolution that first combines information from the temporal dimension and then from the channel dimension.	10
5.1	Audio quality (MOS scores) and the efficiency (MACS) trade-off of SqueezeWave family models.	14

List of Tables

- 5.1 A comparison of SqueezeWave and WaveGlow. GT represents the ground truth audio. SW-128L has a configuration of $L=128$, $G=256$, SW-128S has $L=128$, $G=128$, SW-64L has $L=64$, $G=256$, and SW-64S has $L=64$, $G=128$. Mean opinion scores (MOS) measures the generated audio quality. GMACs measures the computing cost of synthesizing 1 second of 22kHz audio. The MAC reduction ratio is reported in the column “Ratio”. We also report number of parameters and actual inference speeds, number of samples generated per second, on a Mackbook Pro and a Raspberry Pi. 13

Acknowledgments

I would like to sincerely thank Bichen Wu and Daniel Rothchild for their continuous advice on the project. They enabled me to grow both in research and non-technical areas. I would also like to thank Bohan Zhai and Flora Xue for collaborating on this work to navigate through different ideas and project stages. And I appreciate Prof. Kurt Keutzer and Prof. Joseph Gonzalez for their continuous support and invaluable insights into my project. I'll be forever grateful for having the opportunity to work with all of them.

Chapter 1

Introduction

1.1 From the Cloud to the Edge

As the popularity of smart phones and demand of artificial intelligence in everyday life increases, deep learning applications on edge devices are playing an increasingly important role. Most of these applications rely on complex, deep neural networks that require significant computing resources for both training and inferences. However, the computational resources of edge devices are relatively limited for models of these size and are often inadequate to directly run network models that require significant computing resources. As a result, a widely used solution is to run models in the cloud in order to send results to the edge devices.

However, this paradigm is being challenged by the following trends. First, data privacy has been of increasing concern in the industry. When customers are required to send their private data to the cloud, they face the risk of exposing sensitive information. For example, intelligent home assistant systems have been accused of eavesdropping incessantly by sending user's private data to the cloud without their knowledge¹. Efficient inference at the edge could protect users' sensitive data, because user data never leave the device. Second, the computing power of edge devices has been continuously improving. Leveraging the computation resources on the edge devices could significantly reduce the need for cloud computing. For example, moving the model inference from cloud to the edge. Third, edge applications are involving an increasing amount of interactions that require low-latency between the user and the device. For exmaples, both home-assistant systems and map-direction applications have involved increasing interactions with speech. These interactions require low latency and even can function without internet connections.

A promising direction to explore is through efficient neural networks. They require significant lower computational resources with lower latency, and also avoid exposing user's data by performing inference at the edge. These efficient neural networks can aid in the areas

¹<https://www.washingtonpost.com/technology/2019/05/06/alexa-has-been-eavesdropping-you-this-whole-time>

of computer vision, speech, and natural language understanding. In the past years, many efficient neural networks have been proposed, such as [6, 18, 7, 21, 20, 4, 11, 8]. However, these networks focus on computer vision and NLP tasks. As one of the most important parts of edge applications, efficient speech synthesis models can be both desirable and necessary.

1.2 Efficient Speech Synthesis

Speech synthesis systems are designed for generating raw audio waveforms. Text to Speech(TTS) is one of the most common tasks in this area and has been increasingly used in many edge applications. For example, map applications read out directions to drivers; real-time translation applications speak text translated from one language to another; automatic food ordering system in the drive-thru window interact with drivers through speech. Modern TTS systems typically consist of two stages: 1) Given an input text, a synthesizer is able to generate acoustic features (e.g., mel-spectrogram, phoneme, F0 frequencies). 2) From the generated acoustic features, a vocoder model generates raw audio waveforms. In this work, we will further explore the second stage: lightweight vocoder models.

High quality and real-time speech synthesis remain challenging on edge devices. Existing vocoders, either suffer from slow inference speed due to their auto-regressive architectures, or require expensive computing power beyond the mobile device processors can afford. We will discuss the detail of these models in the related work section.

In this work, we propose SqueezeWave, an extremely lightweight flow-based vocoder for on-device speech synthesis. SqueezeWave is able to synthesize human-like real-time audio waveforms with 61-214x fewer MACS than the current state-of-the-art. This is achieved by carefully designing a flow-based architecture, adopting depthwise separable convolutions, and making other optimizations for edge devices. Our experiments showed that on an Intel i7 CPU(2019 Macbook Pro), SqueezeWave generates raw audio waveforms at a speed of 123K - 303K samples per second, or 5.6x - 13.8x faster than real-time. Even on a Raspberry Pi 3B+ with a Broadcom BCM2837 CPU, SqueezeWave is able to reach a near real-time speed of 15.6K samples per second.

The code, trained models, and generated speech samples are publicly available at <https://github.com/tianrengao/SqueezeWave>. SqueezeWave has also been integrated into Nvidia Nemo^{2,3}, an open-source toolkit for developing state-of-the-art conversational AI models.

²<https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/tts/intro.html>

³<https://github.com/NVIDIA/NeMo>

Chapter 2

Related Works

A full Text to Speech pipeline includes a synthesizer and a vocoder. We mainly introduce related works for vocoders, where most of this report’s contributions focus on.

WaveNet[14], proposed in 2016, is a deep generative model for raw audio. WaveNet proposed to generate long sequence audio samples using dilated convolutions with increasing receptive fields by orders of magnitude. WaveNet for TTS task is conditioned on linguistic features and logarithmic fundamental frequency values, and achieves human-like synthesized audio quality. However, WaveNet and its variant models WaveRNN[2] are auto-regressive models, which means that the latter generated sample depends on previous generated samples. The inherently serial nature of this computation makes it difficult for performing parallel hardware acceleration. A number of works have tried to solve the inference speed problem by training a simple feed-forward network through distillation. Parallel WaveNet and Clarinet [13, 15] proposed to use Probability Density Distillation, which uses a trained WaveNet as the teacher model to train a student network with compound loss functions. They both use Inverse Auto-regressive Flows to enable parallel inference. As discussed in [17], these models are more complex to implement than auto-regressive models and can be inefficient and difficult to reproduce.

The state-of-the-art vocoder, WaveGlow [17], shows that an auto-regressive flow is unnecessary for speech synthesizer vocoders. Instead, WaveGlow uses a flow-based architecture, which was proposed in NICE and Glow [3, 12] for computer vision tasks. Flow-based architectures have the advantages of the parallelizability of both training and synthesis, and the tractability of the latent space distribution. With a flow-based architecture, WaveGlow makes parallel hardware acceleration possible, and successfully generates faster than real-time speech. However, the computational cost of WaveGlow is high. To generate 1 second of 22kHz speech, WaveGlow requires 229G MACs, which is far beyond the computing power that mobile device processors can afford. Therefore, high quality and real-time speech synthesis remain challenging due to the constrain of computation resources on edge devices.

Chapter 3

Preliminaries

3.1 Flow-based Generative Models

Flow-based model, proposed by [3, 12], is a category of generative models that capture complex data distributions. A flow-based model $p_\theta(x)$ with parameters θ models the unknown true data distribution $\mathbf{x} \sim p(\mathbf{x})$ by learning an invertible transformation $f_\theta : \mathbf{x} \rightarrow \mathbf{z}$, where $\mathbf{z} \sim p(\mathbf{z})$ is a tractable distribution, such as a spherical multivariate Gaussian distribution.

During inference, a latent-variable \mathbf{z} is first drawn from $p(\mathbf{z})$. Then, samples are generated by performing $\mathbf{x} = f_\theta^{-1}(\mathbf{z})$. During training, the optimization procedure maximizes the log likelihood of the data under the tractable distribution in the latent space. Using the change of variables formula, this log likelihood can be computed as:

$$\log(p(\mathbf{x})) = \log(p(\mathbf{z})) + \sum_{i=1}^k \log \left| \det \left(\frac{d\mathbf{z}}{d\mathbf{x}} \right) \right|$$

To make the computation of log determinant tractable and straightforward, the architecture of the model is designed to be a sequence of k invertible transformations $f_\theta = f_1 \circ f_2 \circ \dots \circ f_k$, where each f_i converts between hidden state h_{i-1} and the next hidden state h_i :

$$\mathbf{x} \xleftarrow{f_1} \mathbf{h}_1 \xleftarrow{f_2} \mathbf{h}_2 \xleftarrow{f_3} \dots \xleftarrow{f_{k-1}} \mathbf{h}_{k-1} \xleftarrow{f_k} \mathbf{z}$$

Therefore the log likelihood can be further written as:

$$\log(p(\mathbf{x})) = \log(p(\mathbf{z})) + \sum_{i=1}^k \log \left| \det \left(\frac{\partial f_i(\mathbf{h}_{i-1})}{\mathbf{h}_{i-1}} \right) \right|$$

Where $\partial f_i(\mathbf{h}_{i-1})/\mathbf{h}_{i-1}$ is the Jacobian matrix of the i^{th} flow f_i . The f_i are typically chosen to have triangular Jacobian matrices in order to make the above computation tractable. The input \mathbf{h}_i to each bijection is split in half channel-wise into $\mathbf{h}_{i,a}$ and $\mathbf{h}_{i,b}$, while a non-invertible coupling network is run on $\mathbf{h}_{i,b}$ to obtain the affine transformation coefficients $\log \mathbf{s}$ and \mathbf{t} .

Then :

$$\begin{aligned} \log \mathbf{s}, \mathbf{t} &= \text{coupling}(\mathbf{h}_{i,b}) \\ \mathbf{h}_{i+1,a} &= \mathbf{s} \odot \mathbf{h}_{i,a} + \mathbf{t} \\ \mathbf{h}_{i+1,b} &= \mathbf{h}_{i,b} \end{aligned}$$

Then, \mathbf{h}_{i+1} , the output of this flow is a concatenation of $\mathbf{h}_{i,a}$ and $\mathbf{h}_{i,b}$, followed by an invertible 1x1 convolution to mix channels. The bijection is invertible, though the coupling layer itself is not invertible. $\log \mathbf{s}$ and \mathbf{t} can be computed again by performing coupling layer using $\mathbf{h}_{i,b} = \mathbf{h}_{i+1,b}$, then $\mathbf{h}_{i,a}$ can be reconstructed by undo the affine transformation of $\mathbf{h}_{i+1,a}$.

3.2 Architecture of WaveGlow

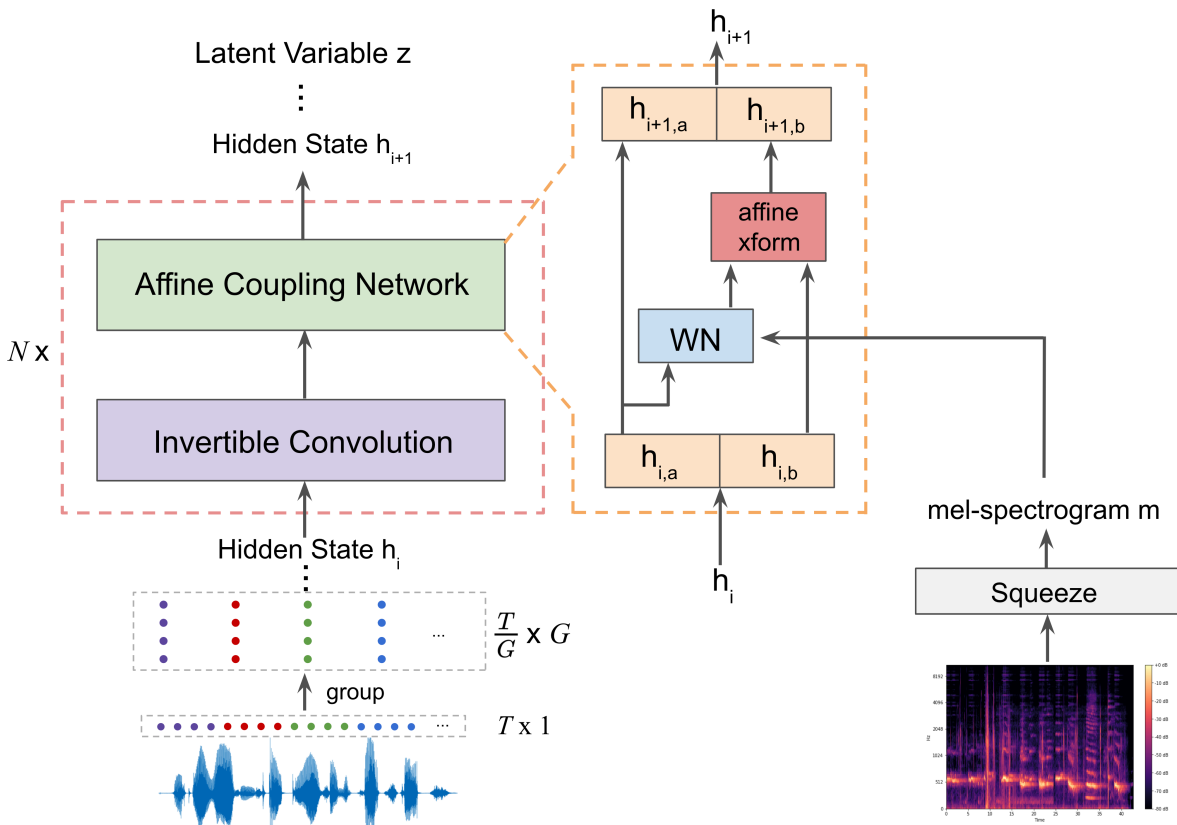


Figure 3.1: Architecture of WaveGlow.

WaveGlow Architecture WaveGlow is a flow-based generative model, so it inherits the architecture described in the above section, as described in Figure 3.1. It consists of a sequence of bijections that progressively transform a waveform of shape $T \times 1$ into a latent space, where T is the temporal length of the waveform, and G is the number of groups. WaveGlow first groups the input waveform’s samples into $G = 8$ groups, each with $\frac{T}{G}$ samples. Then the grouped waveform with shape $\frac{T}{G} \times G$ is taken by a sequence of bijections and is progressively transformed into a latent space. Each bijection f_i takes \mathbf{h}_i as input and produces \mathbf{h}_{i+1} as output. In each bijection, after processing the input feature \mathbf{h}_i by an invertible point-wise 1x1 convolution, the output is split into two halves along the channel dimension, $\mathbf{h}_{i,a}$ and $\mathbf{h}_{i,b}$, each with shape $\frac{T}{G} \times \frac{G}{2}$. Then $\mathbf{h}_{i,a}$ is passed into a WaveNet like layer(WN) to compute affine coupling coefficients conditioned on the mel-spectrogram m of shape $T_m \times C_m$:

$$\begin{aligned} \log \mathbf{s}_i, \mathbf{t}_i &= \text{WN}(\mathbf{h}_{i,a}, m) \\ \mathbf{h}_{i+1,b} &= \mathbf{s}_i \odot \mathbf{h}_{i,b} + \mathbf{t}_i \\ \mathbf{h}_{i+1,a} &= \mathbf{h}_{i,a} \end{aligned}$$

Then, $\mathbf{h}_{i+1,a}$ and $\mathbf{h}_{i+1,b}$ are concatenated along the channel dimension.

WN Architecture The WaveNet-like layer(WN), illustrated in Figure 3.2, possesses an architecture similar to that of WaveNet and parallel WaveNet. The WN layer takes in the previous hidden state $\mathbf{h}_{i,a}$ containing waveform information, and performs affine coupling transformations conditioned on the mel-spectrogram m . First, $\mathbf{h}_{i,a}$ is processed by a point-wise convolution named *start* to increase its channel size from $\frac{G}{2} = 4$ to $C = 256$. Then, the output is processed by a dilated 1D convolution with kernel size of 3 named *in_layer*. Meanwhile, the mel-spectrogram \mathbf{m} is being upsampled from length $T_m = 63$ to match with the waveform feature’s temporal length $T = 2,000$. Then its output is processed by another dilated 1D convolution named *cond_layer* with kernel size of 3 to increase its channel size from $C_m = 4$ to $C = 256$. So far, both the waveform feature and the mel-spectrogram feature have the same shape of $T \times C = 2000 \times 256$. Next, these two features are combined through the *gate* operation, which is similar to WaveNet [14]. Then the output is then processed by a convolution named *res_skip_layer* to double its channel size. The output of this layer has a shape of $T \times 2C = 2000 \times 512$. At last, the output is split along the channel size into two halves, each of shape $T \times C = 2000 \times 256$. The two outputs are correspondingly added back to the waveform feature through a residual connection, and to an accumulative output feature through a skip connection. This structure is repeated 8 times, and the final accumulative output feature is passed into a convolution layer labeled *end*. This convolution computes the transformation factors \mathbf{s}_i and \mathbf{t}_i and decreases the channel size from $C = 512$ back to $G = 8$.

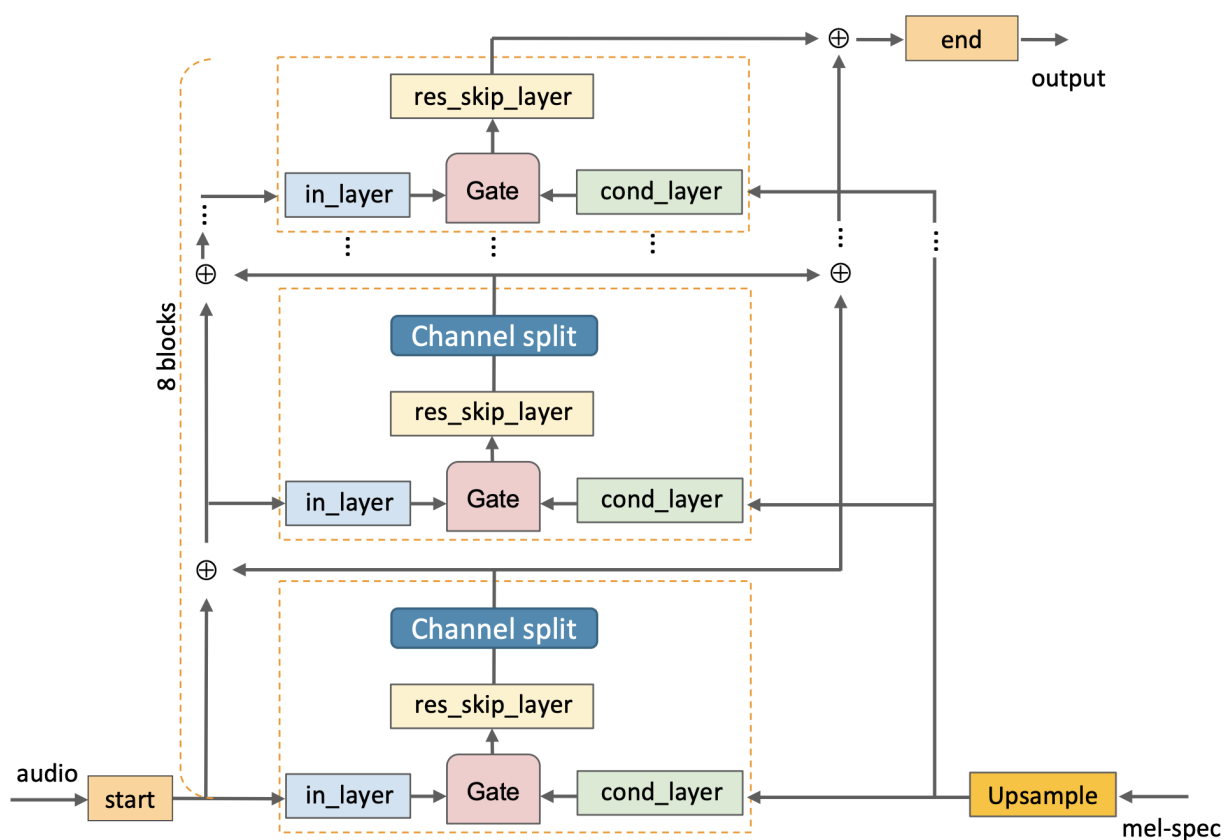


Figure 3.2: Architecture of WN in WaveGlow.

Computational Complexity We calculated the computational cost of WaveGlow based on its PyTorch implementation. WaveGlow requires 229G MACs to generate 1 second of 22kHz waveform. WN block is responsible for the majority of the computation. The details of the calculation can be found in our source code.

Chapter 4

Methodology

4.1 Redesign Bijections in Flow

Inspired by WaveGlow, we used a flow-based architecture that consists of a series of bijection blocks. Each bijection contains an invertible 1x1 convolution and an affine coupling layer, which is a WaveNet like architecture (WN) conditioned on mel-spectrograms. We designed the bijections with the following techniques to optimize the model’s expressive power and reduce the computing cost. The improved WN architecture is illustrated in Figure 4.1.

Information Bottlenecks Within each flow, the coupling networks have the most non-linearity expressive power. However, because the bijections do not change the shape of \mathbf{h}_{i+1} from \mathbf{h}_i so that the flow can remain invertible, there exist information bottlenecks between flows, preventing the coupling networks in consecutive flows from freely communicating [1]. For example in WaveGlow, as in figure 3.2, the input hidden state \mathbf{h}_{i+1} and the output hidden state \mathbf{h}_i both have a channel size of $G = 8$, while the intermediate channel size in bijections has a size of 256 or 512. Although the large intermediate channel size increases model capacity, the significant change in channel size creates information bottlenecks, alleviating the expressive power of WN blocks. To solve this problem, we designed a much larger G .

Reduction in Temporal Dimension WN is a wavenet like block that consists of several 1D convolutional layers. The computational complexity of 1D convolutional layers is linear in the temporal length L . In WaveGlow, the significant temporal length of the grouped input waveform is $L = \frac{T}{G} = 2000$, which is a large number and leads to high computational complexity. We designed a much lesser $L = 256$ or $L = 512$ in temporal dimension. Combining with the larger G in the above paragraph, we implemented two settings: $L = 64, G = 256$ or $L = 128, G = 128$, among the same number of total samples for the input waveform feature ($64 \times 256 = 128 \times 128 = 16,384$ samples).

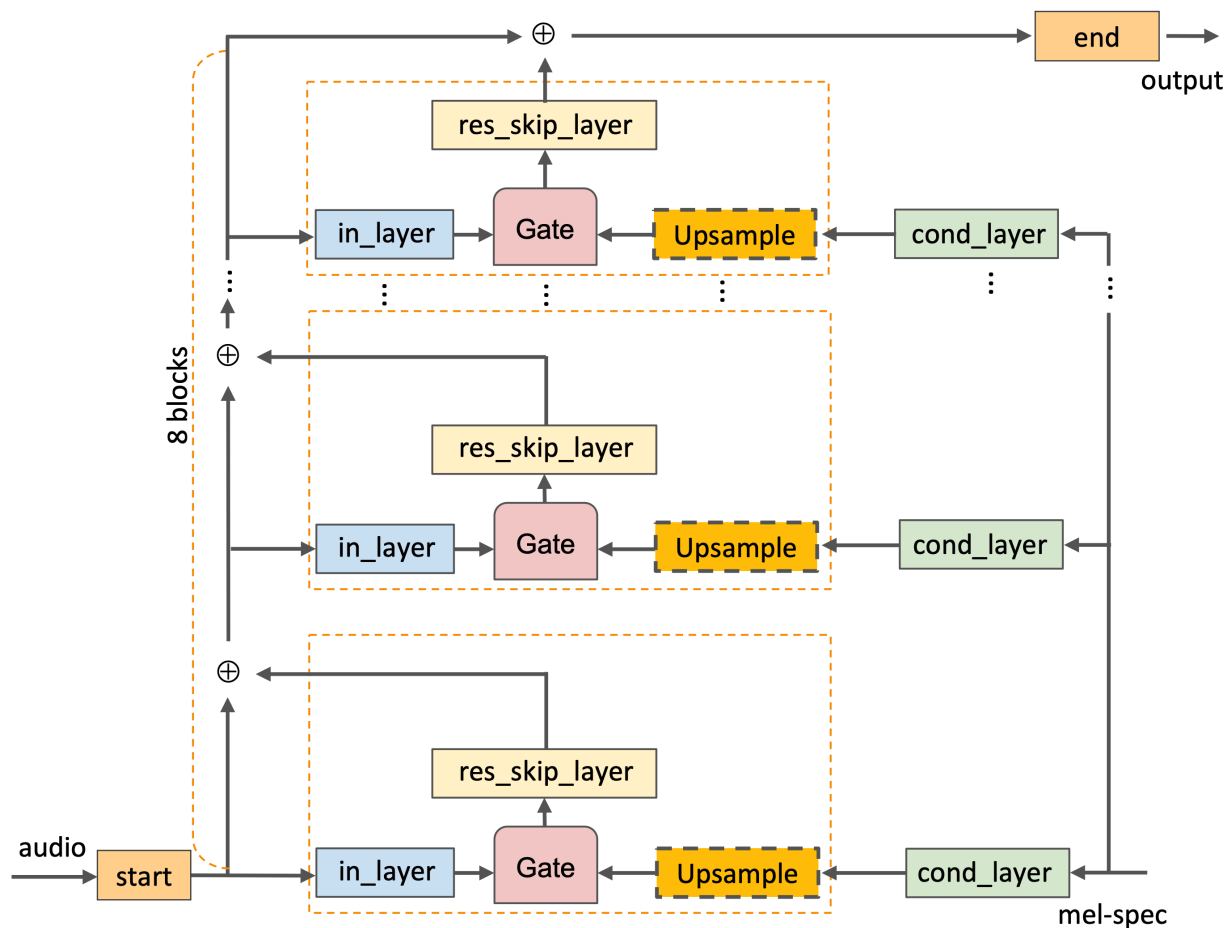


Figure 4.1: Architecture of the WN function in SqueezeWave.

Feature Projection Method We designed a simple but effective method to project the mel-spectrogram feature into the latent space, while avoiding additional computations that was in WaveGlow. Similar to WaveNet, we fused the waveform feature and the mel-spectrogram feature by a gate operation, in order to condition on mel-spectrogram. The gate operation requires the two input features to have the same shape. Compare to $L = 2,000$, the mel-spectrograms have a much coarser temporal resolution $L_m = 64$. Instead of upsampling L_m to match $L = 2,000$, which is used in WaveGlow, we simply reshaped the input grouped waveform $\mathbf{h}_{i,a}$ to have significantly lesser temporal length of L . As described in the above paragraph, this also helps reducing the computational cost in 1D convolutions. In this way, we can still condition waveform on mel-spectrograms by the gate operation while avoiding the additional computations in *cond_layer* caused by upsampling.

4.2 Depthwise Separable Convolutions

We replaced 1D convolutions in the *in_layer* with depthwise separable convolutions to process 1D audio features. Depthwise separable convolutions are popularized by [6] and are widely used in efficient computer vision models, including [18, 20].

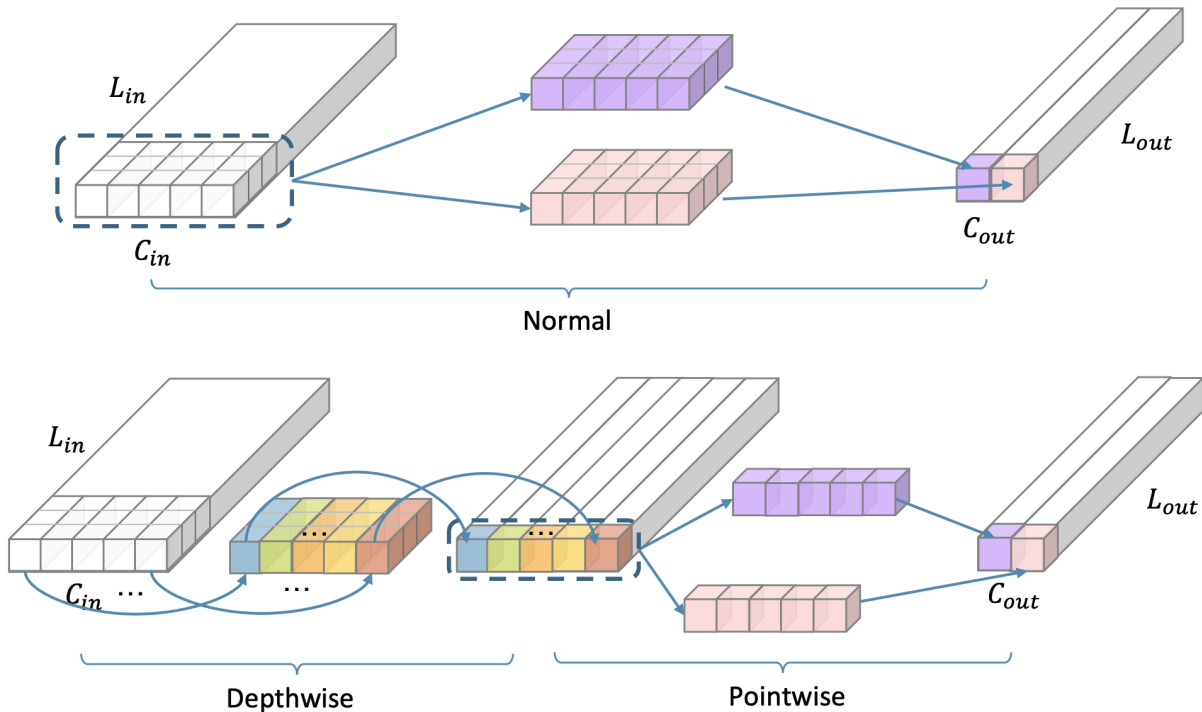


Figure 4.2: Normal convolutions vs. depthwise separable convolutions. Depthwise separable convolutions can be seen as a decomposed convolution that first combines information from the temporal dimension and then from the channel dimension.

To illustrate the benefits of depthwise separable convolutions, consider a 1D convolutional layer that transforms an input with shape $C_{in} \times L_{in}$ into an output with shape $C_{out} \times L_{out}$, where C and L are the number of channels and temporal length of the signal, respectively. For a kernel size K , the kernel has shape $K \times C_{in} \times C_{out}$, so the convolution costs $K \times C_{in} \times C_{out} \times L_{out}$ MACs. A normal 1D convolution combines information in the temporal and channel dimensions in one convolution with the kernel. The depthwise separable convolution decomposes this functionality into two separate steps: (1) a temporal combining layer and (2) a channel-wise combining layer with a kernel of size 1. Step 1 is called a depthwise convolution, and step 2 is called a pointwise convolution. The difference between a normal 1D convolution and a 1D depthwise separable convolution is illustrated in Figure 4.2.

After applying the depthwise separable convolution, the computational cost for step-1 becomes $K \times C_{in} \times L_{in}$ MACs and for step-2, $C_{in} \times C_{out} \times L_{in}$. The reduction of computation is therefore

$$\frac{C_{in} \times C_{out} \times L_{in} + K \times C_{in} \times L_{in}}{K \times C_{in} \times C_{out} \times L_{in}} = \frac{1}{C_{out}} + \frac{1}{K}.$$

In our setup, $K = 3$ and $C_{out} = 512$, so using this technique leads to around 3x MAC reduction in the *in_layers*.

4.3 Other Improvements

In addition to the above techniques, we also made several other improvements. First, since the temporal length is now much lesser, there is a much reduced need for WN block to increase the receptive fields using dilated convolutions. So we replaced all dilated convolutions with regular convolutions, which are more hardware friendly. 2) Figure 3.2 shows that the outputs of the *res_skip_layers* are split into two halves followed by two separate residual/skip connections. We merged them into one branch with the hypothesis that such a split is not necessary since the topologies of the two branches are almost identical. Removing one branch leads to the reduction in the output channel size of the *res_skip_layers*, which reduced its computation complexity by half.

Chapter 5

Experiments

In this report, we use WaveGlow as our baseline, and we compare SqueezeWave to Waveglow in terms of audio quality and the efficiency.

5.1 Experimental Setup

We use a similar experimental setup to that of [17]. We use a single female english speaker dataset LJSpeech [9]. The dataset consists of 13,100 short audio clips paired with corresponding transcription, approximately 24 hours in total of length. The speech audio are recorded on a MacBook Pro using its built-in microphone in a home environment, with a sample rate of 22,050kHz. We process the audio data using librosa library to extract mel-spectrograms. The parameters of this step are FFT size 1024, hop size 256, and window size 1024. The baseline model is reproduced by training from scratch using WaveGlow PyTorch implementation on 8 Nvidia V100 GPUs with a batch size 24. And the SqueezeWave model is trained on Nvidia Titan RTX GPUs with a batch size 96 for 600k iterations. We have 4 variants of SqueezeWave with different configuration of L and G . SW-128L has a configuration of $L=128$, $G=256$, SW-128S has $L=128$, $G=128$, SW-64L has $L=64$, $G=256$, and SW-64S has $L=64$, $G=128$. The dataset train-test split, detailed model configuration and the training recipe are available on our public GitHub repository.

5.2 Audio Quality and Efficiency Analysis

Audio Quality The metric we used to evaluate our model is Mean Opinion Score(MOS) with scale from 1 to 5, as in [19, 14, 17, 16]. We crowd-sourced our MOS evaluation on Amazon Mechanical Turk. We randomly selected 10 fixed sentences for each model, and asked the listeners to rate the naturalness of the sentences. Each listener was only allowed to rate the same sentence once(i.e. no more than 10 different sentences), in order to ensure the diversity of listeners on the same system. But they were allowed to rate multiple sentences. For each sentence, we collected 100 valid ratings. Invalid ratings were detected by a set of

Models	MOS	GMACs	Ratio	Params	Macbook Pro	Raspberry Pi
GT	4.62 ± 0.04	–	–	–	–	–
WaveGlow	4.57 ± 0.04	228.9	1	87.7 M	4.2K	Failed
SW-128L	4.07 ± 0.06	3.78	61	23.6 M	123K	5.2K
SW-128S	3.79 ± 0.05	1.07	214	7.1 M	303K	15.6K
SW-64L	3.77 ± 0.05	2.16	106	24.6 M	255K	9.0K
SW-64S	2.74 ± 0.04	0.69	332	8.8 M	533K	21K

Table 5.1: A comparison of SqueezeWave and WaveGlow. GT represents the ground truth audio. SW-128L has a configuration of $L=128$, $G=256$, SW-128S has $L=128$, $G=128$, SW-64L has $L=64$, $G=256$, and SW-64S has $L=64$, $G=128$. Mean opinion scores (MOS) measures the generated audio quality. GMACs measures the computing cost of synthesizing 1 second of 22kHz audio. The MAC reduction ratio is reported in the column “Ratio”. We also report number of parameters and actual inference speeds, number of samples generated per second, on a Mackbook Pro and a Raspberry Pi.

hidden quality assurance tests, for example, rating the ground truth a lower score than an obviously unnatural noise waveform. We reported MOS scores with 95% confidence intervals.

Tabel 5.1 compares audio quality of SqueezeWave and WaveGlow. WaveGlow achieved MOS scores comparable to those for ground-truth audio. Quantitatively, MOS scores of the SqueezeWave models were lower than WaveGlow, but qualitatively, their sound qualities were similar, except that audio generated by SqueezeWave contained some background noise. Noise cancelling techniques could have been applied to improve the quality. Readers can find synthesized audio of all the models from our source code.

Efficiency We compare WaveGlow and SqueezeWave in terms of three metrics: 1) MACs of generating one second of 22kHz audio, 2) number of model parameters, and 3) actual speech generation speed, in generated samples per second, on a Macbook Pro and a Raspberry Pi 3b+. As shown in the Tabel 5.1, WaveGlow was extremely expensive, for it required 228.9 GMACs. Compared to WaveGlow, SqueezeWave models were significantly efficient. The largest model, SW-128L, with a configuration of $L=128$, $G=256$ required 61x fewer MACs than WaveGlow. With reduced temporal length or channel size, SqueezeWave requires significantly lower MACs: SW-64L required 2.16 GMACs, 106x fewer than WaveGlow, and SW-128S required 1.07 GMACs, 214x fewer than WaveGlow. We also trained an extremely lightweight model, SW-64S, with $L=64$, $G=128$. The model only required 0.69 GMACs, which is 332x fewer than WaveGlow.

To measure the actual speech generation speed on edge devices, we deploy WaveGlow and SqueezeWave to a Macbook Pro with an Intel i7 CPU and a Raspberry Pi 3B+ with a Broadcom BCM2837B0 CPU. In Tabel 5.1, We report the number of samples generated per second by each model. On a Mackbook, SqueezeWave can reach a sample rate of 123K-303K,

30-72x faster than WaveGlow, or 5.6-13.8x faster than real-time (22kHz). On a Raspberry Pi computer, WaveGlow failed to run, but SqueezeWave could still reach 5.2k-21K samples per second. SW-128S in particular could reach near real-time speed while maintaining good quality.

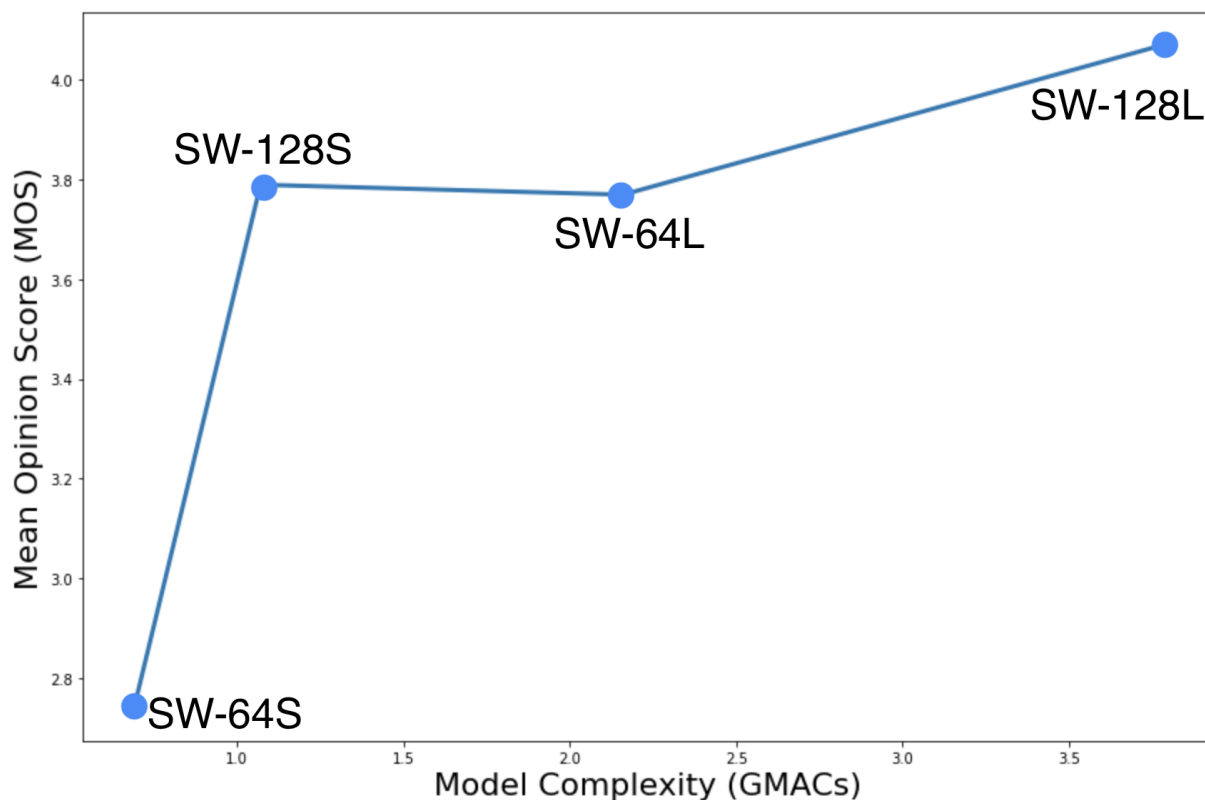


Figure 5.1: Audio quality (MOS scores) and the efficiency (MACS) trade-off of SqueezeWave family models.

Quality and Efficiency Trade-off In Figure 5.1, we show the trade-off between audio quality and the efficiency of SqueezeWave family models. As we can see, as the model complexity decreases, the MOS scores decreases very slowly except for SW-64S. Among the variants, SW-128L has the highest MACs but generates highest quality audios. SW-128S requires 1.07 GMACs but still generate relative high quality audios. Our smallest model, SW-64S only required 0.69 GMACs is able to generate acceptable audios.

Chapter 6

Conclusion

In this work, we discussed the motivation and benefit of moving computations from the cloud to the edge. We focus on efficient vocoders that perform inference at the edge. We presented SqueezeWave, a flow-based lightweight vocoder that can generate audio of similar quality to WaveGlow with significantly fewer MACs. Our experiments show that SqueezeWave is able to synthesize speech on edge devices. On a Raspberry Pi 3B+ with a Broadcom BCM2837 CPU, our smallest SqueezeWave variant is able to reach a near real-time speech generation. On an Intel i7 CPU(2019 Macbook Pro), SqueezeWave generates raw audio waveforms at a speed 5.6x - 13.8x faster than real-time.

In the future, it could be desirable to also move neural network training to the edge, in order to further leverage the mobile computing resource. One of the main bottlenecks for this goal is the memory constrain on edge devices. RevNet[5] and iRevNet[10] proposed that reversible neural networks require constant memory consumption during training because there is no need to cache activations while performing backpropagation. This makes it possible to train deep neural networks with constant memory cost, which is an interesting direction to explore for on-device training in the future.

Bibliography

- [1] Jianfei Chen et al. *VFlow: More Expressive Generative Flows with Variational Data Augmentation*. 2020. arXiv: 2002.09741 [stat.ML].
- [2] Xiaoliang Dai et al. “ChamNet: Towards Efficient Network Design Through Platform-Aware Model Adaptation”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.
- [3] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*. 2015. arXiv: 1410.8516 [cs.LG].
- [4] Amir Gholami et al. “SqueezeNext: Hardware-Aware Neural Network Design”. In: *arXiv preprint arXiv:1803.10615* (2018).
- [5] Aidan N. Gomez et al. “The Reversible Residual Network: Backpropagation Without Storing Activations”. In: *CoRR* abs/1707.04585 (2017). arXiv: 1707.04585. URL: <http://arxiv.org/abs/1707.04585>.
- [6] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).
- [7] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).
- [8] Forrest N. Iandola et al. “SqueezeBERT: What can computer vision teach NLP about efficient neural networks?”. In: *CoRR* abs/2006.11316 (2020). arXiv: 2006.11316. URL: <https://arxiv.org/abs/2006.11316>.
- [9] Keith Ito. *The LJ Speech Dataset*. <https://keithito.com/LJ-Speech-Dataset/>. 2017.
- [10] Jörn-Henrik Jacobsen, Arnold W. M. Smeulders, and Edouard Oyallon. “i-RevNet: Deep Invertible Networks”. In: *CoRR* abs/1802.07088 (2018). arXiv: 1802.07088. URL: <http://arxiv.org/abs/1802.07088>.
- [11] Xiaoqi Jiao et al. “TinyBERT: Distilling BERT for Natural Language Understanding”. In: *CoRR* abs/1909.10351 (2019). arXiv: 1909.10351. URL: <http://arxiv.org/abs/1909.10351>.
- [12] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: 1807.03039 [stat.ML].

- [13] Aäron van den Oord et al. “Parallel WaveNet: Fast High-Fidelity Speech Synthesis”. In: *CoRR* abs/1711.10433 (2017). arXiv: 1711.10433. URL: <http://arxiv.org/abs/1711.10433>.
- [14] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).
- [15] Wei Ping, Kainan Peng, and Jitong Chen. “ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech”. In: *CoRR* abs/1807.07281 (2018). arXiv: 1807.07281. URL: <http://arxiv.org/abs/1807.07281>.
- [16] Wei Ping et al. “Deep Voice 3: 2000-Speaker Neural Text-to-Speech”. In: *International Conference on Learning Representations*. 2018. URL: <https://openreview.net/forum?id=HJtEm4p6Z>.
- [17] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. “Waveglow: A flow-based generative network for speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3617–3621.
- [18] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.
- [19] J. Shen et al. “Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2018, pp. 4779–4783. DOI: 10.1109/ICASSP.2018.8461368.
- [20] Bichen Wu et al. “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.
- [21] Bichen Wu et al. “Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving”. In: *arXiv preprint arXiv:1612.01051* (2016).