Ternary Diffusion Models

Fredrik Ekholm
fwe21@cam.ac.uk

Abstract

Diffusion Models have shown recent success in image generation and computer vision, but high inference cost has been a barrier to adoption. This work investigate one approach for reducing inference cost: Ternary Weight Model Quantisation. By mapping weights to {-1,0,1} (up to a scaling factor), both memory and computation cost can be drastically decreased. This study investigates variants of this technique applied to a diffusion model. The code is available at https://github.com/istlemin/ternary-diffusion.

1 Introduction

Diffusion models have recently come to dominate the field of image generation and computer vision [1, 2, 3, 4]. Other than unconditional and prompt-based image generation, success of diffusion models has also been shown in tasks such as image segmentation [5, 6], in-painting [7], medical inverse problems [8], optical flow [9] and more. However, the sampling process is computationally expensive due to the iterative sampling process and the large neural network typically used. Decreasing sampling latency without significant loss in performance would enable use of diffusion model in real-time scenarios and on less capable devices.

There are two main ways of decreasing the inference latency of diffusion models: reducing number of denoising iterations needed for sampling, and reducing the latency of the noise-prediction neural network. The first has received significant attention [10, 11, 12], where the previously thousands of iterations needed are reduced to less than 50. The second approach - applying model compression to the noise-predictor - have seen *some* research, with papers on pruning, quantisation and knowledge distillation [13, 14, 15, 16], but many promising compression techniques remain unexplored. Some techniques applied to other types of models may transfer directly, but compression of diffusion models may also pose additional challenges such as the shifting activation distribution over different diffusion time-steps [13].

EfficientDM [14] successfully quantise diffusion models to 8 and 4 bit weights. Previous works in image classification [17] and transformers [18] has shown that even lower bit quantisation can be possible. TernaryBERT uses knowledge distillation to quantise each weight to -1,0, or 1, representing it with only two bits, with no loss in performance on classification tasks. This work adapts this method to diffusion models, and investigate it's viability.

2 Related Work

Diffusion Model Acceleration Denoising Diffusion Probabilistic Models (DDPMs) [1] model image generation as reversing a Markovian diffusion process, iteratively predicting and removing noise to arrive at an image. They typically require thousands of iteration for high quality generation, which has inhibited their adaptation. Therefore, many methods have aimed to decrease the number of sampling iterations required. Denoising Diffusion Implicit Models (DDIMs) replace the diffusion process by a non-Markovian process that's faster to sample from. Other works use grid searches to find a faster sampling trajectory [19], model trajectory search using dynamic programming [20], or formulate the Diffusion Model in terms of an ODE, and use a fast ODE solver for sampling [21].

Consistency models [22] are trained to replicate the behaviour of a diffusion model using a single iteration. These methods all significantly improve sampling speed of diffusion models. To further reduce sampling speed, one have to reduce the latency of the denoising neural network used. This has received less attention, but some successful works exists. Diff-pruning [15] achieves a 50% reduction in FLOPs by applying model pruning. PTQ-Diff [13] evaluate Post Training Quantisation (PTQ) on diffusion models and find that DMs can be quantised to 8-bits without loss in performance. EfficientDM [14] uses Quantisation Aware Training (QAT) with Low-rank Adaptors, to quantise diffusion models to 4-bit weights.

Low bit quantisation Outside diffusion models, 2 bit or even 1 bit quantisation has shown some promising results. [23] show that expensive multiplications can be significantly reduced during training by probabilistically rounding weights to $\{-1, 1\}$ (binarisation) or $\{-1, 0, 1\}$ (ternarisation). Ternary Weight Networks (TWN) [17] extend this idea to model quantisation for cheaper inference by utilising QAT. For image classification tasks [17], ternarisation achieves up to a 16x reduction in model size with barely any reduction of accuracy. Binarisation achieves a 32x reduction while performing slightly worse. TernaryBERT [18] utilises Knowledge Distillation to effectively train a ternary transformer model on natural language classification tasks. By applying hidden layer knowledge distillation from a full-precision teacher model to a ternarised student model, the student performs on par with the teacher.

3 Theoretical Background

3.1 Model architecture

U-Net U-Net [24] has become the standard architecture for many image-to-image tasks, including the noise prediction in a diffusion model. In the following description, non-linearities and normalisation layers have been left out. The U-Net architecture takes in a an image I_{in} and outputs an image I_{out}. Its first part consists of a series of k convolutional down-sampling units $c_{d1}, c_{d2}, ..., c_{dk}$ as well as an input layer c_{in}, producing a series of features maps with decreasing resolution and increasing number of channels:

$$S_1 = c_{d1}(c_{in}(d_1))$$
$$S_i = c_{di}(S_{i-1}) \text{ for } 2 \leq i \leq k$$

The *skip-connections* S_i are then used as input for a series of convolutional up-sampling units $c_{uk}, ..., c_{u1}, c_{out}$:

$$U_5 = c_{u5}(S_k)$$
$$U_i = c_{ui}(S_i || U_{i-1}) \text{ for } 1 \leq i \leq k-1$$
$$I_{out} = C_{out}(U_1)$$

where || denotes concatenation along the feature dimension.

ResUNet Diakogiannis et. al. [25] improve on the base U-Net architecture, by replacing the convolutional layers in c_{di} and c_{ui} by a residual block. A residual block applies two convolutional layers to the input, with activation and normalisation in between, and adds back the input to the output at the end. This ensure that layers don't erase information, and improves training stability and performance of the model. If the output dimensionality of the residual block is different than the input dimensionality, the input is passed through a single convolutional layer with kernel size one to transform to the right number of features, before added to the output.

3.2 Quantisation

Quantisation is a model compression technique that aims to reduce computation and memory requirements of a model by representing parameters and activations using fewer bits than the standard 32-bit floating points. Commonly, quantised values are stored as low-bit integers, together with a few higher precision parameters such as a scaling factor. I utilise two techniques: Min-Max-quantisation and Ternarisation.

Min-Max-Quantisation One of the simplest quantisation techniques is to quantise a set of weights evenly from the lowest occurring value to the highest. Specifically, let $\mathbf{x}$ be a vector of real values and assume quantisation to n-bit integers. A value x_i is then quantised to the value $b_i = round(\frac{x_i - \min(x)}{\max(x) - \min(x)} \cdot 2^n)$. With $\max(x)$ and $\min(x)$ stored as parameters, the quantised weight can be recovered by $\hat{x_i} = b_i \cdot \frac{\max(x) - \min(x)}{2^n} + \min(x)$

Ternarisation Weight ternarisation is a technique where weights can take only three values: (-1,0,1), represented by 2 bits. This lets us replace many of the expensive floating-point multiplications with simpler addition operations during inference [23].

Ternarisation depends on a single parameter α, the scaling factor. The effective quantised weights will be $\hat{\mathbf{w}} = \alpha \mathbf{b}$, where $\mathbf{b} \in \{-1, 0, 1\}^n$ with n being the number of parameters in w. We aim to find the value of α and quantised weights b_i that minimises the distances between ternarised and full precision weights $\mathbf{w}$. Thus we need to solve the following optimisation problem:

$$\begin{aligned} \min_{\alpha, \mathbf{b}} \quad & ||\mathbf{w} - \alpha \mathbf{b}||_2^2 \\ \text{s.t.} \quad & \alpha > 0, \mathbf{b} \in \{-1, 0, 1\}^n \end{aligned} \tag{1}$$

Let $\mathbf{I}_\Delta(\mathbf{w})$ be a thresholding function defined by

$$\mathbf{I}_\Delta(\mathbf{w})_i = \begin{cases} 1 & \text{if } \mathbf{w}_i > \Delta \\ 0 & \text{if } \Delta > \mathbf{w}_i > -\Delta \\ -1 & \text{if } -\Delta > \mathbf{w}_i \end{cases} \tag{2}$$

It can be shown[17] that there is a Δ such that optimal solution to equation 1 is:

$$\begin{aligned} \mathbf{b} &= \mathbf{I}_\Delta(\mathbf{w}) \\ \alpha &= \frac{||\mathbf{b} \odot \mathbf{w}||_1}{||\mathbf{b}||_1} \end{aligned} \tag{3}$$

Finding the optimal value of Δ requires an expensive sorting operation. However, (Li et al., 2016) [17] instead approximates the solution by

$$\Delta = \frac{0.7||\mathbf{w}||_1}{n} \tag{4}$$

which is close to the optimal if weights are normally distributed.

Quantisation Aware Training Post-Training Quantisation (PTQ) refers to quantising weights after training, and not performing any additional finetuning. For low-bit quantisation (less than 8 bits), this often reduces the quality of the model enough to not be useful. Quantisation Aware Training (QAT) fine-tunes the quantised model to improve its performance. If weights would be stored only in their quantised form, training would stall due to weight updates being smaller than the quantisation step [26, 17]. To alleviate this we keep full-precision weights of the model during training. Let w be the full-precision weights. In every training iteration, we calculate quantised weights $\hat{w} = Q_w(w)$ where Q_w is the combined quantisation function for all weights (which might apply different functions on different parts of the weights). The quantised weights are used to run a forward pass of the model, and calculate a loss L. A gradient of the quantised weights w.r.t loss $\frac{\partial L}{\partial \hat{w}}$ is calculated. This gradient is used to apply an update step to the full-precision weights.

3.3 Hidden Layer Knowledge Distillation

Knowledge Distillation (KD) is the process of using a more capable teacher model to supervise a less capable student model. KD was first introduced for classification problems, where the predicted logits of the teacher model gives a stronger supervision signal to the student than using the hard labels from the dataset [27]. This was later extended to use internal state from the teacher model as supervision signal, where a loss is calculated based on the difference in teacher and student activations. There has been a strong research focus on how to map the higher dimensional teacher activations to lower dimensional student activations for comparison, as well as how to map between layers when the student has more layers than the teacher [28, 29, 30]. In this project, the student has the same shape as the teacher but with its weights quantised. This means that there is a trivial mapping between teacher and student activations that could be used to calculate a knowledge distillation loss. However, giving the student too much supervision might be too constraining, and not let it adapt to more useful internal representations for a quantised model. It might therefore be useful to only compare activations from some layers, or transform activations before comparison or compare proxy values calculated from weights. For CNNs, [28] calculates attention-maps to compare between student and teacher. Specifically, let H^S and H^T be the outputs from the same layer in the student and teacher model, encoded as a sequence of feature-vectors. An *attention KD-loss* is then calculated as:

$$\begin{aligned} A_i^S &= \frac{||H_i^S||_2}{\sum_i ||H_i^S||_2} \\ A_i^T &= \frac{||H_i^T||_2}{\sum_i ||H_i^T||_2} \\ L_{att} &= ||A^S - A^T||_1 \end{aligned}$$

TinyBERT proposes learning projection matrices from the student to the teacher, giving a *projection KD-loss*:

$$L_{pro} = ||p(H^S) - H^T||_1$$

where p is a single linear layer, optimised together with the student model. We also define *direct KD-loss* as:

$$L_{dir} = ||H^S - H^T||_1$$

4 Implementation

The implementation is based on an open-source minimal DDIM [31] implementation[1], written in PyTorch [32]. I alter this implementation by adding support for quantisation aware training, ternarisation and hidden layer knowledge distillation.

4.1 Architecture

I use the ResUNet architecture described above, with $k = 4$ downsampling and upscaling units, with hidden dimensions of 16, 32, 64 and 128 respectively. Each down-sampling unit except the first and last halves the resolution, for a final size of 16x16 with 128 feature channels. In addition to a default ResUNet architecture, a time embedding vector representing the diffusion time-step is added to the output of each residual block. The time embedding vector is calculated using a sinusoidal embedding, transformed by a single linear layer to match the dimensionality. This follows the standard architecture for diffusion model, as introduced in both DDPM and DDIM [1, 31]. For full details of the network, see the provided implementation.

4.2 Quantisation

I implement Min-Max-quantisation and Ternarisation. By default, ternarisaiton is applied to weights in convolutional and linear layers. For convolutional layers, ternarisation is done once per output channel, meaning that in the quantised model, a different α-value is needed for each output channel. This adds negligible overhead in model size, but improves performance. Activations are quantised using Min-Max-quantisation, performed once per channel. Biases and normalisation layer parameters are not quantised.

As special hardware support would be needed to perform the quantised operations, the quantisation is simply simulated during training. This is done by rounding the weights and activations to the value their quantised versions would represent, while still keeping them as full-precision floats.

[1]`https://github.com/filipbasara0/simple-diffusion`

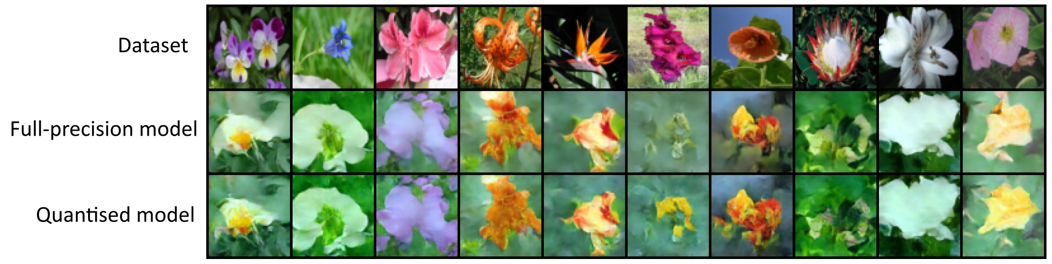

Figure 1: Top row: Sample images from the Oxford Flowers Dataset. Middle row: Images generated by the full-precision model. Bottom row: Images generated by the ternarised model, using the same seed.

4.3 Knowledge Distillation

The UNet model is modified to return a configurable subset of intermediate activations,which can be used to calculate the knowledge distillation loss. I compare the following option:

- **All-layers**: Using the output from all convolutional layers, including the intermediate convolutions in the residual blocks.
- **Skips**: Using only the skip-connections $S_1, ..., S_4$.
- **Upscale**: Using only the outputs from the upscaling units $U_1, ..., U_4$

The final loss is then calculated as

$$L = ||I^S_{out} - I^T_{out}||_1 + \frac{\sum_{i=1}^{k} L_{kd}(H^S_i, H^T_i)}{k}$$

where I^S_{out} and I^T_{out} are the outputs of the student and teacher models respectively, and $H_1, ..., H_k$ are the intermediate activations.

5 Evaluation

Due to computational restrictions, we evaluate the method on a single diffusion model. The model was trained for 100 epochs on the Oxford Flowers dataset [33], using a learning rate of 10^{-3} with a single cycle cosine schedule [34]. It achieves an FID score [35] of 108.1, and figure 1 show some of the generated flowers.

Computing an FID of a model is expensive. Therefore, in order to evaluate the models I primarily use the Mean Absolute Error (MAE) between the output of the full-precision model and the quantised model, for the final 100 steps of training. This is not an ideal measure, for example as the quantised model may generate high quality images that aren't similar to the ones generated by the full-precision model, but it serves as a proxy during experiments.

I investigate three different aspects of the quantisation process, with the aim of finding the best performing setup:

- Learning rate and learning rate schedule. By default, a Half-Exponential schedule with learning rate 10^{-1} for quantised parameters and 10^{-2} for other parameters is used.
- Which layers to quantise. By default, all convolutional layers except the input and output layers and the time embedding layers are quantised.
- Which activations to use for calculating the KD loss, and what loss function to use. By default only the output is used, which is compared between teacher and student using L1-loss.

5.1 Learning Rates

During early experiments, I found that learning rate and learning rate schedule greatly influences the results. I use a separate learning rate for the quantised parameters and the non-quantised parameters. I compare four different learning rate schedules:

Table 1: Results from a single epoch of KD using different learning rates and learning rate schedulers.

Scheduler	Quantised LR	Non-Quantised LR	Student-Teacher MAE
Half-Exponential	10^{-1}	10^{-2}	**0.0479**
Half-Exponential	10^{-1}	10^{-4}	0.0482
Half-Exponential	1	10^{-3}	0.0489
Half-Exponential	10^{-1}	10^{-3}	0.0497
Half-Exponential	10^{-2}	10^{-3}	0.0760
Half-Exponential	10^{-1}	10^{-3}	**0.0497**
Exponential	10^{-1}	10^{-3}	0.0721
Constant	10^{-1}	10^{-3}	0.0772
Cosine	10^{-1}	10^{-3}	0.0831
Constant	10^{-1}	10^{-3}	0.0845
Constant	10^{-1}	10^{-3}	0.176

Table 2: size of model and MAE after a single epoch of KD for different sets of layers excluded from quantisation.

Non-Quantised Layers				Size of	
I/O	Time Embedding	Residual	MAE	quantised layers [MB]	Total size [MB]
X			0.0428	4.1	6.3
X		X	0.0462	4.2	5.4
X	X		0.0468	4.2	5.4
X			0.0498	4.4	4.5
			0.133	4.4	4.5
	X	X	0.15	4.1	6.3

- **Constant**: Learning rate not changing during training.
- **Exponential**: Learning rate decaying exponentially to 10^{-7}.
- **Half-Exponential**: Constant Learning rate for first half of training, then decaying exponentially to 10^{-7}.
- **Cosine**: Learning rate decaying according to the first half of a cosine period.

Table 1 shows the results of altering learning rates and schedulers. All models were trained for a single epoch, without hidden layer knowledge distillation. We notice that the quantised learning rate need to be unusually high for the best performance, and the Half-Exponential scheduler is significantly outperforming other options. It is likely that training for more epochs make these differences smaller, as the training will then spend more time at any given learning rate. The learning rate of the non-quantised weights has minimal impact on the MAE.

5.2 Which layers to quantise?

As some layers might be more sensitive to quantisation than others, it might be possible to achieve a lower-MAE, while only making the model slightly larger, by choosing layers to not quantise. I explore three options for layers to exclude from quantisation:

- **I/O**: The input and output convolutional layers, as they are small and have a large impact on the performance.
- **Residual**: layers used for scaling feature size in residual connections, as leaving these in full precision allow gradients to propagate more accurately earlier layers
- **Time Embedding**: The linear layers responsible for the time embeddings.

I try six different combinations of layers, and for each train the model for one epoch. The results are found in Table 2. The input and output layers are crucial to performance of the quantised model, and excluding them from quantisation causes a negligible size increase. Both the time embedding layers and the residual layers can be excluded for a slight improvement in MAE, but does increase the size of the model by about 20%.

Table 3: MAE for different KD-losses and sets of activations and loss functions used for Knowledge Distillation

Activations used	KD-loss	MAE
Output Only	Direct	0.0471
All-layers	Direct	0.0465
Skips	Direct	0.0484
Upscale	Direct	0.0549
All-layers	Attention	0.0467
Skips	Attention	0.0516
All-layers	Projection	0.0506
Skips	Projection	0.0532

5.3 Knowledge Distillation Parameters

As discussed in Section 2, previous works have shown ternarisation to significantly benefit from hidden layer knowledge distillation. I compare three different set of activations to be used to calculate the KD loss: **All-layers**, **Skips**, **Upscale** (as introduced in Section 4.3). I also compare the three loss functions direct KD-loss, projection KD-loss and attention KD-loss, as introduced in section 3.3. A quantised model is trained for one epoch for a number of combinations of activation sets and losses, and the results are found in Table 3. Only All-layers outperforms not Output Only, and only slightly. This result is not in line with previous work [28, 18]. One possible explanation is that comparing model outputs directly already gives a rich enough signal for diffusion models, meaning that additional hidden layer supervision only serves to limit the student model in finding new suitable intermediate representations.

5.4 Final Model

Given results from previous sections, I train a final model. It is trained for 5 epochs, using the Half-Exponential learning rate schedule with learning rates 10^{-2} and 10^{-1} for the full-precision and quantised parameters respectively. The residual layers and input/output layers are not quantised. All-layers, direct KD-loss is used. The quantised model has a FID score of 113.62, which is slightly worse than the full-precision models 108.1. Figure 1 shows samples of images generated by the full-precision and quantised models. Visually, there is no clear difference in quality.

The quantised model is 14 times smaller than the full precision variant, an improvement compared to the 8 times size reduction by presented by previous diffusion model quantisation techniques [13, 14]. In terms of computational latency, the improvement is less clear. In my implementation, there is no improvement as the ternarisation is simulated using full-precision floats. In practice, with the right hardware support the ternarised model could be made to run significantly faster. All expensive Convolutions and Matrix Multiplications, which make up over 99% of all FLOPs in a ResUNet model, could be turned into masked int8 additions. [23]. Typical GPUs are not optimised for such operations, but if specialised hardware was made it is estimated that 8-bit additions require 0.8% the energy and 0.5% the area compared to 32-bit floating point multiplication, and 15% the energy compared to 8-bit multiplications [36].

6 Conclusion

This project has shown that Ternarisation of diffusion models is possible, without significantly sacrificing the quality of the model. The final model generates images that not clearly distinguishable from the full-precision model, while only requiring 7% the memory to store the model. With appropriate hardware, the ternarised model can also run inference with significantly fewer multiplications [23], with potential to massively improve speed and energy efficiency. I have also demonstrated that hidden layer knowledge distillation does not significantly improve training, and choosing appropriate learning rate and learning rate schedules is the most significant parameter.

Limitations and future work The main limitation of this project is the use of a single small model trained on the comparatively small dataset of Oxford Flowers. For this techniques to be useful in practice, it has to scale well to larger models on harder tasks. It is possible that different datasets, different size of model, and different architecture all influence the behaviour and performance of ternarisation. Experimenting with this is left for future work.

General limitations of ternarisation should also be mentioned. The main ones compared to other model compression techniques is that a potentially expensive training procedure is needed, and that special hardware would be needed to get all performance benefits.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[2] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG].

[3] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[4] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV].

[5] Weimin Tan, Siyuan Chen, and Bo Yan. *DifFSS: Diffusion Model for Few-Shot Semantic Segmentation*. 2023. arXiv: 2307.00773 [cs.CV].

[6] Tomer Amit et al. *SegDiff: Image Segmentation with Diffusion Probabilistic Models*. 2022. arXiv: 2112.00390 [cs.CV].

[7] Andreas Lugmayr et al. *RePaint: Inpainting using Denoising Diffusion Probabilistic Models*. 2022. arXiv: 2201.09865 [cs.CV].

[8] Yang Song et al. *Solving Inverse Problems in Medical Imaging with Score-Based Generative Models*. 2022. arXiv: 2111.08005 [eess.IV].

[9] Saurabh Saxena et al. *The Surprising Effectiveness of Diffusion Models for Optical Flow and Monocular Depth Estimation*. 2023. arXiv: 2306.01923 [cs.CV].

[10] Robin San-Roman, Eliya Nachmani, and Lior Wolf. *Noise Estimation for Generative Diffusion Models*. 2021. arXiv: 2104.02600 [cs.LG].

[11] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG].

[12] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. arXiv: 2102.09672 [cs.LG].

[13] Yuzhang Shang et al. *Post-training Quantization on Diffusion Models*. 2023. arXiv: 2211.15736 [cs.CV].

[14] Yefei He et al. *EfficientDM: Efficient Quantization-Aware Fine-Tuning of Low-Bit Diffusion Models*. 2023. arXiv: 2310.03270 [cs.CV].

[15] Gongfan Fang, Xinyin Ma, and Xinchao Wang. *Structural Pruning for Diffusion Models*. 2023. arXiv: 2305.10924 [cs.LG].

[16] Weijian Luo. *A Comprehensive Survey on Knowledge Distillation of Diffusion Models*. 2023. arXiv: 2304.04262 [cs.LG].

[17] Fengfu Li et al. *Ternary Weight Networks*. 2022. arXiv: 1605.04711 [cs.CV].

[18] Wei Zhang et al. *TernaryBERT: Distillation-aware Ultra-low Bit BERT*. 2020. arXiv: 2009.12812 [cs.CL].

[19] Defang Chen et al. *A Geometric Perspective on Diffusion Models*. 2023. arXiv: 2305.19947 [cs.CV].

[20] Daniel Watson et al. *Learning to Efficiently Sample from Diffusion Probabilistic Models*. 2021. arXiv: 2106.03802 [cs.LG].

[21] Cheng Lu et al. *DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps*. 2022. arXiv: 2206.00927 [cs.LG].

[22] Yang Song et al. *Consistency Models*. 2023. arXiv: 2303.01469 [cs.LG].

[23] Zhouhan Lin et al. *Neural Networks with Few Multiplications*. 2016. arXiv: 1510.03009 [cs.LG].

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[25] Foivos I. Diakogiannis et al. "ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 162 (Apr. 2020), 94–114. ISSN: 0924-2716. DOI: `10.1016/j.isprsjprs.2020.01.013`. URL: `http://dx.doi.org/10.1016/j.isprsjprs.2020.01.013`.

[26] Matthieu Courbariaux et al. *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*. 2016. arXiv: `1602.02830 [cs.LG]`.

[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: (Mar. 2015). arXiv: `1503.02531 [stat.ML]`.

[28] Sergey Zagoruyko and Nikos Komodakis. *Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer*. 2017. arXiv: `1612.03928 [cs.CV]`.

[29] Xiaoqi Jiao et al. *TinyBERT: Distilling BERT for Natural Language Understanding*. 2020. arXiv: `1909.10351 [cs.CL]`.

[30] Jangho Kim, SeongUk Park, and Nojun Kwak. *Paraphrasing Complex Network: Network Compression via Factor Transfer*. 2020. arXiv: `1802.04977 [cs.CV]`.

[31] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: `2010.02502 [cs.LG]`.

[32] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: `1912.01703 [cs.LG]`.

[33] Maria-Elena Nilsback and Andrew Zisserman. "Automated Flower Classification over a Large Number of Classes". In: *Indian Conference on Computer Vision, Graphics and Image Processing*. 2008.

[34] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: `1608.03983 [cs.LG]`.

[35] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: `1706.08500 [cs.LG]`.

[36] Amir Gholami et al. *A Survey of Quantization Methods for Efficient Neural Network Inference*. 2021. arXiv: `2103.13630 [cs.CV]`.