Append for MixMix

A. Additional Derivation and Main Proofs

In this section, we first recap the results from Gretton *et al.* [2], then we prove Theorem 4.2. From [2, Theorem 5], as known that when the RKHS is universal, we have $p = q$ if and only if $||\mu_p - \mu_q||^2_{\mathcal{H}} = 0$. The proof is illustrated as follows. First, $p = q$ implies $\text{MMD}^2[\mathcal{H}, X, Z] = 0$. Then, we only have to prove the converse. When $\mathcal{H}$ is universal, for any given $\varepsilon > 0$ and $f \in C(\mathcal{X})$, there exists a $g \in \mathcal{H}$ such that $||f - g||_\infty < \varepsilon$. Now, we simplify the notion of expectation $E_{x\sim p}, E_{z\sim q}$ to E_x, E_z and make an expansion that

$$\begin{aligned}|E_x f(x) - E_z f(z)| \leq |E_x f(x) - E_x g(x)| + \\ |E_x g(x) - E_z g(z)| + |E_z g(z) - E_z f(z)|\end{aligned} \tag{1}$$

By the universality of the RHKS, there exist a $g \in \mathcal{H}$ so that the first and the third term satisfy

$$|E_x f(x) - E_x g(x)| \leq E_x |f(x) - g(x)| \leq \varepsilon \tag{2}$$

For the second term, since $g \in \mathcal{H}$, $|E_x g(x) - E_z g(z)|$ should be no grater than $\sup(\text{MMD}[\mathcal{H}, X, Z])$. Since the squared MMD can be represented by $||\mu_p - \mu_q||^2_{\mathcal{H}}$ and is 0, we can find the second term is 0 for sure. Therefore, for any $\varepsilon > 0$, we have

$$|E_x f(x) - E_z f(z)| \leq 2\varepsilon \text{ for any } f \in C(\mathcal{X}). \tag{3}$$

Thus $p = q$ by Lemma 4.2.

A.1. Proof of Theorem 4.2

The proof of this theorem relies on the recent advance in ReLU networks universality. Without loss of generalizability, we will assume all the m networks has same maximum width w. Given an input domain $\mathcal{X} \subseteq \mathbb{R}^d$ and an output codomain $\mathcal{Y} \subseteq \mathbb{R}$, we define $L^p(\mathcal{X}, \mathcal{Y})$ as the class of L^p functions from $\mathcal{X}$ to $\mathcal{Y}$, endowed with the L^p- norm: $||f||_p = (\int_{\mathcal{X}} ||f(x)||_p^p dx)^{1/p}$. Park *et al.* [10] show that, for any $p \in [1, \infty]$, ReLU networks of width w are universal in $L^p(\mathbb{R}^d, \mathbb{R})$ if and only if $w \geq d + 1$.

We will show that the number of neural network assembled together (i.e., m) can be translated into the width of a single neural networks. Thus, we only have to ensure $m \geq \text{ceil}(\frac{d+1}{w})$. We assume the inputs as well as outputs are normalized to $[0, 1]^d$, which can be easily generalized to other cases. Following [10], we will construct each neural network into an encoder-memorizer-decoder structure. In the encoder structure, we empoly a quantization function that can quantize a scalar into binary representation. Denote $\mathcal{B}_n = \{0, 2^{-n}, 2 \times 2^{-n}, \ldots, 1 - 2^{-n}\}$ as the fixed-point set for bit-width n, the quantization function is given by:

$$q_n(x) = Binary(\max\{b \in \mathcal{B}_n : \ b \leq x\}), \tag{4}$$

where the $Binary$ function means converting the decimal representations into binary representations. For example, scalar 0.5 will be quantized to 100 using q_3 and 1000 using q_4. The quantization function will produce error that is always less than or equal to 2^{-n}. Thus, the error can be made arbitrarily small by choosing a large n.

For quantization of the input vector $X \in [0, 1]^d$, we will simply concatenate each quantized element in the vector, given by:

$$\texttt{encode}_n(X) = \sum_{i=1}^{d} q_n(X_i) \times 2^{-(i-1)n} \tag{5}$$

Note that multiplying $2^{-(i-1)n}$ for binary representations is equivalent to perform right shift $(i - 1)n$ bits. In other words, the inputs vector are encoded into a dn-bit binary representation. For the output scalar y, we can also encode it into a k-bit binary fixed-point number. Now we will construct the memorizer. The memorizer can map each quantized input to each quantized output, given by:

$$\texttt{memorize}_{n,k}(\texttt{encode}_n(X)) = \texttt{encode}_k(y) \tag{6}$$

This memorizer function can map one scalar to another scalar. And again, the information loss can be made arbitrarily small by choosing some n and k. Finally, we use a decoder in network to map the binary representation into decimal representation. The decoder can be viewed as the inverse encoder, though we cannot completely restore the original value. Han *et al.* [3] prove that any continuous function with d-dimensional input to n-dimensional outputs can be approximated using ReLU networks of width $d + n$. Therefore, the encoder-memorizer-decoder structure only requires $d+1, 2, 2$ width for each sub-module. Assume a target function f, the network function can be constructed as $q_k \circ g \circ q_n$, where g is the memorizer. Thus, for any $\varepsilon > 0$, we have

$$\sup_{x \in [0,1]^d} ||f(x) - q_k \circ g \circ q_n(x)||_\infty \le \varepsilon, \tag{7}$$

by choose large enough n, k so that $w_f(2^{-n}) + 2^{-k} \le \varepsilon$ where w_f is the modulus of continuity of f : $||f(x) - f(x'||_\infty \le w_f(||x - x'||_\infty)\ \forall x, x' \in [0,1]^d$.

In fact, the quantization representation, as well as the concatenation of the quantized input, can be assembled from different networks. For example, we can use km-bit to encode the output scalar. This implementation gives us flexibility to split the encoded output into m different k-bit quantized output scalar. And by simple linear transformation can we merge m outputs into a scalar. The same split-then-merge operation can be applied into the encoder of the input vectors. Each network can concatenate $\frac{d}{m}$ elements in the input vector.

As a consequence, each network's memorize will map the split input to the split output. Again, the information loss for each network can be made arbitrarily small by choosing sufficiently large encoding bitwidth.

B. Implementation Details

B.1. Pre-trained Model Zoo Composition

We summarize the information in Table 1.

B.2. Learning the Loss Weight

We find the BN statistic loss will vary along with models. This is because the depth and width in each model are not identical. Therefore, it is not practical to manually set the loss weight (λ in Eq. **??**) for different models. To address this problem, we adopt a similar strategy in [6] to learn the loss weight by backpropagation. In particular, we first normalized each loss term to 1 after the first calculation of the loss function. Denote the normalized loss term as $\hat{L}$, the adaptive loss weight is formulated by

$$\min_{X,\alpha} = \sum_i \left(\frac{1}{\alpha_i^2} \hat{L}_i(X) + \alpha_i^2 \right), \tag{8}$$

where $\hat{L}_i(X)$ is the normalized loss term, e.g. BN statistic loss and cross-entropy loss, and α_i is the learnable loss weight to balance the loss function. Eq. (8) can tune the loss weight based on the magnitude of each normalized loss term and prevent gradient domination. For example, if L_i becomes two low, α_i will decrease so that the gradient of $\frac{1}{\alpha_i^2} L_i$ will increase accordingly, thus preventing the gradient domination. It is worthwhile to note that we do not spend much time in finding the best hyper-parameters and learning rule for α, yet the performance of MixMix still outstrips existing methods.

B.3. Implementation Details

This section describes the hyper-parameters for image synthesis and data-free compression in detail.

For image synthesis, we use the multi-resolution training pipelines, that is to say, we first downsample the images to 112×112 and then use full resolution training to speed-up the training. We use 2.5k iterations for low-resolution optimization and 2.5k iterations for high-resolution optimization. The batch size for each GPU is set to 8. Before forward process of the images, we randomly apply flip and color jitter to the images to simulate the data augmentation. Then we apply Data Mixing ensure exact inversion. The bounding box edge ratio of the data mixing is sampled from $U(0.1, 0.4)$. We use Adam optimization with betas to (0.5, 0.5) and do not apply any L2 regularization. For optimization of loss weight, we set a constant learning rate 1e-3. The learning rate of images is set to 0.25 followed by a cosine decay schedule. After each update of the images, we clip the images to prevent the value from increasing too high.

For post-training quantization experiments, we adopt exactly the same hyper-parameters in its original paper [7]. BRECQ optimizes the rounding mechanism of weight quantization. It adopts the block-by-block feature reconstruction to optimize the weights, formalized by

$$\begin{aligned} &\arg\min_{\mathbf{v}}\ \mathbb{E}[\mathbf{W}\mathbf{x} - \hat{\mathbf{W}}\mathbf{x}] + \lambda \sum_i (1 - |2\sigma(\mathbf{v}_i) - 1|^\beta), \\ &\quad \text{subject to } \hat{\mathbf{W}} = s \times \text{clip}\left(\lfloor \frac{\mathbf{W}}{s} \rfloor + \sigma(\mathbf{v}), n, p \right). \end{aligned} \tag{9}$$

The variable $\mathbf{v}$ can determine rounding up or rounding down by using a sigmoid function $\sigma(\cdot)$. The regularization term in Eq. (9) ensures $\sigma(\mathbf{v})$ can converge to 0 or 1. n and p are the limit integer restricted by bit-width and λ, β are the hyper-parameters. λ is set to 0.1 and β will decrease in training process to enhance the regularization.

For quantization-aware training, we use the intermediate feature quantization loss proposed in [4] and the loss weight is set to 300. To prevent overfitting of the model, we use random color jitter (0.2, 0.2, 0.2, 0.1), random grad scale (probability=0.2), random Gaussian blur (1, 2, probability=0.5) and random horizontal flip. We freeze the update of BN statistics when the training iteration reaches 10000 since we find updating BN statistics will alter the real activation distribution. The learning rate is set to 0.04 and will be multiplied by 0.1 at iteration 4000, 10000, 20000. Weight decay is set to the same as the full precision model. A recent benchmark paper [8] gives a comprehensive study of QAT, we do not use such settings since generating 1.2M synthesized images taks too much time. We also avoid studing mixed precision quantization [1, 9] due to the various search space in existing work. We find image quality does not necessarily corresponds to the precision search results. The quality of images should be verified via training.

Table 1. Pre-trained model zoo for generating MixMix dataset. AD means average down and deep stem layers as proposed in [5].

Model	Acc.	Model	Acc.	Model	Acc.
ResNet-34	74.03	MobileNetV2_1.0	73.15	DenseNet-201	77.59
ResNet-50AD	79.03	MobileNetV2_2.0	77.81	ShuffleNetV2_1.5	72.82
ResNet-101AD	80.23	MNasNet_1.0	74.02	ShuffleNetV2_2.0	74.47
ResNet-152AD	80.87	MNasNet_2.0	76.68	MobileNetV3_Large_1.0	75.29
RegNetX-600MF	74.23	VGG16_BN	73.68	MobileNetV3_Large_1.4	77.15
RegNetX-1600MF	77.29	SE-Net-50	79.00	NASNet-7ms	75.77
RegNetX-3200MF	78.72	DenseNet-121	75.77	NASNet-10ms	77.71

For data-free pruning experiments, we use a *two-stage* algorithm to prune the network. In stage1, we directly find the weights that has lowest L1 norm, and then add a 0-1 mask to the weights. Then, we jointly train the mask and the weights, given by:

$$\arg\min_{M,\hat{\mathbf{W}}} \mathbb{E}[\mathbf{W}\mathbf{x} - (M \odot \hat{\mathbf{W}})\mathbf{x}] \tag{10}$$

After reconstruction, we will absorb the mask and the weights and then find the weights with the lowest L1 norm again. In Stage2, we will only optimize the weights and freeze mask, so that the sparsity will hold after the reconstruction. In each stage we optimize weights or mask for 5k iterations with learning rate 4e-5 followed by a cosine annealing schedule.

C. The Choice of Model Zoo

Sometimes it is not possible to find 20 pretrained models on some dataset. Therefore we evaluate a tiny version of MixMix. Here we build two tiny zoo: the first one contains MBV2, MBV3 and MNasNet; the second one contains Res-18, -34, -50. We call them MixMix-mobile and MixMix-residual, respectively.

Generally, we find add more models can improve the performance compared with single model method. However, we still observed a decrease in generalizability on both MixMix-mobile and MixMix-residual. Therefore we encorage mixing models from multiple architecture familiy to increase the generalizability.

Data Source	MobileNetV2	MNasNet	ResNet-18	ResNet-50
Training Set	64.63	58.86	69.52	74.72
Single-Model	59.81	55.48	69.08	74.05
MixMix	64.01	57.87	70.59	74.58
MixMix-mobile	64.13	57.45	69.23	74.13
MixMix-residual	61.65	56.98	69.74	74.61

References

[1] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[2] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[3] Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

[4] Matan Haroush, Itay Hubara, Elad Hoffer, and Daniel Soudry. The knowledge within: Methods for data-free model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2020.

[5] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.

[6] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[7] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

[8] Yuhang Li, Mingzhu Shen, Jian Ma, Yan Ren, Mingxin Zhao, Qi Zhang, Ruihao Gong, Fengwei Yu, and Junjie Yan. MQBench: Towards reproducible and deployable model quantization benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[9] Yuhang Li, Wei Wang, Haoli Bai, Ruihao Gong, Xin Dong, and Fengwei Yu. Efficient bitwidth search for practical mixed precision neural network. *arXiv preprint arXiv:2003.07577*, 2020.

[10] Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. Minimum width for universal approximation. In *International Conference on Learning Representations*, 2021.

[11] Mingzhu Shen, Kai Han, Chunjing Xu, and Yunhe Wang. Searching for accurate binary neural architectures. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.