Supplementary Material
MUSE:Feature Self-Distillation with Mutual Information and Self-Information

Yu Gong*[2]
gongyug@sfu.ca
Ye Yu*[1]
yu.ye@microsoft.com
Gaurav Mittal[1]
gaurav.mittal@microsoft.com
Greg Mori[2]
mori@cs.sfu.ca
Mei Chen[1]
mei.chen@microsoft.com

[1] Microsoft
[2] Simon Fraser University

1 Theoretical Details

The proposed ***MUSE*** incorporates two essential components—mutual information (MI) and self-information (SI). As discussed in the main manuscript, we utilize neural networks to estimate both MI and SI based on the state-of-the-art MI neural estimator. We describe here in more detail the MI neural estimator and its implementation. One may refer to Mutual Information Neural Estimation (MINE) [1] and DeepInfoMax [3] for more details if interested. Here, we briefly give an introduction on MI neural estimator and the way we use it to formulate our own MI and SI estimators.

Let X and Y denote two random variables with marginal densities $X \sim p(x)$ and $Y \sim p(y)$, respectively. One form of the MI between X and Y is the Kullback–Leibler (KL) divergence between the joint and the product of the marginals .

$$I(X;Y) = D_{\mathrm{KL}}(p(x,y)||p(x)p(y)) \tag{1}$$

KL divergence representation. Two types of representation of KL divergence can be used to formulate the bound of MI. One is to use Donsker-Varadhan representation [2],

$$I(X;Y) \geq \mathbb{E}_{p(x,y)}[T] - \log\left(\mathbb{E}_{p(x)p(y)}[e^T]\right) \tag{2}$$

Another one is f-divergence representation [4],

$$I(X;Y) \geq \mathbb{E}_{p(x,y)}[T] - \mathbb{E}_{p(x)p(y)}[e^{T-1}] \tag{3}$$

* Authors with equal contribution. Work was done when Yu Gong was a research intern at Microsoft.

MINE [1] proposed to use neural networks ϕ to approximate the function $T_\phi(x,y) : \Omega \mapsto \mathbb{R}$ in R.H.S. of Eq. 2 and Eq. 3. It is empirically shown to effectively bound the MI.

Multi-view formulation. If X and Y are the features in a neural network, there is a non-linear mapping modeled by the hidden layers that $Y \sim g(X)$. Let $X^{(1)}$ and $X^{(2)}$ be different *views* of X and encoded by arbitrary functions g_1 and g_2. It is shown in [5] that $\max_{g_1,g_2} I\big(g_1(X^{(1)}); g_2(X^{(2)})\big)$ can serve as a lower bound of $\max_g I(X; g(X))$. Therefore, a multi-view formulation on X or Y can be constructed to allow more modeling flexibility, while maximizing this new lower bound. We follow DeepInfoMax [3] to estimate MI. It employs the *global* and *local* structure to better exploit the patterns from different levels in a convolutional neural network.

2 Experimental Details

All implementations are conducted in PyTorch. For all the models with official or publicly verified code, we rerun the baseline experiments on our computing machines. For those without code, we re-implement them based on the details provided by the original papers.

2.1 MI estimator

Empirically, we find that the realization of the MI estimator in DeepInfoMax [3] provides more stable and better performance on intermediate features. Hence, we follow [3] to formulate a global-local view to prioritize global or local information flexibly. It is based on a Jensen-Shannon MI estimator (Eq. 3) using the softplus function,

$$\hat{I}^{(\mathrm{JSD})}_{\theta,\phi}(F_i; F_k) := \mathbb{E}_{\mathbb{P}}[-\mathrm{sp}\big(-T_\phi(f_i, f_k)\big)] - \mathbb{E}_{\tilde{\mathbb{P}}}[\mathrm{sp}\big(T_\phi(f_i, f_k)\big)] \tag{4}$$

where θ and ϕ are the parameters of the backbone network and MI neural estimator, respectively. F_k is the feature of the base module. $\mathbb{P}$ is the joint density $p(f_i, f_k)$ directly drawn from the network. $\tilde{\mathbb{P}}$ is the product of the marginal densities. sp is the softplus function. To sample from $\tilde{\mathbb{P}}$, one trick[1] is used to avoid extra sampling of the dataset — rotating the feature map by one instance to create a new permutation of instances, so each feature is now paired with the feature from another image, rather than drawn from the joint density. For feature F, the implementation of its reordered samples in PyTorch is `F_prime = torch.cat((F[1:], F[0].unsqueeze(0)), dim=0)`.

To estimate MI, for each feature of interest F_i, a global-view discriminator and a local-view discriminator are constructed between F_i and the base feature. In the following, we introduce the experimental settings in Section 4.1 of main manuscript — Module 4 as the base, and estimating MI between Module 1-3 and Module 4 with ***global*** and ***local*** structure, as mentioned in Section 3.2 of main paper. In MUSE, MI is estimated between intermediate features and last feature of a CNN, though the original MI estimators are only utilized for input-output pair.

Global Discriminator. Before concatenating f_i with f_4, two downsampling convolutional layers with a kernel size of 3 are used to reduce the dimensionality. For the case of using Module 4 as the base, f_4 is the output of the last convolutional layer. Then, the downsampled

[1] https://github.com/DuaneNielsen/DeepInfomaxPytorch

f_i and f_4 are concatenated and fed to two 512-unit fully-connected (FC) layers to produce a 1 dim. scalar.

Local Discriminator. We concatenate the global feature vector (f_4 in our case) with feature map f_i at every location. And then two downsampling convolutional layers with a kernel size of 3 is used to reduce the dimensionality of the concatenated feature map to a scalar value (1-dimensional).

Self Discriminator. To estimate SI, we use the same strategy as global discriminator but with two features from an identical distribution. We first concatenate the features and then feed the feature to two 512-unit FC layers to produce a scalar. We do not use the global-local view for self-information, since the input features are from the same distribution.

The discriminator $T_\phi(f_i, f_k)$ is used to score the feature pair. The MI is estimated by the summation of Eq. 4 in both global and local views. The self-information is estimated by the single view of the self discriminator.

2.2 Self-distillation

Model Architecture. In all the backbone architectures including a feature extractor and some task-specific heads, we first decompose the feature extractor into multiple modules (four in our settings). In each module, the intermediate feature is first fed to a bottleneck layer to have a consistent dimension (512-dimensional). Then the same task-specific heads will be added to the output of each module to perform the same task individually. The entire network is jointly trained using gradient-based optimization. The training details can be found in the Experiments section of the main manuscript.

For different backbone networks, the decomposition might be different based on the actual architectures. We aim to simplify the decomposition—if the backbone network has explicit stages like ResNet and the number of stages is the same as the length of decomposition, we directly decompose each stage as a module. For those without obvious decomposition configuration, we decompose them at the point where the feature map size is changed, possibly making different features more diverse in terms of size. As pointed out in Section 4.1 in main paper of ***model decomposition***, for classification backbone networks, our experiments are based on a 4-module decomposition. The decomposition points of each backbone network are (we use the original terms of the building blocks for each architecture):

- VGG19: layer 6, layer 11, layer 16;
- ResNet: stage 1, stage 2, stage 3;
- DenseNet121: stage 1, stage 2, stage 3;
- NASNet: cell layer 3, cell layer 6, cell layer 9;
- EfficientNet-B0: block 4, block 7, block 15;

Baselines. For self-distillation on image classification in main manuscript Section 4.1, we choose two lines of experiments to validate that MUSE effectively outperforms other state-of-the-arts. We first compare MUSE with BYOT as it shares the same idea of module decomposition. As the authors did not release their code, our implementation is based on a

public re-implementation code[2] and follows reported details to reproduce all numbers; We also compare with other state-of-the-art self-distillation methods. We run the released code with their hyperparameter settings, or directly adopt the reported numbers.

2.3 Online / Offline distillation

In online / offline distillation settings , MUSE is calculated between the intermediate features of the student networks (Module 1-3) and the last feature of the teacher networks. Note that only the last feature of the teacher network is used. For online distillation, we follow the conventional settings: two identical networks and two different networks as teacher-stduent. We report the average accuracy for two identical networks. We consider the Net 2 as teacher network. On CIFAR100, resnet with MUSE is trained using SGD with momentum 0.9, weight decay 5e-4, batch size 64, and the learning rate initialized as 0.05 and divided by 10 after epoch 150, 180 and 210; ShuffleNetV1 with MUSE is trained using SGD with momentum 0.9, weight decay 5e-4, batch size 64, and the learning rate initialized as 0.01 and divided by 10 after epoch 150, 180 and 210. For offline distillation, we freeze the pretrained teacher network. On CIFAR100, ResNet with MUSE is trained using SGD with momentum 0.9, weight decay 5e-4, batch size 128, and the learning rate initialized as 0.1 and divided by 10 after epoch 150, 180 and 210. The decomposition is only done for the student network (resnet and ShuffleNet):

- ResNet: stage 1, stage 2, stage 3;
- ShuffleNetV1: Maxpool, stage 2, stage 3;

References

[1] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *ICML*, 2018.

[2] M. D. Donsker and S. R.S. Varadhan. Asymptotic evaluation of certain markov process expectations for large time. iv. *Communications on Pure and Applied Mathematics*, 36 (2):183–212, 1983.

[3] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

[4] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *NeurIPS*, 2016.

[5] Michael Tschannen, Josip Djolonga, Paul K. Rubenstein, Sylvain Gelly, and Mario Lucic. On mutual information maximization for representation learning. In *ICLR*, 2020.

[2] https://github.com/luanyunteng/pytorch-be-your-own-teacher