Boost Vision Transformer with GPU-Friendly Sparsity and Quantization Appendix

Chong Yu[1,2] Tao Chen[3,*] Zhongxue Gan[1,*] Jiayuan Fan[1]
[1]Academy for Engineering and Technology, Fudan University [2]NVIDIA Corporation
[3]School for Information Science and Technology, Fudan University

1. Outline

In this **Appendix**, we will provide some supplementary materials and more experimental results for the proposed **GPUSQ-ViT** compression scheme beyond the tight page limitation in manuscript. The detailed outline is as follows.

Section 2 aims to further support the contents in **Section 3** of the manuscript.

- **Sub section 2.1** derives the storage saving of 2:4 structured sparsity for FP16, INT8, and INT4 data types.
- **Sub section 2.2** provides the details and whole workflow of the **GPUSQ-ViT** algorithm.

Section 3 aims to further support the contents in **Section 4** of the manuscript.

- **Table 1** provides the hyper-parameters settings of each experiment for easy reproduction.
- **Sub section 3.1** provides **GPUSQ-ViT** compression results on other classification models, and proves it is ***orthogonal*** to token reduction compression method.
- **Sub section 3.2** provides visualization results of critical feature maps from vision transformer models.
- **Sub section 3.3** further explains the influence of adjustment factors with the ablation study, and analyzes why **GPUSQ-ViT** has such a good compression effect.

2. Boost vision transformer on GPU

GPUSQ-ViT mainly contains **2:4 structured sparse pruning** and **sparse-distillation-aware QAT** workflows. We further explain the 2:4 sparse pattern for storage saving in section 2.1, and provide the detailed steps of **GPUSQ-ViT** algorithm in section 2.2.

2.1. Fine-grained structured sparsity on GPU

The 2:4 sparsity uses 2-bit metadata per non-zero element to indicate the position of two non-zero elements in every four adjacent elements in a row of matrix A. For matrix A with FP16 data format, storing four adjacent elements as a dense pattern requires $4 \times 16bits = 64bits$, while storing as a 2:4 sparse pattern requires $2 \times 16bits + 2 \times 2bits = 36bits$. For matrix A with INT8 data format, storing as dense and 2:4 sparse pattern requires $4 \times 8bits = 32bits$ and $2 \times 8bits + 2 \times 2bits = 20bits$, respectively. The 2:4 sparsity instruction for the INT4 data format differs from FP16 and INT8. Matrix A is defined as a pair-wise structured sparse at a granularity of 4:8. In other words, each chunk of eight adjacent elements in a row of matrix A has four zero and four non-zero values. Further, the zero and non-zero values are clustered in sub-chunks of two elements each within the eight-wide chunk, i.e., each two-wide sub-chunk within the eight-wide chunk must be all zeroes or all non-zeroes. Only the four non-zero values are stored in the compressed matrix, and two 2-bit indices in the metadata indicate the position of the two two-wide sub-chunks with non-zero values in the eight-wide chunk of a row of matrix A. So for matrix A with INT4 data format, storing as dense and sparse pattern also requires $8 \times 4bits = 32bits$ and $4 \times 4bits + 2 \times 2bits = 20bits$, respectively. In conclusion, the sparse format for FP16, INT8, and INT4 lead to 43.75%, 37.5%, and 37.5% savings in storage. **GPUSQ-ViT** will firstly compress the model as 2:4 FP16 sparse, then further quantize to 2:4 INT8 or INT4 sparse for best efficiency.

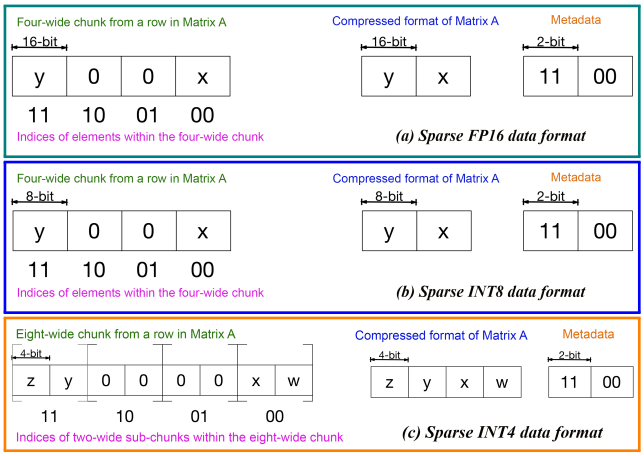

Figure 1. Storage formats for ***2:4 fine-grained structured sparse pattern*** and metadata with FP16, INT8 and INT4 operators. (w,x,y,z denote the non-zero elements.)

Algorithm 1: GPUSQ-ViT: vision transformers compression with ***2:4 sparsity*** and ***sparse-distillation-aware QAT***

Input: Dense floating-point model $\boldsymbol{M}_{DF}$ which contains K stages of transformer blocks, Input images x

Data: Distillation temperature t, Loss adjustment factors for hard label, soft logits and feature: α, β, γ, Adjustment factors for feature-based distillation of critical layers in different stages: $\theta_i, i = 1 \cdots K$, Overall pruning loss threshold δ_{prune}, Overall calibration loss threshold $\delta_{calibrate}$

Output: Sparse quantized model $\boldsymbol{M}_{SQ}$

/* ***2:4 structured sparse pruning*** compression workflow */

Initialize sparse floating-point model $\boldsymbol{M}_{SF}$ with the weight parameters from dense floating-point model $\boldsymbol{M}_{DF}$

while *Overall sparse pruning loss:* $\boldsymbol{L}_{prune}$ *is larger than threshold* δ_{prune} **do**

Get feature maps of critical layers from $\boldsymbol{M}_{DF}$ and $\boldsymbol{M}_{SF}$, e.g., patch embedding: $F_{patch}^{\boldsymbol{M}_{DF}}$ and $F_{patch}^{\boldsymbol{M}_{SF}}$, last layer of last transformer block in stage i: $F_{tfblock_i}^{\boldsymbol{M}_{DF}}$ and $F_{tfblock_i}^{\boldsymbol{M}_{SF}}$, and the final projection layer: $F_{fproj}^{\boldsymbol{M}_{DF}}$ and $F_{fproj}^{\boldsymbol{M}_{SF}}$

// Calculate ***feature-based distillation loss*** with mean-squared-error (MSE) criterion

$$\boldsymbol{L}_{feature}^{prune} = \boldsymbol{L}_{MSE}\left(F_{patch}^{\boldsymbol{M}_{DF}}, F_{patch}^{\boldsymbol{M}_{SF}}\right) + \sum_{i=1}^{K}\left[\theta_i \cdot \boldsymbol{L}_{MSE}\left(F_{tfblock_i}^{\boldsymbol{M}_{DF}}, F_{tfblock_i}^{\boldsymbol{M}_{SF}}\right)\right] + \boldsymbol{L}_{MSE}\left(F_{fproj}^{\boldsymbol{M}_{DF}}, F_{fproj}^{\boldsymbol{M}_{SF}}\right)$$

// Calculate ***hard label distillation loss*** with cross entropy (CSE) criterion

if *Ground truth labels:* $label_{Ground}$ *of input images* x *exists* **then**

$$\boldsymbol{L}_{hard\_label}^{prune} = \boldsymbol{L}_{CSE}\left(label_{Ground}, \boldsymbol{M}_{SF}\left(x; T=1\right)\right)$$

else

$$\boldsymbol{L}_{hard\_label}^{prune} = \boldsymbol{L}_{CSE}\left(\boldsymbol{M}_{DF}\left(x; T=1\right), \boldsymbol{M}_{SF}\left(x; T=1\right)\right)$$

end

// Calculate ***soft logits distillation loss*** with Kullback Leibler divergence (KLD) criterion

$$\boldsymbol{L}_{soft\_logits}^{prune} = \boldsymbol{L}_{KLD}\left(\boldsymbol{M}_{DF}\left(x; T=t\right), \boldsymbol{M}_{SF}\left(x; T=t\right)\right)$$

Calculate the overall sparse pruning loss: $\boldsymbol{L}_{prune} = \alpha * \boldsymbol{L}_{hard\_label}^{prune} + \beta * \boldsymbol{L}_{soft\_logits}^{prune} + \gamma * \boldsymbol{L}_{feature}^{prune}$

Minimize the overall sparse pruning loss w.r.t weight parameters of sparse floating-point model $\boldsymbol{M}_{SF}$

end

/* ***sparse-distillation-aware QAT*** compression workflow */

Initialize sparse quantized model $\boldsymbol{M}_{SQ}$ by PTQ the weight parameters from ***trained*** sparse floating-point model $\boldsymbol{M}_{SF}$

while *Overall quantization calibration loss:* $\boldsymbol{L}_{calibrate}$ *is larger than threshold* $\delta_{calibrate}$ **do**

Get feature maps of critical layers from $\boldsymbol{M}_{SF}$ and $\boldsymbol{M}_{SQ}$, e.g., patch embedding: $F_{patch}^{\boldsymbol{M}_{SF}}$ and $F_{patch}^{\boldsymbol{M}_{SQ}}$, last layer of last transformer block in stage i: $F_{tfblock_i}^{\boldsymbol{M}_{SF}}$ and $F_{tfblock_i}^{\boldsymbol{M}_{SQ}}$, and the final projection layer: $F_{fproj}^{\boldsymbol{M}_{SF}}$ and $F_{fproj}^{\boldsymbol{M}_{SQ}}$

Calculated the sparse-distillation-aware weight factor:

$$W_j = \begin{cases} \boldsymbol{L}_{MSE}\left(F_{patch}^{\boldsymbol{M}_{DF}}, F_{patch}^{\boldsymbol{M}_{SF}}\right) / \boldsymbol{L}_{feature}^{prune}, & j = 0 \\ \theta_i \cdot \boldsymbol{L}_{MSE}\left(F_{tfblock_j}^{\boldsymbol{M}_{DF}}, F_{tfblock_j}^{\boldsymbol{M}_{SF}}\right) / \boldsymbol{L}_{feature}^{prune}, & j = 1, \cdots, K \\ \boldsymbol{L}_{MSE}\left(F_{fproj}^{\boldsymbol{M}_{DF}}, F_{fproj}^{\boldsymbol{M}_{SF}}\right) / \boldsymbol{L}_{feature}^{prune}, & j = K+1 \end{cases}$$

// Calculate ***feature-based calibration loss*** with mean-squared-error (MSE) criterion

$$\boldsymbol{L}_{feature}^{calibrate} = W_0 \cdot \boldsymbol{L}_{MSE}\left(F_{patch}^{\boldsymbol{M}_{SF}}, F_{patch}^{\boldsymbol{M}_{SQ}}\right) + \sum_{i=1}^{K}\left[W_i \cdot \boldsymbol{L}_{MSE}\left(F_{tfblock_i}^{\boldsymbol{M}_{SF}}, F_{tfblock_i}^{\boldsymbol{M}_{SQ}}\right)\right] + W_{K+1} \cdot \boldsymbol{L}_{MSE}\left(F_{fproj}^{\boldsymbol{M}_{SF}}, F_{fproj}^{\boldsymbol{M}_{SQ}}\right)$$

// Calculate ***hard label calibration loss*** with cross entropy (CSE) criterion

if *Ground truth labels:* $label_{Ground}$ *of input images* x *exists* **then**

$$\boldsymbol{L}_{hard\_label}^{calibrate} = \boldsymbol{L}_{CSE}\left(label_{Ground}, \boldsymbol{M}_{SQ}\left(x; T=1\right)\right)$$

else

$$\boldsymbol{L}_{hard\_label}^{calibrate} = \boldsymbol{L}_{CSE}\left(\boldsymbol{M}_{SF}\left(x; T=1\right), \boldsymbol{M}_{SQ}\left(x; T=1\right)\right)$$

end

// Calculate ***soft logits calibration loss*** with Kullback Leibler divergence (KLD) criterion

$$\boldsymbol{L}_{soft\_logits}^{calibrate} = \boldsymbol{L}_{KLD}\left(\boldsymbol{M}_{SF}\left(x; T=t\right), \boldsymbol{M}_{SQ}\left(x; T=t\right)\right)$$

Calculate the overall quantization calibration loss: $\boldsymbol{L}_{calibrate} = \alpha * \boldsymbol{L}_{hard\_label}^{calibrate} + \beta * \boldsymbol{L}_{soft\_logits}^{calibrate} + \gamma * \boldsymbol{L}_{feature}^{calibrate}$

Minimize the overall quantization calibration loss w.r.t weight and scale factor parameters of sparse quantized model $\boldsymbol{M}_{SQ}$

end

2.2. Overall GPUSQ-ViT compression method

In **GPUSQ-ViT**, **2:4 structured sparse pruning** aims to compress the dense floating-point model $\boldsymbol{M}_{DF}$ as sparse floating-point $\boldsymbol{M}_{SF}$. **Sparse-distillation-aware QAT** aims to further compress the sparse floating-point model $\boldsymbol{M}_{SF}$ as the sparse quantized model $\boldsymbol{M}_{SQ}$ on data format. The details about **GPUSQ-ViT** are provided in Algorithm 1.

Network	Optimizer	Initial LR	LR schedule	Momentum	Weight Decay	Epochs	Batch Size	GPU Num
DeiT-Tiny[1]	AdamW	0.0005	Cosine Annealing	0.9	0.05	300	256	8
DeiT-Small[1]	AdamW	0.0005	Cosine Annealing	0.9	0.05	300	256	8
DeiT-Base[1]	AdamW	0.0005	Cosine Annealing	0.9	0.05	300	64	16
Swin-Tiny (224^2)[2]	AdamW	0.001	Cosine Annealing	0.9	0.05	300	128	8
Swin-Small (224^2)[2]	AdamW	0.001	Cosine Annealing	0.9	0.05	300	128	8
Swin-Base (224^2)[2]	AdamW	0.001	Cosine Annealing	0.9	0.05	300	128	8
Swin-Base (384^2)[2]	AdamW	0.00002	Cosine Annealing	0.9	1.0e-08	30	64	8
Swin-V2-Tiny (256^2)[2]	AdamW	0.001	Cosine Annealing	0.9	0.05	300	128	8
Swin-V2-Small (256^2)[2]	AdamW	0.001	Cosine Annealing	0.9	0.05	300	64	8
Swin-V2-Base (256^2)[2]	AdamW	0.001	Cosine Annealing	0.9	0.05	300	64	8
A-ViT-Tiny (224^2)[3]	AdamW	0.0005	Cosine Annealing	0.9	0.05	100	128	4
A-ViT-Small (224^2)[3]	AdamW	0.0003	Cosine Annealing	0.9	0.05	100	96	4
Mask R-CNN (Swin-Tiny)[4]	AdamW	0.0001	Multi-Step (milestone: 27,33)	0.9	0.05	36	2	8
Mask R-CNN (Swin-Small)[4]	AdamW	0.0001	Multi-Step (milestone: 27,33)	0.9	0.05	36	2	8
Cascade Mask R-CNN (Swin-Tiny)[4]	AdamW	0.0001	Multi-Step (milestone: 27,33)	0.9	0.05	36	2	8
Cascade Mask R-CNN (Swin-Small)[4]	AdamW	0.0001	Multi-Step (milestone: 27,33)	0.9	0.05	36	2	8
Cascade Mask R-CNN (Swin-Base)[4]	AdamW	0.0001	Multi-Step (milestone: 27,33)	0.9	0.05	36	2	8
DETR (ResNet50)[5]	AdamW	0.0001	Step (every 200 steps)	0.9	0.0001	500	2	8
Deformable DETR (ResNet50)[6]	AdamW	0.0002	Step (every 40 steps)	0.9	0.0001	50	2	8
UperNet (Swin-Tiny)[7]	AdamW	0.00006	Polynomial	0.9	0.01	127	2	8
UperNet (Swin-Small)[7]	AdamW	0.00006	Polynomial	0.9	0.01	127	2	8
UperNet (Swin-Base)[7]	AdamW	0.00006	Polynomial	0.9	0.01	127	2	8

Table 1. Experiments hyper-parameters for the classification, object detection and segmentation models tested in this paper.

3. Experiments

For the experiments in this paper, we choose PyTorch with version 1.12.0 as the framework to implement all algorithms. The results of the dense model training, sparse compression, and QAT experiments are obtained with A100 GPU clusters. All the reference algorithms use the default data type provided in public repositories.

For classification (DeiT[1], Swin V1 and V2[2], A-ViT[3]), object detection (Mask R-CNN[4], DETR[5], Deformable-DETR[6]) and segmentation (UperNet[7]) networks, we follow the hyper-parameters settings in public repositories marked by the footnotes and detailed list in Table 1. Multiple A100 GPUs are used for data-parallel training in each training or fine-tuning experiment.

3.1. Compression efficacy for classification task

To evaluate the compression efficacy of **GPUSQ-ViT** and make the comparison with prior arts on the image classification task, DeiT [2][1], Swin Transformer V1 and V2 [1][2], A-ViT [3][3] are chosen as the experiment target models. The results of DeiT and Swin Transformer V1 are already shown in the manuscript. In **Appendix**, we compress the Swin Transformer V2 and A-ViT models with **GPUSQ-ViT**. We do not have full comparison results with the state-of-the-art vision transformer compression methods, because the prior methods do not try to compress these models. For **GPUSQ-ViT**, the loss adjustment factors for hard label, soft logits and feature-based losses apply $\alpha = 1$, $\beta = 10$, and $\gamma = 5$), respectively. The model size and FLOPs comparison results are shown in Table 2.

Model	Method	Input	Format	Params (M)	FLOPs (G)	Top-1 Acc(%)	Top-5 Acc(%)
A-ViT-Tiny	*Baseline*	224^2	FP32	5	0.8	71.4	90.4
	GPUSQ-ViT		INT8	0.78 (6.4×)	0.03 (31×)	71.4 (+0.0)	90.5 (+0.1)
	GPUSQ-ViT		INT4	0.39 (12.7×)	0.01 (62×)	70.9 (-0.5)	89.9 (-0.5)
A-ViT-Tiny	*Baseline*	224^2	FP32	22	3.6	78.8	93.9
	GPUSQ-ViT		INT8	3.44 (6.4×)	0.12 (31×)	78.9 (+0.1)	94.1 (+0.2)
	GPUSQ-ViT		INT4	1.72 (12.7×)	0.06 (62×)	78.5 (-0.3)	93.6 (-0.3)
Swin-V2-Tiny	*Baseline*	256^2	FP32	28	6.6	82.8	96.2
	GPUSQ-ViT		INT8	4.38 (6.4×)	0.21 (31×)	82.9 (+0.1)	96.2 (+0.0)
	GPUSQ-ViT		INT4	2.19 (12.7×)	0.11 (62×)	82.4 (-0.4)	96.0 (-0.2)
Swin-V2-Small	*Baseline*	256^2	FP32	50	12.6	84.1	96.8
	GPUSQ-ViT		INT8	7.81 (6.4×)	0.41 (31×)	84.1 (+0.0)	96.7 (-0.1)
	GPUSQ-ViT		INT4	3.91 (12.7×)	0.20 (62×)	84.0 (-0.1)	96.6 (-0.2)
Swin-V2-Base	*Baseline*	256^2	FP32	88	21.8	84.6	97.0
	GPUSQ-ViT		INT8	13.75 (6.4×)	0.70 (31×)	84.6 (+0.0)	97.0 (+0.0)
	GPUSQ-ViT		INT4	6.88 (12.7×)	0.35 (62×)	84.4 (-0.2)	96.9 (-0.1)

Table 2. Effectiveness of **GPUSQ-ViT** on classification task.

We can apply **GPUSQ-ViT** to compress each vision transformer model as INT8 and INT4 versions. For INT8 compressed models, **GPUSQ-ViT** can bring 6.4× reduction for model size and 31× reduction for FLOPs with negligible accuracy drop. For INT4 compressed models, **GPUSQ-ViT** can get 12.7× and 62× reduction for model size and FLOPs with a small accuracy drop.

Moreover, A-ViT method already compress the vision transformer models by automatically reducing the number of tokens that are processed in the model inference process. We can find the **GPUSQ-ViT** compression is ***orthogonal*** to such token reduction compression method, i.e., **GPUSQ-ViT** can further compress the A-ViT models to a more efficient version with negligible accuracy drop.

3.2. Visualization results

To compare between dense and **GPUSQ-ViT** compressed models in visualization, we choose the output of layer norm after patch embedding, each patch merging layer after the last Swin Transformer block in each stage, and the final layer norm of the model, to generate the Class Activation Map (CAM) results. The CAM results of Swin-V1-Tiny, Swin-V1-Base, Swin-V2-Tiny, Swin-V2-Base are shown in Figure 2, 3, 4, and 5, respectively.

[1] https://github.com/facebookresearch/deit
[2] https://github.com/microsoft/Swin-Transformer
[3] https://github.com/NVlabs/A-ViT
[4] https://github.com/SwinTransformer/Swin-Transformer-Object-Detection
[5] https://github.com/facebookresearch/detr
[6] https://github.com/fundamentalvision/Deformable-DETR
[7] https://github.com/SwinTransformer/Swin-Transformer-Semantic-Segmentation

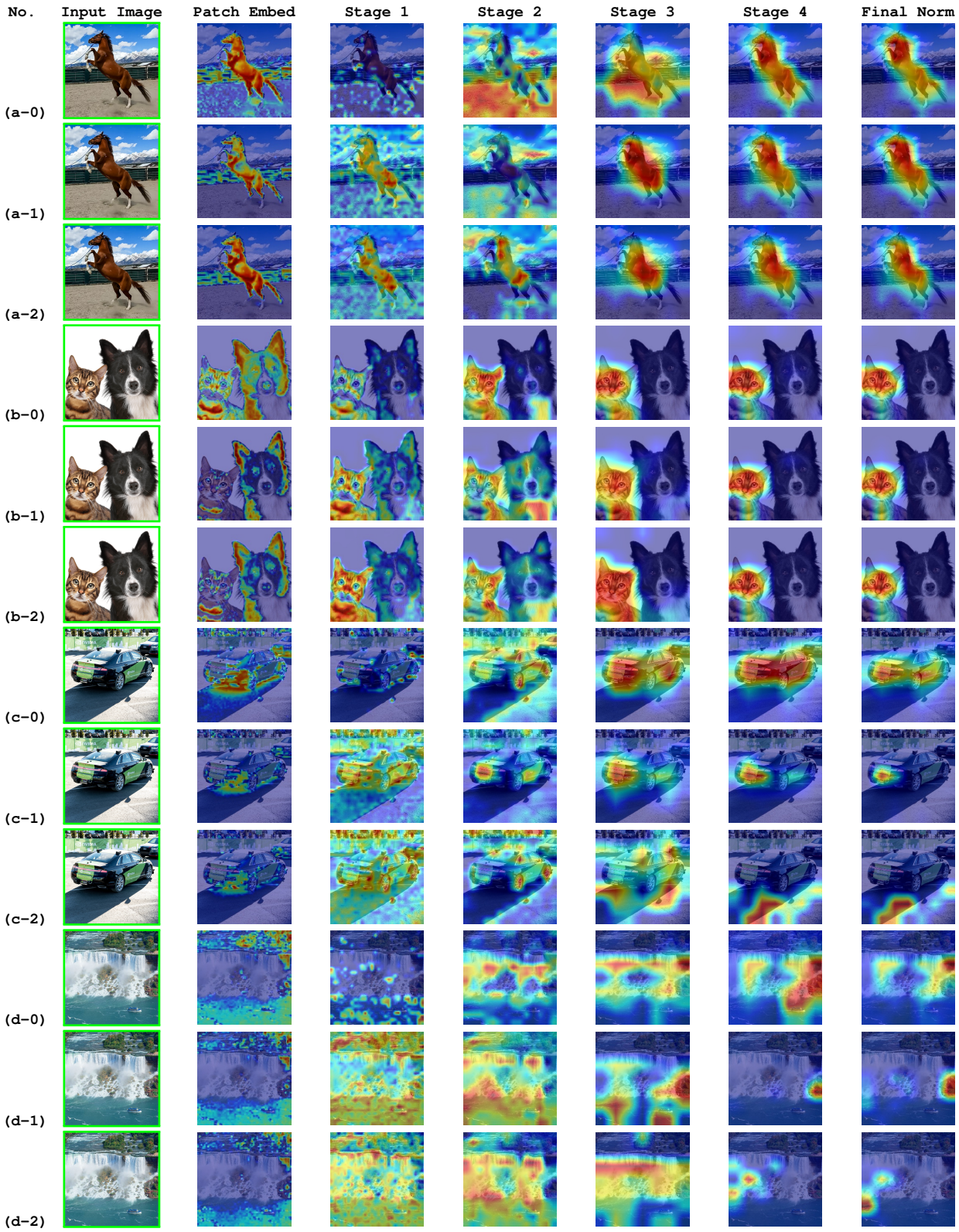

Figure 2. Attention map visualization for Swin Transformer V1 Tiny. We choose the output of layer norm after patch embedding, each patch merging layer after the last Swin Transformer block in each stage, and the final layer norm of the model, to generate the Class Activation Map (CAM) visualization results. **(a/b/c/d-0)** are CAM results of dense pretrained model on ImageNet-1K dataset, **(a/b/c/d-1)** are CAM results of **GPUSQ-ViT** compressed INT8 models. **(a/b/c/d-2)** are CAM results of **GPUSQ-ViT** compressed INT4 models.

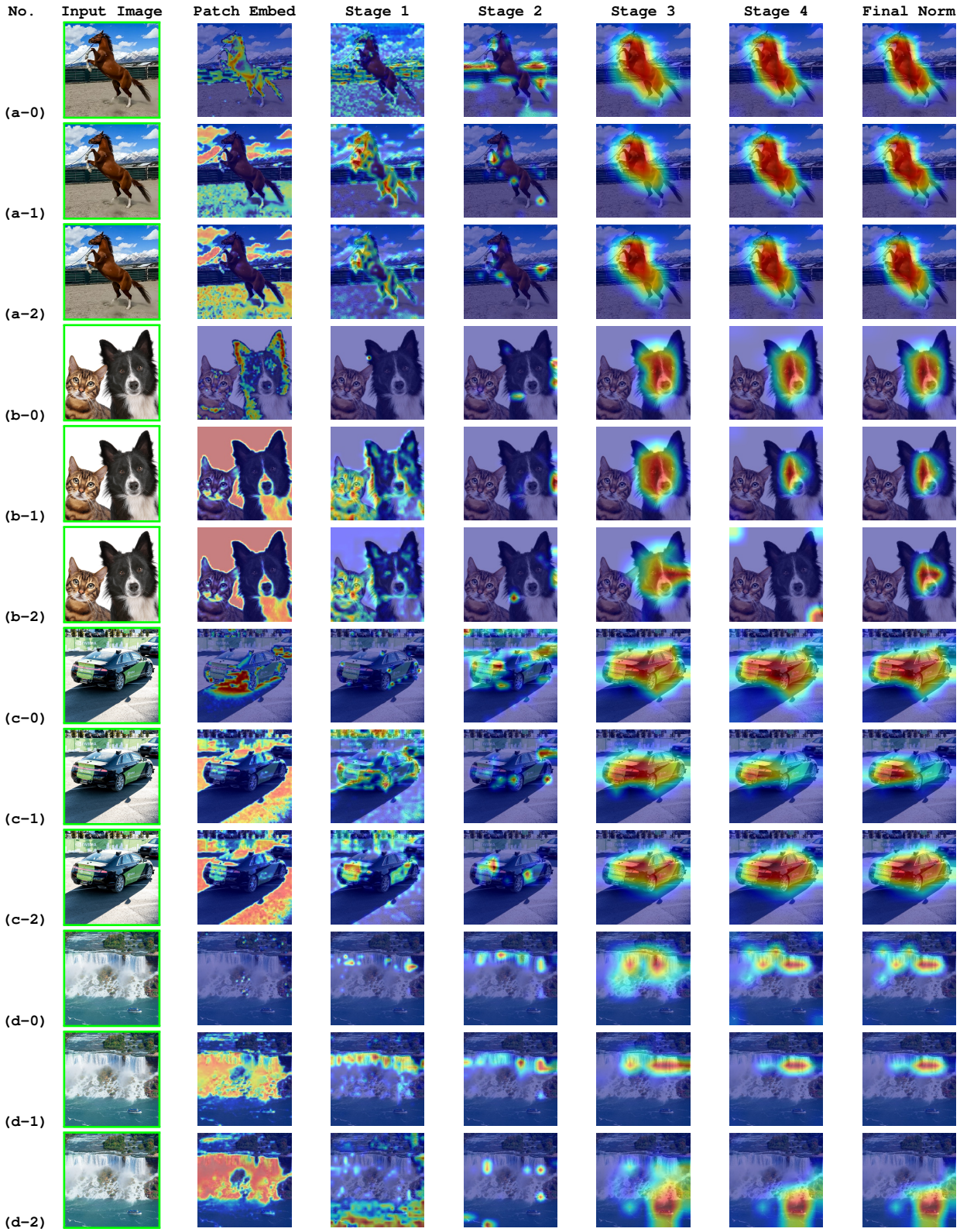

Figure 3. Attention map visualization for Swin Transformer V1 Base. We choose the output of layer norm after patch embedding, each patch merging layer after the last Swin Transformer block in each stage, and the final layer norm of the model, to generate the Class Activation Map (CAM) visualization results. **(a/b/c/d-0)** are CAM results of dense pretrained model on ImageNet-1K dataset, **(a/b/c/d-1)** are CAM results of **GPUSQ-ViT** compressed INT8 models. **(a/b/c/d-2)** are CAM results of **GPUSQ-ViT** compressed INT4 models.

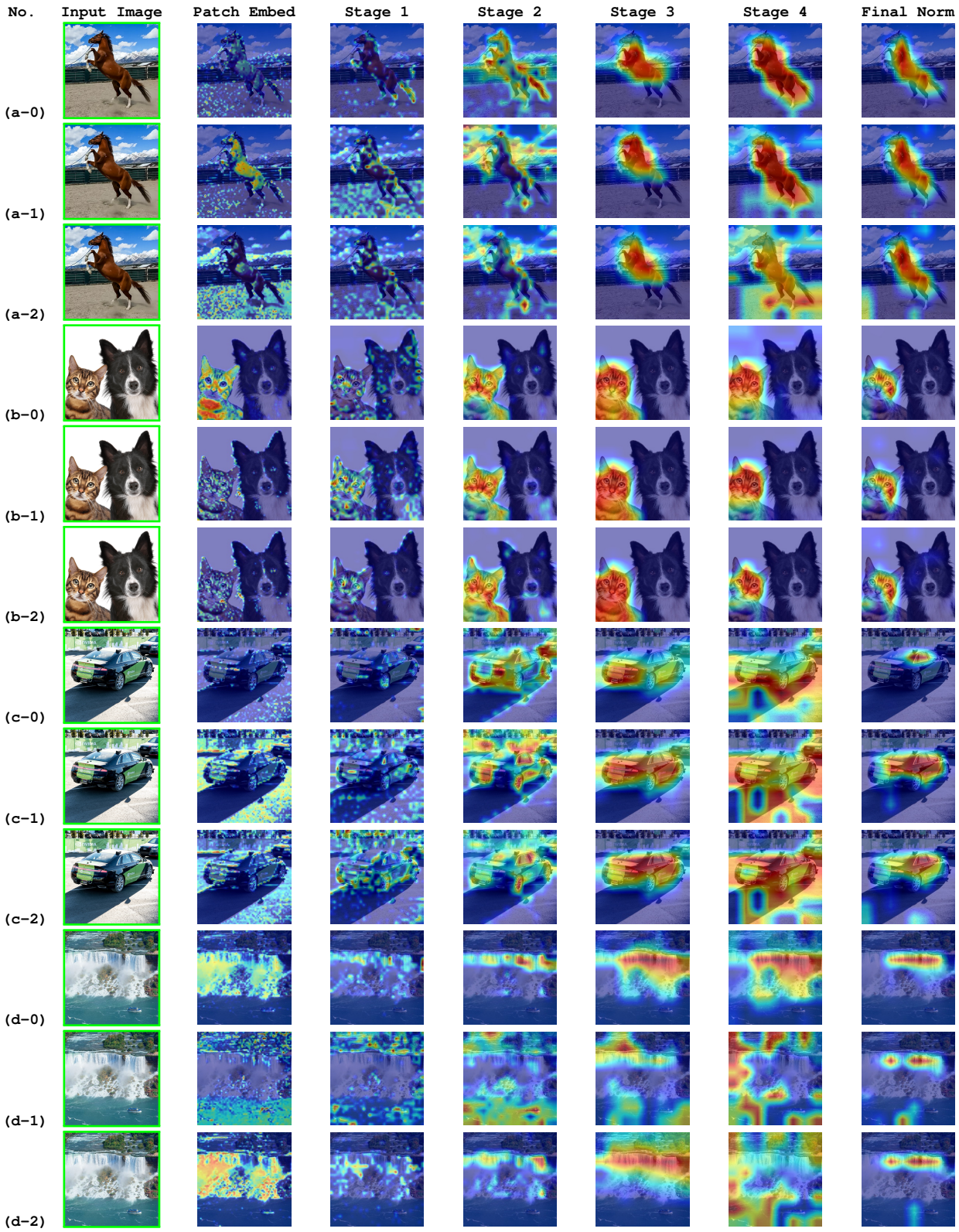

Figure 4. Attention map visualization for Swin Transformer V2 Tiny. We choose the output of layer norm after patch embedding, each patch merging layer after the last Swin Transformer block in each stage, and the final layer norm of the model, to generate the Class Activation Map (CAM) visualization results. **(a/b/c/d-0)** are CAM results of dense pretrained model on ImageNet-1K dataset, **(a/b/c/d-1)** are CAM results of **GPUSQ-ViT** compressed INT8 models. **(a/b/c/d-2)** are CAM results of **GPUSQ-ViT** compressed INT4 models.

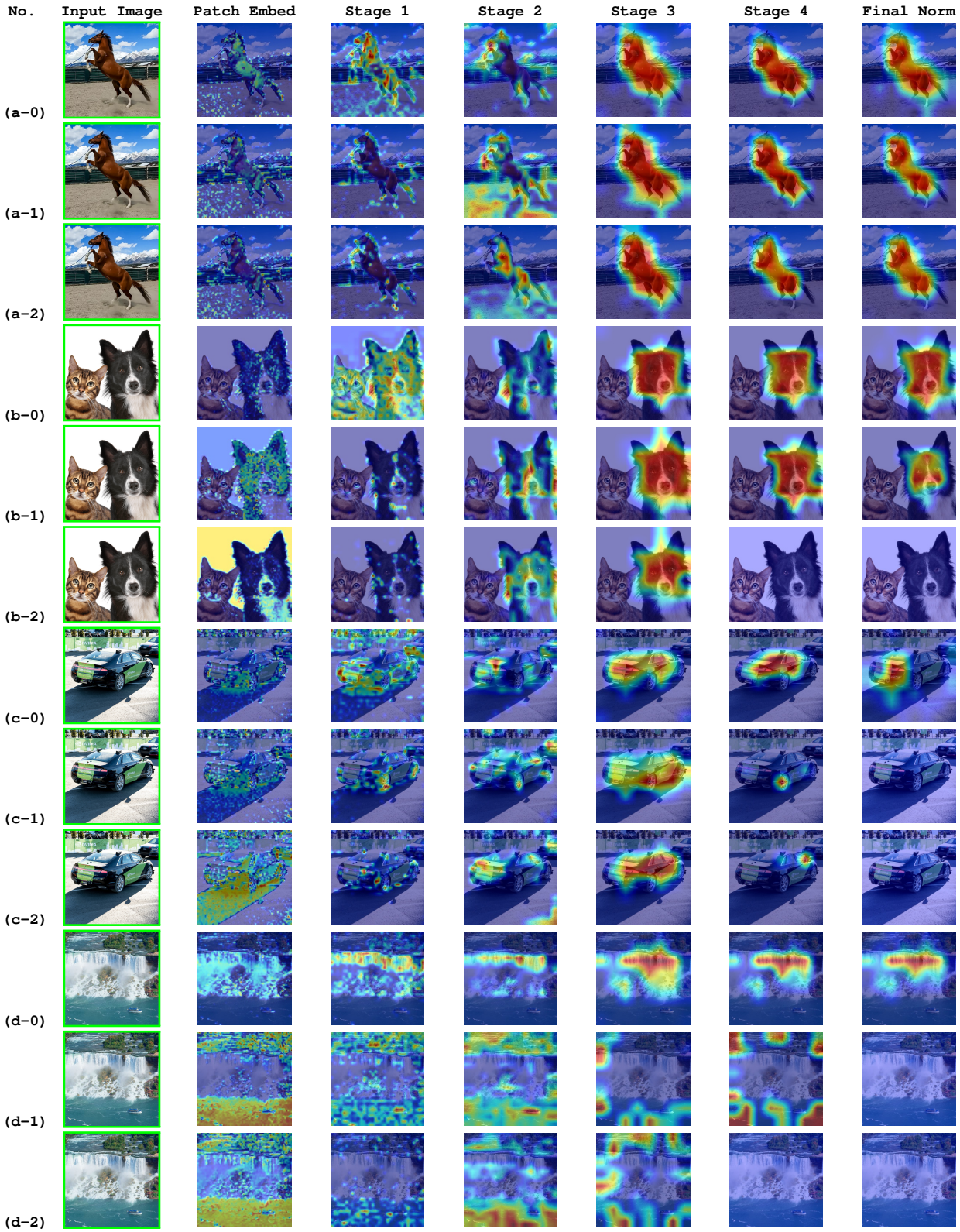

Figure 5. Attention map visualization for Swin Transformer V2 Base. We choose the output of layer norm after patch embedding, each patch merging layer after the last Swin Transformer block in each stage, and the final layer norm of the model, to generate the Class Activation Map (CAM) visualization results. **(a/b/c/d-0)** are CAM results of dense pretrained model on ImageNet-1K dataset, **(a/b/c/d-1)** are CAM results of **GPUSQ-ViT** compressed INT8 models. **(a/b/c/d-2)** are CAM results of **GPUSQ-ViT** compressed INT4 models.

Model	Factor α	Factor β	Factor γ	Enable QAT Weight Factor	GPUSQ-ViT (INT8)		GPUSQ-ViT (INT4)	
					Top-1 Acc(%)	Top-5 Acc(%)	Top-1 Acc(%)	Top-5 Acc(%)
DeiT-Base (224^2)	1	10	5	✓	82.9 (+1.1)	96.4 (+0.8)	81.6 (-0.2)	95.5 (-0.1)
	1	10	5	✗	82.4 (+0.6)	96.1 (+0.5)	80.1 (-1.7)	94.3 (-1.3)
	1	0	5	✓	82.7 (+0.9)	96.2 (+0.6)	81.3 (-0.5)	95.2 (-0.4)
	1	10	0	✓	82.2 (+0.4)	95.8 (+0.2)	80.8 (-1.0)	94.8 (-0.8)
	1	20	5	✓	82.9 (+1.1)	96.4 (+0.8)	81.6 (-0.2)	95.6 (+0.0)
	1	30	5	✓	82.9 (+1.1)	96.5 (+0.9)	81.6 (-0.2)	95.6 (+0.0)
	1	10	10	✓	82.8 (+1.0)	96.5 (+0.9)	81.5 (-0.3)	95.5 (-0.1)
	1	10	2.5	✓	82.8 (+1.0)	96.5 (+0.9)	81.5 (-0.3)	95.6 (+0.0)
Swin-Base (224^2)	1	10	5	✓	83.4 (-0.1)	96.4 (-0.1)	83.2 (-0.3)	96.3 (-0.2)
	1	10	5	✗	82.9 (-0.6)	96.0 (-0.5)	81.5 (-2.0)	94.9 (-1.6)
	1	0	5	✓	83.2 (-0.3)	96.2 (-0.3)	82.9 (-0.6)	96.0 (-0.5)
	1	10	0	✓	82.7 (-0.8)	95.7 (-0.8)	82.4 (-1.1)	95.5 (-1.0)
	1	20	5	✓	83.4 (-0.1)	96.4 (-0.1)	83.2 (-0.3)	96.3 (-0.2)
	1	30	5	✓	83.4 (-0.1)	96.4 (-0.1)	83.2 (-0.3)	96.4 (-0.1)
	1	10	10	✓	83.3 (-0.2)	96.4 (-0.1)	83.1 (-0.4)	96.3 (-0.2)
	1	10	2.5	✓	83.3 (-0.2)	96.4 (-0.1)	83.1 (-0.4)	96.4 (-0.1)

Table 3. Ablation study of the loss adjustment factors and sparse-distillation-aware weight factors of **GPUSQ-ViT** method.

From the CAM visualization results, we can find that the **GPUSQ-ViT** compressed models try to mimic the feature maps of the dense pretrained models in most cases. But the mimicking still fails for some layers of the **GPUSQ-ViT** compressed models. That's the reason why we can see a small accuracy drop between dense pretrained and **GPUSQ-ViT** compressed models, especially for **GPUSQ-ViT** compressed INT4 models.

3.3. Ablation study of GPUSQ-ViT

The ablation study to measure the influence of the different adjustment factors for the hard label, soft logits, and feature-based losses (α, β, γ) and enabling sparse-distillation-aware weight factor on **GPUSQ-ViT** compressed model accuracy is shown in Table 3.

By comparing the ablation results of ***row 1*** and ***row 2*** for each model, we can find enabling sparse-distillation-aware weight factor has an apparent boost for the compressed models' accuracy. Such a boost effect is more influential on INT4 than INT8 model, because disabling this weight factor will see a more significant drop in INT4 compressed model. The potential reason is sparse-distillation-aware weight factor indicates how much influence the quantization error from each critical layer has on the final accuracy. So the distillation process can focus on mimicking the layers with more accuracy influence, which is more effective for limited quantized bits.

Then, by comparing the ablation results of ***row 3*** and ***row 4*** for each model, we can find disabling the feature-based distillation will lead to a more severe influence than disabling the soft logits distillation. It indicates that mimicking feature maps is very helpful for accuracy compensation in **GPUSQ-ViT** compression.

Finally, by comparing the ablation results of ***row 1***, ***row 5*** and ***row 6*** for each model we can find **GPUSQ-ViT** is relatively robust to the soft logits loss adjustment factor. By comparing the ablation results of ***row 1***, ***row 7*** and ***row 8*** for each model we can find **GPUSQ-ViT** is also robust to the feature-based loss adjustment factor, i.e., within the close range of $\beta = 10$ and $\gamma = 5$ the accuracy of compressed models are stable.

References

[1] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 3

[2] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021. 3

[3] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10809–10818, 2022. 3