

---

# JAFAR: Jack up Any Feature at Any Resolution

## Supplementary Material

---

### A Class Activation Maps Evaluation

To evaluate Class Activation Maps (CAMs), we employ a frozen pre-trained ViT-B/16 model as the backbone and extract Grad-CAMs. We randomly sample 2,000 images from the ImageNet validation set for which the model produces correct predictions. For each image, we compute the gradients with respect to the predicted class, average them, and use the result to weight the corresponding activation maps. The weighted activations are then summed to produce the final CAM. These activation maps are upsampled from  $14 \times 14$  to  $224 \times 224$ , resulting in high-resolution CAMs. For each CAM, we generate a masked version of the input image by applying a binary mask that highlights regions positively associated with the model’s prediction. Formally, a masked image is obtained as  $x_{\text{masked}} = x \odot \mathbb{1}_{\text{CAM}_c(x) > 0}$ . These masked images are then used to compute the evaluation metrics.

**Average Drop** Average Drop (A.D) quantifies how much the model’s confidence in the predicted class decreases when it is presented with the masked image instead of the full image. For a single image, the metric is defined as:

$$\mathbf{A.D} = \frac{1}{N} \sum_{i=1}^N \frac{\max(0, Y_i^c - O_i^c)}{Y_i^c} \cdot 100, \quad (4)$$

where  $Y_i^c$  denotes the model’s output score for class  $c$  when using the full image, and  $O_i^c$  denotes the score when using the masked version derived from the explanation map. The final A.D value is computed by averaging over a set of  $N$  images.

**Average Increase** Average Increase (A.I) measures how often the model’s confidence in the predicted class is higher when using the masked image than when using the full image. It is defined as:

$$\mathbf{A.I} = 100 \cdot \sum_{i=1}^N \frac{\mathbb{1}_{Y_i^c < O_i^c}}{N}, \quad (5)$$

where  $Y_i^c$  is the model’s output score for class  $c$  when using the full image, and  $O_i^c$  is the score when using the masked image based on the explanation map. The metric reflects the percentage of images where the explanation-based input yields a higher confidence score than the original image.

**Average Gain [50]** Average Gain (A.G) quantifies the improvement in predictive confidence for the target class when using the masked image instead of the full image. It is defined as:

$$\mathbf{A.G} = \frac{1}{N} \sum_{i=1}^N \frac{\max(0, O_i^c - Y_i^c)}{1 - Y_i^c} \cdot 100, \quad (6)$$

where  $Y_i^c$  is the model’s output score for class  $c$  on the full image, and  $O_i^c$  is the score when using the masked version derived from the explanation map. This metric captures how much the explanation enhances the model’s confidence, normalized by the room for improvement  $(1 - Y_i^c)$ .

**Coherency [44]** A Class Activation Map should highlight all the relevant features that contribute to a model’s prediction, while suppressing irrelevant ones in a coherent and consistent manner. Consequently, for a given input image  $x$  and a target class  $c$ , the CAM should remain unchanged when the image is conditioned on the CAM itself. This self-consistency can be expressed as:

$$\text{CAM}_c(x \odot \text{CAM}_c(x)) = \text{CAM}_c(x) \quad (7)$$

where  $\odot$  denotes element-wise multiplication. This condition implies that the CAM produced from the masked image should be identical to the original CAM, ensuring that the explanation is stable. Following the approach in [44], we use the Pearson Correlation Coefficient between the original CAM and the CAM obtained after masking:

$$\text{Coherency}(x) = \frac{\text{Cov}(\text{CAM}_c(x \odot \text{CAM}_c(x)), \text{CAM}_c(x))}{\sigma_{\text{CAM}_c(x \odot \text{CAM}_c(x))} \sigma_{\text{CAM}_c(x)}} \quad (8)$$

where Cov denotes the covariance and  $\sigma$  the standard deviation of each CAM. Since the Pearson Correlation Coefficient ranges from  $-1$  to  $1$ , we normalize it to the range  $[0, 1]$  and express it as a percentage for interpretability. A coherency score of 100% indicates that the attribution method is fully invariant to input perturbations guided by its own explanations.

**Complexity** In addition to ensuring that a CAM is coherent—preserving predictive features while discarding irrelevant ones—it is also desirable for the CAM to be as simple as possible. That is, it should highlight the minimal subset of pixels necessary to explain the model’s prediction. To quantify this notion of simplicity, we use the  $\ell_0$  norm as a proxy for the Complexity of a CAM:

$$\text{Complexity}(x) = \|\text{CAM}_c(x)\|_0, \quad (9)$$

where  $\|\cdot\|_0$  counts the number of non-zero (i.e., activated) pixels. A lower Complexity score indicates that the attribution method focuses on fewer, more relevant regions, thereby producing more concise and interpretable explanations.

**ADCC [44]** Since each individual metric captures a distinct aspect of CAM quality, we compute an aggregated evaluation metric—Average DCC (ADCC)—which combines Coherency, Complexity, and Average Drop into a single score using the harmonic mean:

$$\text{ADCC}(x) = 3 \left( \frac{1}{\text{Coherency}(x)} + \frac{1}{1 - \text{Complexity}(x)} + \frac{1}{1 - \text{A.D}(x)} \right)^{-1} \quad (10)$$

ADCC offers a unified, single-valued measure that enables direct and consistent comparison. By balancing coherency, sparsity (via low complexity), and confidence preservation (via low Average Drop), it provides a more comprehensive assessment of attribution quality.

## B Additional Details on Baselines

- **Large Image:** For the Large Image baseline, we upsampled the original image via bilinear upsampling and use it as input to the foundation vision encoder. During evaluation on downstream tasks (see Tabs. 1 and 2), we upsample the input to the maximum ratio that fits in memory (i.e.,  $\times 8$ ), and subsequently apply bilinear upsampling to the resulting feature map to match the target output resolution. Due to the high computational cost and training time, we omit results on the COCO dataset. For efficiency, on Open-Vocabulary segmentation we limit upsampling to a  $\times 2$  ratio in Tab. 3.
- **Strided:** To obtain higher-resolution feature maps, we modify the stride of the ViT backbone to produce more patches. While the stride typically equals the patch size (e.g., 14 in DINOv2), we reduce the former to 6 in our experiments corresponding to a  $\times 2.3$  upsampling. We then apply bilinear upsampling to the resulting feature map to reach the desired output resolution.
- **CARAFE [22]:** For CARAFE, we use the CARAFEPack module from MMCV [51], stacking four upsampling stages with a  $\times 2$  ratio each, resulting in a final feature map upsampled by a factor of  $\times 16$ .
- **SAPA [23]:** We adopt the default implementation from the official [SAPA repository](#), stacking four SAPA upsampling modules, each with an upsampling factor of 2. This results in a final feature map with a total upsampling factor of  $\times 16$ .
- **DySample [24]:** We adopt the default implementation from the official [Dysample repository](#), stacking four Dysample upsampling modules, each with an upsampling factor of 2. This results in a final feature map with a total upsampling factor of  $\times 16$ .



- **ReSFU** [25]: We use the default implementation from the official **ReSFU repository**, performing a direct upsampling to the target output resolution.
- **FeatUp** [28]: For FeatUp, we use the scalable JBU variant from the official **FeatUp repository** stacking 4 upsampling modules to achieve a total upsampling factor of  $\times 16$ . We re-train FeatUp using the provided training scripts. While the original paper trains FeatUp on the COCO dataset for 2,000 steps with a batch size of 4 and evaluates on the same dataset, we train it on the ImageNet training set for 50,000 steps ( $25\times$  more) using the same batch size, ensuring a fair comparison across methods.
- **LiFT** [27]: For LiFT, we slightly adapt the official **implementation** by resizing the intermediate representations after the downsampling module to ensure compatibility with backbones using a patch size of 14 (i.e., a downsampling factor of 14). In the official code, LiFT performs a  $\times 2$  upsampling and then relies on a bilinear interpolation to upsample the features to the target output resolution.

We summarize in Tab. 6 the differences between upsamplers.

Table 6: Comparison of feature upsampling methods.

Method	Task-Agnostic	Direct Upsampling	Lightweight Inference
CARAFE - SAPA - DySample	✗	✗	✓
ReSFU	✗	✓	✓
LiFT	✓	✗	✓
FeatUp (JBU)	✓	✗	✓
FeatUp (Implicit)	✓	✓	✗
JAFAR	✓	✓	✓

## C Additional Visualizations

We provide in the following subsections additional comparison visualizations for upsampled feature maps Supp. C.1, class activation maps predictions Supp. C.2, depth estimation Supp. C.3 and semantic segmentation Supp. C.4.

### C.1 Feature Visualization

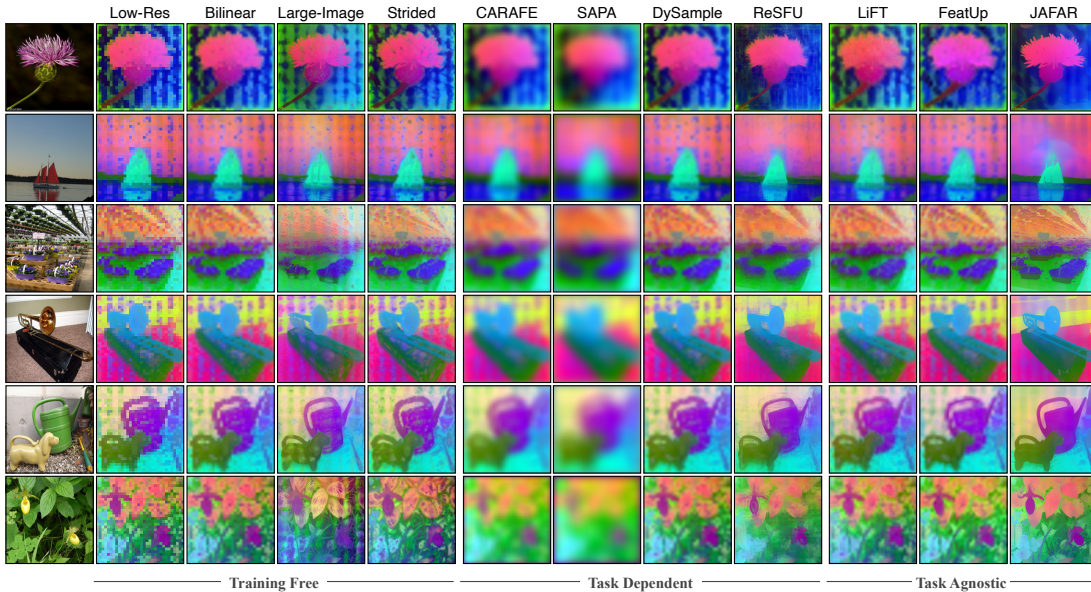


Figure 5: **PCA Feature Visualization.** DINOv2 ViT-S/14 features at  $32 \times 32$  resolution from the ImageNet validation set are upsampled to  $448 \times 448$ .

Fig. 5 shows additional PCA visualizations of upsampled feature maps. Starting from  $32 \times 32$  feature maps extracted using a DINOv2-S/14 backbone, each is upsampled to  $448 \times 448$  using different baseline methods. These baselines—whether training-free, task-dependent, or task-agnostic—tend to introduce varying degrees of blurriness and visual artifacts. In contrast, JAFAR, while remaining task-agnostic, produces sharp, content-aware features with minimal artifacts.

## C.2 Class Activation Maps

We present additional Grad-CAM visualizations based on ViT-B/16 features from the ImageNet validation set in Fig. 6. Except for the “Low-Res” column, where features remain at their original  $14 \times 14$  resolution, all feature maps are upsampled to  $224 \times 224$  before Grad-CAM extraction. The explainability maps generated by our upsampling approach are noticeably sharper and more accurate, exhibiting fewer artifacts compared to those from alternative methods. Notably, CARAFE and LiFT fail to produce meaningful explanations in this setting, suggesting that the training of these methods does not transfer effectively to these ViT-based features.

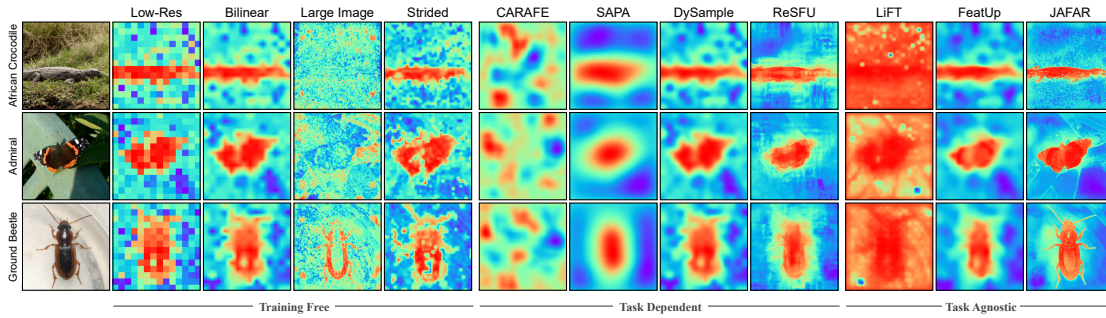


Figure 6: Class Activation Maps comparison.

## C.3 Depth Estimation

Fig. 7 presents additional examples of linear probe transfer learning for depth estimation on the COCO-Stuff dataset. Feature maps of size  $32 \times 32$ , extracted from a DINOv2-S/14 backbone, are upsampled to  $448 \times 448$  using the various baseline methods. A linear probe is then trained on these features to predict depth, using supervision from a Depth-AnythingV2 model. The results demonstrate that both FeatUp variants produce high-quality features well-suited for transfer learning in depth estimation tasks.

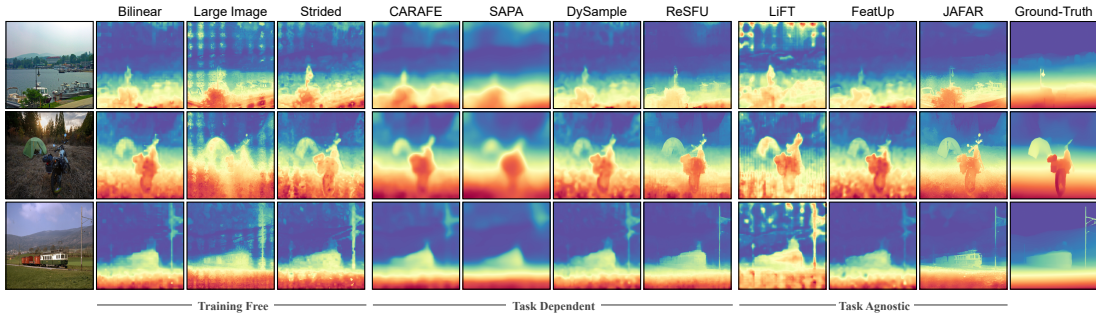


Figure 7: Depth Estimation Visualization.

## C.4 Semantic Segmentation

Fig. 8 presents examples of linear probe transfer learning for semantic segmentation on the COCO-Stuff dataset. Feature maps of size  $32 \times 32$ , extracted from a DINOv2-S/14 backbone, are upsampled to  $448 \times 448$  using the various baseline methods. JAFAR produces more coherent segmentation results, offering improved delineation of both object boundaries and background regions.

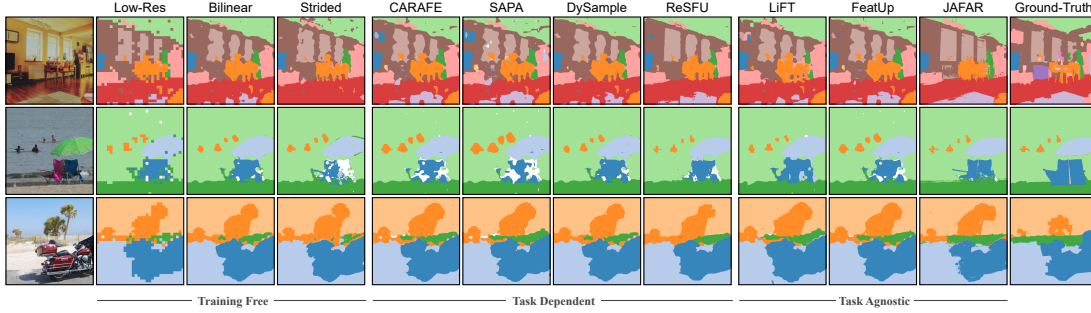


Figure 8: Semantic Segmentation Visualization.

## D Additional Comparison With Task-Agnostic Baselines

### D.1 FeatUp & LiFT

In Tab. 1, we reported results obtained by training FeatUp and LiFT within our own codebase. To complement this evaluation, we present an additional comparison in Tabs. 7 and 8 using the official released checkpoints—FeatUp on DINOv2 ViT-S/14 and LiFT on DINO ViT-S/16, respectively.

Table 7: **Semantic Segmentation Comparison between JAFAR and FeatUp.** JAFAR is evaluated using FeatUp’s original feature extractor and compared against the official JBU checkpoint on the DINOv2 ViT-S/14 backbone.

Model	224			448		
	ADE20K	Cityscapes	VOC	ADE20K	Cityscapes	VOC
Bilinear	24.50	38.18	64.10	28.19	49.35	70.64
FeatUp	29.26	42.84	71.90	<b>32.87</b>	53.17	77.57
JAFAR	<b>30.04</b>	<b>48.52</b>	<b>75.36</b>	<u>32.29</u>	<b>56.45</b>	<b>79.06</b>

Table 8: **Semantic Segmentation Comparison between JAFAR and LiFT.** JAFAR is evaluated using LiFT’s original feature extractor and compared against the checkpoint on the DINO ViT-S/16 backbone.

Model	224			448		
	ADE20K	Cityscapes	VOC	ADE20K	Cityscapes	VOC
Bilinear	15.89	32.52	32.15	16.44	36.56	33.13
LiFT	<u>16.97</u>	<u>36.31</u>	<u>35.30</u>	<u>16.98</u>	<u>40.01</u>	<u>35.07</u>
LiFT-iterative	15.62	37.68	31.25	14.96	39.97	29.43
JAFAR	<b>21.08</b>	<b>39.93</b>	<b>47.36</b>	<b>21.44</b>	<b>43.93</b>	<b>48.18</b>

Across all datasets and output resolutions—with the exception of ADE20K at 448 for FeatUp—JAFAR consistently delivers significant improvements in semantic segmentation performance. We also evaluate a LiFT-iterative baseline, which stacks two LiFT  $\times 2$  upsamplers as described in the original LiFT paper. However, this iterative approach does not outperform the simpler method of applying a single LiFT  $\times 2$  upsampler followed by bilinear interpolation.

### D.2 LoftUp

We present an additional comparison with LoftUp [53], a recent baseline which relies on segmentation masks from SAM [54] during training and employs a two-stage pipeline that includes a self-distillation phase. For a fair evaluation, we tested JAFAR within LoftUp’s official codebase using a DINOv2 ViT-S/14 backbone.

Table 9: **Semantic Segmentation Comparison between JAFAR and LiFT.** JAFAR is evaluated using Loftup’s original feature extractor and compared against the check-point on the DINOv2 ViT-S/14 backbone.

Resolution		Cityscapes		COCO	
		mIoU ( $\uparrow$ )	Acc ( $\uparrow$ )	mIoU ( $\uparrow$ )	Acc ( $\uparrow$ )
56	LoFTUp	15.30	75.50	25.33	54.06
	JAFAR	<b>19.09</b>	<b>79.34</b>	<b>28.79</b>	<b>57.86</b>
	JAFAR + distillation	18.35	79.05	28.60	57.49
112	LoFTUp	32.02	86.28	48.89	73.09
	JAFAR	<b>34.56</b>	87.19	50.67	74.56
	JAFAR + distillation	33.63	<b>87.47</b>	<b>50.76</b>	<b>74.60</b>
224	LoFTUp	50.83	<b>91.55</b>	59.79	<b>80.04</b>
	JAFAR	51.45	91.25	59.76	79.93
	JAFAR + distillation	<b>51.84</b>	91.53	<b>59.90</b>	<b>80.04</b>
448	LoFTUp	<b>62.49</b>	93.69	62.25	81.43
	JAFAR	61.49	93.46	62.02	81.30
	JAFAR + distillation	62.30	<b>93.76</b>	<b>62.36</b>	<b>81.45</b>

As shown in Tab. 9, JAFAR outperforms LoftUp at lower upsampling resolutions (56 and 112) and delivers comparable performance at higher resolutions (224 and 448). Different from LoftUp, JAFAR uses a simpler and more efficient single-stage strategy: it operates entirely at low resolution and does not rely on external annotations. Nevertheless, the self-distillation mechanism introduced in LoftUp is complementary to our approach and can be seamlessly integrated into JAFAR’s pipeline. To demonstrate this, we implemented a similar distillation objective and report the results as JAFAR + distillation in Tab. 9. While the gains are minimal at lower resolutions, this enhancement provides a clear boost at higher resolutions (224 and 448). Lastly, JAFAR is considerably more lightweight, with just 0.7 million parameters compared to 4.3 million in LoftUp.

## E Inference Time

In Tab. 10, we compare the inference times of various methods using a batch size of 1 and input images of resolution 448, across multiple output resolutions. The experiments are conducted on a single A100 GPU.

Table 10: **Parameter Count and Inference Time Comparison (ms).**

Model	# Params (M)	56 <sup>2</sup>	112 <sup>2</sup>	224 <sup>2</sup>	448 <sup>2</sup>
<i>Upsamplers</i>					
FeatUp	0.2	5.7	8.0	14.9	64.9
JAFAR	0.7	4.0	5.7	16.6	94.0
LiFT	1.2	0.9	0.9	1.0	1.5
LoftUp	4.3	3.8	8.9	24.5	145.1
<i>Other configurations</i>					
Large Image (x8)	-	6.2	34.7	348	5 558
Strided (1)	-	11.4	136.0	2 482	48 090

Additionally, Tab. 11 reports the memory usage of each model during both forward and backward passes, evaluated across multiple output resolutions using a fixed batch size of 1 and input resolution of 448.

Table 11: Memory usage (in GB) for different models and resolutions.

<b>Pass</b>	<b>Model</b>	<b># Params (M)</b>	$56^2$	$112^2$	$224^2$	$448^2$
Forward	Bilinear	0.0	0.4	0.5	0.5	0.8
	FeatUP	0.2	0.6	0.8	1.6	4.8
	JAFAR	0.7	0.6	0.6	1.1	7.7
	LoftUp	4.3	0.6	0.7	1.8	12.3
Backward	FeatUP	0.2	0.7	0.9	2.1	7.4
	JAFAR	0.7	0.7	1.1	3.5	26.0
	LoftUp	4.3	0.7	1.3	4.6	26.5