AM-RADIO: Agglomerative Vision Foundation Model Reduce All Domains Into One

Supplementary Material

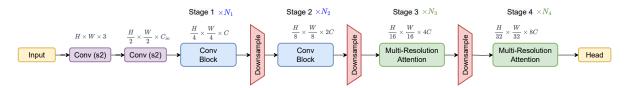


Figure 5. High level architecture of the ERADIO network architecture. Overall architecture is composed of multiple stages: 1) the stem, 2) 2 convolutional blocks from YOLOv8, 3) 2 transformer blocks with multi-resolution windowed self attention.

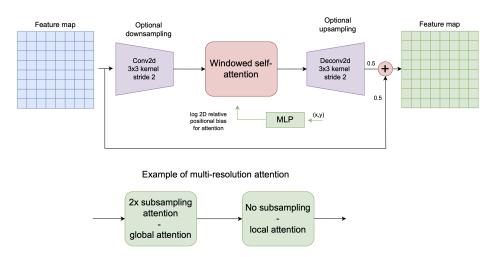


Figure 6. Multi-resolution attention for E-RADIO

A. E-RADIO architecture details

The architecture of E-RADIO is illustrated in Figure 5. It is a hybrid CNN-Transformer architecture. First 2 stages follow convolution paradigm and have the C2f architecture from YOLOv8 model [29]. The last 2 stages have the Transformer architecture with windowed attention and multi-resolution attention (MRA) structure. Every stage, except the last one, are followed by downsample block. We implement it as a strided convolution with 3x3 kernel and stride 2, followed by batch normalization layer.

A.1. Multi-Resolution Attention

Standard transformers struggle to scale with high input image resolution because of quadratic complexity of the attention. SWIN [39] proposed to use windowed attention to reduce the complexity of attention. We reuse windowed attention in the E-RADIO. To address for missing communication between windows, SWIN introduced window shifting, unfortunately, it has non-negligible compute cost. Instead, we propose multi-resolution attention inspired by EdgeViT's Local-Global-Local attention [45]. The idea is illustrated in Figure 6. Every layer in the transformer will have a local windowed attention with optional subsampling via convolutional operator. For example, if susbampling is dissabled, then it is just a standard windowed attention. If the subsampling ratio is 2, then the feature map is downsampled by a factor of 2, windowed attention is performed, and then the feature map is upsampled to the original resolution with deconvolution. For FasterVIT2 models, we interleave subsampled attention with ratio 2 and the normal attention with no subsampling.

A.2. Configurations

All models in the family follow the same configuration except the embedding dimension (hide dimension). We simply scale it up with bigger models. Other parameters:

- Input resolution is 224
- In-stem contains 2 3x3 convolutions with stride 2
- Total stages: 2 convolutional and 2 transformer
- First stage takes input feature size of 56x56, has 3 layers with C2f structure from YOLO8 [29].
- Second stage takes input feature size of 28x28, has 3 layers of C2f.

- Third stage takes features of size 14x14, has 5x multi-resolution attention, window size 7.
- Forth stage takes features of size 7x7, has 5x windowed attention of window size 7.
- Embedding dimension for different model variants: XT 64, T 80, S 96, B 128, L 192. The smallest XT and T models have [1, 3, 4, 5] layers for each of 4 stages.
- Output features have resolution of 14x14 and are obtained by upsampling the features of stage 4 by 2x with deconvolution and adding to stage 3 features of size 14x14.

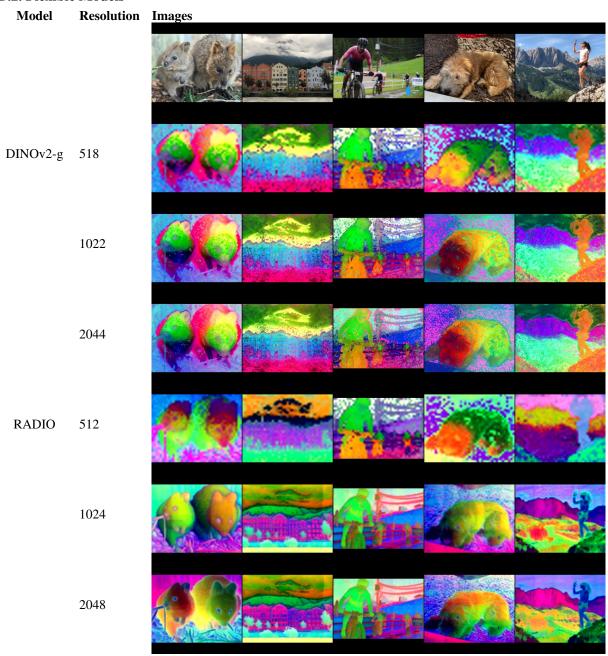
B. PCA Visualizations

We visualize various models using PCA to reduce the model's spatial feature dimensionality down to 3 dimensions, and directly map those to RGB. Most models are only able to handle square inputs at fixed resolutions, however DINOv2 and RADIO can handle arbitrary resolutions and aspect ratios, so we visualize them in both settings.

B.1. Square Models

B.1. Square Mod	lels	
Model	Resolution	Images
OpenCLIP-H/14	224	
MetaCLIP-H/14	224	
SigLIP-M/14	384	
InternViT-6B	224	
	448	
DFN CLIP	378	
OpenAI CLIP	336	
DINOv2-g	518	
SAM-H	1024	
RADIO	512	
	1024	

B.2. Flexible Models



C. ViTDet Augmentation

The following python code shows how the alternating window/global architecture of ViTDet [34] can be applied to a transformer. We take advantage of the fact that transformers are permutation invariant *after position encodings have been applied*, and thus it's easy to organize the patch order such that contiguous chunks of patches belong to the same window. Once reordered in this way, alternating between windowed and global attention is achieved simply by absorbing the windows into the batch dimension or returning to the original shape respectively. We also enforce that the final transformer layer always applies global attention.

```
from einops import rearrange
def reorder_patches(patches: torch.Tensor,
```

```
patched_size: Tuple[int, int],
                    window_size: int):
    p_i dxs = torch.arange(patches.shape[1])
    p_i dxs = rearrange(p_i dxs, '(wy y wx x) \rightarrow (wy wx y x)',
                       wy=patched_size[0] // window_size, y=window_size,
                       wx=patched_size[1] // window_size, x=window_size)
    p_{idxs} = p_{idxs}.reshape(1, -1, 1).expand_as(patches)
    return torch.gather(patches, p_idxs), p_idxs
def vitdet_aug(blocks: nn. Sequential,
               patches: torch. Tensor,
               patched_size: Tuple[int, int],
               window_sizes: List[int],
               num_windowed: int):
    B, T, C = patches.shape
    window_size = sample(window_sizes)
    sq_window_size = window_size ** 2
    patches , p_idxs = reorder_patches(patches, patched_size, window_size)
    period = num\_windowed + 1
    for i, block in enumerate(blocks[:-1]):
        if i % period == 0:
            patches = patches.reshape(B * sq_window_size, -1, C)
        elif i % period == num_windowed:
            patches = patches.reshape(B, T, C)
        patches = block(patches)
    # Always use global attention with the last block
    patches = patches.reshape(B, T, C)
    patches = blocks[-1](patches)
    # Finally, put the patches back in input order
    ret = torch.empty_like(patches)
    ret = ret.scatter(dim=1, index=p_idxs, src=patches)
    return ret
```

D. Comparison with SAM-CLIP [56]

Concurrently with our work, SAM-CLIP was introduced as a method of fusing SAM and CLIP into a single model. Due to the concurrency of effort, we don't compare our model with the full suite of metrics demonstrated in their method, however, we do have some overlap in key metrics such as Zero-Shot ImageNet-1k, and ADE20k semantic segmentation via linear probing. We present the comparison in table 9, however we note that there are enough differences between these two models that we can't conclude one way or another what is the superior approach. Instead we'll argue that DINOv2 does a better job of ADE20k linear probing than SAM, and thus our significantly higher quality on this metric is likely due to the inclusion of DINOv2, which is a key introduction with our approach.

E. Automatic Loss Balancing

E.1. Uncertainty

Following [11], we have:

$$L(x) = \sum_{k} \frac{1}{2\sigma_k^2} L_k(x) + \log \sigma_k \tag{4}$$

Family	Model	Zero-Shot	ADE20k
SAM	ViTDet-H/16		28.2
DFN CLIP	ViT-H/14	83.9	31.7
SAM-CLIP	ViTDet-B/16	71.7	38.4
RADIO	ViT-H/14	82.7	51.3

Table 9. We compare our common key metrics with those demonstrated in SAM-CLIP [56]. We note that there are numerous differences between the two approaches, including model capacity and architecture. SAM-CLIP uses the ViT-B variant of SAM as a starting point, which implies it's a ViTDet-B/16 architecture. As a result of this choice, their metrics are computed at a resolution of 1024. RADIO trains a vanilla ViT-H/14 from scratch, and as a result of the flexibility gained via the CPE method, we evaluate Zero-Shot ImageNet1k at a resolution of 432, and we run ADE20k linear probing at a resolution of 512 using the exact same weights. We note that Zero-Shot quality is largely determined by the quality of the CLIP teacher and the capacity of the student. We attribute our superior quality on ADE20k semantic segmentation largely to our inclusion of DINOv2 as a teacher.

where the σ_k values are predicted by the student. In practice, the student predicts $b := \log \sigma_k^2$ for numerical stability, to avoid division by zero, and to regress unconstrained scalar values.

We make some minor modifications to (4) to make training a bit more stable in our setting. We replace the manual λ scalars with the learned uncertainty weights, and add the loss term for large uncertainties. Altogether, this yields:

$$\lambda_k = \frac{e^{-b_k}}{2}$$

$$L(x) = \sum_k \lambda_k L_k(x) + \frac{b_k}{2}$$
(5)

Let $b_i^{(s|v)}(x'|\Theta_i^{(s)})$ be a learned function predicting balance parameters for teacher i and summary weight (s) or feature vector weight (v), we transform equation (5) slighty to:

$$\psi(x) = \log(1 + e^{x})
\lambda_{i}^{(m)} = e^{-b_{i}^{(m)}(x')}
L(x) = \sum_{i} \sum_{m \in \{s,v\}} \lambda_{i}^{(m)} L_{i}^{(m)}(x) + \psi\left(b_{i}^{(m)}(x')\right)$$
(6)

The function $\psi(x)$ is the familiar "softplus" nonlinear activation function. We drop the division by 2 on the left because, assuming outputs are initially $b \sim \mathcal{N}(0, \sigma^2)$, then the loss weights will initially have an expected value of 1, matching the naive weighting. On the right, we replace $\frac{b_k}{2}$ with $\psi(x)$ for a few reasons:

- When $x \gtrsim 4$, then $\psi(x) \approx x$, yielding the same expression as before.
- When $x \approx 0$, then $\psi'(x) \approx \frac{1}{2}$, yielding the same expression as before.
- When x < 0, which translates to a loss weight > 1, $\psi'(x) \to 0$, improving stability as the weight gets larger.
- It has range $(0, \infty)$ which aesthetically enforces the loss to be greater than zero.

E.2. AdaLoss

In addition to uncertainty auto-balancing, we also explored AdaLoss [25]. In this formulation, we have:

$$\lambda_i^{(m)} = \frac{1}{\mathbb{E}(L_i^{(m)})}$$

$$L(x) = \sum_i \sum_{m \in \{s, v\}} \lambda_i^{(m)} L_i^{(m)}(x)$$
(7)

F. Visual Question Answering Samples

Figures 8 to 12 show sample questions from our Visual Question Answering datasets, together with sample answers when using our vision encoders in a LLaVA setup.

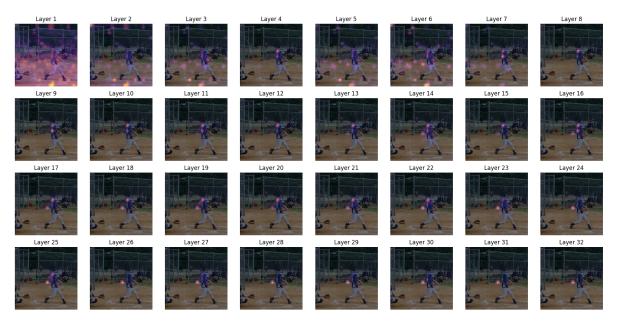


Figure 7. Visualization of the LLaVA attention maps over the visual features produced by a RADIO encoder. We use one sample image from the GQA[27] validation set and one associated question: "What color is the helmet in the middle of the image?". For each layer in the language model, we retrieve attention scores for all positions of the visual tokens, average them over all attention heads, and overlay corresponding heat maps with the input image. We can see that as we progress through the layers, the model's attention focuses on the relevant part of the image. The model's answer is "Blue".



Figure 8. Sample questions from the GQA[27] and their answers from our LLaVA models, using various image encoders. Answers are painted green when they match the ground truth, pink otherwise.



Figure 9. Sample questions from the GQA[27] and their answers from our LLaVA models, using various image encoders. Answers are painted green when they match the ground truth, pink otherwise.



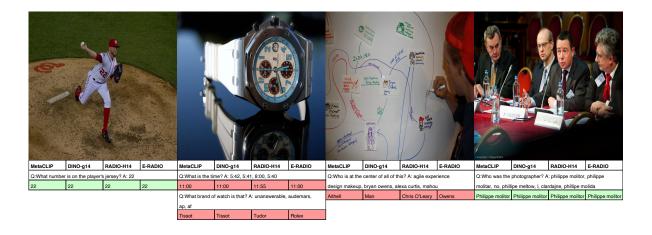


Figure 10. Sample questions from the TextVQA [51] dataset and their answers from our LLaVA models, using various image encoders. Answers are painted green when they match the ground truth, pink otherwise.

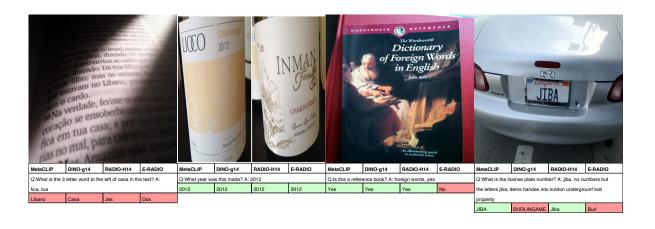




Figure 11. Sample questions from the TextVQA [51] dataset and their answers from our LLaVA models, using various image encoders. Answers are painted green when they match the ground truth, pink otherwise.

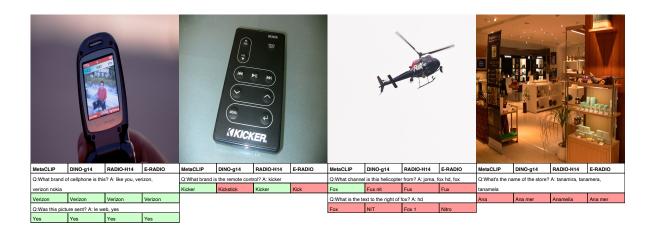




Figure 12. Sample questions from the TextVQA [51] dataset and their answers from our LLaVA models, using various image encoders. Answers are painted green when they match the ground truth, pink otherwise.