DIFFERENTIABLE RESIZING: RESOLUTION LAYERS

Clayton A. Harper, Eli J. Laird, Mitchell A. Thornton, Eric C. Larson

Darwin Deason Institute for Cybersecurity, Southern Methodist University, Dallas, TX

ABSTRACT

Tuning feature map resolution in convolutional neural networks is critical for characterizing performance tradeoffs with memory efficiency and computational complexity. However, most methods rely on manually tuned decimation and upsampling rates to control feature map resolution. Recent advancements have explored differentiable approaches to dynamically learn feature map resolutions, but are constrained to a preset range of rates or are exclusively applicable to either decimation or upsampling, but not both. Addressing these limitations, we propose Resolution Layers that dynamically learn both decimation and upsampling rates during model training. We evaluate Resolution Layers in object detection, achieving a 5.4% increase in mean average precision while using a model that is 19.5% smaller than traditional baseline methods. Additionally, we evaluate the approach in multiple classification tasks, showing it maintains robust performance even when feature map resolution is reduced by 90%.

Index Terms— learnable resolution, architecture optimization, frequency transforms

1. INTRODUCTION

Many components in neural networks are non-differentiable, turning critical design choices into hyperparameters rather than learnable quantities. Models are often trained with handcrafted pyramids, meaning layerwise decimation or upsampling is hard-coded into the network's design. These choices directly impact accuracy, memory footprint, and latency, yet they are not optimized via gradient descent [1]. Instead, practitioners sweep across settings and release multiple model variants (*e.g.*, YOLOv8 nano/small/large) to balance performance and efficiency [2].

The challenge in learning these parameters is that many architectural components are discrete. Gradients are backpropagated through continuous variables, but key dimensionality-changing operators—such as pooling, strided convolutions, and resizing—are controlled by integers like stride, kernel size, or target resizing lengths. With standard formulations, gradients cannot flow to these quantities: networks adapt weights, but the architecture itself remains fixed. In this work, we use the term *learning resolution* to mean treating the spatial or temporal size of feature maps as a trainable parameter, rather than a predefined design choice.

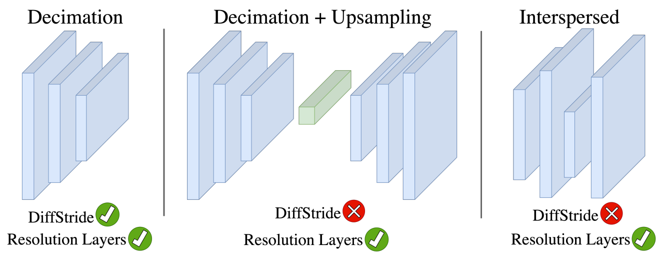

Fig. 1. Capability differences between DiffStride [3] and Resolution Layers (ours).

Several approaches have sought to overcome this barrier by introducing continuous relaxations [4–11]. These methods make resolution partially learnable, but in constrained ways: some restrict changes to resolution reduction or upsampling alone (e.g., DiffStride and FouriScale respectively [3, 12]), while others learn among fixed multiplicative factors (e.g., Shape Adapters and SSHF [13, 14]). While valuable, such approaches cannot flexibly capture cases where resolution must increase as well as decrease, or where allocation should vary freely across layers and branches (see Figure 1). This leaves important gaps in architectures such as autoencoders and detectors, which often rely on both decimation and upsampling blocks [2], and in tasks where optimal compute–accuracy trade-offs depend on fine-grained resolution control.

We introduce *Resolution Layers*, a lightweight module that formulates a layer's output length, height, and width as *learnable* continuous variables. This allows networks to decide when and where to allocate resolution, balancing accuracy and computation end-to-end. For instance, a model may upsample early to capture fine detail, decimate to save compute, or pursue different strategies across branches of the same architecture. Crucially, these decisions are learned directly via gradient descent, rather than being hand-crafted or heuristically chosen.

2. RELATED WORK

Several approaches have attempted to make resolution a learnable component of network design. Shape Adaptors [13] interpolate between pooled and unpooled branches but are limited

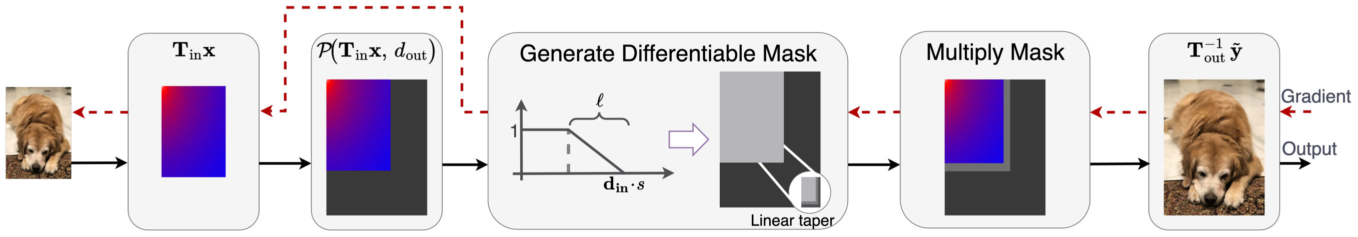
$\mathbf{T}_{\text{in}}\mathbf{x}$
$\mathcal{P}(\mathbf{T}_{\text{in}}\mathbf{x},\, d_{\text{out}})$
Generate Differentiable Mask
1
ℓ
$\mathbf{d}_{\text{in}}\cdot s$
Linear taper
Multiply Mask
$\mathbf{T}_{\text{out}}^{-1}\tilde{\mathbf{y}}$
Gradient
Output

Fig. 2. Resolution layer overview following notation in Algorithm 1.

to a set of pre-defined scaling factors. DynOPool extends this by supporting both up- and downsampling through dynamically optimized pooling, though at the cost of added complexity [11]. Scale-Space HyperNetworks introduce a learnable scaling factor for efficient high-resolution imaging, but require training an auxiliary network to propose candidate scales [14]. In diffusion models, FouriScale replaces CNN layers with a frequency-domain dilation scheme improving upsampling consistency [12].

The work most directly aligned with ours is DiffStride [3], which formulates downsampling as a differentiable cropping operation in the Fourier domain. DiffStride is compelling for its elegance: resizing is expressed as a transform–inverse pair, making the method both simple and efficient. However, its formulation is restricted to resolution reduction. Our work extends this principle to encompass both decimation and upsampling in a *unified* framework, thereby broadening DiffStride's applicability to tasks that demand flexible resolution control.

3. RESOLUTION LAYERS

Resolution Layers (see Figure 2) extend the transform-inverse formulation of DiffStride. As illustrated in Figure 1, this formulation can represent (i) pure decimation (left), (ii) architectures with both up- and downsampling branches (middle), and (iii) interspersed stacks of up- and downsampling layers (right). Each Resolution Layer exposes a single learnable scale $s > 0$ per resizable dimension (e.g., one parameter in 1D, two in 2D), making resolution a trainable quantity.

We highlight the key departures from DiffStride: (i) a general transform-inverse method that is not tied to the FFT (with the FFT as a natural but non-exclusive choice), (ii) a unified padding/slicing operator that handles both truncation and zero-padding, enabling upsampling as well as decimation, and (iii) small adjustments that improve interpretability. We present the 1D case for clarity; the 2D case follows by applying the same steps separably along rows and columns.

Notation. We use *underbraces* to indicate resulting dimensionality of terms and operations. Let $d_{\text{in}} \in \mathbb{N}$ be the input length, $q := d_{\text{in}}s$ the continuous target length, $d_{\text{out}} \in \mathbb{N}$ the output length, and $\mathbf{k} = [1, \ldots, d_{\text{out}}]^\top$ the index vector. We write $\odot$ for the Hadamard (element-wise) product.

Pseudocode. We first present the algorithm as pseudocode, then define each step mathematically.

Algorithm 1 Resolution Layer Forward Pass (1D)

```
Require: $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$, learnable $s > 0$, taper length $\ell$
Require: Generator $\mathcal{T}(n)$ returning $\mathbf{T}_n \in \mathbb{R}^{n\times n}$
1: $q \leftarrow d_{\text{in}} \cdot s$          ▷ continuous target length (differentiable)
2: $d_{\text{out}} \leftarrow \lceil q \rceil$ .detach()          ▷ discrete working length (non-differentiable)
3: $\tilde{\mathbf{x}} \leftarrow \mathbf{T}_{\text{in}}\,\mathbf{x}$          ▷ forward transform
4: $\hat{\mathbf{x}} \leftarrow \mathcal{P}(\tilde{\mathbf{x}},\, d_{\text{out}})$          ▷ pad/truncate to $d_{\text{out}}$
5: $\mathbf{k} \leftarrow [1, \ldots, d_{\text{out}}]^\top$
6: $\mathbf{w} \leftarrow \operatorname{clip}\big((q + 1 - \mathbf{k})/\ell,\ 0,\ 1\big)$          ▷ relaxed mask $\mathbf{w}(q)$
7: $\tilde{\mathbf{y}} \leftarrow \mathbf{w} \odot \hat{\mathbf{x}}$          ▷ apply relaxed mask
8: $\mathbf{y} \leftarrow \mathbf{T}_{\text{out}}^{-1}\,\tilde{\mathbf{y}}$          ▷ inverse transform with $\mathbf{y} \in \mathbb{R}^{d_{\text{out}}}$
9: return $\mathbf{y}$
```

Transform. A key insight is that the employed transforms are generated as a function of dimensionality: $\mathbf{T}_n = \mathcal{T}(n) \in \mathbb{R}^{n\times n}$. This lets the basis adapt automatically to resolution, *e.g.*, $\mathbf{T}_{\text{in}} \in \mathbb{R}^{d_{\text{in}}\times d_{\text{in}}}$ and $\mathbf{T}_{\text{out}} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{out}}}$. This property is critical for upsampling: it ensures gradients propagate through a basis that matches the learned output size. The FFT, used in DiffStride, is a natural but non-exclusive instantiation of $\mathcal{T}(\cdot)$.

Pad/Truncate. We unify length adjustment with an operator $\mathcal{P}$ that either truncates or zero-pads to produce the working length d_{out}. Unlike DiffStride, which slices only (enforcing $d_{\text{out}} \le d_{\text{in}}$), this definition also supports zero-padding when $d_{\text{out}} > d_{\text{in}}$, enabling learnable upsampling. For $\tilde{\mathbf{x}} \in \mathbb{R}^{d_{\text{in}}}$, $\hat{\mathbf{x}} \in \mathbb{R}^{d_{\text{out}}}$, and $m = \min(d_{\text{out}}, d_{\text{in}})$,

$$\hat{\mathbf{x}}_k = \big[\mathcal{P}(\tilde{\mathbf{x}},\, d_{\text{out}})\big]_k = \begin{cases} \tilde{x}_k, & k \le m, \\ 0, & k > m. \end{cases}$$

Differentiable mask. We define a continuously relaxed mask, $\mathbf{w}$, characterized by a linear ramp of length $\ell > 0$, tapering from 1 to 0 near the boundary similarly to [3]. However, we shift the ramp by 1 so that the number of nonzero elements equals d_{out}, improving interpretability (*i.e.*, $d_{\text{out}} = \lceil q \rceil$):

$$\underbrace{\mathbf{w}(q)}_{d_{\text{out}}} :\ w_k(q) = \operatorname{clip}\left(\frac{q+1-k}{\ell}, 0, 1\right), \quad k = 1, \ldots, d_{\text{out}}.$$

We use $\ell = 4$ in our experiments.

Forward pass.

$$\underbrace{\mathbf{y}}_{d_{\text{out}}} = \underbrace{\mathbf{T}_{\text{out}}^{-1}}_{d_{\text{out}} \times d_{\text{out}}} \Big(\underbrace{\mathbf{w}(q)}_{d_{\text{out}}} \odot \underbrace{\mathcal{P}\big(\underbrace{\mathbf{T}_{\text{in}}}_{d_{\text{in}} \times d_{\text{in}}} \underbrace{\mathbf{x}}_{d_{\text{in}}}, d_{\text{out}} \big)}_{d_{\text{out}}} \Big).$$

4. GRADIENT UPDATE

A natural concern with learnable upsampling is that zero-padding creates dead regions with no gradient. However, with $q = d_{\text{in}} s$, both the relaxed taper $\mathbf{w}(q)$ and the output transform $\mathbf{T}_{\text{out}} = \mathcal{T}(d_{\text{out}})$ depend on q (hence on s). Thus, even when $\mathcal{P}$ inserts zeros, gradients flow through the taper and the changed output basis. Concretely, let $\frac{\partial L}{\partial \mathbf{y}}$ be the upstream gradient. By the chain rule, $\frac{\partial L}{\partial s} = \frac{\partial L}{\partial q} \frac{\partial q}{\partial s} = d_{\text{in}} \frac{\partial L}{\partial q}$, yielding:

$$\frac{\partial L}{\partial s} = d_{\text{in}} \sum_{k=1}^{d_{\text{out}}} \Big[\underbrace{\mathbf{T}_{\text{out}}^{-\top} \frac{\partial L}{\partial \mathbf{y}}}_{d_{\text{out}}} \; \underbrace{\mathcal{P}(\mathbf{T}_{\text{in}} \mathbf{x}, d_{\text{out}})}_{d_{\text{out}}} \; \underbrace{\frac{\partial \mathbf{w}(q)}{\partial q}}_{d_{\text{out}}} \Big]_k .$$

Taper-only simplification. The gradient of the differentiable mask is given by:

$$\underbrace{\frac{\partial \mathbf{w}(q)}{\partial q}}_{d_{\text{out}}} = \begin{cases} \frac{1}{\ell}, & q + 1 - \ell < k < q + 1, \\ 0, & \text{otherwise}, \end{cases}$$

with breakpoints at $k = q + 1 - \ell$ and $k = q + 1$. We *define* $\frac{\partial w_k}{\partial q} := 0$ at these endpoints for stability.

Final gradient. Since $\frac{\partial \mathbf{w}(q)}{\partial q}$ is nonzero only on the taper,

$$\frac{\partial L}{\partial s} = \frac{d_{\text{in}}}{\ell} \sum_{k=\lfloor q+1-\ell \rfloor + 1}^{d_{\text{out}}} \Big[\underbrace{\mathbf{T}_{\text{out}}^{-\top} \frac{\partial L}{\partial \mathbf{y}}}_{d_{\text{out}}} \; \underbrace{\mathcal{P}(\mathbf{T}_{\text{in}} \mathbf{x}, d_{\text{out}})}_{d_{\text{out}}} \Big]_k .$$

Even when $d_{\text{out}} > d_{\text{in}}$ and $\mathcal{P}$ pads zeros, the nonzero taper entries are mapped by $\mathbf{T}_{\text{out}}^{-\top}$, whose columns depend on the learned output dimensionality. As d_{out} changes, the output basis changes, allowing gradient flow in the upsampling case.

5. EXPERIMENTS AND RESULTS

We test whether *Resolution Layers* improve accuracy by allocating resolution more effectively than alternatives. Without constraints, a Resolution-enabled model may trivially enlarge intermediate feature-map sizes. Any apparent gain could then be attributed to increased capacity rather than improved allocation. To prevent this confound, we evaluate all methods under a comparable *complexity budget* that fixes the model's aggregate resolution. Within this budget, allocation is learned—not prescribed: the model determines which layers warrant more or less resolution. Therefore, improvements can be attributed to improved allocation under comparable capacity.

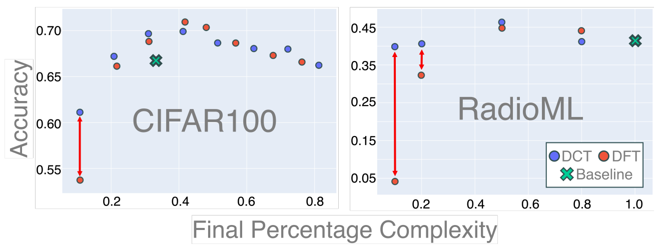

Fig. 3. Transform choice under matched budgets on CIFAR-100 and RadioML: similar at high budgets, DCT substantially better near 10% complexity.

Complexity definition. We define model complexity as the sum of layerwise complexities, $C = \sum_l c_l$. In the 1D case, the layer complexity is $c_l = q_l \, s_l$ (input length q_l times learned scale s_l for layer l). In the 2D case, the layer complexity is $c_l = q_{h,l} s_{h,l} + q_{w,l} s_{w,l}$, where w and h denote height and width respectively. In all Resolution models, we initialize with $s = 1$ (nominal resolution) to avoid initialization bias. For baselines without learnable scales, C is computed from fixed feature-map sizes as the baseline complexity.

Because the target sizes q are continuous, the aggregate complexity C is differentiable and can be incorporated directly into the objective. This makes feature-map resolution an optimizable variable rather than a fixed hyperparameter, enabling auxiliary losses on complexity while letting the model learn the allocation from data.

5.1. Matched Budgets: Impact of Transform Choice

We study two instantiations of $\mathcal{T}$: the discrete Fourier transform (DFT, as in DiffStride) and the discrete cosine transform (DCT)—to isolate how the basis affects accuracy at comparable complexity. To compare one-to-one, we train with the standard cross-entropy classification loss, $\mathcal{L}_{\text{cls}}$, and add a quadratic penalty that pulls the aggregate complexity toward a target: $\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda \big(\frac{C}{C_{\text{full}}} - p^\star \big)^2$, where C is the realized aggregate complexity, C_{full} is the no-resizing reference (all s=1), and $p^\star \in (0, 1]$ is the target fraction of C_{full}. Normalizing by C_{full} expresses the budget in percentage terms, making targets comparable across datasets. We set $\lambda = 100$ to strongly encourage adherence to $p^\star$, so complexities concentrate near the target. Sweeping $p^\star$ then yields trade-off curves that reveal where DCT and DFT diverge as the budget tightens, enabling a direct comparison without capacity confounds.

Datasets and architectures. We use simple, established backbones in two distinct domains to isolate transform effects. For **CIFAR-100** (vision), we adopt **ResNet-18** and replace traditional fixed-rate resizing (e.g., strided convolutions, pooling) at their standard stages with *Resolution Layers* in the same locations [15, 16]. For **RadioML 2018.01A** (radio) [17], we adapt the 1D CNN from [18]: seven 1D convolutions (kernel size of 3, 64 channels, same padding), followed by global mean-and-variance pooling and three linear layers $[128,\ 128,\ 24]$ using ReLU activations and a final softmax (24 classes). The baseline has no resizing, so we *insert a Resolution Layer after each convolution.*

Training. For CIFAR-100, models are trained for 80 epochs with stochastic gradient descent (SGD) (momentum 0.99) and batch size 32; the learning rate starts at 0.1 with $\times 0.1$ decays at epochs 20, 40, and 60. For RadioML 2018.01A, we train for 60 epochs with batch size 128; the learning rate starts at 0.01 with $\times 0.1$ decays at epochs 15, 30, and 45.

Results. Figure 3 shows the model performance versus complexity for DCT and DFT models. We also include *fixed-architecture baselines* retaining the original resizing with complexities computed from the fixed feature-map sizes. Accuracy on RadioML is expected to be lower as evaluation spans signal to noise ratios (SNR) from -20 to $30\,\text{dB}$, where reliable classification at very low SNR is not anticipated.

At baseline complexity ($p^\star = C_{\text{base}}/C_{\text{full}}$), Resolution models exceed the accuracy of the fixed-architecture baselines on both CIFAR-100 and RadioML. This indicates that allowing the network to learn *where and when* to allocate resolution yields better use of a given budget than prescribing a fixed layout. The DCT and DFT are comparable at higher budgets, but the difference widens under aggressive compression: near a 10% budget, DCT attains substantially higher accuracy. Accordingly, we adopt DCT for the remainder of the experiments.

5.2. Capped Budget: Object Detection

While fixed complexity targets are useful for trade-off curves, real deployments are usually constrained by a *maximum* compute or memory budget. To reflect this, we use a hinge loss on complexity that penalizes only when the budget is exceeded.

For this experiment, we evaluate Resolution Layers on *object detection* using a YOLO-style architecture [2] with a backbone for feature extraction, a neck for multi-scale fusion via PANet [19], and a prediction head. Concretely, we use **YOLOv8-XS**, tuned and provided by KerasCV [20] with a ResNet-50 backbone [16] pre-trained on ImageNet [21], and report mAP at IoU 0.5 on Pascal VOC 2007 [22].

To isolate the effect of learned resolution and avoid confounding from pretraining, we keep the ImageNet backbone frozen, and enable Resolution Layers *only* in the PANet (using the DCT, per Section 5.1). To remain compatible with the YOLO head and loss, PANet outputs must have fixed shapes, so we apply fixed-DCT resizing at the PANet outputs (frequency-domain zero-padding followed by the inverse DCT). Inside a Resolution-enabled PANet, two branches destined for concatenation may learn different resolutions; given branch shapes (h_1, w_1) and (h_2, w_2), we set the target $(h_{\text{out}}, w_{\text{out}}) = (\max(h_1, h_2),\ \max(w_1, w_2))$ and DCT-resize *both* branches to $(h_{\text{out}}, w_{\text{out}})$ before concatenation.

Capped objective. We cap *PANet complexity* C_{PAN} (sum over PANet only) at the baseline budget $C_{\text{base}} = 11{,}500$ by optimizing $\mathcal{L} = \mathcal{L}_{\text{YOLO}} + \lambda_{\text{hinge}} \max(0,\ C_{\text{PAN}} - C_{\text{base}})$, where $\mathcal{L}_{\text{YOLO}}$ is the standard YOLOv8 loss (box, objectness, class) [2, 20]. We sweep $\lambda_{\text{hinge}} \in \{0, 10^{-3}, 10^{-2}\}$ to probe the accuracy–complexity trade-off under a common cap; note that $\lambda_{\text{hinge}}{=}0$ removes the penalty entirely, allowing the model to freely explore PANet resolutions without any complexity constraint.

Training. We train all models for 80 epochs with the backbone frozen, using SGD and a piecewise learning rate from 1×10^{-3} down to 1×10^{-5}.

Results. At baseline-matched complexity, Resolution-enhanced PANets improve mAP over the tuned baseline, echoing the matched-budget experiments: when resolution is learned rather than prescribed, the network uses the complexity budget more effectively. Furthermore, Resolution models achieve *higher mAP at lower total PANet complexity* (up to +4.5% mAP with 19.5–31.0% less complexity than YOLOv8-XS; Table 1). Notably, with no penalty ($\lambda_{\text{hinge}}{=}0$), the model attains 47.1% mAP at complexity 9,260, surpassing the tuned baseline (44.7% mAP at 11,500); this suggests conventional fixed designs may carry excess resolution, and that making resolution learnable can discover leaner, higher-accuracy configurations without exhaustive hyperparameter sweeps. These gains persist across the hinge weights tested λ_{hinge}, indicating that the effect is not tied to a single regularization value.

Table 1. Object detection results on Pascal VOC 2007. *Resolution models add two parameters (height- and width-rates) per Resolution layer. Best values are shown in **bold**.

Model	**mAP$_{0.5}$ ↑**	**PANet complexity ↓**	**Trainable params.**
KerasCV baseline	44.7%	11,500	1.28M
DiffStride	*Unable to upsample*		
Resolution (ours)			1.28M*
$\lambda = 0$	**47.1%**	9,260	
$\lambda = 0.001$	46.9%	8,665	
$\lambda = 0.01$	47.0%	**7,929**	

6. CONCLUSION

We introduce Resolution Layers, a simple, transform-agnostic mechanism that makes feature map resolution learnable, supporting both downsampling and upsampling in a fully differentiable way. Across image, radio, and object detection domains, learned allocation outperforms fixed layouts at comparable complexity, underscoring the value of letting networks decide where and when to spend resolution. Further, results show treating resolution as a first-class, differentiable variable yields better higher accuracy even while reducing overall complexity.

7. REFERENCES

[1] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.

[2] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi, “Real-time flying object detection with yolov8,” *ArXiv*, vol. abs/2305.09972, 2023.

[3] Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour, “Learning strides in convolutional neural networks,” *ICLR*, 2022.

[4] George Papandreou, Iasonas Kokkinos, and Pierre-André Savalle, “Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 390–399.

[5] Yunho Jeon and Junmo Kim, “Active convolution: Learning the shape of convolution for image classification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4201–4209.

[6] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei, “Deformable convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[7] Fabian Groh, Patrick Wieschollek, and Hendrik PA Lensch, “Flex-convolution: Million-scale point-cloud learning beyond grid-worlds,” in *Asian Conference on Computer Vision*. Springer, 2018, pp. 105–122.

[8] Silvia L Pintea, Nergis Tömen, Stanley F Goes, Marco Loog, and Jan C van Gemert, “Resolution learning in deep convolutional networks using scale-space theory,” *IEEE Transactions on Image Processing*, vol. 30, pp. 8342–8353, 2021.

[9] Luke Wood and Eric C Larson, “Parametric spectral filters for fast converging, scalable convolutional neural networks,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 2800–2804.

[10] David W. Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn, “CKConv: Continuous kernel convolution for sequential data,” in *International Conference on Learning Representations*, 2022.

[11] Dong-Hwan Jang, Sanghyeok Chu, Joonhyuk Kim, and Bohyung Han, “Pooling revisited: Your receptive field is suboptimal,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 539–548.

[12] Linjiang Huang, Rongyao Fang, Aiping Zhang, Guanglu Song, Si Liu, Yu Liu, and Hongsheng Li, “Fouriscale: A frequency perspective on training-free high-resolution image synthesis,” in *European Conference on Computer Vision*, 2024.

[13] Shikun Liu, Zhe Lin, Yilin Wang, Jianming Zhang, Federico Perazzi, and Edward Johns, “Shape adaptor: A learnable resizing module,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[14] Jose Javier Gonzalez Ortiz, John V. Guttag, and Adrian V. Dalca, “Scale-space hypernetworks for efficient biomedical imaging,” in *37th Conference on Neural Information Processing Systems*, 2023.

[15] Alex Krizhevsky, Geoffrey Hinton, et al., “Learning multiple layers of features from tiny images,” 2009.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[17] Timothy James O’Shea, Tamoghna Roy, and T. Charles Clancy, “Over-the-air deep learning based radio signal classification,” in *IEEE Journal of Selected Topics in Signal Processing*. IEEE, 2018, vol. 12:1, pp. 168–179.

[18] Clayton A. Harper, Lauren Lyons, Mitchell A. Thornton, and Eric C. Larson, “Enhanced automatic modulation classification using deep convolutional latent space pooling,” in *2020 54th Asilomar Conference on Signals, Systems, and Computers*, 2020.

[19] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia, “Path aggregation network for instance segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8759–8768.

[20] Luke Wood, Zhenyu Tan, Ian Stenbit, Jonathan Bischof, Scott Zhu, François Chollet, Divyashree Sreepathihalli, Ramesh Sampath, et al., “Kerascv,” `https://github.com/keras-team/keras-cv`, 2022.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[22] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2015.