ScaleFlow: Efficient Deep Vision Pipeline with Closed-Loop Scale-Adaptive Inference

Yuyang Leng
George Mason University
Fairfax, VA, United States
yleng2@gmu.edu

Renyuan Liu
George Mason University
Fairfax, VA, United States
rliu23@gmu.edu

Hongpeng Guo
University of Illinois
Urbana-Champaign
Champaign, IL, United States
hg5@illinois.edu

Songqing Chen
George Mason University
Fairfax, VA, United States
sqchen@gmu.edu

Shuochao Yao
George Mason University
Fairfax, VA, United States
shuochao@gmu.edu

ABSTRACT

Deep visual data processing is underpinning many life-changing applications, such as auto-driving and smart cities. Improving the accuracy while minimizing their inference time under constrained resources has been the primary pursuit for their practical adoptions. Existing research thus has been devoted to either narrowing down the area of interest for the detection or miniaturizing the deep learning model for faster inference time. However, the former may risk missing/delaying small but important object detection, potentially leading to disastrous consequences (e.g., car accidents), while the latter often compromises the accuracy without fully utilizing intrinsic semantic information. To overcome these limitations, in this work, we propose ScaleFlow, a closed-loop scale-adaptive inference that can reduce model inference time by progressively processing vision data with increasing resolution but decreasing spatial size, achieving speedup without compromising accuracy. For this purpose, ScaleFlow refactors existing neural networks to be scale-equivariant on multiresolution data with the assistance of wavelet theory, providing predictable feature patterns on different data resolutions. Comprehensive experiments have been conducted to evaluate ScaleFlow. The results show that ScaleFlow can support anytime inference, consistently provide 1.5× to 2.2× speed up, and save around 25% ∼ 45% energy consumption with < 1% accuracy loss on four embedded and edge platforms.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Computer vision**.

KEYWORDS

Scale-adaptive Inference, Object Detection, Scale-equivariant, Wavelet Transform, Edge Computing, Anytime Inference

MM '23, October 29-November 3, 2023, Ottawa, ON, Canada

ACM ISBN 979-8-4007-0108-5/23/10.
https://doi.org/10.1145/3581783.3612412

ACM Reference Format:
Yuyang Leng, Renyuan Liu, Hongpeng Guo, Songqing Chen, and Shuochao Yao. 2023. ScaleFlow: Efficient Deep Vision Pipeline with Closed-Loop Scale-Adaptive Inference. In *Proceedings of the 31st ACM International Conference on Multimedia (MM '23), October 29-November 3, 2023, Ottawa, ON, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3581783.3612412

1 INTRODUCTION

With the continuous advancement of machine/deep learning techniques and widely deployed mobile and ubiquitous cameras, visual data processing has become an essential building block for many of today's applications, such as self-driving cars, extended reality, and smart cities. Such applications all require complex semantic understanding and localization of visual input, for which deep learning is now the predominant solution [4, 6, 21]. As the core of such visual data processing, deep neural networks, however, are notorious for consuming excessive runtime resources, posing significant challenges in speeding up deep vision pipelines on embedded, mobile, and edge systems.

Many efforts have been made to improve the runtime efficiency of deep neural networks by leveraging the temporal correlations or the historical spatial distributions of objects to filter out frames or spatial areas with a low probability of containing objects of interest [8, 17, 19, 21]. However, besides being application-specific and relying on assumptions such as stationary cameras with fixed angles and locations or requiring extensive offline profiling and tuning before each deployment, these solutions often risk missing (or delaying) timely detection of small but important objects and potentially leading to disastrous consequences (e.g., car accidents). On the other hand, plenty of efforts have also been made to harness the parameter redundancy of deep learning models by reducing the model complexity, which includes model compression [13, 14, 41], weight quantization [7, 9, 28], efficient neural network design and search [16, 22, 35, 36]. Although these techniques can simplify constructing a miniature model, they often trade accuracy for reduced inference time and fail to exploit intrinsic semantic information to allocate computation at runtime for inputs of varying complexity.

To address the limitations of existing research, we propose ScaleFlow, a closed-loop scale-adaptive inference pipeline that can be integrated into any existing deep vision system. Unlike prior efforts that are mostly based on feedforward open-loop optimization, ScaleFlow dynamically allocates computing resources using inference

results as feedback to deliver progressively better outcomes. ScaleFlow refactors the inference pipeline in the scale domain such that the inference result of lower-resolution data can identify regions of high uncertainty (e.g., areas with overlapping items, small objects, or complex scenes) for further processing with higher-resolution input in a closed-loop manner.

The design of ScaleFlow, however, must address the following two key challenges. First, the closed-loop inference over the scale domain necessitates the deep vision model to have predictable behavior over the scale space. Just as time-invariant makes time-based closed-loop systems more robust and stable, we want to make deep vision models in closed-loop scale-adaptive inference scale-equivariant so that all objects detected using high-resolution input can also be identified using the corresponding lower-resolution input (with higher uncertainties). While many architectures have attempted to train on different scales [11, 31], they are all data-driven and fail to provide the necessary guarantees. Therefore, ScaleFlow requires deep neural networks to be scale equivariance so that high-complexity regions are not overlooked during the processing of low-resolution inputs. Unfortunately, all existing scale-equivariant designs do not take model efficiency into account [34, 38, 40]. We must either re-upscale the "down-scaled" data towards the maximum feasible resolution, incurring even more inference time or suffer significant performance degradation due to the discretization. In ScaleFlow, we leverage multiresolution wavelet analysis [3] to construct closed-form scale-equivariant mappings, which strictly preserve the scale-equivariant property while incurring almost no additional overhead when processing downscaled inputs. Additionally, our approach for achieving scale-equivariant mapping using wavelets is designed to be highly versatile and can be easily integrated into a wide range of convolutional networks through the replacement of their convolutional layers. This plug-and-play method streamlines implementation by minimizing the need for significant architectural modifications to the underlying network.

The second challenge in designing ScaleFlow is to produce regions of uncertainty in an effective and efficient way for further processing at a higher resolution. On the one hand, existing deep vision models, such as object detection networks, estimate objects' semantic classes and locations. However, they only provide a way of quantifying class uncertainty, not location uncertainty [11, 29–31, 42, 44]. On the other hand, recent works on location uncertainty leverage sampling-based Bayesian inference [20, 25], incurring significant runtime overhead in practice. We propose an efficient way of estimating classification and localization uncertainty by exploiting the model's existing "redundant" computation. Most object detection neural networks produce considerably more bounding box candidates than their final output before the final layer. Rather than following the convention of treating most candidates as "false positives" and pruning them using techniques such as Non-Maximum Suppression (NMS), we reuse them to estimate output uncertainties. As a result, we can quantify both classification and localization uncertainty with almost no extra expense at runtime.

To evaluate the performance of ScaleFlow, we implement webcam object detection applications with ScaleFlow on four distinct platforms, including an embedded GPU platform (NVIDIA Xavier) [2], a low-power AI accelerator (Intel NCS2) [1], a tiny embedded device (Raspberry Pi 4B), and an edge GPU server (with NVIDIA 3080). We select several representative object detection neural network models, YOLOv3 [31], CenterNet [44], RetinaNet [23], and FCOS [37] to evaluate our ScaleFlow as an object-detection service with both live streaming from a webcam and pseudo streaming from the COCO dataset [24]. Compared to other state-of-the-art baselines, ScaleFlow allows anytime inference and can provide at least 1.5× ∼ 2.2× end-to-end speed-up and save around 25% ∼ 45% energy consumption with less than 1% accuracy loss on four IoT computation platforms.

The highlights of our contributions include:

- We design a closed-loop scale-adaptive pipeline that can support anytime inference and adaptively allocate computing resources based on the complexity of inference tasks.
- We propose closed-form scale-equivariant mappings for neural network operations based on wavelet analysis concerning both runtime efficiency and scale equivariance.
- We devise an efficient uncertainty estimation technique by reusing the existing computation in object detection models.
- We implement ScaleFlow on four diverse platforms, achieving at least 1.5× ∼ 2.2× end-to-end speed-up and saving around 25% ∼ 45% energy consumption with less than 1% accuracy loss compared to other SOTA techniques.

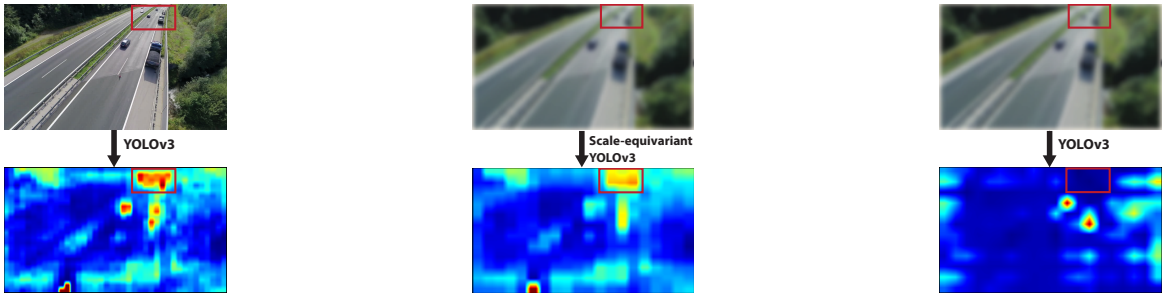
2 A MOTIVATIONAL STUDY

In this section, we explore opportunities for improving neural network speed through scale-adaptive designs, using a motivational study with YOLOv3 [31] object detection on COCO dataset [24]. Our scale-adaptive inference operates on two image resolutions: full (640×640) and downscaled (320×320). It involves: i) processing the downscaled image; ii) cropping and merging areas of uncertainty; and iii) processing the cropped full-resolution image. The uncertainty metrics include objectness score, class and location uncertainty. The employed neural networks can be identical, separately trained, or coupled by specific rules such as scale-equivariant mapping.

Two commonly used options are training two independent models with two different resolutions [1] and training a single model with two resolutions as data augmentation [2]. We denote these two scale-adaptive baselines as "ScaleFlow-IND" and "ScaleFlow-DataAug". Assume we have a satisfactory method for identifying uncertainty regions at this moment. As illustrated in Figure 2a, when achieving comparable mean average precision (mAP) to the YOLOv3 model with full-resolution inputs, both ScaleFlow-IND and ScaleFlow-DataAug take more time for inference.

To understand the underlying cause, in Figures 1a and 1c, we plot feature maps of YOLOv3 with full-resolution input and ScaleFlow-IND with downscaled input (the feature map of ScaleFlow-DataAug has a similar pattern and is omitted). We can see several cars on the top of the input image (Figure 1a), marked by a red rectangle. YOLOv3 can detect these vehicles with full-resolution input, as indicated by its feature map's high-intensity response (Figure 1a). The corresponding region in ScaleFlow-IND's feature map shows a low-intensity response, similar to other background regions (Figure 1c). As a result, ScaleFlow-IND might have difficulty distinguishing between the marked region containing cars and other background regions. So it has to crop a large area for further processing at full resolution. To validate this hypothesis, we measure the minimum

[1] https://pjreddie.com/darknet/yolo/
[2] https://github.com/ultralytics/yolov3/releases/download/v9.0/yolov3.pt

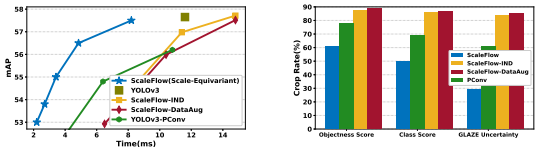


(a) YOLOv3 Train: full resolution; Test: full resolution

(b) Scale-equivariant YOLOv3 Test: downscaled input

(c) YOLOv3 Train: downscaled input; Test: downscaled input

Figure 1: Feature maps (red: high-intensity & blue: low-intensity) from different YOLOv3 models. (a) A YOLOv3 model is trained and tested with full-resolution images; (b) A scale-equivariant YOLOv3 model is trained with full-resolution images but tested with downscaled images; (c) A YOLOv3 model is trained and tested with downscaled-resolution images.



(a) Mean average precision (mAP) and time tradeoff.

(b) The min cropped area (%) to achieve full-resolution mAP.

Figure 2: System performance of scale-adaptive inferences.

percentage of cropped area that each model needs to attain comparable performance to the original YOLOv3 at full resolution. As shown in Figure 2b, even with the best uncertainty estimation technique (as discussed later), ScaleFlow-IND and ScaleFlow-DataAug need more than 80% cropped area to achieve comparable mAP.

Since models at two resolutions are trained independently or with data-driven strategies like data augmentation, downscaled models do not get enough input details to make good decisions. They tend to underestimate blurry, unclear, and tiny objects to decrease false positives and increase detection rates. This underestimate is bad for scale-adaptive inference since the model cannot distinguish between background and uncertain areas.

Moreover, existing scale-equivariant models do not fit the scale-adaptive inference due to runtime efficiency (i.e., processing downscaled input at full resolution) or performance degradation arising from discretization or dilation [34, 38, 40]. We thus implement a traditional scale-equivariant model using strided convolution at full resolution (i.e., an option provided in [38]) and call it "PConv". As shown in Figure 2, PConv cannot achieve the same mAP as YOLOv3 due to the limitation of network capacity at full resolution. Therefore, one essential challenge for ScaleFlow is to develop efficient and effective scale-equivariant mappings for neural network operations. As illustrated in Figures 1a and 1b, with scale-equivariant mappings, ScaleFlow obtains well-calibrated information from low-resolution input and significantly reduces the workload at full resolution.

Another essential part is the uncertainty measure used to detect and crop areas of uncertainty for further processing with a neural network at full resolution. As shown in Figure 2b, the choice of uncertainty measure is not a critical issue with baselines (ScaleFlow-IND and ScaleFlow-DataAug) since their downscaled neural networks do not provide well-calibrated predictions. For ScaleFlow, however, we need a better uncertainty quantification design to precisely identify the areas of uncertainty for better run-time efficiency. The objectness score and entropy of class scores are widely used uncertainty metrics [10, 25]. Unfortunately, the objectness score cannot distinguish between confident and uncertain outputs, while the class score does not include localization uncertainty. ScaleFlow, therefore, requires an efficient uncertainty estimation component that accounts for both class and localization uncertainties.

3 SCALEFLOW DESIGN

This section presents the detail of ScaleFlow design, containing scale-equivariant mapping and uncertainty estimation. Figure 3 sketches a single feedback loop unrolling of scale-adaptive inference: (1) execute the neural network with the downscaled input, (2) cluster output bounding boxes and estimate their uncertainty as feedback, (3) pick high-uncertainty areas and merge them at a higher resolution, (4) execute the neural network with the merged high-resolution input, and (5) cluster all output bounding boxes.

3.1 Scale-Equivariant Mapping

In this section, we present scale-equivariant mappings for neural network operations and thus enable a scale-equivariant network that can directly process downscaled inputs.

3.1.1 Preliminary: Wavelet Transform. It is known that we can decompose the signal into a linear combination of the orthogonal basis. Wavelet transform leverages scaled and shifted versions of the bandpass mother wavelet function $\psi(n)$ and lowpass scaling function $\phi(n)$ as the basis. Scaled and shifted forms are denoted as

$$\psi_{j,k}[n] = 2^{-j/2}\psi((n-k)/2^j), \quad \phi_{j,k}[n] = 2^{-j/2}\phi((n-k)/2^j),$$

where $n, k, j \in \mathbb{Z}$. k and j control the shifting and scaling of a function. With a *larger* scaling index j, the function's support *expands*, concentrating more on *lower* frequency components.

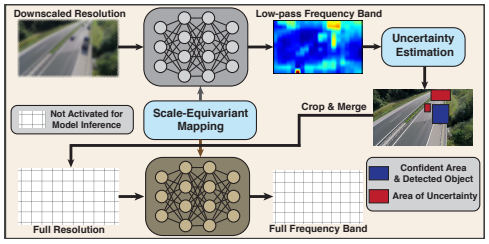


Figure 3: The overview of scale-adaptive inference.

Thanks to the wavelet's recursive equivalence property, the discrete wavelet transform and its reconstruction can be computed using a fast recursive algorithm. We can compute the $(j+1)$-th level

coefficients of $\{\psi_{j+1,k}, \phi_{j+1,k}\}$ based on the j-th level coefficients of $\{\phi_{j,k}\}$ and vice versa by defining two analysis filters:

$$h_L[n] = \sum_k \phi_{1,0}[k]\phi_{0,0}[k-2n], \quad h_H[n] = \sum_k \psi_{1,0}[k]\phi_{0,0}[k-2n]. \quad (1)$$

For orthogonal wavelet, $h_L[n]$ and $h_H[n]$ satisfy a quadrature mirror filter relationship [3]. The frequency magnitude responses of two analysis filters are symmetric around $\pi/2$. For Haar wavelet, $h_L[n] = [1/\sqrt{2}, 1/\sqrt{2}]$ and $h_H[n] = [1/\sqrt{2}, -1/\sqrt{2}]$.

Then, we denote $x^L_{j+1}[k]$ and $x^H_{j+1}[k]$ as the wavelet decomposition coefficients of $\phi_{j,k}$ and $\psi_{j,k}$, respectively ($x^L_0[k]$ represents the original discrete signal). We can therefore formulate wavelet transform, including decomposition and reconstruction, as cascading filter bank using recursive equations.

Wavelet filter bank decomposition $\mathcal{L}_h(x)$:

$$\begin{aligned} \mathcal{L}_{h_L}(x^L_j) : x^L_{j+1}[k] &= \sum_n h_L[n] x^L_j[n+2k], \\ \mathcal{L}_{h_H}(x^L_j) : x^H_{j+1}[k] &= \sum_n h_H[n] x^L_j[n+2k]. \end{aligned} \quad (2)$$

These operations (2) can also be interpreted as *the stride-2 convolution* with kernels $h_L[n]$ and $h_H[n]$ (denoted as $\mathcal{L}_{h_L}(\cdot)$ and $\mathcal{L}_{h_H}(\cdot)$) in the deep-learning dialect for efficient implementation using tensor programming libraries such as TensorFlow and PyTorch.

Wavelet filter bank reconstruction:

$$\begin{aligned} x^L_j[k] &= \sum_n \tilde{h}_L[k-2n] x^L_{j+1}[n] + \sum_n \tilde{h}_H[k-2n] x^H_{j+1}[n] \\ &= x^L_{j+1} \otimes \tilde{h}_L + x^H_{j+1} \otimes \tilde{h}_H \end{aligned} \quad (3)$$

where we call $\tilde{h}_L[n]$ and $\tilde{h}_H[n]$ synthesis filters. Since the analysis filters are quadrature mirror filters, $\tilde{h}_L[n] = h_L[n]$ and $\tilde{h}_H[n] = h_H[n]$. Similarly, given synthesis filters with length 2 (such as Haar), operation (3) can be viewed as *the addition of two Kronecker product* (denoted as $\otimes$) for efficient implementation.

According to equation (2), the wavelet coefficients are computed by recursively computing the coefficients at each scale, with $x^L_0[k]$ initialized as the discrete signal x. At each step, the signal is decomposed into low-frequency and high-frequency coefficients by (stride-2) convolving the low-frequency coefficients obtained from the previous step with the low-pass and high-pass analysis filters (i.e., h_L and h_H described by equation (1)). The wavelet transform thus effectively partitions the signal into perfectly reconstructable frequency bands defined by the wavelet functions.

3.1.2 Scale-Equivariant Convolution. Without loss of generality, we denote the 2×2 wavelet analysis filters (2) and synthesis filters (3) as $\mathcal{H} = \{h_{LL}, h_{LH}, h_{HL}, h_{HH}\}$. We use Haar wavelet for practical deployments in the paper to validate our designs. However, the scale-equivariant mappings proposed in the following sections are not limited to a specific wavelet type.

We start by building the scale-equivariant convolution, the most widely used operation in computer vision tasks. Convolution can be formulated as $\mathbf{Y}[i,j,c'] = \sum_{m,n,c} \mathbf{W}[m,n,c,c'] \cdot \mathbf{X}[i+m, j+n, c] + \mathbf{b}[c']$, where $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{W}$, and $\mathbf{b}$ denotes the input tensor, the output tensor, the convolution kernel, and the bias vector respectively. We assume spatial resolutions are both even.

We extend wavelet bank decomposition $\mathcal{L}_h(\cdot)$ defined by (2) to 3D input and output tensors by performing 2D wavelet decomposition on each channel individually, $\mathcal{L}_h(\mathbf{X}) : \mathcal{L}_h(\mathbf{X})[:,:,c] = \mathcal{L}_h(\mathbf{X}[:,:,c])$. To simplify the analysis, we omit other settings, such as padding (zero padding), dilation rate (= 1), and stride (= 1) at first, and then revisit these factors later. Since the wavelet transform halves the data resolution at each decomposition level, we have a tensor input with the full resolution, $\mathbf{X}$, and downscaled input, $\mathbf{X}_{LL} = \mathcal{L}_{h_{LL}}(\mathbf{X})$, obtained by wavelet decomposition.

The objectives of scale-equivariant convolution are twofold:

(1) To create an efficient equivariant operation that can operate directly on downscaled resolution while producing the same results as original convolution at full resolution followed by wavelet decomposition.
(2) To make the convolution that operates on the full resolution semipermeable in wavelet frequency bands to support the scale-equivariant mapping.

To fulfill the first objective, we want to derive the kernel $\mathbf{W}_L$ and the bias $\mathbf{b}_L$ for convolution on the downscaled resolution, so that for all $\mathbf{X}_{LL} = \mathcal{L}_{h_{LL}}(\mathbf{X})$, there exists

$$\mathbf{W}_L * \mathbf{X}_{LL} + \mathbf{b}_L = \mathcal{L}_{h_{LL}}(\mathbf{W} * (\mathbf{X}_{LL} \otimes h_{LL}) + \mathbf{b}), \quad (4)$$

where $\mathbf{X}_{LL} \otimes h_{LL}$ means that we reconstruct/approximate the input $\mathbf{X}$ at its full resolution only with its 2D low-pass wavelet frequency band $\mathbf{X}_{LL}$. We can reformulate the equation (4) as:

$$\begin{aligned} &\sum_{m',n',c} \mathbf{W}_L[m',n',c,c'] \cdot \mathbf{X}_{LL}[i+m', j+n', c] + \mathbf{b}_L[c'] \\ &= \sum_{p,q} h_{LL}[p,q] \Big(\sum_{m,n,c} \mathbf{W}[m,n,c,c'] \cdot \\ &\quad (\mathbf{X}_{LL} \otimes h_{LL})[2i+p+m, 2j+q+n, c] + \mathbf{b}[c'] \Big) \end{aligned} \quad (5)$$

We begin by determining the bias vector $\mathbf{b}_L$ for downscaled computation. Since the bias term is a scalar added to each channel, it is straightforward to derive the following equation:

$$\mathbf{b}_L = \sum_{p,q} h_{LL}[p,q] \cdot \mathbf{b} \quad (6)$$

To derive a convolution kernel for downscaled resolution, we interpret a stride-1 convolution with a $K \times K$ kernel $\mathbf{W}$ as four stride-2 convolutions with $\hat{K} \times \hat{K}$ kernels, where $\hat{K} = 2 \cdot \lceil (K+1)/2 \rceil$, by padding $\mathbf{W}$ with zeros on the {right-and-bottom, left-and-bottom, right-and-top, left-and-top} border, denoted as $\mathcal{W} = \{\mathbf{W}_{0,0}, \mathbf{W}_{1,0}, \mathbf{W}_{0,1}, \mathbf{W}_{1,1}\}$. Thus, for $p \in \{0,1\}$ and $q \in \{0,1\}$,

$$\begin{aligned} \mathbf{Y}[2i+p, 2j+q] &= \sum_{m,n} \mathbf{W}[m,n] \mathbf{X}[2i+p+m, 2j+q+n] \\ &= \sum_{m',n'} \mathbf{W}_{p,q}[m',n'] \mathbf{X}[2i+m', 2j+n'] \end{aligned} \quad (7)$$

We can represent $\mathbf{W}_{p,q}$ with wavelet transform as filter bank reconstruction (3). Moreover, we denote $\mathcal{L}_h(\mathbf{W}_{p,q})$ as $\mathbf{W}^{(h)}_{p,q}$, and $\mathbf{X}^{(K)}_{LL}[i,j]$ as $\mathbf{X}_{LL}[i{:}i+K, j{:}j+K, :]$. According to the mixed-product property [33], we can reformulate equations (5) and (7) as

$$\begin{aligned} &\sum_{p,q} h_{LL}[p,q] \sum_{m,n} \Big(\sum_{f \in \mathcal{H}} \mathbf{W}^{(f)}_{p,q} \otimes h_f \Big) \odot \Big(\mathbf{X}^{(\hat{K}/2)}_{LL}[i,j] \otimes h_{LL} \Big) \\ &= \sum_{p,q} h_{LL}[p,q] \sum_{m,n} \sum_{f \in \mathcal{H}} \Big(\mathbf{W}^{(f)}_{p,q} \odot \mathbf{X}^{(\hat{K}/2)}_{LL}[i,j] \Big) \otimes (h_f \odot h_{LL}) \\ &= \sum_{n',m'} \mathbf{X}^{(\hat{K}/2)}_{LL}[i,j] \odot \Big(\sum_{p,q} h_{LL}[p,q] \sum_{f \in \mathcal{H}} \mathbf{W}^{(f)}_{p,q} \sum_{p',q'} h_f \odot h_{LL} \Big) \end{aligned}$$

where $\odot$ denotes the Kronecker product (i.e., the element-wise multiplication), and $\mathcal{H} = \{h_{LL}, h_{LH}, h_{HL}, h_{HH}\}$.

Therefore, we can derive the convolution kernel for downscaled computation as follows:

$$\mathbf{W}_L = \sum_{p,q} h_{LL}[p,q] \sum_{f \in \mathcal{H}} \mathbf{W}^{(f)}_{p,q} \sum_{p',q'} (h_f \odot h_{LL})[p',q']$$

When we have orthonormal wavelet analysis filters, such as Haar wavelet, we can further simplify the expression of $\mathbf{W}_L$ as

$$\mathbf{W}_L = \mathcal{L}_{h_{LL}}\left(\sum_{p,q} h_{LL}[p,q]\cdot \mathbf{W}_{p,q}\right) \tag{8}$$

Based on equations (6) and (8), we can develop a scale-equivariant convolution operation that computes directly on the downscaled resolution for efficient processing. However, the scale equivariance is established based on the assumption that the input tensor $\mathbf{X}$ contains information only from the low-pass wavelet frequency band $\mathbf{X}_{LL}$. Thus, we must determine whether, except for the low-pass band, other wavelet frequency bands in the input tensor can influence the low-pass band in the output tensor (i.e., checking whether the convolution operation is semipermeable in wavelet frequency bands). Without loss of generality, assume that wavelet analysis filters are orthonormal, for $\kappa \in \{LH, HL, HH\}$, we have

$$\mathcal{L}_{h_{LL}}(\mathbf{W} * (\mathbf{X}_\kappa \otimes h_\kappa)) = \mathcal{L}_{h_\kappa}\left(\sum_{p,q} h_{LL}[p,q]\cdot \mathbf{W}_{p,q}\right) * \mathbf{X}_\kappa. \tag{9}$$

We can derive the above equation by adopting a similar procedure as (4)-(8). Therefore, unless the convolution kernel $\mathbf{W}$ has special structures (as we will discuss later), the convolution is not semipermeable in wavelet frequency bands.

Then, we need to design a convolution operation at full resolution that is semipermeable in wavelet frequency bands. The idea is simple: we will replace the low-pass band in the convolution output with the result of the downscaled convolution operation (4). Similar to (7), our semipermeable convolution consists of four stride-2 convolutions with kernels $\widetilde{\mathcal{W}} = \{\widetilde{\mathbf{W}}_{0,0}, \widetilde{\mathbf{W}}_{1,0}, \widetilde{\mathbf{W}}_{0,1}, \widetilde{\mathbf{W}}_{1,1}\}$. Thus, for $p \in \{0,1\}$ and $q \in \{0,1\}$, we have

$$\begin{aligned} &\mathrm{Y}[2i+p, 2j+q] = \sum_{m',n'} \widetilde{\mathbf{W}}_{p,q}[m',n']\mathrm{X}[2i+m', 2j+n'] + \mathbf{b} \\ &\text{s.t. } \widetilde{\mathbf{W}}_{p,q} = \left(\mathbf{W}_{p,q} - \sum_{p',q'} h_{LL}[p',q']\cdot \mathbf{W}_{p',q'} + (\mathbf{W}_L \otimes h_{LL})\right) \end{aligned} \tag{10}$$

where $\{\mathbf{W_{p,q}}\}$ is the zero-padded kernel set $\mathcal{W}$ defined in (7) and $\mathbf{W}_L$ is the downscaled convolution kernel defined in (8). The kernels $\{\widetilde{\mathbf{W}}_{p,q}\}$ can be precomputed offline after training, which does not incur additional inference overhead.

To this end, we can derive a scale-equivariant mapping for convolution operationat the original full resolution and the halved downscaled resolution in two steps:

(1) For the downscaled resolution, we formulate the convolution with kernel and bias according to equations (8) and (6).
(2) For the original full resolution, we formulate the convolution according to equation (10).

The scale-equivariant mapping for convolution does not create additional learnable parameters and depends only on $\mathbf{W}$ and $\mathbf{b}$, so it has no impact on the standard training/tuning procedure.

In addition, to simplify the analysis, the previous formulation assumes a convolution with zero padding, a dilation rate of one, and a stride of one. Now, we revisit these configurations. Assume that we have a convolution with dilation rate d, padding around the input tensor with p pixels, and stride s.

Dilation: we can treat a $k \times k$ dilated convolution with a dilation rate d as a normal $(k+(k-1)(d-1))\times(k+(k-1)(d-1))$ convolution by inserting holes (i.e., 0) between the $k \times k$ kernel elements. Then, using the same steps, we can create a scale-equivariant mapping for dilated convolution. Moreover, suppose the dilation rate d is an even integer. In that case, the dilated convolution is semipermeable in Haar wavelet frequency bands (i.e., equation (9) equals zero for any input using the Haar wavelet). As a result, we can use the original dilated convolution at its full resolution.

Padding: we pad the input with p pixels around. If p is an even integer, we generate the scale-equivariant mapping in the same way. If p is an odd integer, we set padding to $p+1$ and extend the $k \times k$ kernel into $(k+1)\times(k+1)$ by padding 0 around since the input at full and downscaled resolutions must be aligned.

Stride: when stride= s, as in equation (7), we interpret a stride-s convolution with a $k \times k$ convolution as four stride-$2s$ convolutions with $\hat{k} \times \hat{k}$, where $\hat{k} = 2 \cdot \lceil (k+s)/2 \rceil$, by padding the kernel with zeros on the right-and-bottom, left-and-bottom, right-and-top, left-and-top border. Then, we can follow the same steps to build the scale-equivariant mapping.

3.1.3 Scale-Equivariant Activation Function. The majority of activation functions, $f(\cdot)$, are element-wise nonlinear functions, making it difficult to discover a scale-equivariant equivalent, $f_L(\cdot)$, that operates at the downscaled resolution. It is thus necessary to reformulate the activation function to support the scale-equivariant mapping. However, changing the frequency response of the activation function output has a significant influence on its local element-wise spatial response. This is also why, to the best of our knowledge, most existing frequency-domain convolutional neural networks use activation functions in the spatial domain [12, 32, 39].

Fortunately, most widely adopted activation functions, such as ReLU, leaky-ReLU, GELU [15], and Swish [27], can be interpreted as modulating the amplitude of a signal by gating functions: $f(x) = g(x)\cdot x$. The gating function $g(x)$ is a sign function in ReLU, a shifted-and-scaled sign function in leaky-ReLU, a standard Gaussian cumulative distribution function in GELU, and a sigmoid function in Swish. Therefore, our scale-equivariant activation function replaces the trigger of the gating function with low-pass wavelet band coefficients: $f(\mathbf{X}) := \left(g(\mathbf{X}_{LL}) \otimes \mathbb{1}_2\right) \odot \mathbf{X}$ and $f_L(\mathbf{X}_{LL}) := g(\mathbf{X}_{LL}) \odot \mathbf{X}_{LL}$, where $\mathbb{1}_2$ denotes a 2×2 matrix of ones, $f(\cdot)$ denotes the activation function at the full resolution, and $f_L(\cdot)$ denotes the activation function at the downscaled resolution.

3.2 Uncertainty Estimation

ScaleFlow exploits a new opportunity by treating the existing unwanted or redundant outputs as extra information to estimate predictive (i.e., classification and localization) uncertainty. Most object detection neural networks output considerably more bounding boxes than the ground-truth objects in the image.

On the one hand, Non-Maximum Suppression (NMS) [5, 18] is intrinsically a clustering algorithm as it picks out "cluster heads" and prunes other bounding boxes.

ScaleFlow substitutes the pruning operation with the grouping operation, making NMS a clustering algorithm. With little overhead, we can merge the clustering computation into the existing NMS computation in the object detection pipeline.

On the other hand, given the clustering assignment, a "sampled" bounding box j in cluster i contains a prediction vector, including location coordinates $\mathbf{L}_j = [x_j, y_j, w_j, h_j]$, objectness score $[o_j]$, and class scores $\mathbf{C}_j$. Since class information is represented as discrete probability, we can evaluate classification uncertainty as the expectation of class entropy approximated by samples:

$$U_{cls}(\text{cluster}_i) = \textstyle\sum_j o_j \cdot H(\mathbf{C}_j) \tag{11}$$

where $U_{cls}(\cdot)$ denotes the classification uncertainty, and $H(\cdot)$ denotes the entropy. Since the objectness score (0~1) captures the quality of bounding box estimations, we quantify classification uncertainty as a weighted sum of entropy. . For localization uncertainty, ScaleFlow defines "mean" as the cluster head's bounding box. Since Intersection over Union (IoU) is a term widely adopted to describe the overlap between two boxes (ranged from 0~1), we use 1 − IoU to define the "deviation" of a bounding box from its cluster head.

$$U_{loc}(\text{cluster}_i) = \sum_j o_j \cdot \big(1 - \text{IoU}(\mathbf{L}_j, \mathbf{L}_{head_i})\big) \tag{12}$$

where $U_{loc}(\cdot)$ denotes the localization uncertainty, $head_i$ denotes the head of cluster i, and $\text{IoU}(\cdot,\cdot)$ denotes the IoU measurement.

Algorithm 1 illustrates the entire non-maximum clustering (NMC) procedure, including the phases of bounding box clustering (lines 2-12) and uncertainty quantification (lines 13-15). Moreover, like NMS, the NMC algorithm was designed as a sequential algorithm. We build on previous work, cluster-NMS [43], which formulates NMS as a matrix operation and benefits from parallel acceleration such as GPU. Since parallel NMC is not the major contribution of our work, we omit it here for simplicity.

Algorithm 1: Non-Maximum Clustering + Uncertainty ($\mathcal{B}$)

```
1  C ← [ ]; sort B in descending order of objectness score o_i ;
2  while len(B) > 0 do
3      curCluster = [B.pop(0)] ;                    /* cluster head */
4      for B in B do
5          iou = IoU(B, curCluster[0]);
6          if iou > iou_threshold then
7              curCluster.append(B) ;               /* cluster member */
8              B.remove(B) ;
9          end
10     end
11     C.append(curCluster);
12 end
13 for cluster in C do
14     U_cls(cluster) and U_loc(cluster) according to (11) (12);
15 end
```

During the closed-loop scale-adaptive inference, we feed the output of the downscaled neural network into the NMC algorithm. It generates bounding box cluster heads with associated classification and localization uncertainties (U_{cls} and U_{loc}). We have two thresholds: λ_{cls} and λ_{loc}. If $U_{cls} > \lambda_{cls}$ or $U_{loc} > \lambda_{loc}$, we mark the corresponding bounding box cluster head as the area of uncertainty and send its full-resolution cropped image into the scale-equivariant full-resolution neural network for further processing.

The question that remains is how to determine uncertainty thresholds, $\vec{\lambda} = [\lambda_{cls}, \lambda_{loc}]$. It turns out to be a relatively simple task because our uncertainty estimation algorithm manifests nearly perfect bimodal properties across different detection models and sub-datasets. All output uncertainties naturally separated into two distinct peaks: one with low uncertainty, containing the background or well-detected objects; the other with high uncertainty, containing the objects that require a second detection pass with a high-resolution network. In practice, we choose a random subset of 500 images to approximate the 2D histogram of classification and localization uncertainty. The heuristic for threshold selection is to place the threshold at the upper boundary of the low-uncertainty peak so that the portion left only contains high-uncertainty areas. To estimate the upper boundary, we follow these three steps: 1) Separate two peaks with histogram bimodal method [26]; 2) extract the center of low-uncertainty peak $\vec{\mu}$ and its standard deviation $\vec{\sigma}$; 3) we set the threshold as $\vec{\lambda} = \vec{\mu} + \alpha \cdot \vec{\sigma}$, where $\alpha = 2$ by default. In practice, when we set α between 2 and 5, mAP drops only 1%, and execution time increases at most by 2%.

4 EVALUATION

We evaluate the performance of ScaleFlow using five sets of experiments: accuracy-speed tradeoff, end-to-end system performance, inference time & overhead analysis, ablation studies of technical components, and anytime inference.

4.1 System Setup

4.1.1 Hardware & Implementation. We implement ScaleFlow and other baselines on four hardware platforms with different computation capabilities, and a webcam is connected to each device.

- NVIDIA Jetson AGX Xavier (Xavier) is an embedded GPU platform equipped with a 512-core Volta GPU and an 8-core ARM 64-bit CPU. Xavier uses the JetPack 5.0 development environment, including CUDA 11.4 and cuDNN 8.3.
- Edge Server with NVIDIA GeForce RTX 3080 (RTX 3080), CUDA 11.0, and cuDNN 8.0.
- Raspberry Pi 4 Model B (Raspberry Pi or RPi 4) is an embedded platform equipped with a quad-core Cortex-A72 64-bit SoC.
- Intel Neural Compute Stick 2 (NCS2) is a plug-and-play AI inference unit equipped with a Myriad Vision Processing Unit (VPU). All neural networks will be first exported into ONNX and then optimized by the OpenVINO Toolkit.
- NexiGo N660P 1080P 60FPS webcam. The highest resolution could be 1920×1080, and the default is 640×480.

4.1.2 Dataset. We use MS *COCO* (Microsoft Common Objects in Context) dataset training and testing [24]. By default, we will rescale each image with its longer dimension equals 640, and its height/width ratio is unchanged.

4.1.3 Baselines. We compare the proposed scale-equivariant scale-adaptive inference (ScaleFlow) with four baselines.

- *YOLOv3/CenterNet-Resolution* is a set of models trained at different resolutions.
- *YOLOv3/CenterNet-Compress* is a set of models with different backbone complexities (# layers or # channels) obtained by model compression or architecture search. .
- *ScaleFlow-IND* is a scale-adaptive baseline by taking two neural networks trained individually at different resolutions.
- *ScaleFlow-DataAug* is also a scale-adaptive baseline. It takes a single neural network trained with data at different resolutions.
- *PConv* is another scale-adaptive baseline using pyramid-convolution [38] to ensure scale-equivariant.
- *RetinaNet/FCOS/EfficientDet-320/640*: These models have an input resolution of either 320 or 640 pixels.

We choose five neural network architectures designed for object detection, YOLOv3 [31], CenterNet [44], RetinaNet [23], and FCOS [37], EfficientDet [36]. Due to space constraints, some experiments only use YOLOv3 and CenterNet as representative anchor-based and anchor-free models.

4.2 ScaleFlow Accuracy-Speed Tradeoff

This section evaluates the tradeoff between mean inference time and mAP score (i.e., the accuracy metric) of two object detection neural networks on different platforms. We generate pseudo streaming using the COCO dataset to fairly compare the accuracy differences between the ScaleFlow and baseline models.

(a) NVIDIA Xavier (b) NVIDIA RTX 3080 (c) Intel NCS2 (d) Raspberry Pi 4

Figure 4: YOLOv3: Tradeoff between inference time and mAP on different platforms.

(a) NVIDIA Xavier (b) NVIDIA RTX 3080 (c) Intel NCS2 (d) Raspberry Pi 4

Figure 5: CenterNet: Tradeoff between inference time and mAP on different platforms.

Model	mAP50	Time	Model	mAP50	Time
RetinaNet			YOLOv3		
640	51.2	10.9ms	640	57.9	11.6ms
320	44.5	4.5ms	320	53.5	3.4ms
PConv	48.1	7.4ms	PConv	56.1	10.5ms
ScaleFlow	**50.8**	**6.8ms**	**ScaleFlow**	**57.4**	**7.9ms**
FCOS			CenterNet		
640	52.2	9.8ms	640	49.5	9.4ms
320	42.1	4.1ms	320	44.5	3.1ms
PConv	45.6	6.9ms	PConv	46.5	9.0ms
ScaleFlow	**51.6**	**6.3ms**	**ScaleFlow**	**49.2**	**6.0ms**

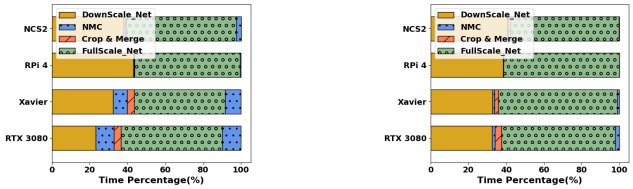
Table 1: ScaleFlow on diverse object detection models.

As shown in Table 1, ScaleFlow consistently achieves good speedup with little mAP degradation. More detailed tradeoffs are shown in Figure 4 and 5, where ScaleFlow consistently outperforms others. If we focus on the best-mAP points (less than one mAP difference) for all models, ScaleFlow attains 1.5× ∼ 1.9× and 1.6× ∼ 2× speed-up compared to the second-best algorithm for YOLOv3 and CenterNet, respectively. YOLOv3/CenterNet-Resolution is the second-best model, implying that tuning input resolution might be a good way to find an efficient model. The variant of PConv can also improve the inference speed to a certain extent. However, the pyramid convolution with strides limits the model complexity at higher resolution, which inevitably degrades the accuracy.

4.3 End-to-End System Performance

We evaluate the execution time and energy consumption of object detection applications on four different platforms. Object detection pipelines continually process video streaming generated by a connected camera. Every system has a small buffer that can hold two frames. When the buffer is full, the incoming frame will be dropped. We operate each object detection pipeline in its best-performance mode (in terms of mAP in Section 4.2). We run each pipeline for 3 minutes on every system five times, measuring per-frame processing time and estimating per-frame energy consumption.

As shown in Table 2, ScaleFlow achieves the best performance with an average of 1.6×, 1.7×, 2.1×, and 2.2× speed-up on RTX 3080, Raspberry Pi, NVIDIA Xavier, and NCS2, respectively. ScaleFlow also consistently achieves the best performance in energy consumption across all platforms, with around 25% - 45% per-frame saving. We also observed that the PConv model had power consumption and inference speed similar to the ScaleFlow model on several devices. Note that the accuracy (i.e., mAP) of the PConv model is substantially lower, because PConv employs convolution with strides at higher resolution, the capabilities of object detection models are naturally limited.



(a) YOLOv3 (b) CenterNet

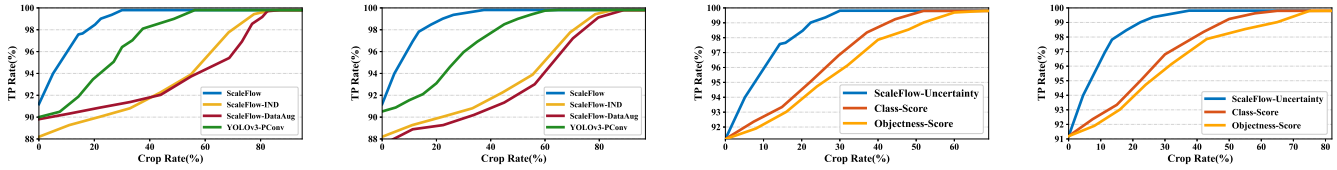
Figure 6: Inference time analysis of ScaleFlow pipeline when achieving the best mAP.

4.4 Inference Time Analysis

This section provides an inference time analysis of the ScaleFlow pipeline when achieving its best mAP performance. As shown in Figure 6, ScaleFlow follows five steps: (1) execute the neural network with the downscaled input (denoted as Downscale_Net), (2) Use NMC (Algorithm 1) to cluster output bounding boxes and estimate their uncertainty (denoted as NMC), (3) pick high-uncertainty areas and merge them at full resolution (denoted as Crop & Merge), (4) execute the neural network with the merged full-resolution input (denoted as FullScale_Net), and (5) use NMC to cluster all output bounding boxes. We clock each of the five stages and provide the percentage of time each step takes from end-to-end latency. To begin with, the computational overhead for Crop&Merge is acceptable (at most 3.3% ∼ 3.7% on Xavier platform). NMC is a time-consuming component of the YOLOv3 architecture on GPU devices. On the RTX 3080 and NVIDIA Xavier, two NMC operations use 19.5% and 15.6% of the inference time, respectively. However, we do not deduct the time consumed by the existing NMS (non-maximal suppression) from NMC. NMS is a well-known time-consuming procedure in anchor-based neural networks like YOLOv3. The NMS in YOLOv3 with a full-resolution input takes 1.3 ms on RTX 3080, while the NMC in ScaleFlow takes 1.6 ms on RTX 3080. Therefore, the extra functions (i.e., clustering and uncertainty quantification) introduced in NMC only consume 0.3 ms and account for about 3.7% of the total inference time. On the other hand, NMC takes much less time (< 3%) in CenterNet architecture because anchor-free design produces orders of magnitude fewer bounding boxes, reducing the complexity of NMS/NMC.

	NVIDIA Xavier		NVIDIA RTX 3080		Rasperry Pi		NCS2	
YOLOv3	$107 \pm 9ms$	$4.2 \pm 0.5J$	$11.6 \pm 2ms$	$8.0 \pm 0.8J$	$7520 \pm 250ms$	$35.4 \pm 3.5J$	$1650 \pm 312ms$	$2.15 \pm 0.32J$
IND	$120 \pm 9.5ms$	$4.8 \pm 0.65J$	$15.04 \pm 3.5ms$	$8.9 \pm 0.92J$	$8125 \pm 1650ms$	$42.5 \pm 5.1J$	$1740 \pm 325ms$	$2.40 \pm 0.41J$
DataAug	$124 \pm 9.52ms$	$4.9 \pm 0.67J$	$15.22 \pm 3.5ms$	$9.3 \pm 0.93J$	$8220 \pm 1658ms$	$45.0 \pm 5.5J$	$1750 \pm 350ms$	$2.42 \pm 0.42J$
PConv	$86 \pm 7.4ms$	$3.5 \pm 0.51J$	$9.88 \pm 2ms$	$7.46 \pm 0.66J$	$6552 \pm 125ms$	$29.2 \pm 4.8J$	$1502 \pm 215ms$	$2.35 \pm 0.35J$
ScaleFlow	$\mathbf{51 \pm 5ms}$	$\mathbf{2.5 \pm 0.42J}$	$\mathbf{7.21 \pm 2ms}$	$\mathbf{6.33 \pm 0.54J}$	$\mathbf{4466 \pm 120ms}$	$\mathbf{23.5 \pm 3.1J}$	$\mathbf{745 \pm 200ms}$	$\mathbf{1.24 \pm 0.21J}$
CenterNet	$135.15 \pm 10.2ms$	$5.8 \pm 0.7J$	$10.42 \pm 2.3ms$	$7.95 \pm 0.58J$	$9252 \pm 184ms$	$44.25 \pm 4.1J$	$10524 \pm 210ms$	$1.92 \pm 0.28J$
IND	$145.25 \pm 11.4ms$	$6.15 \pm 0.8J$	$12.22 \pm 3.51ms$	$8.24 \pm 0.75J$	$10122 \pm 192ms$	$46.12 \pm 5.2J$	$1131 \pm 252ms$	$2.12 \pm 0.3J$
DataAug	$148.12 \pm 11.6ms$	$6.32 \pm 0.82J$	$12.56 \pm 3.84ms$	$8.55 \pm 0.81J$	$10240 \pm 201ms$	$46.55 \pm 5.7J$	$1159 \pm 280ms$	$2.15 \pm 0.33J$
PConv	$106.51 \pm 8.5ms$	$5.12 \pm 0.7J$	$7.25 \pm 2.56ms$	$7.03 \pm 0.5J$	$7685 \pm 140ms$	$41.2 \pm 4.2J$	$720 \pm 172ms$	$1.48 \pm 0.22J$
ScaleFlow	$\mathbf{71.84 \pm 5.5ms}$	$\mathbf{3.39 \pm 0.51J}$	$\mathbf{5.85 \pm 1.5ms}$	$\mathbf{5.92 \pm 0.35J}$	$\mathbf{5095 \pm 122ms}$	$\mathbf{27.51 \pm 3.6J}$	$\mathbf{580 \pm 167ms}$	$\mathbf{1.05 \pm 0.19J}$

Table 2: End-to-end per-frame power consumption and execution time on object detection systems. Bold numbers represent the best results. The upper part is for YOLOv3, and the lower is for CenterNet.



(a) YOLOv3: scale-equivariant mapping **(b) CenterNet: scale-equivariant mapping** **(c) YOLOv3: uncertainty estimation** **(d) CenterNet: uncertainty estimation**

Figure 7: The lowest percentage of cropped area (after performing the crop & merge) combined with the confident detection from the downscaled output for achieving y percent of the true-positive rate.

4.5 Ablation Study

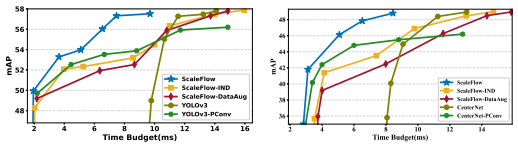
In this section, we validate the efficacy of our proposed technical components in Section 3.1 and 3.2 with ablation studies. There are two groups of experiments, and their simplified versions can be found in Section 2. In the first group, we substitute scale-equivariant mapping with alternative methods of training object detection neural networks at two resolutions while maintaining ScaleFlow-Uncertainty as our uncertainty component. In the second group, we substitute ScaleFlow-Uncertainty with other traditional uncertainty estimating techniques, but we keep scale-equivariant mapping for neural networks at two resolutions. In all these experiments, we use a development set to tune the thresholds. We track the lowest percentage of cropped area (after performing crop & merge) from the downscaled output that can include y percent of ground-truth objects detectable at full resolution (i.e., true-positive rate).

The ablation studies with scale-equivariant mapping are illustrated in Figures 7a and 7b. ScaleFlow with scale-equivariant mapping outperforms the other baselines by a large margin. The ablation studies with ScaleFlow-uncertainty are illustrated in Figures 7c and 7d. Our ScaleFlow-Uncertainty can give both classification and localization uncertainty estimations, saving roughly 30% of cropped area compared to the Class-Score baseline.

4.6 Anytime Inference

Anytime inference requires a model to make a progression of predictions that might be halted at any time. Our proposed scale-adaptive pipeline progressively processes vision data with increasing resolution but decreasing spatial size, which supports anytime inference by nature. Since the performance of anytime inference is consistent across platforms, we chose the RTX 3080 as our reference hardware.

We test all scale-adaptive variants, including ScaleFlow, ScaleFlow-IND, ScaleFlow-DataAug, and PConv, since they naturally support anytime inference. We also include the original YOLOv3/CenterNet model with full-resolution input. As shown in Figure 8, all models, including YOLOv3/CenterNet, offer a certain level of adaptation. Static models (YOLOv3/CenterNet) may



(a) YOLOv3 **(b) CenterNet**

Figure 8: Anytime inference on RTX 3080 with the tradeoff between mAP and time budgets.

"adapt" to varied time budgets due to how we preprocess image data in COCO. We re-scale each image during data preprocessing such that its longer dimension equals 640, and its height/width ratio remains intact. As a result, even with a static model, each input image has a varied size, which results in varying inference times.

ScaleFlow outperforms all other baselines by a large margin. Thanks to scale-adaptive inference, ScaleFlow is prepared to provide inference results after completing the downscaled part. Another finding is that the downscaled neural network in our ScaleFlow model beats the downscaled neural networks in the other three baselines. It indicates that scale-equivariant mapping also acts as a valuable regularization approach for neural networks.

5 CONCLUSION

In this paper, we presented ScaleFlow, which supports anytime inference and dynamically allocates computing resources in a closed-loop scale-adaptive manner. ScaleFlow proposes an efficient scale-equivalent mapping based on wavelet analysis to support robust closed-loop adaptation in scale space. Furthermore, we reinterpreted the object detection output as a Bayesian sampling procedure to enable efficient uncertainty estimation. With comprehensive evaluations, ScaleFlow consistently provides 1.5× to 2.2× speedup and saves around 25% ∼ 45% energy consumption with < 1% accuracy loss, with diverse hardware platforms and network architectures.

ACKNOWLEDGEMENTS

This work is in part supported by the National Science Foundation grants IIS-2107200, CNS-2038658, and CNS-2007153.

REFERENCES

[1] 2022. Intel Neural Compute Stick 2. https://rb.gy/nv8h4m.

[2] 2022. Nvidia Jetson AGX Xavier. https://rb.gy/samxtg.

[3] Ali N Akansu, Richard A Haddad, and Paul A Haddad. 2001. *Multiresolution signal decomposition: transforms, subbands, and wavelets*. Academic press.

[4] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. 2019. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 96–109.

[5] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. 2017. Soft-NMS–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*. 5561–5569.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[7] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5918–5926.

[8] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 155–168.

[9] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[10] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 557–570.

[11] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*. 6569–6578.

[12] Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron Elmore, and Michael Franklin. 2019. Band-limited training and inference for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 1745–1754.

[13] Petko Georgiev, Sourav Bhattacharya, Nicholas D Lane, and Cecilia Mascolo. 2017. Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 1–19.

[14] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[15] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[17] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 269–286.

[18] Borui Jiang, Ruixuan Luo, Jiayuan Mao, Tete Xiao, and Yuning Jiang. 2018. Acquisition of localization confidence for accurate object detection. In *Proceedings of the European conference on computer vision (ECCV)*. 784–799.

[19] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. 2021. Flexible high-resolution object detection on edge devices with tunable latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 559–572.

[20] Florian Kraus and Klaus Dietmayer. 2019. Uncertainty estimation in one-stage object detection. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 53–60.

[21] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 359–376.

[22] Edgar Liberis and Nicholas D Lane. 2023. Differentiable Neural Network Pruning to Enable Smart Applications on Microcontrollers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 4 (2023), 1–19.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[25] Dimity Miller, Lachlan Nicholson, Feras Dayoub, and Niko Sünderhauf. 2018. Dropout sampling for robust object detection in open-set conditions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 3243–3249.

[26] P Daniel Ratna Raju and G Neelima. 2012. Image segmentation by using histogram thresholding. *International Journal of Computer Science Engineering and Technology* 2, 1 (2012), 776–779.

[27] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).

[28] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, 525–542.

[29] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[30] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[31] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[32] Oren Rippel, Jasper Snoek, and Ryan P Adams. 2015. Spectral representations for convolutional neural networks. *Advances in neural information processing systems* 28 (2015).

[33] Kathrin Schacke. 2004. On the kronecker product. *Master's thesis, University of Waterloo* (2004).

[34] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. 2020. Scale-Equivariant Steerable Networks. In *International Conference on Learning Representations*.

[35] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[36] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2020. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10781–10790.

[37] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. 2020. Fcos: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 4 (2020), 1922–1933.

[38] Xinjiang Wang, Shilong Zhang, Zhuoran Yu, Litong Feng, and Wayne Zhang. 2020. Scale-Equalizing Pyramid Convolution for Object Detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[39] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. 2016. Cnnpack: Packing convolutional neural networks in the frequency domain. *Advances in neural information processing systems* 29 (2016).

[40] Daniel Worrall and Max Welling. 2019. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems* 32 (2019).

[41] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 278–291.

[42] Shuochao Yao, Yiran Zhao, Huajie Shao, Aston Zhang, Chao Zhang, Shen Li, and Tarek Abdelzaher. 2018. Rdeepsense: Reliable deep mobile computing models with uncertainty estimations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), 1–26.

[43] Zhaohui Zheng, Ping Wang, Dongwei Ren, Wei Liu, Rongguang Ye, Qinghua Hu, and Wangmeng Zuo. 2021. Enhancing geometric factors in model learning and inference for object detection and instance segmentation. *IEEE Transactions on Cybernetics* (2021).

[44] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. 2019. Objects as points. *arXiv preprint arXiv:1904.07850* (2019).