ThumbNet: One Thumbnail Image Contains All You Need for Recognition

Chen Zhao Bernard Ghanem
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
{chen.zhao, bernard.ghanem}@kaust.edu.sa

Abstract

Although deep convolutional neural networks (CNNs) have achieved great success in the computer vision community, its real-world application is still impeded by its voracious demand of computational resources. Current works mostly seek to compress the network by reducing its parameters or parameter-incurred computation, neglecting the influence of the input image on the system complexity. Based on the fact that input images of a CNN contain much redundant spatial content, we propose in this paper an efficient and unified framework, dubbed as ThumbNet, to simultaneously accelerate and compress CNN models by enabling them to infer on one thumbnail image. We provide three effective strategies to train ThumbNet. In doing so, ThumbNet learns an inference network that performs equally well on small images as the original-input network on large images. With ThumbNet, not only do we obtain the thumbnail-input inference network that can drastically reduce computation and memory requirements, but also we obtain an image downscaler that can generate thumbnail images for generic classification tasks. Extensive experiments show the effectiveness of ThumbNet, and demonstrate that the thumbnail-input inference network learned by ThumbNet can adequately retain the accuracy of the original-input network even when the input images are downscaled 16 times.

1. Introduction

Recent years have witnessed not only the growing performance of deep convolutional neural networks (CNNs), but also their expanding computation and memory costs [3]. Though the intensive computation and gigantic resource requirements are somewhat tolerable in the training phase thanks to the powerful hardware accelerators (*e.g.* GPUs), when deployed in real-world systems, a deep model can easily exceed the computing limit of hardware devices. Mobile phones and tablets, which have constrained power supply and computational capability, are almost intractable to run deep networks in real-time. A cloud service system, which needs to respond to thousands of users, has an even more stringent requirement of computing latency and memory. Therefore, it is of practical significance to accelerate and compress CNNs for test-time deployment.

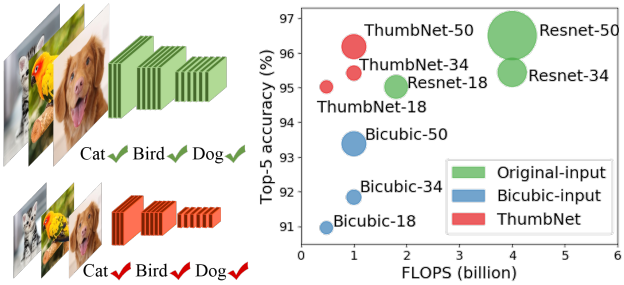


Figure 1: **Same CNN with different input sizes**. We accelerate a CNN by inferring on thumbnail images. Compared to the original-input network shown on the top-left, the thumbnail-input CNN shown on the bottom-left has the same architecture but smaller feature maps in all convolutional layers, hereby tremendously reducing computation and memory consumption, as shown on the right (circle sizes indicate memory consumption). The proposed ThumbNet can well retain the accuracy of the original-input networks, significantly outperforming the bicubic-input networks that input small images downscaled via bicubic interpolation.

Before delving into the question of how to speed up deep networks, let us first analyze what dominates the computational complexity of a CNN. We calculate the total time complexity of convolutional layers as follows [12]:

$$\mathcal{O}\left(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right), \tag{1}$$

where l is the index of a layer and d is the total number of layers; n_{l-1} is the number of input channels and n_l is the number of filters in the l-th layer; s_l^2 is the spatial size of the filters and m_l^2 is the spatial size of the output feature map.

Decreasing any factor in Eq. (1) can lead to reduction of total computation. One way is to sparsify network parameters by filter pruning [20, 11], which by defining some mechanism to prioritize the parameters, sets unimportant ones to zero. However, some researchers claim that these methods usually require sparse BLAS libraries or even spe-

cialized hardware. Hereby, they propose to prune filters as a whole [22, 25]. Another method is to decrease the number of filters by low rank factorization [8, 17, 19, 36]. If a more dramatic change in network structure is permissible, knowledge distillation [14, 31] can do the trick. It generates a new network, which can be narrower (with fewer filters) or shallower (with fewer layers), by transferring hidden information from the original network. Moreover, there are other approaches to lower the convolution overload by means of fast convolution techniques (*e.g*. FFT [27] and the Winograd algorithm [18]), and quantization [11, 34] and binarization [6, 30].

All the above methods attempt to accelerate or compress neural networks from the viewpoint of network parameters, thereby neglecting the significant role that the spatial size of feature maps are playing in the overall complexity. According to Eq. (1), the required computation is diminished as the spatial size of feature maps decreases. Moreover, the memory required to accommodate those feature maps at run-time will also be reduced. Given a CNN architecture, we can simply decrease the spatial size of all feature maps by reducing the spatial size of the input image.

In this paper, we propose to use a thumbnail image, *i.e*. an image of lower spatial resolution than its original-sized counterpart, as test-time network input to accelerate and compress CNNs of any architecture and of any depth and width. This thumbnail-input network can dramatically reduce computation as well as memory requirements, as shown in Fig. 1.

Contributions. (1) We propose a unified framework called ThumbNet to train a thumbnail-input network that can tremendously reduce computation and memory consumption while maintaining the accuracy of the original-input network. **(2)** We present a supervised image downscaler that generates a thumbnail image with good discriminative properties and a natural chromatic look. This downscaler is reliably trained by exploiting *supervised image downscaling*, *distillation-boosted supervision*, and *feature-mapping regularization*. The ThumbNet generated images can replace their original-sized counterparts and be stored for other classification-related tasks, reducing resource requirements in the long run.

2. Related Work

In this section, we give an overview of the most related work to our proposed ThumbNet method in the literature.

2.1. Knowledge Distillation

Knowledge Distillation (KD) [14] was introduced as a model compression framework, which aims to reduce the computational complexity of a deep neural network by transferring knowledge from its original architecture (teacher) to a smaller one (student). The student is penalized according to the discrepancy between the softened versions of the teacher's and student's output logits[1]. It claims that this teacher-student paradigm easily transfers the generalization capability of the teacher network to the student network in that the student not only learns the characteristics of the correct labels but can also benefit from the invisible finer structure in the wrong labels. There are some extensions to this work, *e.g*. using intermediate representations as hints to train a thin-and-deep student network [31], applying it in object detection models [4], and using it to enhance networks resilience to adversarial samples [29]. These works mostly focus on learning a new network architecture. In our paper, we utilize the idea of KD to train the same network architecture with thumbnail images as input.

2.2. Auto-Encoder

An auto-encoder [15] is an unsupervised neural network, which learns a data representation of reduced dimensions by minimizing the difference between input and output. It consists of two parts, an encoder which maps the input to a latent feature, and a decoder which reconstructs the input from the latent feature. Early auto-encoders are mostly composed of fully-connected layers [15, 2]. These days, with the popularity of CNNs, some researchers propose to incorporate convolution to an auto-encoder and design a convolutional auto-encoder [33, 26], which utilizes convolution / pooling to downscale the size of images in the encoder and utilizes deconvolution [35] / unpooling in the decoder to restore the original image size. Though the downscaled images from the encoder are effective for reconstructing the original images, they do not perform well for classification due to lack of supervision. In our work, instead of using convolutional auto-encoder as a downscaler, we incorporate it into ThumbNet as unsupervised pre-training to regularize the classification task.

2.3. Small-Input network

We find one recent work [5] (denoted here as LWAE), which also attempts to accelerate a neural network by using small images as input. It decomposes the original input image into two low-resolution sub-images, one with low frequency which is fed into a standard classification network, and one with high frequency which is fused with features from the low-frequency channel by a lightweight network to obtain the classification results. Compared to LWAE, our ThumbNet is able to achieve higher network accuracy with one single thumbnail-image, resulting in fewer requirements for computation, memory and storage.

[1] 'Logits' is a variable name in Tensorflow, which refers to the output of a neural network before the softmax activation function in the end. We adopt this name in this paper for conciseness.

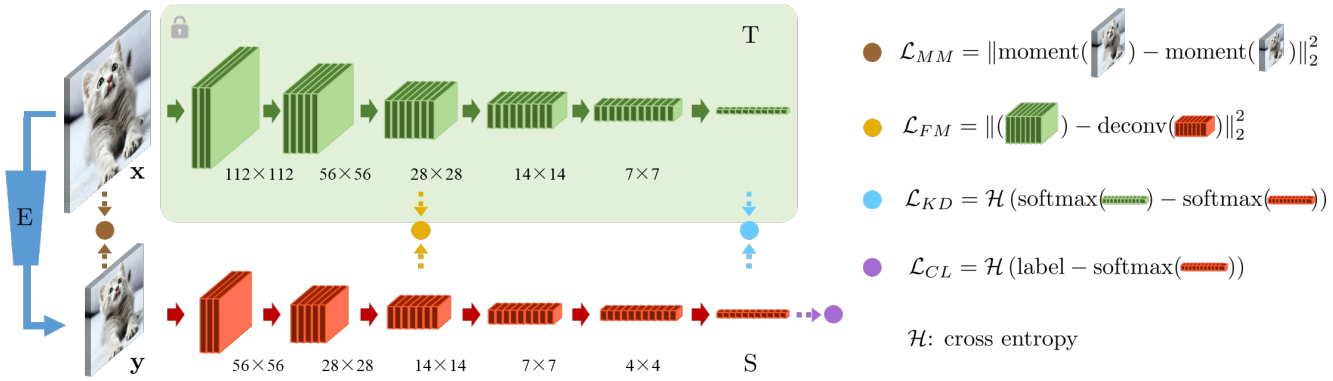


Figure 2: **ThumbNet Architecture**. Green: well-trained network T; red: inference network S; blue: downscaler E. Blocks represent feature maps, solid arrows contain network operations such as convolution, the rectified linear unit (ReLU) [28], pooling, batch normalization [16], etc. The numbers below each feature map are their spatial resolution. Dots represent the four losses, which are moment-matching (MM) loss in brown, feature-mapping (FM) loss in yellow, knowledge-distillation (KD) loss in cyan and classification (CL) loss in purple.

3. ThumbNet Model and Learning

3.1. Network Architecture

We illustrate the architecture of ThumbNet in Fig. 2. The well-trained network T takes an input image of a large size, e.g. 224×224, passes it through stacked convolutional layers and fully-connected layers, and produces K logits, where K is the number of classes. Its well-trained parameters $\mathbf{W}_{\mathrm{T}}$, are not changed during the whole training process of ThumbNet and only provide guidance for the inference network. The inference network S, which takes as input an image of a smaller size, e.g., 112×112. Each layer of S (except for the first fully-connected layer if its size is influenced by the input size as in VGG), has exactly the same shape and size as its corresponding layer in T. Each feature map of S has the same number of channels as its corresponding feature map of T but is smaller in the spatial size. The parameters in S, denoted as $\mathbf{W}_{\mathrm{S}}$, are the main learning objectives of ThumbNet. The downscaler E generates a thumbnail image from the original input image, whose parameters are denoted as $\mathbf{W}_{\mathrm{E}}$.

3.2. Details of Network Design

3.2.1 Supervised Image Downscaling

Traditional image downscaling methods (*e.g.* bilinear and bicubic [21]) do not consider the discriminative capability of the downscaled images, which as a consequence lose critical information for classification. We instead exploit CNN to adaptively extract discriminative information from the original images to tailor to the classification goal.

For computational efficiency and simplicity, our supervised image downscaler E merely comprises two convolutional layers, each with a 5×5 convolutional operation followed by batch normalization and ReLU. In the first layer, there are more output channels than input channels to empower the network to learn more intermediate features, and in the second layer there are exactly 3 output channels to restore the image. The stride size of each layer depends on the required downscaling ratio. Compared to bicubic, this learnable downscaler not only adaptively trains the filters, but also incorporates non-linear operations and high-dimensional feature projection. By denoting $\mathcal{E}$ as all the nested operations in our downscaler, we obtain a small image $\mathbf{y}$ from the original image $\mathbf{x}$ via the following:

$$\mathbf{y} = \mathcal{E}\left(\mathbf{x}; \mathbf{W}_{\mathrm{E}}\right). \tag{2}$$

A significant consideration in designing this downscaler is that the generated small image should remain visually pleasant and recognizable, *e.g.* the information in the color channels should not be destroyed or misaligned. If we think of natural images as pixels in each color channel conforming to some distribution, then the generated small image should conform to the same distribution with similar moments. Hereby, we devise a moment-matching (MM) loss calculated as follows:

$$\mathcal{L}_{MM}\left(\mathbf{W}_{\mathrm{E}}\right) = \frac{1}{3}\left\|\mu\left(\mathbf{x}\right) - \mu\left(\mathbf{y}\right)\right\|_2^2 + \lambda\frac{1}{3}\left\|\sigma\left(\mathbf{x}\right) - \sigma\left(\mathbf{y}\right)\right\|_2^2, \tag{3}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ compute the first and second moments of an image respectively with respect to its each color channel, and λ is a tunable parameter that balances the two moments. This MM loss encourages that the mean and the variance (loosely approximating the distribution) in each color channel of the generated image stay close to those of the original image, which has been used for other application in the literature such as deep generative model learning [23] and style transfer [24].

Please note that this downscaler is not trained independently with merely the MM loss, but incorporated into the

whole architecture of ThumbNet and trained together with other components and other losses. This includes the classification loss, which provides supervision to the image downscaling process guiding it to generate a small image that is discriminative for accurate classification. Hence, we attribute E a supervised image downscaler. Once trained, this downscaler can be used not only for generating small images as input of the inference network, but also for other classification-related tasks.

3.2.2 Distillation-Boosted Supervision

A straightforward way to train the network is to minimize the classification (CL) loss defined as follows:

$$\mathcal{L}_{CL}\left(\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}}\right) = \mathcal{H}\left(\mathbf{b}, \mathcal{S}\left(\mathbf{y}; \mathbf{W}_{\mathrm{S}}\right)\right), \tag{4}$$

where $\mathbf{y}$ is calculated as in Eq. (2), $\mathbf{b}$ indicates the ground-truth labels, $\mathcal{S}\left(\cdot\right)$ denotes all the nested functions in the inference network S, and $\mathcal{H}$ refers to cross entropy. This loss seeks to match the predicted label with the ground-truth label and is a typical cost function for supervised learning. The problem is that the information embedded in the well-trained model is not exploited. To address this shortcoming, we propose to distill the learned knowledge in the original network and transfer it to the inference network. Therefore, apart from the CL loss, we also enforce the computed probabilities of each class in the inference network to match those in the well-trained network.

Let $\mathcal{S}_0\left(\cdot\right)$ and $\mathcal{T}_0\left(\cdot\right)$ denote the deep nested functions before softmax in the inference network and the well-trained network, respectively. Then, their logits are calculated as:

$$\mathbf{a}_{\mathrm{S}} = \mathcal{S}_0\left(\mathbf{y}; \mathbf{W}_{\mathrm{S}}\right), \ \mathbf{a}_{\mathrm{T}} = \mathcal{T}_0\left(\mathbf{x}\right). \tag{5}$$

Following [14], we define the knowledge-distillation (KD) loss as the cross entropy between the two softened probabilities:

$$\mathcal{L}_{KD}\left(\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}}\right) = \mathcal{H}\left(\mathrm{softmax}\left(\frac{\mathbf{a}_S}{\tau}\right), \mathrm{softmax}\left(\frac{\mathbf{a}_T}{\tau}\right)\right), \tag{6}$$

where τ is the temperature to soften the class probabilities and is usually greater than 1. With the aid of this KD loss, the supervised training process of ThumbNet can benefit from the well-trained model to learn finer discriminative structures, so we call it distillation-boosted supervision.

3.2.3 Feature-Mapping Regularization

It is widely observed that unsupervised pre-training, *e.g.* using an auto-encoder, can help with supervised learning tasks as a form of regularization [9]. Inspired by this, we design the feature-mapping (FM) regularization to pre-train ThumbNet.

In Fig. 2, with the FM loss (the yellow dot) as a dividing line, ThumbNet is partitioned into two segments. In order to give a clearer sense of the rationale, we re-illustrate the left segment of ThumbNet along with the FM loss from a different point of view in Fig. 3 (note that the MM loss is left out and the deconvolution in the FM loss is unrolled for the sake of clarity). We can see that it is analogous to an auto-encoder, which is trained by minimizing the difference between the pre-processed input and the output.

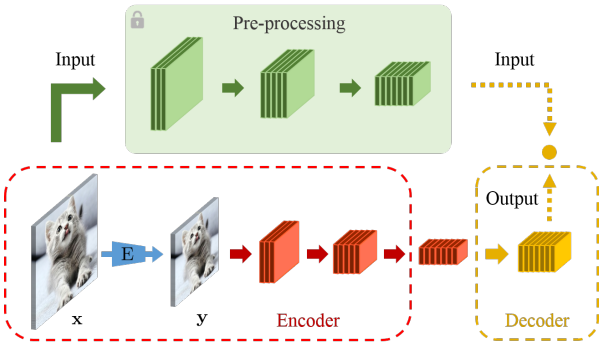


Figure 3: **Feature-mapping regularization**. This re-illustrates the left segment of ThumbNet along with the FM loss, which is essentially an auto-encoder. The layers in the red dashed box constitute the encoder, the layers in the yellow dashed box constitute the decoder, and the feature map in the middle is learned latent representation. The shaded area can be viewed as a pre-processing for the input. The yellow solid arrow represents deconvolutional layers and the yellow block is the upscaled feature map.

In the decoder, each deconvolutional layer has a stride size 2 and the number of layers is determined by the downscaling ratio. The decoder does not change the number of channels but upscales the feature map in S to match the spatial size of the corresponding feature map in T. We compute their mean square error as the FM loss:

$$\mathcal{L}_{FM}\left(\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}_l}, \mathbf{W}_{\mathrm{D}}\right) = \frac{1}{2N}\left\|\mathcal{T}_l\left(\mathbf{x}\right) - \mathcal{D}\left(\mathcal{S}_l\left(\mathbf{y}; \mathbf{W}_{\mathrm{S}_l}\right); \mathbf{W}_{\mathrm{D}}\right)\right\|_2^2, \tag{7}$$

where N is the product of all dimensions of the intermediate feature map in T. $\mathcal{T}_l$ represents the nested functions in the left segment of the original network, $\mathcal{S}_l$ represents the nested functions in the left segment of the inference network, $\mathcal{D}$ represents operations in the deconvolutional layers, and $\mathbf{W}_{\mathrm{S}_l}$ and $\mathbf{W}_{\mathrm{D}}$ refer to the parameters in the left segment of S and in the deconvolutional layers, respectively.

3.3. Training Details

In order to take advantage of the feature-mapping regularization to pre-train ThumbNet, we use a two-phase training strategy, which is summarized in Algorithm 1.

In Algorithm 1, the $\mathcal{R}$ terms refer to l_2 regularization for the corresponding parameters, and θ, α, β are the tradeoff weights. The first phase is shown in Line 2, in which we perform an unsupervised training by minimizing the MM

Algorithm 1 **Two-Phase Training of ThumbNet**

The well trained parameters $\mathbf{W}_{\mathrm{T}}$ of the original network are provided as input to the algorithm. All the trainable parameters are initialized as random values which are denoted by $\mathbf{W}_{\mathrm{S}}^{0}$, $\mathbf{W}_{\mathrm{E}}^{0}$, $\mathbf{W}_{\mathrm{D}}^{0}$. The output values of the first phase are denoted using the superscript 1, which are then input into the second phase as initialization.

1: **Input:** $\mathbf{W}_{\mathrm{T}}$, $\mathbf{W}_{\mathrm{S}}^{0}$, $\mathbf{W}_{\mathrm{E}}^{0}$, $\mathbf{W}_{\mathrm{D}}^{0}$

2: $\mathbf{W}_{\mathrm{E}}^{1}, \mathbf{W}_{\mathrm{S}_l}^{1}, \mathbf{W}_{\mathrm{D}}^{1} \leftarrow \underset{\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}_l}, \mathbf{W}_{\mathrm{D}}}{\arg\min} \mathcal{L}_{MM}\left(\mathbf{W}_{\mathrm{E}}\right) + \alpha \mathcal{L}_{FM}\left(\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}_l}, \mathbf{W}_{\mathrm{D}}\right) + \frac{1}{2}\theta \mathcal{R}_{\mathrm{E},\mathrm{S}_l,\mathrm{D}}$

3: $\mathbf{W}_{\mathrm{E}}^{*}, \mathbf{W}_{\mathrm{S}}^{*} \leftarrow \underset{\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}}}{\arg\min} \mathcal{L}_{CL}\left(\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}}\right) + \beta \mathcal{L}_{KD}\left(\mathbf{W}_{\mathrm{E}}, \mathbf{W}_{\mathrm{S}}\right) + \frac{1}{2}\theta \mathcal{R}_{\mathrm{E},\mathrm{S}}$

4: **Output:** $\mathbf{W}_{\mathrm{E}}^{*}$, $\mathbf{W}_{\mathrm{S}}^{*}$

loss and the FM loss. By doing this, we obtain the parameters $\mathbf{W}_{\mathrm{E}}^{1}$ and $\mathbf{W}_{\mathrm{S}_l}^{1}$, which provide initialization for the second phase. The second phase is shown in Line 3, which performs the distillation-boosted supervised learning by minimizing the KD loss and CL loss. In this phase, we train the whole ThumbNet end-to-end to learn the optimal values for the parameters $\mathbf{W}_{\mathrm{E}}$ and $\mathbf{W}_{\mathrm{S}}$. Note that for the parameters $\mathbf{W}_{\mathrm{E}}$ and $\mathbf{W}_{\mathrm{S}_l}$, which are already trained in the first phase, we only finetune them with a small learning rate in the second phase. In contrast, we use a relatively large learning rate for the untrained parameters $\mathbf{W}_{\mathrm{S}_r}$ in the right segment.

We set the hyper-parameters in ThumbNet as follows. We use a starting learning rate 0.1, and divide it by 10 when the loss plateaus. We use a momentum 0.9 for the optimizer and the weight decay θ is 0.0001. The parameters α and β in Algorithm 1 are 1.0 and 0.5, respectively; τ in Eq. (6) is 2; λ in Eq. (3) is 0.1. In the 2nd phase, the learning rate of the pre-trained parameters $\mathbf{W}_{\mathrm{E}}$ and $\mathbf{W}_{\mathrm{S}_l}$ is set to 0.01 times that of the other parameters, meaning their learning rate starts from 0.001 and is decreased in the same fashion.

Table 1: **Configuration of different methods**. (a) is the baseline, (b) does not use any of the three techniques, (c)-(e) contain one or two, and (f) consists of all of them.

Methods \ Techniques	SD	KD	FM
(a) Original / Direct	–	–	–
(b) Bicubic downscaler	×	×	×
(c) Supervised downscaler	✓	×	×
(d) Bicubic + distillation	×	✓	×
(e) Supervised + distillation	✓	✓	×
(f) **ThumbNet**	✓	✓	✓

4. Experiments

In this section, we demonstrate the performance of network S trained via ThumbNet with respect to classification accuracy and resource requirements. We also provide experiments to verify that the learned downscaler has generic applicability to other classification-related tasks as well.

4.1. Effective and Efficient Inference

4.1.1 Baseline and Comparative Methods

In order to demonstrate the performance of our ThumbNet, we implement six different comparative methods for training a backbone network (*e.g.* Resnet-50), which are introduced in the following.

(a) Original / Direct. 'Original' refers to the network model trained on the original-sized images. The testing performance of the model on an original-sized image is the upperbound baseline of all the comparative methods. For networks like Resnet, which use global average pooling instead of fully-connected layers at the end, the 'Original' model can be directly used to test on a small image without altering the network structure. We refer to the case of directly using the 'Original' model for inference on small images without retraining as 'Direct', which is the lowerbound baseline of all the methods.

(b) Bicubic downscaler. This trains the network from scratch on images that are downscaled with the bicubic method.

(c) Supervised downscaler. This trains the network from scratch on small images that are downscaled with the supervised downscaler in ThumbNet. The downscaler and the network are trained end-to-end in one pass based on the MM loss and the CL loss.

(d) Bicubic + distillation. This trains the networks on bicubic-downscaled small images with the aid of distillation from the 'Original' models. The network is trained based on the KD loss and the CL loss.

(e) Supervised + distillation. This trains the networks on supervised-downscaled small images with the aid of distillation from the 'Original' models. The supervised downscaler and the network are trained end-to-end based on the MM loss, the KD loss and the CL loss.

(f) ThumbNet. This is our proposed ThumbNet, which is trained on supervised-downscaled small images with the aid of distillation from the 'Original' models as well as feature mapping regularization. It is trained based on the four losses as described in Section 3.3.

In Table 1, we list the configuration of each method

Table 2: **Error rates (%) for object recognition on Imagenet**. The four experiments on the left use the dataset Imagenet100. By comparing (c) to (b) or comparing (e) to (d), we can see the benefits of supervised image downscaling. By comparing (e) to (c), we can see the benefits of distillation-boosted supervision. By comparing (f) to (e), we can see that the benefits of feature-mapping regularization.

	VGG-11		Resnet-18		Resnet-34		Resnet-50		Resnet-18 / ImagenetFull	
Methods	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
(a) Original	13.64	4.36	17.54	4.98	15.06	4.56	12.72	3.50	29.71	10.46
(a) Direct	/	/	37.26	16.94	34.48	14.32	29.42	11.80	50.74	26.15
(b) Bicubic downscaler	18.20	6.60	23.98	9.04	22.42	8.16	19.04	6.62	36.18	15.18
(c) Supervised downscaler	16.14	5.10	19.70	7.06	18.44	6.16	16.84	5.66	34.87	13.98
(d) Bicubic + distillation	17.14	5.84	20.34	6.64	18.28	5.84	14.96	4.48	34.76	14.02
(e) Supervised + distillation	17.00	5.78	17.44	5.16	15.46	4.62	15.12	3.96	33.02	12.54
(f) **ThumbNet**	**15.72**	**4.96**	**17.32**	**4.98**	**15.30**	**4.58**	**13.96**	**3.82**	**32.26**	**12.13**

with respect to the three techniques used in ThumbNet: supervised image downscaling (SD), knowledge-distillation boosted supervision (KD) and feature mapping regularization (FM). The hyper-parameters in all the methods are the same, as specified in Section 3.3.

4.1.2 Object Recognition on ImageNet Dataset

We evaluate the performance of our proposed ThumbNet on the task of object recognition with the benchmark dataset ILSVRC 2012 [7], which consists of over one million training images drawn from 1000 categories. Besides using the full dataset (referred to as ImagenetFull), we also form a smaller new dataset by randomly selecting 100 categories from ILSVRC 2012 and refer to it as Imagenet100 (the categories in Imagenet100 are given in the **supplementary material**). With Imagenet100, we can efficiently compare all the methods on a variety of backbone networks. In addition, by partitioning ILSVRC 2012 into two parts (the other part with 900 categories is referred to as Imagenet900), we can also evaluate the downsampler on unseen data categories (see Section 4.3 for details). For backbone networks, we consider various architectures (ResNet [13] and VGG [32]) and various depths (from 11 layers to 50 layers).

In Table 2, we demonstrate the performance of all the methods with four different backbone networks in terms of top-1 and top-5 error rates on the validation data. The input image size of 'Original' is 224×224 and the input image size of the other methods is 112×112, meaning that in this experiment, the image downscaling ratio is $4 : 1$. Thus, compared to 'Original', our ThumbNet only uses $1/4$ FLOPS and GPU memory, which will be detailed in Section 4.1.4, and it also preserves the accuracy of the original models. 'Direct' is a baseline of inference on small images, compared to which our ThumbNet improves by large margins for all the networks. Moreover, by comparing different pairs of methods, we can obviously observe the contribution of each technique.

Table 3: **Error rates (%) for scene recognition on Places**. ThumbNet retains the accuracy of the original-input network when downscaling the image 16 times.

	Resnet-18		VGG-11	
Methods	Top-1	Top-5	Top-1	Top-5
(a) Original	21.11	3.28	19.75	3.61
(a) Direct	66.94	33.97	/	/
(b) Bicubic downscaler	32.08	7.83	27.83	10.17
(c) Supervised downscaler	25.94	5.31	24.28	6.58
(d) Bicubic+distillation	26.00	4.47	22.69	4.92
(e) Supervised+distillation	24.33	3.94	**21.31**	3.94
(f) **ThumbNet**	**22.78**	**3.69**	21.58	**3.72**

4.1.3 Scene Recognition on Places Dataset

We also apply our proposed ThumbNet to the task of scene recognition using the benchmark dataset Places365-Standard [37]. This dataset consists of 1.8 million training images from 365 scene categories. We randomly select 36 categories from Places365-Standard as our new dataset Places36 (the chosen categories are given in the **supplementary material**).

In Table 3, we report the error rates on the Places36 validation dataset using two backbone networks Resnet-18 and VGG-11. The input image size of 'Original' is 224×224 and the input image size of the other methods is 56×56, meaning that in this experiment, the image downscaling ratio is $16 : 1$. Thus, compared to 'Original', our ThumbNet only uses $1/16$ FLOPS and GPU memory, which will be detailed in Section 4.1.4. In terms of recognition accuracy, ThumbNet nearly preserves the accuracy of the original models, where the Top-5 accuracy drops only 0.41% for Resnet-18 and only 0.11% for VGG-11. Compared to 'Direct', our ThumbNet improves by 44.16% for Top-1 accuracy and 30.28% for Top-5 accuracy.

4.1.4 Resource Consumption

To evaluate the test-time resource consumption of the networks, we measure their FLOPS computation and the memory consumption for their feature maps. Fig. 4 plots the number of FLOPS required by each method to classify one

image for object recognition on Imagenet100 and scene recognition on Places36 with the two backbone networks Resnet-18 and VGG-11. For the task of object recognition, in which the images are downscaled 4 times, the small-input networks use only $1/4$ FLOPS compared to the original model. For the task of scene recognition, in which the images are downscaled 16 times, the small-input networks use only $1/16$ FLOPS compared to the original model. Similar to the computation reduction, after downscaling the images by 4, the memory occupation is about $1/4$ of the original model and after downscaling by 16, the memory occupation is about $1/16$ of the original model.

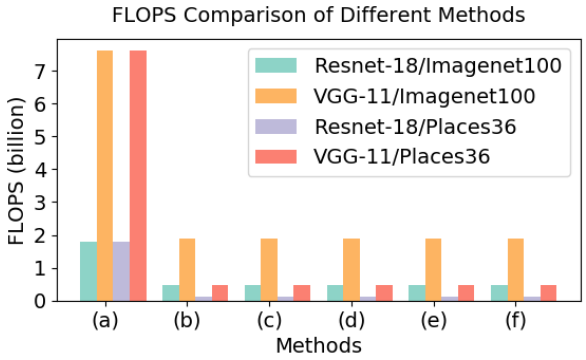


Figure 4: **Computation comparison of different methods**. For object recognition on Imagenet100, ThumbNet uses $1/4$ FLOPS compared to 'Original'. For scene recognition on Places36, ThumbNet uses only $1/16$ FLOPS compared to 'Original'.

4.2. Comparison to State-of-the-Art Methods

We compare our ThumbNet to LWAE [5] with respect to classification accuracy and resource consumption. Considering that the trained models of LWAE provided by the authors use different backbone networks on different datasets from ours, we re-implement LWAE in Tensorflow [1] (the same as our ThumbNet) with the same backbone networks on the same datasets as ours for fair comparison. We follow the instructions in the paper for setting the hyper-parameters and training LWAE. To evaluate both methods, we test their inference accuracy in terms of top-1 and top-5 error rates, and their inference efficiency in terms of number of FLOPS, number of network parameters, memory consumption of feature maps, and storage requirements for input images.

Table 4 lists the results of testing a batch of 224×224 color images using the two methods as well as the benchmark networks for the four tasks: object recognition on Imagenet100 with VGG-11 and Resnet-18, scene recognition on Places36 with VGG-11 and Resnet-18. For the object recognition tasks, the images are downscaled by 4 in LWAE and ThumbNet; and for the scene recognition tasks, the images are downscaled by 16 in LWAE and ThumbNet. The batch size in these experiments is set to 32.

Seen from Table 4, ThumbNet has an obvious advantage over LWAE in terms of efficiency owing to its one single input image and one single network for inference. ThumbNet only computes one network, whereas LWAE has to compute an extra branch for fusing the high-frequency image. Therefore, when downscaling an image by the same ratio, ThumbNet uses fewer FLOPS, has fewer network parameters, requires less memory for the intermediate feature maps, and stores only one small image (as compared to two images for LWAE). Regarding classification accuracy, ThumbNet has lower top-1 and top-5 errors in the first two tasks. For the 3$^{\text{rd}}$ task, ThumbNet has a much lower top-5 error and roughly the same top-1 error as LWAE. For the 4$^{\text{th}}$ task, LWAE is slightly better than ThumbNet at the cost of higher computational complexity and memory usage.

4.3. Evaluation of the Supervised Downscaler

4.3.1 Visual Comparison of Downscaled Images

In Fig. 5, we show an example of the generated thumbnail images by our downscalers as compared to other different methods for the task of object recognition on Imagenet100 with Resnet-18. We can see that the images of 'Supervised downscaler', 'Supervised + distillation', ThumbNet, and 'W/o MM' (ThumbNet without the MM loss) have noticeable edges compared to the 'Bicubic' one. This is because these downscalers are trained in a supervised way with the classification loss being considered. The retained edge information is helpful for making discriminative decisions. Also, note that the thumbnail image of ThumbNet is natural in color owing to the MM loss, whereas the image of 'W/o MM' is obviously yellowish because the color channels are more easily messed up when the MM loss is not utilized. The top-1 error rates of ThumbNet and 'W/o MM' are 17.32% and 17.90%, respectively, and their top-5 error rates are 4.98% and 5.26%, respectively. This shows that adding the MM loss does not deteriorate the network accuracy but produces more pleasant small images, which contain generic information that also benefits other tasks.

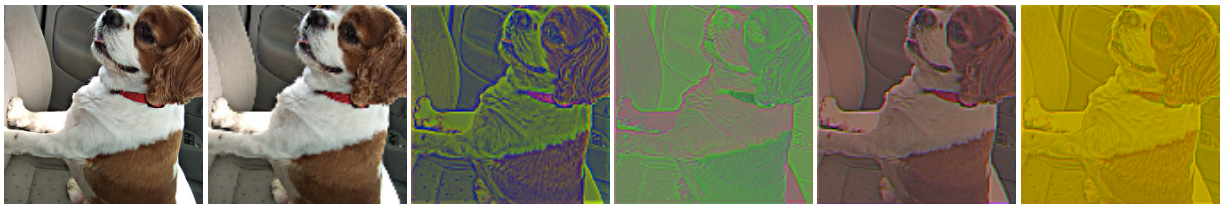
4.3.2 Does the Supervised Downscaler Generalize?

In order to verify that the ThumbNet downscaler is also useful apart from serving the specific inference network, we consider three different scenarios for applying our learned downscaler. Suppose that it is trained on the dataset A with the backbone network F. We have a new dataset A_{new} and a different network F_{new}. The three scenarios are as follows: (1) downscaling A to generate small images to train F_{new}; (2) downscaling A_{new} to generate small images to train F; and (3) downscaling A_{new} to generate small images to train F_{new}.

Table 5 reports the results of these scenarios using the downscaler learned from the object recognition task with Resnet-18, which downscales an image by 4. In this case,

Table 4: **Accuracy and efficiency comparison with state-of-the-art**. B and M stand for billion and million respectively, and MB is short for MegaBytes. The feature maps in all networks are represented by 32-bit floating-point numbers, and the input images are represented by three 8-bit integers. The 'FLOPS' columns show the total number of FLOPS in all convolutional and fully-connected layers.

Tasks	Metrics / Methods	Error rates		FLOPS		Parameters		Feature Memory		Image Storage	
		Top-1	Top-5	# (B)	↓ rate	# (M)	↓ rate	Size (MB)	↓ rate	Size (MB)	↓ rate
Imagenet /VGG	VGG-orig	13.64%	4.36%	243.37	1×	129.18	1×	2118.36	1×	4.82	1×
	LWAE [5]	17.98%	6.42%	65.61	3.71×	67.78	1.91×	668.94	3.17×	2.41	2×
	ThumbNet	**15.72%**	**4.96%**	**61.04**	**3.99**×	**45.29**	**2.85**×	**530.98**	**3.99**×	**1.20**	**4**×
Imagenet /Resnet	Resnet-orig	17.54%	4.98%	58.04	1×	11.22	1×	658.40	1×	4.82	1×
	LWAE [5]	21.06%	6.90%	17.37	3.34×	11.94	0.94×	212.23	3.10×	2.41	2×
	ThumbNet	**17.32%**	**4.98%**	**15.52**	**3.74**×	**11.22**	**1**×	**166.88**	**3.95**×	**1.20**	**4**×
Places /VGG	VGG-orig	19.75%	3.61%	243.36	1×	128.91	1×	2118.33	1×	4.82	1×
	LWAE [5]	**21.53%**	4.58%	16.58	14.67×	56.50	2.28×	168.95	12.54×	0.60	8×
	ThumbNet	21.58%	**3.72%**	**15.09**	**16.13**×	**28.25**	**4.56**×	**133.17**	**15.91**×	**0.30**	**16**×
Places /Resnet	Resnet-orig	21.11%	3.28%	58.03	1×	11.19	1×	658.38	1×	4.82	1×
	LWAE [5]	**22.39%**	**3.06%**	4.48	12.96×	11.91	0.94×	54.51	12.08×	0.60	8×
	ThumbNet	22.78%	3.69%	**4.13**	**14.05**×	**11.19**	**1**×	**42.88**	**15.35**×	**0.30**	**16**×



Original Bicubic Super. downscaler Super. + dist. ThumbNet W/o MM

Figure 5: **Visual comparison of the thumbnail images generated by different downscalers**. 'Original' is of size 224×224; the others are 112×112. 'W/o MM' means ThumbNet without the MM loss.

the dataset A is Imagenet100; the network F is Resnet-18. We use Imagenet900 and Caltech256 [10] as the new dataset A_{new}, and VGG-11 as the new network F_{new}. The first two columns correspond to Scenario (1); the 3rd and 4th columns correspond to Scenario (2); the last column corresponds to Scenario (3), where we use the downscaler to generate thumbnail images for the dataset Caltech256, and use these thumbnail images to train VGG-11.

Table 5: **Performance of the ThumbNet downscaler on different networks and datasets**.

Networks/ Datasets	VGG/ Imagenet100		Resnet/ Imagenet900		VGG/ Caltech256	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Original	13.64	4.36	28.80	10.17	30.68	18.56
Bicubic	18.20	6.60	35.52	14.56	32.80	19.63
ThumbNet	**16.74**	**6.42**	**33.54**	**13.22**	**32.04**	**18.56**

The 1st row shows the performance of the respective networks trained on the original-sized images, while the 2nd row shows their performance when trained on the small images downscaled by bicubic interpolation. The 3rd row shows the performance of the networks trained on the small images downscaled by our ThumbNet downscaler. We can see that the 3rd row obviously outperforms the 2nd row in all scenarios, indicating that our supervised downscaler tends to generalize to other datasets and other network architectures. In fact, it is very promising to see that in Scenario (3) when both the dataset and the network are new to the downscaler, it can still bring about significant gains compared to the bicubic naive downscaler, leading to the same top-5 error rate as the original network.

5. Conclusions

In this paper, we propose a unified framework ThumbNet to tackle the problem of accelerating run-time deep convolutional network from a novel perspective: the input image. Since reducing the input image size lowers the computation and memory costs of a CNN, we seek an inference network that can retain original accuracy when tested on one thumbnail image. Experimental results show that, with our ThumbNet, we are able to learn an inference network that dramatically reduces resource consumption without compromising recognition accuracy. Moreover, we have a supervised downscaler as a side product, which can be utilized for generic classification purposes, thus, generalizing to datasets and network architectures that it was not exposed to in training. This work can be used in addition to other methods for network acceleration or compression to speed up inference without incurring additional overheads.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 7

[2] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988. 2

[3] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016. 1

[4] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker. Learning efficient object detection models with knowledge distillation. In *Advances in Neural Information Processing Systems*, pages 742–751, 2017. 2

[5] T. Chen, L. Lin, W. Zuo, X. Luo, and L. Zhang. Learning a wavelet-like auto-encoder to accelerate deep neural networks. In *AAAI*, 2018. 2, 7, 8

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 2

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6

[8] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014. 2

[9] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010. 4

[10] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007. 8

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1, 2

[12] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 5353–5360. IEEE, 2015. 1

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[14] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2, 4

[15] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. 2

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015. 3

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 2

[18] A. Lavin and S. Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016. 2

[19] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 2

[20] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 1

[21] T. M. Lehmann, C. Gonner, and K. Spitzer. Survey: Interpolation methods in medical image processing. *IEEE transactions on medical imaging*, 18(11):1049–1075, 1999. 3

[22] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 2

[23] Y. Li, K. Swersky, and R. Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727, 2015. 3

[24] Y. Li, N. Wang, J. Liu, and X. Hou. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036*, 2017. 3

[25] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017. 2

[26] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011. 2

[27] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013. 2

[28] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 3

[29] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016. 2

[30] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2

[31] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 2

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[33] V. Turchenko, E. Chalmers, and A. Luczak. A deep convolutional auto-encoder with pooling-unpooling layers in caffe. *arXiv preprint arXiv:1701.04949*, 2017. 2

[34] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Pro-*

ceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4820–4828, 2016. 2

[35] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010. 2

[36] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016. 2

[37] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 6