CVNets: High Performance Library for Computer Vision

Sachin Mehta∗
Apple
USA
Farzad Abdolhosseini
Apple
USA
Mohammad Rastegari
Apple
USA

ABSTRACT
We introduce CVNets, a high-performance open-source library for training deep neural networks for visual recognition tasks, including classification, detection, and segmentation. CVNets supports image and video understanding tools, including data loading, data transformations, novel data sampling methods, and implementations of several standard networks with similar or better performance than previous studies. Our source code is available at: https://github.com/apple/ml-cvnets.

CCS CONCEPTS
• Computing methodologies → Computer vision tasks.

KEYWORDS
Computer vision, deep learning, image and video understanding

1 INTRODUCTION
With the rise of deep learning, significant progress has been made in visual understanding tasks, including novel light- and heavy-weight architectures, dedicated hardware and software stacks, advanced data augmentation methods, and better training recipes. There exist several popular libraries that provide implementations for different tasks and input modalities, including Torchvision [12], TensorflowLite [4], timm [16], and PyTorchVideo [3]. Many of these libraries are modular and are designed around a particular task and input modality, and provide implementations and pre-trained weights of different networks with varying performance. However, reproducibility varies across these libraries. For example, Torchvision library uses advanced training recipes (e.g., better augmentation) to achieve the same performance for training MobileNetV3 on the ImageNet dataset [2] as TensorflowLite with simple training recipes.

We introduce CVNets, a PyTorch-based deep learning library for training computer vision models with higher performance. With CVNets, we enable researchers and practitioners in academia and industry to train either novel or existing deep learning architectures with high-performance across different tasks and input modalities. CVNets is a modular and flexible framework that aims to train deep neural networks faster with simple or advanced training recipes. Simple recipes are useful for research in resource-constrained environments as they train models for fewer epochs with basic data augmentation (random resized crop and flipping) as compared to advanced training recipes, which trains model for $2 \times 4 \times$ longer with advanced augmentation methods (e.g., CutMix and MixUp).

With simple recipes (similar to the ones in original publications) and variable batch sampler (Section 3.1), CVNets improves the performance of ResNet-101 significantly (Figure 1) on the ImageNet dataset while for advanced training recipes with the same batch size and number of epochs, it delivers similar performance to previous methods while requiring $1.3 \times$ fewer optimization updates.

Figure 1: CVNets can be used to improve the performance of different deep neural networks on the ImageNet dataset [14] significantly with simple training recipes (e.g., random resized cropping and horizontal flipping). The official MobileNetV1 [7] and ResNet [6] results are from TensorflowLite [4] and Torchvision [12], respectively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Sachin Mehta, Farzad Abdolhosseini, and Mohammad Rastegari. 2022. CVNets: High Performance Library for Computer Vision. In Proceedings of the 30th ACM International Conference on Multimedia (MM ’22), Oct. 10–14, 2022, Lisboa, Portugal. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3503161.3548540

∗Project lead and main contributor
CNETS LIBRARY DESIGN

CNets follows the design principles below:

Modularity. CNets provides independent components; allowing users to plug-and-play different components across different visual recognition tasks for both research and production use cases. CNets implement different components, including datasets and models for different tasks and input modalities, independently. For example, different classification backbones (e.g., ResNet-50) trained in CNets can be seamlessly integrated with object detection (e.g., SSD) or semantic segmentation (e.g., DeepLabv3) pipelines for studying the generic nature of an architecture.

Flexibility. With CNets, we would like to enable new use cases in research as well as production. We designed CNets such that new components (e.g., models, datasets, loss functions, data samplers, and optimizers) can be integrated easily. We achieve this by registering each component. As an example, ADE20k dataset for the task of segmentation is registered in CNets as:

```python
@register_dataset(name="ade20k", task="segmentation")
```

To use this dataset for training, one can use dataset.name and dataset.category as command line arguments.

Reproducibility. CNets provide reproducible implementations of standard models for different computer vision tasks. Each model is benchmarked against the performance reported in original publications and previous best reproduction studies. The pre-trained weights of each model are released online to enable future research.

Compatibility. CNets is compatible with hardware accelerated frameworks (e.g., CoreML) and domain-specific libraries (e.g., PyTorchVideo). The models from domain-specific libraries can be easily consumed in the CNets, as shown in Listing 1; reducing researchers overhead in implementing new components or sub-modules in CNets.

3 CNETS LIBRARY COMPONENTS

CNets include efficient data sampling (Section 3.1) and training methods (Section 3.2), in addition to standard components (e.g., optimizers; Section 3.3), which are discussed below.

3.1 Data Samplers

CNets offer data samplers with three sampling strategies: (1) single-scale with fixed batch size, (2) multi-scale with fixed batch size, and (3) multi-scale with variable batch size. These sampling strategies are visualized in Figure 2a and discussed below:

- **Single-scale with fixed batch size (SSc-FBS):** This method is the default sampling strategy in most deep learning frameworks (e.g., PyTorch, Tensorflow, and MixNet) and libraries built on top of them (e.g., the timm library [16]). At the t-th training iteration, this method samples a batch of b images per GPU with a pre-defined spatial resolution of height H and width W.

- **Multi-scale with fixed batch size (MSc-FBS):** The SSc-FBS method allows a network to learn representations at a single scale (or resolution). However, objects in the real-world are composed of standard models for different computer vision tasks. Each model is sampled randomly at a defined spatial resolution of height H and width W.

- **Multi-scale with variable batch size (MSc-VBS):** Networks trained using the MSc-FBS methods are more robust to scale changes as compared to SSc-FBS [10]. However, depending on the maximum spatial resolution in S, MSc-FBS methods may have a higher peak GPU memory utilization (see Figure 2c) as compared to SSc-FBS; causing out-of-memory errors on GPUs with limited memory. For example, SSc-FBS with S = \{(128,128), (192,192), (224,224), (320,320)\} and b = 256 would need about 2x more GPU memory (for images only) than SSc-FBS with a spatial resolution of (224,224) and b = 256. To address this memory issue, we extend MSc-FBS to variably-batch sizes in our previous work [10]. For a given sorted set of spatial resolutions S = \{(H_1, W_1), (H_2, W_2), \ldots , (H_n, W_n)\} and a batch size b for a maximum spatial resolution of (H_n, W_n), a spatial resolution (H_t, W_t) ∈ S with a batch size of b_t = H_tW_t/b is sampled randomly at t-th training iteration on each GPU. These samplers offer different training costs and performance for different models, as shown in Figure 2c. Compared to SSc-FBS and MSc-FBS, MSc-VBS is a memory-efficient sampler that speeds-up the training significantly while maintaining performance.

Variably-sized video sampler. Samplers discussed above can be easily extended for videos. CNets provide variably-sized sampler for videos, wherein different video-related input variables (e.g., number of frames, number of clips per video, and spatial size) can be controlled for learning space- and time-invariant representations.

3.2 Sample efficient training

Previous works [8, 9, 11] remove and re-weight data samples to reduce optimization updates (or number of forward passes) at the

Listing 1: Registering a video classification model from PyTorchVideo inside CNets on the Kinetics-400 dataset

```python
from pytorchvideo.models import resnet

@register_video_cls_models("resnet_3d")
class ResNet3d(BaseVideoEncoder):
    def __init__(self, opts):
        super().__init__(opts=opts)
        self.model = resnet.create_resnet(
            input_channel=3,
            model_depth=50,
            model_num_class=400,
            norm=nn.BatchNorm3d,
            activation=nn.ReLU,
        )
```
Figure 2: Training models with different sampling methods on ImageNet dataset. Models trained with MSc-VBS delivers similar performance, trains faster with fewer optimizer updates, and generalizes better (higher train loss; similar validation loss) as compared to the ones trained with SSc-FBS and MSc-FBS. Notably, models trained with MSc-VBS require similar computational resources as SSc-FBS. Here, models are trained with simple training recipes.

Figure 3 shows results for ResNet-50 trained with and without SET using MSc-VBS. ResNet-50 without SET requires 22% more optimization updates while delivering similar performance; demonstrating the effectiveness of SET on top of MSc-VBS. Note that SET has an overhead. Therefore, the reduction in optimization updates do not translate to reduction in training time. We believe SET can serve as a baseline in this direction and inspire future research to improve training speed while maintaining performance.

### 3.3 Standard components

CVNets support different tasks (e.g., image classification, detection, segmentation), data augmentation methods (e.g., flipping, random resized crop, RandAug, and CutMix), datasets (e.g., ImageNet-1k/21k for image classification, Kinetics-400 for video classification,
MS-COCO for object detection, and ADE20k for segmentation), optimizers (e.g., SGD, Adam, and AdamW), and learning rate annealing methods (e.g., fixed, cosine, and polynomial).

4 BENCHMARKS
CVNets support different visual recognition tasks, including classification, detection, and segmentation. We provide comprehensive benchmarks for standard methods along with pre-trained weights.

Classification on ImageNet dataset. CVNets implement popular light- and heavy-weight image classification models. The performance of some of these models on the ImageNet dataset is shown in Table 1. With CVNets, we are able to achieve better performance (e.g., MobileNetV1/v2) or similar performance (ResNet-50/101) with fewer optimization updates (faster training).

Detection and segmentation. Similar to image classification, CVNets can be used to train standard detection and segmentation models with better performance. For example, SSD with ResNet-101 backbone trained with CVNets at a resolution of 384 × 384 delivers a 1.6% better mAP than the same model trained at a resolution of 512 × 512 as reported in [5]. Similarly, on the task of semantic segmentation on the ADE20k dataset using DeepLabv3 with MobileNetv2 as the backbone, CVNets delivers 1.1% better performance than MSSegmentation library [1] with 2× fewer epochs and optimization updates. For more details, please see our benchmarking results at https://github.com/apple/ml-cvnets.

5 CONCLUSION
This paper introduces CVNets, a modular deep learning library for visual recognition tasks with high performance. In future, we plan to continue enhancing CVNets with novel and reproducible methods. We welcome contributions from the research and open-source community to support further innovation.

REFERENCES
[1] MM Segmentation Contributors. 2020. MM Segmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark.
[2] Jia Deng et al. 2009. ImageNet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition.
[3] Haoqi Fan et al. 2020. ImageNet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition.
[4] Martin Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.
[5] Cheng-Yang Fu et al. 2017. Dsd: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659 (2017).
[6] Andrew G Howard et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).
[7] Krishnateja Killamsetty et al. 2021. GLISTER: Generalized based Data Subset Selection for Efficient and Robust Learning. Proceedings of the AAAI Conference on Artificial Intelligence 35 (May 2021), 8110–8118.
[8] Baharan Mirzasoleiman et al. 2020. Coresets for Data-efficient Training of Machine Learning Models. In Proceedings of the 37th International Conference on Machine Learning.
[9] Adam Paszke et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. 8024–8035.
[10] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition.
[11] Olga Russakovsky et al. 2015. Imagenet large scale visual recognition challenge. International journal of computer vision 115, 3 (2015), 211–252.
[12] Hugo Touvron et al. 2021. Training data-efficient image transformers & distillation through attention. In International Conference on Machine Learning. PMLR, 10347–10357.
[13] Ross Wightman, Hugo Touvron, and Hervé Jégou. 2021. Resnet strikes back: An improved training procedure in timm. arXiv preprint arXiv:2110.00476 (2021).

Table 1: Classification on the ImageNet dataset.

| Model                  | Params | FLOPs | Top-1 (in %) |
|------------------------|--------|-------|--------------|
|                        |        |       | CVNets (Ours) | Prev. |
| MobileNetV1-0.25       | 0.5 M  | 46.3 M | 54.2         | 49.8  |
| MobileNetV1-1.0        | 4.2 M  | 568.7 M| 74.1         | 70.9  |
| MobileNetV2-0.25       | 1.5 M  | 58.5 M | 53.6         | –     |
| MobileNetV2-1.0        | 3.5 M  | 308.7 M| 72.9         | 72.0  |
| MobileNetV3-Large      | 5.4 M  | 210.7 M| 75.1         | 75.2 (TFC) |
| ResNet-101 (simple recipe) | 44.5 M | 7.7 G | 79.8         | 77.4 (★) |
| ResNet-101 (adv. recipe) | 44.5 M | 7.7 G | 81.8         | 81.8 (★) |
| ViT-Tiny               | 5.7 M  | 1.3 G  | 72.9         | 72.2 (★) |

† Torchvision [12] requires 1.3x more optimization updates (forward passes) as compared to CVNets * Results are from [15].