**Abstract**

SCENIC is an open-source JAX library with a focus on transformer-based models for computer vision research and beyond. The goal of this toolkit is to facilitate rapid experimentation, prototyping, and research of new architectures and models. SCENIC supports a diverse range of tasks (e.g., classification, segmentation, detection) and facilitates working on multi-modal problems, along with GPU/TPU support for large-scale, multi-host and multi-device training. SCENIC also offers optimized implementations of state-of-the-art research models spanning a wide range of modalities. SCENIC has been successfully used for numerous projects and published papers and continues serving as the library of choice for rapid prototyping and publication of new research ideas.

1. Introduction

It is an exciting time for research in computer vision using attention based models. With new architectures like ViT [12] taking the world by storm, there exists a clear demand for software and machine learning infrastructure to support easy and extensible neural network architecture research. As attention models [6, 12, 19, 28] and MLP-only [30] architectures become more popular, we expect to see even more research in the coming years pushing the field forward.

We introduce SCENIC, an open-source JAX library for fast and extensible research in vision and beyond. SCENIC has been successfully used to develop classification, segmentation, and detection models for images, videos, and audio among other modalities, including multi-modal setups. SCENIC offers an efficient and easy to use setup for transfer learning by providing pipelines required for upstream pre-training, e.g., using large scale data and/or self-supervised training objectives, as well as downstream evaluation, e.g., zero-shot learning, few-shot learning, linear probe, and full fine-tuning on several tasks and datasets.

SCENIC strives to be a unified, all-in-one codebase for modeling needs, currently offering implementations of state-of-the-art models in various tasks like ViT [12], DETR [8], CLIP [22], MLP Mixer [30], T5 [23], BERT [11], ResNet [13], and U-Net [24]. On top of that, SCENIC has been used in numerous Google projects and research papers such as ViViT [6], OmniNet [28], TokenLearner [25], MBT [21], PolyViT [19], MTV [31], studies on scaling behaviour [5, 10, 29] and efficiency [9] of various models among others [15–18, 20]. We anticipate more research projects, with diverse flavors, to be open-sourced in the SCENIC repository in the near future.

SCENIC is developed in JAX [7] and uses Flax [14] as the neural network library, relies on TFDS [4] and DMVR [2] for implementing the input pipeline of most of the tasks and datasets, and makes use of for common training loop functionalities offered by CLU [1]. JAX is an ultra-simple to use library that enables automatic differentiation of native Python and NumPy functions. Moreover, it supports multi-host and multi-device training on accelerators including GPUs and TPUs, making it ideal for large-scale machine learning research.

In a nutshell, SCENIC is a (i) set of shared light-weight libraries solving commonly encountered tasks when training large-scale (i.e. multi-device, multi-host) models in vision and beyond; and (ii) a number of projects containing fully fleshed out problem-specific training and evaluation loops using these libraries.

SCENIC is designed to propose different levels of ab-
The code in SCENIC is organized in **project-level** part, that is customized code for specific projects or baselines or **library-level** part, that implements common functionalities and general patterns that are adapted by the majority of projects.

Philosophy  SCENIC aims to facilitate the rapid prototyping of large-scale models. To keep the code simple to understand and extend, SCENIC design prefers forking and copy-pasting over adding complexity or increasing abstraction. We only upstream functionality to the library-level when it proves to be widely useful across multiple models and tasks.

Minimizing support for various use-cases in the library-level code helps us to avoid accumulating generalizations that result in the code being complex and difficult to understand. Note that complexity or abstractions of any level can be added to project-level code.

2. A Brief Primer on JAX and Flax

Before we give into the design of SCENIC, we provide a quick background on JAX [7]. JAX provides composable transformations of Python and NumPy programs that offer differentiation (e.g., `jax.grad`), vectorization (e.g., `jax.vmap`), JIT-compilation to GPU/TPU (e.g., `jax.jit`), and more. JAX is designed to operate on pure and statically composed (PSC) functions. That means JAX transformations are only applicable to functions that (1) make no use of a global state, (2) are deterministic, and (3) are representable as a static data dependency graph on a set of primitives (e.g., Accelerated Linear Algebra (XLA) operators). Many machine learning systems can be implemented as a set of PSC Python functions and using JAX transformation, we can easily benefit high efficiency of XLA for running compute-heavy parts of these systems.

While JAX offers the building blocks needed for developing machine learning systems in the form of function transformations, there are other libraries on top of it to address the higher-level needs of researchers. Most notably Flax [14] which is a neural network library for JAX, providing neural network layers as well as utilities and patterns. SCENIC uses JAX and Flax underneath to develop a highly efficient framework for agile development of research ideas, in the large-scale, yet flexible setup.

3. SCENIC Design

SCENIC offers a unified framework that is sufficiently flexible to support projects in a wide range of needs without having to write complex code. SCENIC contains optimized implementations of a set of research models operating on a wide range of modalities (video, image, audio, and text), and supports several datasets. This again is made possible by its flexible and low-overhead design. In this section, we go over different parts and discuss the structure that is used to organize projects and library code.
3.1. Library-level code

The goal is to keep the library-level code minimal and well-tested and to avoid introducing extra abstractions to support minor use-cases. Shared libraries provided by SCENIC are split into:

- **dataset**lib**: Implements IO pipelines for loading and pre-processing data for common tasks and benchmarks. All pipelines are designed to be scalable and support multi-host and multi-device setups, taking care of dividing data among multiple hosts, incomplete batches, caching, pre-fetching, etc.

- **model**lib**: Provides several abstract model interfaces (e.g., `ClassificationModel` or `SegmentationModel` in `model_lib/models`) with task-specific losses and metrics; neural network layers in `model_lib/layers`, focusing on efficient implementation of attention and transformer primitives; and finally accelerator-friendly implementations of bipartite matching algorithms [3] in `model_lib/matchers`.

- **train**lib**: Provides tools for constructing training loops and implements several optimized trainers (e.g., classification trainer and segmentation trainer) that can be forked for customization.

- **common**lib**: General utilities, such as logging and debugging modules, and functionalities for processing raw data.

3.2. Project-level code

SCENIC supports the development of customized solutions for specialized tasks and data via the concept of the “project”. There is no one-fits-all recipe for how much code should be re-used by a project.

Projects can consist of only configuration files and use the common models, drivers, tasks/data that live in library-level code, or they can simply fork any of the mentioned functionalities (e.g., classification trainer and segmentation trainer) that can be forked for customization.

3.3. SCENIC BaseModel

A solution usually has several parts: data/task pipeline, model architecture, losses and metrics, training and evaluation, etc. Given that much of the research done in SCENIC is trying out different architectures, SCENIC introduces the concept of a “model”, to facilitate “plug-in/plug-out” experiments. A SCENIC model is defined as the network architecture, the losses that are used to update the weights of the network during training, and metrics that are used to evaluate the output of the network. This is implemented as `BaseModel`.

`BaseModel` is an abstract class with three members: a `build_flax_model`, a `loss_fn`, and a `get_metrics_fn`. `build_flax_model` function returns a flax model. A typical usage pattern is depicted below:

```python
# Get model class:
model_cls = model_lib.models.get_model_cls(
    "fully_connected_classification")
# Build the model, metrics, and losses
model = model_cls(
    config, dataset.meta_data)
# Initialize the model parameters
flax_model = model.build_flax_model
dummy_input = jnp.zeros(
    input_shape, model_input_dtype)
model_state, params = flax_model.init(
    rng, dummy_input, train=False)
.pop("params")
```

And this is how to call the model:

```python
variables = {
    # Trainable parameters
    "params": params,
    # Model state
    # (e.g., batch statistics from BatchNorm)
    **model_state
}
logits, new_model_state = flax_model.apply(
    variables, inputs, ...)
```

Abstract classes for defining SCENIC models are declared in `model_lib/base_models`. These include the `BaseModel` that all models inherit from, as well as `ClassificationModel`, `MultiLabelClassificationModel`, `EncoderDecoderModel` and `SegmentationModel` that respectively define losses and metrics for classification, sequence-to-sequence, and segmentation tasks. Depending on its needs, a SCENIC project can define new base class or override an existing one for its specific tasks, losses and metrics.

A typical model loss function in SCENIC expects predictions and a batch of data:
Finally, a typical get_metrics_fn returns a callable, metric_fn, that calculates appropriate metrics and returns them as a Python dictionary. The metric function, for each metric, computes \( f(x_i, y_i) \) on a mini-batch, where \( x_i \) and \( y_i \) are inputs and labels of \( i \)th example, and returns a dictionary from the metric name to a tuple of metric value and the metric normalizer (typically, the number of examples in the mini-batch). It has the API:

```python
# Metric function:
metric_fn(
    logits: jnp.ndarray,
    label: jnp.ndarray,
) -> Dict[str, Tuple[float, int]]
```

Given metric and normalizer values collected from examples processed by all devices in all hosts, the model trainer is then responsible for aggregating and computing the normalized metric value for the evaluated examples.

Importantly, while the design pattern above is recommended and has been found to work well for a range of projects, it is not forced, and there is no issue deviating from the above structure within a project.

4. Applications

SCENIC has been used in various applications and tasks. SCENIC provides not only different metrics to evaluate quality of different methods on task at hand, like accuracy in classification, but also unified tools to assess efficiency of methods in terms of training and inference throughput, parameter size, and FLOPs.

In this Section, we present some imperial results showcasing some of the baselines in SCENIC.

Table 1. Image classification models implemented in SCENIC.

<table>
<thead>
<tr>
<th>Model</th>
<th>Pretrain dataset</th>
<th>ImageNet Top-1</th>
<th>GFLOPs</th>
<th>Params(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP-Mixer-B/16 [30]</td>
<td>-</td>
<td>76.14</td>
<td>12.7</td>
<td>59.8</td>
</tr>
<tr>
<td>ViT-B/16 [26]</td>
<td>-</td>
<td>79.73</td>
<td>17.6</td>
<td>86.5</td>
</tr>
<tr>
<td>MLP-Mixer-L/16</td>
<td>ImageNet21K</td>
<td>80.23</td>
<td>12.7</td>
<td>59.8</td>
</tr>
<tr>
<td>ViT-L/16</td>
<td>ImageNet21K</td>
<td>84.30</td>
<td>17.6</td>
<td>86.5</td>
</tr>
</tbody>
</table>

Table 1 presents example baseline models for image classification task that are implemented in SCENIC. The full end-to-end upstream training and downstream fine-tuning/evaluation code with support for data parallelism on multi-host/multi-device is available for all SCENIC baselines. Note that for each of these baseline models, checkpoints for different variants, in terms of size and configurations, are publically available.

SCENIC contains many state-of-the-art models, along with training/evaluation code as well as checkpoints for tasks beyond classification and image modality. Table 2 provides a few examples showcasing the diversity of tasks and modalities. These models are either originally developed in SCENIC and became the de facto baseline for the task they address (e.g., ViViT), or carefully ported to SCENIC to not only reproduce the original results but also be efficient on different accelerators using JAX.

This has already enabled many researchers to build on top of the existing SOTA models and try out new ideas on different setups and modalities, which is one of the main missions of SCENIC.

Table 2. Other models available in SCENIC. Full training and evaluation code which reproduces the original results is publicly available for all these models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Modalities</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeTR [8]</td>
<td>Image</td>
<td>Object detection</td>
</tr>
<tr>
<td>ViViT [6]</td>
<td>Video</td>
<td>Video classification</td>
</tr>
<tr>
<td>TokenLearner [25]</td>
<td>Image, Video</td>
<td>Image/Video classification</td>
</tr>
<tr>
<td>OmniNet [28]</td>
<td>Image</td>
<td>Image classification</td>
</tr>
<tr>
<td>PolyViT [19]</td>
<td>Image, Audio, Video</td>
<td>Image/Audio/Video classification</td>
</tr>
</tbody>
</table>

For each of the above methods as well as others implemented in SCENIC, there is a dedicated README in the GitHub repo of SCENIC, containing information about the detailed configurations, experimental results, how to train, how to evaluate, along with brief explanation of the methods and references.

5. Conclusion

Machine Learning (ML) infrastructure is a cornerstone of ML research. Enabling researchers to quickly try out new ideas, and to rapidly scale them up when they show promise, accelerates research. Furthermore, history suggests that methods that leverage the computation available at the time are often the most effective [27]. SCENIC embodies our experience of developing the best research infrastructure, and we are excited to share it with the broader community. We hope to see many more brilliant ideas being developed using SCENIC, contributing to the amazing progress made by the ML community for improving lives through AI.
Acknowledgment

SCENIC has been developed and improved with the help of many amazing contributors. We would like to especially thank Dirk Weissenborn, Andreas Steiner, Marvin Ritter, Aravindh Mahendran, Samira Abnar, Sunayana Rane, Josip Djolonga, Lucas Beyer, Alexander Kolesnikov, Xiaohua Zhai, Rob Romijnders, Rianne van den Berg, Jonathan Heek, Olivier Teboul, Marco Cuturi, Lu Jiang, Mario Lučić, and Neil Houlsby for their direct or indirect contributions to SCENIC.

References


