Scenic: A JAX Library for Computer Vision Research and Beyond

Mostafa Dehghani
Google Brain
dehghani@google.com

Alexey Gritsenko
Google Brain
agritsenko@google.com

Anurag Arnab
Google Research
aarnab@google.com

Matthias Minderer
Google Brain
mjlm@google.com

Yi Tay
Google Research
yitay@google.com

Abstract

Scenic *is an open-source*[1] *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.

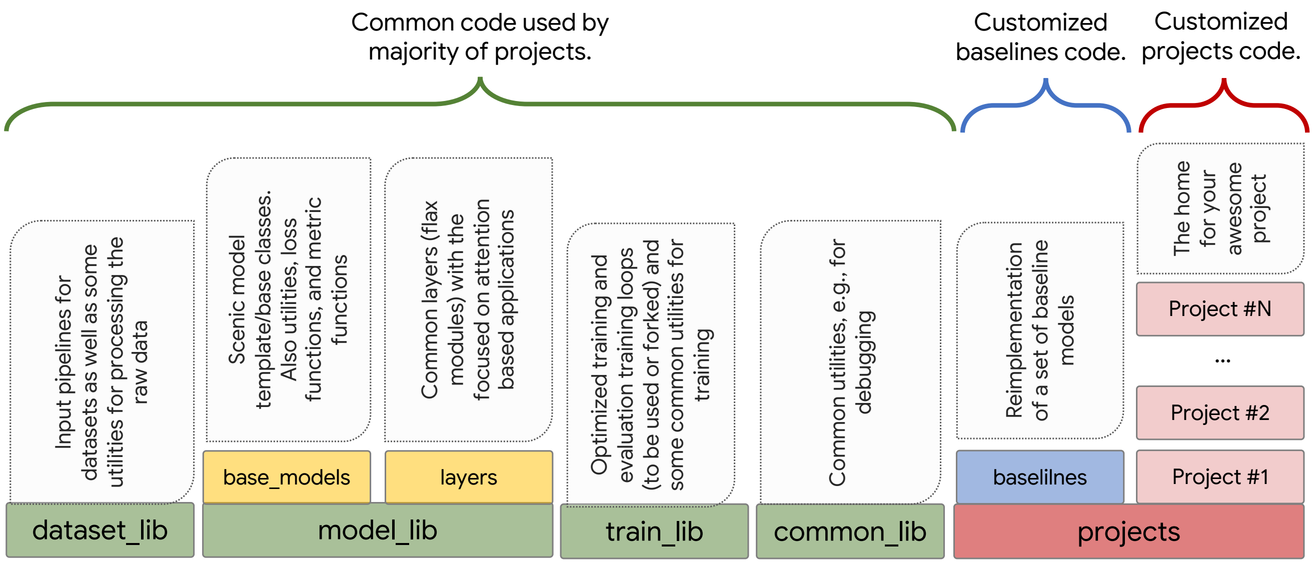
Scenic strives to be a unified, all-in-one codebase for modeling needs, currently offering implementations of state-of-the-art models in various tasks like ViT [12], DETR [8], CLIP [22], MLP Mixer [30], T5 [23], BERT [11] ResNet [13], and U-Net [24]. On top of that, Scenic has been used in numerous Google 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-

[1] https://github.com/google-research/scenic



Common code used by majority of projects.
Customized baselines code.
Customized projects code.
Input pipelines for datasets as well as some utilities for processing the raw data
Scenic model template/base classes. Also utilities, loss functions, and metric functions
Common layers (flax modules) with the focused on attention based applications
Optimized training and evaluation training loops (to be used or forked) and some common utilities for training
Common utilities, e.g., for debugging
Reimplementation of a set of baseline models
The home for your awesome project
Project #N
...
Project #2
Project #1
base_models
layers
baselilnes
dataset_lib
model_lib
train_lib
common_lib
projects

Figure 1. 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.

straction. It supports projects from those that only require changing hyper-parameters, to those that need customization on the input pipeline, model architecture, losses and metrics, and the training loop. To make this happen, the code in SCENIC is organized as either **project-level** code, which refers to customized code for specific projects or baselines, or **library-level** code, which refers to common functionalities and general patterns that are adapted by the majority of projects (Fig. 1). The project-level code lives in the `projects` directory.

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/base_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, trainers, tasks/data that live in library-level code, or they can simply fork any of the mentioned functionalities and redefine, layers, losses, metrics, logging methods, tasks, architectures, as well as training and evaluation loops. The modularity of library-level code makes it flexible enough to support projects falling anywhere on the "run-as-is" to "fully-customized" spectrum.

Common baselines such as a Vision Transformer (ViT), DETR, CLIP, T5, BERT, MLP-Mixer, ResNet, UNet, etc., are implemented in the `projects/baselines`. Forking models in this directory is a good starting point for new projects.

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:

```
# 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:

```
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:

```
# Loss function:
loss_fn(
    logits: jnp.ndarray,
    batch: Dict[str, jnp.ndarray]
    ) -> float
```

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 ith 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:

```
# Metric function:
metric_fn(
    logits: jnp.ndarry,
    label: jnp.ndarry,
    ) -> 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.

Model	Pretrain dataset	ImageNet Top-1	GFLOPs	Params(M)
ResNet-50 [13]	-	77.08	3.97	25.5
MLP-Mixer-B/16 [30]	-	76.14	12.7	59.8
ViT-B/16 [26]	-	79.73	17.6	86.5
MLP-Mixer-L/16	ImageNet21K	80.23	12.7	59.8
ViT-L/16	ImageNet21K	84.30	17.6	86.5

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.

Model	Modalities	Task
DeTR [8]	Image	Object detection
BERT [11]	Text	Self-supervised pre-training, GLUE/SuperGLUE benchmark
ViViT [6]	Video	Video classification
TokenLearner [25]	Image, Video	Image/Video classification
OmniNet [28]	Image	Image classification
PolyViT [19]	Image, Audio, Video	Image/Audio/Video classification

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

[1] CLU - common loop utils. `http://github.com/google/CommonLoopUtils`. 1

[2] DMVR: Deepmind video readers. `https://github.com/deepmind/dmvr`. 1

[3] Optimal Transport Tools (OTT), a toolbox for everything wasserstein. `https://github.com/google-research/ott`. 3

[4] TensorFlow Datasets, a collection of ready-to-use datasets. `https://www.tensorflow.org/datasets`. 1

[5] Samira Abnar, Mostafa Dehghani, Behnam Neyshabur, and Hanie Sedghi. Exploring the limits of large scale pre-training. *arXiv preprint arXiv:2110.02095*, 2021. 1

[6] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision transformer. *arXiv preprint arXiv:2103.15691*, 2021. 1, 4

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 1, 2

[8] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020. 1, 4

[9] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *arXiv preprint arXiv:2110.12894*, 2021. 1

[10] Mostafa Dehghani, Yi Tay, Alexey A Gritsenko, Zhe Zhao, Neil Houlsby, Fernando Diaz, Donald Metzler, and Oriol Vinyals. The benchmark lottery. *arXiv preprint arXiv:2107.07002*, 2021. 1

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1, 4

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 4

[14] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. 1, 2

[15] Charles Herrmann, Kyle Sargent, Lu Jiang, Ramin Zabih, Huiwen Chang, Ce Liu, Dilip Krishnan, and Deqing Sun. Pyramid adversarial training improves vit performance. *arXiv preprint arXiv:2111.15121*, 2021. 1

[16] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. *arXiv preprint arXiv:2112.01455*, 2021. 1

[17] Gang Li, Gilles Baechler, Manuel Tragut, and Yang Li. Learning to denoise raw mobile ui layouts for improving datasets at scale. *arXiv preprint arXiv:2201.04100*, 2022. 1

[18] Yang Li, Gang Li, Xin Zhou, Mostafa Dehghani, and Alexey Gritsenko. Vut: Versatile ui transformer for multi-modal multi-task user interface modeling. *arXiv preprint arXiv:2112.05692*, 2021. 1

[19] Valerii Likhosherstov, Anurag Arnab, Krzysztof Choromanski, Mario Lucic, Yi Tay, Adrian Weller, and Mostafa Dehghani. Polyvit: Co-training vision transformers on images, videos and audio. *arXiv preprint arXiv:2111.12993*, 2021. 1, 4

[20] Chengzhi Mao, Lu Jiang, Mostafa Dehghani, Carl Vondrick, Rahul Sukthankar, and Irfan Essa. Discrete representations strengthen vision transformer robustness. *arXiv preprint arXiv:2111.10493*, 2021. 1

[21] Arsha Nagrani, Shan Yang, Anurag Arnab, Aren Jansen, Cordelia Schmid, and Chen Sun. Attention bottlenecks for multimodal fusion. *arXiv preprint arXiv:2107.00135*, 2021. 1

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 1

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019. 1

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015. 1

[25] Michael S Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. TokenLearner: What can 8 learned tokens do for images and videos? *arXiv preprint arXiv:2106.11297*, 2021. 1, 4

[26] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021. 4

[27] Richard S Sutton. The bitter lesson, march 2019. *URL http://www. incompleteideas. net/IncIdeas/BitterLesson. html*, 1. 4

[28] Yi Tay, Mostafa Dehghani, Vamsi Aribandi, Jai Gupta, Philip Pham, Zhen Qin, Dara Bahri, Da-Cheng Juan, and Donald Metzler. OmniNet: Omnidirectional representations from transformers. *arXiv preprint arXiv:2103.01075*, 2021. 1, 4

[29] Yi Tay, Mostafa Dehghani, Jinfeng Rao, William Fedus, Samira Abnar, Hyung Won Chung, Sharan Narang, Dani Yogatama, Ashish Vaswani, and Donald Metzler. Scale efficiently: Insights from pre-training and fine-tuning transformers. *arXiv preprint arXiv:2109.10686*, 2021. 1

[30] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. MLP-Mixer: An all-MLP architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021. 1, 4

[31] Shen Yan, Xuehan Xiong, Anurag Arnab, Zhichao Lu, Mi Zhang, Chen Sun, and Cordelia Schmid. Multi-view transformers for video recognition. *arXiv preprint arXiv:2201.04288*, 2022. 1