
Swift for TensorFlow Models

This repository contains many examples of how Swift for TensorFlow can be used to build machine learning applications, as well as the models, datasets, and other components required to build them. These examples are intended to demonstrate best practices for the use of Swift for TensorFlow APIs and act as end-to-end tests to validate the function and performance of those APIs.

Active development occurs on the [main](#) branch, and that is kept current against the [main](#) branch of the Swift compiler and the [main](#) branch of the Swift for TensorFlow APIs.

For stable snapshots, use the [tensorflow-xx](#) branch that corresponds to the toolchain you are using from the Swift for TensorFlow releases. For example, for the 0.12 release, use the [tensorflow-0.12](#) branch.

To learn more about Swift for TensorFlow development, please visit [tensorflow/swift](https://tensorflow.org/swift).

Examples

The examples within this repository are all designed to be run as standalone applications. The easiest way to do this is to use Swift Package Manager to build and run individual examples. This can be accomplished by changing to the root directory of the project and typing something like

```
1 swift run -c release [Example] [Options]
```

For Windows, an additional flag may be required:

```
1 swift run -Xswiftc -use-ld=lld -c release [Example] [Options]
```

This will build and run a specific example in the release configuration. Due to significant performance differences between debug and release builds in Swift, we highly recommend running the examples from a release build. Some examples have additional command-line options, and those will be described in the example's README.

The following is a catalog of the current examples, grouped by subject area, with links to their location within the project. Each example should have documentation for what it is demonstrating and how to use it.

Image classification

- A custom model training against CIFAR-10
- LeNet-5 training against MNIST
- MobileNet V1 training against Imagenette

-
- MobileNet V2 training against Imagenette
 - ResNet-56 training against CIFAR-10
 - ResNet-50 training against ImageNet
 - VGG-16 training against Imagewoof

Text

- BERT training against CoLA
- Pretrained GPT-2 performing text generation
- GPT-2 training against WikiText-2
- WordSeg

Generative models

- 1-D Autoencoder
- 2-D Autoencoder
- 1-D Variational Autoencoder
- CycleGAN
- GAN
- DCGAN
- pix2pix

Reinforcement learning

- Blackjack
- CartPole
- Catch
- FrozenLake
- MiniGo

Standalone

- Differentiable Shallow Water PDE Solver
- Fast Style Transfer
- Fractals
- Growing Neural Cellular Automata
- Neural Collaborative Filtering using MovieLens

-
- PersonLab Human Pose Estimator
 - Regression using BostonHousing

Components

Beyond examples that use Swift for TensorFlow, this repository also contains reusable components for constructing machine learning applications. These components reside in modules that can be imported into separate Swift projects and used by themselves.

These components provide standalone machine learning models, datasets, image loading and saving, TensorBoard integration, and a training loop abstraction, among other capabilities.

The Swift for TensorFlow models repository has acted as a staging ground for experimental capabilities, letting us evaluate new components and interfaces before elevating them into the core Swift for TensorFlow APIs. As a result, the design and interfaces of these components may change regularly.

Models

Several modules are provided that contain reusable Swift models for image classification, text processing, and more. These modules are used within the example applications to demonstrate the capabilities of these models, but they can also be imported into many other projects.

Image classification Many common image classification models are present within the ImageClassificationModels module. To use them within a Swift project, add ImageClassificationModels as a dependency and import the module:

```
1 import ImageClassificationModels
```

These models include:

- DenseNet121
- EfficientNet
- LeNet-5
- MobileNetV1
- MobileNetV2
- MobileNetV3
- ResNet
- ResNetV2
- ShuffleNetV2
- SqueezeNet

-
- VGG
 - WideResNet
 - Xception

Recommendation Several recommendation models are present within the RecommendationModels module. To use them within a Swift project, add RecommendationModels as a dependency and import the module:

```
1 import RecommendationModels
```

These models include:

- DLRM
- MLP
- NeuMF

Text Several text models are present within the TextModels module. To use them within a Swift project, add TextModels as a dependency and import the module:

```
1 import TextModels
```

These models include:

- BERT
- GPT-2
- WordSeg

Datasets

In addition to the machine learning model itself, a dataset is usually required when training. Swift wrappers have been built for many common datasets to ease their use within machine learning applications. Most of these use the Epochs API that provides a generalized abstraction of common dataset operations.

The Datasets module provides these wrappers. To use them within a Swift project, add Datasets as a dependency and import the module:

```
1 import Datasets
```

These are the currently provided dataset wrappers:

- BostonHousing

-
- CIFAR-10
 - MS COCO
 - CoLA
 - ImageNet
 - Imagenette
 - Imagewoof
 - FashionMNIST
 - KuzushijiMNIST
 - MNIST
 - MovieLens
 - Oxford-IIIT Pet
 - WordSeg

Model checkpoints

Model saving and loading is provided by the Checkpoints module. To use the model checkpointing functionality, add Checkpoints as a dependency and import the module:

```
1 import Checkpoints
```

Image loading and saving

The ModelSupport module contains many shared utilities that are needed within the Swift machine learning examples. This includes the loading, saving, and processing of still images via the `stb_image` library. Animated images can also be written out as GIF files from multiple tensors.

Experimental support for `libjpeg-turbo` as an accelerated image loader is present, but has not yet been incorporated into the main image loading capabilities.

Generalized training loop

A generalized training loop that can be customized via callbacks is provided within the Training Loop module. All of the image classification examples use this training loop, with the exception of the Custom-CIFAR10 example that demonstrates how to define your own training loop from scratch. Other examples are being gradually converted to use this training loop.

TensorBoard integration

TensorBoard integration is provided in the TensorBoard module as a callback for the generalized training loop. TensorBoard lets you visualize the progress of your model as it trains by plotting model statistics as they update, or to review the training process afterward.

The GPT2-WikiText2 example demonstrates how this can be used when training your own models.

Benchmarks and tests

A core goal of this repository is to validate the proper function of the Swift for TensorFlow APIs. In addition to the models and end-to-end applications present within this project, a suite of benchmarks and unit tests reside here.

The benchmarks are split into a core of functionality, the SwiftModelsBenchmarksCore module, and a Benchmarks command-line application for running these benchmarks. Refer to the documentation for how to run the benchmarks on your system.

The unit tests verify functionality within models, datasets and other components. To run them using Swift Package Manager on macOS or Linux:

```
1 swift test
```

and to run them on Windows:

```
1 swift test -Xswiftc -use-ld=lld -c debug
```

Using CMake for Development

In addition to Swift Package Manager, CMake can be used to build and run Swift for TensorFlow models.

Experimental CMake Support

There is experimental support for building with CMake. This can be used to cross-compile the models and the demo programs.

It is highly recommended that you use CMake 3.16 or newer to ensure that `-B` and parallel builds function properly in the example commands below. To install this version on Ubuntu, we recommend following the instructions at Kitware's apt repo.

Prerequisite: Ninja build tool. Find installation commands for your favorite package manager here.

macOS:

```
1 # Configure
2 cmake \
3 -B /BinaryCache/tensorflow-swift-models \
4 -D BUILD_TESTING=YES \
5 -D CMAKE_BUILD_TYPE=Release \
6 -D CMAKE_Swift_COMPILER=$(TOOLCHAINS=tensorflow xcrun -f swiftc) \
7 -G Ninja \
8 -S /SourceCache/tensorflow-swift-models
9 # Build
10 cmake --build /BinaryCache/tensorflow-swift-models
11 # Test
12 cmake --build /BinaryCache/tensorflow-swift-models --target test
```

Linux:

```
1 # Configure
2 cmake \
3 -B /BinaryCache/tensorflow-swift-models \
4 -D BUILD_TESTING=NO \
5 -D CMAKE_BUILD_TYPE=Release \
6 -D CMAKE_Swift_COMPILER=$(which swiftc) \
7 -G Ninja \
8 -S /SourceCache/tensorflow-swift-models
9 # Build
10 cmake --build /BinaryCache/tensorflow-swift-models
```

Windows:

```
1 set DEVELOPER_LIBRARY_DIR=%SystemDrive%/Library/Developer/Platforms/
   Windows.platform/Developer/Library
2 :: Configure
3 "%ProgramFiles%\CMake\bin\cmake.exe"
   ^
4 -B %SystemDrive%/BinaryCache/tensorflow-swift-models
   ^
5 -D BUILD_SHARED_LIBS=YES
   ^
6 -D BUILD_TESTING=YES
   ^
7 -D CMAKE_BUILD_TYPE=Release
   ^
```

```
8 -D CMAKE_Swift_COMPILER=%SystemDrive%/Library/Developer/Toolchains/  
   unknown-Asserts-development.xctoolchain/usr/bin/swiftpc.exe  
9 -D CMAKE_Swift_FLAGS="-sdk %SDKROOT% -I %DEVELOPER_LIBRARY_DIR%/  
   XCTest-development/usr/lib/swift/windows/x86_64 -L %  
   DEVELOPER_LIBRARY_DIR%/XCTest-development/usr/lib/swift/windows" ^  
10 -G Ninja  
   ^  
11 -S %SystemDrive%/SourceCache/tensorflow-swift-models  
12 :: Build  
13 "%ProgramFiles%\CMake\bin\cmake.exe" --build %SystemDrive%/BinaryCache/  
   tensorflow-swift-models  
14 :: Test  
15 "%ProgramFiles%\CMake\bin\cmake.exe" --build %SystemDrive%/BinaryCache/  
   tensorflow-swift-models --target test
```

Bugs

Please report model-related bugs and feature requests using GitHub issues in this repository.

Community

Discussion about Swift for TensorFlow happens on the swift@tensorflow.org mailing list.

Contributing

We welcome contributions: please read the Contributor Guide to get started. It's always a good idea to discuss your plans on the mailing list before making any major submissions.

We have labeled some issues as “good first issue” or “help wanted” to provide some suggestions for where new contributors might be able to start.

Code of Conduct

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

The Swift for TensorFlow community is guided by our Code of Conduct, which we encourage everybody to read before participating.