



Prabhat
Kumar
Sahu



Jun 13,
2021

·
4 min
read



How To Install TensorFlow on M1 Mac (The Easy Way)



Tensorflow

Last year in November 2020 apple releases their first ARM64-based M1 chip. It got a lot of attention from everyone.

Being a tech enthusiast and a

programmer, I was amazed to see the performance of the new apple M1 chip. The benchmarks were really good.

Recently only some months back, I bought myself an M1 Macbook Pro with 8Gigs of RAM and 512GB of SSD.

I wanted to set up and test how machine learning frameworks are working in this new chip. Here are the setup instructions for Tensorflow.

At this time of writing Apple has released its Tensorflow 2.5 for Apple Silicon.

It's very easy to set up.

Before jumping into, I hope Homebrew is already installed in your system if not you can install it by running the following in your terminal

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.  
sh)"
```

Here are the things that we are going to do.

- Step 1: Xcode Command Line Tools
- Step2: Install Miniforge
- Step3: Create a virtual environment with python3.8
- Step4: Install Tensorflow 2.5 and its dependencies
- Step5: Install Jupyter Notebook, Pandas
- Step6: Run a Benchmark by training the MNIST dataset

Step1: Install Xcode Command Line Tools

I have already installed Xcode Command Line Tools on my mac. If it's not already installed in your system, you can install it by running the following command below in your terminal.

```
xcode-select --install
```

Step2: Install Miniforge

Install miniforge for arm64 (Apple Silicon) from [miniforge GitHub](#).

Miniforge enables installing python packages natively compiled for Apple Silicon.

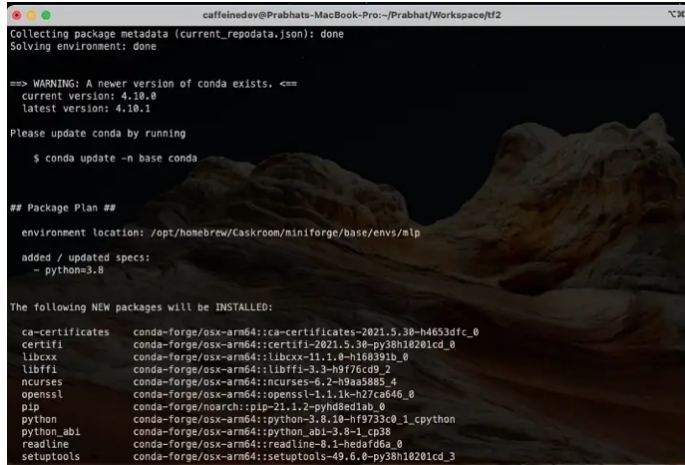
After the installation of miniforge, by default, it gives us one base environment. You can turn off the default base env by running

```
conda config --set  
auto_activate_base false
```

Step3: Create a virtual environment

Let's create a virtual environment named *mlp* with python3.8 installed.

```
conda create --name mlp  
python=3.8
```

A terminal window titled 'caffeinedev@Prabhats-MacBook-Pro:~/Prabhat/Workspace/mlp' showing the output of 'conda update -n base conda'. It displays a warning about a newer version of conda (4.10.1) and lists the packages to be installed, including ca-certificates, certifi, libgcc, libffi, ncurses, openssl, pip, python, python_abi, readline, and setuptools.

```
caffeinedev@Prabhats-MacBook-Pro:~/Prabhat/Workspace/mlp
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.10.0
  latest version: 4.10.1

Please update conda by running

  $ conda update -n base conda

## Package Plan ##

  environment location: /opt/homebrew/Caskroom/miniforge/base/envs/mlp
added / updated specs:
  - python=3.8

The following NEW packages will be INSTALLED:

ca-certificates  conda-forge/osx-arm64::ca-certificates-2021.5.30-h4653dfe_0
certifi          conda-forge/osx-arm64::certifi-2021.5.30-py38h10201cd_0
libgcc           conda-forge/osx-arm64::libgcc-11.1.0-h168391b_0
libffi           conda-forge/osx-arm64::libffi-3.3-h977cd9_2
ncurses          conda-forge/osx-arm64::ncurses-6.2-h9a5885_4
openssl          conda-forge/osx-arm64::openssl-1.1.1k-h27ca646_0
pip              conda-forge/noarch::pip-21.1.2-pyhd8ed1ab_0
python           conda-forge/osx-arm64::python-3.8.10-hf9733c0_1_cpython
python_abi       conda-forge/osx-arm64::python_abi-3.8-1_cp38
readline         conda-forge/osx-arm64::readline-8.1-hedaf66a_0
setuptools       conda-forge/osx-arm64::setuptools-49.6.0-py38h10201cd_3
```

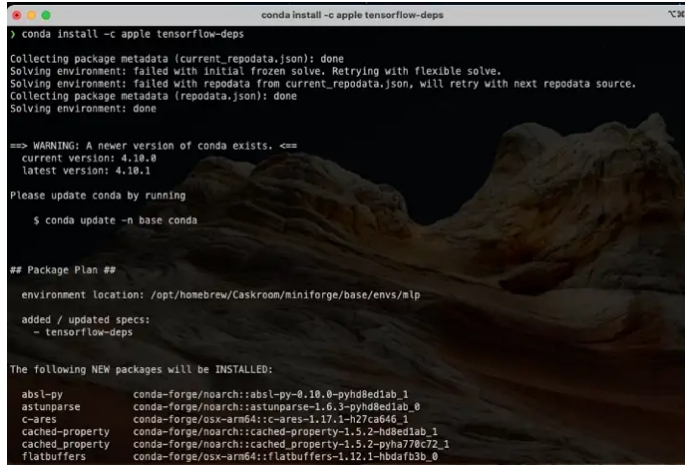
And activate it by running

```
conda activate mlp
```

Step4: Installing Tensorflow-MacOS

Install the Tensorflow dependencies:

```
conda install -c apple
tensorflow-deps
```

A terminal window titled 'conda install -c apple tensorflow-deps' showing the output of the command. The output includes messages about collecting package metadata, solving the environment, and a warning about a newer version of conda. It also shows the package plan for tensorflow-deps, listing several new packages to be installed.

```
conda install -c apple tensorflow-deps

Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.10.0
  latest version: 4.10.1

Please update conda by running

  $ conda update -n base conda

## Package Plan ##

  environment location: /opt/homebrew/Caskroom/miniforge/base/envs/mlp
  added / updated specs:
    - tensorflow-deps

The following NEW packages will be INSTALLED:

  absl-py           conda-forge/noarch::absl-py-0.10.0-pyhd8ed1ab_1
  astunparse        conda-forge/noarch::astunparse-1.6.3-pyhd8ed1ab_0
  c-ares            conda-forge/osx-arm64::c-ares-1.17.1-h27ca846_1
  cached-property   conda-forge/noarch::cached-property-1.5.2-pyha770c72_1
  flatbuffers        conda-forge/osx-arm64::flatbuffers-1.12.1-hbda6b3b_0
```

Install base TensorFlow:

```
pip install tensorflow-macos
```

Install metal plugin:

```
pip install tensorflow-metal
```

Step5: Install Jupyter Notebook & Pandas

```
conda install -c conda-forge -y
pandas jupyter
```

Step6: Run a Benchmark by training the MNIST dataset

Let's install Tensorflow Datasets

```
pip install tensorflow_datasets
```

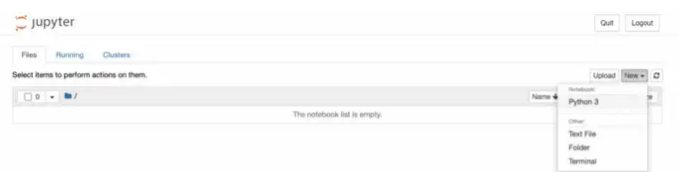
Make sure your conda environment is activated.

Let's open a Jupyter Notebook and do the benchmark. In your terminal run

```
jupyter notebook
```

It will open a browser window

Create a new python3 notebook

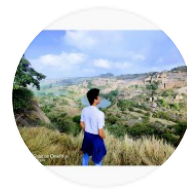


Let's first import TensorFlow and check

```
import tensorflow as tf
```

Get unlimited access

Search



Prabhat Kumar Sahu

139 Followers

{ ML Engineer 🖥️ }

Follow

More from Medium



Ja... in Tow...

GPU-Acceleration Comes to...



Ni... in Tow...

Installing PyTorch on Apple M1...



De... in M...

Install TensorFlow on Mac





```
print("Num GPUs Available: ",
      len(tf.config.experimental.list_
physical_devices('GPU')))
```

```
scroll output; double click to hide
In [1]: import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
Num GPUs Available: 1

In [3]: tf.config.list_physical_devices('GPU')
Out[3]: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

In [ ]:
```

Let's run a benchmark

I got the code snippet from

<https://github.com/apple/tensorflow-macos/issues/25>

Copy and paste this code below in the new notebook

```
%%time

import tensorflow as tf

import tensorflow_datasets as tfds

print("TensorFlow version:",
      tf.__version__)
print("Num GPUs Available: ",
      len(tf.config.experimental.list_
physical_devices('GPU')))
tf.config.list_physical_devices(
'GPU')

(ds_train, ds_test), ds_info =
tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
```

on mac...



Da... in To...



**How to
Manage
Conda...**

[Help](#) [Status](#) [Writers](#) [Blog](#) [Careers](#)
[Privacy](#) [Terms](#) [About](#)
[Text to speech](#)


```
        as_supervised=True,
        with_info=True,
    )

    def normalize_img(image, label):
        """Normalizes images: `uint8`
        -> `float32`."""
        return tf.cast(image,
            tf.float32) / 255., label

    batch_size = 128

    ds_train = ds_train.map(
        normalize_img,
        num_parallel_calls=tf.data.experimental.AUTOTUNE)
    ds_train = ds_train.cache()
    ds_train =
    ds_train.shuffle(ds_info.splits[
        'train'].num_examples)
    ds_train =
    ds_train.batch(batch_size)
    ds_train =
    ds_train.prefetch(tf.data.experimental.AUTOTUNE)

    ds_test = ds_test.map(
        normalize_img,
        num_parallel_calls=tf.data.experimental.AUTOTUNE)
    ds_test =
    ds_test.batch(batch_size)
    ds_test = ds_test.cache()
    ds_test =
    ds_test.prefetch(tf.data.experimental.AUTOTUNE)

    model =
    tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32,
            kernel_size=(3, 3),

            activation='relu'),
        tf.keras.layers.Conv2D(64,
            kernel_size=(3, 3),

            activation='relu'),

        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        #
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128,
```

```

activation='relu'),
#
tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10,
activation='softmax')
])
model.compile(

loss='sparse_categorical_crossentropy',

optimizer=tf.keras.optimizers.Adam(0.001),
    metrics=['accuracy'],
)

model.fit(
    ds_train,
    epochs=12,
    validation_data=ds_test,
)

```

You will get the output below. The time to run may be different

```

469/469 [=====] - ETA: 6s - loss: 0.1583 - accuracy: 0.9521
2021-10-28 19:29:43.839236: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enabled.
469/469 [=====] - 11s 21ms/step - loss: 0.1583 - accuracy: 0.9521 - val_loss: 0.0471 - val_accuracy: 0.9846
Epoch 2/12
469/469 [=====] - 10s 26ms/step - loss: 0.0438 - accuracy: 0.9867 - val_loss: 0.0423 - val_accuracy: 0.9866
Epoch 3/12
469/469 [=====] - 9s 20ms/step - loss: 0.0278 - accuracy: 0.9911 - val_loss: 0.0400 - val_accuracy: 0.9874
Epoch 4/12
469/469 [=====] - 9s 20ms/step - loss: 0.0180 - accuracy: 0.9944 - val_loss: 0.0311 - val_accuracy: 0.9908
Epoch 5/12
469/469 [=====] - 10s 26ms/step - loss: 0.0134 - accuracy: 0.9957 - val_loss: 0.0345 - val_accuracy: 0.9890
Epoch 6/12
469/469 [=====] - 9s 20ms/step - loss: 0.0097 - accuracy: 0.9968 - val_loss: 0.0424 - val_accuracy: 0.9867
Epoch 7/12
469/469 [=====] - 9s 20ms/step - loss: 0.0080 - accuracy: 0.9973 - val_loss: 0.0365 - val_accuracy: 0.9905
Epoch 8/12
469/469 [=====] - 10s 26ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0351 - val_accuracy: 0.9898
Epoch 9/12
469/469 [=====] - 10s 26ms/step - loss: 0.0062 - accuracy: 0.9979 - val_loss: 0.0519 - val_accuracy: 0.9878
Epoch 10/12
469/469 [=====] - 9s 20ms/step - loss: 0.0056 - accuracy: 0.9981 - val_loss: 0.0545 - val_accuracy: 0.9876
Epoch 11/12
469/469 [=====] - 10s 26ms/step - loss: 0.0046 - accuracy: 0.9983 - val_loss: 0.0406 - val_accuracy: 0.9900
Epoch 12/12
469/469 [=====] - 9s 20ms/step - loss: 0.0042 - accuracy: 0.9986 - val_loss: 0.0544 - val_accuracy: 0.9882
CPU times: user 60.3 s, sys: 29.2 s, total: 1min 59s
Wall time: 1min 55s
Out[10]: <keras.callbacks.History at 0x2c0d8dbb>

```

It took a total of 1min 57s second to run 12 epochs with 128 batch size.

You can use any other code editor or

any other way to do the test benchmark also.

Conclusion

As you can see Tensorflow is successfully installed in our system and we are able to run some code also.

If you have any questions, recommendations, or critiques, I can be reached via [Twitter](#) or via my [mail](#). Feel free to reach out to me.

References:

- <https://blog.tensorflow.org/2020/11/accelerating-tensorflow-performance-on-mac.html>
- <https://developer.apple.com/metal/tensorflow-plugin/>
- https://github.com/apple/tensorflow_macos



354



22

