Prabhat Kumar Sahu Follow
Jun 13, 2021 · 4 min read · Listen

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

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

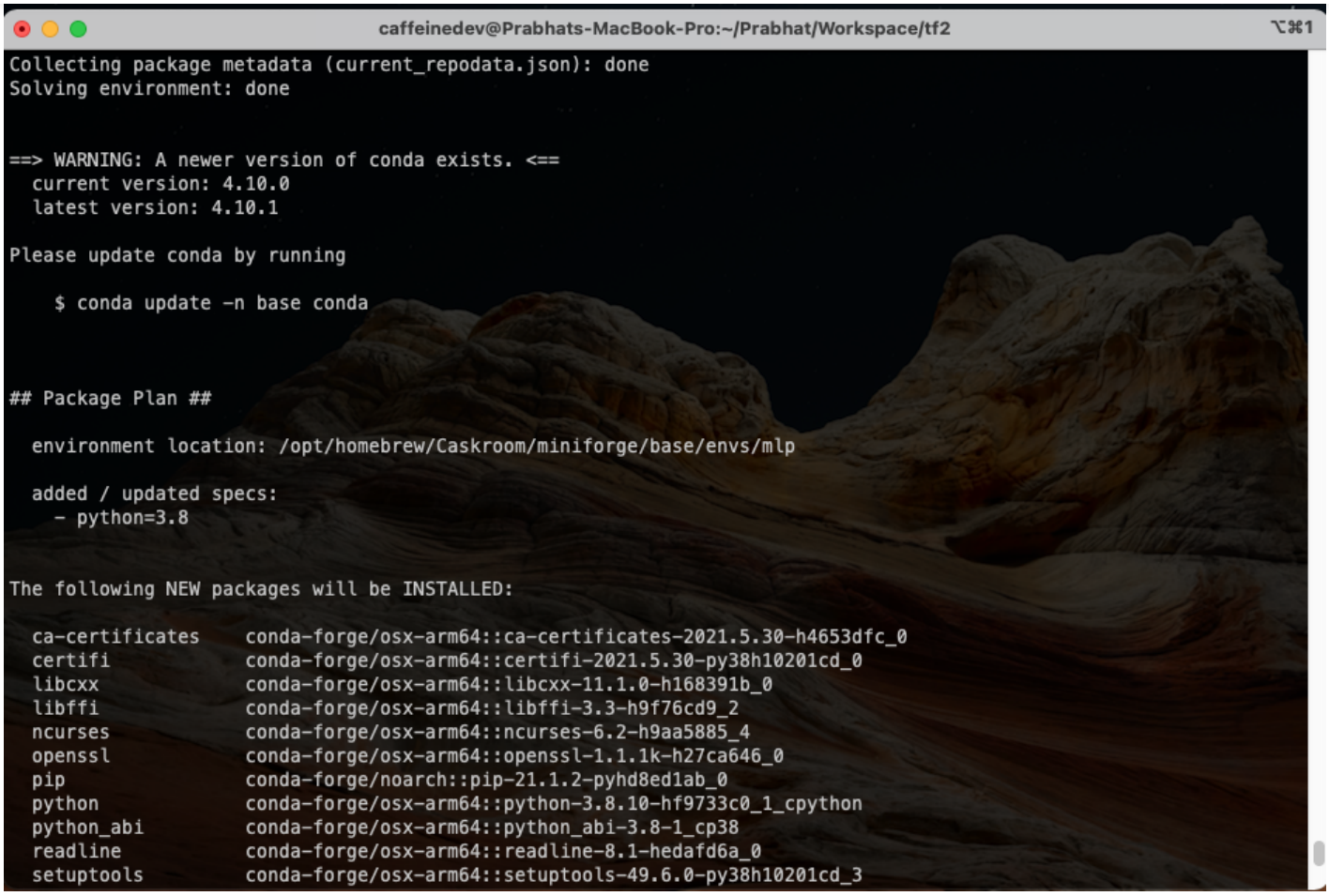
Install miniforge for arm64 (Apple Silicon) from miniforge GitHub.

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



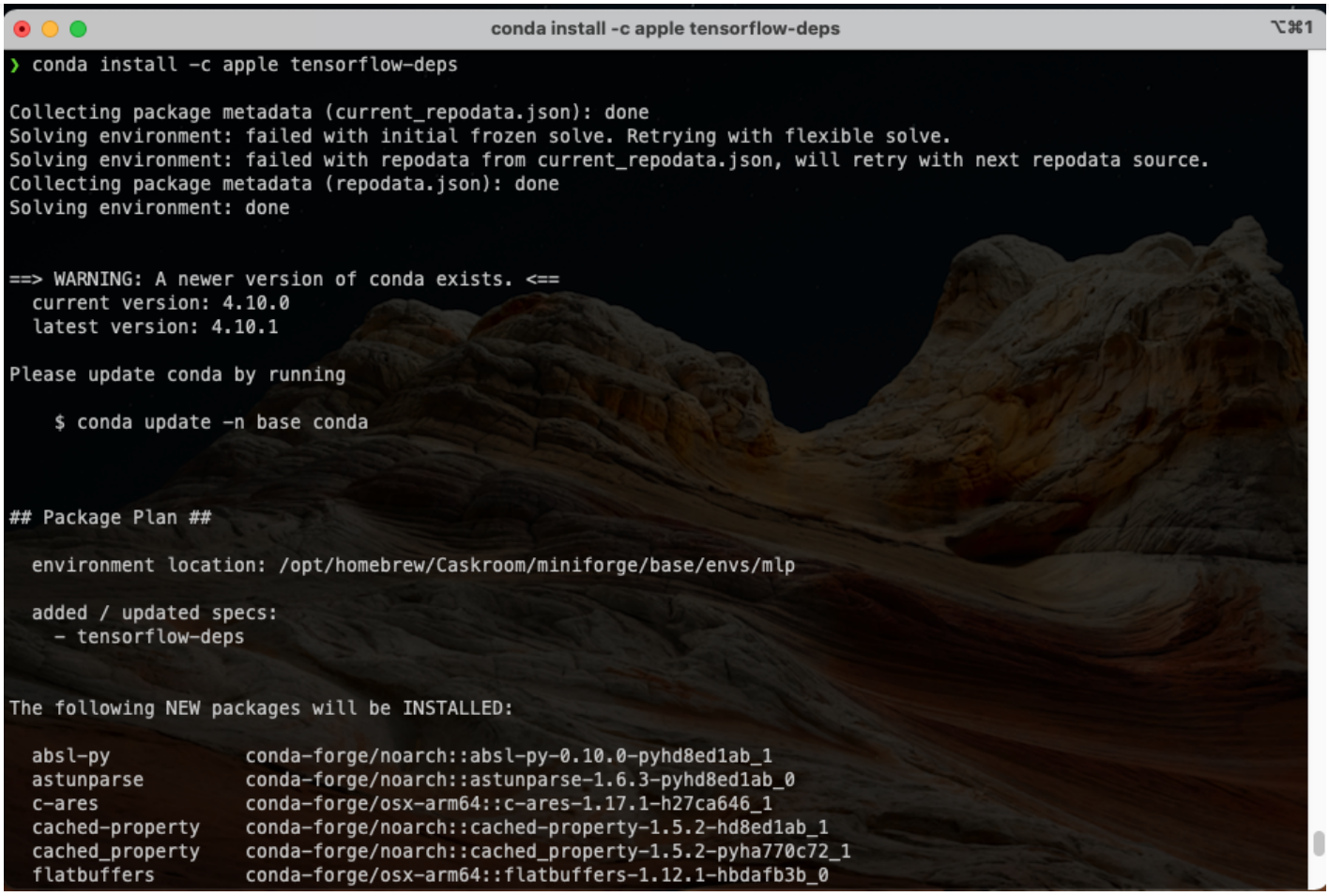
And activate it by running

```
conda activate mlp
```

Step4: Installing Tensorflow-MacOS

Install the Tensorflow dependencies:

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



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

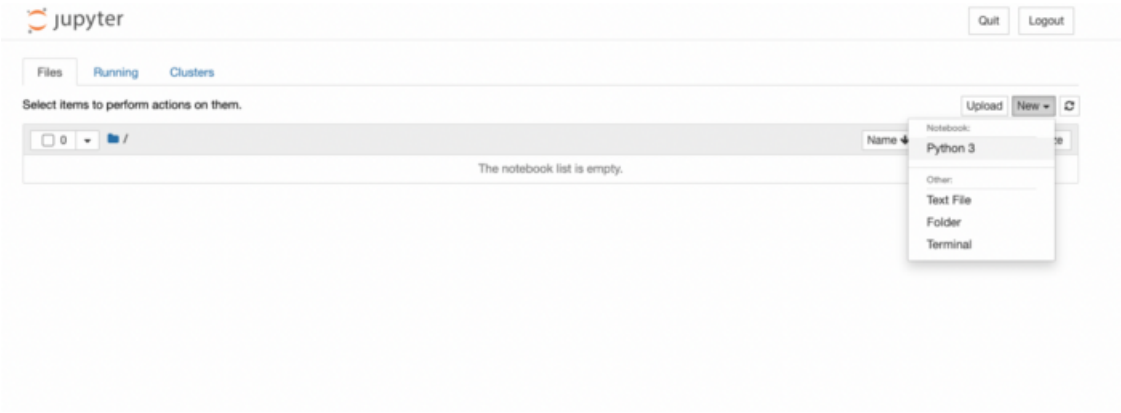
Make sure your conda environment is activated.

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

```
jupyter notebook
```

It will open a browser window

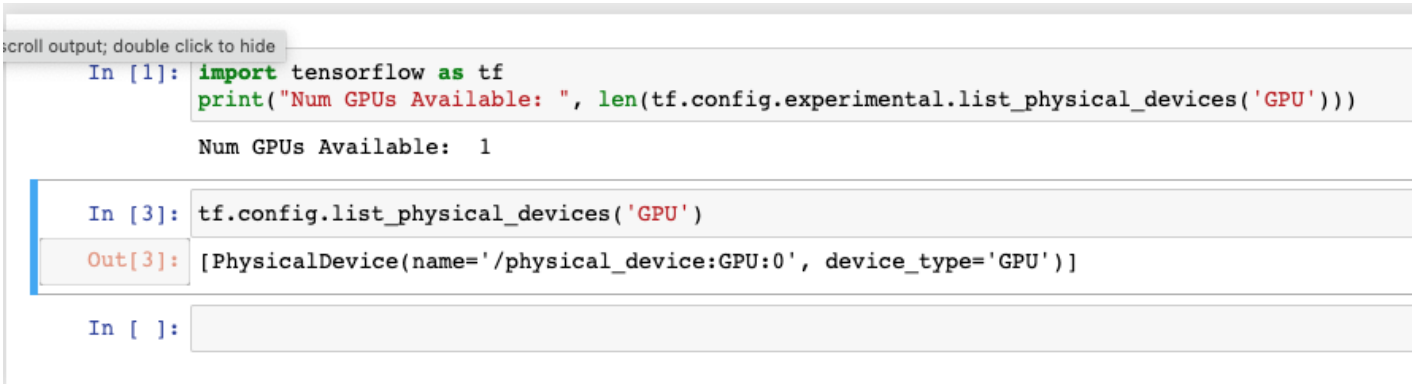
Create a new python3 notebook



Let's first import TensorFlow and check

```
import tensorflow as tf

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



Let's run a benchmark

I got the code snippet from

```
%%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,
    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,
```


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

```
469/469 [==============================] - ETA: 0s - 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 20ms/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 20ms/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 20ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0351 - val_accuracy: 0.9898
Epoch 9/12
469/469 [==============================] - 10s 20ms/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: 0.9870
Epoch 11/12
469/469 [==============================] - 10s 20ms/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 40.3 s, sys: 29.2 s, total: 1min 9s
Wall time: 1min 55s
Out[10]: <keras.callbacks.History at 0x2c0d8dbb0>
```

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