DeepLabCut analyze_videos error solved: “Failed to get convolution algorithm. This is probably because cuDNN failed to initialize, so try looking to see if a warning log message was printed above.”

We recently had an issue with Deeplabcut that was pointing to some mismatch version of the underlying CUDA and cuDNN libraries, but after trying quite a few solutions, believe that it was a memory usage error that was fixed by cleaning up any changes we had made with cuDNN driver installation, extra pip installations, and creation of other conda environments, followed by a machine restart. The standard tools for checking processes on Nvidia gpu in Windows do not give the necessary output, so this was a difficult problem to analyze.

The main punchline is that the current Deeplabcut install for Windows10 that installs tensorflow-gpu=1.13.1 with cuDNN=7 and for CUDA 10 works for us, even though we are using CUDA 9 (though downgrading cuDNN to the CUDA 9 build also works). Just follow their instructions.

Our **original** setup was the following:
OS: Windows 10
Graphics card: Nvidia GeForce RTX 2070
Nvidia driver: 436.48
tensorflow: tensorflow-gpu 1.12.0, installed via conda
cuDNN: 7.6.0, installed by conda
CUDA: CUDA 9.0

Our **successful** setup that followed these instructions is the following:
OS: Windows 10
Graphics card: Nvidia GeForce RTX 2070
Nvidia driver: 436.48
tensorflow: tensorflow-gpu 1.13.0, installed via conda
cuDNN: 7.6.0, installed by conda (both the CUDA 10 and CUDA 9 build worked for us)
CUDA: CUDA 9.0

The problem that we had was that running the “analyze_videos” command was producing errors that ended with a “Failed to get convolution algorithm. This is probably because cuDNN failed to initialize, so try looking to see if a warning log message was printed above.” This was a similar issue that others had seen, and the main suggestion was to downgrade to tensorflow-gpu 1.8.

Utilizing Anaconda3 to install tensorflow-gpu 1.8 gave many conflicts, even in a brand new environment, so we attempted to use a pip install of tensorflow-gpu 1.8. Install worked, but we ran into a new error: “could not create cudnn handle: CUDNN_STATUS_NOT_INITIALIZED.” At some point I found a git issues repo or stackoverflow post that mentioned how this problem came from pip installs of tensorflow 1.8, but can no longer locate it.

We tried a few more test setups of different tensorflow versions, all with the same results. At one point I even came across the following error: “Blas GEMM launch failed”. After doing some digging it sounded like these could also point to the GPU running out of memory, so I tracked the GPU usage through Windows task manager, and saw that the memory spiked when running the troublesome command. After doing a machine restart and reverting back to the original configuration from the DeepLabcut instructions, things worked fine.

Unfortunately, I couldn’t think of a way to monitor GPU usage on a mac beyond that. The method on a Mac or Linux machine would be to use the “nvidia-smi” utility to see which processes are running in the background. I had run into issues on a Linux machine when running tensorflow in a Jupyter notebook where it was rather greedy, and the

process had to be killed before freeing up enough memory for another task to run. However, on a Windows machine this capability is pretty bare

Background/extra stuff.

Deeplabcut is a video classification software that uses a TensorFlow backend to run a deep neural net for image classification. Utilizing TensorFlow with GPU capability often requires the correct versioning of two separate drivers: CUDA, the base libraries for GPU computation, and cuDNN a library for running deep neural nets on the GPU. The version compatibility for these pieces (Tensorflow-gpu, CUDA, cuDNN, physical graphic card, OS, and possibly Nvidia driver) all have to be just right to avoid errors. Successful configurations up until recently were documented

Deeplabcut provides installation instructions to create a conda environment that should give a stable configuration (tensorflow-gpu 1.13.0, Cuda 10.0, CuDNN 7.*, and any compatible driver). **I**n order to run an alternative piece of software (kilosort) we required Cuda 9.0 on this machine**.** Our current graphics card is capable of supporting Cuda 10.0 (it actually came installed), but rather than trying to manage two version of CUDA on windows (performing on Linux or Mac might be more straightforward), we stuck with just CUDA 9 and tried to work with it.

We have successfully used deepLabCut on our other machine in the lab with a slightly different configuration that was installed by a previous lab member: seems to be a pip install of tensorflow-gpu 1.8.0, CuDNN 7.0.5, and Cuda 9.0, Nvidia GeForce RTX 2060 with driver 417.71.