Sparce Conv を使ってみる

現在の物体検出(特に，3D オブジェクト検出)では，スパースな畳み込みニューラルネットワークが使用されているようである PV-RCNN：3D オブジェクト検出のためのポイントボクセル機能セットの抽象化．そこで，自分も同様の畳み込み層を使用したネットワークを構築したいと思ったので，本日は自分の VSCode 上で GitHub 上に公開されている Sparce Convolution が使用できるようにする．

Sparse Convolutional Networks とは？

Sparse Convolutional Networks とは，3D 点群データなどの「疎な」データを効率的に処理することを目的として改良された畳み込み層を有するニューラルネットワークである 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks．

このネットワークの `pyTorch` 実装は，spconv で公開されていた．
さらに，上記の論文で使用されたネットワークの実装はSparseConvNetで公開されていた．

SPConv パッケージダウンロード

注意！
この Qiita 記事 によると，こちらの `spconv` を使用した方が良いようであった(こちらの方が上述のリポジトリより最近更新されていた).

Docker for windows から GPU を使用

`spconv` のリポジトリを見る限り，`windows` のサポートは終了したようなので，仮想環境で `linux` を立ち上げて、そのうえで使用できないかを検討する.

1. まず，Windows 上の Docker で GPU を使おう を参考に以下のコマンドを `docker for windows` 起動状態で実行する.

```
docker run --rm -it --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark -numbodies=512000
```

すると，以下のようなエラーが発生した.

```
$ docker: Error response from daemon: OCI runtime create failed: container_linux.go:370:
starting container process caused: process_linux.go:459: container init caused:
Running hook #0::error running hook: exit status 1,stdout: ,stderr: nvidia-container-cli:
initialization error: driver error: failed to process request: unknown.
```

このエラーについて検索したところ，Docker - OCI runtime create failed エラー ("process_linux.go:449: container init caused ")で `cuda` のバージョンが合っていないためであると説明されていた．そこで，power-shell 上で `nvidia-smi` を実行する．

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 461.92       Driver Version: 461.92       CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce GTX 166... WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   54C    P8    19W /  N/A |    153MiB /  6144MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

ここで，`CUDA Version` の欄が 11.0 以上あれば問題ないそうだが、自分の PC には `Version: 11.2` がインストールされているためエラーの原因は別にあるようであった．

次にヒットしたWSL2 で CUDA を利用するなら自動アップデートに気を付けようという話とその対処法 というサイトには，『自動アップデートされたあと nvidia-driver のバージョンが wsl2 から GPU を扱うためには適していないバージョンのものに勝手に置き換わってしまう』と書かれており，その対処法として `Display Driver` を再インストールするものが紹介されていました．

そこで，次にこの方法を試します．

1. `wsl --shutdown` のコマンドを実行し，`wsl` をシャットダウンする．
2. DirectML サポートを含む WSL 上の CUDA 用の NVIDIA ドライバーから，`GEFORCE` 用のドライバーをダウンロードする．
3. ドライバーをインストールし再起動する．
4. `wsl` はシャットダウン状態なので power-shell 上で `wsl` コマンドを実行し，wsl を起動する．
5. wsl が起動できたかの確認として，`docker run -d -p 80:80 docker/getting-started` を実行する．

ここで，もし以下のようなエラーが発生した場合：

```
docker: Error response from daemon: Head
https://registry-1.docker.io/v2/docker/getting-started/manifests/latest:
unauthorized: incorrect username or password.
```

このエラーが発生するのは，docker login での Sign in について によると docker にログインできていないことが原因のようなので，

```
$ docker login
```

コマンドを実行して，docker にログインすることで動作させることができる．
これにより，`docker` と `wsl` が動作していることがわかったので，満を持して以下のコマンドを実行する．

```
docker run --rm -it --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark -numbodies=512000
```

が，やはり同じエラーが発生した．

次にヒットしたサイトCan not use nvidia-docker.には，NIDIA のドライバー version が古すぎるという指摘がありました.

そこで，再度 `GEFORCE Experience` を実行したところ更新があり，`nvidia-smi` で表示されていた `CUDA Version` などに変化がありました.

```
Mon Apr 12 17:08:15 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.14       Driver Version: 470.14       CUDA Version: 11.3     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A    0C    P8    12W /  N/A |    153MiB /  6144MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

ここで，そもそものインストールコマンドを疑い始めました．そこで，Google が開発している 機械学習用パッケージである `Tensorflow` の `docker image` で試してみることにしました．これは，機械学習の際に `GPU` を `docker image` 上から呼べるようにしたものであり，Docker にインストール方法が全て記載されています．

その方法は簡単で，`power-sehll` 上で以下のコマンドを実行するだけです．

```
$ docker run -it --rm --runtime=nvidia tensorflow/tensorflow:latest-gpu python
```

しかし，今度は以下のような別のエラーが発生しました．

```
$ docker: Error response from daemon: Unknown runtime specified nvidia.
```

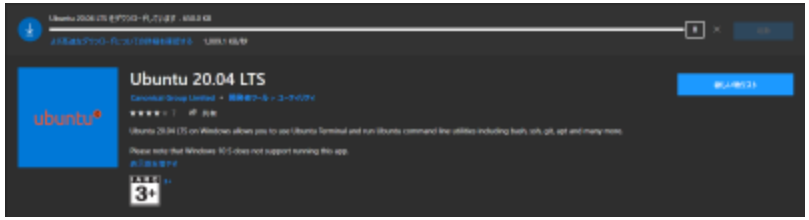
そして，これについて検索したところ，`NVIDIA Docker` や `NVIDIA Docker2` といったものがあることを知り，それぞれの関係がわからなかったので，NVIDIA Docker って今どうなってるの？ (20.09 版) を参考に何を使えば良いのかを確認した．このサイトは，NVIDIA に勤務されている方が書かれており，とても見やすかった．

WSL + docker + cuda

先ほどのサイトによると，NVIDIA 製 GPU を Docker 上で動作させるための最新パッケージは，`NVIDIA Container Toolkit` であり `NVIDIA Docker2` パッケージでインストール可能であるらしい．また，環境構築方法は ついに WSL2+docker+GPU を動かせるようになったらしいので試してみる によると，`Docker for windows` からではなく，`WSL Ubuntu 20.04 LTS` 上に `Docker` をインストールするようである．そこで，こちらの方法を試してみる．

1. WSL Ubuntu 20.04 LST のセットアップ

1. Microsoft Store より `Ubuntu 20.04 LTS` をインストールする．



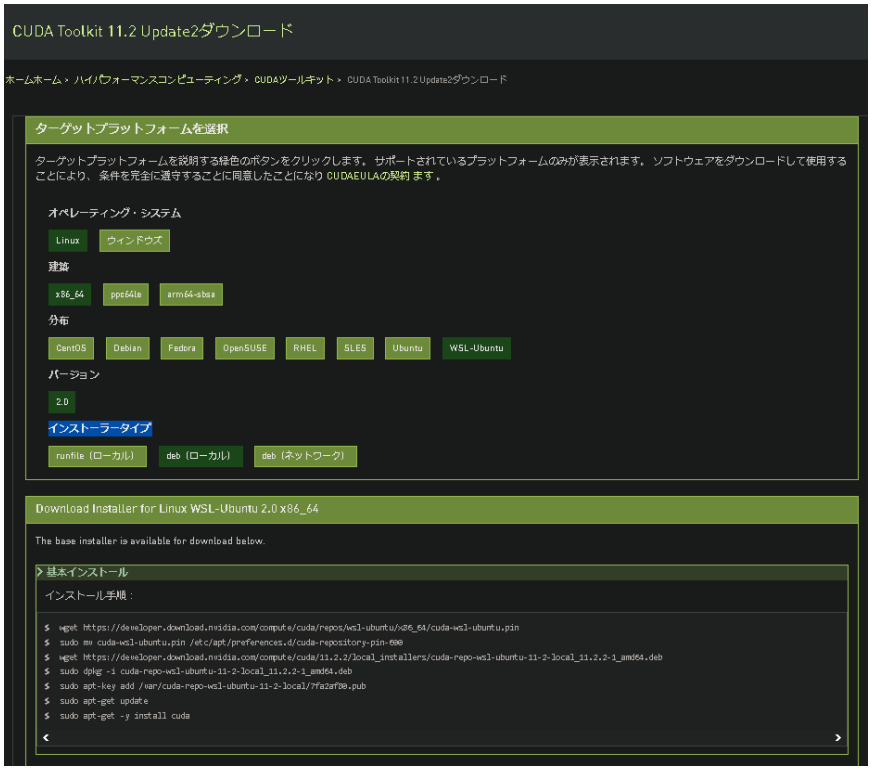
2. `Ubuntu 20.04 LTS` を立ち上げ，`username` と `passward` を設定する．
3. 以下の 2 つのコマンドを実行し，パッケージを最新のものにする．

```
$ sudo apt update
$ sudo apt upgrade
```

4. `uname -r` を実行し，`WSL2` の正しいカーネルで動作しているかを確認する．ここで，`4.19.121-microsoft-WSL2-standard` 以上の version が表示されれば成功．

2. NVIDIA ドライバをインストール

NVIDIA Docker って今どうなってるの？(20.09 版) によると，次に NVIDIA ドライバをインストールするらしい．そこで，CUDA Toolkit の Web サイト から WSL-Ubuntu 用のドライバをインストールする．



図に示したようにダウンロード項目を選択していくと，以下のようなコマンドが表示される．

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin
$ sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget https://developer.download.nvidia.com/compute/cuda/11.2.2/local_installers/cuda-repo-wsl-ubuntu-11-2-local_11.2.2-1_amd64.deb
$ sudo dpkg -i cuda-repo-wsl-ubuntu-11-2-local_11.2.2-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-wsl-ubuntu-11-2-local/7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

この最後のコマンドを `sudo apt-get -y install cuda-drivers` とすることで，最新のドライバだけが綺麗にインストールされる．

しかし，`cuda-drivers` に設定するとエラーが発生し，インストールされなかった．そこで，もとのままコマンドを実行するとインストールできた．

3. Docker のインストール

まず，以下のコマンドで `Docker` 自体をインストールする．

```
$ curl https://get.docker.com | sh
```

次に，以下のコマンドで `Ubuntu` を立ち上げた時に `Docker` が自動的に起動するようにする．

```
sudo systemctl start docker && sudo systemctl enable docker
```

しかし，実行してみると次のようなエラーが発生した．

```
System has not been booted with systemd as init system (PID 1). Can't operate.
Failed to connect to bus: Host is down
```

このエラーについて 【WSL2】systemctl が動かない問題をきちんと解決する では，『普通 Linux が起動するときに，最初のプロセスとして `systemd` が起動し，すべてのプロセスの親として振る舞う(PID 1)．しかし，`WSL` では `init` が親として振る舞う(PID 1)』ことが原因であると書かれていた．そこで，自分の `WSL` が同じ状況にあるか，確認する．

そのために，以下のコマンドを実行する

```
ps aux
```

以下は，コマンドを実行した結果である．

```
miki@DESKTOP-Q123T6P:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    892   584 ?        Sl   Apr12   0:00 /init
root        47  0.0  0.0    892    84 ?        Ss   Apr12   0:00 /init
root        48  0.0  0.0    892    84 ?        S    Apr12   0:00 /init
miki        49  0.0  0.0  10052  5076 pts/0    Ss   Apr12   0:00 -bash
miki     14484  0.0  0.0  10616  3216 pts/0    R+   07:07   0:00 ps aux
```

上記の結果より，自分の `WSL` でも `PID 1` は `/init` が走っていることがわかった．よって，先ほどの記事で紹介されていた方法を使用して，`systemd` を `PID 1` に持って来る．

3.1. systemd を PID 1 に

`WSL2` を使用している場合，この問題を解決するには以下のリポジトリを使用することができる．

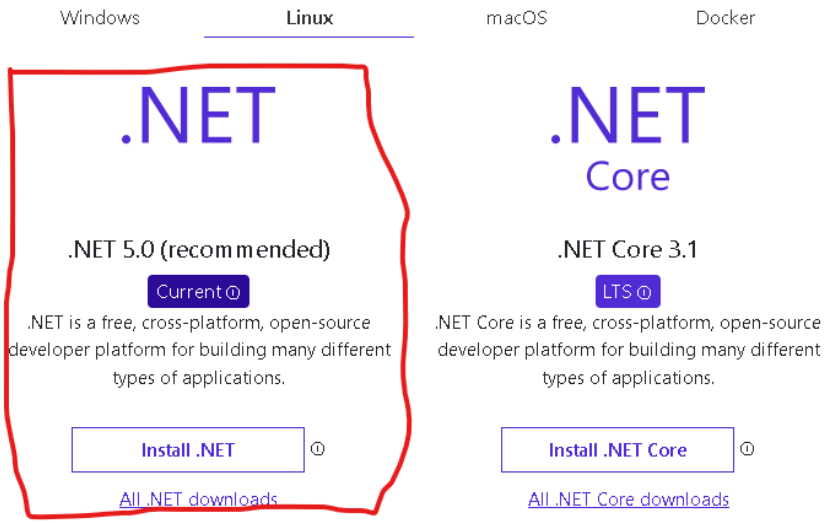
https://github.com/arkane-systems/genie

リポジトリの `Readme` によると，Linux-OS が Debian 系の場合，
daemonize, dbus, dotnet-runtime-5.0, gawk, libc6, libstdc++6, policykit-1, systemd, systemd-container.
をインストールする必要があるとのこと．その中でも，dotnet-runtime-5.0 はインストールに癖があるとのことなので，まず dotnet-runtime-5.0 からインストールする．

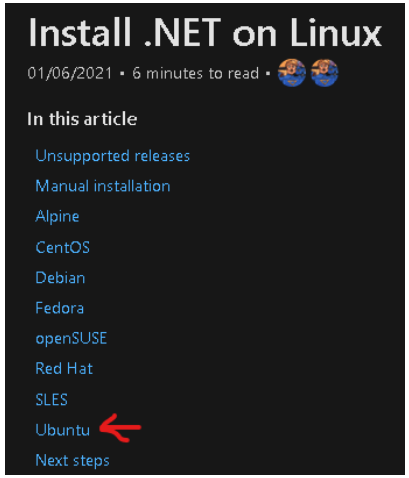
dotnet-runtime-5.0 は、次のインストール手順に従う必要がある。

https://dotnet.microsoft.com/download/

サイトにアクセスすると，ダウンロードする OS などが聞かれる．



先ほどの画像の `Install .NET` を選択すると，以下のようなディストリビューションごとにダウンロード方法を説明しているサイトに飛ぶ.



`Ubuntu` を選択すると，インストールしたい `.Net` を使用している OS のバージョンで動作可能かを表した表があり，そこで自分の Ubuntu のバージョン(今回は 20.04LTS) で `.Net` が動かせるかを確認する．動作可能なら，下の Install .NET on Ubuntu に飛ぶ.

飛んだ先は、OS のバージョン毎にインストール方法が説明されている．今回は `Ubuntu 20.04 LTS` を使用しているので，その部分のインストールコマンドに従う.

本日は、ここまでとして以降の作業は明日行う.