NVIDIA DEEP LEARNING INSTITUTE

第100回お試しアカウント付き並列プログラミング講習会
「REEDBUSH スパコンを用いたGPUディープラーニング入門」

ハンズオン#1: REEDBUSH-Hでのディープラーニング

山崎和博
NVIDIA, ディープラーニング ソリューションアーキテクト

AGENDA

ハンズオン#1のテーマ: 画像分類

今日の環境とスクリプトの書き方おさらい

タスク#1: テストジョブ投入

タスク#2: シングルGPUでの学習ジョブ

ハンズオン#1のゴール

バッチジョブの流し方に慣れる

- Reedbushのディレクトリ構成などを把握する
- Reedbush上でジョブを流す方法を把握する
- シングルGPUでの学習方法を理解する

ハンズオン#1のテーマ: 画像分類

画像分類問題

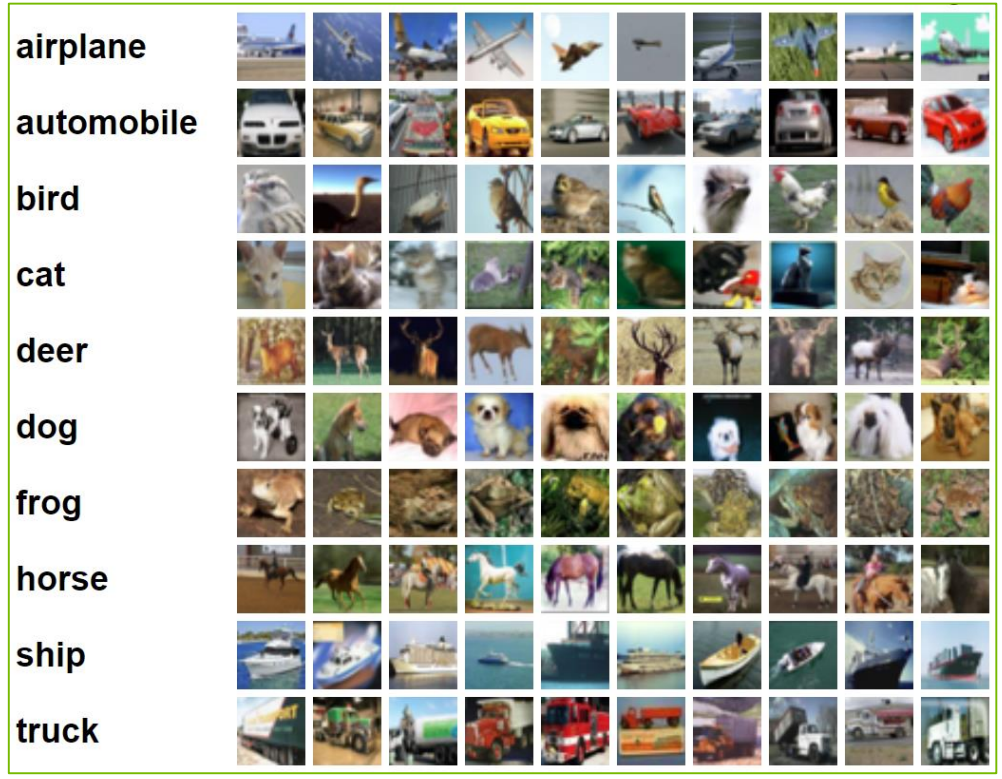
画像が何か？を当てる問題

- 1枚の画像が与えられたときに、それが「何か」を当てる問題
  - 例: 数字が写った画像に対して、0-9までのラベルを割り振る
  - 画像認識問題とも呼ばれる
- 「どこに」、「どんな形状で」写っているかは問わない
  - 場所まで当てるタスクは、物体検出
  - 形状まで当てるタスクは、セマンティックセグメンテーション

本日のデータセット

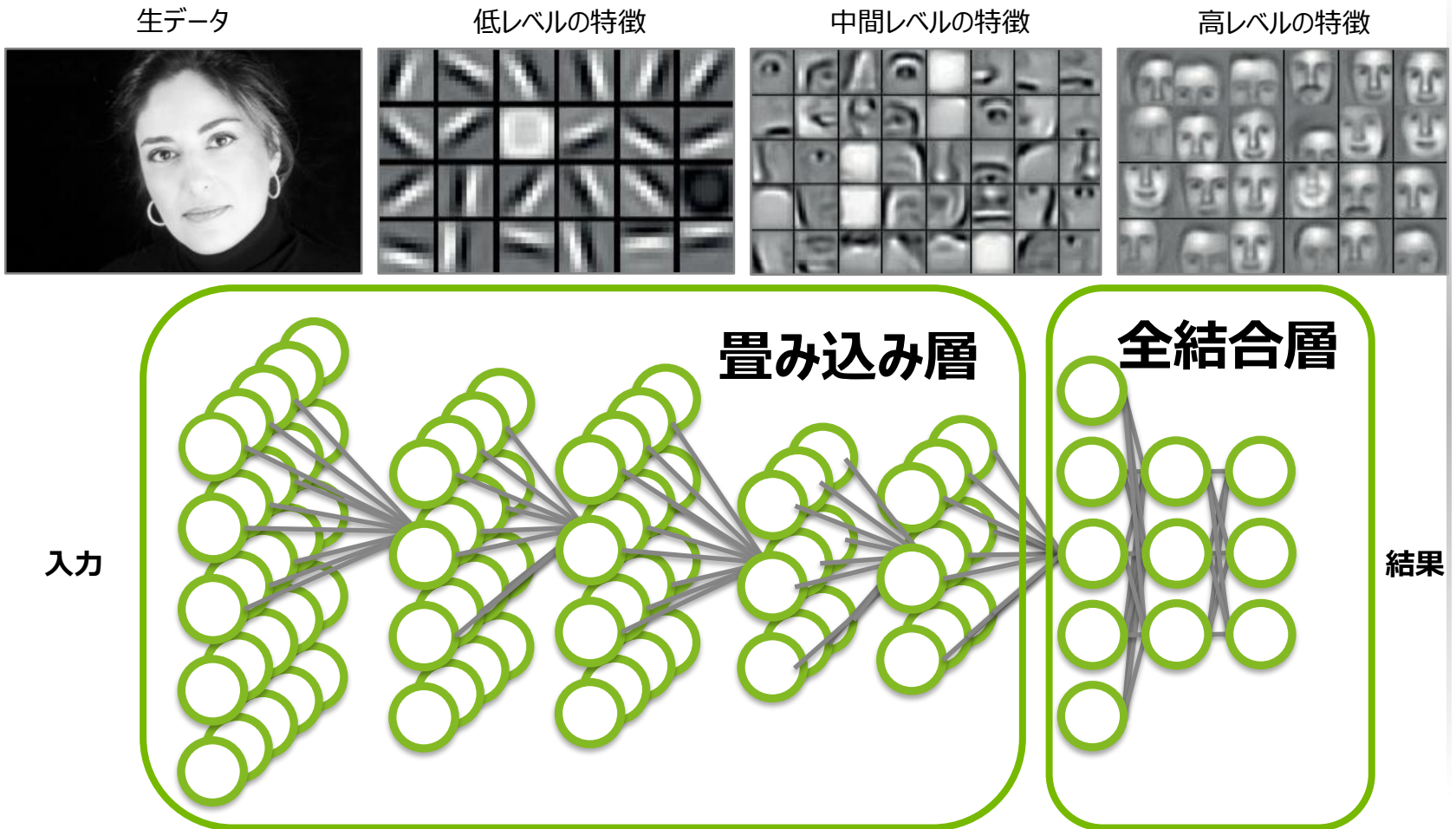
10クラス分類問題

- CIFAR-10 データセット
  https://www.cs.toronto.edu/~kriz/cifar.html
- サイズ：32x32、RGBカラー、
  ピクセル値 0 - 255
  トレーニング用データ：5万(5000/class)
  テスト用データ：1万
- 入力ベクタサイズ：1024 (= 32 x 32)
- 出力は10種類のクラス



畳み込みニューラルネットワーク(CNN)

画像分類でよく用いられるネットワーク

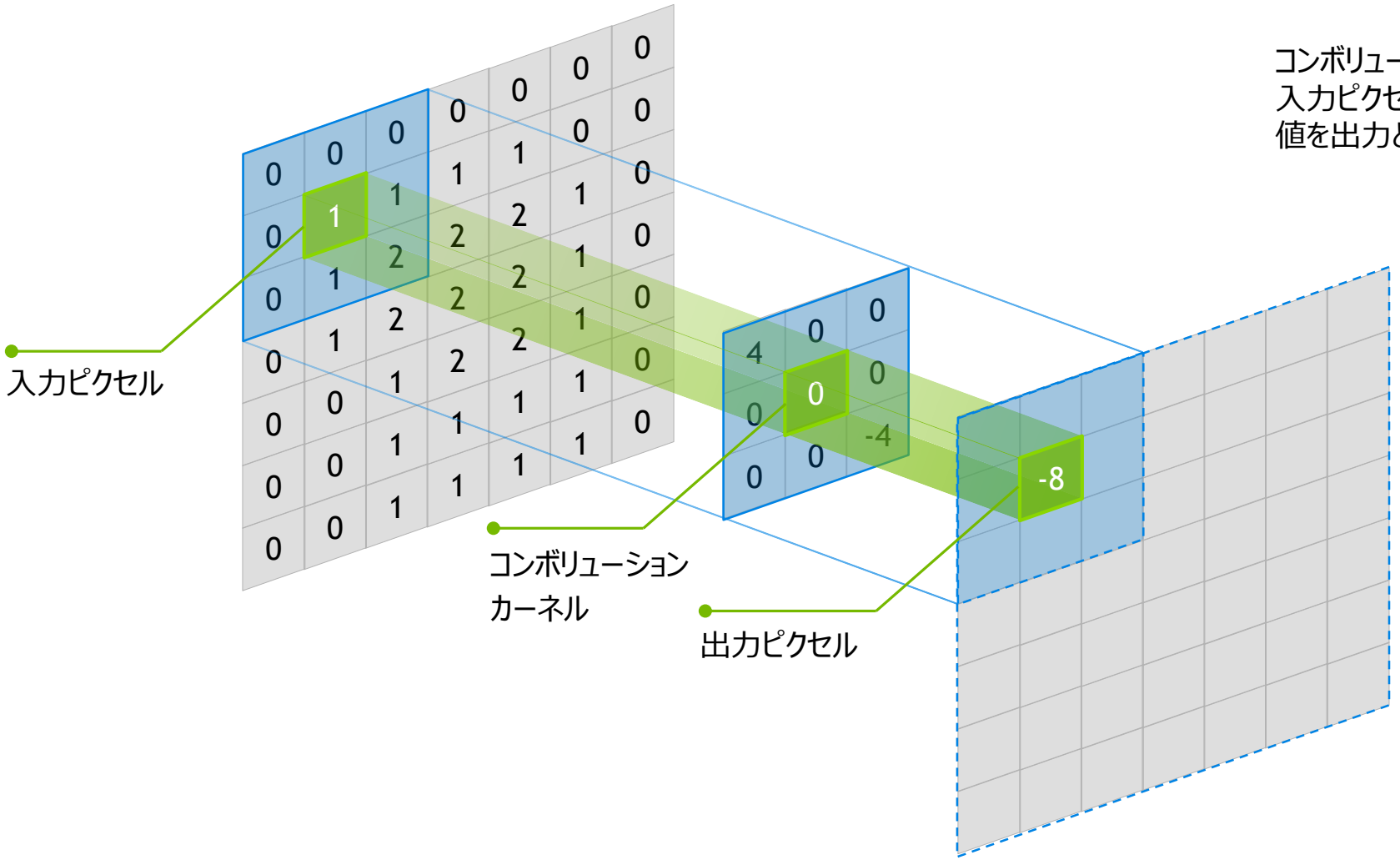


アプリケーションの構成要素例:

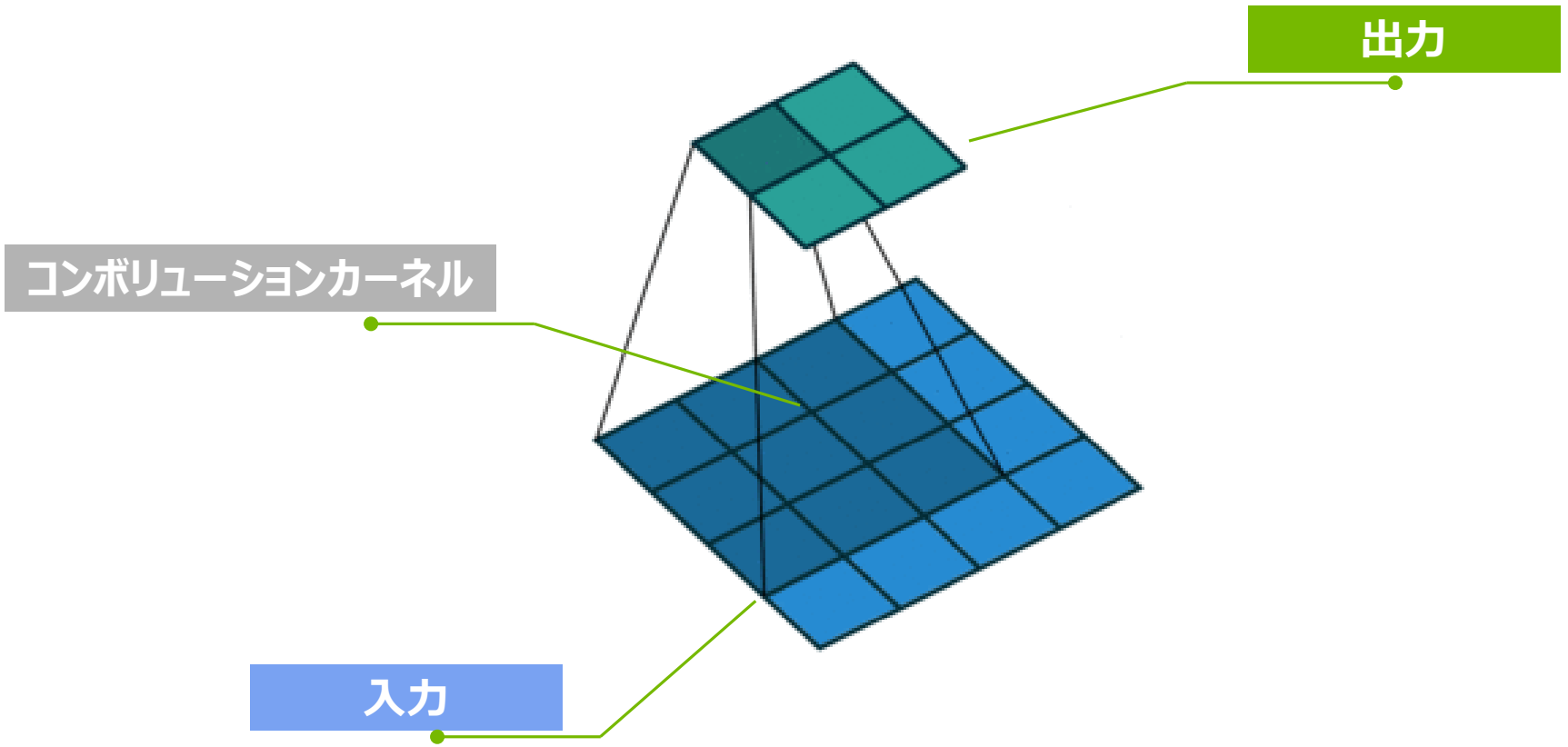
タスクの目的
顔の同定
トレーニングデータ
1千万-1億 のイメージ
ネットワークアーキテクチャ
十から数百のレイヤー
10億のパラメータ
学習アルゴリズム
~30 Exaflops
1-30 日(1GPU)

畳込み層(CONVOLUTIONAL LAYER)

コンボリューションカーネルの係数と、
入力ピクセルを掛け、足し合わせた
値を出力とする。



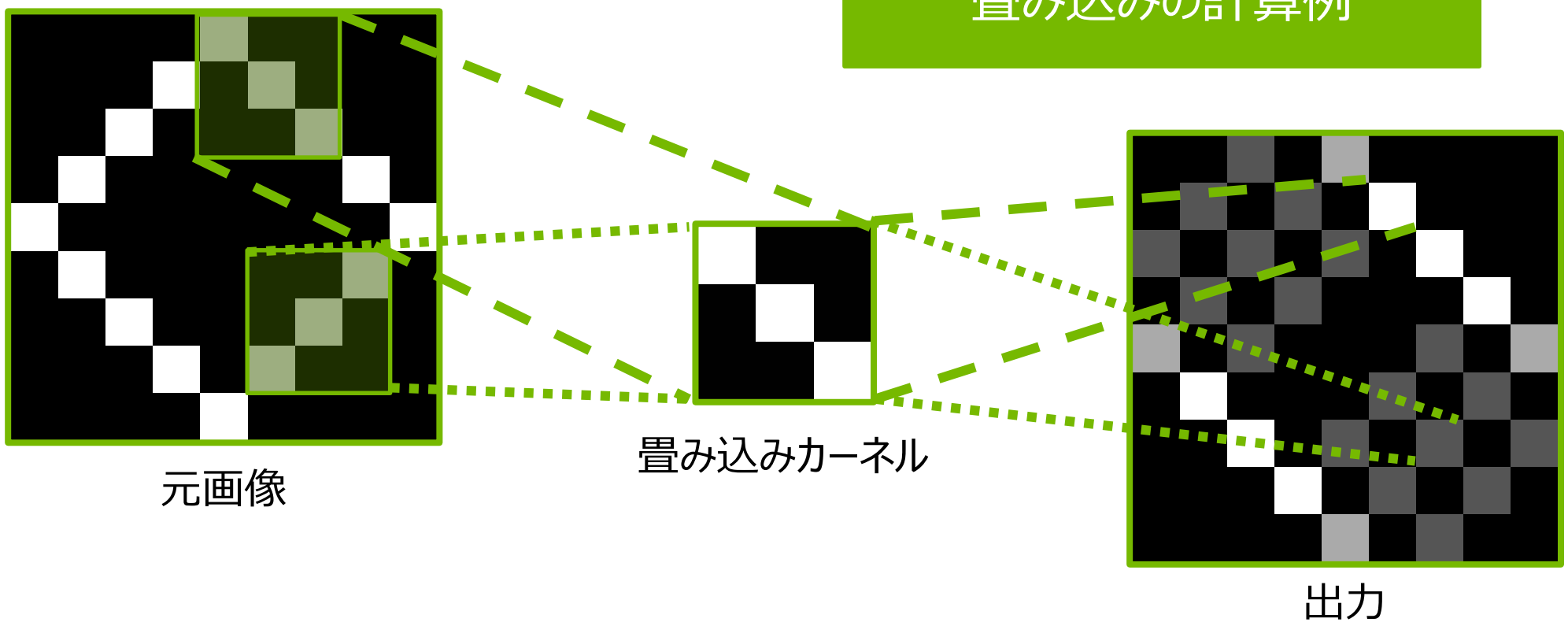
CNN: 畳み込み層



vdumoulin/conv_arithmetic, https://github.com/vdumoulin/conv_arithmetic

CNN: 畳み込み層

畳み込みの計算例

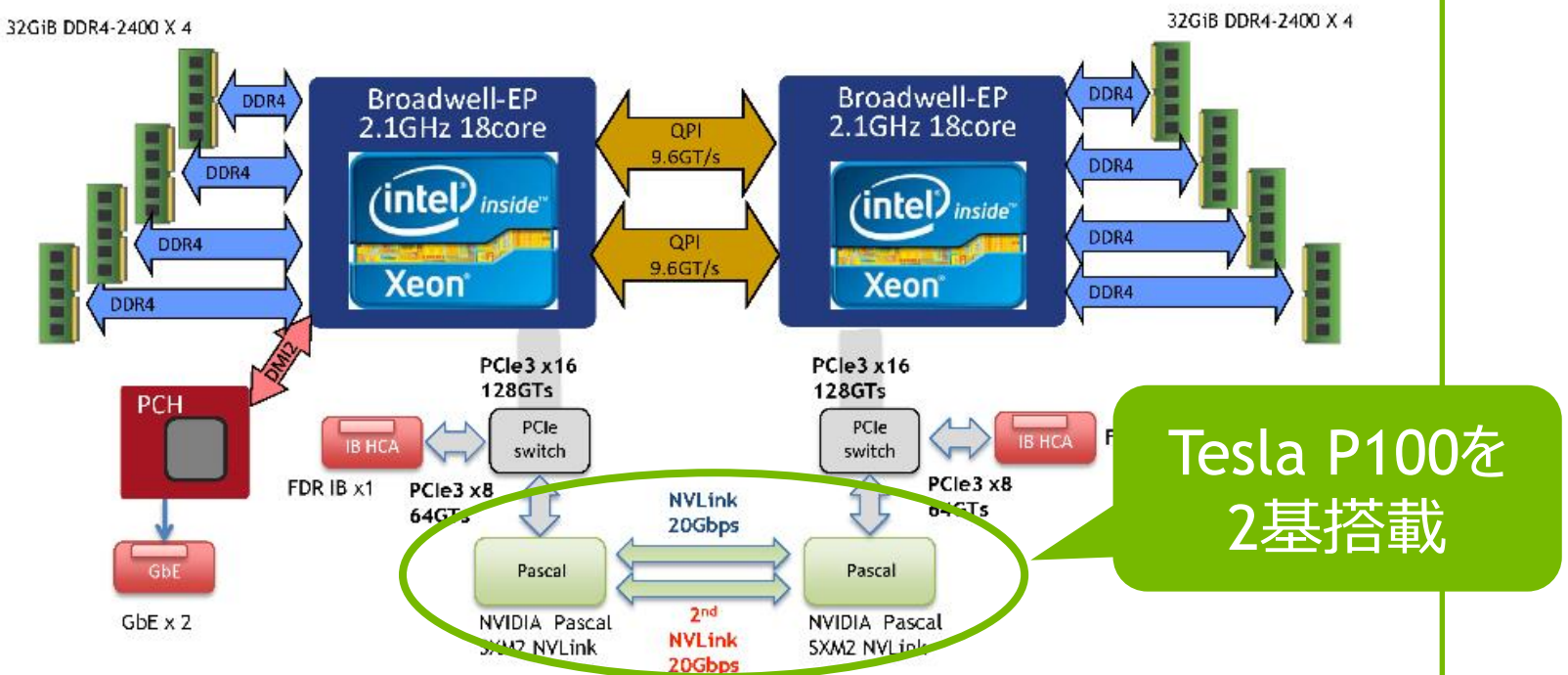


今日の環境とスクリプトの書き方おさらい

REEDBUSH-H

GPU搭載スーパーコンピュータ

計算ノードブロック図



https://www.cc.u-tokyo.ac.jp/system/reedbush/reedbush_intro.html より

ジョブの実行

- 実行したい内容をスクリプトにまとめ、バッチジョブとして実行
  - ジョブは指定したキューに追加され、順番が来たら実行される
  - キューに割り当てられているノード数を超えてジョブが投入されたら待つ
- 本講習会では **h-tutorial** を指定
  - ノード数の指定もできるが、講習会中はジョブあたり最大2ノードが上限
- 講習会ユーザのグループ名として **gt00** を指定

ジョブスクリプト

実行条件指定

「#PBS」から始まる行が条件指定

実行条件はスクリプトの先頭行に記述
（コマンドラインオプションとしても指定可）

```
#!/bin/bash
#PBS -q (対象ジョブキュー名)
#PBS -l (利用ノード数など)
#PBS -W (利用グループ名)
#PBS -l (実行時間制限)
#PBS -N (ジョブ名)

...
```

ジョブスクリプト

実行条件指定

```
#!/bin/bash
#PBS -q (対象ジョブキュー名)
#PBS -l (利用ノード数など)
#PBS -W (利用グループ名)
#PBS -l (実行時間制限)
#PBS -N (ジョブ名)

...
```

利用ノードなどのパラメータは、以下をコロン「:」区切りで指定

- select ノード数
- mpiprocs mpiプロセス数/ノード
- ompthreads スレッド数/プロセス

例) `-l select=1:mpiprocs=1`

ジョブスクリプト

実行環境の指定

```
#!/bin/bash
#PBS （省略）

cd ${PBS_O_WORKDIR}
. /etc/profile.d/modules.sh

...
```

「PBS_O_WORKDIR」で
作業ディレクトリを参照

「/etc/profile.d/modules.sh」
で環境変数などを設定

ジョブスクリプト

必要モジュールの読み込みなど

```
#!/bin/bash
#PBS （省略）
...

module list

module load cuda9/9.0.176 ...

module avail
...
```

現在ロード済み
モジュールを表示

モジュールの
追加読み込み

Reedbushで利用可能
なモジュールを表示

タスク#1: テストジョブ投入

ジョブを投入する

動作環境をジョブ経由で確認

以下の作業を完了させる。

1. 作業ディレクトリ(/lustre/gt00/**xxxxxx**)へ移動
    1. xxxxxxは各自のアカウントで、cdwコマンドを使っても移動可能
2. テンプレートスクリプトを作業ディレクトリへコピー
    1. /lustre/gt00/share/lecture/20180531_dl_intro/contents.tgz
       （参考回答は同じディレクトリのanswer.tgz）
    2. コマンド例:
       cp /lustre/gt00/share/lecture/20180531_dl_intro/contents.tgz ./

ジョブを投入する

動作環境をジョブ経由で確認

3. 解凍したら、build/contents/へ移動
    1. 解凍コマンド例: tar xf contents.tgz
    2. 移動コマンド: cd build/contents/
4. run_test_job.shを完成させる
    1. 変更箇所は2ページ先

ジョブを投入する

動作環境をジョブ経由で確認

4. スクリプトの修正完了したら実行
    1. qsub -j oe run_test_job.sh
5. 結果を確認
    1. スクリプトと同じディレクトリに test_job.oxxxxxxx というファイルを確認
    2. cat test_job.oxxxxxx で内容をコンソール出力できる

スクリプトの変更箇所

```
1  #!/bin/sh
2  #PBS -q xxxxxx
3  #PBS -l xxxxxx
4  #PBS -W group_list=xxxxxx
5  #PBS -l walltime=00:02:00
6  #PBS -N test_job
7
8  echo "[$(date)]
9  xxxxxx
10 xxxxxx
11
12 # load and check
13 echo "[$(date)] load modules."
14 xxxxxx cuda9/9.1.85 anaconda3/4.3.0 openmpi/gdr/2.1.2/intel intel/18.1.163 chainermn/1.2.0
15
16 echo "[$(date)] loaded modules."
17 xxxxxx
18
19 echo "[$(date)]
20 python -c "impo                                    ner.__version__));"
21 python -c "import chainermn; print('ChainerMN ver.: {}'.format(chainermn.__version__));"
22
23 echo "[$(date)] fin."
```

ジョブ実行条件

作業ディレクトリへの移動

環境変数等の設定

モジュールロードと確認

テストジョブ実行結果

```
[Mon May 28 13:35:24 JST 2018] start.
[Mon May 28 13:35:24 JST 2018] load modules.
[Mon May 28 13:35:24 JST 2018] loaded modules.
Currently Loaded Modulefiles:
  1) intel/18.1.163             5) anaconda3/4.3.0
  2) intel-mpi/2018.1.163       6) openmpi/gdr/2.1.2/intel
  3) pbsutils                   7) chainermn/1.2.0
  4) cuda9/9.1.85
[Mon May 28 13:35:24 JST 2018] check chainer and chainermn
version.
Chainer ver.: 3.3.0
ChainerMN ver.: 1.2.0
[Mon May 28 13:35:29 JST 2018] fin.
```

ロード済みモジュール

スクリプトの変更箇所(答え)

```
1  #!/bin/sh
2  #PBS -q xxxxxx
3  #PBS -l xxxxxx
4  #PBS -W group_list=xxxxxx
5  #PBS -l walltime=00:02:00
6  #PBS -N test_job
7
8  echo "[$(date)
9  xxxxxx
10 xxxxxx
11
12 # load and check modules.
13 echo "[$(date)] load modules."
14 xxxxxx cuda9/9.
15
16 echo "[$(date)
17 xxxxxx
18
19 echo "[$(date)] check chainer and chainermn version."
20 python -c "import chainer; print('Chainer ver.: {}'.format(chainer.__version__));"
21 python -c "import chainermn; print('ChainerMN ver.: {}'.format(chainermn.__version__));"
22
23 echo "[$(date)] fin."
```

#PBS -q **h-tutorial**
#PBS -l **select=1:mpiprocs=1:ompthreads=1**
#PBS -W group_list=**gt00**

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

module load cuda9/9.1.85 anaconda3/4.3.0
openmpi/gdr/2.1.2/intel intel/18.1.163 chainermn/1.2.0
module list

タスク#2: シングルGPUでの学習ジョブ

シングルGPUで学習ジョブを流す

簡単な例で、ジョブの流し方を把握する

以下の作業を完了させる。

1. train_cifar_single_gpu.pyに「PUT YOUR CODE」という箇所があるので、書き換えてプログラムを完成させる
2. スクリプトの修正完了したら実行
    1. qsub -j oe run_single_gpu_training.sh
3. 結果を確認
    1. cat logs/single_gpu_log_xxxxxx.txt

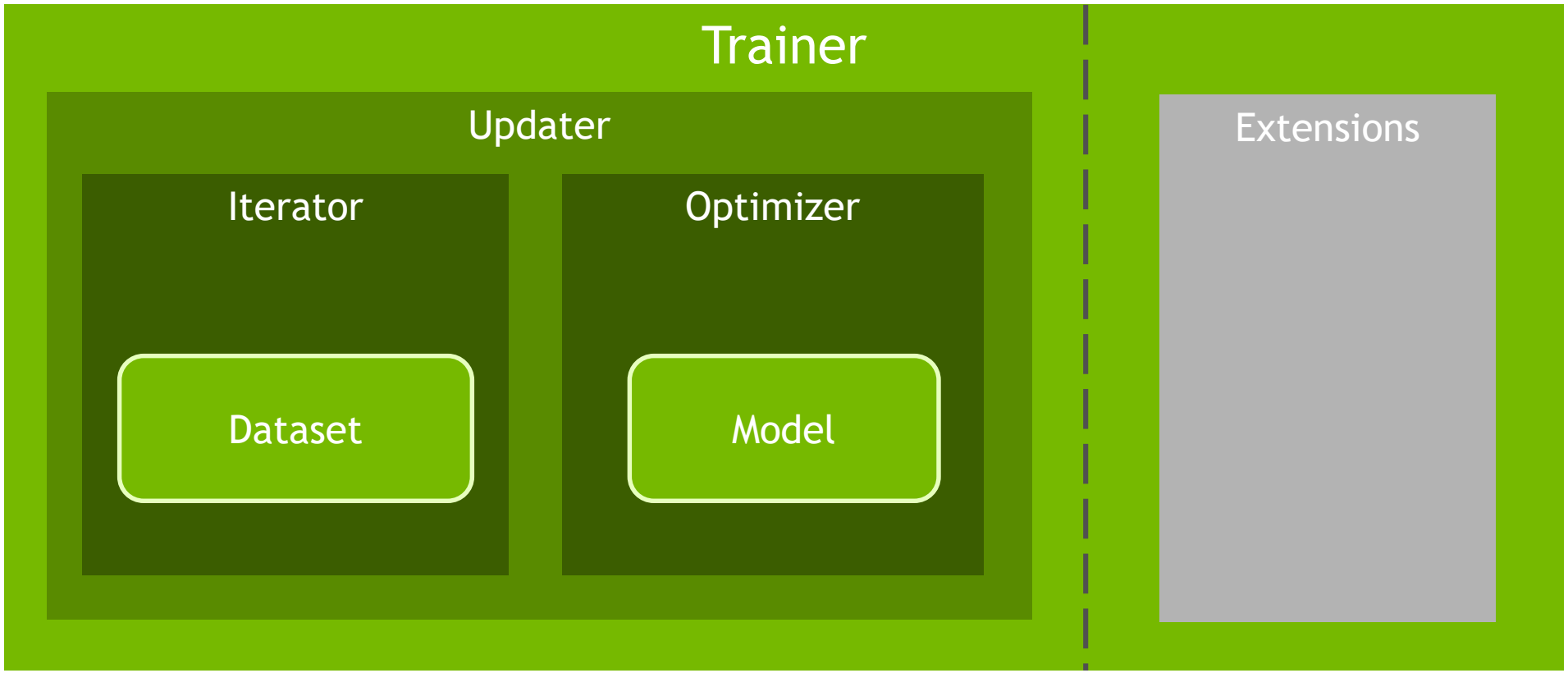
CHAINERを使った学習

Trainer利用と直接実装

- 標準的な処理しかしないなら、Trainerを利用するのが簡便でよい
  - v1.11.0から導入
- カスタマイズも容易
- 学習フローをすべてスクラッチで実装することも可能
  - (trainerあり) https://github.com/chainer/chainer/blob/master/examples/cifar/train_cifar.py
  - (trainerなし) https://github.com/chainer/chainer/blob/master/examples/cifar/train_cifar_custom_loop.py

CHAINERを使った学習

Trainer利用と直接実装

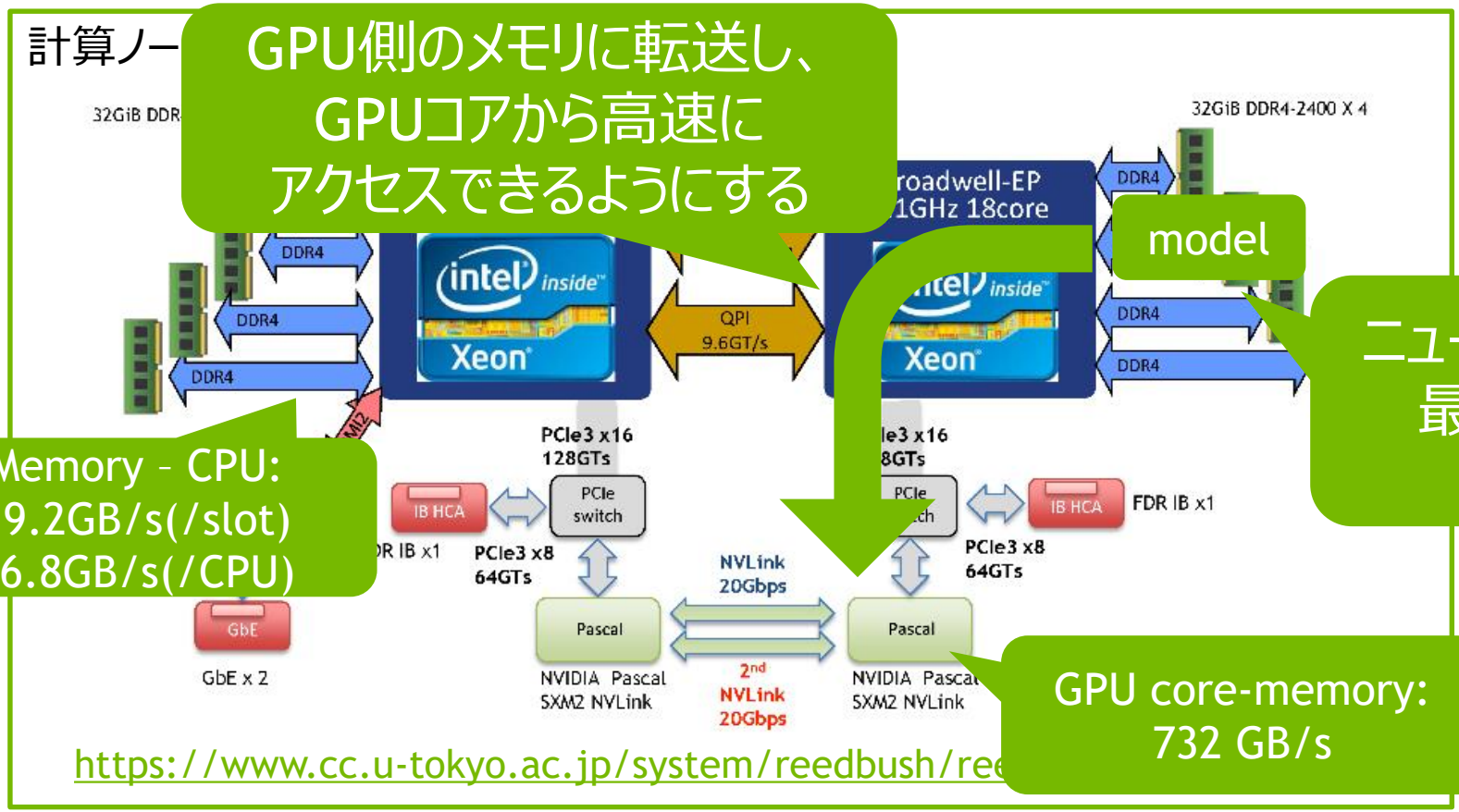


CHAINERのAPI

GPUの操作

- どのGPUを使うか指定する
  - chainer.backends.cuda.get_device_from_id(device_id).use()
  - get_device_from_id()はcupy.cuda.Deviceを返す
- GPUで処理したいオブジェクトを、CPUのメモリからGPUのメモリへ転送
  - 学習モデルをコピーしたいのでmodel.to_gpu()

GPUのメモリへ転送？



GPU側のメモリに転送し、
GPUコアから高速に
アクセスできるようにする

model

ニューラルネットの変数は
最初CPU側メモリに
(FW次第です)

Memory - CPU:
19.2GB/s(/slot)
76.8GB/s(/CPU)

GPU core-memory:
732 GB/s

CHAINERのAPI

学習/評価の実行

- ネットワークの更新ロジック(シングルGPU/マルチGPU/etc.)を指定
  - `chainer.training.updaters.StandardUpdater(iterator, optimizer, device)`
  - 学習データのイテレータとoptimizer、使用するGPUのIDを渡す
- テストデータでの評価をするクラス
  - `chainer.training.extensions.Evaluator(iterator, target, device)`
  - テストデータのイテレータとモデルオブジェクト、使用GPUのIDを渡す

CHAINERのAPI

学習結果の保存と再利用

- 学習結果の保存
  - `chainer.serializers.save_npz(file, obj)`
- 学習結果の読み込み
  - `chainer.serializers.load_npz(file, obj)`
  - 読み込んだ学習結果を使って推論する場合は、`obj(data)`もしくは、`obj.predictor(data)`のように呼び出す

スクリプトの修正を試みましょう

コンソールに戻ってください！

スクリプトの変更箇所(答え)

```
82
83      # Set up a dataset and prepare train/test data iterator.
84      print('Using CIFAR10 dataset: {}'.format(args.dataset))
85      train, test = dataset_cifar10.load_cifar10(args.dataset)
86      train_iter = chainer.iterators.SerialIterator(train, args.batchsize)
87      test_iter = chainer.iterators.SerialIterator(test, args.batchsize,
88                                                   repeat=False, shuffle=False)
89
90      # Make a model.
91      class_labels = 10
        model = L.Classifier(models.VGG.VGG(class_labels))

        # Make a specified GPU current
        chainer.cuda.get_device_from_id("""PUT YOUR CODE""").use()
        model."""PUT YOUR CODE""" # Copy the model to the GPU
97

                                                    mSGD(args.learnrate)

                                                    eightDecay(5e-4))

103     # Set up a trainer
104     updater = """PUT YOUR CODE"""
105     trainer = training.Trainer(updater, (args.epoch, 'epoch'), out=args.out)
106     evaluator = extensions.Evaluator(test_iter, model, device="""PUT YOUR CODE""")
107
```

model.**to_gpu**()

get_device_from_id(**0**)

updater=**training.StandardUpdater(**
train_iter, optimizer, device=0)

device=**0**

実行結果のモニタリング

- 実行の実体は以下のコマンド
  - python train_cifar_single_gpu.py --epoch 10 --batchsize 64 ... > ${LOGDIR}/single_gpu_log_$(date +%s).txt 2>&1
- qsubコマンド自体のログも出ている
  - デフォルトでは、実行スクリプトと同じ場所に「ジョブ名.o$JOB_ID」で出力
- リアルタイムにモニタリングしたい場合は、前者の自前ログを「tail -f」などで監視するのが良い

シングルGPUでの実行結果 (1/2)

学習データ/検証データの誤差

学習データ/検証データでの精度

```
Start a training script using single GPU.
# Minibatch-siz
# epoch: 10

Using CIFAR10 datase
/lustre/gt00/share/   cture/20180531_dl_intro/dataset//cifar10_32px.p    le.gz
epoch  main/loss   validation/main/loss  main/accuracy  validation/main/accuracy  elapsed_time
1      2.39575     1.93407               0.15579        0.221835                  34.5854
2      1.87273     1.97954               0.259163       0.254479                  67.0101
3      1.65184     2.04599               0.353993       0.268312                  98.2595
4      1.43666     1.33344               0.463468       0.515625                  129.817
5      1.23045     1.1243                0.558883       0.599821                  161.2
6      1.09442     0.969212              0.621859       0.665008                  192.855
7      1.0109      1.03414               0.65709        0.655155                  224.957
8      0.950693    0.927806              0.683519       0.685908                  256.997
9      0.903565    0.846908              0.703684       0.722731                  289.166
10     0.871967    0.829112              0.720411       0.731887                  321.311
Throughput: 1513.0998149446914 [images/sec.] (500000 / 330.4474662289722)
```

シングルGPUでの実行結果 (2/2)

```
# qsubの実行ログに出力

(前略)
[Mon May 28 13:41:32 JST 2018]                                    ained model.
Start an inference script using
# Target model: result/single_g
# Target data directory: /lustre/gt0     re/lecture/20180531_dl_intro/dataset//inference/

[00]: file[cat_01.npz] is truck.
[01]: file[cat_02.npz] is deer.
[02]: file[cat_03.npz] is frog.
[03]: file[cat_04.npz] is frog.
[04]: file[cat_05.npz] is airplane.
[05]: file[dog_01.npz] is cat.
[06]: file[dog_02.npz] is automobile.
[07]: file[dog_03.npz] is horse.
[08]: file[dog_04.npz] is dog.
[09]: file[dog_05.npz] is airplane.
(後略)
```

CIFAR-10以外の実データ
推論結果

あまり当たっていないのは
学習回数が少ないため

シングルGPUでの実行結果 (2/2)

```
# qsubの実行ログに出力

(前略)
[Mon May 28 13:41
Start an inferenc
# Target model: r
# Target data dir                                    nference/

[00]: file[cat_01
[01]: file[cat_02
[02]: file[cat_03
[03]: file[cat_04
[04]: file[cat_05
[05]: file[dog_01
[06]: file[dog_02
[07]: file[dog_03
[08]: file[dog_04
[09]: file[dog_05.
(後略)
```

110エポック回してみると以下のような結果

```
[00]: file[cat_01.npz] is truck.
[01]: file[cat_02.npz] is deer.
[02]: file[cat_03.npz] is cat.
[03]: file[cat_04.npz] is cat.
[04]: file[cat_05.npz] is airplane.
[05]: file[dog_01.npz] is dog.
[06]: file[dog_02.npz] is dog.
[07]: file[dog_03.npz] is dog.
[08]: file[dog_04.npz] is dog.
[09]: file[dog_05.npz] is deer.
```

より当たるように
(画像は講習会のみ掲載)

(OPTION) 追加で試す

早く終わった方のために

- Resume機能を実装してみる
  - 更に長時間学習させるために必要
  - 前述の`chainer.serializers.load_npz(file, obj)`を使う
- ジョブ間の依存関係を考慮して実行してみる
  - qsubのオプション(-W)を使うと実現可能
  - 詳細は「Reedbush システム利用手引書（概要・Reedbush-U 編）」の「4.1.2.5.チェーンジョブ(-W)」を参照

NVIDIA | DEEP LEARNING INSTITUTE

www.nvidia.com/dli