

# GPU を使った画像分類 (Linux/Chainer)

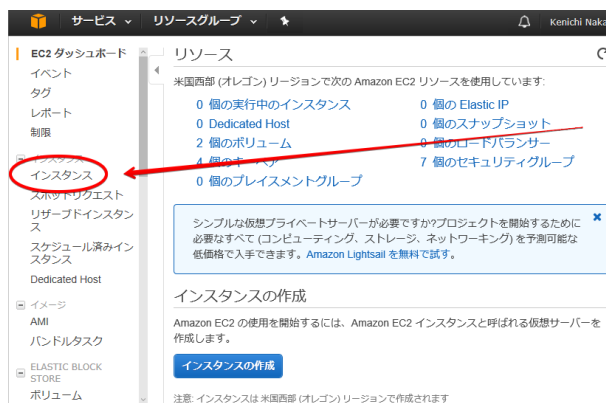
## 1 ログイン

アマゾン EC2 にログインします。

AWS サービス



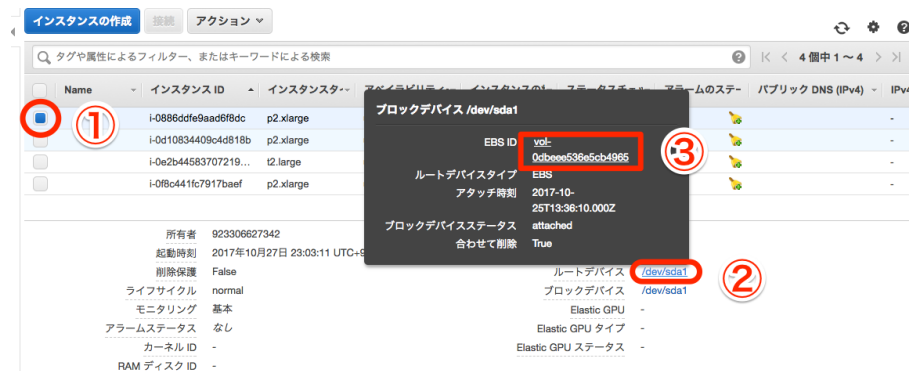
EC2 サービスを選択します



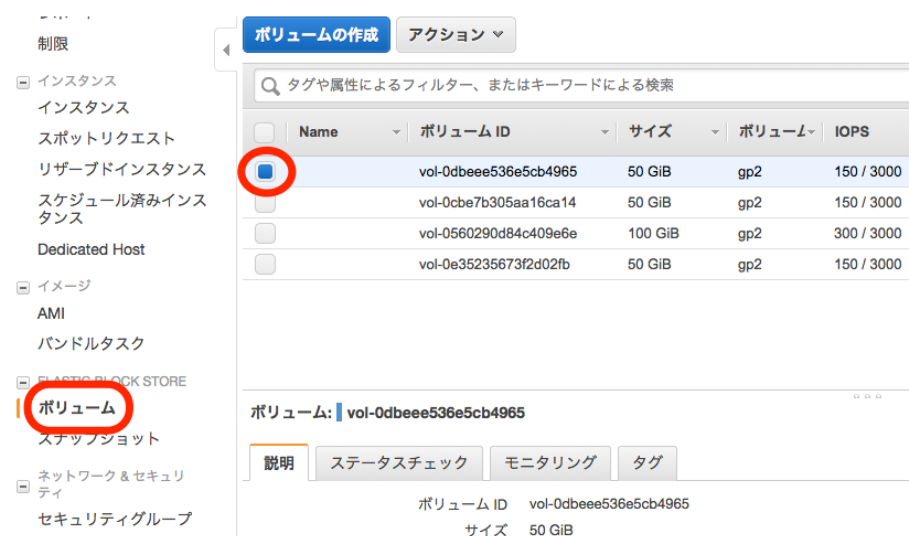
「インスタンス」をクリックします

## 2 インスタンスのボリューム容量の変更

インスタンスにアタッチされているボリュームの ID (EBS ID) を確認します。



インスタンスにアタッチされているボリュームを選択します。





「アクション」から「ボリュームの変更」を選択します。

ボリュームの変更ウィンドウが表示されます。

ボリュームの変更

×

ボリューム ID

vol-0dbeee536e5cb4965

ボリュームタイプ

汎用 SSD (GP2)

?

①

サイズ

100

(最小: 1 GiB、最大: 16384 GiB)

?

iops

300 / 3000

(1 GiB あたり 3 IOPS のベースライン、最小 100 IOPS、3000 IOPS にバースト可能)

?

②

キャンセル

変更

1. 変更後のボリュームの容量 (GiB、ギガバイト) を入力します。今回は、100 GByte に設定します。  
2. 「変更」ボタンを押します。

## ボリュームの変更



ボリューム vol-0dbeee536e5cb4965 を変更してもよろしいですか？

パフォーマンスの変更が完全に反映されるまでに時間がかかることがあります。

新しく割り当てられた領域を使用するには、ボリュームで OS ファイルシステムを拡張することが必要になる場合があります。

[Linux](#) および [Windows](#) での EBS ボリュームのサイズ変更について詳細を参照してください。

キャンセル

いいえ

はい

「はい」ボタンを押します。

ボリュームの変更

ボリュームの作成 アクション

タグや属性によるフィルター、またはキーワードによる検索

Name	ボリューム ID	サイズ	ボリューム	IOPS	スナップショット	作成日	アベイラビリティ	状態
	vol-0dbeee536e5cb4965	100 GiB	gp2	300 / 3000	snap-0ddb44...	2017年10...	us-west-2a	in-use - optimizing (15%)
	vol-0cbe7b305aa16ca14	50 GiB	gp2	150 / 3000	snap-05f37a1c...	2017年9...	us-west-2a	in-use
	vol-0560290d84c409e6e	100 GiB	gp2	300 / 3000	snap-0d51081...	2017年8...	us-west-2c	in-use
	vol-0e35235673f2d02fb	50 GiB	gp2	150 / 3000	snap-03f9ba57...	2017年8...	us-west-2c	in-use

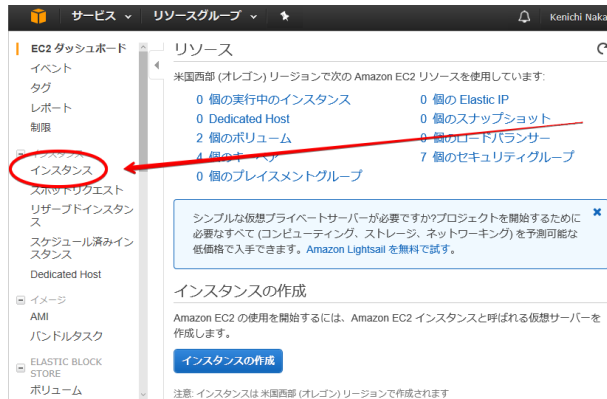
上記からボリュームを選択してください

1. 左側のメニューから「ボリューム」を選択します。

変更したボリュームのステータスが「in-use - optimizing」になっている場合には、変更が完了するまで待ちます。

変更したボリュームのステータスが「in-use - completed」になっていることを確認します。

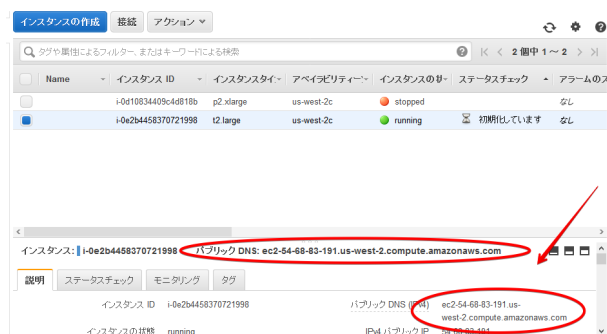
### 3 インスタンスの起動



「インスタンス」をクリックします



インスタンスが立ち上がっていない場合、「アクション」から「インスタンスの状態/開始」を選択し、インスタンスを立ち上げます。



インスタンスの状態が「running」になったら、インスタンスの「パブリック DNS」を確認します。

「アマゾン EC2 インスタンスの設定」の「Putty セッションの開始」を参考に、SSH クライアント (PuTTY) でインスタンスに接続します。

## 4 環境設定

### 4.1 授業で使わないファイルの削除

```
cd ~/src
sudo rm -r cntk
sudo rm -r torch
sudo rm -r pytorch
```

### 4.2 Anaconda のインストール

```
cd ~
mkdir Downloads
cd Downloads
wget https://repo.continuum.io/archive/Anaconda3-4.4.0-Linux-x86_64.sh
md5sum Anaconda3-4.4.0-Linux-x86_64.sh
```

「50f19b935dae7361978a04d9c7c355cd」であることを確認します。

```
bash ~/Downloads/Anaconda3-4.4.0-Linux-x86_64.sh

Please, press ENTER to continue改行を押す

Do you approve the license terms? [yes|no]
>>> yes

Anaconda3 will now be installed into this location:
/home/ubuntu/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/ubuntu/anaconda3] >>>改行を押す

Do you wish the installer to prepend the Anaconda3 install location
```

```
to PATH in your /home/ubuntu/.bashrc ? [yes|no]
[no] >>> yes
```

Thank you for installing Anaconda3!

Share your notebooks and packages on Anaconda Cloud!  
Sign up for free: <https://anaconda.org>

.bashrc が変更されているので、再度読み込みます。

```
cd ~
source .bashrc
```

### 4.3 chainer 実行環境の作成

python2.7 chainer2.0.2 をインストールした環境を作成します。

```
conda create --name=chenv python=2.7
```

Proceed と聞かれたら、y を入力し改行します。

今作った環境に入ります。

```
source activate chenv
```

環境の名前が行頭に表示されます。

### 4.4 chainer のインストール

chainer2.0.2 をインストールします。

```
pip install chainer=="2.0.2"
```

### 4.5 opencv のインストール

opencv をインストールします。

```
conda install -c menpo opencv
```

Proceed と聞かれたら、y を入力し改行します。

### 4.6 Pillow(Python Imaging Library) のインストール

```
conda install Pillow
```

Proceed と聞かれたら、y を入力し改行します。

## 4.7 matplotlib のインストール

```
conda install matplotlib
```

Proceed と聞かれたら、y を入力し改行します。

## 4.8 h5py のインストール

```
conda install h5py
```

Proceed と聞かれたら、y を入力し改行します。

## 4.9 cupy のインストール

cupy をインストールします。cupy は numpy 互換インターフェイスの CUDA 実装ライブラリです。既存のライブラリと同じインターフェイスで GPU 計算を行うことができます。

```
pip install cupy
```

cupy はビルドに時間がかかります。動きが止まったように見えることがありますが、そのまましばらくお待ち下さい。

# 5 GPU インスタンスを使った画像認識

今回は、GPU インスタンスを使って画像認識を行ってみましょう。

## 5.1 授業関連ファイルの取得

授業の関連ファイルをダウンロードします。

```
cd ~  
git clone https://github.com/k1nk/lesson.git
```

lesson フォルダが作成されます。その下に関連ファイルがあります。

作業用のディレクトリに移動します。

```
cd ~/lesson/ch4/chainer_imagenet_tools
```

前回、訓練用データセットでは正答率が90%を超えるのに、検証用データセットでは正答率が70%程度になっていました。過学習の状態となっていることがわかります。



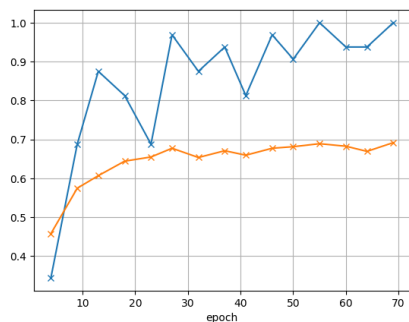


図1 accuracy

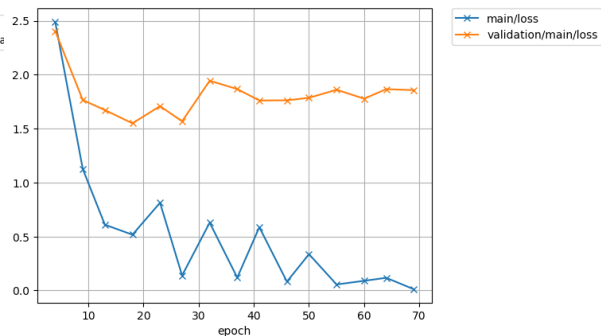


図2 loss

過学習の状態を直すには、

- データを増やす
- モデルが問題に対して複雑すぎる場合には、より単純なモデルを用いる

の2つのアプローチがあります。

## 5.2 画像データを増やす

画像にエフェクトをかけて画像を増やします。increase\_pictures.py は、1つの画像に、8種類のエフェクトをかけます。その結果、1つの画像が元の画像を含めて9枚の画像になります(図3から図11参照)。

## 5.3 データを準備する

画像データを切り取り、サイズを  $256 \times 256$  へ変更します。

```
python crop_pictures.py 101_ObjectCategories
```

101\_ObjectCategories\_cropped フォルダが作成され、中に変更後の画像ファイルが保存されます。

画像データを増やします。

```
python increase_pictures.py 101_ObjectCategories_cropped
```

increase\_pictures.py を実行して 101\_ObjectCategories\_cropped に入っている画像を増やします。増えたあとの画像は、101\_ObjectCategories\_cropped\_increased フォルダに保存されます。このデータを用いてデータを作成し、学習を行います。

訓練用・評価用・テスト用データと、その一覧およびラベル一覧データを作成します。

```
python make_train_cv_test.py 101_ObjectCategories_cropped_increased
images
```

images\_train、images\_cv、images\_test という3つのフォルダが作成され、それぞれに、訓練用、評価用、テスト用のデータが格納されます。全体のデータのうち、訓練用データを60%、評価用データを20%、テス



図 3 元画像



図 4 ハイコントラスト



図 5 ローコントラスト



図 6 ガンマ変換  $\gamma < 1$

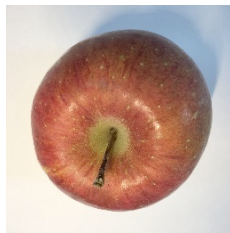


図 7 ガンマ変換  $\gamma > 1$

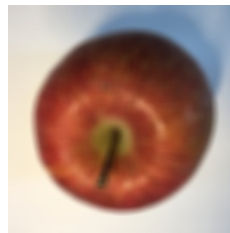


図 8 平滑化

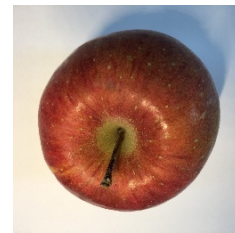


図 9 ヒストグラム均一化



図 10 ガウシアンノイズ



図 11 ペッパーノイズ

ト用データを 20%としています。また、訓練用、評価用、テスト用のデータの一覧が、train.txt、cv.txt、test.txt に作成されます。ラベルの一覧が labels.txt として作成されます。

訓練用データの平均画像ファイルを作成します。

```
python compute_mean.py train.txt
```

訓練用データの平均画像ファイルである、mean.npy が作成されます。

## 5.4 モデルを単純にする

AlexNet は 1000 種類のカテゴリ分類を行うことを予定して作成されています。今回は、102 種類なので、少なくとも出力層のノードの数は 102 へ減らすことができます。また、前半の Convolution 層で画像の特徴を把握し、把握した特徴にもとづいて後半の密結合層により分類を行っているのではと思われます。今回は、分類の数が 102 と少ないので密結合層を 2 つから 1 つに減らしてみます。これらの変更を行ったモデルが、alex\_mini2.py に記載されています。

## 5.5 訓練を実行する

訓練を実行します。

```
python train_imagenet.py -E 13 -a alex_mini2 -g 0 train.txt cv.txt
```

訓練の結果は、result ディレクトリーの下に保存されています。

```
ls result
cat result/log
```

result/log は、訓練の経過を記録したログファイルです。model\_iter\_\*\*\*\*\* は一定のイテレーション時点で保存されたモデルです。snapshot\_iter\_\*\*\*\*\* は一定のイテレーション時点で保存されたスナップショットです。このファイルを使って、その時点から訓練を再開できます。

通常は、ここで、モデルの階層の数などのハイパーパラメータを変えたものをいくつか訓練して、ハイパーパラメータの調整を行います。

result/loss.png と、result/accuracy.png をダウンロードして、内容を確認してみましょう。

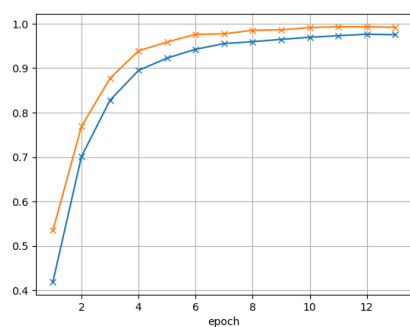


図 12 accuracy

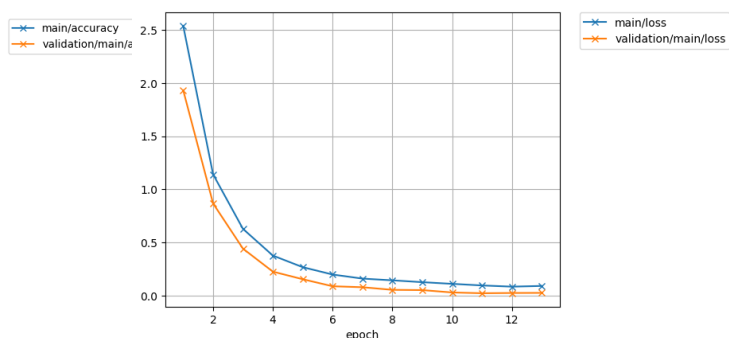


図 13 loss

図 13 では validation loss の方が、training/loss よりも小さくなっています。これは、モデルの中で Dropout 関数を使っているため、訓練中は、ネットワークの一部を使って訓練しているのですが、評価にあたっては、ネットワークの全体を使っているためと思われます。

## 5.6 モデルをテストする

最終的に作成されたモデルは評価用データを使って他のモデルと比較検討され、パラメータの調整が行われています。したがって、最終的に作成したモデルがどの程度一般的によいものとなっているかを評価用データを用いて評価すると、実際よりも良い値となってしまいます。そのため、最終的なモデルを評価するには、別に用意しておいたテスト用データを用います。

モデルをテストします。

```
python test_model.py test.txt -g 0 -a alex_mini2 -m result/  
model_iter_18000  
{'test:/loss': array(0.06707880645990372, dtype=float32), 'test:/  
accuracy': array(0.982890248298645, dtype=float32)}
```

テスト用データを用いて、正答率98.2%まで、改善することができました。

## 5.7 予測する

今回、作成したモデルを使って予測してみましょう。

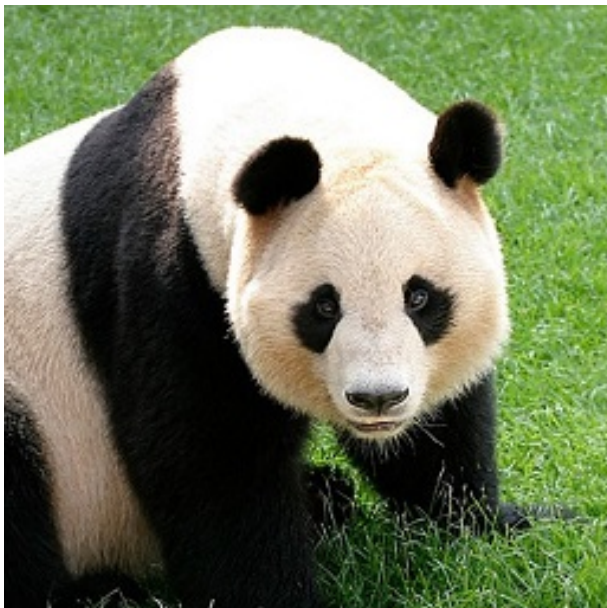


図 14 what\_is\_this\_1.jpg

「what\_is\_this\_1.jpg」が何かを予測します。

```
python inspection_alex_mini2.py what_is_this_1.jpg -m result/  
model_iter_18000  
#1 | panda | 100.0%
```

```
#2 | dalmatian | 0.0%  
#3 | yin_yang | 0.0%
```

panda と認識されています。

## 5.8 より進んだ課題

- nin googlenet resnet など、他のモデルについても試してみましょう。
- 今回は、AlexNet を元に、問題に適したモデルを作成しました。AlexNet あるいはその他のモデルを参考に自分のモデルを作成し、訓練の実行、評価、テストを行ってみましょう。