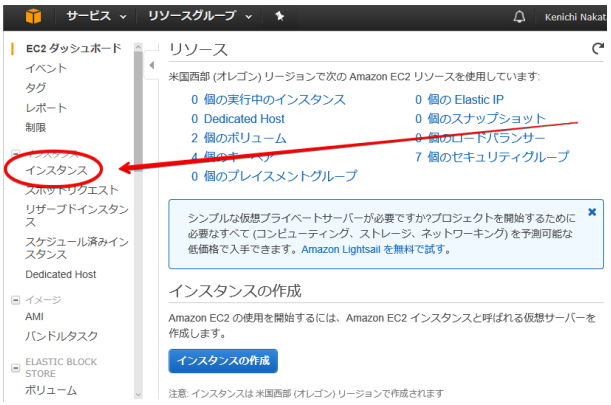
GPU を使った画像分類 (Linux/Chainer)

1　ログイン

アマゾン EC2 にログインします。

EC2 サービスを選択します

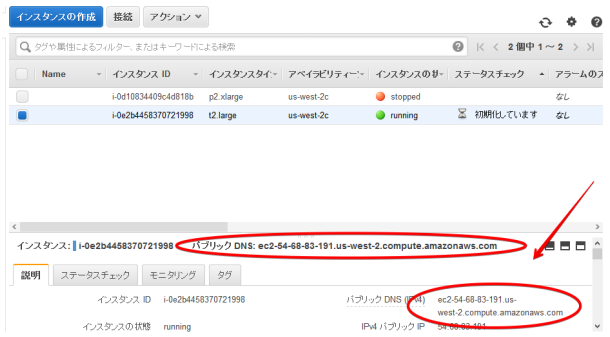


「インスタンス」をクリックします

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。

3 環境設定

3.1 Anadonda2 のインストール

Anadonda は、すでにインストールされていますが、PATH 等の変更をする必要があります。

```
cd ~
ls -la
```

ユーザのホームディレクトリには、「.bashrc」というファイルがあります。先頭にピリオドがあるファイルは通常は表示されません。そこで、ls でファイルを表示する際に、先の例のように「a」オプションをつけて確認します。.bashrc は、シェル（bash）ログインしたときに毎回読込まれる設定ファイルです。.bashrc には、環境変数 PATH などの設定が記載されています。コマンドを入力した際に、環境変数 PATH に記載された順番に、実行するコマンドのある場所が検索されます。同じコマンドが複数の場所にある場合、最初に見つかった場所にあるコマンドが実行されます。

ここでは、.bashrc の最後の行に、以下の行を追加します。これにより、$HOME/src/anaconda2/bin　にあるコマンドが他のコマンドに優先して実行されるようになります。

```
export PATH=$HOME/src/anaconda2/bin:$PATH
```

ファイルの編集には、vim,emacs,nano 等のエディタを利用します。

```
nano .bashrc
```

（参考）エディタを利用しない場合には、以下のコマンドを実行しても末尾に先に記載した行を追加できます。

```
echo "export PATH=\$HOME/src/anaconda2/bin:\$PATH" >> ~/.bashrc
```

.bashrc が変更されているので、再度読み込みます。

```
cd ~
source .bashrc
```

anaconda2 の所有者を変更します。

```
sudo chown -R ubuntu:ubuntu ~/src/anaconda2/
```

3.2 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。

3.3 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"
```

3.4 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。

3.5 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。

3.6 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。

3.7 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。

3.8 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。

4　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。

4.1　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％程度となっていました。過学習の状態となっていることがわかります。

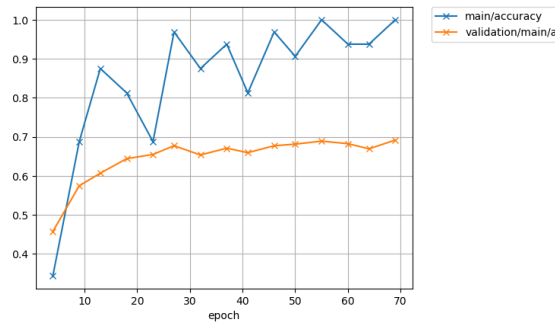


図 1　accuracy

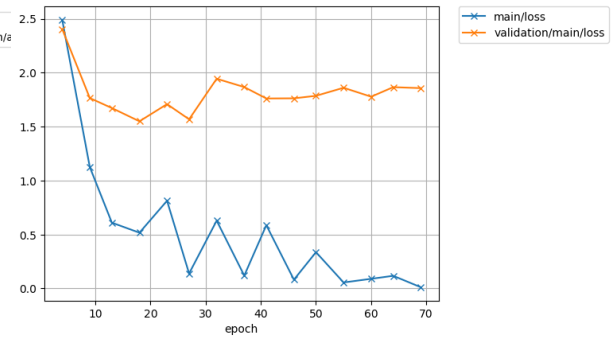


図 2　loss

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。

4.2　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)。

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

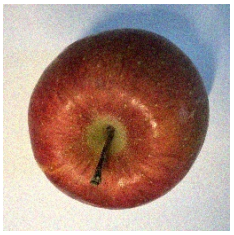


図 10　ガウシアンノイズ

図 11　ペッパーノイズ

4.3　データを準備する

画像データを切り取り、サイズを 256 × 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。

4.4 モデルを単純にする

AlexNet は１０００種類のカテゴリー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。

4.5 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、内容を確認して見ましょう。

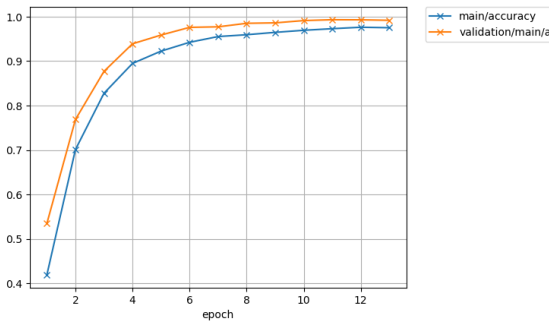


図 12　accuracy

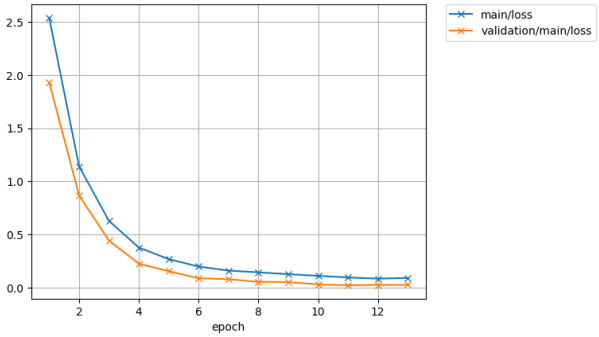


図 13　loss

図 13 では validation loss の方が、trainning/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。

4.6　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。

4.7　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。「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
```

結果は以下の通りとなります。

図 14　what_is_this_1.jpg

```
#1 | panda | 100.0%
#2 | dalmatian |  0.0%
#3 | yin_yang |  0.0%
```

panda と認識されています。

4.8　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。