NVIDIA DEEP LEARNING INSTITUTE

ハンズオンラボ
DIGITS による物体検出入門

山崎和博

ディープラーニング ソリューションアーキテクト
エヌビディア

AGENDA

ディープラーニングとは

Qwiklabs/DIGITSの使い方

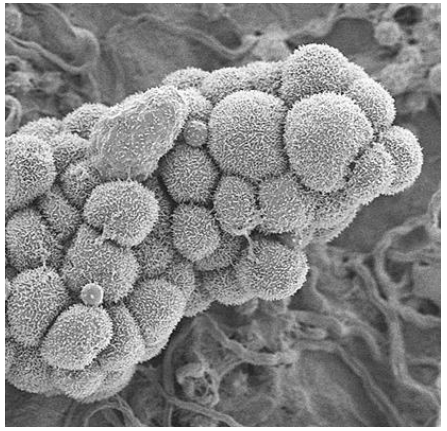
DIGITSによる物体検出入門ハンズオン

ディープラーニングとは

様々な分野でディープラーニングを応用

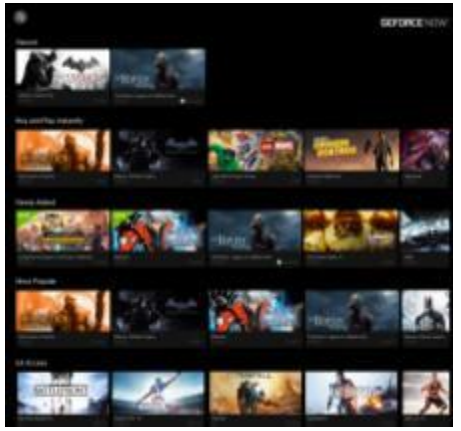
インターネットとクラウド

画像分類
音声認識
言語翻訳
言語処理
感情分析
推薦



医学と生物学

癌細胞の検出
糖尿病のランク付け
創薬

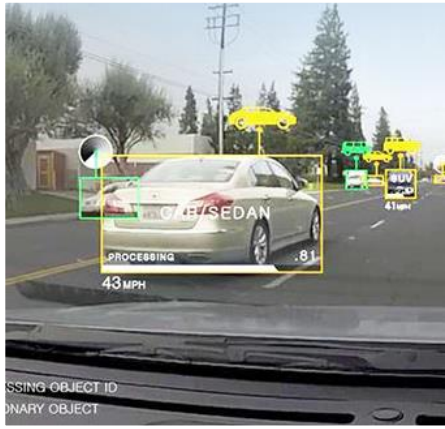


メディアとエンターテイメント

字幕
ビデオ検索
リアルタイム翻訳

セキュリティと防衛

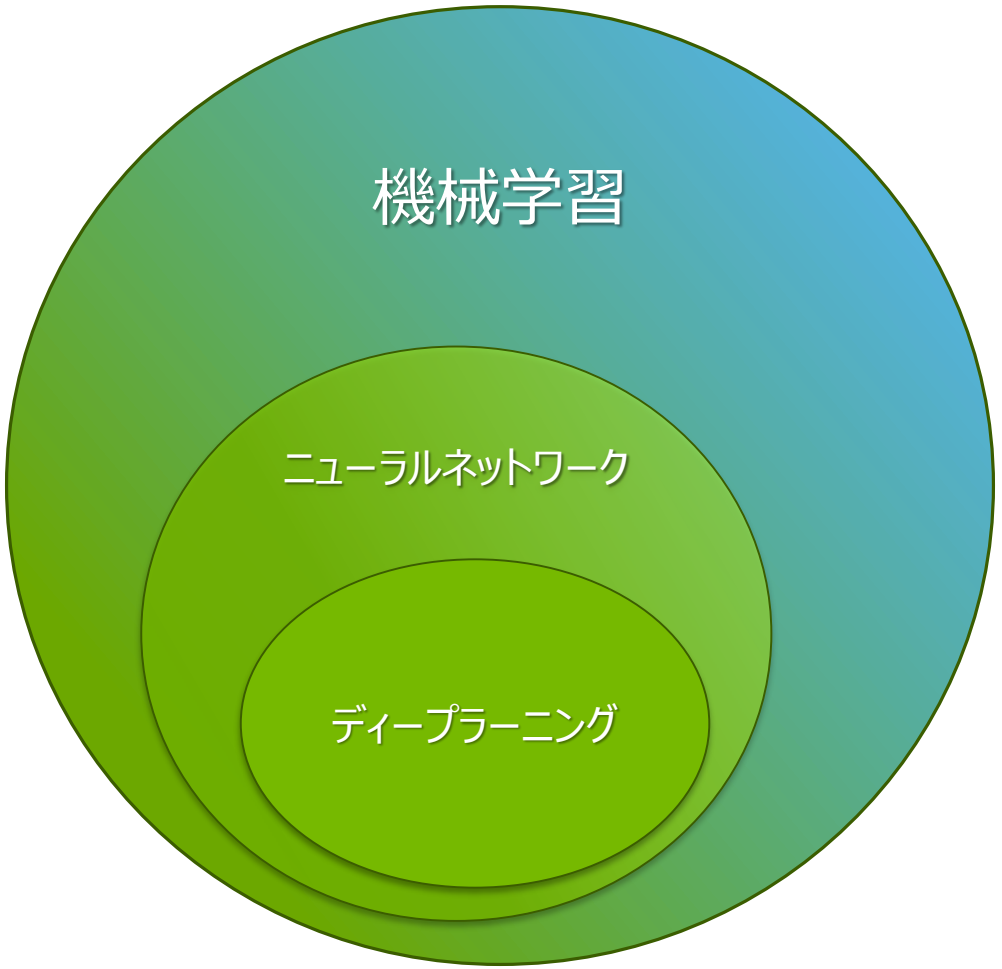
顔検出
ビデオ監視
衛星画像



機械の自動化

歩行者検出
白線のトラッキング
信号機の認識

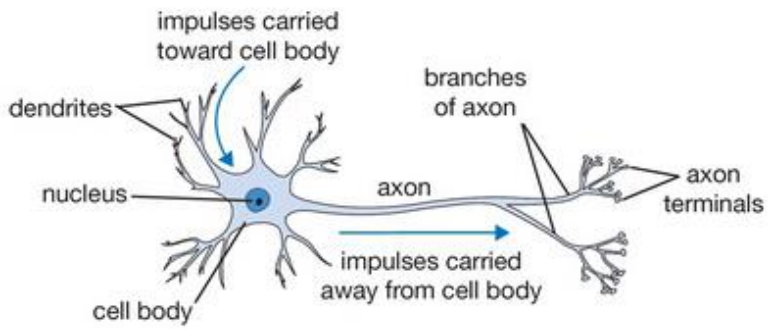
ディープラーニングは機械学習の一分野



人工ニューロン

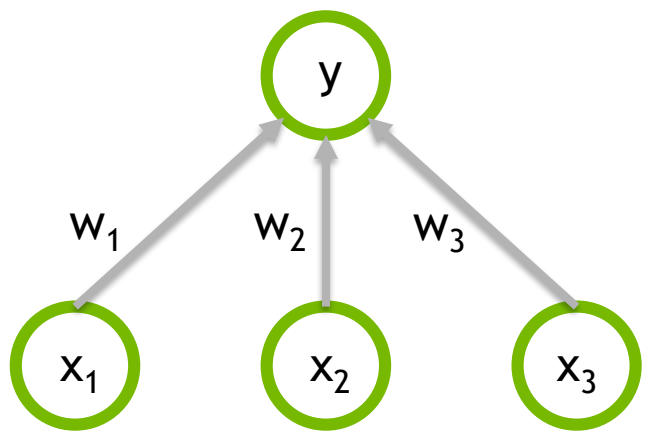
神経回路網をモデル化

神経回路網



スタンフォード大学cs231講義ノートより

人工ニューロン（パーセプトロン）

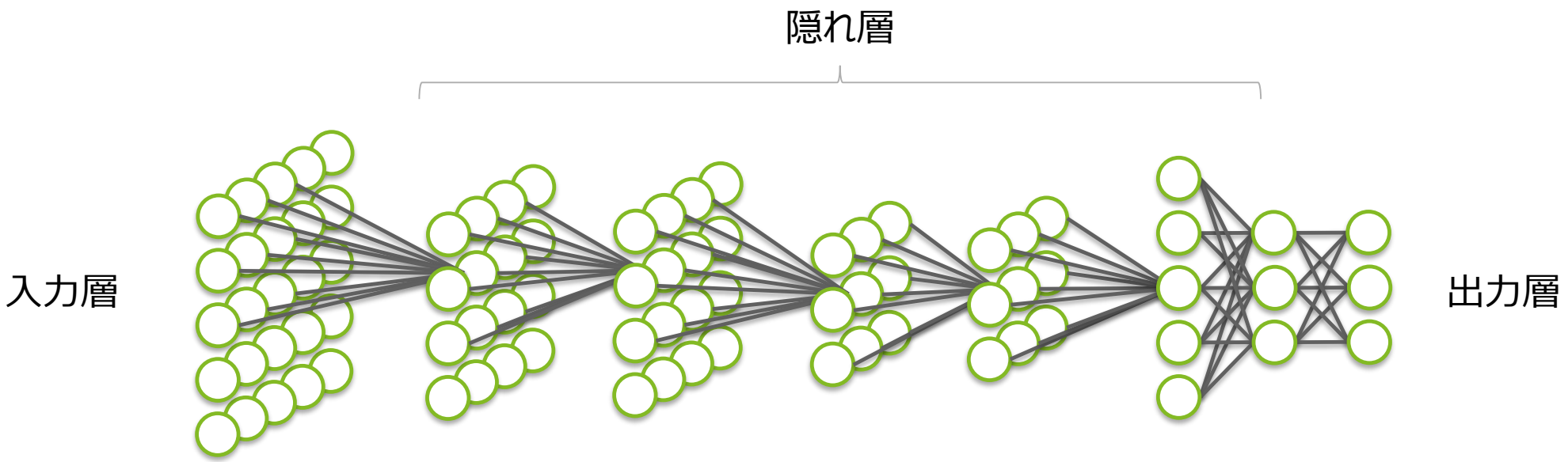


$y=F(w_1x_1+w_2x_2+w_3x_3)$

$F(x)=\max(0,x)$

人工ニューラルネットワーク

トレーニングできる単純な数学的なユニットの集合は、
複雑な機能を学ぶことができる



人工ニューラルネットワークは、十分なトレーニングデータが与えられれば、
生の入力データから出力を決定する、非常に複雑な関数を近似することができる。

ディープラーニングの恩恵

ディープラーニングとニューラルネットワーク

- ロバスト性
  - 特徴量の設計を行う必要がない
  - 特徴は、学習用データのバラつきの影響を押さえながら、自動的に学習・獲得される
- 一般性
  - 同じニューラルネットワークのアプローチを、多くの異なるアプリケーションやデータに適用できる
- スケーラブル
  - より多くのデータで大規模並列化を行う事でパフォーマンスが向上する

畳込みニューラルネットワーク(CNN)

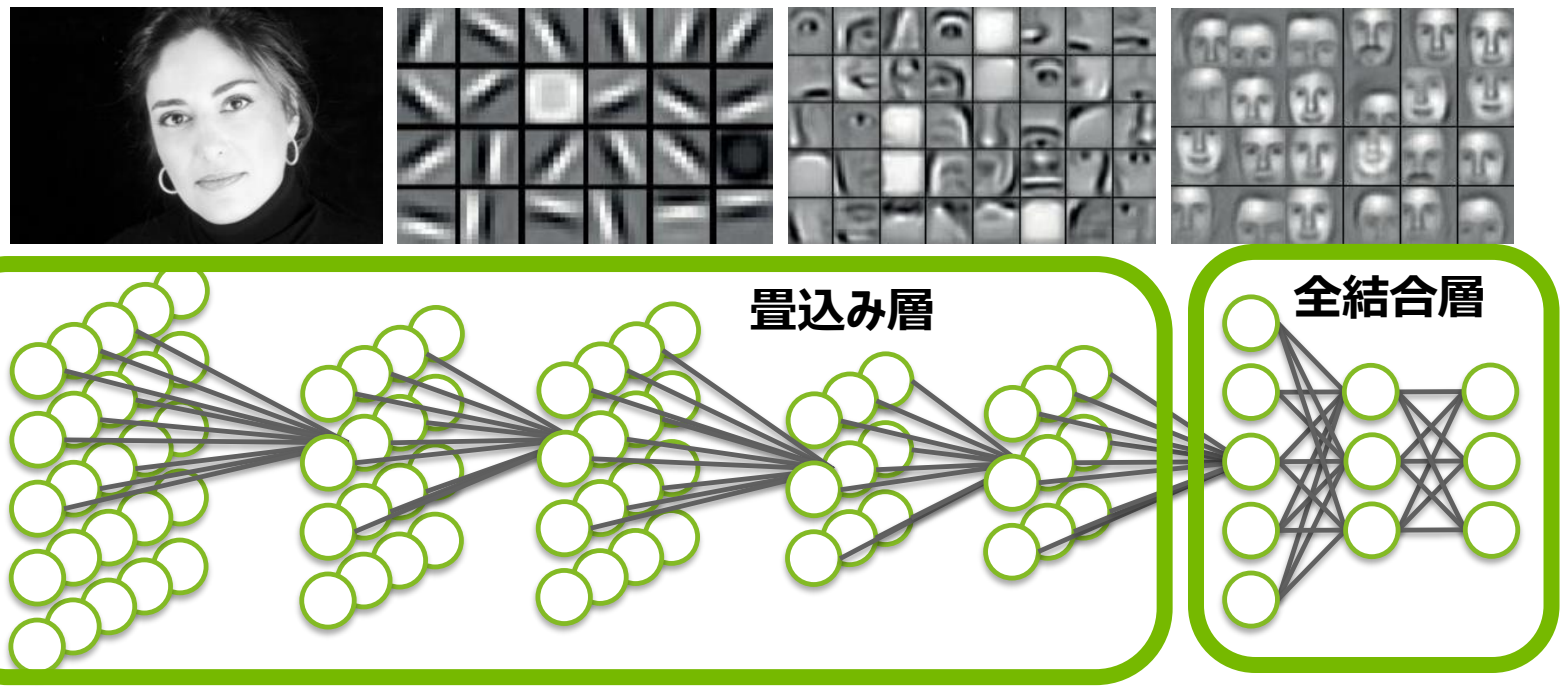
- 画像認識・画像分類で使われる、高い認識精度を誇るアルゴリズム。畳込み層で画像の特徴を学習

目的
顔認識

トレーニングデータ
1,000万～1億イメージ

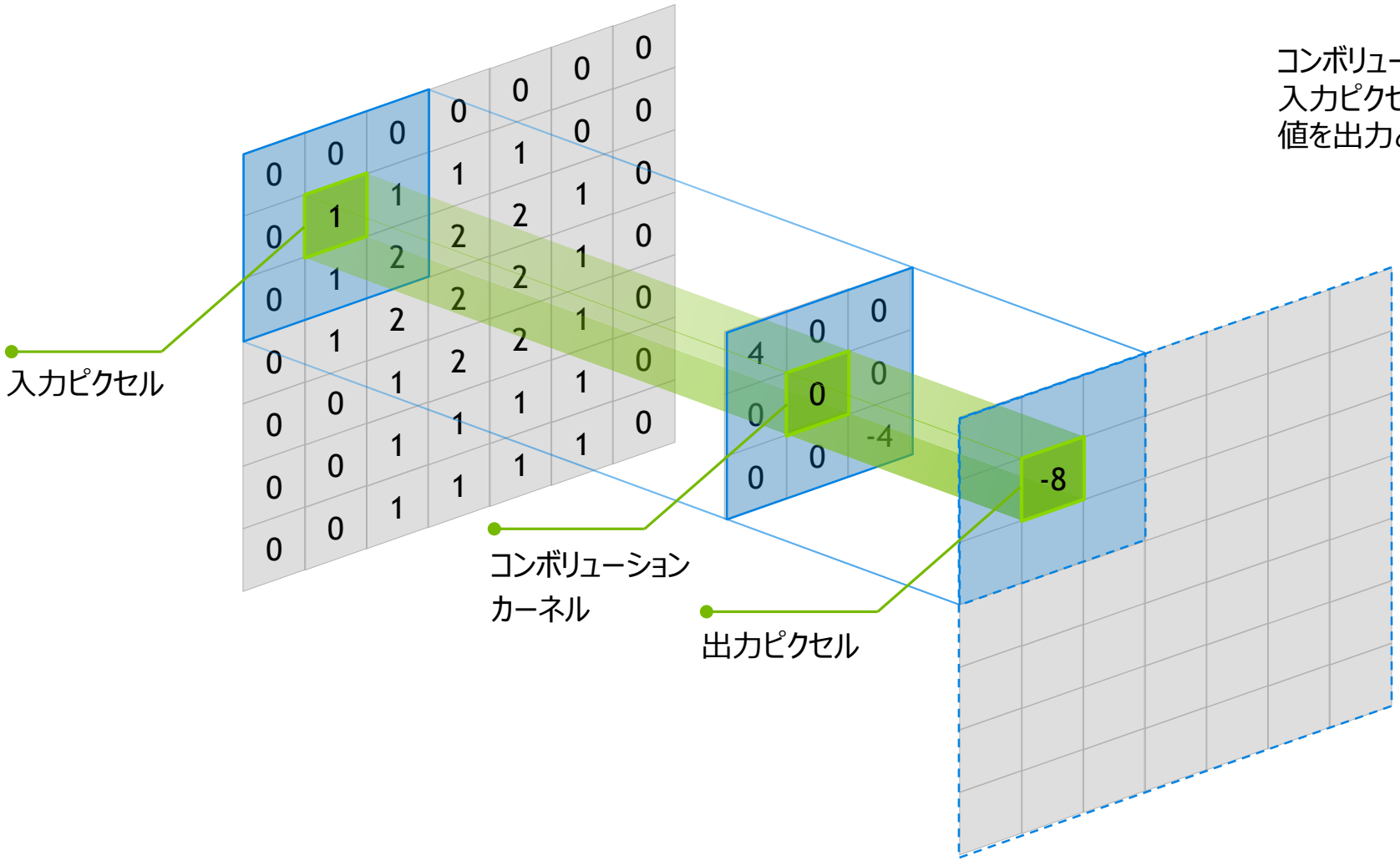
ネットワークアーキテクチャ
10 層
10 億パラメータ

ラーニングアルゴリズム
30 エクサフロップスの計算量
GPU を利用して30日



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。



ディープラーニング フレームワーク

ディープラーニング フレームワーク

GPUで高速化されたディープラーニング フレームワークが多数存在

https://developer.nvidia.com/deep-learning-frameworks

ディープラーニング フレームワーク

	Caffe	Torch7	Theano	TensorFlow	Chainer
インターフェース	C++/Python/Matlab	Lua/C	Python	C/C++/Python	Python
cuDNN	5	5	5	5	5
ライセンス	BSD-2	BSD	BSD	Apache 2.0	MIT
マルチGPU (1ノード)	○	○		○	○
モデルの柔軟性	△	◎	◎	○	◎
CNN	○	○	○	○	○
RNN	○ #2033	○	○	○	○
RBM	×	○	○	○	○
備考	高速 Caffe Model Zoo	多数のアルゴリズムをサポート	自動微分	自動微分 TensorBoard	Define by Run CuPy

CAFFE

CAFFE とは？

オープンソースのディープラーニング フレームワーク

- Berkeley Vision and learning Center (BVLC) において開発
- 多くのコントリビュータにより構成されるオープンソースコミュニティ
- C++/CUDA による実装
  - 高速で十分に検証された実装
  - シームレスな GPU によるアクセラレーション
- コマンドライン、Python, MATLAB インターフェース
- リファレンスモデルやサンプルもある

caffe.berkeleyvision.org
http://github.com/BVLC/caffe

CAFFE の機能

データのプリプロセスと管理

データフォーマット

LevelDB・LMDB
データベース

インメモリ
(C++・Python のみ)

HDF5

画像ファイル

プリプロセスツール

生画像から LevelDB/LMDB を作成

トレーニング用と検証用のデータセット作成(シャッフル付き)

平均画像の生成

データ変換

画像のトリミング、リサイズ、スケーリング、位置反転

平均値を引く

CAFFE の機能

ニューラルネットワークの定義

- **Protobuf モデルフォーマット**
  - ネットワーク構造や学習パラメータの定義に使われる
  - 様々なフォーマットに対応
    - Caffeのオートジェネレータで生成可能
    - 構文チェック機能も存在
  - コーディング不要

```
name: "conv1"
type: "Convolution"
bottom: "data"
top: "conv1"
convolution_param {
        num_output: 20
        kernel_size: 5
        stride: 1
        weight_filler {
                type: "xavier"
        }
}
```

CAFFE の機能

ニューラルネットワークの定義

Loss関数:

分類

- Softmax
- Hinge loss

線形回帰

- Euclidean loss

多値分類

- Sigmoid cross entropy loss

などなど……

使用可能なレイヤー種別:

- Convolution
- Pooling
- Normalization

利用可能な関数:

- ReLU
- Sigmoid
- Tanh

などなど……

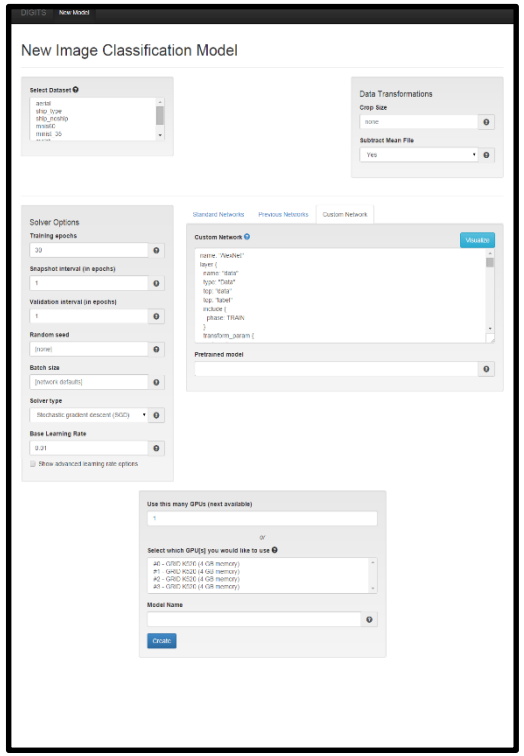
DIGITS

NVIDIA DIGITS

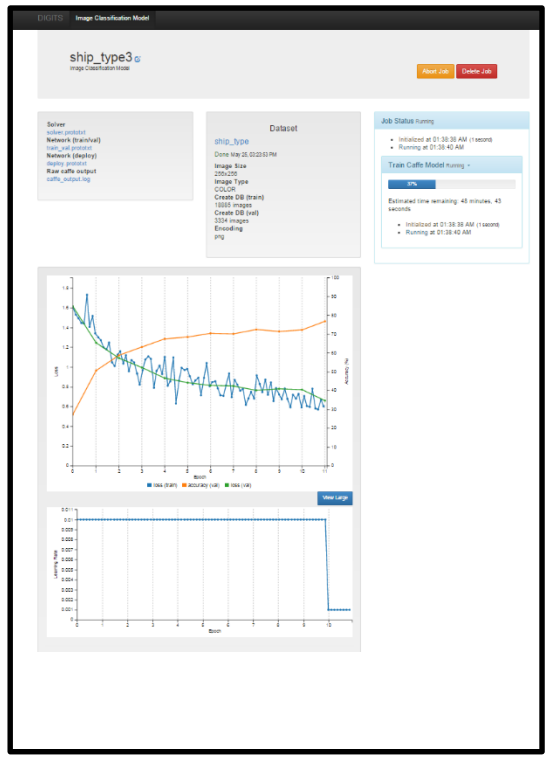
GPUで高速化されたディープラーニング トレーニング システム

学習データの作成

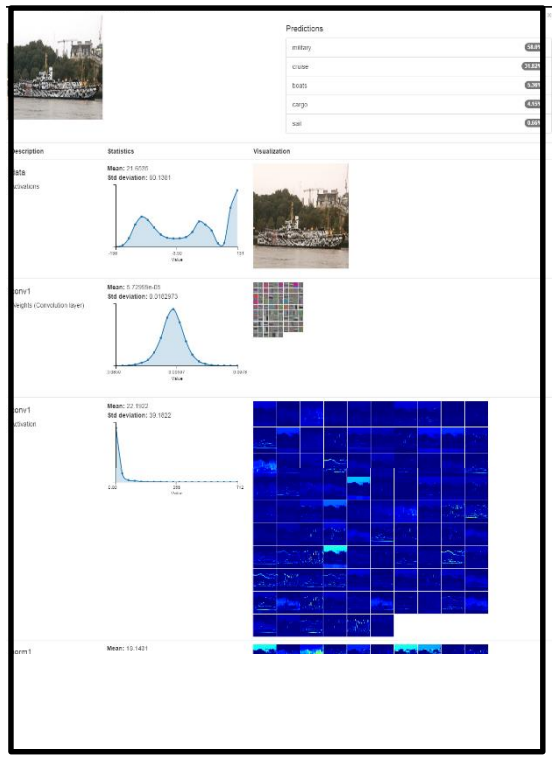
モデルの作成



学習過程の可視化



モデルのテスト

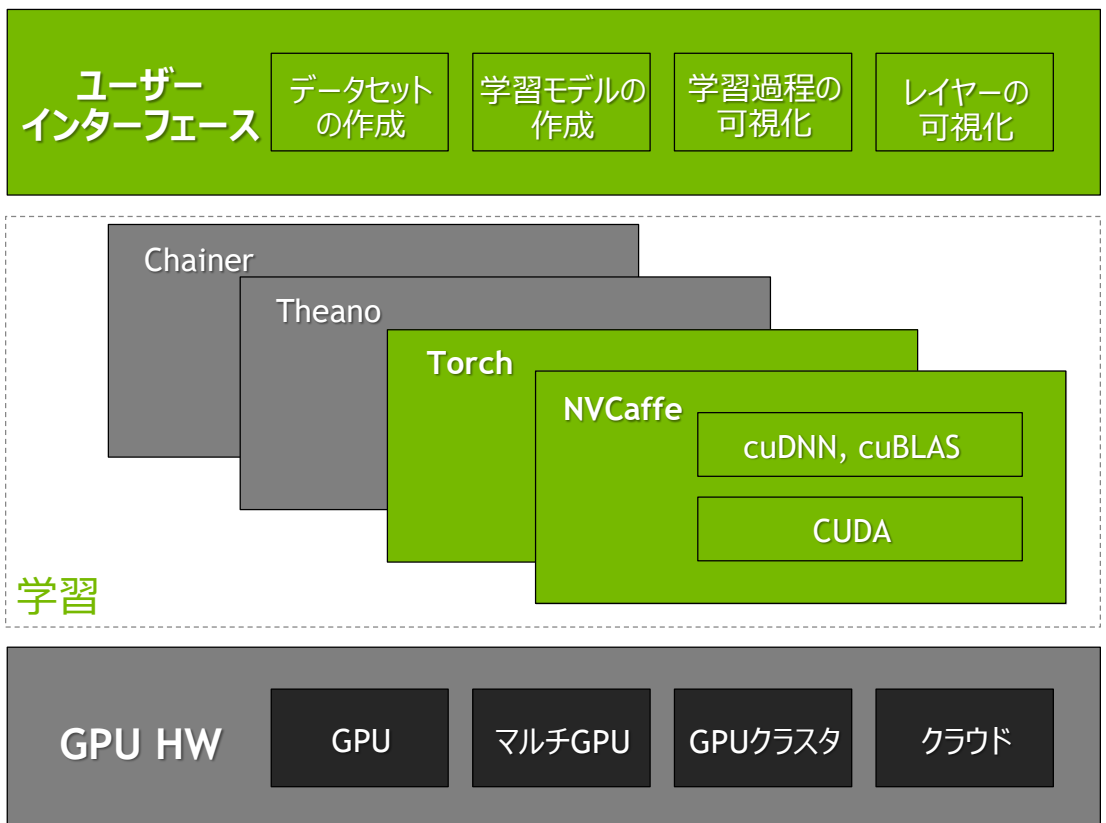


http://developer.nvidia.com/digits

NVIDIA DIGITS

GPUで高速化されたディープラーニング トレーニング システム

- 手元のPCからWebブラウザでアクセス可能なディープラーニング トレーニング システム



- 画像分類と物体検出の為のDNNのデザインと可視化の機能を提供
- NVIDIAが最適化したフレームワークで高速に学習が可能
- ハイパーパラメータのチューニングを強力にサポート
- 学習のジョブを簡単にスケジューリング、リアルタイムにaccuracyとlossを監視
- 様々な学習データフォーマット、フレームワークに対応

ハンズオンを開始しましょう

DIGITSによる物体検出入門

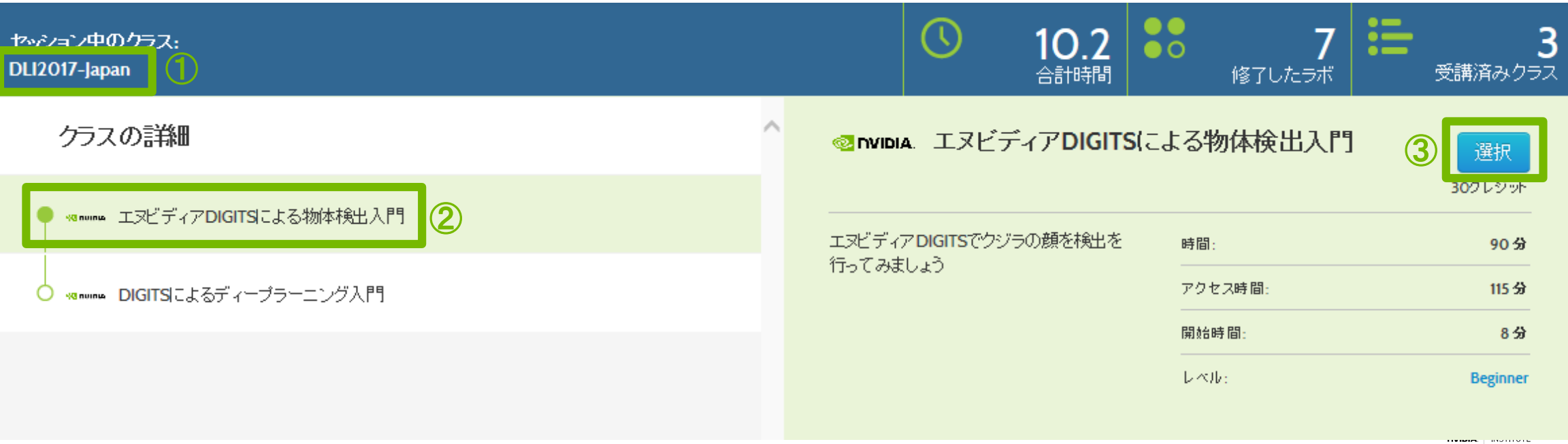
ログインからラボの開始まで

- https://nvlabs.qwiklab.com にアクセス
- ログイン (or 新規ユーザ作成)
- クラス “**DLI20170522-Japan**” を選択
- “エヌビディアDIGITSによる物体検出入門”を選択→ “選択”ボタンを押す
- “ラボを開始”ボタンを押す
- **わからないことがあったら、会場のアシスタントに聞いてください！**

DIGITSによる物体検出入門

ラボの開始

https://nvlabs.qwiklab.com/にアクセス
[DLI20170522-Japan]->[エヌビディアDIGITSによる物体検出]を選択



DIGITSによる物体検出入門

ラボの開始

DIGITSによる物体検出入門

DIGITSの使い方: ワークフロー

1. 学習データセットの作成
2. 学習モデルの作成、学習開始

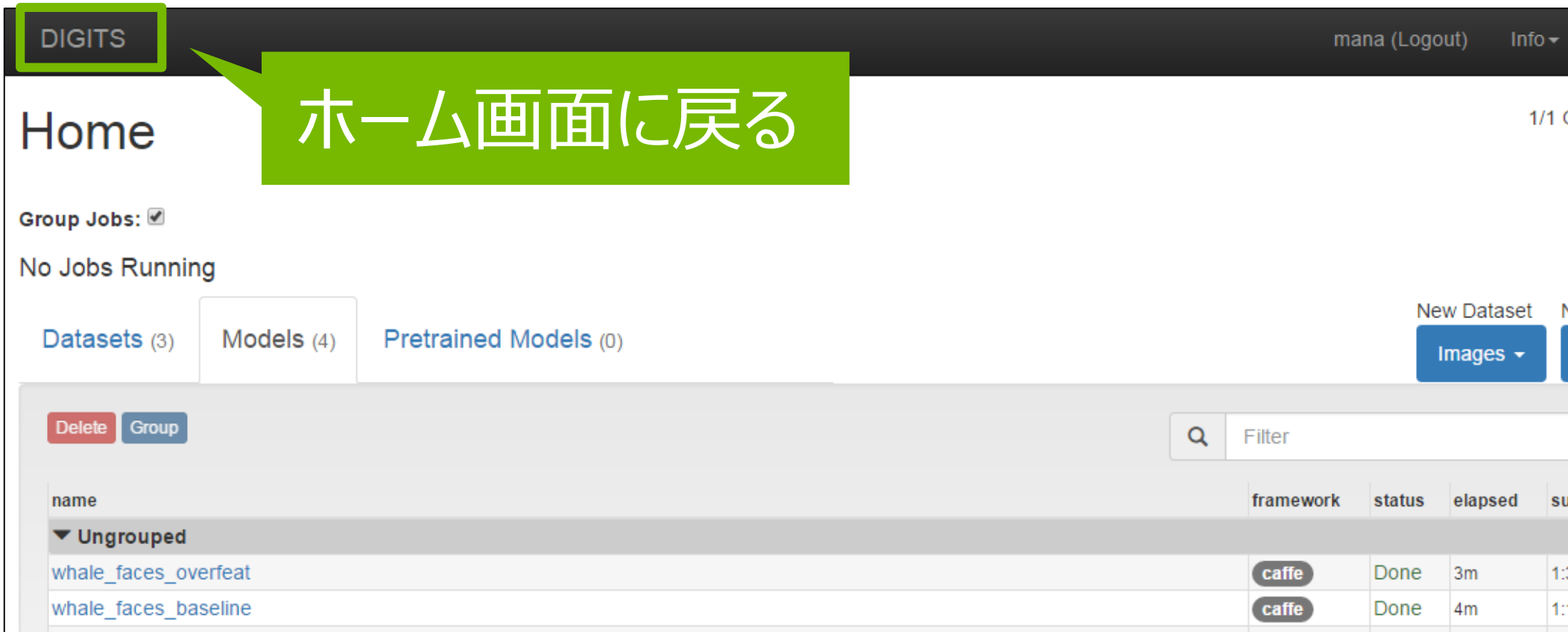
学習

3. 学習済みモデルのテスト

推論
(インファレンス)

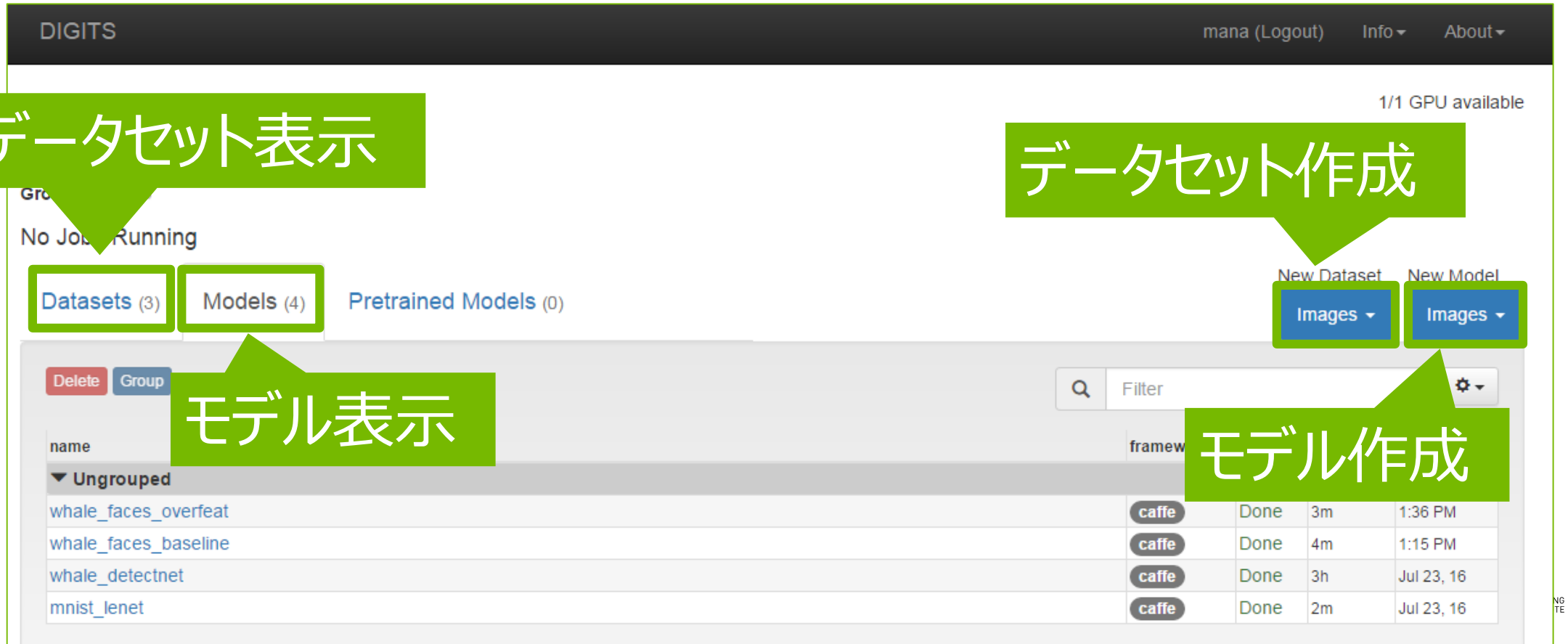
DIGITSによる物体検出入門

DIGITSの使い方: ホーム画面への戻り方



DIGITSによる物体検出入門

DIGITSの使い方: データセット/モデルの表示

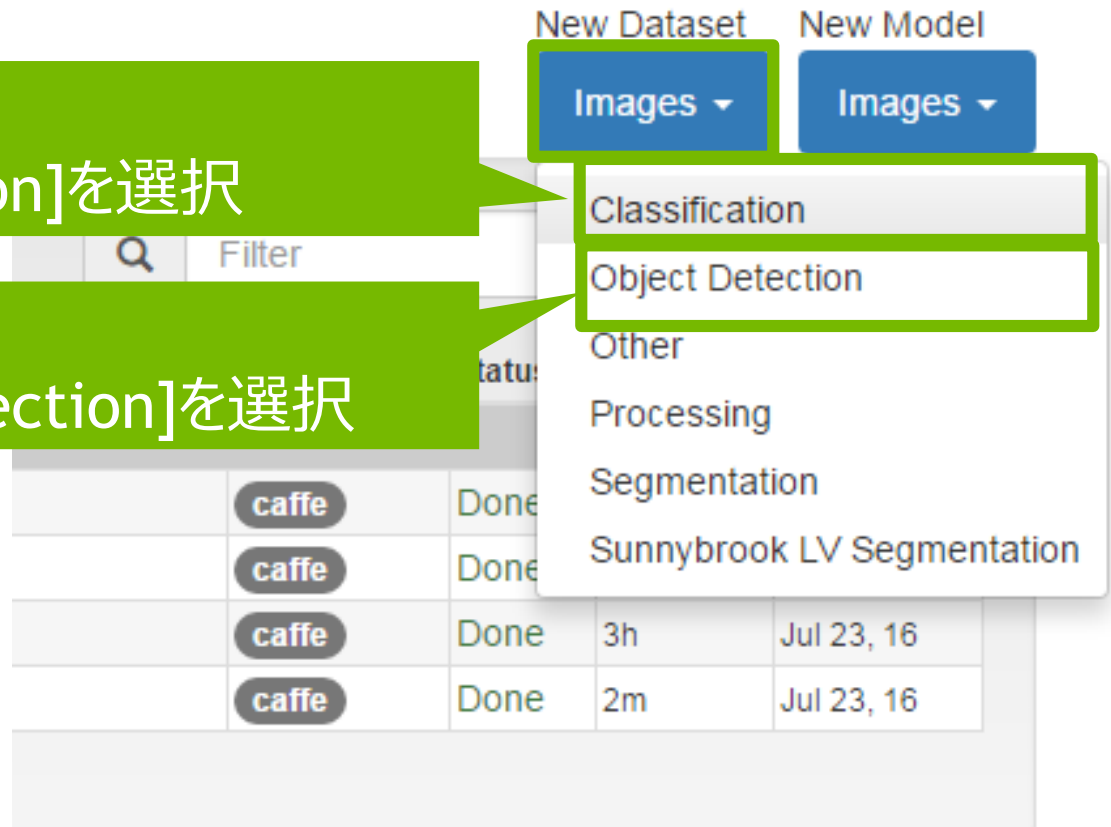


DIGITSによる物体検出入門

DIGITSの使い方: データセット/モデルの作成

画像分類
[Classification]を選択

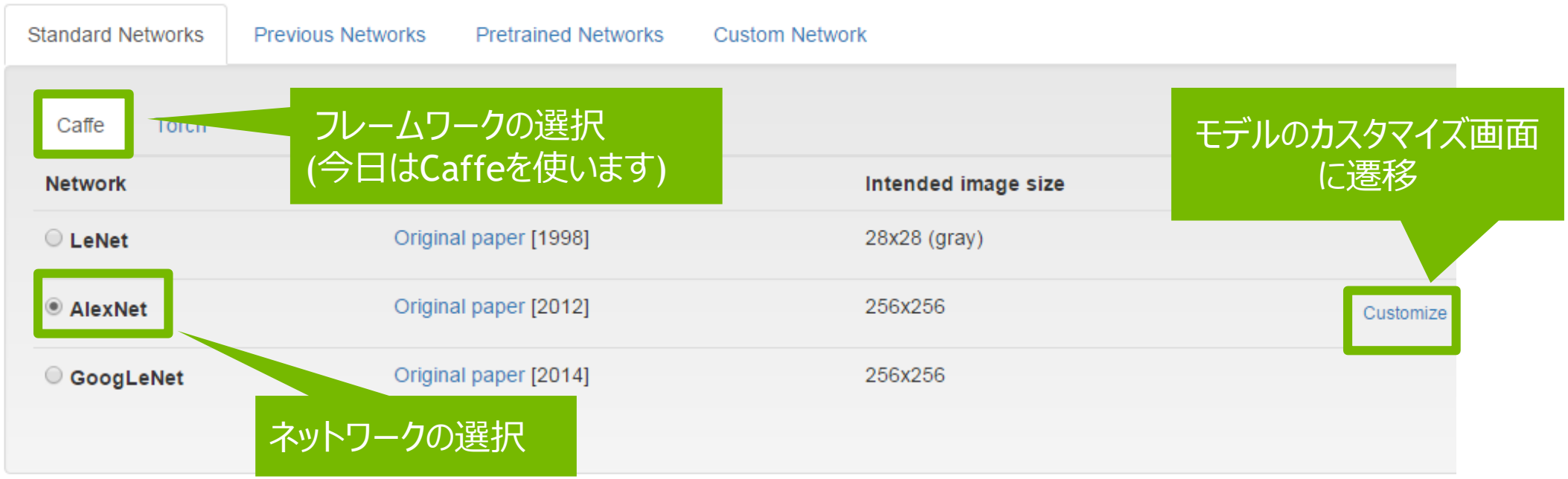
物体検出
[Object Detection]を選択



DIGITSによる物体検出入門

DIGITSの使い方: ネットワーク構造の変更方法

- DIGITSは、Caffeを使って学習を行う事が出来る



DIGITSによる物体検出入門

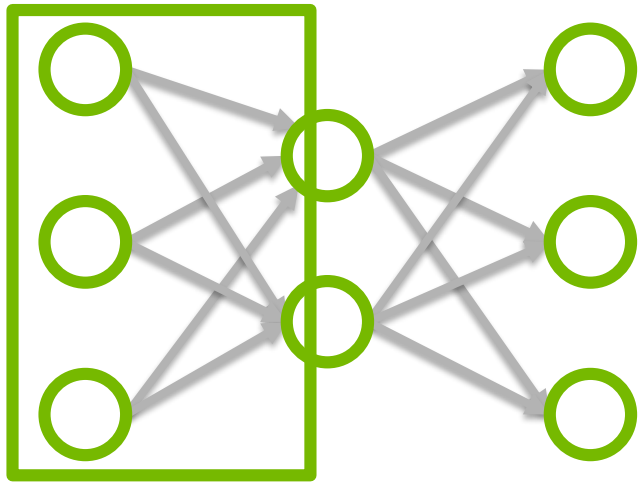
DIGITSの使い方：ネットワーク構造の変更方法

- Caffeのモデル定義ファイル(prototxt)を書き換える

Custom Network Visualize

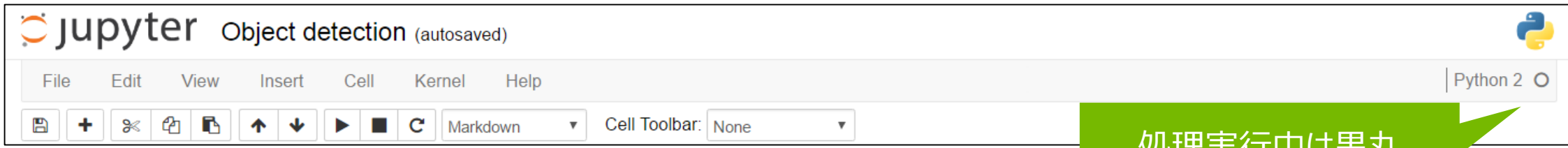
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
```

Layer{}で囲まれた部分が一つの層



DIGITSによる物体検出入門

Jupyter notebook上での処理実行

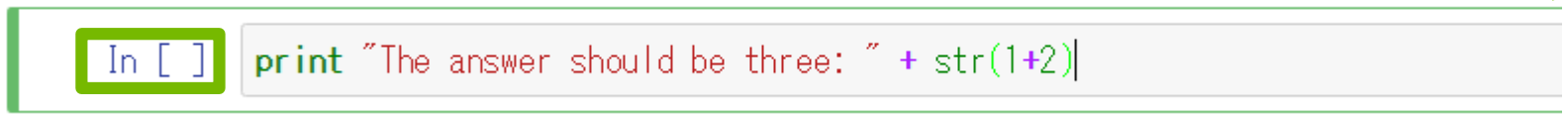


再生ボタンでセル中の処理を実行
(Shift+Enterでも良い)

処理実行中は黒丸
それ以外は白丸

選択されていると緑の枠が
表示される

In[ ]と書いてあるセル中の処理は、実行可能



ラボ開始

ここをクリック

ディープラーニングによる物体検出

クジラの顔を検出する

- データセット

  Right Whale Recognition
  https://www.kaggle.com/c/noaa-right-whale-recognition

物体検出の手法

本ラボでそれぞれ説明します

1. スライディング ウィンドウ

   画像全域をスキャンして検出

2. 候補の生成と分類

   検出対象となる領域の候補を生成し、検出

3. 全畳み込みネットワーク (FCN)

   畳み込みネットワークのみを用いて検出

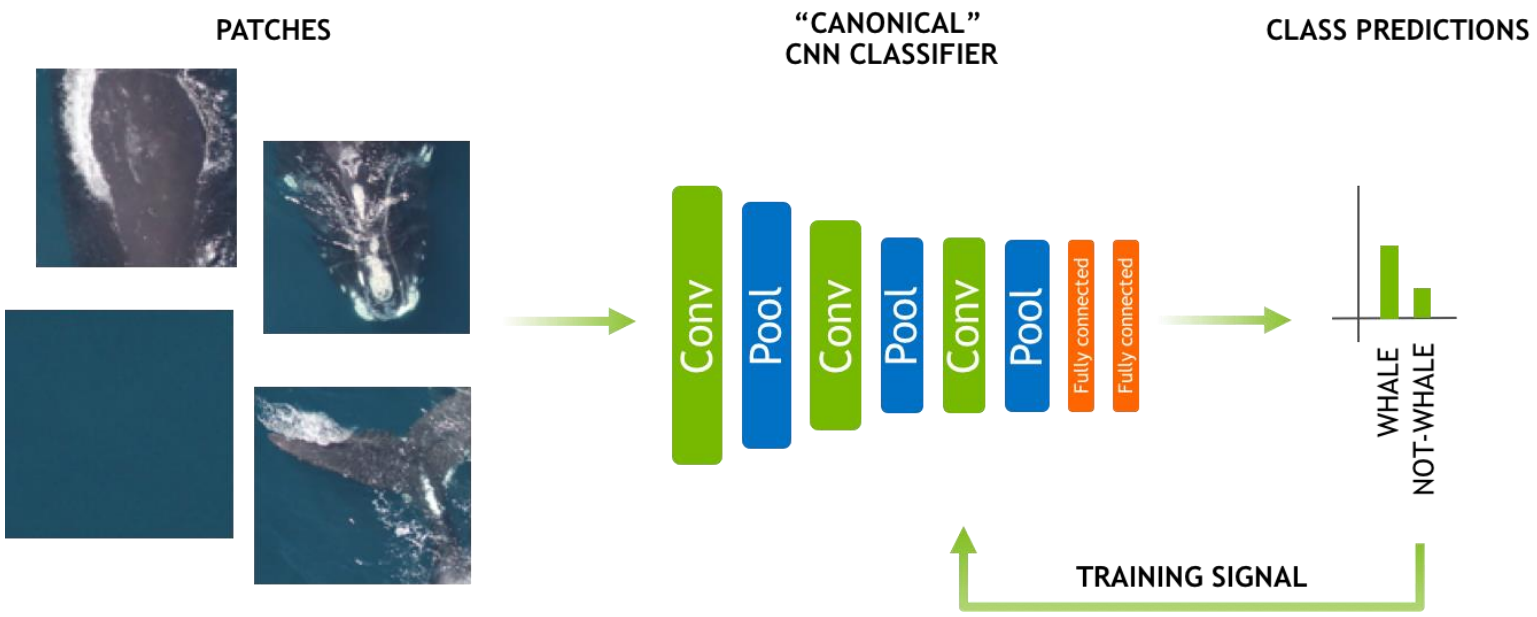
4. DetectNet

   回帰を併用し、物体を囲む矩形領域を推定

1. スライディング ウィンドウ

学習

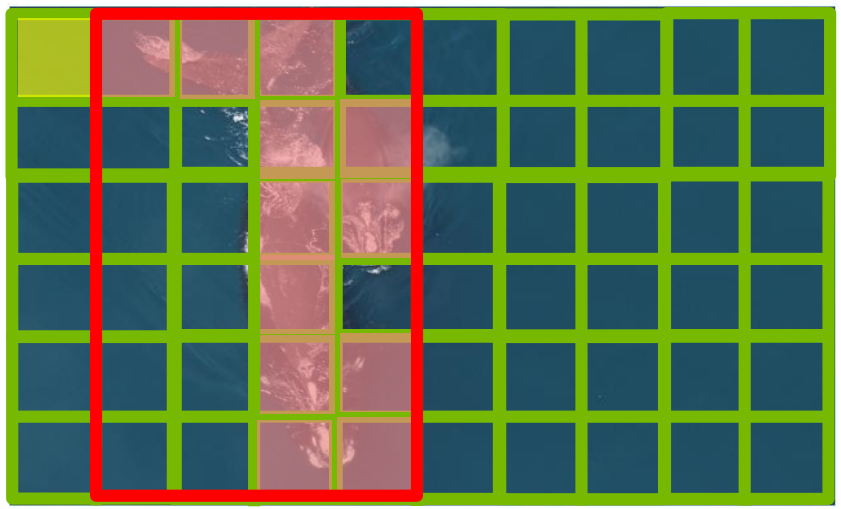
ディープニューラルネットワークに「鯨」か「非鯨」の判別を学習させる

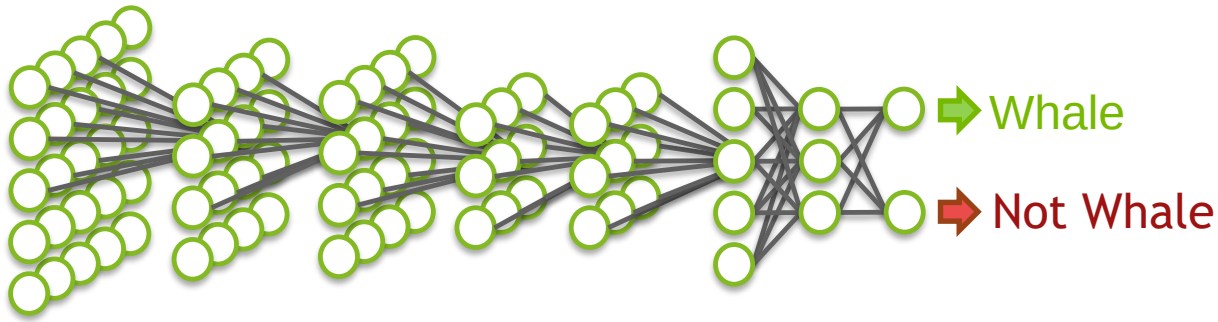


1. スライディング ウィンドウ

検出

画像を分割、それぞれのパッチ(領域)に対して検出を試みる





1. スライディング ウィンドウ

ハンズオン

1. DIGITSを起動
2. 「鯨」「非鯨」を判別するニューラルネットワークを作成

   [New Dataset / Images]->[Classification]からデータセットを作成

   [New Model / Images]->[Classification]からモデルを作成し、学習
3. 学習後のモデルを用いて推論処理を行う

1. スライディング ウィンドウ

DIGITSを開く

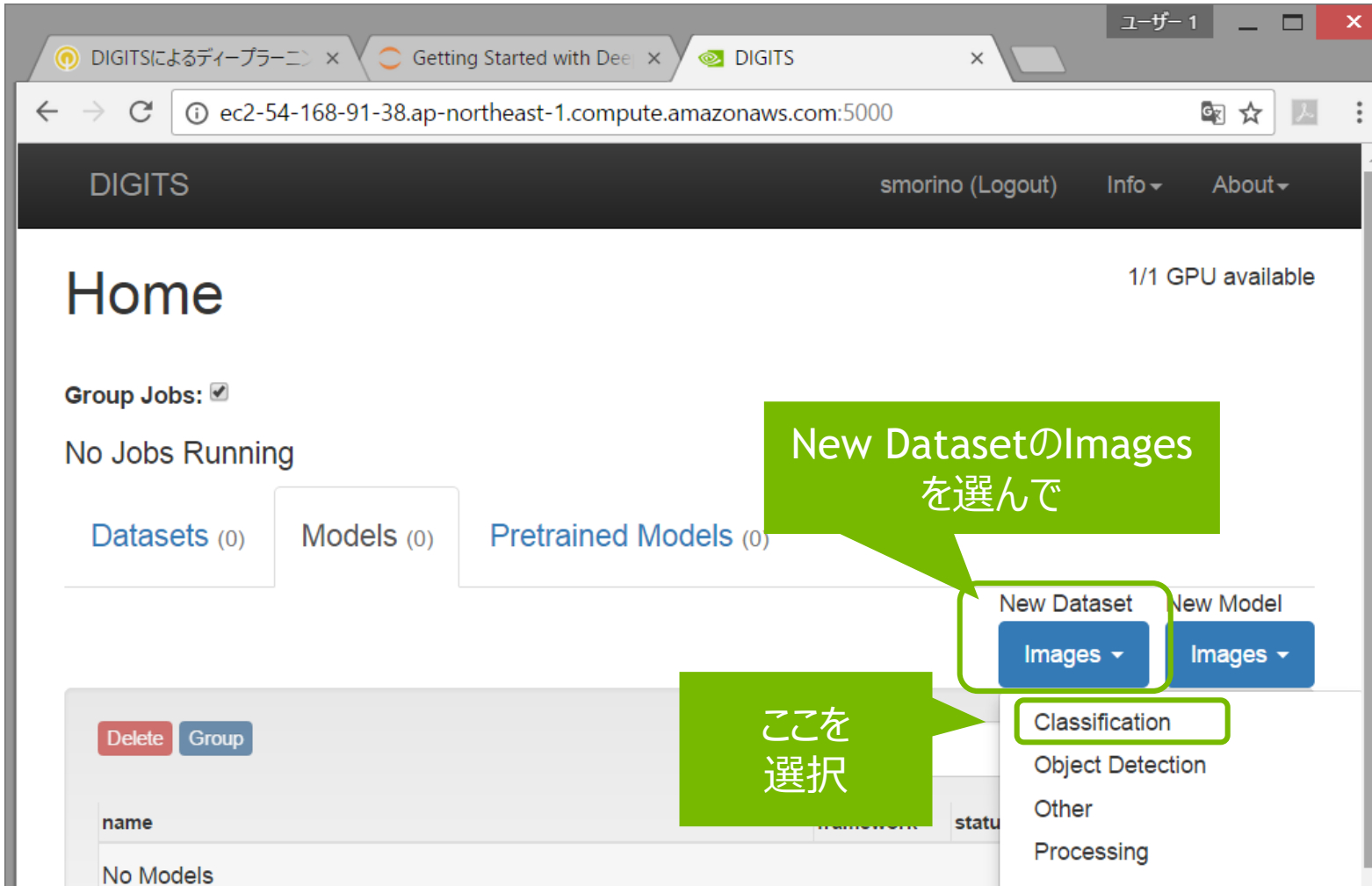
↓

Imagesから

“Classification”を選択

↓

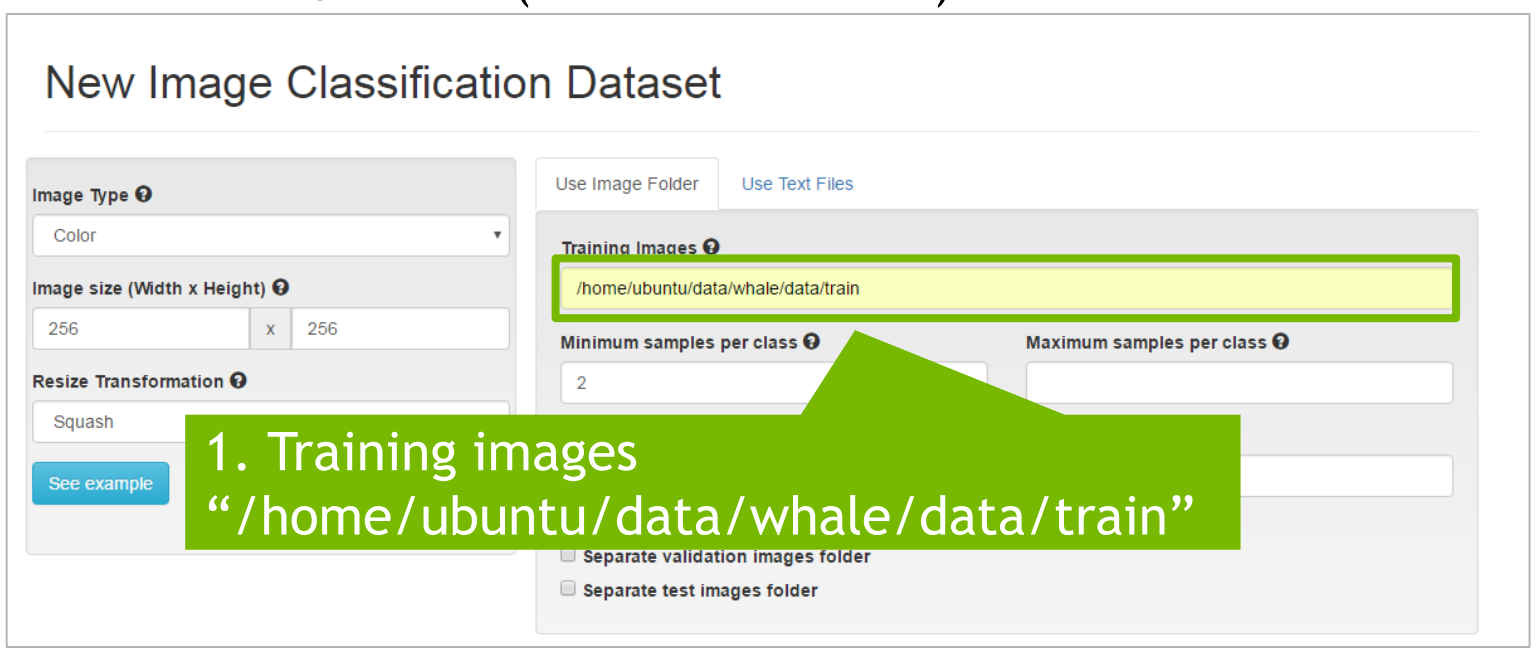
初回だけログインを求められるので何か入力（小文字でお願いします）

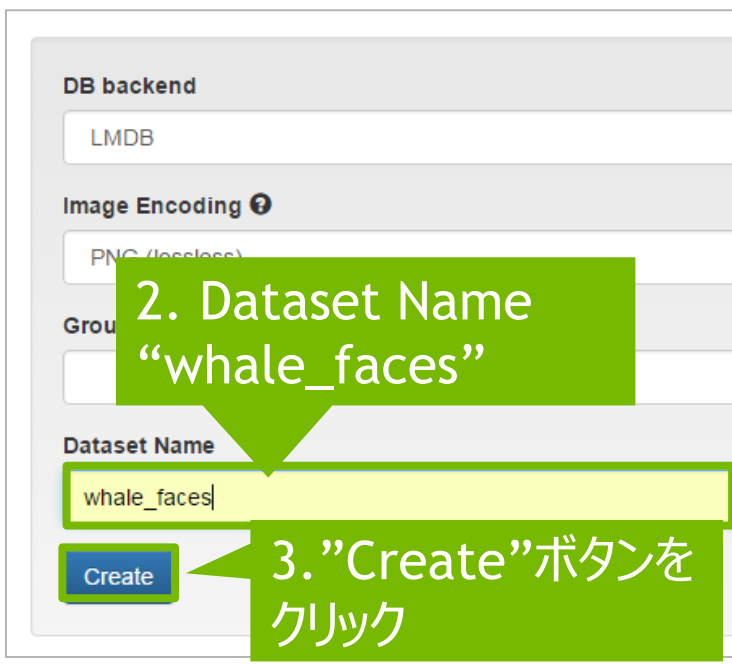


1. スライディング ウィンドウ

データセット作成

1. DIGITSを開く
2. [New Dataset / Images] -> [Classification]を選択する
3. 以下を入力 (2 ～ 3分かかります)

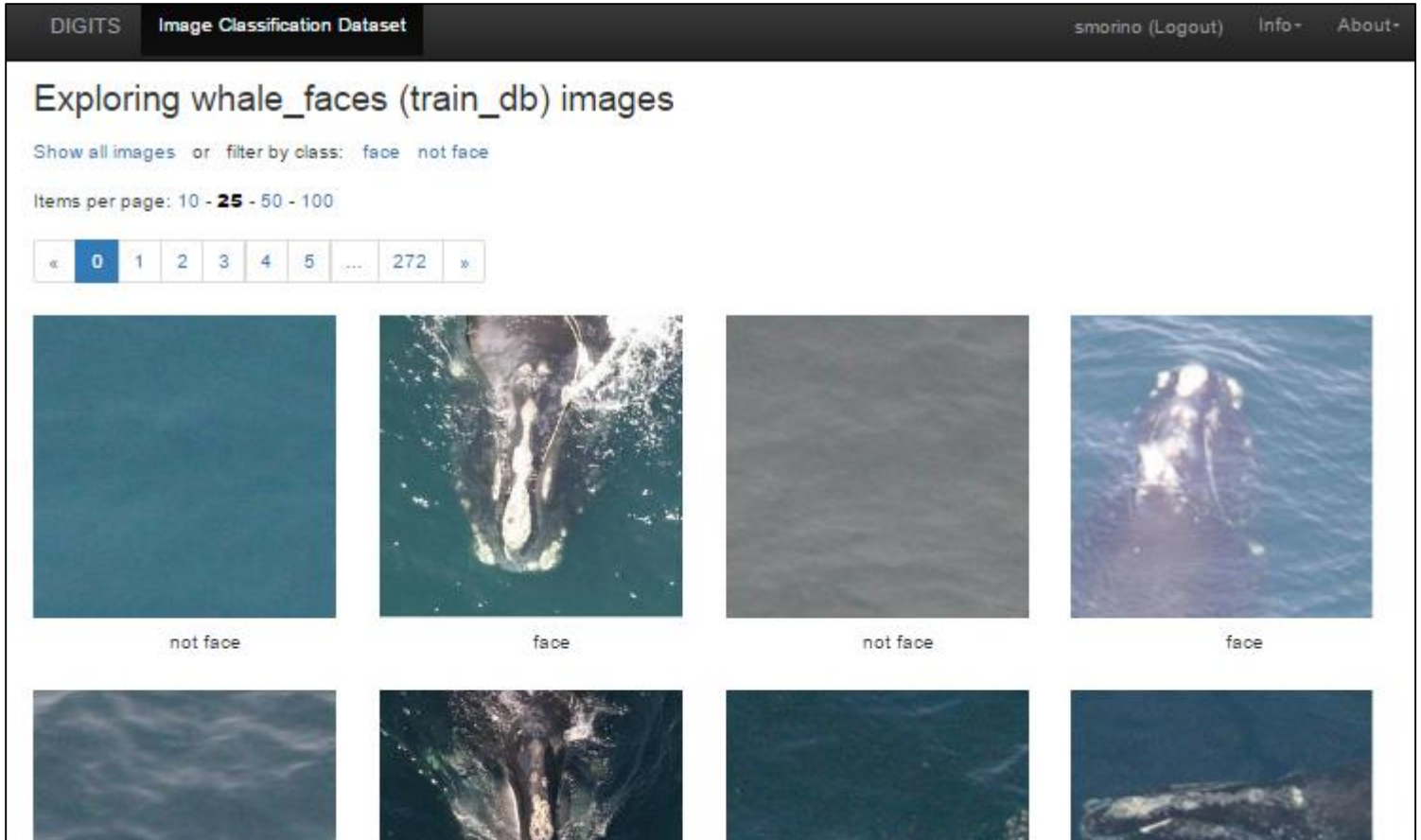




1. スライディング ウィンドウ

学習データを見てみる

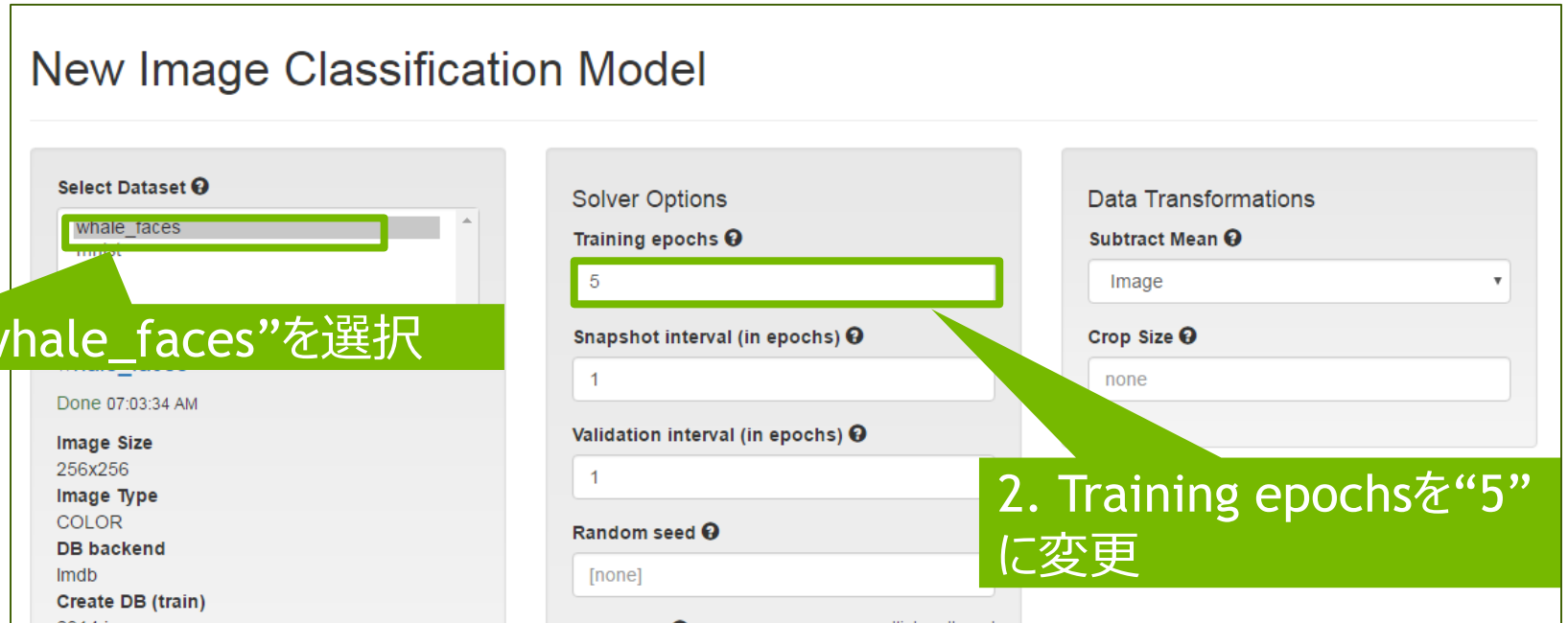
データセットの作成完了後、
“Explorer the db”ボタンを
押すと右の画面



1. スライディング ウィンドウ

モデルの作成、学習

1. DIGITSのホーム画面に移動
2. [New Model / Images] -> [Classification]を選択



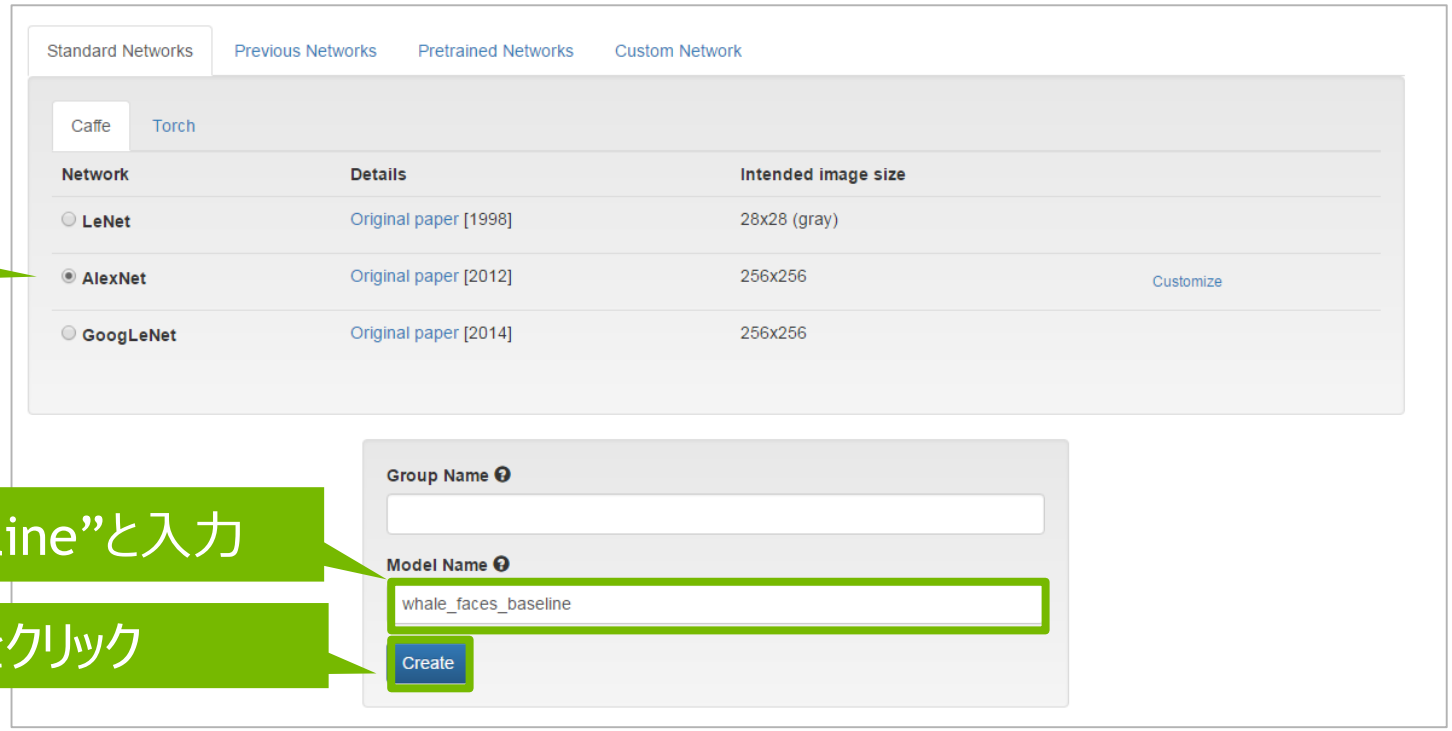
1. スライディング ウィンドウ

モデルの作成、学習

3. AlexNetを選択

4. “whale_faces_baseline”と入力

5. Createボタンをクリック

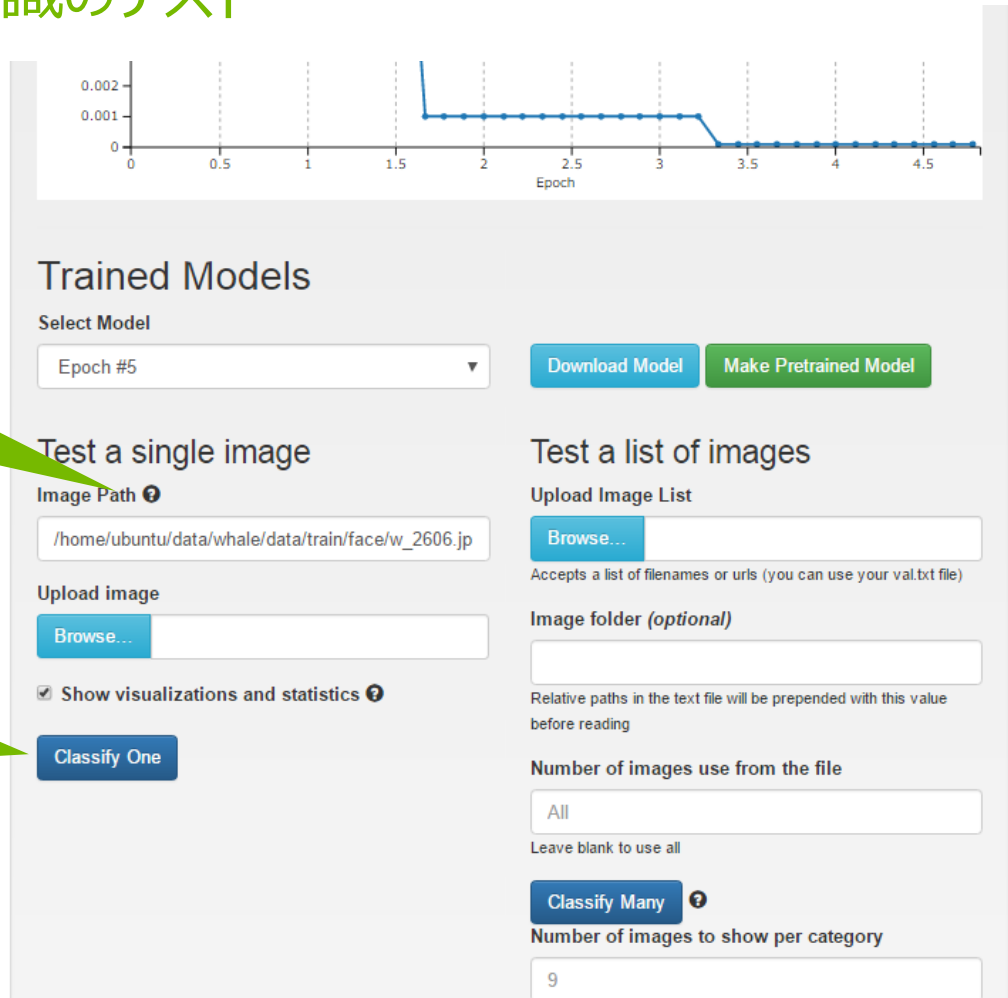


1. スライディング ウィンドウ

1画像を用いた認識のテスト

1. Image Path を入力
/home/ubuntu/data/whale/data/train/face/w_2606.jpg

2. [Classify One]ボタンを押す



1. スライディング ウィンドウ

ジョブIDの確認

学習が終わったら、ジョブIDを控える

1. スライディング ウィンドウ

推論

Jupyter notebookに戻り、先ほど学習したモデルを用いて推論(インファレンス)処理を行う

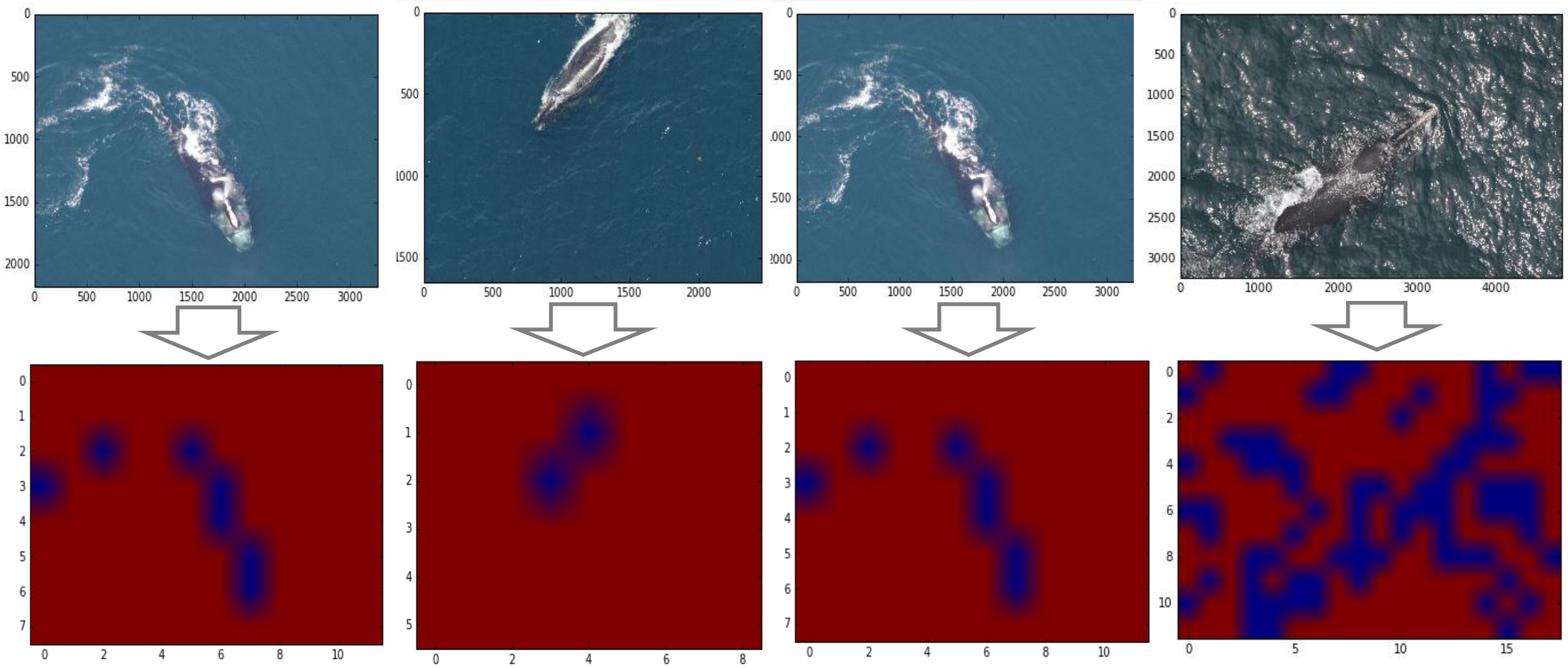
In [3]:

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as p
import caffe
import time

JOB_NUM = '20170416-081849-e436'  ## Remember to set this to be the job number for your model

MODEL_FILE = '/home/ubuntu/digits/digits/jobs/' + JOB_NUM + '/deploy.prototxt'                # Do not change
PRETRAINED = '/home/ubuntu/digits/digits/jobs/' + JOB_NUM + '/snapshot_iter_270.caffemodel'    # Do not change

# Choose a random image to test against
RANDOM_IMAGE = str(np.random.randint(10))
IMAGE_FILE = 'data/samples/w_' + RANDOM_IMAGE + '.jpg'

# Tell Caffe to use the GPU
caffe.set_mode_gpu()
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(MODEL_FILE, PRETRAINED,
                       channel_swap=(2,1,0),
                       raw_scale=255,
```

先ほど作成した、モデルのジョブIDに変更

1. スライディング ウィンドウ

認識結果



1. スライディング ウィンドウ

まとめ

高精度な、クジラの顔画像分類器が作成された。

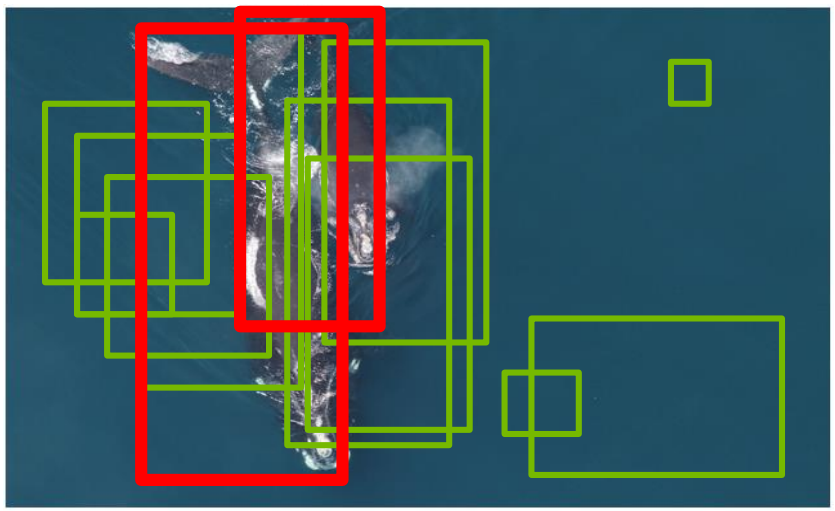
認識精度 約98%

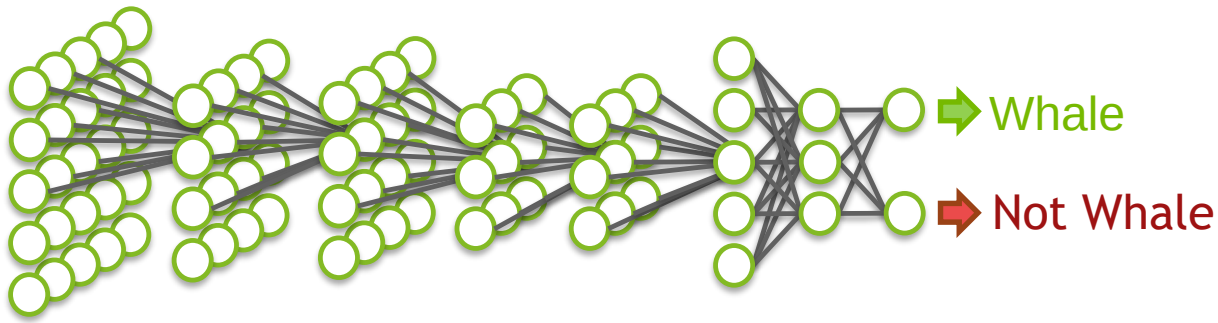
処理時間 約10秒

2. 候補の生成と分類

考え方

顔に対応する領域候補を生成
ニューラルネットワーク(CNN)で検出を行う





注. ハンズオンは行いません

2. 候補の生成と分類

特性

利点

対象となる候補領域数の削減による高速化

候補生成アルゴリズム次第で、物体の位置特定精度を向上可能

欠点

より複雑な多段処理パイプライン

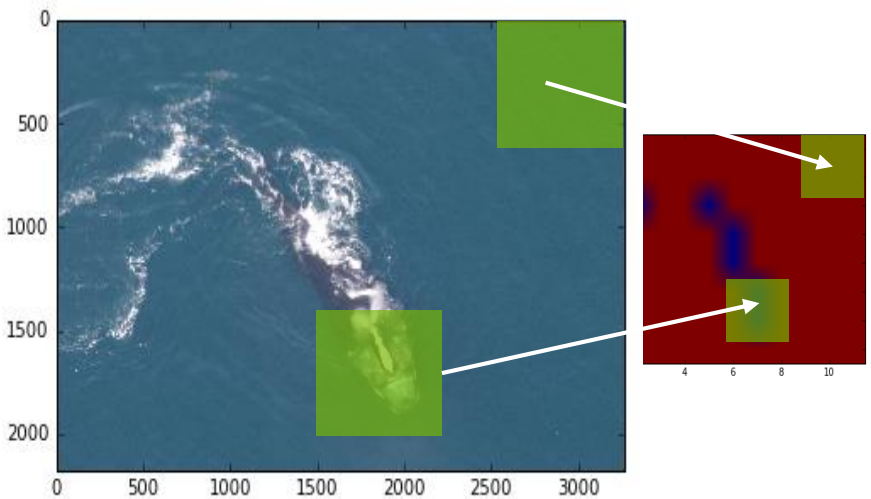
候補を生成するための追加モデルの構築や学習

誤検出率は低くない

生成される候補の数によって推論時間が変わる

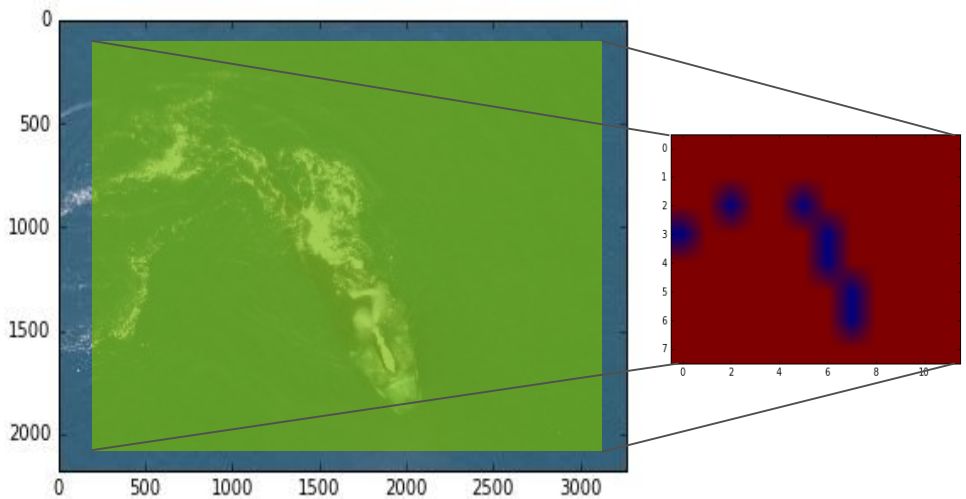
3. 全畳み込みネットワーク (FCN)

考え方



スライディング ウインドウ

一つの領域で、一個の判定結果



FCN

一つの領域から、ヒートマップとして
判定結果を出力

3. 全畳み込みネットワーク (FCN)

ハンズオン

1. 検出方法1で作成した「鯨」と「非鯨」を分類するモデルを再利用

   [Clone Job]でモデルを複製

2. 全畳み込みネットワークに書き換える

   ネットワーク定義を書きかえる　(fc6 – fc8まで)

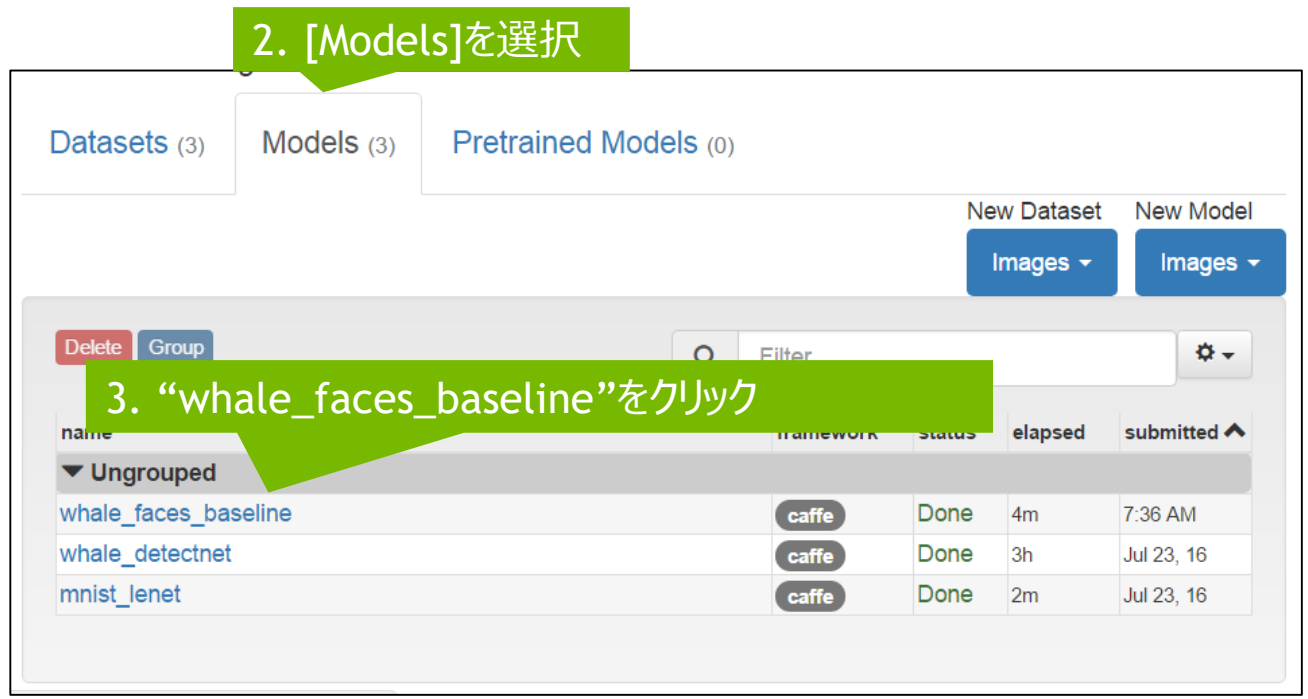
   再度、学習を行う

3. 学習後のモデルを用いて推論処理を行う

3. 全畳み込みネットワーク (FCN)

モデル作成

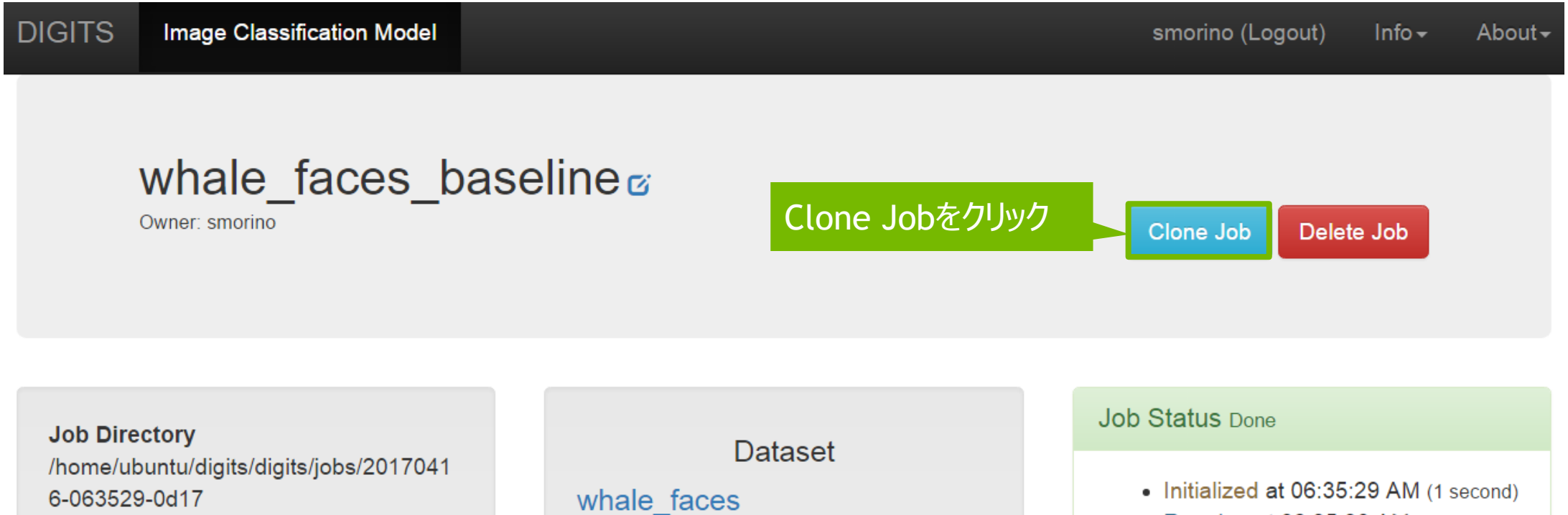
1. DIGITSのホーム画面に戻る
2. [Models]を選択
3. ”whale_faces_baseline”をクリック
(先ほど作成したモデルに移動)



3. 全畳み込みネットワーク (FCN)

ハンズオン

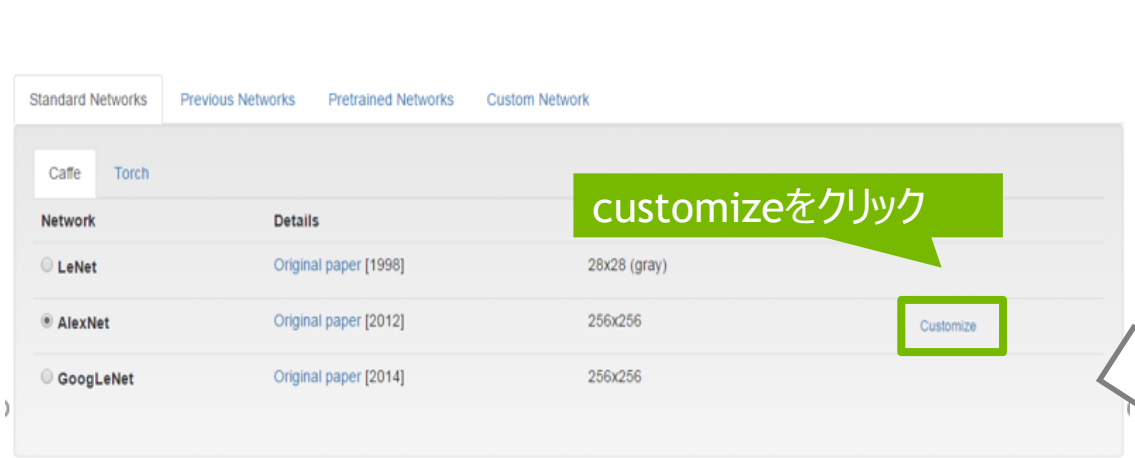
1. [Clone Job]でモデルを複製



3. 全畳み込みネットワーク (FCN)

ネットワーク定義の書き換え

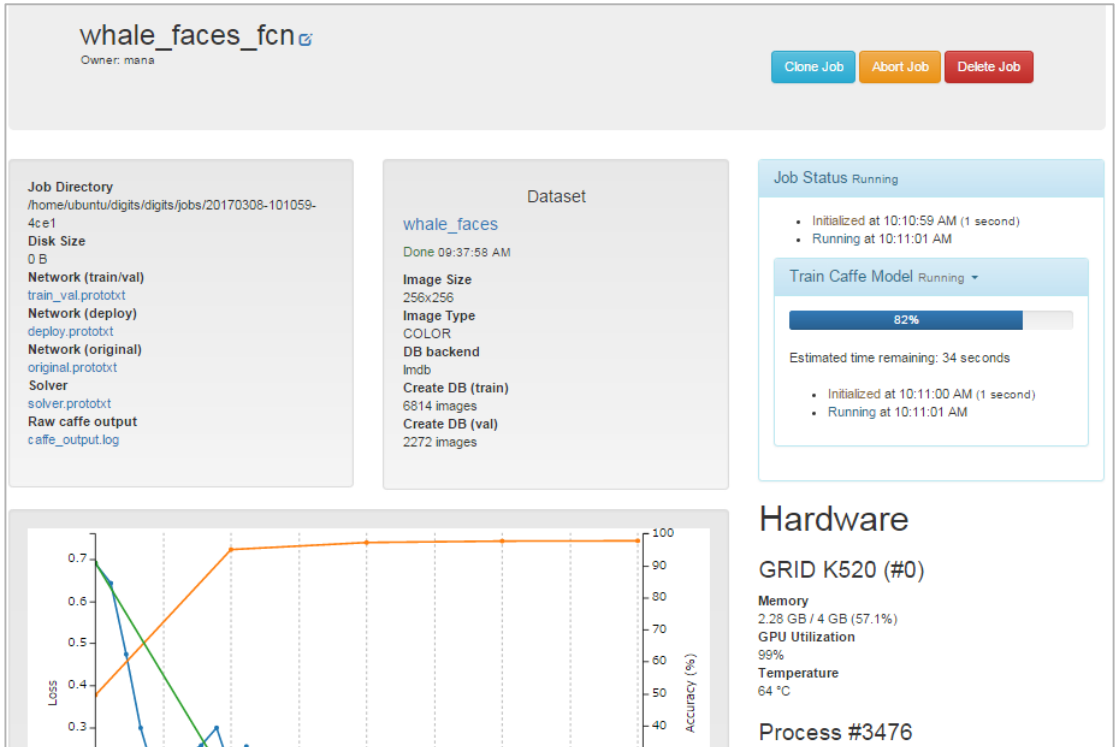
1. [Standard Networks]->[(AlexNet横の)customize]をクリック
2. テキストボックス内部のネットワーク定義を書き換える



3. 全畳み込みネットワーク (FCN)

学習の開始

[Model Name]を”whale_faces_fcn”として、[Create]ボタンをクリック



3. 全畳み込みネットワーク (FCN)

推論

1. 学習が終わったら、ジョブIDを控える

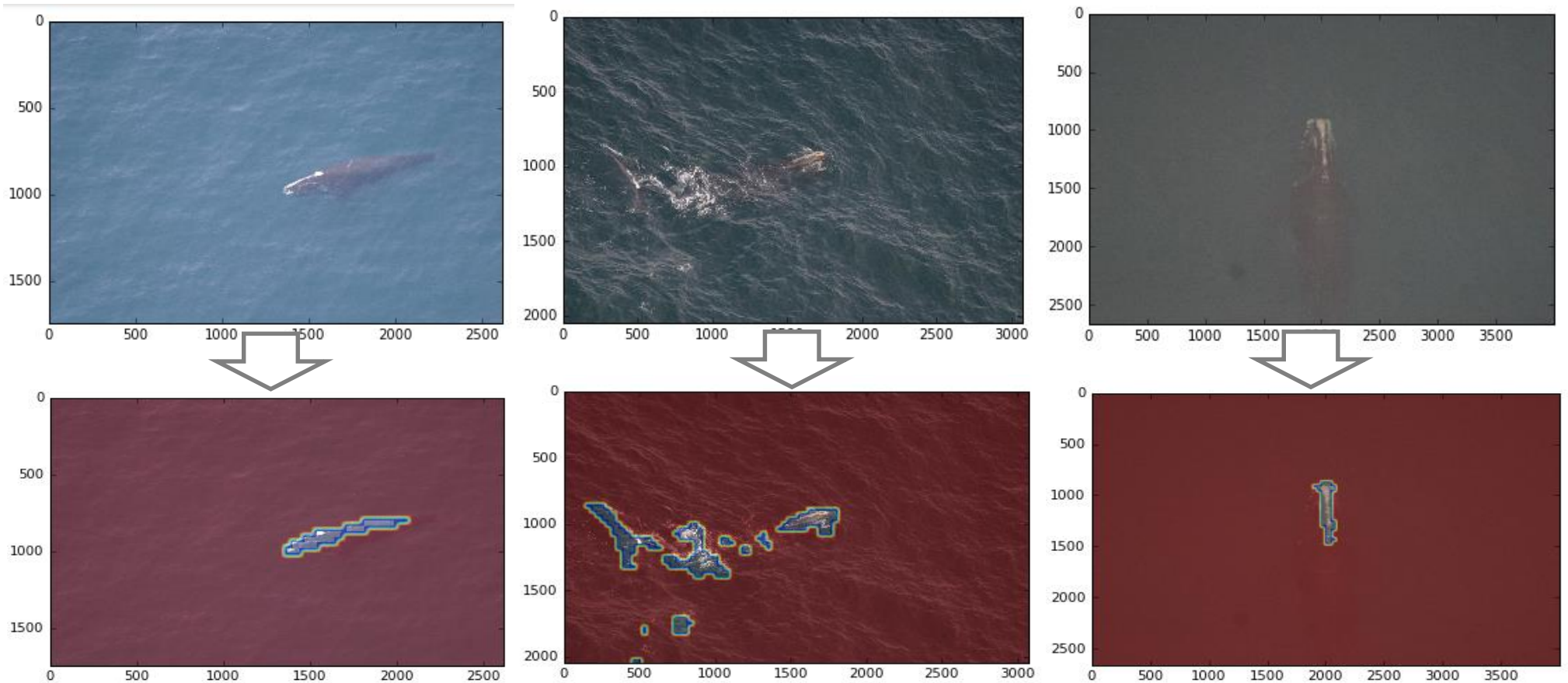
2. スクリプト中のジョブIDを更新して、実行

```
In [ ]: %matplotlib inline

        import numpy as np
        import matplotlib.pyplot as plt
        import caffe
        import copy
        from scipy.misc import imresize
        import time

        JOB_NUM = '20170416-083932-709c' ## Remember to set this to be the job number for your model

        MODEL_FILE = '/home/ubuntu/digits/digits/jobs/' + JOB_NUM + '/deploy.prototxt'
        PRETRAINED = '/home/ubuntu/digits/digits/jobs/' + JOB_NUM + '/snapshot_iter_270.caffemodel'

        # Choose a random image to test against
        RANDOM_IMAGE = str(np.random.randint(10))
        IMAGE_FILE = 'data/samples/w_' + RANDOM_IMAGE + '.jpg'
```

モデルのジョブIDに変更

3. 全畳み込みネットワーク (FCN)

検出結果



3. 全畳み込みネットワーク (FCN)

まとめ

処理時間

スライディングウインドウ ：約10 秒

全畳み込みネットワーク：約1.5秒

(画像の大きさに応じ、1.5 ～ 6秒程度)

性質

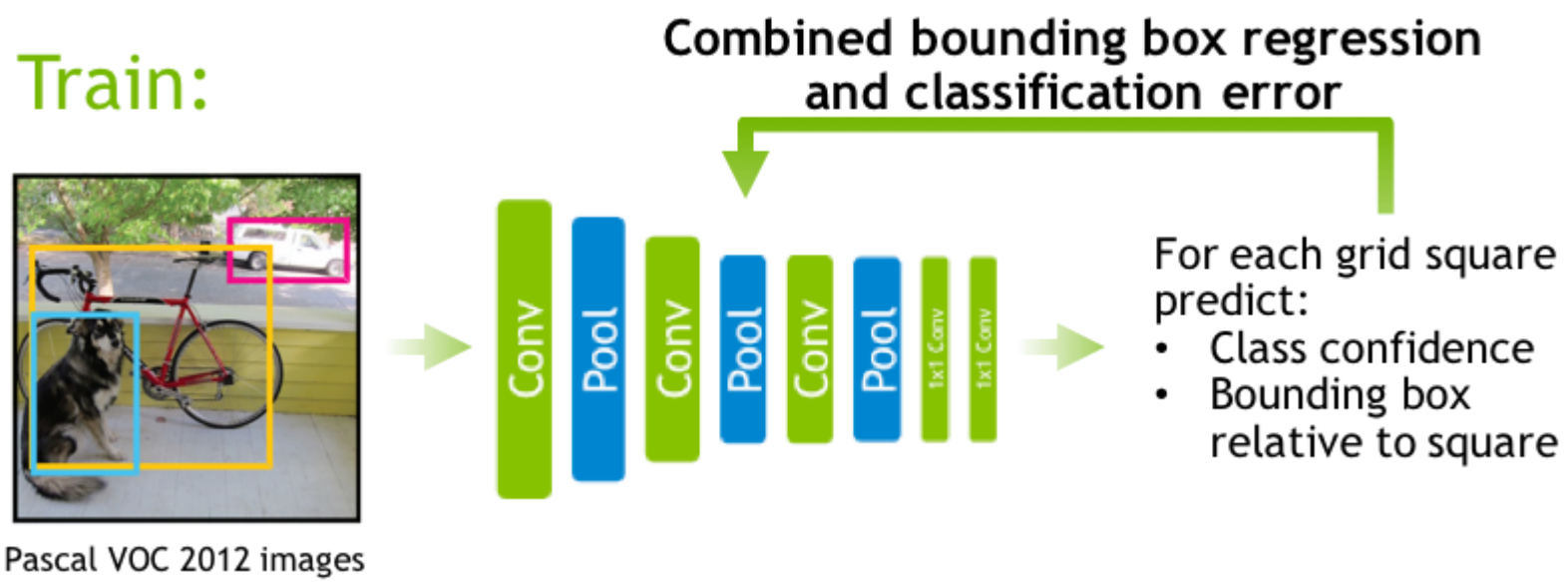
多くの場合、FCNの方が高精度、より多くのクジラを発見。

砕ける波や海面に反射する日光により混乱することもあるが、
適切なデータ拡大を使用することにより、誤検出を軽減。

4. DETECTNET

考え方: 学習時

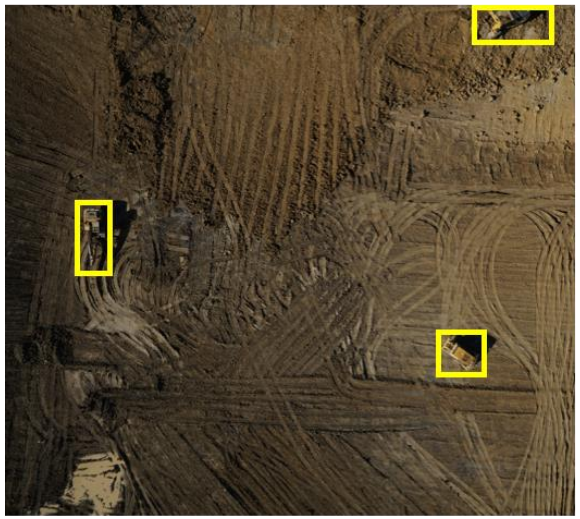
1. 画像をグリッドに分割
2. グリッド単位でバウンディングボックス・信頼度を予測できるよう、ネットワークに学習させる



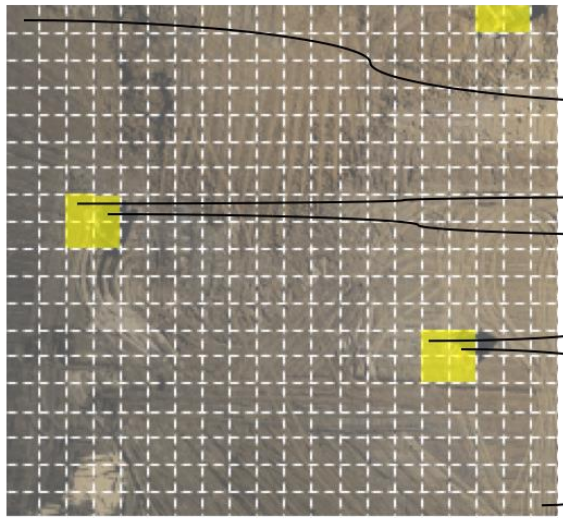
4. DETECTNET

考え方: 学習時

学習用データの考え方



Training image with bounding box annotations



Bounding boxes mapped to grid squares

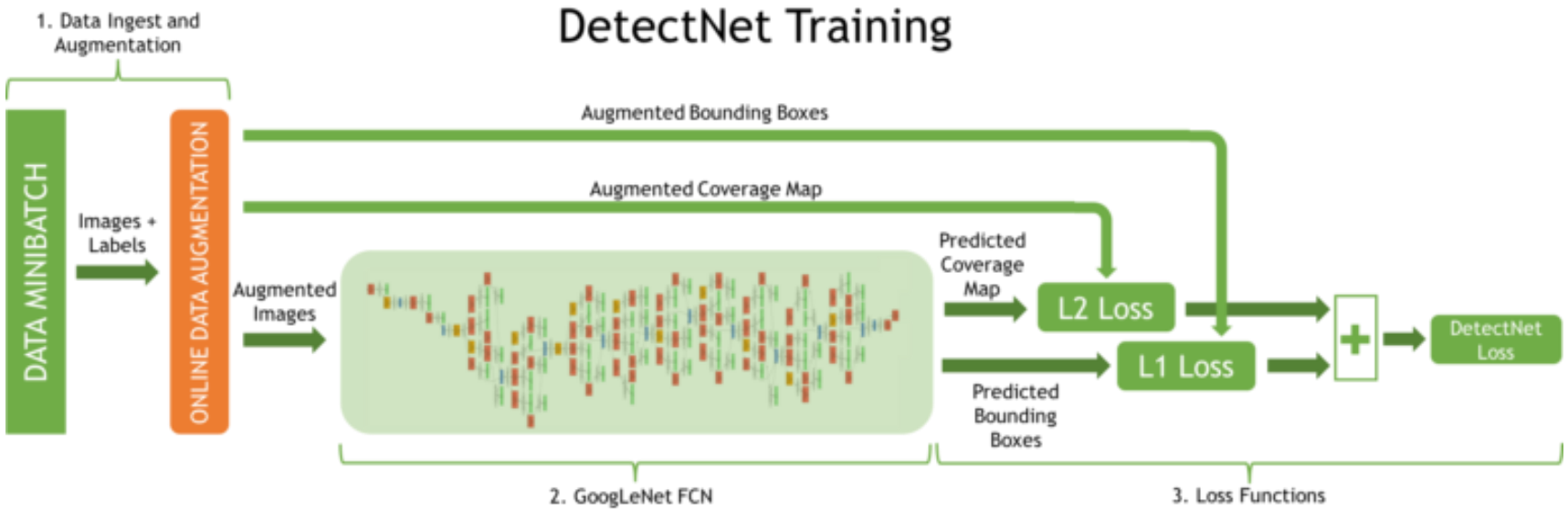
Bounding box coordinates in pixels
relative to center of grid square

class	x_1	y_1	x_2	y_2	coverage
dontcare	0	0	0	0	0
…	…	…	…	…	…
digger	-2	-8	18	24	1
digger	-18	-8	2	24	1
…	…	…	…	…	…
digger	-6	-8	22	24	1
digger	-24	-8	8	24	1
…	…	…	…	…	…
dontcare	0	0	0	0	0

DetectNet input data representation

4. DETECTNET

考え方: ネットワーク、文献

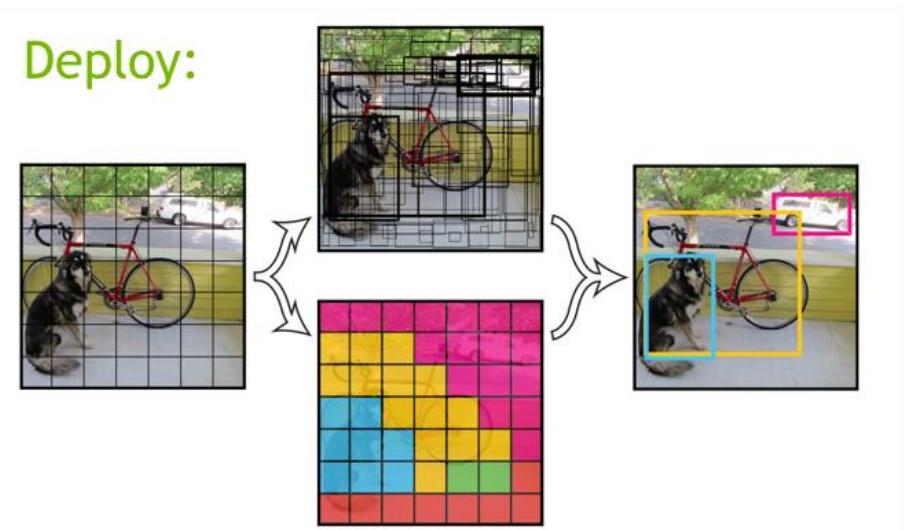


DetectNet: Deep Neural Network for Object Detection in DIGITS
https://devblogs.nvidia.com/parallelforall/detectnet-deep-neural-network-object-detection-digits/

4. DETECTNET

考え方: 推論

1. 画像をグリッドに分割
2. グリッドごとに、バウンディングボックス・占有度(Coverage)を推論
3. バウンディングボックスをクラスタリング
4. 精度(mAP)を算出



4. DETECTNET

ハンズオン

1. DIGITSでDetectNetを使った鯨の位置検出を行うモデルを作成する
   - [New Datasets / Images]->[Object Detection]から、“whales_detectnet”という名前でデータセットを作成する
   - 事前に準備されている“whale_detectnet”を選択し、[Clone Job]でモデルを複製
   - 設定を行い、再学習
2. DIGITSのテスト機能を使い、学習済みのモデルのテストを行う

4. DETECTNET

データセット作成

1. [Datasets]->[Object Detection]を選択
2. 以下を入力

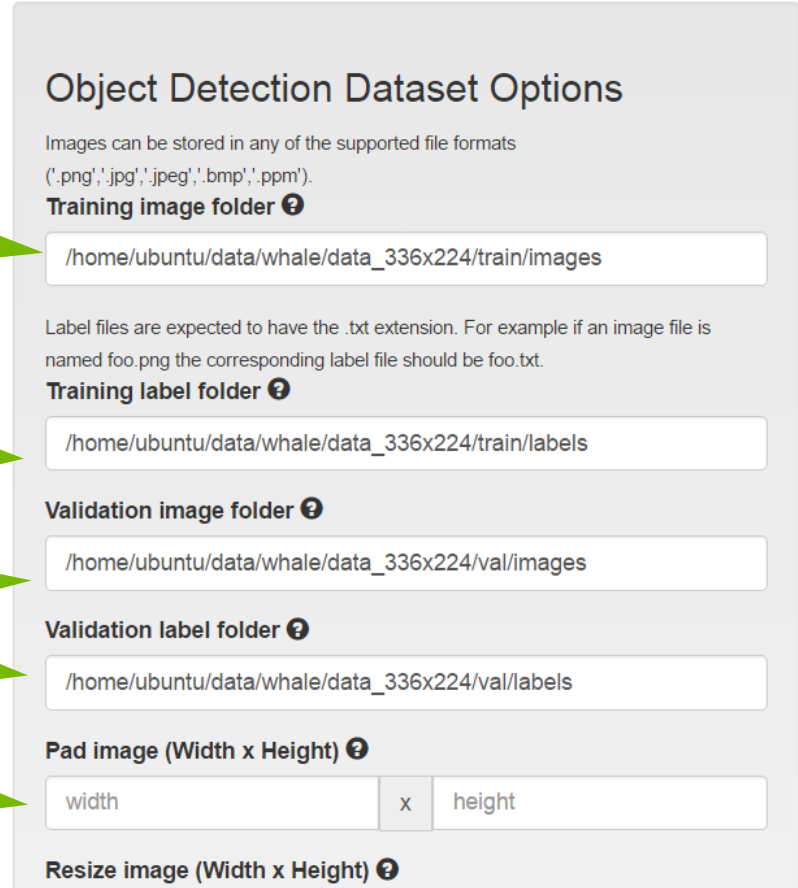
1. Training image folder
”/home/ubuntu/data/whale/data_336x224/train/images”

2. Training label folder
“/home/ubuntu/data/whale/data_336x224/train/labels”

3. Validation image folder
“/home/ubuntu/data/whale/data_336x224/val/images”

4. Validation label folder
“/home/ubuntu/data/whale/data_336x224/val/labels”

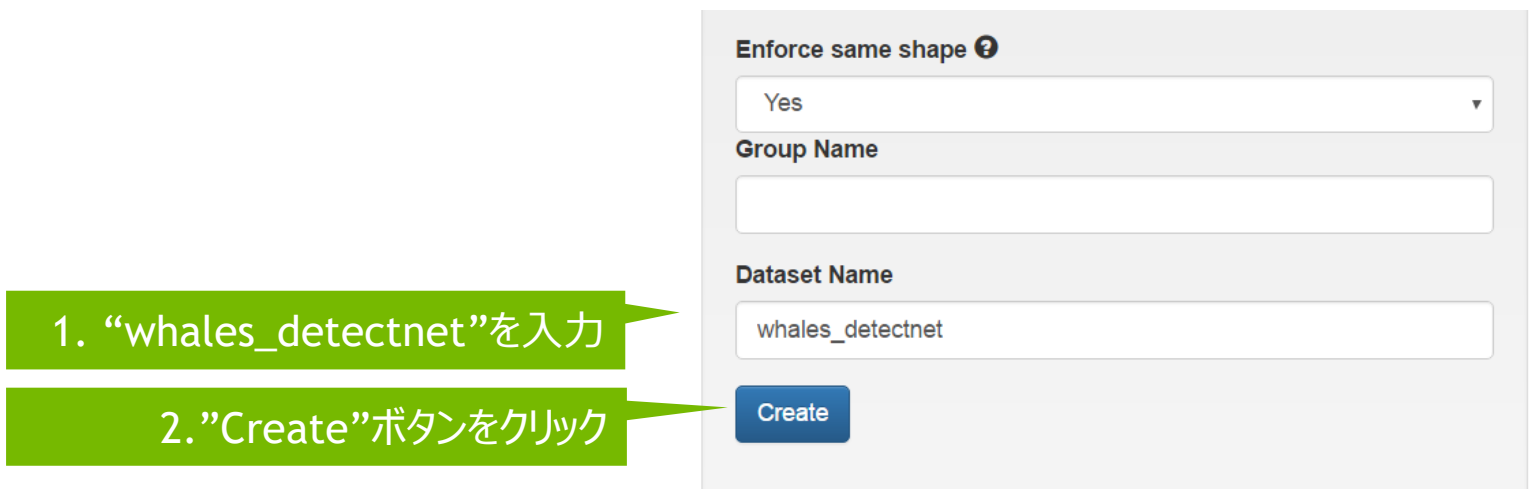
5. Pad Image
値が入っている場合、削除し、空にする



4. DETECTNET

データセット作成

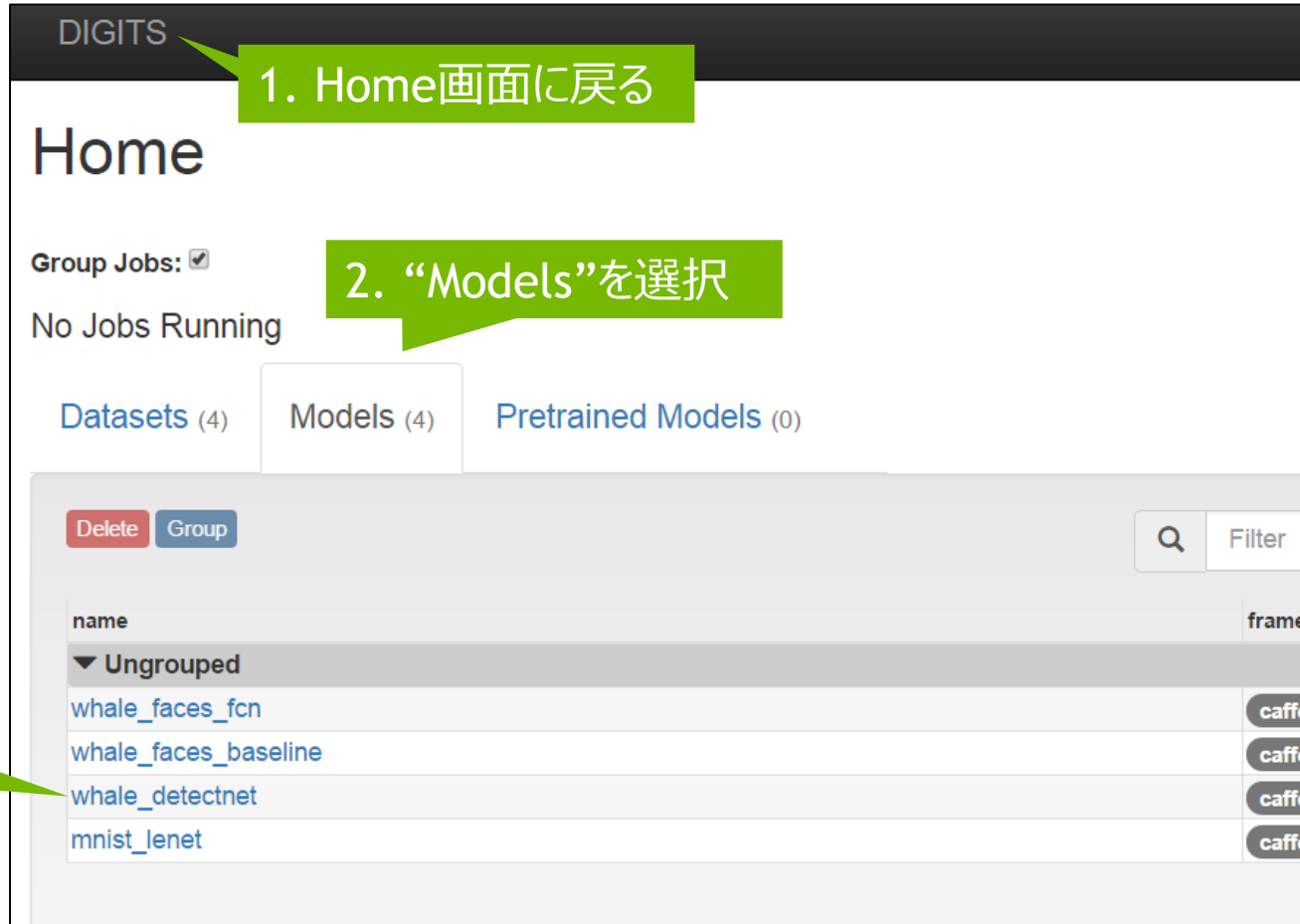
1. ”whales_detectnet”という名前で、データセットを作成する



4. DETECTNET

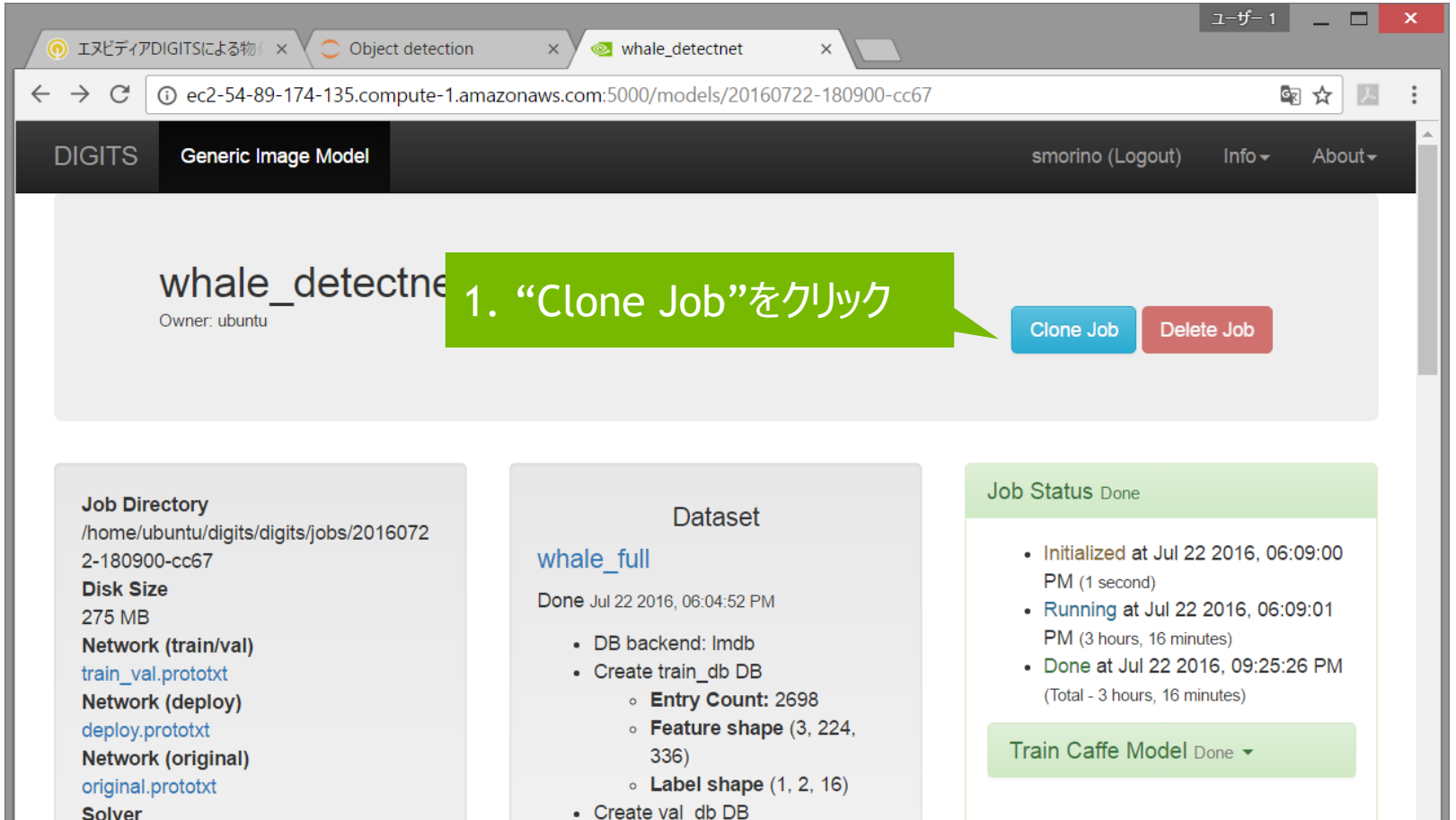
作成済みのモデルを使う

1. DIGITSのHome画面に戻る
2. “Models”を選択
3. “whale_detectnet”をクリック



4. DETECTNET

作成済みのモデルを使う

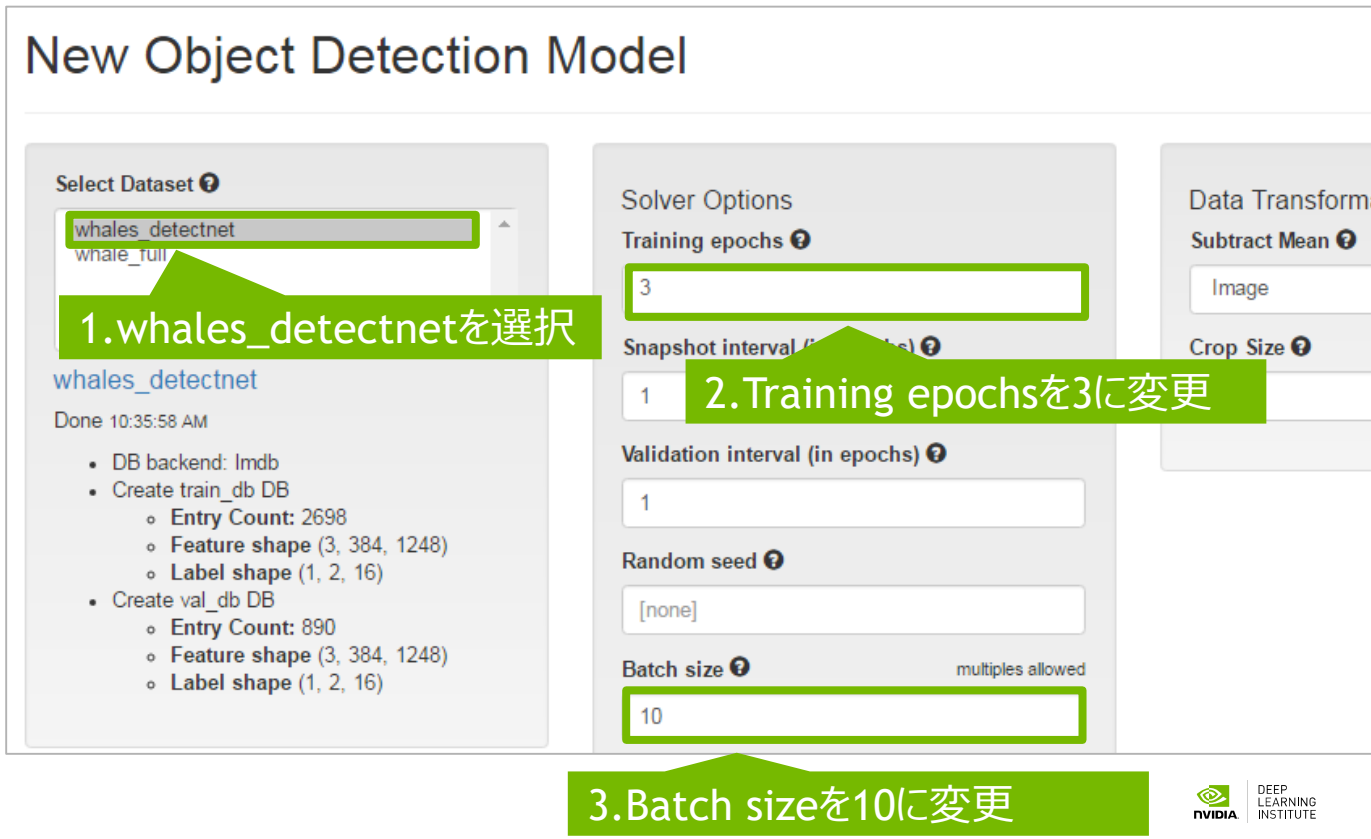


4. DETECTNET

学習: 3 epochのみ

1. “whale_detectnet”を選択後、[Clone Job]でモデルを複製
2. 右の設定を行う

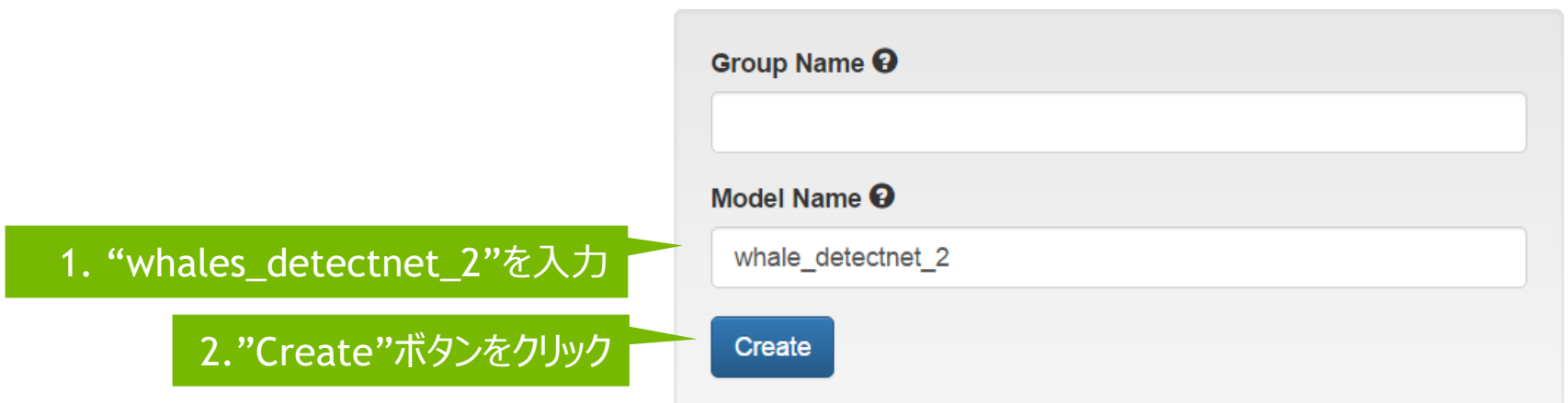
時間の都合で3 epochだけ、
学習します（**8分ほどかかります**）



4. DETECTNET

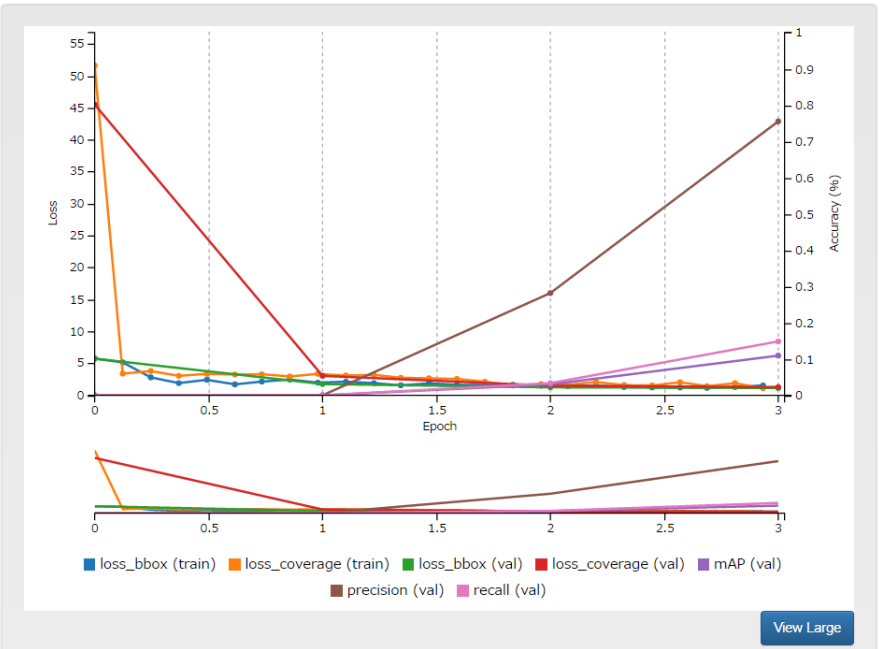
学習: 3 epochのみ

1. モデル名を”whales_detectnet_2”とする
2. “Create”ボタンをクリックして、学習を開始する

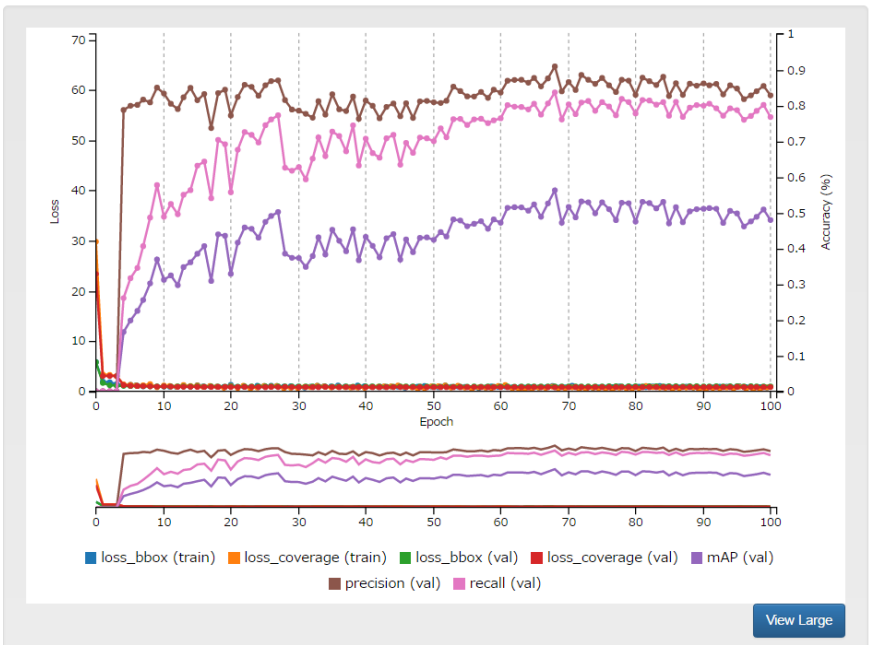


4. DETECTNET

学習: 3 epoch ～ 100 epoch



3 epochまで



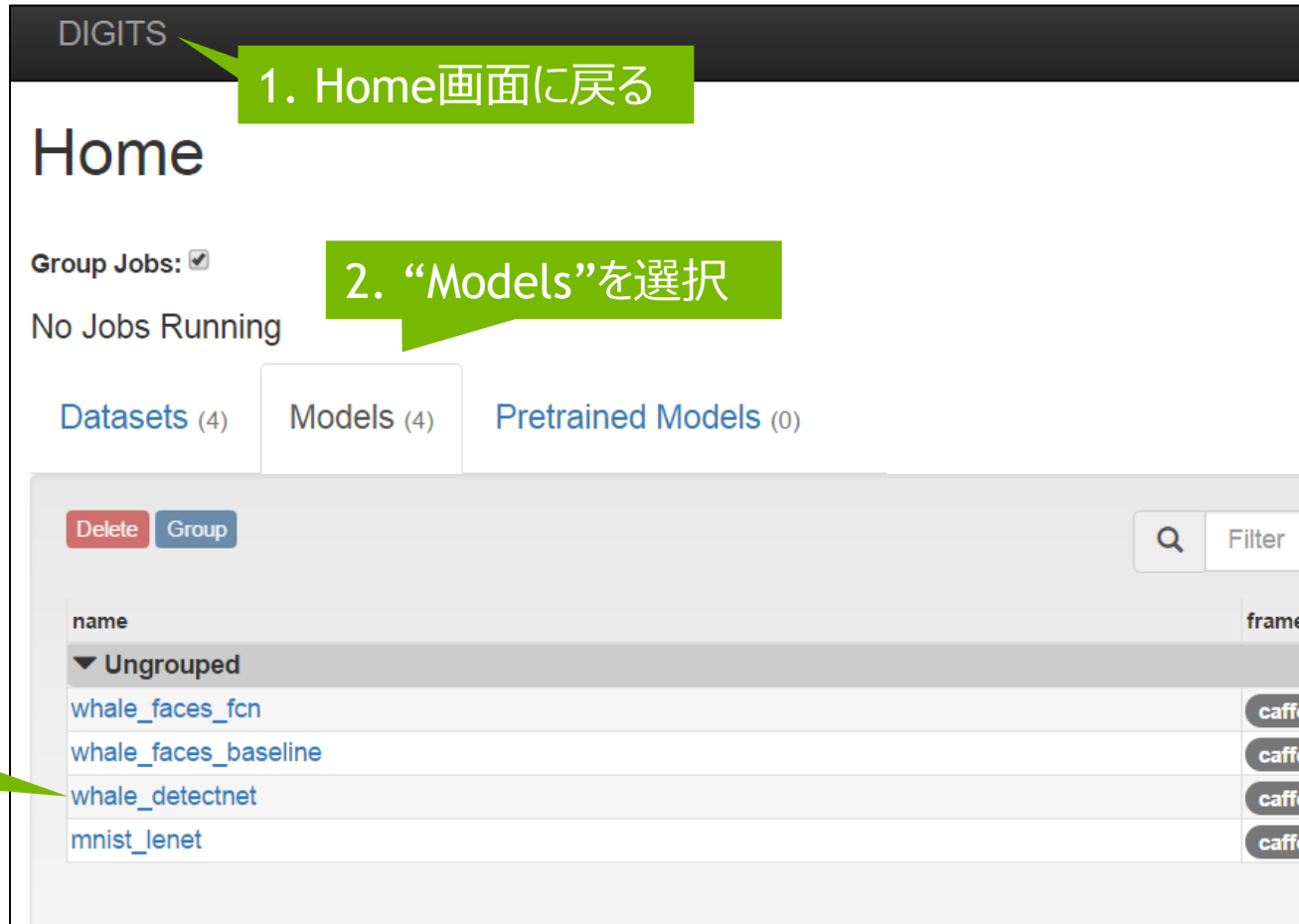
100 epoch

4. DETECTNET

推論: 100 epoch版

1. DIGITSのHome画面に戻る
2. ”Models”を選択
3. “whale_detectnet”をクリック

whale_detectnetは、100 epoch 完了時の学習結果



4. DETECTNET

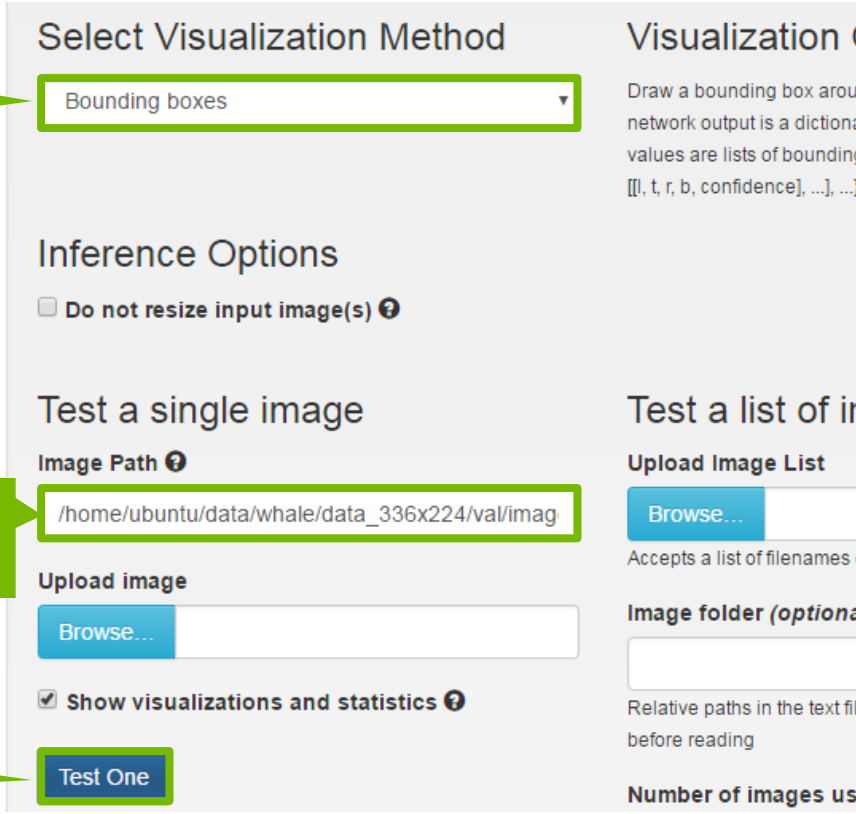
推論: 100 epoch版

1. DIGITSのテスト機能を使い、学習済みのモデルをテストする

1.Bounding boxesを選択

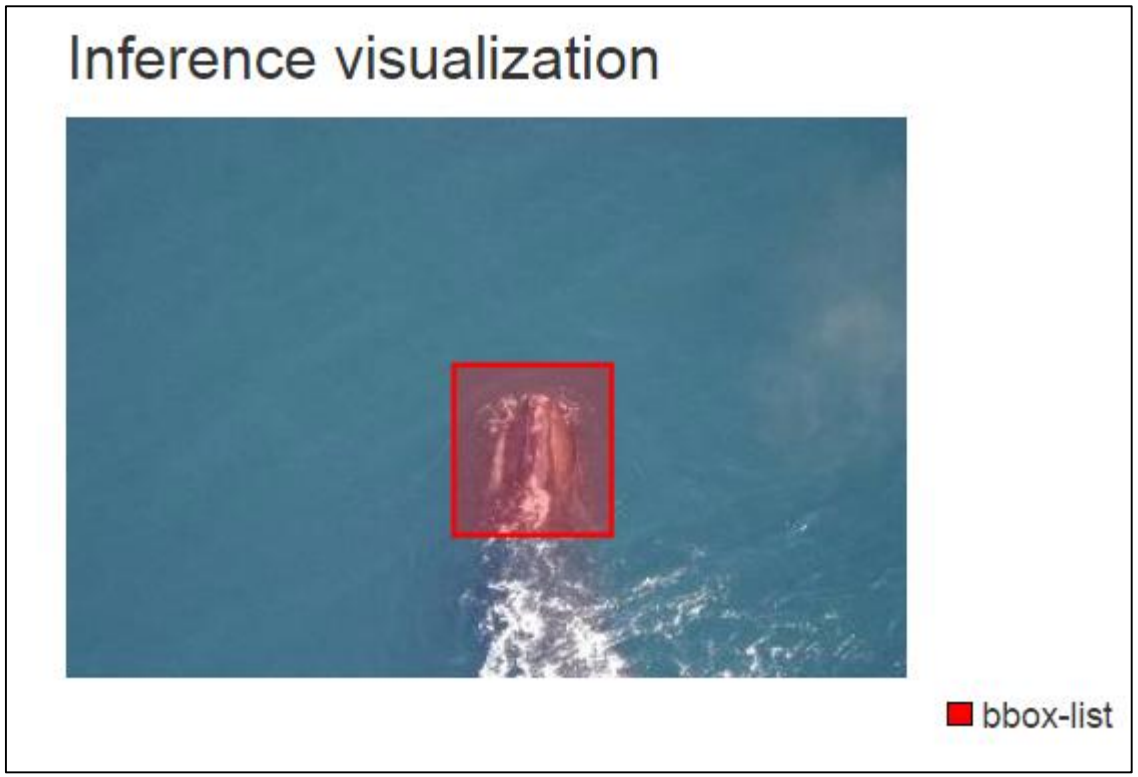
2.Image Path
“/home/ubuntu/data/whale/data_336x224/val/images/000000118.png”

3. “Test One”ボタンをクリック



4. DETECTNET

まとめ



ほとんどのクジラの顔を
正確に検出

誤検出率が非常に低い

336x224ピクセル画像の
平均処理時間 “22ミリ秒”

THANK YOU!