

使う「ディープラーニング」

技術系職員研修委員会 一般研修

1回目2019.3月 2回目2019.8月

実験教育支援センター 土屋明仁

今回の研修の目的

自分が使っているP Cでディープラーニングを“動かす”。

「データを与えて欲しい結果を得る」ための情報ツールとして、とにかくまずはD N Nを使ってみる。

- **1日目**：ハードウェア・ソフトウェア環境構築
- **2日目**：D N Nのつかい方を学ぶ

理論的な知識については

別の機会に・・・

自分で勉強する場合のおすすめ参考図書

「Excelで作るディープラーニング:

ロジスティック回帰から画像認識まで」

(kindle版のみしかありません)

ニューラルネットワークの計算アルゴリズムを、具体的に分かりやすく解説している。エクセルを使って実装しているので特別なソフトウェアが不要で、計算の流れも追いやすい。

1 日 目

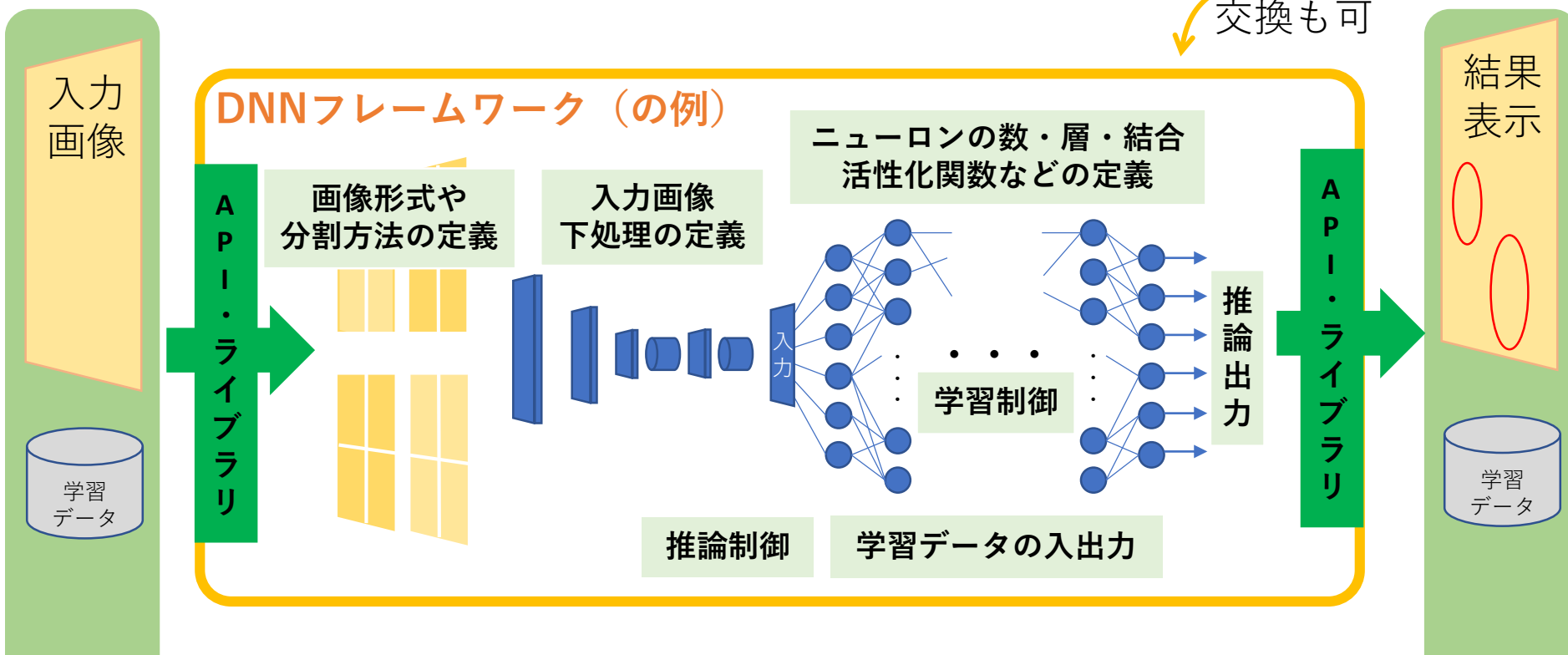
ハードウェア・ソフトウェア環境構築

1 日目 基礎知識 1 / 3

• DNNのフレームワーク (DNNモデル)

別の種類の
DNNフレームワーク

交換も可



- USBカメラから動画をキャプチャ
- フレーム画像をフレームワークに渡す
- etc...

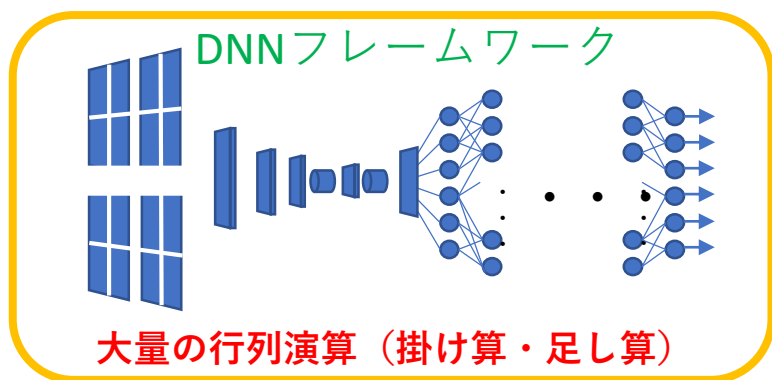
アプリケーションプログラム

- DNNによる推論結果をフレーム画像に描画
- 外部装置を制御する
- etc...

1 日目 基礎知識 2 / 3

- 人気のフレームワーク
多くの種類があり特徴もそれぞれ
 - Tensorflow . . . Google
Google開発なので利用者も多い
 - Caffe Yangqing Jia, UCB
主に画像認識向け、速い
 - PyTorch Adam Paszke et al, NYU, Facebook
Python向け、人気急上昇中
 - Darknet Joseph Redmon
個人的に気に入っている、とにかく速い
- フレームワークのWrapperもある
 - Keras、Sonnet (Tensorflow)、Gluon (MXNet) etc...

1日目 基礎知識 3 / 3



入力画像に対する下処理
畳み込み、プーリング
ニューラルネットワークの計算
推論、誤差逆伝播

⇒ GPU搭載PCだとDNNが速く計算できる

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{bmatrix}$$

・CPUで逐次計算

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 9 & 6 & 3 \\ 8 & 5 & 2 \\ 7 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 9+16+21 & 6+10+12 & 3+4+3 \\ 36+25+42 & 24+25+24 & 12+10+6 \\ 63+64+63 & 42+40+36 & 21+16+9 \end{bmatrix} = \begin{bmatrix} 46 & 28 & 10 \\ 26 & 73 & 28 \\ 190 & 118 & 46 \end{bmatrix}$$

ある赤枠の計算（タスク）が終わったら
次のタスクを実行・・・9回の計算が必要

・GPUで並列計算

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 9 & 6 & 3 \\ 8 & 5 & 2 \\ 7 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 9+16+21 & 6+10+12 & 3+4+3 \\ 36+25+42 & 24+25+24 & 12+10+6 \\ 63+64+63 & 42+40+36 & 21+16+9 \end{bmatrix} = \begin{bmatrix} 46 & 28 & 10 \\ 26 & 73 & 28 \\ 190 & 118 & 46 \end{bmatrix}$$

全体を9つのタスクに分割し、タスクを
同時に実行・・・1回の計算で済む

1 日目 使用機材

- GPUを搭載していないPCでは・・・



Intel Movidius NCS(Neural Compute Stick)

- 足し算 掛け算に特化していて100Gflops 消費電力 1 W
- PCのUSB3.0に接続 (USB2.0でも可らしい)、複数接続可能
- NCS SDK(ライブラリ)、Python3(開発言語)、Ubuntu 16.04
- パターン認識サンプルが用意されてるのでお手軽に・・・

1日目 目標

Movidius

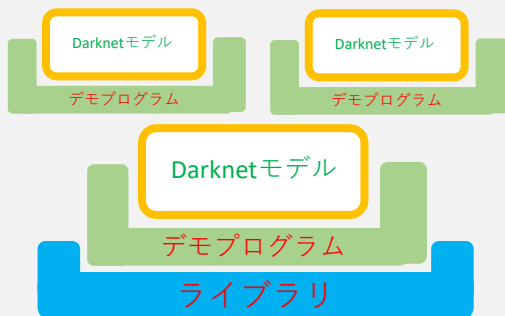
ノートPC (GPU搭載無し)

Windows10 64bit

Step1. Virtual BOX setup

Step2. Ubuntu 16.04 LTS guest OS setup

Step.4 土屋おすすめの
フレームワークをイン
ストール
Darknet-YOLO-NCS



Step2.5 Movidiusのデバイ
ス認識をホストOS(win)から
ゲストOS(ubuntu) に自動的
にわたす設定

Step.5 ラベリングツール
labellmg.py インストールと
使い方の練習

Step3. NC SDK setup
デモプログラム実行
Caffe モデルに対応



宿題

学習データ（写真）へのタグ付け

◎ニューラルネットワークに認識

させたい物体を決める（複数可）

◎その物体の写真を、物体ごとに100枚用意

◎Ubuntuにインストールした `labelImg.py` で
タグ付けする

宿題：todo 1



物体の写真を、認識させたい
物体ごとに100枚用意する

宿題：todo2

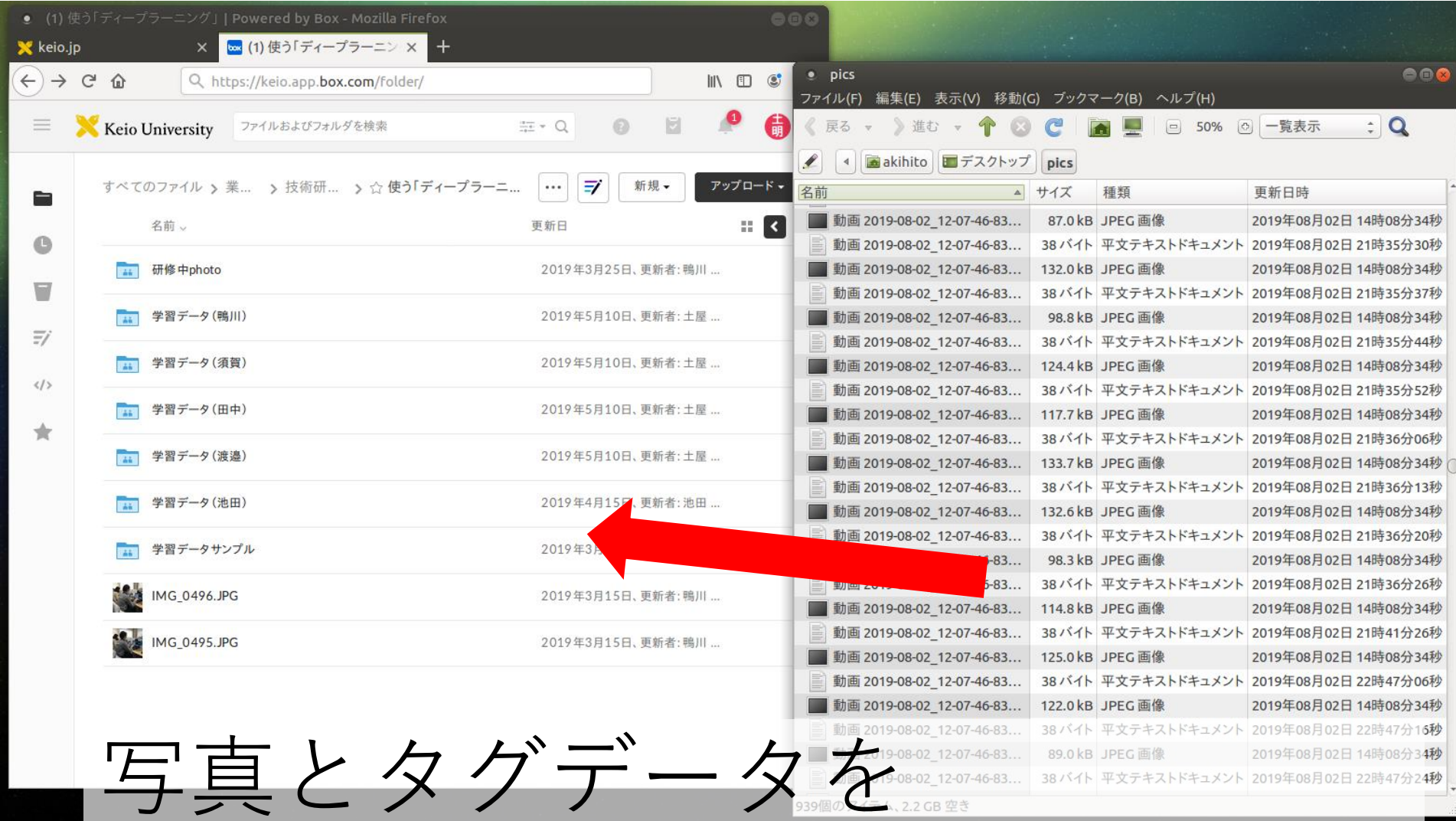


生成するタグデータのフォーマット設定を忘れずに

PASCAL VOC ではなく
YOLOにする

labelImg.py でタグ付けする

宿題：todo3



The screenshot shows a web browser window with the URL `https://keio.app.box.com/folder/`. The page displays a list of files and folders under the heading "すべてのファイル". A red arrow points from the file "IMG_0496.JPG" in the list to a secondary window titled "pics". This window shows a detailed view of the file, including its name, size, type, and update time. Below this, there is a table listing other files with similar names.

名前	サイズ	種類	更新日時
動画 2019-08-02_12-07-46-83...	87.0 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時35分30秒
動画 2019-08-02_12-07-46-83...	132.0 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時35分37秒
動画 2019-08-02_12-07-46-83...	98.8 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時35分44秒
動画 2019-08-02_12-07-46-83...	124.4 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時35分52秒
動画 2019-08-02_12-07-46-83...	117.7 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時36分06秒
動画 2019-08-02_12-07-46-83...	133.7 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時36分13秒
動画 2019-08-02_12-07-46-83...	132.6 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時36分20秒
動画 2019-08-02_12-07-46-83...	98.3 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時36分26秒
動画 2019-08-02_12-07-46-83...	114.8 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 21時41分26秒
動画 2019-08-02_12-07-46-83...	125.0 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 22時47分06秒
動画 2019-08-02_12-07-46-83...	122.0 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 22時47分15秒
動画 2019-08-02_12-07-46-83...	89.0 kB	JPEG 画像	2019年08月02日 14時08分34秒
動画 2019-08-02_12-07-46-83...	38 バイト	平文テキストドキュメント	2019年08月02日 22時47分24秒

写真とタグデータを
BOXにアップする

2 日 目

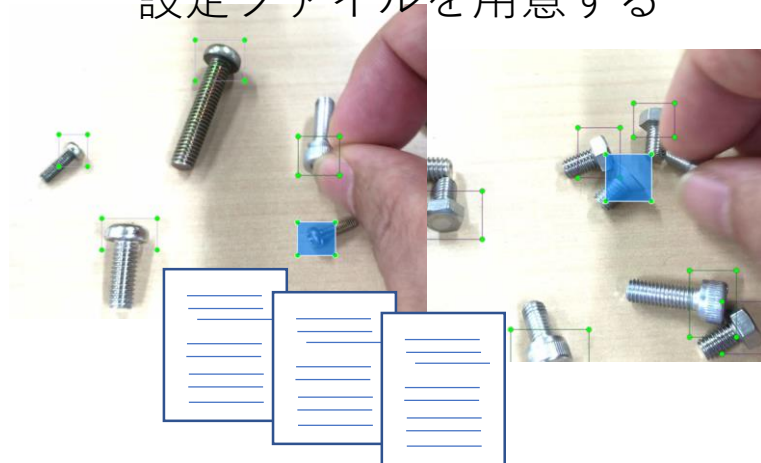
オリジナルデータの学習とデモンストレーション

2日目 目標

- ①学習データの準備
- ②各種学習の設定
- ③学習開始
- ④モデル変換（Darknet ⇔ Caffe）
- ⑤Movidiusでの物体認識

2日目 目標

1. 学習用の写真データとタグデータ、学習に必要な設定ファイルを用意する



2. darknet-yoloで学習



オリジナルの
Darknetモデル

モデル変換
プログラム

オリジナルの
Caffeモデル

3. DarknetモデルをCaffeモデルに変換

4. Movidiusで物体認識



Darknet – Yolo でトレーニング

Step1. トレーニングデータをこの図のように配置する

yoloインストールディレクトリ下のcfgディレクトリ下にoriginalディレクトリを作成、さらにその下にsampleディレクトリを作成する。

sampleディレクトリ下にbackup、datasetディレクトリを作成する。

/home/nvidia/work/darknet-yolov3/cfg/original/sample/				
名前	サイズ	更新日時	パーミッション	所有者
backup		2019/09/02 8:27:13	rw-rw-r--	nvidia
dataset		2019/08/07 21:30:46	rw-rw-r--	nvidia
classes.txt	1 KB	2019/08/05 11:09:37	rw-rw-r--	nvidia
dataset.txt	1 KB	2019/08/05 10:55:42	rw-rw-r--	nvidia
train.sh	1 KB	2019/08/05 10:56:29	rw-rw-r--	nvidia
train0.sh	1 KB	2019/08/05 10:31:17	rw-rw-r--	nvidia
yolo-obj.cfg	2 KB	2019/08/05 10:31:34	rw-rw-r--	nvidia
		2019/08/05 11:11:45	rw-rw-r--	nvidia

backup . . . yoloが学習済みのデータを保存する場所
dataset . . . 学習に使う写真とタグデータを置く場所
classes.txt . . . labelImgが生成した、認識させる物の一覧データ
dataset.txt . . . 学習用データとテスト用データの設定を書く
yolo-obj.cfg . . . yoloニューラルネットワークの設定ファイル

Darknet – Yolo でトレーニング

Step1. backupディレクトリとは

yoloが学習の過程で学習済みのデータを保存する場所

/home/nvidia/work/darknet-yolov3/cfg/original/sample/backup/				
名前	サイズ	更新日時	パーミッション	所有者
↑		2019/08/05 10:51:16	rw-rwxr-x	nvidia
yolo-obj.backup	61,664 KB	2019/08/07 21:30:46	rw-rw-r--	nvidia
yolo-obj.backup.save	61,664 KB	2019/08/08 8:09:42	rw-rw-r--	nvidia
yolo-obj_900.weights	61,664 KB	2019/08/05 12:27:15	rw-rw-r--	nvidia
yolo-obj_10000.weights	61,664 KB	2019/08/06 0:42:33	rw-rw-r--	nvidia
yolo-obj_20000.weights	61,664 KB	2019/08/06 13:56:46	rw-rw-r--	nvidia
yolo-obj_30000.weights	61,664 KB	2019/08/07 3:46:59	rw-rw-r--	nvidia
yolo-obj_40000.weights	61,664 KB	2019/08/07 21:06:09	rw-rw-r--	nvidia
yolo-obj_final.weights	61,664 KB	2019/08/08 8:09:42	rw-rw-r--	nvidia

yolo-obj_final.weights 最新の学習データ

yolo-obj.backup 一定回数学習毎に学習データがバックアップされる

...






yolo-obj_30000.weights 30000回学習済みの学習データ

yolo-obj_40000.weights 40000回学習済みの学習データ

Darknet – Yolo でトレーニング

Step1. datasetディレクトリとは

学習に使う写真とタグデータを置く場所

/home/nvidia/work/darknet-yolov3/cfg/original/sample/dataset/					
名前	サイズ	更新日時	パーミッション	所有者	
		2019/08/05 10:51:16	rw-rwxr-x	nvidia	
classes.txt	1 KB	2019/08/03 22:57:20	rw-rw-r--	nvidia	
 neji_1.jpg	336 KB	2019/08/02 5:08:26	rw-rw-r--	nvidia	
neji_1.txt	1 KB	2019/08/02 5:50:46	rw-rw-r--	nvidia	
 neji_2.jpg	161 KB	2019/08/02 5:08:34	rw-rw-r--	nvidia	
neji_2.txt	1 KB	2019/08/02 5:50:54	rw-rw-r--	nvidia	
 neji_3.jpg	96 KB	2019/08/02 5:08:34	rw-rw-r--	nvidia	
neji_3.txt	1 KB	2019/08/02 5:50:58	rw-rw-r--	nvidia	
 neji_4.jpg	255 KB	2019/08/02 5:08:34	rw-rw-r--	nvidia	
neji_4.txt	1 KB	2019/08/02 5:51:02	rw-rw-r--	nvidia	

学習用に使う写真(.jpg)と、labellmgで作ったyolo向けタグデータ(.txt)を保存する。classes.txtはlabellmgが自動生成するが、ファイルには認識させる物体の一覧が書かれている。classes.txtは一つ上の階層のディレクトリ「sample」にコピーする。

Darknet – Yolo でトレーニング

Step1.classes.txt とは

```
6  
S  
N  
B
```

labellmgでラベリングしたときに設定した物体の名称がリストアップされている。
labellmgのタグデータ保存ディレクトリに自動的に作成された **classes.txt**をこの場所に移動、あるいはコピーする

学習用に使う写真(.jpg)と、labellmgで作ったyolo向けタグデータ(.txt)を保存する。**classes.txt**はlabellmgが自動生成するが、ファイルには認識させる物体の一覧が書かれている。**classes.txt**は一つ上の階層のディレクトリ「sample」にコピーする。

Darknet – Yolo でトレーニング

Step1.dataset.txt とは

```
classes= 4
train = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/dataset/train.list
valid = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/dataset/test.list
names = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/classes.txt
backup = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/backup
```

学習に必要な各種データの配置を設定するファイル
ソフトによって自動生成されたファイルを使っているが、きほんてき
にはすべて自分で手作業で作成することができる。

識別させる物体の数

```
classes= 4
```

学習に使う写真の一覧 (divide_filesによって生成される)

```
train = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/dataset/train.list
```

学習結果のテストに使う写真の一覧 (divide_filesによって生成される)

```
valid = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/dataset/test.list
```

認識された物体のラベルデータ (labelImgによって生成される)

```
names = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/classes.txt
```

学習時の学習データのバックアップディレクトリ

```
backup = /home/nvidia/work/darknet-yolov3/cfg/original/akihito/backup
```

Darknet – Yolo でトレーニング

Step1. yolo-obj.cfgとは

Yoloバージョン2の学習済みニューラルネットワークモデル darknet19_448.conv.23に対応したニューラルネットワーク設定ファイル。cfg に保存されているものを、オリジナル学習データのディレクトリ `cfg/original/sample` 下にコピーして使う。

123行目の classes を認識させたい物体の数（classes.txtの行数）に。

116行目の filters を $\text{num} * (\text{classes} + \text{coords} + 1)$ に。認識させたい物体の数が4の場合は $\text{filters} = 5 * (4 + 4 + 1) = 45$

```
115: ##### filters=num*(classes+coords+1)
```

```
116: filters=45
```

```
117: activation=linear
```

```
118:
```

```
119: [region]
```

```
120: anchors = 1.08,1.19, 3.42,4.41, 6.63,11.38, 9.42,5.11, 16.62,10.52
```

```
121: bias_match=1
```

```
122: #####
```

```
123: classes=1
```

```
124: coords=4
```

```
125: num=5
```

Darknet – Yolo でトレーニング

Step2. トレーニング開始 開始したら2日ほど待つ

```
% cd /home/nvidia/work/darknet-yolov3/  
% darknet detector train cfg/original/akihito/dataset.txt cfg/original/akihito/yolo-obj.cfg darknet19_448.conv.23
```

darknet19_448.conv.23 はすでにさまざまな物体の認識の学習済みデータだが、初期状態のニューラルネットワークから学習するよりもすでに学習済みのニューラルネットワークから学習したほうが学習収束がはやい。

```
while [ '1' = '1' ] do  
    /home/nvidia/work/darknet-yolov3/darknet detector train /home/nvidia/work/darknet-yo¥¥  
v3/cfg/original/akihito/dataset.txt /home/nvidia/work/darknet-yolov3/cfg/original/akihito/yolo-obj.cfg ¥¥  
/home/nvidia/work/darknet-yolov3/darknet19_448.conv.23  
Done
```

学習が途中で異常終了してしまうことがしばしばあるが、その場合に自動的にまた学習が始まるよう、無限ループで学習プロセスを実行するようなシェルスクリプト。

```
while [ '1' = '1' ] do  
    /bin/cp /home/nvidia/work/darknet-yolov3/cfg/original/akihito/backup/yolo-obj.backup /home/nvidia/work/darknet-  
yolov3/cfg/original/akihito/backup/yolo-obj.backup.save  
    /home/nvidia/work/darknet-yolov3/darknet detector train /home/nvidia/work/darknet-yo¥¥  
lov3/cfg/original/akihito/dataset.txt /home/nvidia/work/darknet-yolov3/cfg/original/akihito/yolo-obj.cfg ¥¥  
/home/nvidia/work/darknet-yolov3/cfg/original/akihito/backup/yolo-obj.backup.save  
done
```

Darknet ⇔ Caffeモデル変換

Step3. 変換準備

~/workspace/YoloV2NCS/models/convertyo.sh モデル変換スクリプトを修正

#!/bin/bash

#####

original sample

filename=tiny-yolo-voc

ここは独自学習データの状況に合わせて

#####

my training

filename=sample

yolocfg=./yolomodels/\$filename.cfg

yoloweight=./yolomodels/\$filename.weights

yolocfgcaffe=./caffemodels/\$filename.prototxt

yoloweightcaffe=./caffemodels/\$filename.caffemodel

echo \$yolocfg

echo \$yoloweight

echo "convert yolo to caffe"

python ../python/create_yolo_prototxt.py \$yolocfg \$yolocfgcaffe

#python ../python/create_yolo_caffemodel.py -m \$yolocfgcaffe -w \$yoloweight -o \$yoloweightcaffe

python3 ../python/create_yolo_caffemodel.py -m \$yolocfgcaffe -w \$yoloweight -o \$yoloweightcaffe

echo "done"

Darknet ⇔ Caffeモデル変換

Step3. 変換準備

~/workspace/YoloV2NCS/src/Region.cppを編集してビルド

```
const int N = 5;
```

```
//////////
```

```
// original sample
```

```
const float biases[N*2] = {1.08,1.19, 3.42,4.41, 6.63,11.38, 9.42,5.11, 16.62,10.52};
```

```
//const float biases[N*2] = {0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828};
```

```
//////////
```

```
// original sample
```

```
const std::string objectnames[] =
```

```
{ "aeroplane","bicycle","bird","boat","bottle","bus","car","cat","chair","cow","diningtable","dog","horse","motorbike","person","pottedplant","sheep","sofa","train","tvmonitor"};
```

```
//////////
```

```
// my training
```

```
const float biases[N*2] = {1.08,1.19, 3.42,4.41, 6.63,11.38, 9.42,5.11, 16.62,10.52};
```

```
const std::string objectnames[] = { "6", "S", "N", "B" };
```

```
Region::Region()
```

```
{  
    initialized = false;  
    . . . 以下省略
```

ここは独自学習データの状況に合わせて

編集後に ~/workspace/YoloV2NCS にて make を忘れず実行

Darknet ⇔ Caffeモデル変換

Step3. 変換準備

~/workspace/YoloV2NCS/detectionExample/ObjectWrapper.pyを編集

... 省略

ObjectWrapper.fifoOutHandle.append(fifoOut)

self.dim = (416,416)

self.blockwd = 12

self.wh = self.blockwd*self.blockwd

self.targetBlockwd = 13

self.classes = 4

self.threshold = 0.2

self.nms = 0.4

ここは独自学習データの状況に合わせて

def __del__(self):

for i in range(ObjectWrapper.devNum):

ObjectWrapper.fifoInHandle[i].destroy()

ObjectWrapper.fifoOutHandle[i].destroy()

ObjectWrapper.graphHandle[i].destroy()

... 省略

Darknet ⇔ Caffeモデル変換

Step3. 変換手順

Darknetモデルデータ(ドアノブ識別をサンプルに)

- ① **sample.weights**
- ② **yolo-obj.cfg**

~/workspace/YoloV2NCS/models/yolomodels に①と②を置く。

~/workspace/YoloV2NCS/modelsにて sh convertyo.sh を実行。

~/workspace/YoloV2NCS/models/caffemodels にCaffeに変換された①②ができる。

~/workspace/YoloV2NCS/modelsにて sh makegraph.sh sample を実行。

~/workspace/YoloV2NCS/modelsに graph ができる。

~/workspace/YoloV2NCSにて、

```
$ mv models/graph ./graph.sample
```

```
$ rm graph
```

```
$ ln -s graph.sample graph
```

Movidiusで 物体認識デモ

```
$ python3 detectionExample/Main.py -video /dev/video0
```

付録

- Divide_filesの使い方

インストール

```
$ git clone https://github.com/demulab/divide_files.git
```

```
$ cd divide_files/
```

```
$ gcc -o divide_files divide_files.c
```

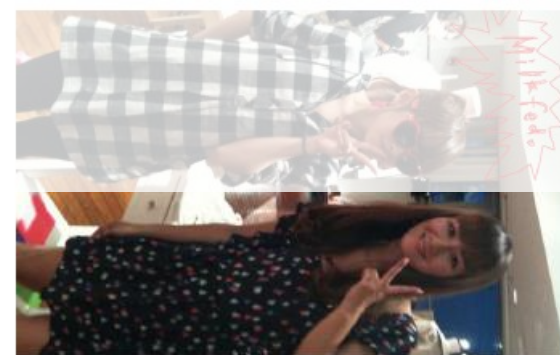
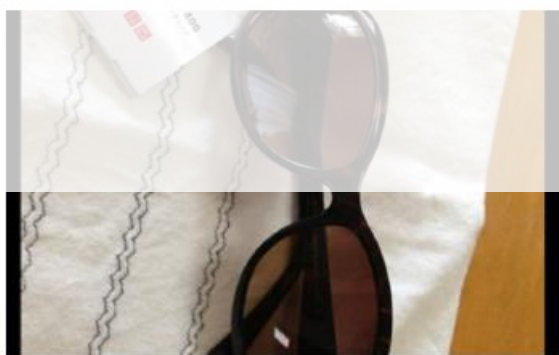
写真データ(とタグデータ)を置いたディレクトリで
divide_filesを実行。

トレーニング用データに**8割**の写真を、テスト用データ
に**2割**のデータをランダムに分けてくれる。それぞれ **test.list**
ファイル、**train.list**ファイル。

池田さん 自転車 (tire、frame)



須賀さん メガネ (glass)



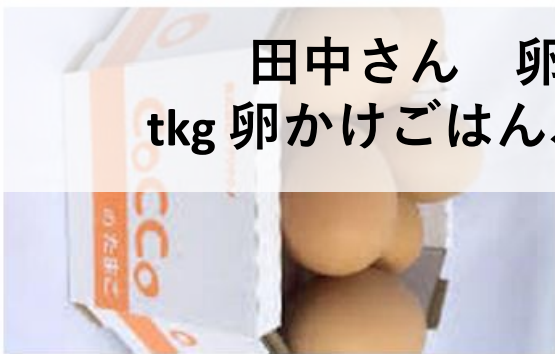
須賀さん メガネ (glass) 認識実行例



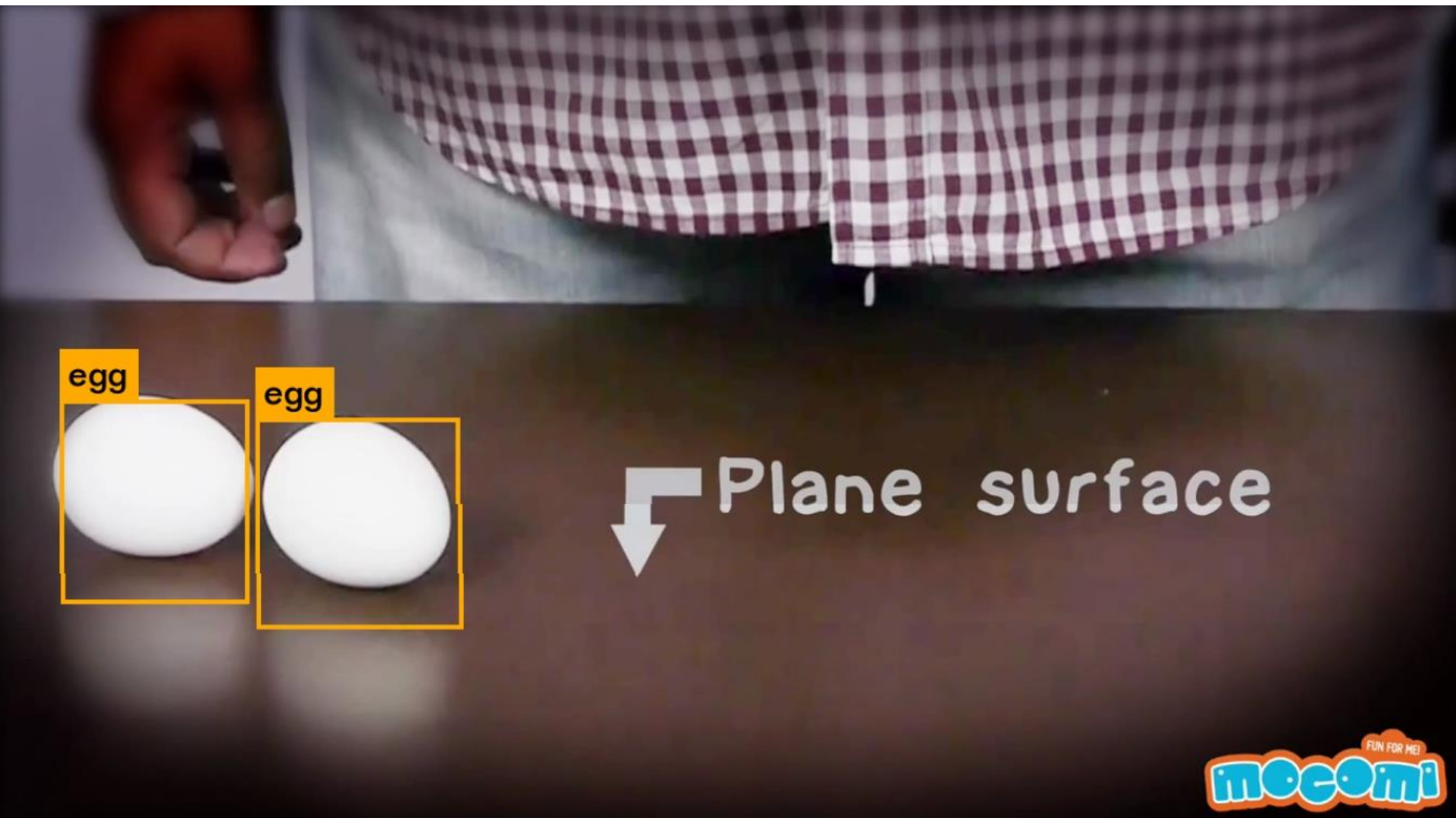
***EVEN WHEN YOU'RE WEARING THEM**



田中さん 卵 (boiled egg ゆで卵、raw egg 生卵、egg 卵、
tkg 卵かけごはん、cooked egg 卵料理、drawed egg 卵のイラスト)



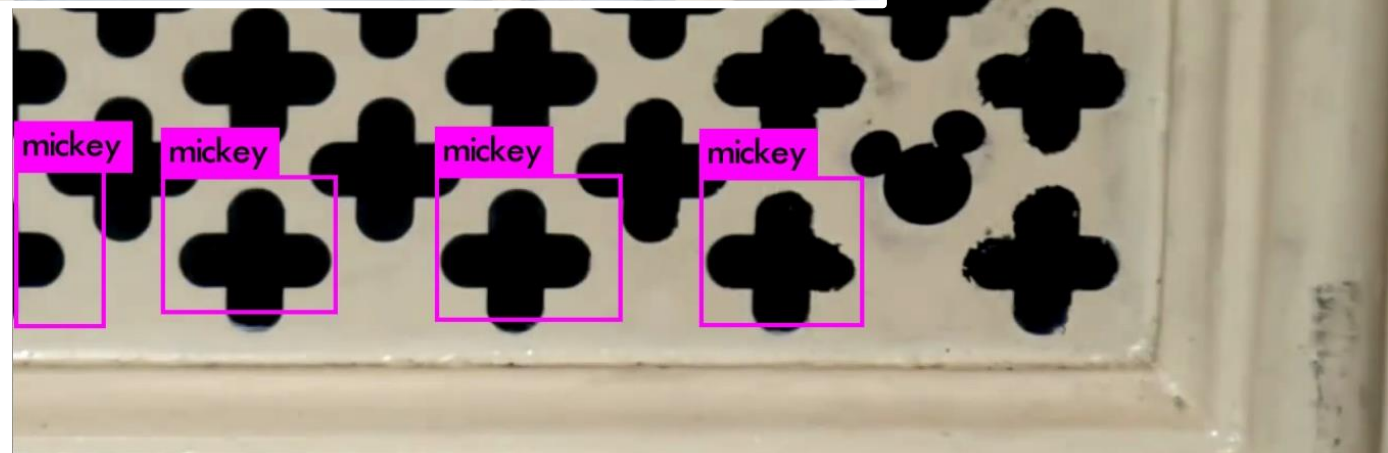
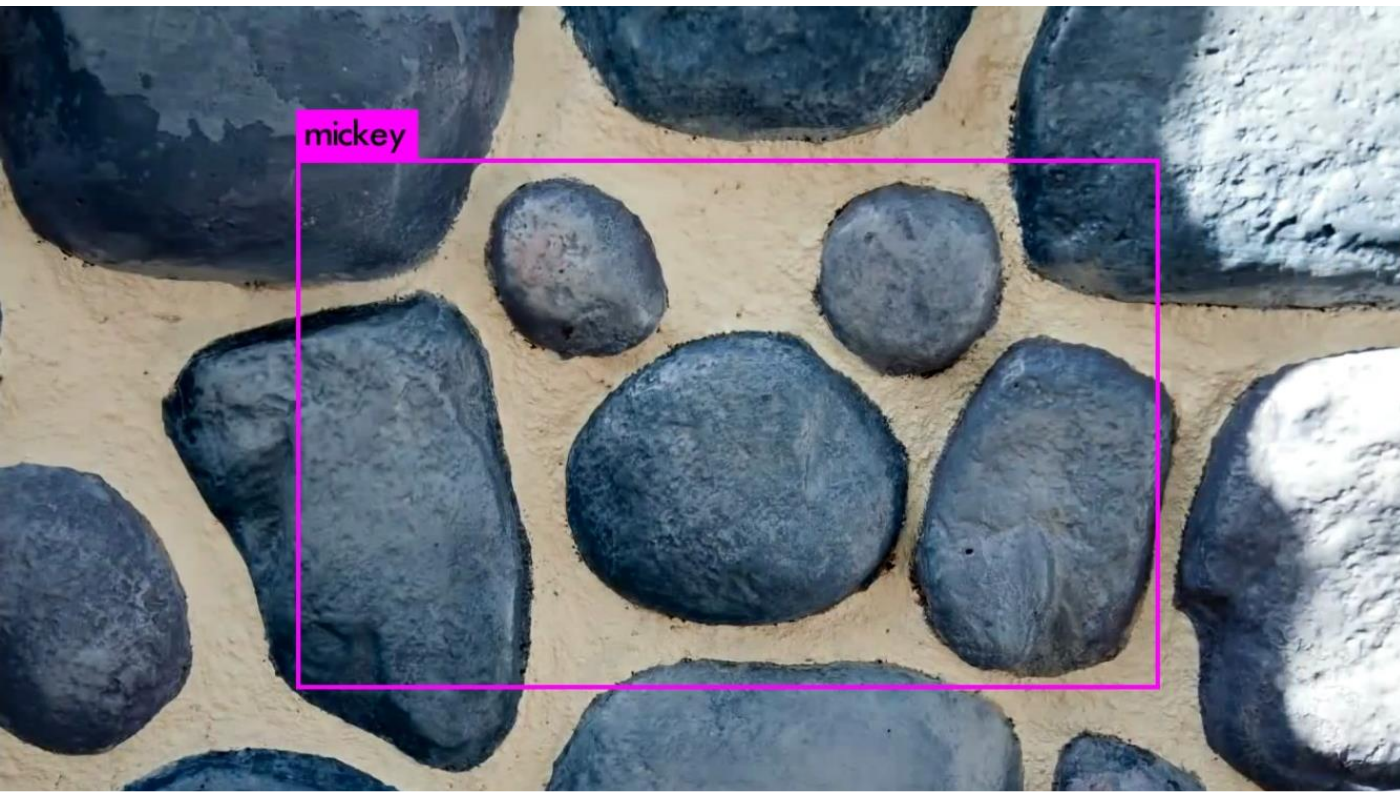
田中さん 卵 認識実行例



鴨川さん 隠れミッキー (mickey)



鴨川さん 隠れミッキー (mickey)

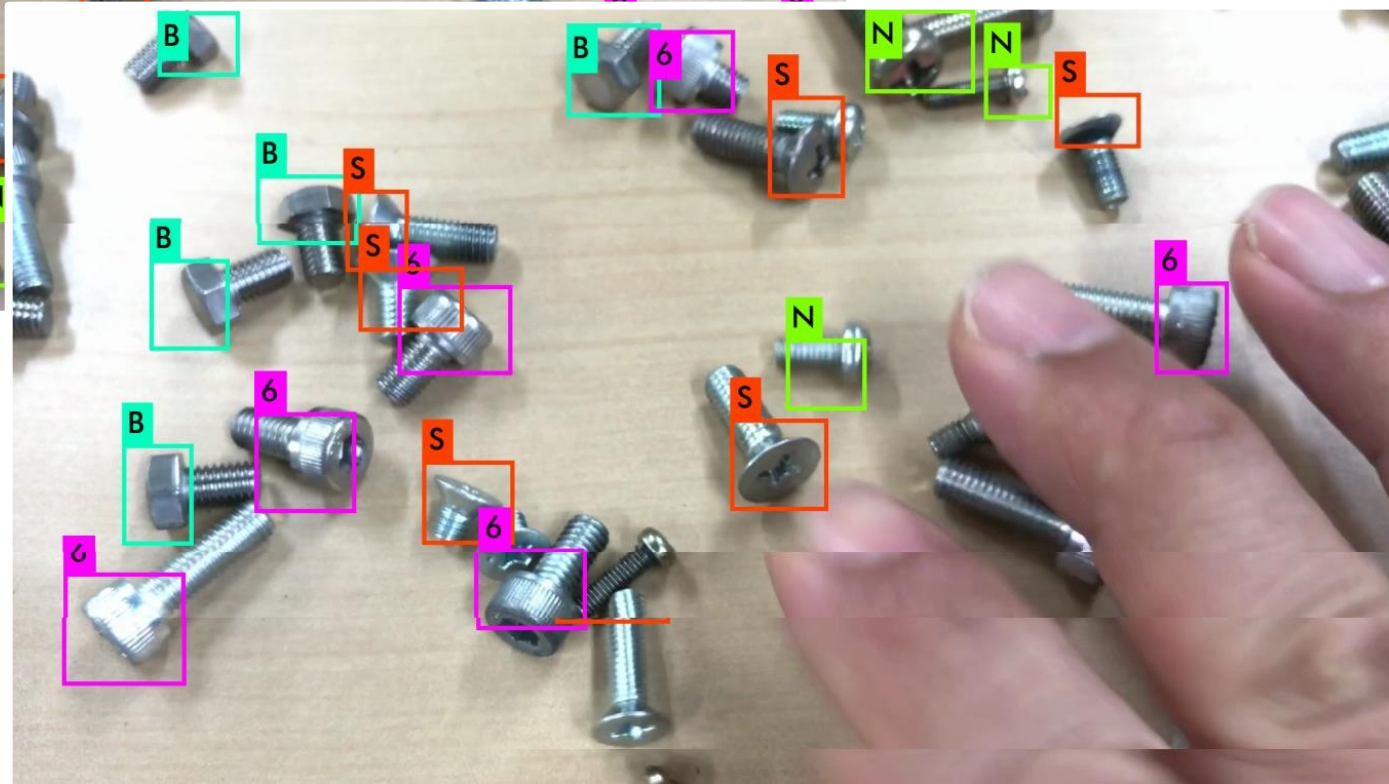
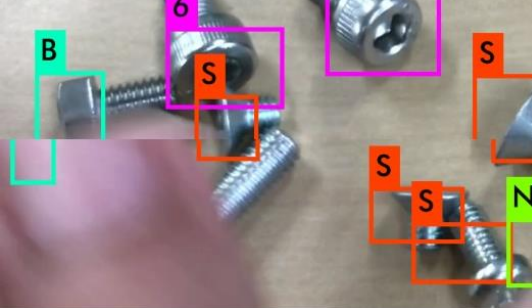
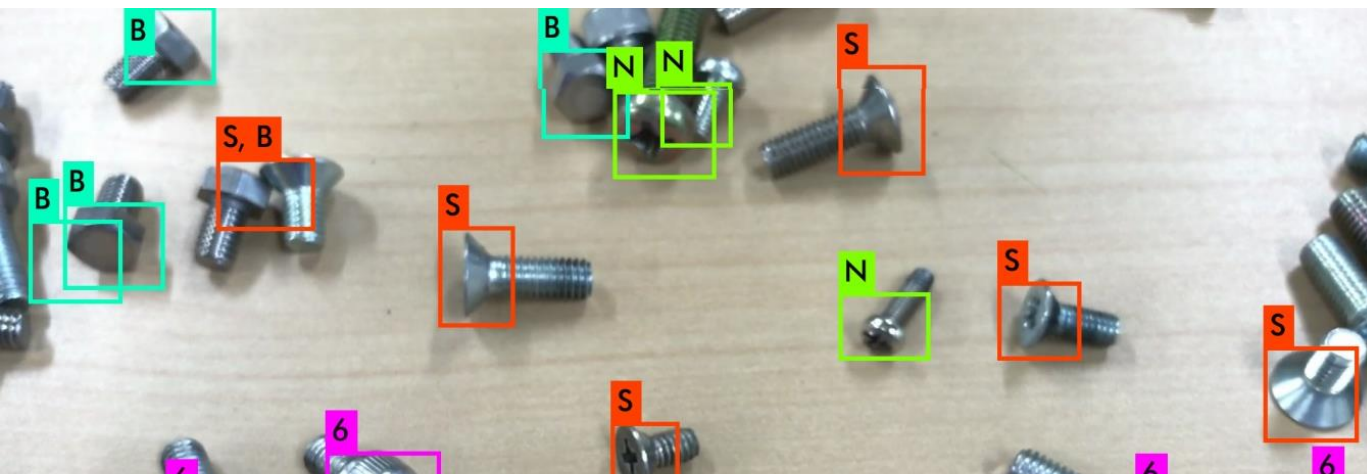


土屋（渡邊さん）

6:六角穴付きボルト, B:ボルト, S:皿ネジ, N:鍋ネジ



土屋（渡邊さん） 6:六角穴付きボルト, B:ボルト, S:皿ネジ, N:鍋ネジ



参加者の感想

本研修に参加した動機は、“使う”ディープラーニングというタイトルの“使う”の部分に惹かれたからである。私が担当している学部2年生の実験でも先生方からAI、ディープラーニングについての実験ができないかという

話が時々挙がっているため、本研修に参加することで、何かそのヒントをつかむことができないかと考えたからである。

研修1回目はWindows10PC上でUbuntuが動く環境構築を行い、ディープラーニングのフレームワークをインストールした。そして、ラベリングツールを使い自分の好きな写真にタグ付けをすることが宿題となった。私は自転車の写真にタグ付けすることにした。この作業は単純ではあるが200近くも行ったので、とても骨が折れた。最終的には残念ながら私がタグ付けした自転車はうまく認識されなかったが、他の参加者の、ねじや扉のノブが認識された動画を見せてもらった。

今回経験させてもらったディープラーニングの実践を元に新しい実験を提案してゆきたい。

参加者の感想

今回ネット上から様々な「隠れミッキー」を学習データとして集めたが、画像サイズや解像度の違いから、座標計算に悪影響を与えてしまい、うまく認識できないデータが出てしまったと思う。

機械学習において大変なことは、

「学習データをたくさん集めること」

「良い学習データを用意すること」

だということが分かった。せっかく膨大な学習データを用意しても、それが”良い問題”でない限り、間違った答えを学習させてしまうことになり、無駄な労力を割くことになってしまうので良い問題を集める事が重要だなと感じた。

参加者の感想

総括的には、思ったより容易に動画の画像認識が出来ることに驚きました。画像が動いている中で、タグ付けされたものが判別されていくのを見て、大変興奮を覚えました。

但し、実行環境を整えるのが大変面倒。**OS**の設定から、他のライブラリなどのインストールなどが大変でした。個人では、この構築で挫折してしまいそうな感じです。

機械学習の作業自体の流れも、何となくですが理解も出来ました。こういったものを使用して、なにか出来ないかを、今後考えて行きたいと思います。大変勉強になりました。ありがとうございました。