情報処理（15週目）
Pythonを用いた深層学習

王 忠奎（wangzk@fc.ritsumei.ac.jp）
立命館大学　ロボティクス学科
2024.07.20

講義の流れ

- 深層学習とは
- Kerasを用いた手書き数字の認識
- YOLOを用いた物体認識

14週目レポート

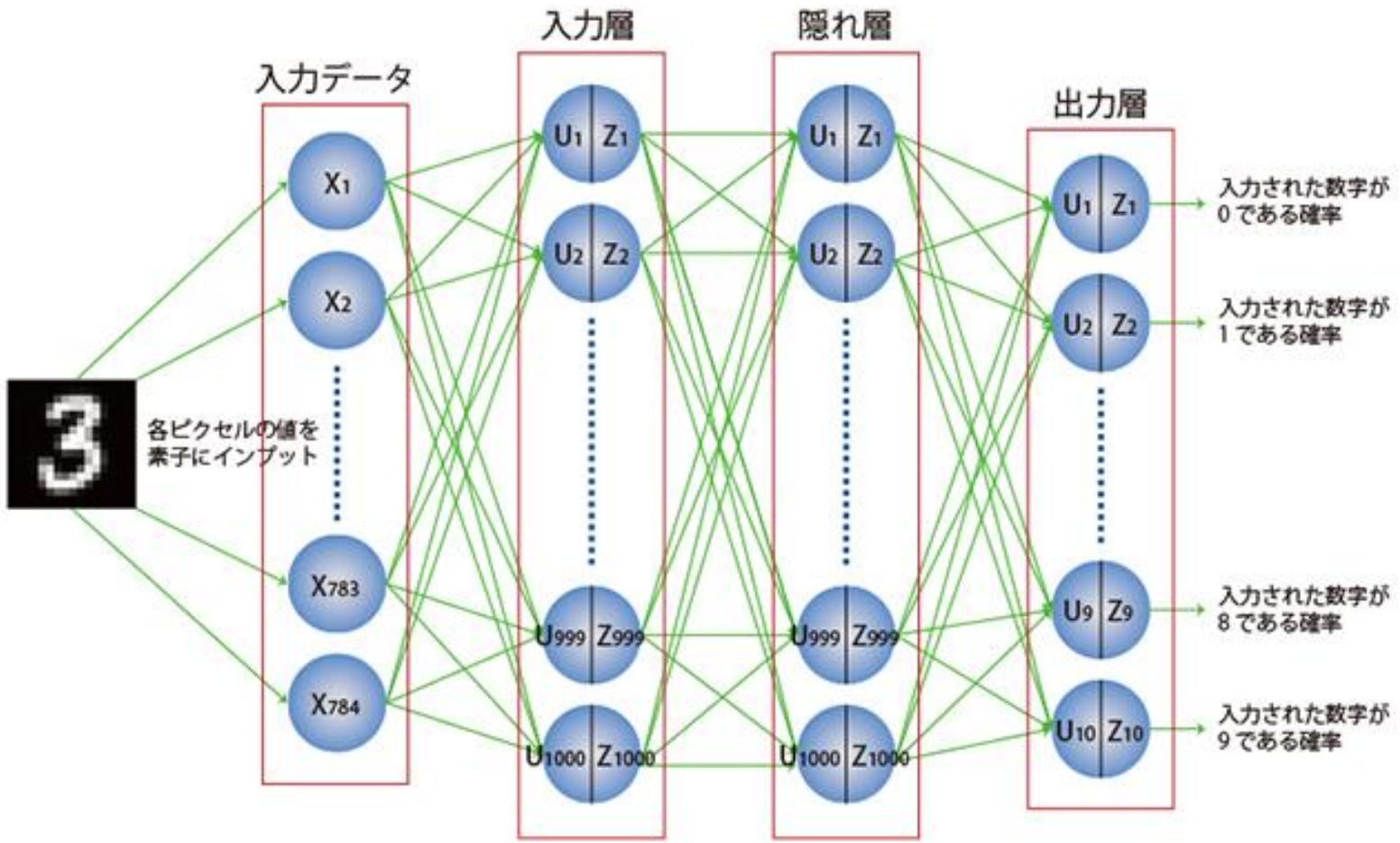
- 写真の1枚を用意して、適切なサイズに変更し、輪郭を抽出してみよう。また、抽出した輪郭を楕円でフィッティングし、表示してみよう。楕円での輪郭フィッティングと描画は下記のメソッドを利用してください。

```
ellipse = cv.fitEllipse(i)  # 楕円でフィッティング、iは輪郭
img_ellipse = cv.ellipse(img, ellipse, (0, 0, 255), 2)  # 画像imgに楕円の描画
```

- for文を利用してすべての輪郭をフィッティングし、描画してみよう。
- 輪郭の点数が 5 点より少ない場合、エラーが出る可能性がありますので、if文でエラーを避けてみよう。
- 写真とプログラムをmanaba+Rで提出してください

深層学習

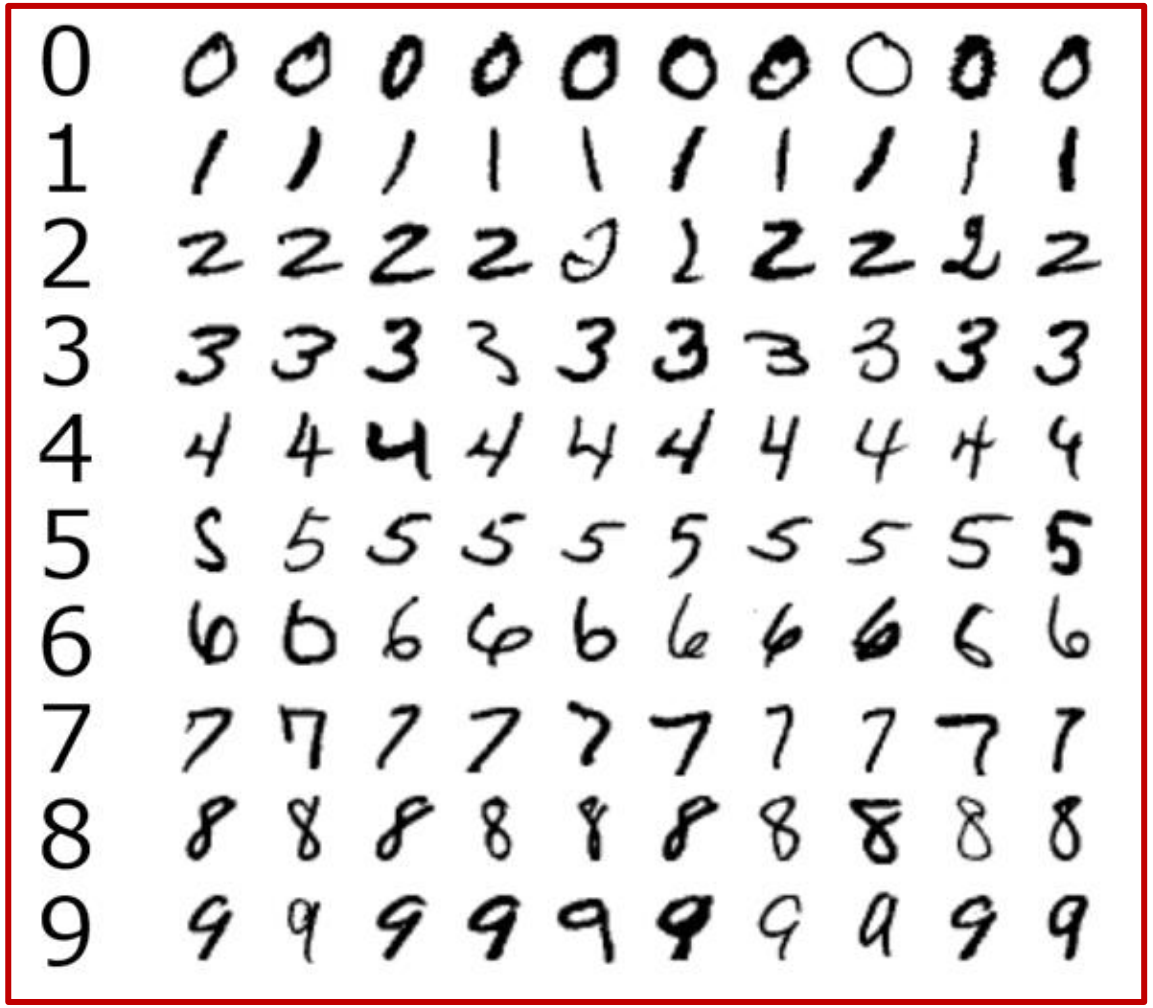
- 機械学習の一種
- 多層構造のニューラルネットワーク
- 特徴の定義や抽出が不要
- 膨大な量の学習データが必要



出典：https://www.hpc.co.jp/library/commentary/aboutdeeplearning/

MNISTデータベース

- The MNIST database（Modified National Institute of Standards and Technology database）は、「0」～「9」の手書き数字の画像データセットである
- 6万枚の訓練データ（画像とラベル）
- 1万枚のテストデータ（画像とラベル）
- 8bitグレースケール（＝色がないモノクローム）： 白「0」～黒「255」の256段階
- 幅28×高さ28（＝784ピクセル）



データ例

機械学習ライブラリ

TensorFlowは、Googleが開発し無料で使える機械学習用のライブラリである。

Kerasは、Pythonの深層学習（ディープラーニング）のライブラリである。

データの入力、正規化、確認

```
import tensorflow as tf  # テンソルフローの導入
import numpy as np
import matplotlib.pyplot as plt
import time  # 実行時間を記録するためのパッケージの導入

mnist = tf.keras.datasets.mnist  # データセットの指定
(x_train, y_train), (x_test, y_test) = mnist.load_data()  # データの読み込み

x_train = tf.keras.utils.normalize(x_train, axis=1) # 訓練データの正規化
x_test = tf.keras.utils.normalize(x_test, axis=1)  # テストデータの正規化

plt.imshow(x_train[0])  # 訓練データの確認
plt.show()  # データの表示
```

深層学習モデルの構築

```
# 深層学習モデルの構築
model = tf.keras.models.Sequential()  # モデルタイプの指定

model.add(tf.keras.layers.Flatten())  # 1次元にデータをフラット化する層

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))  # 中間層の追加

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))  # 中間層の追加

model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))  # 出力層の追加

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])  # 訓練プロセスの作成
```

学習

```
start_time = time.time()  # 訓練開始時刻の記録
# モデルの訓練
history =
model.fit(x_train,y_train,epochs=10,batch_size=1000,verbose=1,validati
on_data=(x_test,y_test))

score = model.evaluate(x_test,y_test,verbose=0)  # モデルのテスト

print('TEST loss:', score[0])  # テストに関する損失関数の結果
print('TEST accuracy:', score[1])  # テストに関する精度
print(f'Computation time: {time.time()-start_time} seconds.')  # 訓練時間
の表示
```

損失関数値の表示

```
plt.figure(1, figsize=(10,4))
plt.subplots_adjust(wspace=0.5)


plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='training', color='black')
plt.plot(history.history['val_loss'], label='test', color='blue')
plt.ylim(0, 1.5)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
```

正答率の表示

```
# 正答率の経過表示
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='training', color='black')
plt.plot(history.history['val_accuracy'], label='test', color='blue')
plt.ylim(0.5, 1.0)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```

テスト例

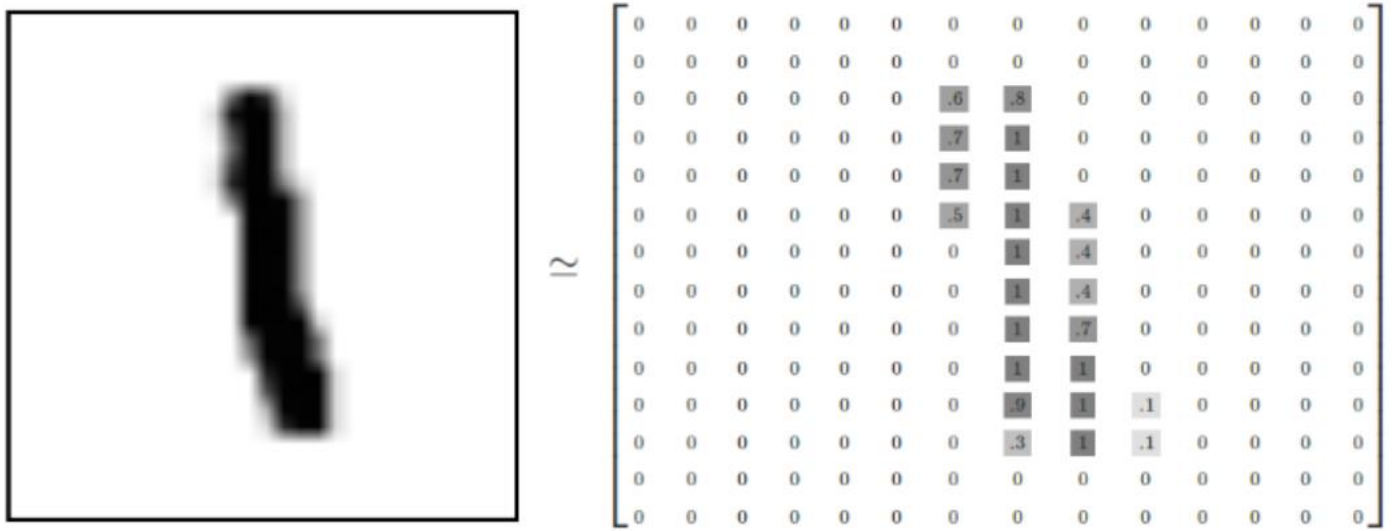
```
predictions = model.predict([x_test])  # 訓練したモデルを用いた予測
print(np.argmax(predictions[0]))  # 予測結果

plt.imshow(x_test[0])
plt.show()
```

YOLOを用いた物体認識

- YOLOの紹介
- 環境の設定（参考資料）
- モデルのダウンロード
- coco datasetを用いた物体認識デモ
- 自分のdatasetを用いた学習とテスト

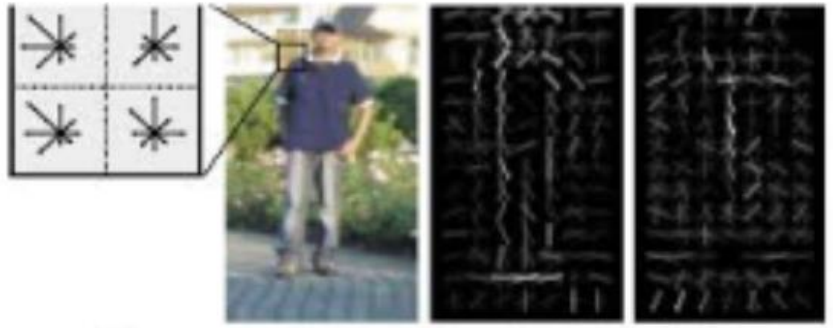
物体認識の歴史

物体認識は商用の数字化した写真が誕生(1990 Dycam Model 1)したのとほぼ当時に物体検出の研究を始めました。



コンピュータグラフィックス学に基づいて手動で設計した特徴(明暗差、角度、向き)で機械学習の分類の問題として解決方法

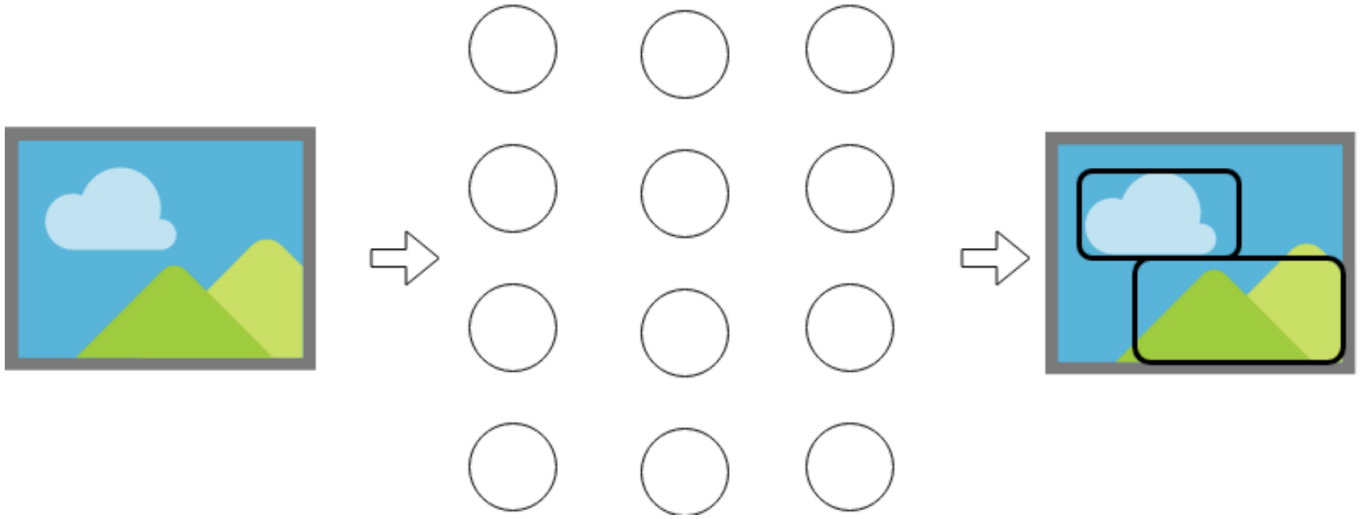
Haar-Like特徴



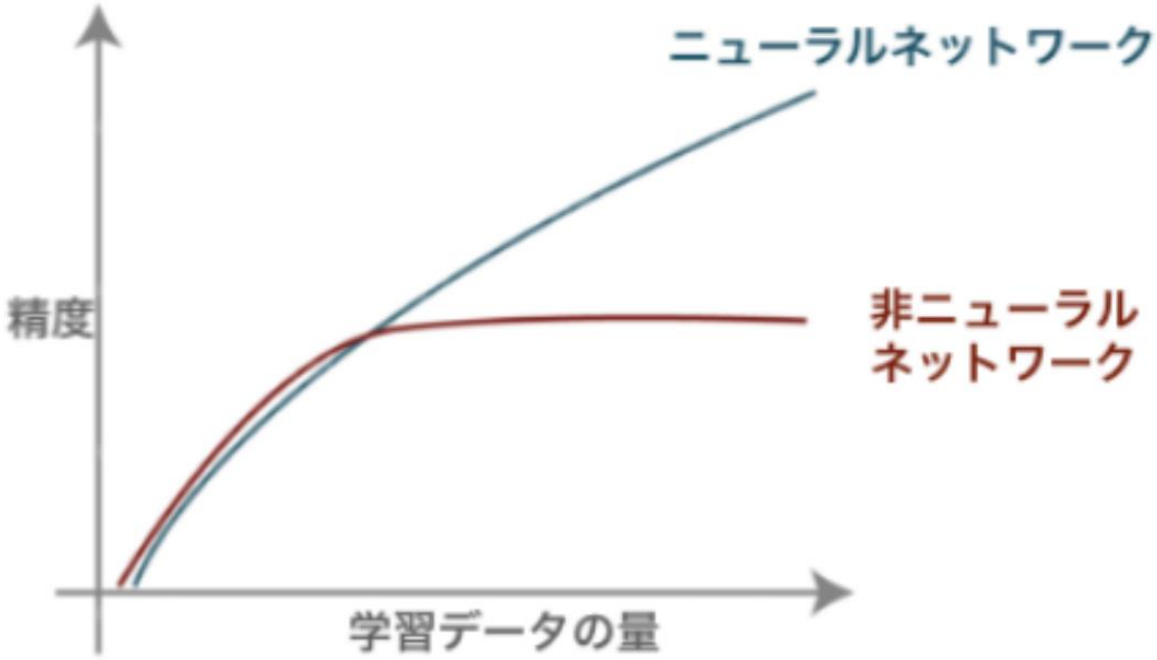
HOG特徴

物体認識手法

入力画像のPIXELごとのRGB値をニューラルネットワークの各ニューラルのInputとして解決方法

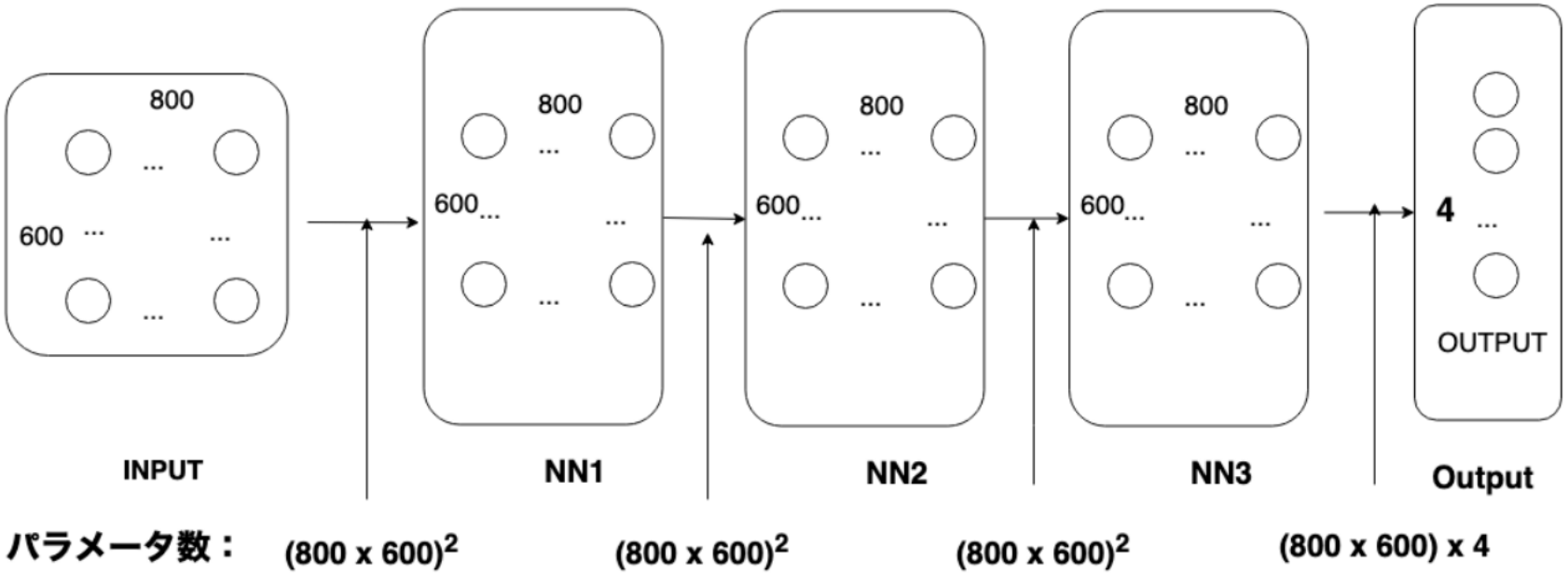


ニューラルネットワークと非ニューラルネットワークの精度の比較



深層学習

五層の全結合ニューラルネットワークで800*600の写真を認識するモデルを学習する場合、パラメーター数は92 億ほどある



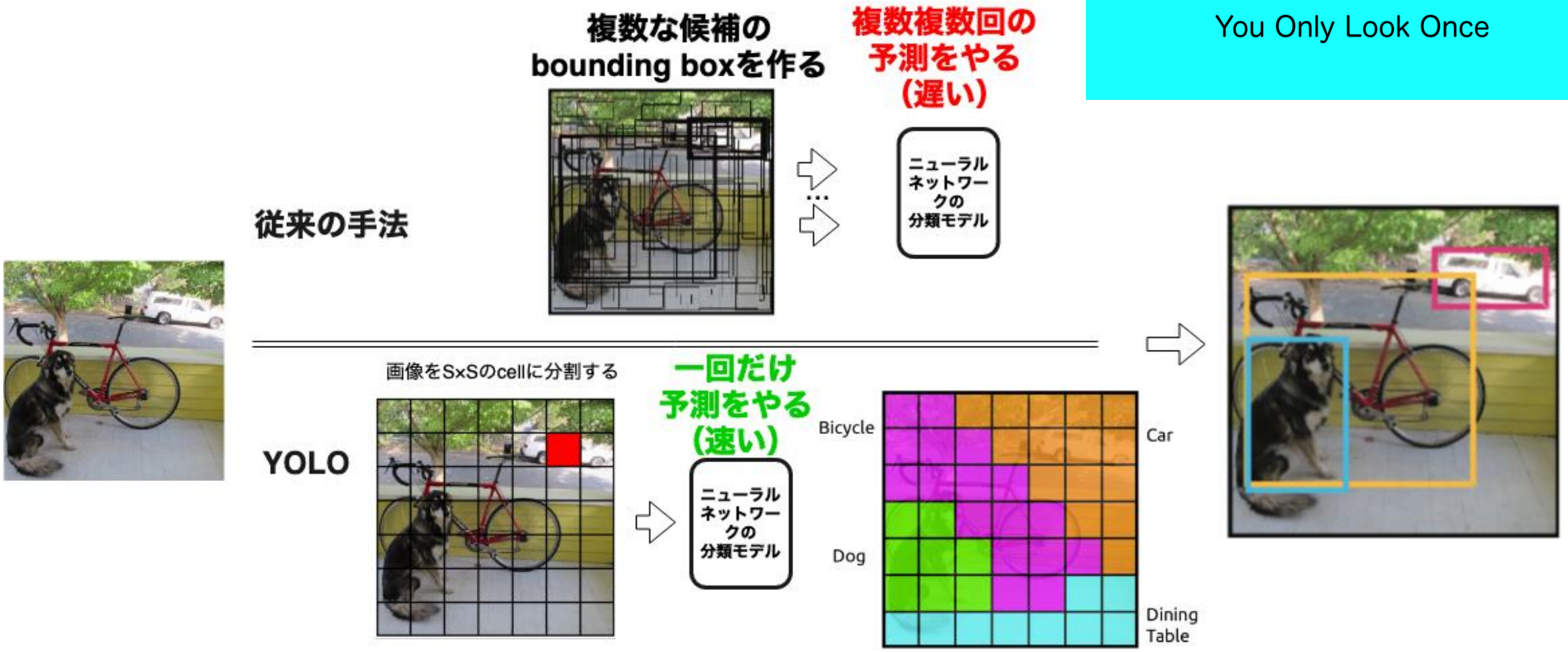
2000年前後の10年間に物体検出を手書き数字の認識、指紋認証など少数なケースに商用化できなかったです。その原因は当時の手法で一般的な物体を検出する精度は人間より大きな格差があったことです。

2010年から毎年、大規模な画像データセット(画像は1400万枚、2万種類)を使った画像認識コンペティション(ILSVRC)が開催されている。2012年ににDeep Learningのモデルが初めて優勝した。2014年にやっとDeep Learningのモデルの精度が人間に勝ちました。

YOLOとは

YOLO

You Only Look Once



出典：https://techblog.gmo-ap.jp/2019/07/30/yolo-summury/

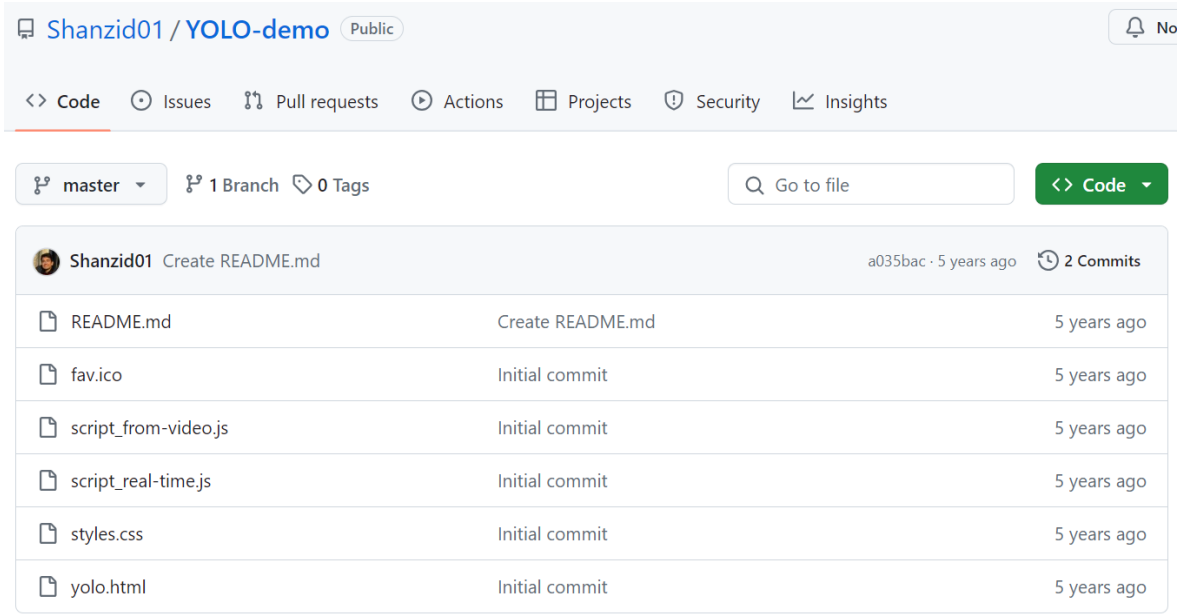
YOLOによる物体認識デモ1

A YOLO object detection demo on browser.

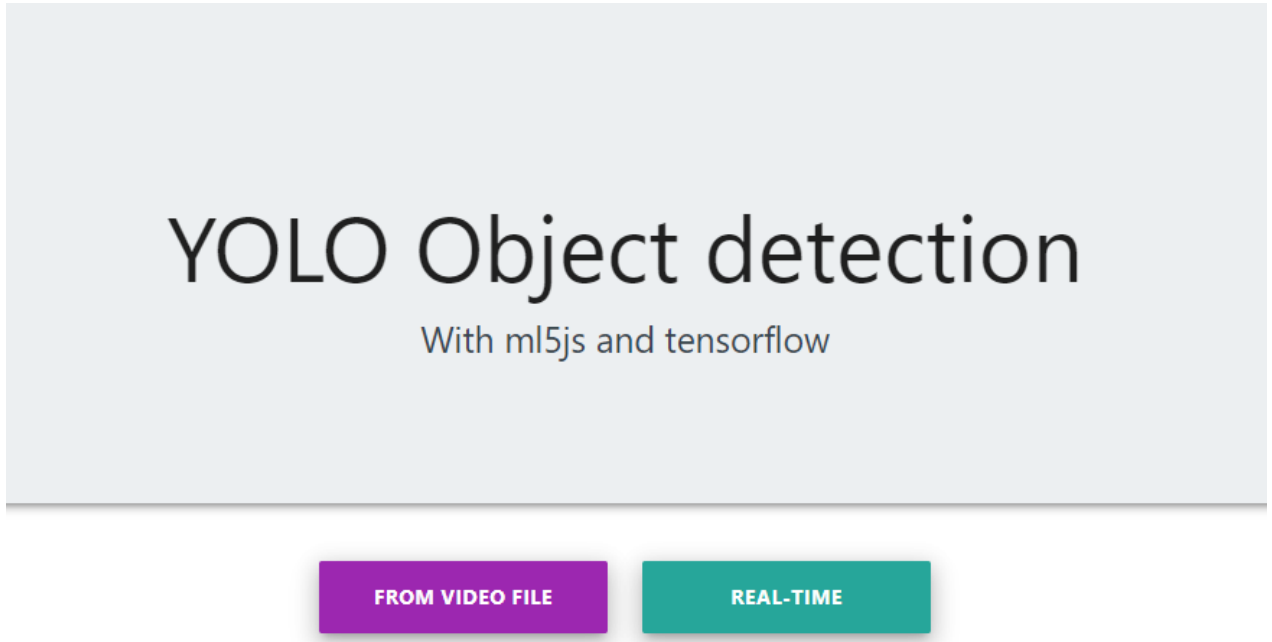
Uses Tensorflow.js and ml5js

Proof of concept. All processing occurs locally.

- Can detect objects in real-time with webcam
- Can detect objects in frames extracted from a selected video file



Githubからファイルのダウンロード → ウェブページで開く



https://github.com/Shanzid01/YOLO-demo

YOLOによる物体認識デモ2

Google colab環境を用いたYOLOの実装

環境構築

```
from google.colab import drive
drive.mount('/content/drive')

%cd ./drive/MyDrive
!git clone https://github.com/Megvii-BaseDetection/YOLOX
```

必要ライブラリ
のインポート

```
%cd YOLOX
!pip install -U pip && pip install -r requirements.txt
!pip install -v -e .
```

```
!pip install cython
!pip install 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

https://qiita.com/hkwsdgea_ttt2/items/970f34e1aa59059f7c69

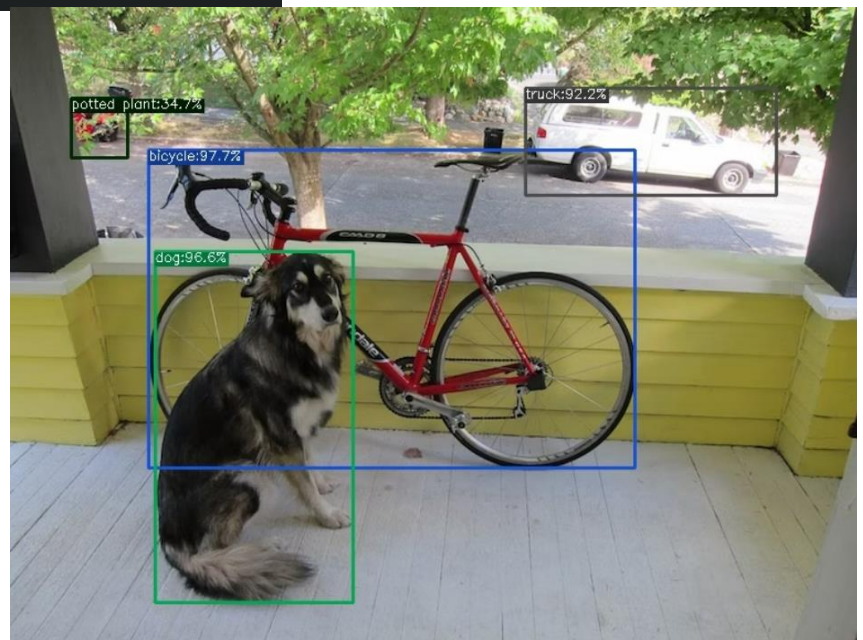
YOLOによる物体認識デモ2（つづき）

習済みモデルの取得

```
!wget https://github.com/Megvii-BaseDetection/YOLOX/releases/download/0.1.1rc0/yolox_x.pth
```

物体認識

```
!python tools/demo.py image -n yolox-x -c yolox_x.pth --path assets/dog.jpg
--conf 0.25 --nms 0.45 --tsize 640 --save_result --device gpu
```



https://qiita.com/hkwsdgea_ttt2/items/970f34e1aa59059f7c69

自分のパソコンでのYOLO実装

手順：

環境の整備 → YOLOモデルのダウンロード →

写真の準備 → アノテーション作業 → トレーニングと検証 → 予測テスト

一定の枚数が必要

- ✓ 手作業がほとんど
- ✓ Label
- ✓ BoundingBox
- ✓ Segmentation（必要に応じて）

- ✓ Datasetの操作（水増しなど）
- ✓ トレーニング用と検証用データ分け
- ✓ 高性能PCの利用（GPUで高速化）

- ✓ 学習済モデル利用
- ✓ 任意の写真
- ✓ 高性能PCは不要

環境整備

YOLOv4

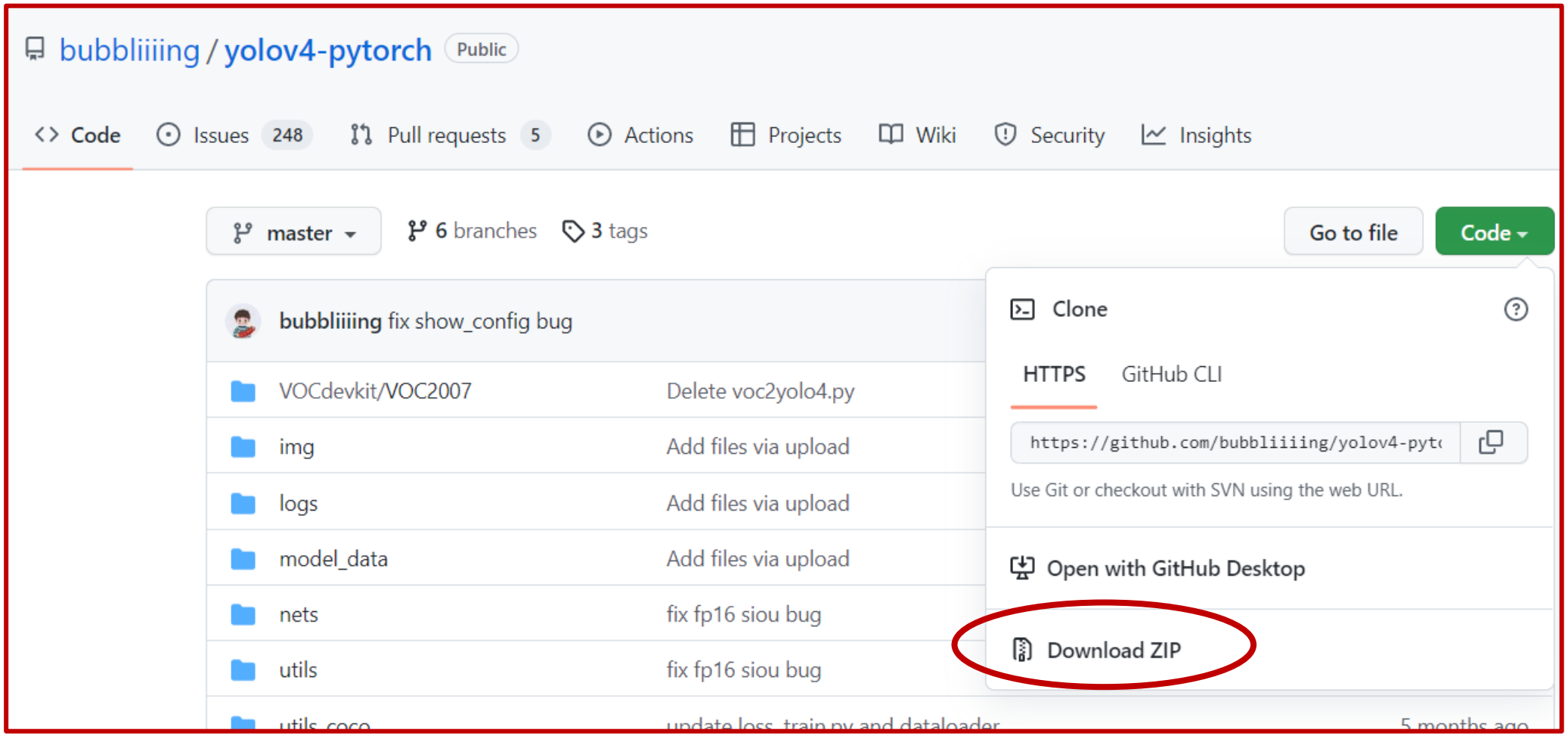
参考資料PDF

1 ：Anacondaのインストール
2 ：cuDNNとCUDAのインストール
3 ：pytorch環境の配置
4 ：必要ライブラリのインストール

scipy==1.4.1
numpy==1.19.2
matplotlib==3.2.1
opencv_python==4.2.0.34
tensorflow_gpu==2.4.0
tensorflow_cpu==2.2.0
tqdm==4.46.1
Pillow==8.2.0
h5py==2.10.0

YOLOモデルのダウンロード

- https://github.com/bubbliiiing/yolov4-pytorch



coco dataset

https://cocodataset.org/#home

COCO
Common Objects in Context

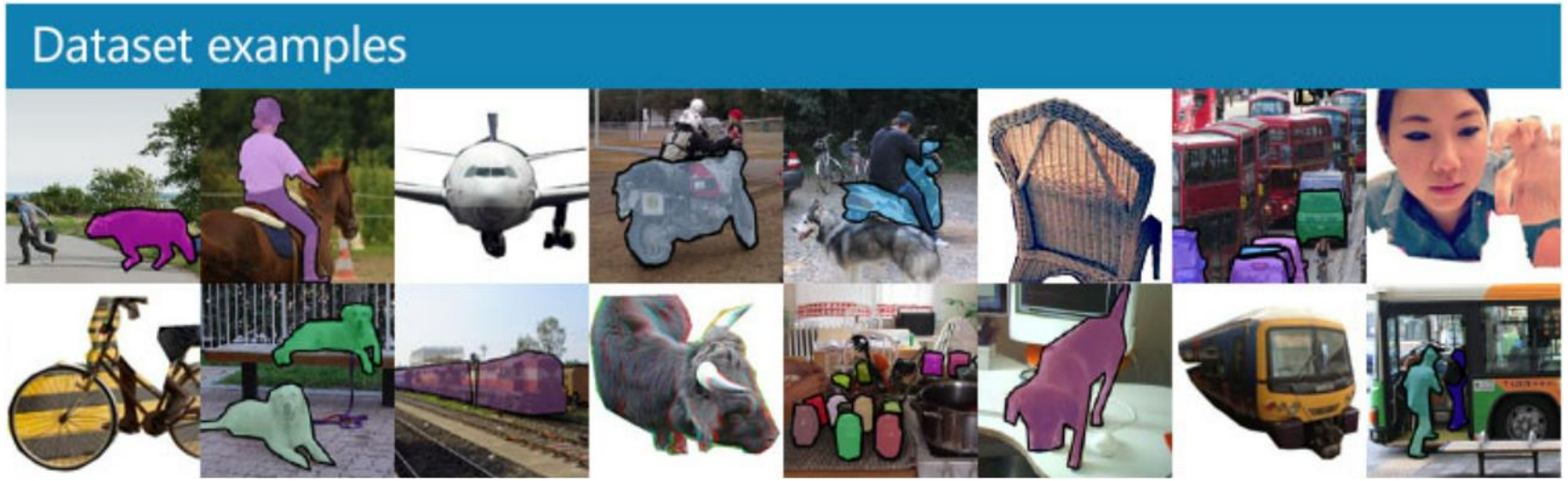
info@cocodataset.org

Home | People | Dataset | Tasks | Evaluate

What is COCO?

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

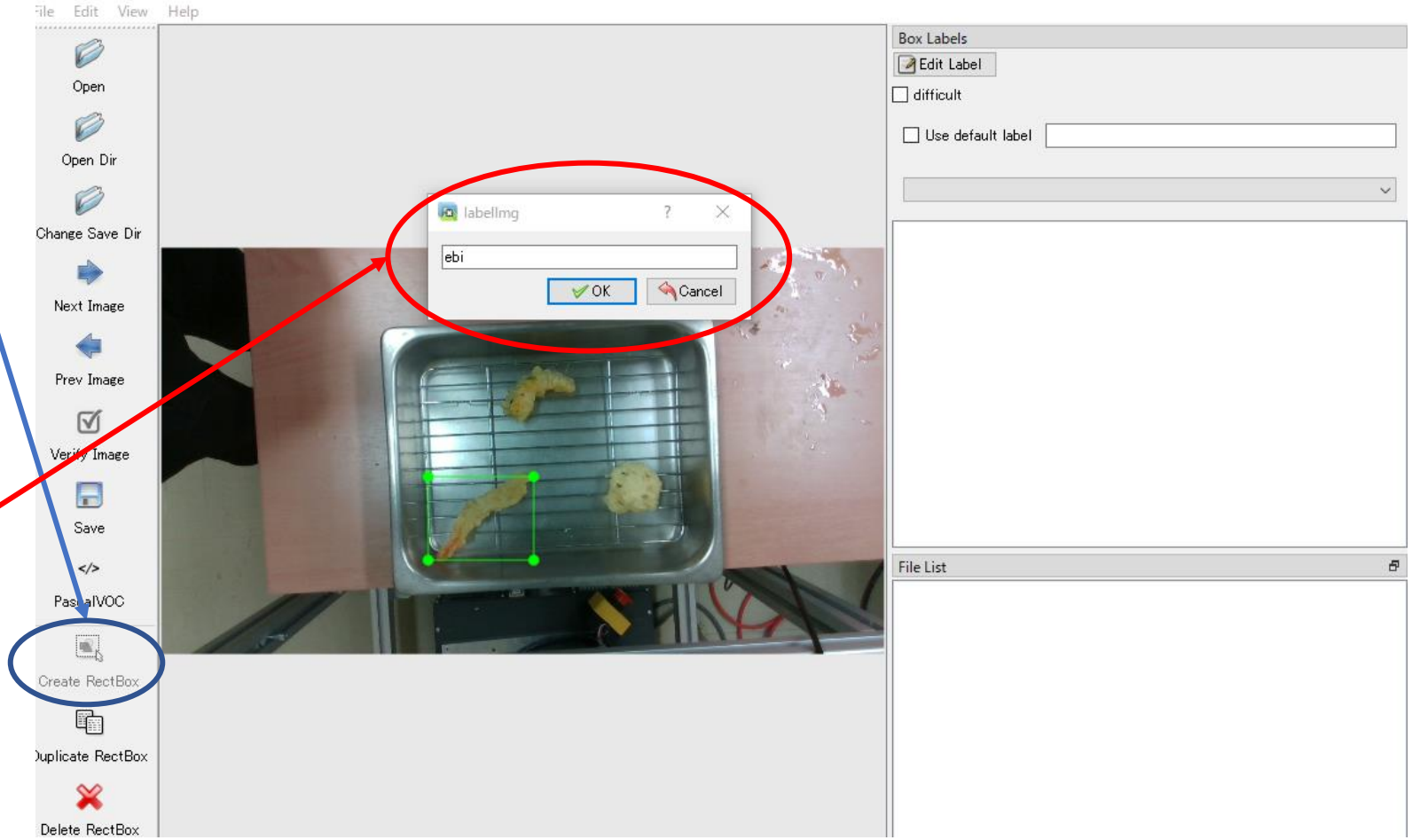
- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints



学習データの準備（アノテーション作業）

Create RectBoxをクリックして、矩形領域を作って、目標をマーキングする

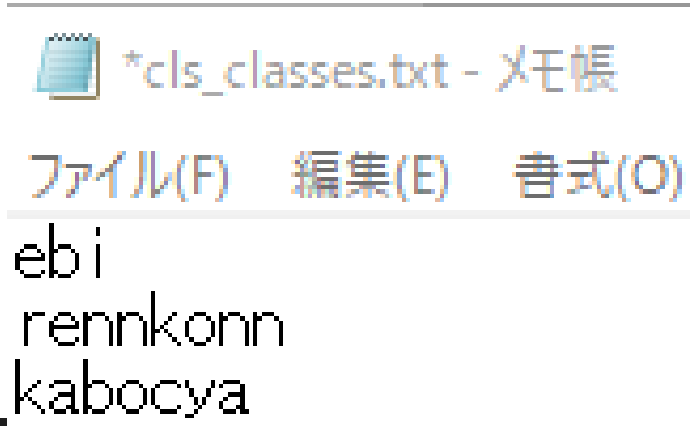
labelImgを
インストールする
必要がある



ターゲット名前の記入

モデルの修正

cls_classes.txtの例



GPUがない場合Falseに変更

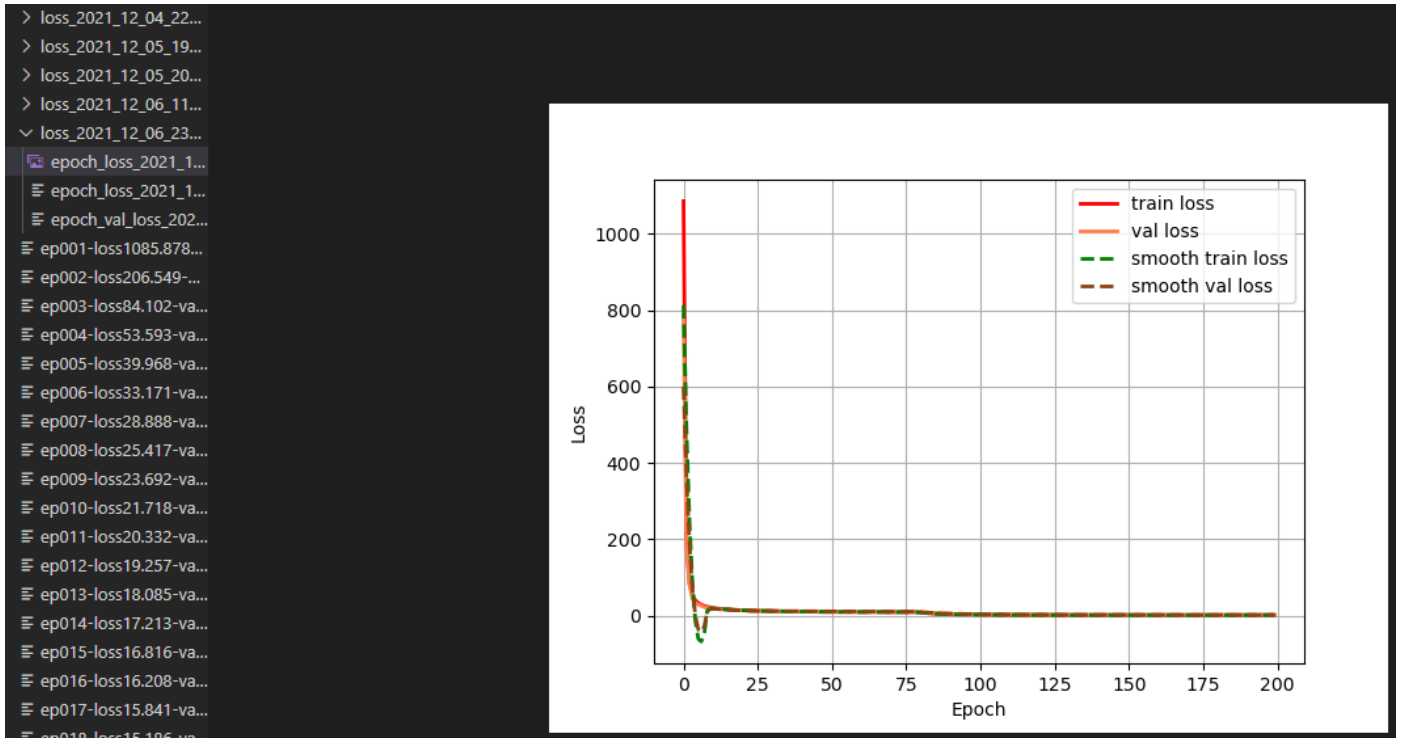
```
38  if __name__ == "__main__":
39      #-------------------------------#
40      #   GPU使ってない場合は、Falseに変更
41      #-------------------------------#
42      Cuda = True
43      #---------------------------------------------------------#
44      #  要注意 classes_pathは自分のラベルファイルを対応する。
45      #---------------------------------------------------------#
46      classes_path    = 'model_data/cls_classes.txt'
47      #---------------------------------------------------------#
```

cls_classes.txtファイルを作り、ラベルの名称を記入する。classes_passのバスはcls_classes.txtのバスに変更

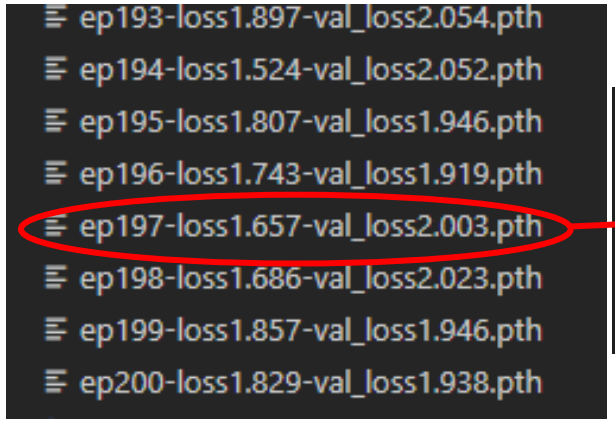
トレーニング

1 ： voc_annotation.py先ず実行する
　　学習用データ写真とテスト用データ写真を分ける

2 ： train.pyを実行して、学習を行う



モデルの検証



最後の学習結果の１つを使い、model_pathに代入

```
31      #-------------------------------------------------------------------
32      "model_path"        : 'logs/ep197-loss1.657-val_loss2.003.pth',
33      "classes_path"      : 'model_data/cls_classes.txt',
34      #-------------------------------------------------------------------
```

yolo.py

最後に（ predict.py ）を実行し、任意写真のバスを入力すれば、物体検出ができる

検出結果