

# 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

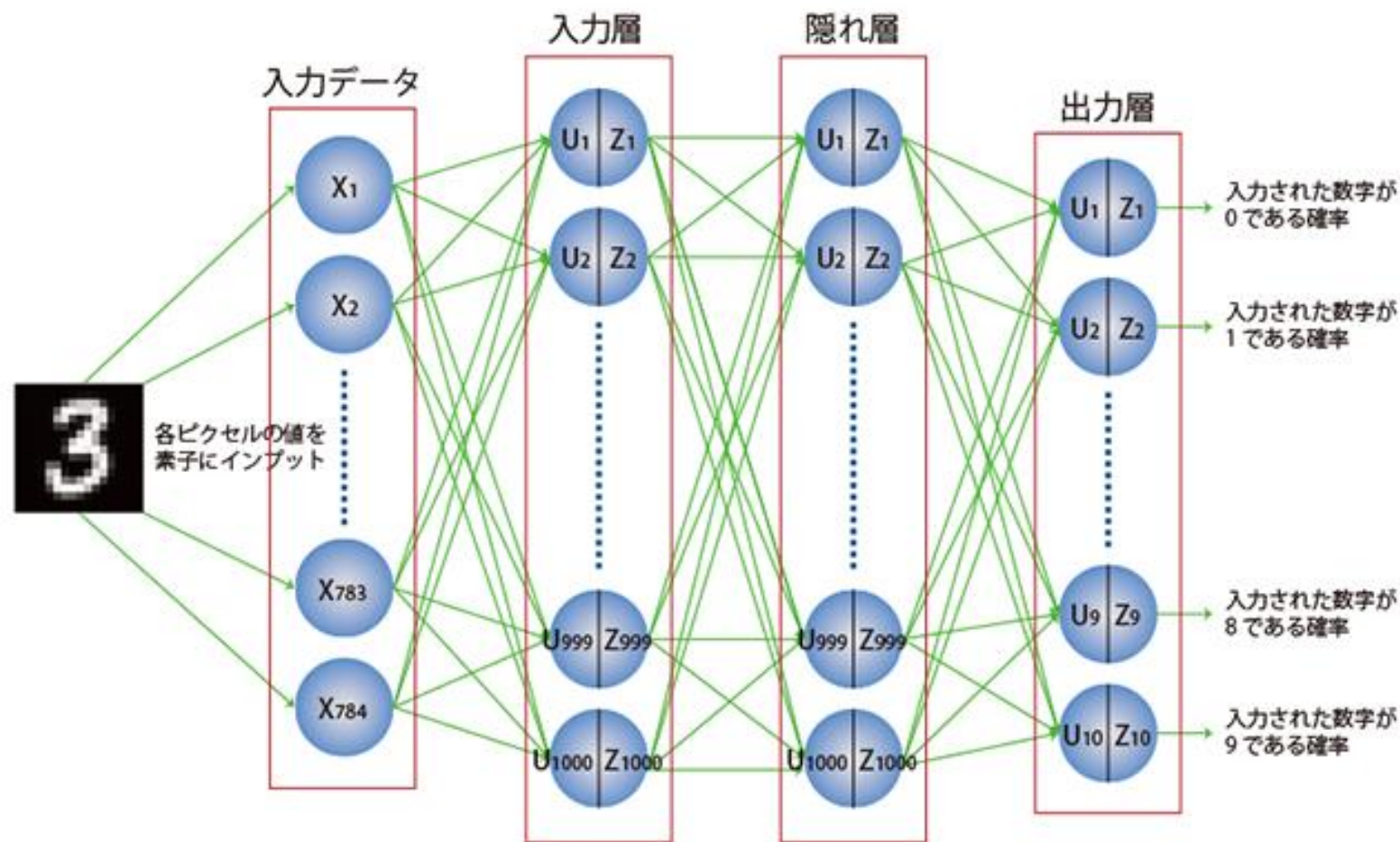
2023.07.14

# 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
- レポート

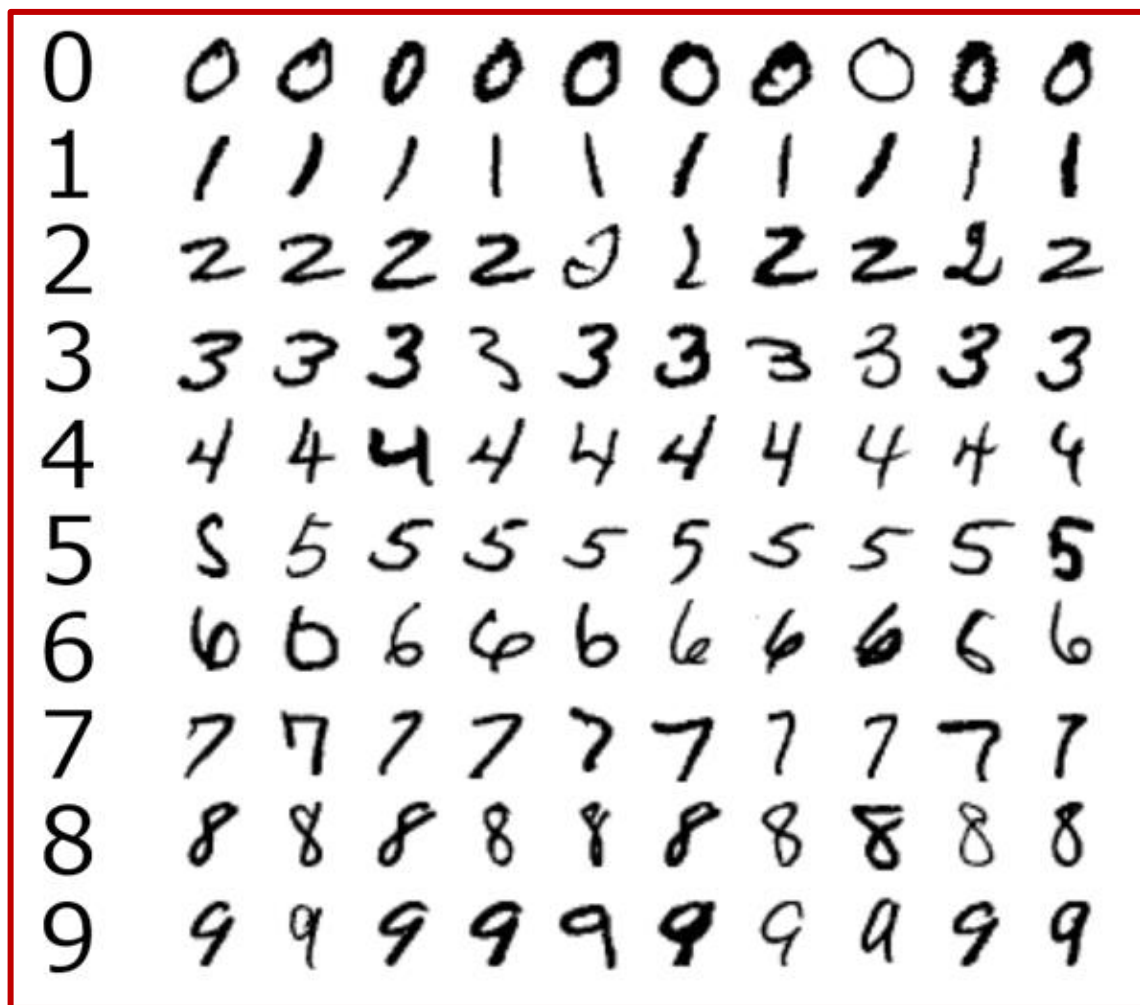
# 深層学習

- 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



# 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*

`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,validation_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

# YOLOとは

# YOLO

You Only Look Once

複数の候補の  
bounding boxを作る

複数複数回の  
予測をやる  
(遅い)

従来の手法

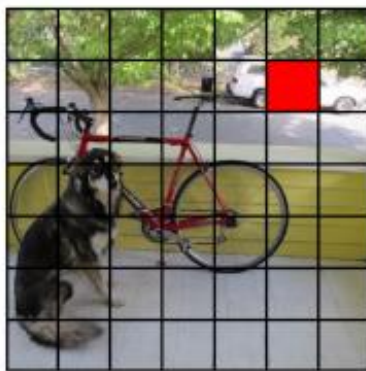


ニューラル  
ネットワー  
クの  
分類モデル



画像をSxSのcellに分割する

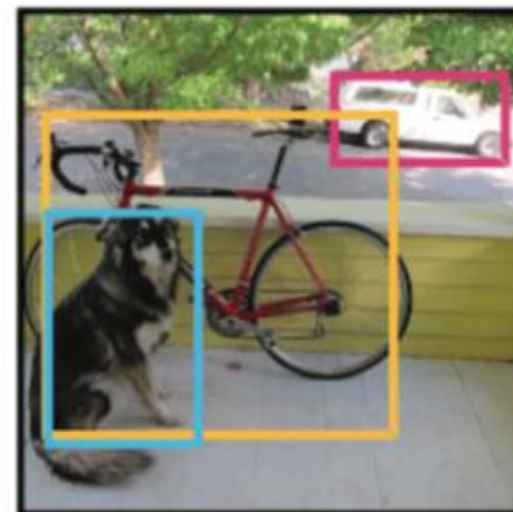
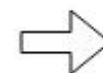
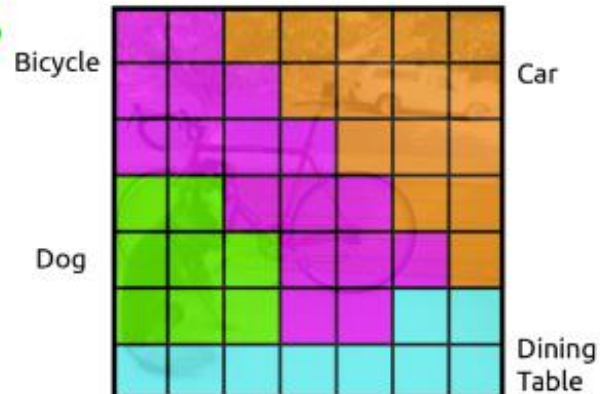
YOLO



一回だけ  
予測をやる  
(速い)



ニューラル  
ネットワー  
クの  
分類モデル



# 環境設定

## YOLOv4

- 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

# モデルのダウンロード

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

The screenshot shows the GitHub repository page for 'bubbliiiing/yolov4-pytorch'. The repository is public and has 248 issues, 5 pull requests, and 3 tags. The current branch is 'master'. The repository contains several folders: 'VOCdevkit/VOC2007', 'img', 'logs', 'model\_data', 'nets', 'utils', and 'utils\_coco'. The 'Download ZIP' button is circled in red.

Repository: bubbliiiing / yolov4-pytorch (Public)

Code Issues (248) Pull requests (5) Actions Projects Wiki Security Insights

master (6 branches, 3 tags) Go to file Code

File/Folder	Commit Message
bubbliiiing	fix show_config bug
VOCdevkit/VOC2007	Delete voc2yolo4.py
img	Add files via upload
logs	Add files via upload
model_data	Add files via upload
nets	fix fp16 siou bug
utils	fix fp16 siou bug
utils_coco	update loss_train.py and dataloader

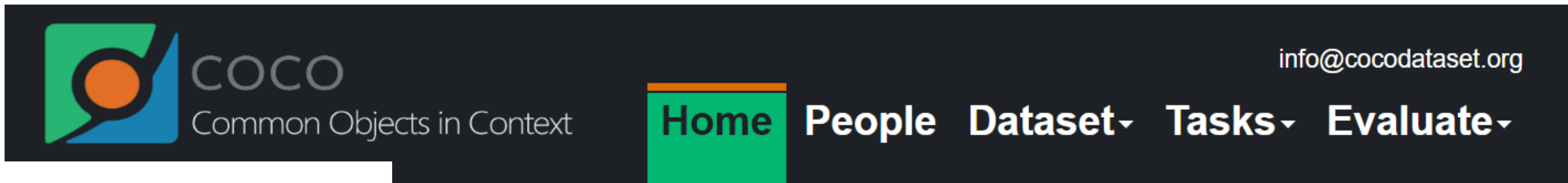
Clone (HTTPS, GitHub CLI) <https://github.com/bubbliiiing/yolov4-pytorch>

Open with GitHub Desktop

**Download ZIP**

# coco dataset

<https://cocodataset.org/#home>

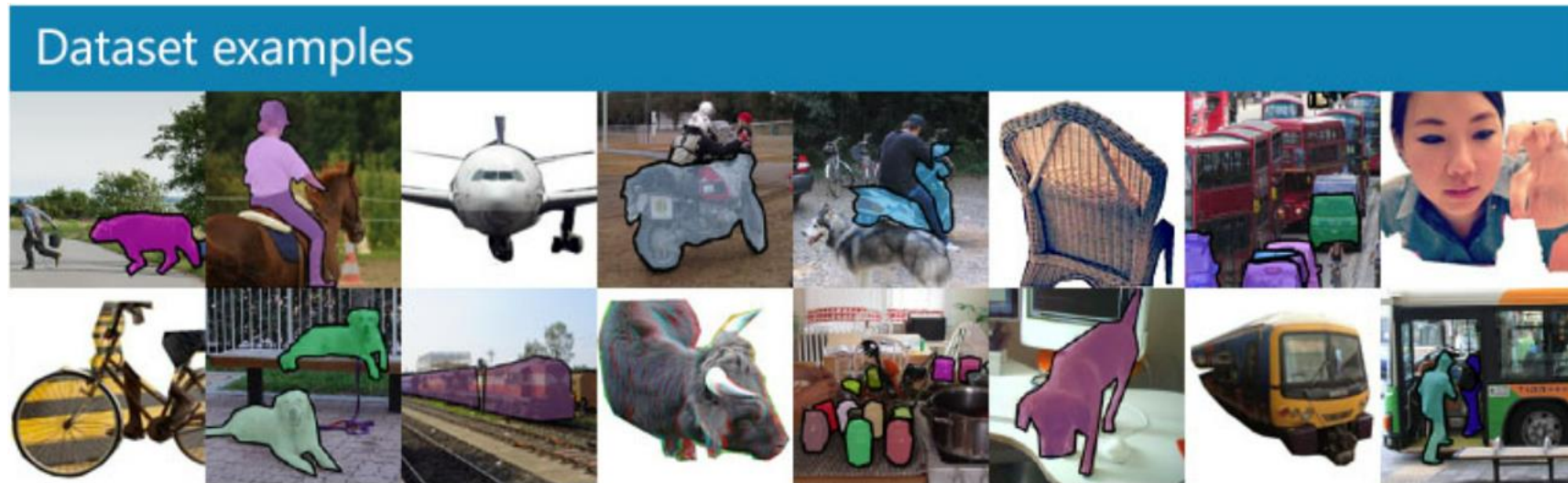


## What is COCO?



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

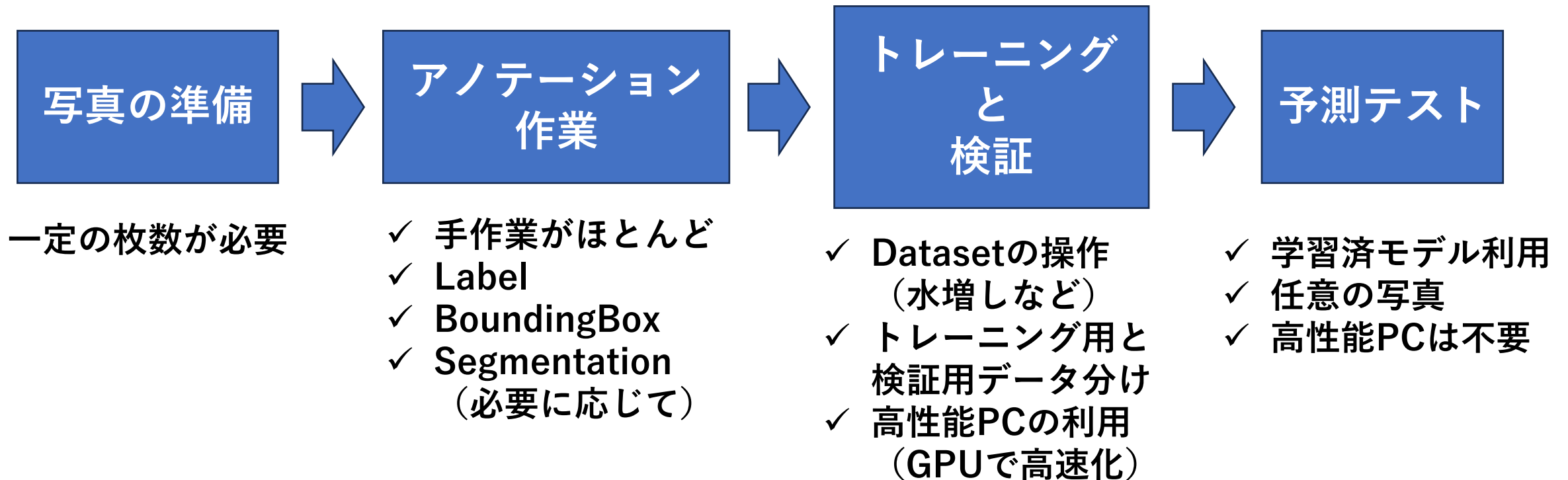
- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints





# 自分のdatasetを用いた学習とテスト

手順：

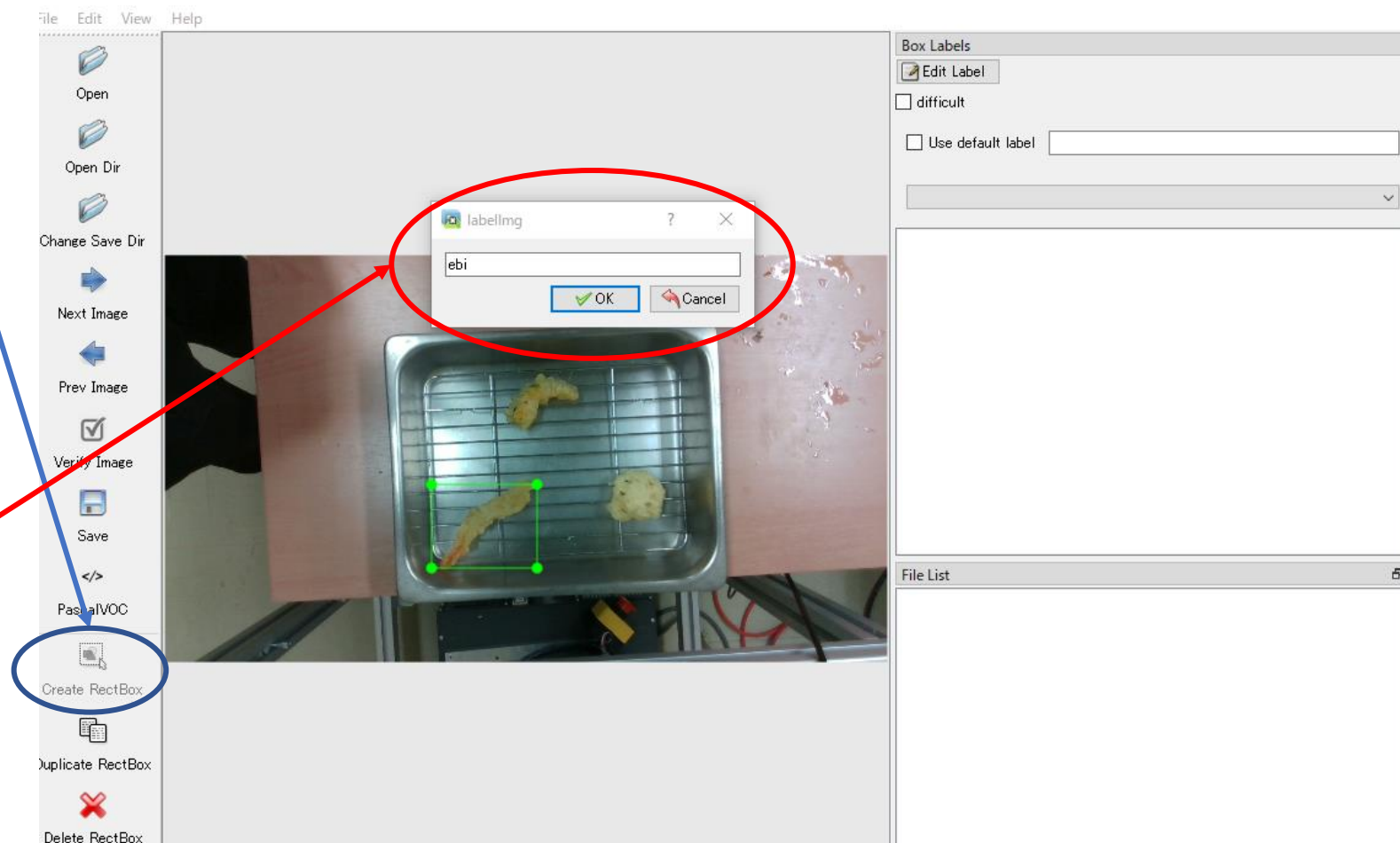


# 学習データの準備（アノテーション作業）

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

labellmg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

\*cls\_classes.txt - メモ帳

ファイル(F) 編集(E) 書式(O)

ebi  
rennkonn  
kabocya

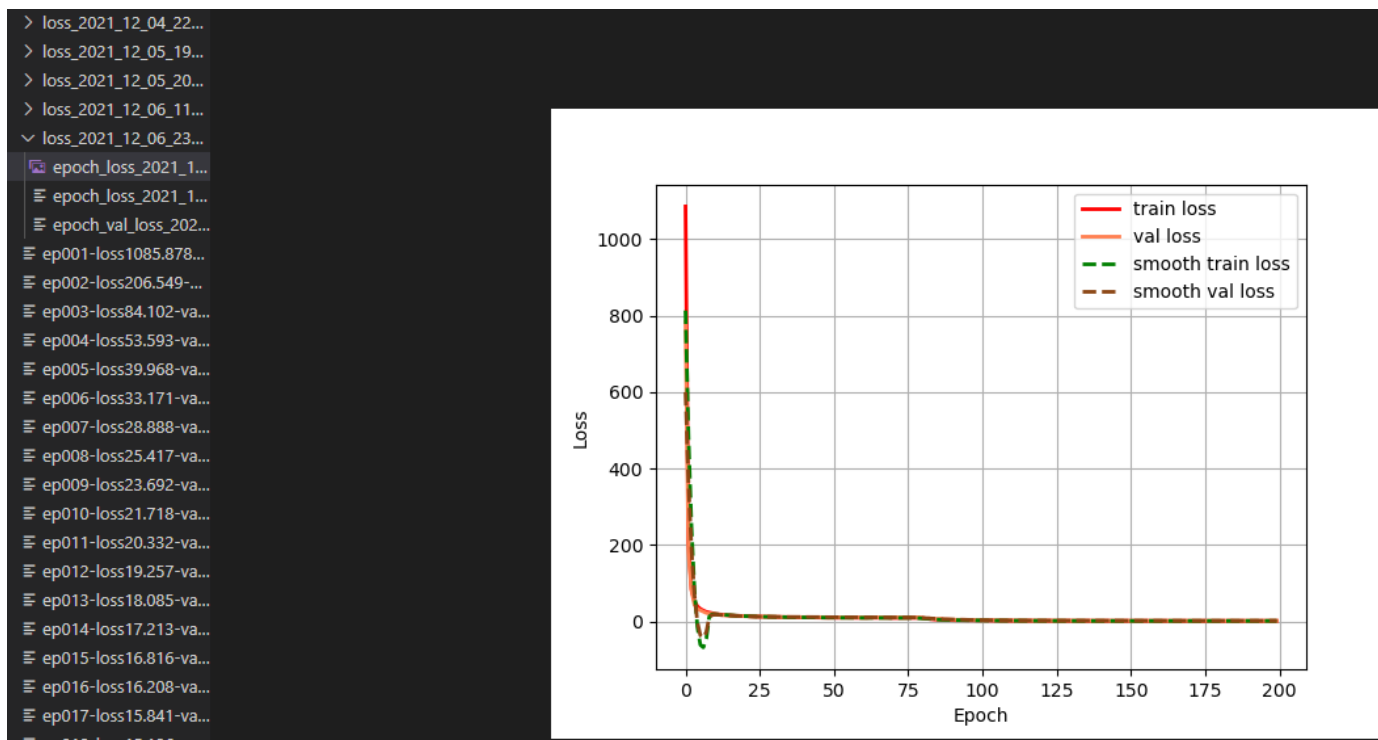
GPUがない場合Falseに変更

```
38 if __name__ == "__main__":  
39     #-----#  
40     # GPU使っていない場合は、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に対応する。  
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のパスはcls\_classes.txtのパスに変更

# 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

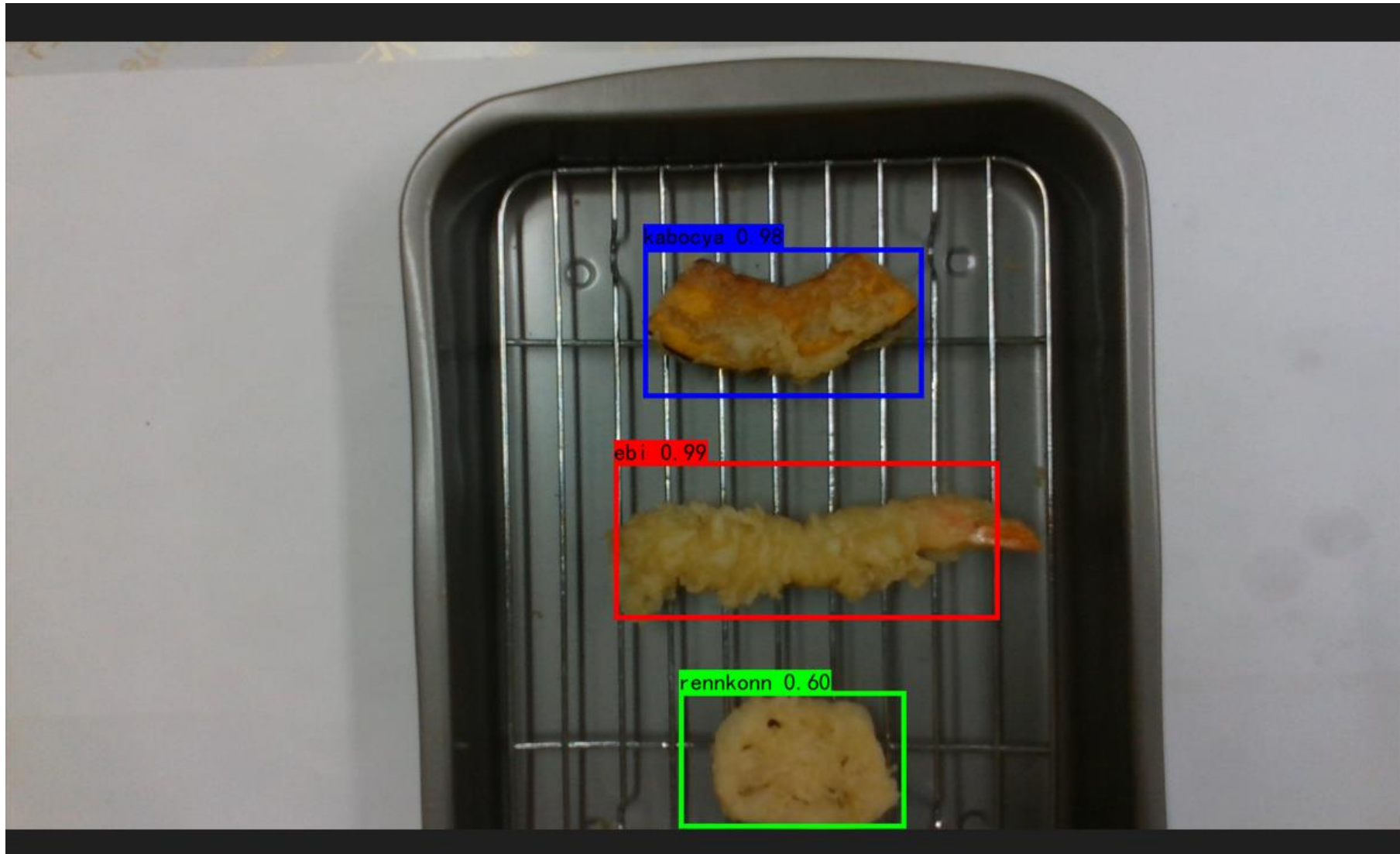
```
≡ ep193-loss1.897-val_loss2.054.pth
≡ ep194-loss1.524-val_loss2.052.pth
≡ ep195-loss1.807-val_loss1.946.pth
≡ ep196-loss1.743-val_loss1.919.pth
≡ ep197-loss1.657-val_loss2.003.pth
≡ ep198-loss1.686-val_loss2.023.pth
≡ ep199-loss1.857-val_loss1.946.pth
≡ ep200-loss1.829-val_loss1.938.pth
```

```
31 #-----
32 "model_path" : 'logs/ep197-loss1.657-val_loss2.003.pth',
33 "classes_path" : 'model_data/cls_classes.txt',
34 #-----
35 # yolo_path 代表生成器对应的相对文件 一般不修改
```

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



# 15週目レポート

- 自分で手書き数字を用意して、手書き数字認識アルゴリズムを用いて認識してみよう。
- 学習プログラムの最後に下記のコードを追加することで、手書き数字を認識できる。
- 写真「handwriting.jpg」下記の手順で用意する。

適切なサイズに変換

数字の部分を**正方形**の形状で切り抜く

グレースケールに変換

二値化にする（**背景が黒、数字が白**）

「handwriting.jpg」として画像保存

```
from PIL import Image
```

```
img = Image.open('handwriting.jpg').convert('L')  
img.thumbnail((28, 28))
```

```
img = np.array(img)
```

```
pred = model.predict(img[np.newaxis])
```

```
print(np.argmax(pred))
```

```
plt.imshow(img)
```

```
plt.show()
```