Atelier Akihabara

ワークショップ 番外編

Pepper x TensorFlow

アトリエ秋葉原 Presents.

このワークショップについて

免責事項

ワークショップ番外編は
アトリエのスタッフが作成したものであり
ソフトバンク公式のものではないことを
ご承知ください。

アトリエについて

実体験とコミュニティーで開発を促進する

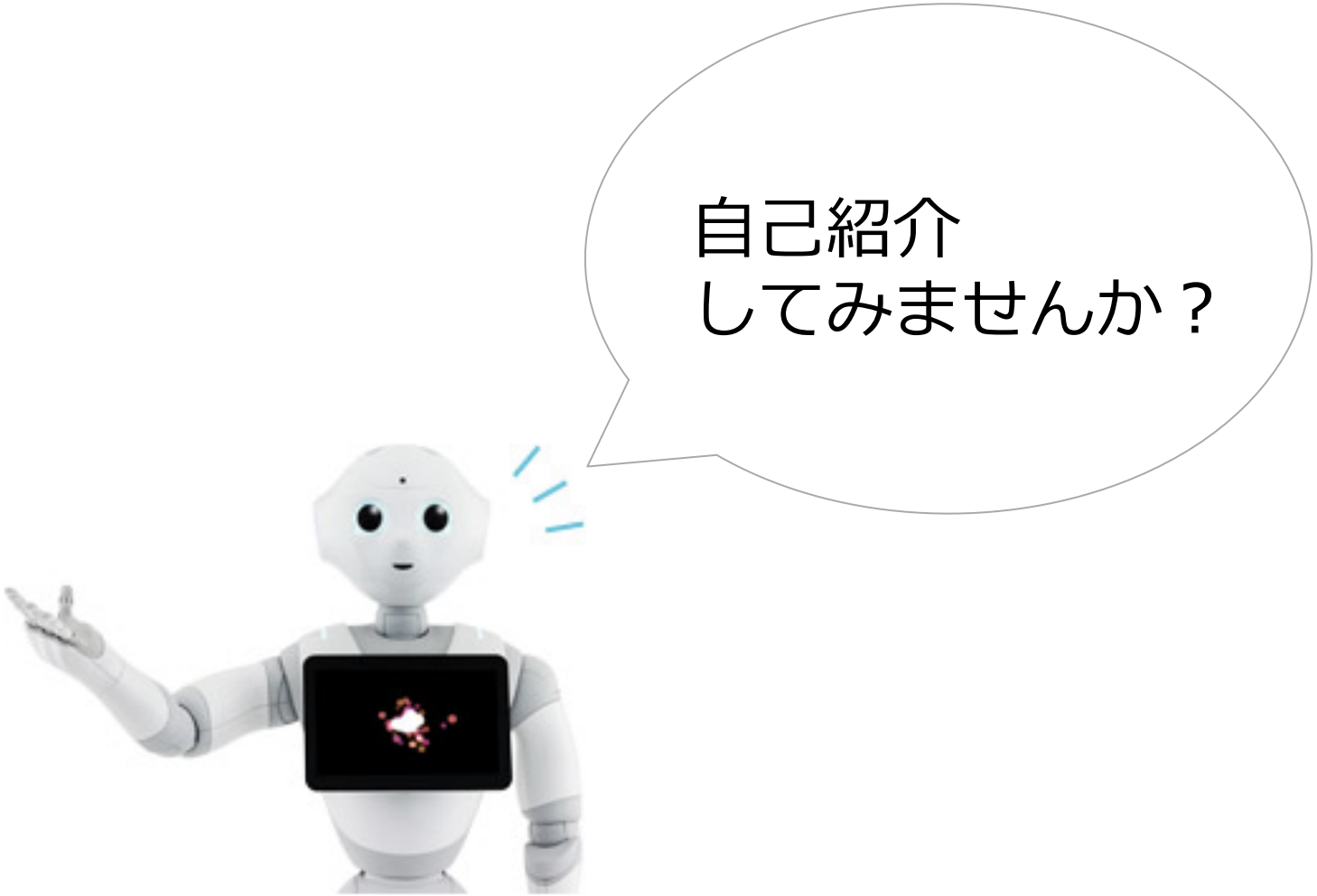
アトリエ

相互
促進

コミュニティー

✓ Pepperのアプリ開発を実体験

✓ 経験や知見を共有


自己紹介
してみませんか？

自己紹介

- **お名前**
- **ご所属**
- **本日の意気込み**
- **開発経験（Pepper/Python/機械学習）**

※ 本ワークショップでは、Pythonプログラムの開発を行います。
Chroregraphe は使用しませんのでご注意ください。

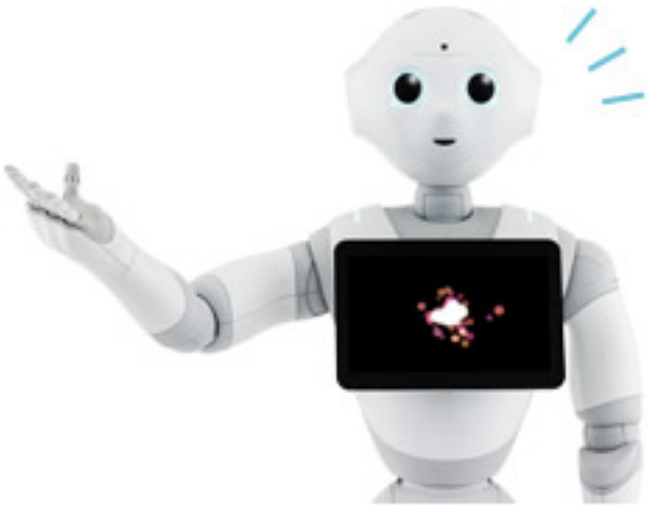
例：

本日の案内を勤めさせていただきます、

松尾　映里 (Matsuo Eri) と申します。

本WSの内容

1. Pepper x Deep Learning
2. 環境構築
3. TensorFlowで物体認識
4. Keras+TensorFlowでリアルタイム物体検出
5. おまけ



Pepper x DeepLearning

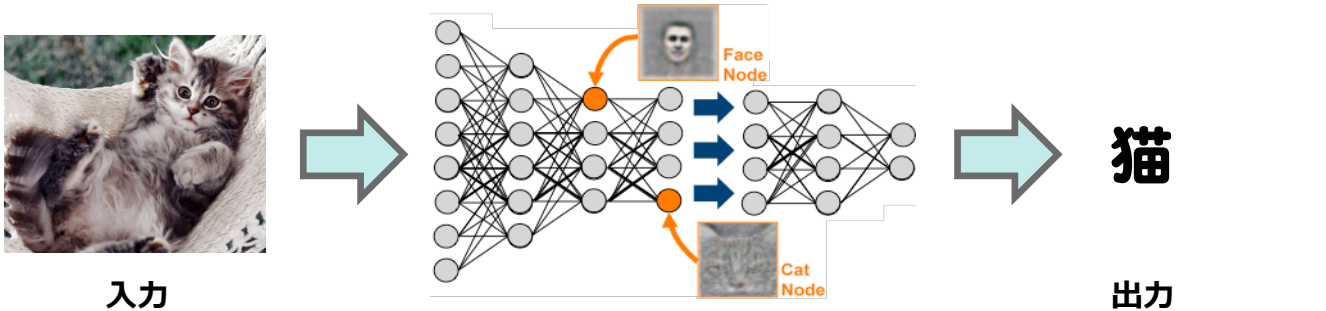
Deep Learning とは

- Deep Learning（深層学習）
  - 生物の脳機能を模した機械学習手法 Neural Network のなかでも「深い」構造を持った **Deep Neural Network による機械学習**
  - 近年の**人工知能**ブームの火付け役

    例：AlphaGo (囲碁) , Ponanza (将棋), 自動運転, Google翻訳, 商品推薦 (Amazon), 音声認識 (Yahoo!), etc.

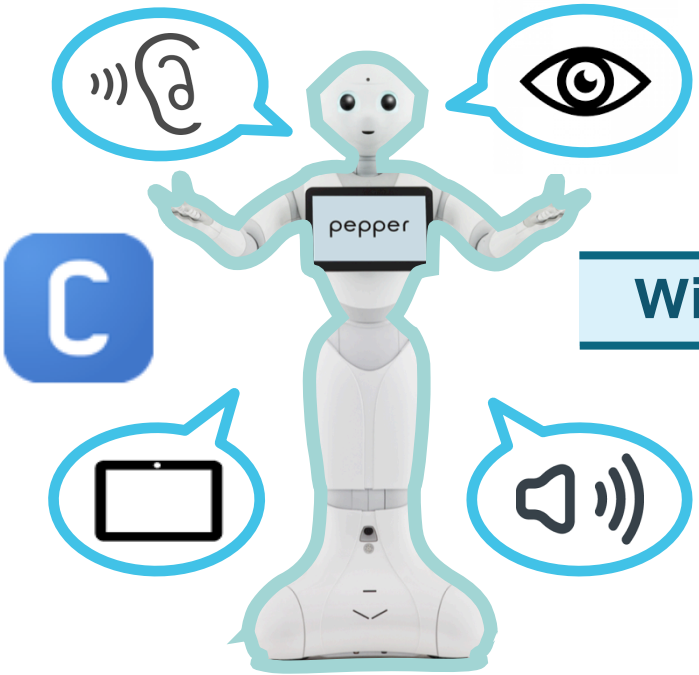
  - 入力情報から出力情報を予測するように**ネットワーク（モデル）を学習**する

    例：画像から何が写っているかを予測（画像認識）



Pepper x Deep Learningの主流

様々な Deep Learning サービスとの連携が広まる



Wi-Fi

Deep Learning
Web API

IBM Watson

IBM Bluemix

Microsoft
Azure

Google Cloud Platform

Choregrapheを使って、Pepperで取得した情報を学習済みAPIサービスで処理する

Deep Learningの導入方法

1．APIサービスを使う


- 既に学習済みのモデルが使用できる
- 学習データを独自に与える場合でも構造を自作する必要がない

⇒ Deep Learning 技術を**今すぐに**活用してみたい方向け

主流手法

2．フレームワーク等を使って開発を行う


theano
Caffe

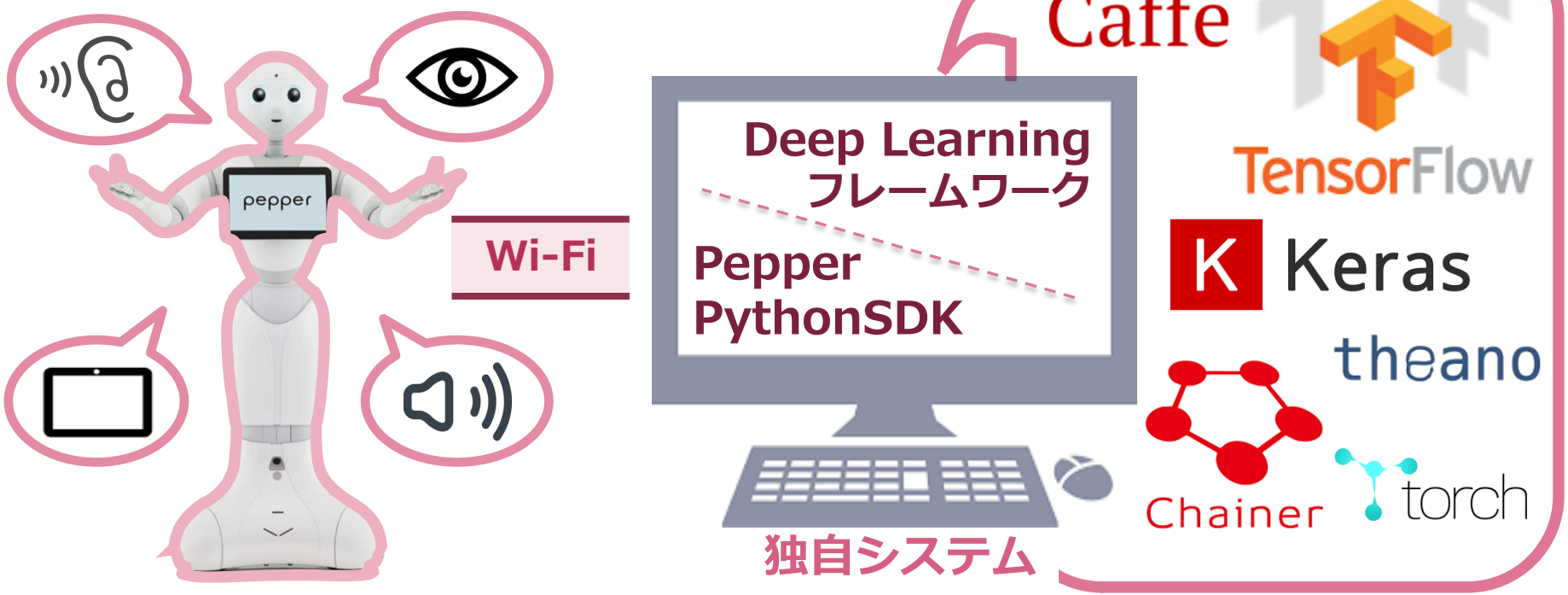
- 自分の手元でモデルを構築・学習できる
- 用途に応じた工夫や他手法と組み合わせるなど柔軟な開発ができる

⇒ **独自の Deep Learning 技術**を導入したい方、
最新手法を取り入れたい方向け

今回の内容

本WSの目標

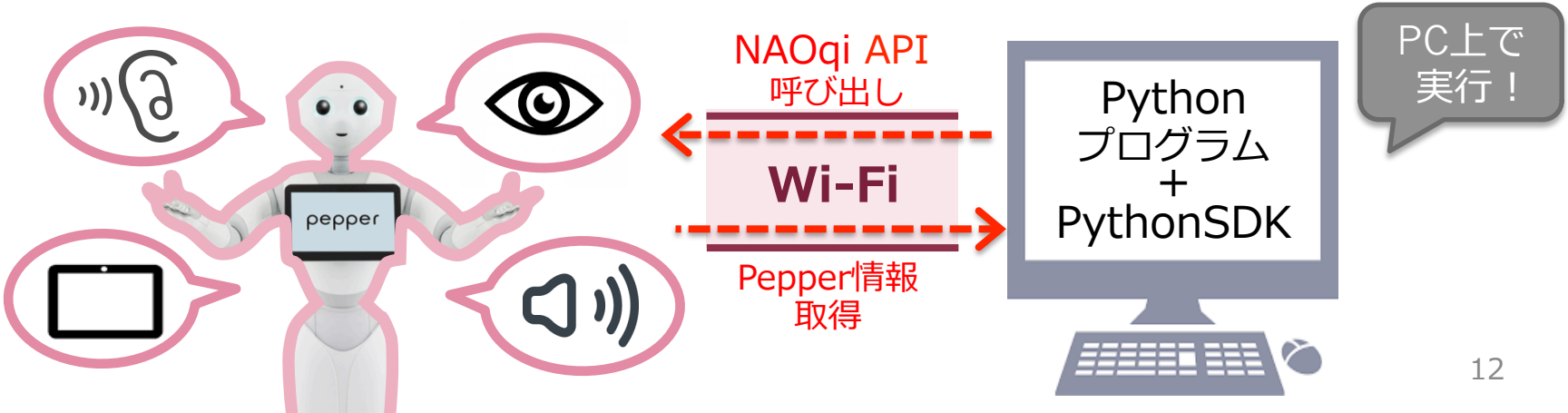
独自の Deep Learning モデルの適用



PythonSDK を使って、Pepperで取得した情報を **Tensorflowモデル** で処理する

PythonSDK

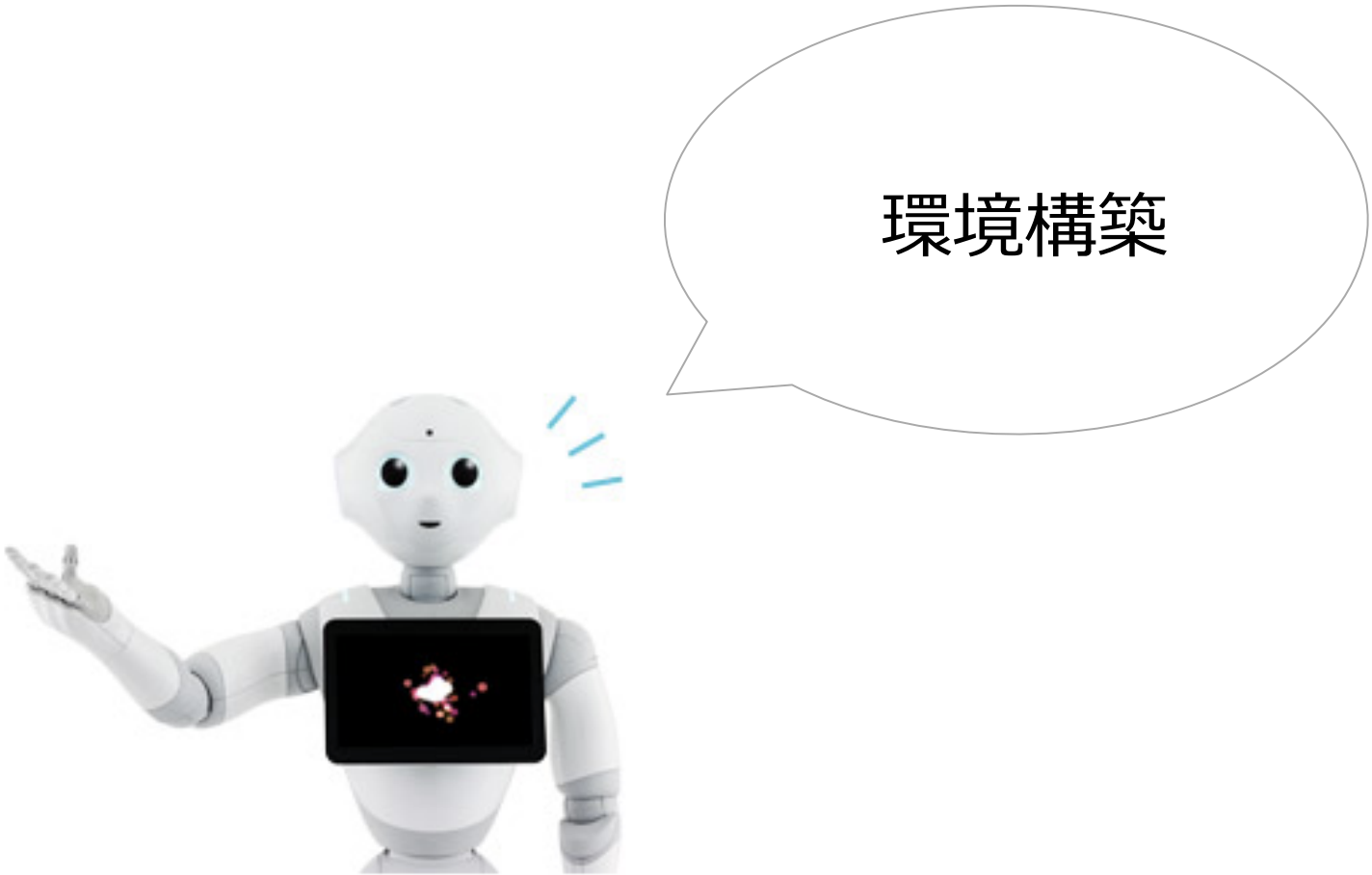
- PythonSDK
  - Pepper をリモートコントロールするためのSDK
  - **PC上で実行したPythonスクリプト**から
    ネットワークを介して**Pepperを操作**できる
    - 実行環境は NAOqi (Pepper) ではなく手元の PC となる
  - Pepper本体に組み込めないライブラリ等を使用したいときなどに便利
    - Pepper上にTensorFlowをインストールするのは難しい
    - 参考：書籍「Pepperソフトウェア小ネタ帳2」



TensorFlow/keras

- TensorFlow 〔https://www.tensorflow.org/〕
  - Google が開発した **Deep Neural Network のための Python ライブラリ**
  - 世界的に最も使われている Deep Learning のフレームワーク
  - オープンソース、商用利用可能

- Keras 〔https://keras.io/ja/〕
  - TensorFlow や Theano をバックエンドとして用いるラッパーライブラリ
    より簡単なコードで記述できる別のライブラリのように使用できるが、
    実際には Tensorflow (あるいは Theano) の機能を利用して実行される
  - つまり、**TensorFlow の機能を簡単に使用できるようにする Python ライブラリ**


環境構築

今回の実行環境

- Mac OS X 10.10.5 Yosemite
  - GPU不使用
- python 2.7.14
- OpenCV 3.3.0
- Tensorflow 1.3.0
- Keras 1.2.2
- PythonSDK 2.5.5.5

事前準備

主にインストールが必要なもの

- PythonSDK
- OpenCV
- Tensorflow
- Keras

  – 他、いくつかのPythonライブラリも必要
  – インストールの手順は個々の環境に大きく依存するので、
  今回のWSでは詳細は割愛

macでの手順例 ①

1) /Library/Frameworks/…下のPython2.7にPythonSDKをインストール
   - /Library/Frameworks/Python.framework/Versions/2.7/…
   - 参考：https://qiita.com/kyohara/items/ae0dd0f57413caa90674
   - **仕様上、macではシステムデフォルトのPythonでしかPythonSDKが使用できないため**

2) /Library/Frameworks/…下のPython環境に必要なパッケージをインストール
   - （例）brew install opencv
     pip install tensorflow==1.3.0, keras==1.2.2
   - 他、必要なライブラリ
     （例）brew install graphviz
     pip install pydot==1.1, graphviz

3) /Library/Frameworks/…下のPythonを使ってプログラムを実行

macでの手順例 ②環境を汚さない方針

1) /Library/Frameworks/...下のPython2.7にPythonSDKをインストール

2) anacondaによる仮想環境を別に用意し、必要なパッケージをインストール
   - 参考：https://qiita.com/yampy/items/37c607fdf77a919cda5d
   - 上記ページはPython3向け設定ファイルなので、Python2向けに変更が必要

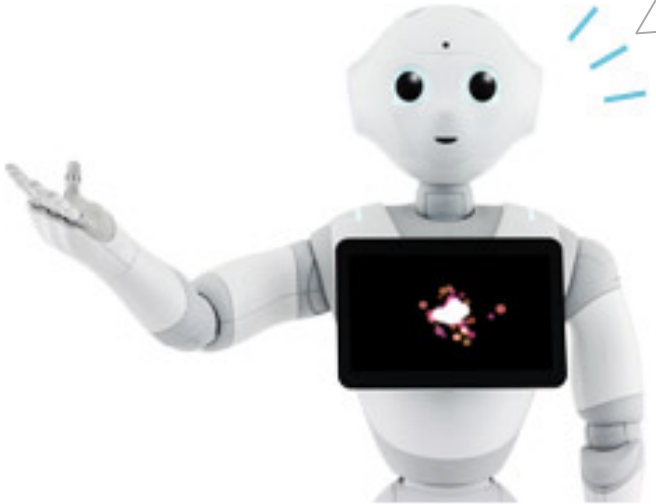
3) /Library/Frameworks/...下のPythonからanaconda仮想環境を参照するよう設定
   - anaconda上の python で参照している環境を sys.path() で表示し、
     export PYTHONPATH=...により実行環境に追加

4) /Library/Frameworks/...下のPythonを使ってプログラムを実行

※ 構築指針：「anacondaによる仮想環境で別途環境構築」
・仕様上、macではシステムデフォルトのPythonでしかPythonSDKが使用できない
・TensorflowやKerasなどのインストールでシステムデフォルトのPython環境に手を加えるのはできるだけ避けたい...

※ システムにそのままインストールしても別にOK！という方は前ページの通り直接インストールした方がずっと楽です！

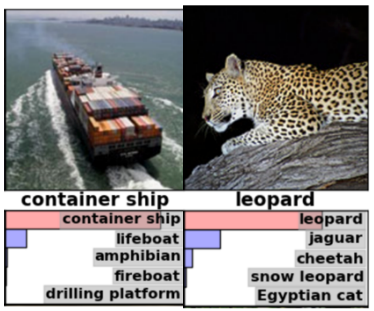
TensorFlowで物体認識



TensorFlowで物体認識

- TensorFlowで学習済みの**物体認識モデル**を使って、
  Pepperが今見ているものを認識してしゃべるプログラムを作ろう！

- アウトライン
  ① **TensorFlow**で学習済みのモデルの読み込み
  ② **PythonSDK**でALVideoDeviceによるPepperのカメラ情報のリモート取得
  ③ Deep Learningモデルによる物体認識の実行
  ④ 認識結果を数値から言語化
  ⑤ ALTextToSpeechによる結果の発話 (→ 2. へ繰り返し)



[https://www.tensorflow.org/tutorials/image_recognition]

classify_bypepper.py に学習済みモデルを対応

- 今回の使用モデル：「Inception-V3」
  - ImageNet2012で学習された、画像を1000クラスに分類するモデル
    - ダウンロード：http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz
    - 論文：https://arxiv.org/abs/1512.00567
  - 本WS資料では「inception-2015-12-05/」にダウンロード済み

- classify_bypepper.py を一部変更（穴埋め形式）
  - ✓ L21　:'＊＊＊　ここを変更(1)　＊＊＊'　→　'inception-2015-12-05/imagenet2012_id_to_label.txt'
  - ✓ L42　:'＊＊＊　ここを変更(2)　＊＊＊'　→　'inception-2015-12-05/classify_image_graph_def.pb'
  - ✓ L98　:'＊＊＊　ここを変更(3)　＊＊＊'　→　'softmax:0'
  - ✓ L100　:'＊＊＊　ここを変更(4)　＊＊＊'　→　'DecodeJpeg/contents:0'
  - 実行例：$ python classify_bypepper.py 192.168.100.1
  - 停止： ctrl-C

自分のPepperの
IPアドレス

classify_bypepper.py [1/3]

```
# --- coding: utf-8 ---
import argparse
import numpy as np
import pickle
from naoqi import ALProxy
import tensorflow as tf
import cv2
```

必要ライブラリ読み込み

```
# Pepper の IP アドレスを引数で取得
parser = argparse.ArgumentParser()
parser.add_argument('ip_ad', help='my IP address(ifconfig)')
args = parser.parse_args()
ip_ad = args.ip_ad
```

実行時引数からPepperのIPアドレスを取得、変数 ip_ad に保存
※実行例：python classify_bypepper.py 192.168.100.1

④

```
# ラベルID(int) -> ラベル名(string)変換用クラス
class NodeLookup(object):
  def __init__(self):
    # node_lookup : dict (ID(int) -> label(string))
    self.node_lookup = {}
    label_lookup_path = 'inception-2015-12-05/imagenet2012_id_to_label.txt' # label file 読み込み
    if not tf.gfile.Exists(label_lookup_path):
      tf.logging.fatal('File does not exist %s', label_lookup_path)
    lines = tf.gfile.GFile(label_lookup_path).readlines()
    for line in lines:
      key, label = line.strip().split('\t')
      self.node_lookup[int(key)] = label
    print self.node_lookup

  def id_to_string(self, node_id):
    if node_id not in self.node_lookup:
      return ''
    return self.node_lookup[node_id]
```

NodeLookup：ラベルIDからラベル名を検索するための Class
Tensorflowでの物体認識結果はID(数値)で返されるので、
人間にわかるように名前(文字列)に変換する

- **初期化関数「__init__()」**
  「inception-2015-12-05/imagenet2012_id_to_label.txt」
  からID-ラベル名の対応を読み出し、対応する辞書を作成
- **関数「id_to_string (ID)」**
  ID(数値)を与えられたらそのラベル名(文字列)を返す

④

```
# ID -> label 辞書 Class の作成
node_lookup = NodeLookup()
```

NodeLookup Class (↑) のインスタンス作成
__init__()が実行され、今後は
node_lookup.id_to_string (ID数値) で検索が可能

classify_bypepper.py [2/3]

```
# 保存された GraphDef から Tensorflow graph の作成
with tf.gfile.FastGFile('inception-2015-12-05/classify_image_graph_def.pb', 'rb') as f:
  graph_def = tf.GraphDef()
  graph_def.ParseFromString(f.read())
  tf.import_graph_def(graph_def, name='')
```

①

```
# NAOqi module の取得
tts = ALProxy('ALTextToSpeech', ip_ad, 9559)
tts.setLanguage('English')
videoDevice = ALProxy('ALVideoDevice', ip_ad, 9559)
```

② ⑤

```
# 上部カメラの監視
AL_kTopCamera = 0
AL_kQVGA = 2   # 0: kQQVGA (160x120), 1: kQVGA (320x240), 2: kVGA (640x480)
AL_kBGRColorSpace = 13   # 0: kYuv, 9: kYUV422, 10: kYUV, 11: kRGB, 12: kHSY, 13: kBGR
fps = 5   # 5, 10, 15, 30
nameID = 'classify_TF'
videoDevice.unsubscribe(nameID) # 前回実行時のプロセスが残っていたら終了
captureDevice = videoDevice.subscribeCamera(
    nameID, AL_kTopCamera, AL_kQVGA, AL_kBGRColorSpace, fps)

width = 640
height = 480
```

②

TensorFlow学習済みモデルの読み込み
「inception-2015-12-05/classify_image_graph_def.pb」から、graph_def にグラフ情報を取得

PythonSDKで「ALTextToSpeech」「ALVideoDevice」取得
引数：「API名」「PepperのIPアドレス」「ポート番号」
ラベル名は英語なので、言語設定を英語に設定しておく

ALVideoDeviceによるカメラ情報取得のための設定
詳細は公式ドキュメントにて

classify_bypepper.py [3/3]

50ミリ秒待ってから繰り返し（q入力で終了）

```
while True:
  result = videoDevice.getImageRemote(captureDevice)  # Pepperから画像取得
  image = np.zeros((height, width, 3), np.uint8)

  if result == None:
    print 'cannot capture.'
  elif result[6] == None:
    print 'no image data string.'
  else:
    values = map(ord, list(result[6]))
    # Pepperから得られる画像情報は一列なのでxy,RGBにマッピング
    i = 0
    for y in range(0, height):
      for x in range(0, width):
        image.itemset((y, x, 0), values[i + 0])
        image.itemset((y, x, 1), values[i + 1])
        image.itemset((y, x, 2), values[i + 2])
        i += 3

    image = cv2.resize(image, (300,300))

    # 画像を一度JPEGで保存
    cv2.imwrite('tmp.jpg', image)

    # JPEGから読み出し
    image_data = tf.gfile.FastGFile('tmp.jpg', 'rb').read()

    # Tensorflow Session 開始
    with tf.Session() as sess:
      # 抽出層の指定
      # 'softmax:0': 1000 labels の正規化された prediction.
      # 'pool_3:0': (最終layer前のlayer) 2048次元の画像特徴量(float)
      softmax_tensor = sess.graph.get_tensor_by_name('softmax:0')
      predictions = sess.run(softmax_tensor,
                             {'DecodeJpeg/contents:0': image_data}) # 実行
      # 'DecodeJpeg/contents:0': 画像（JPEGから読み込んだもの）.
      summary_writer = tf.summary.FileWriter('./graphs', graph=sess.graph) # グラフ保存
      predictions = np.squeeze(predictions) # squeeze:大きさ1の次元を除去

      node_id = predictions.argsort()[-1] # 最も確率が高いラベル
      string = node_lookup.id_to_string(node_id)
      string = string.split(', ')[0] # ラベル名の「,」以降を除去
      score = predictions[node_id]
      print('%s (score = %.5f)' % (string, score))
      tts.say(string) # Pepperでラベル名を発声

    k = cv2.waitKey(50);
    if k == ord('q'):  break; #「q」を入力して終了

tts.setLanguage('Japanese')
videoDevice.unsubscribe(nameID)
```

②

③

④

⑤

Pepperから画像取得・処理

- result[6]に画像値が１次元配列で保存されているので、画像として配列し直す
- 学習されたモデルの入力300x300にリサイズ
- 学習されたモデルはJPEGエンコードの画像を入力とする仕様（？）なので、一度JPEGとして保存＆読み込み

TensorFlow 学習モデルによる物体認識

- softmax layer (最終層) の出力を取得するように設定
  途中の層を取得して他のタスク等に活用するときは別のlayer名を設定する
- 「sess.run」で計算実行、結果を predictions に保存

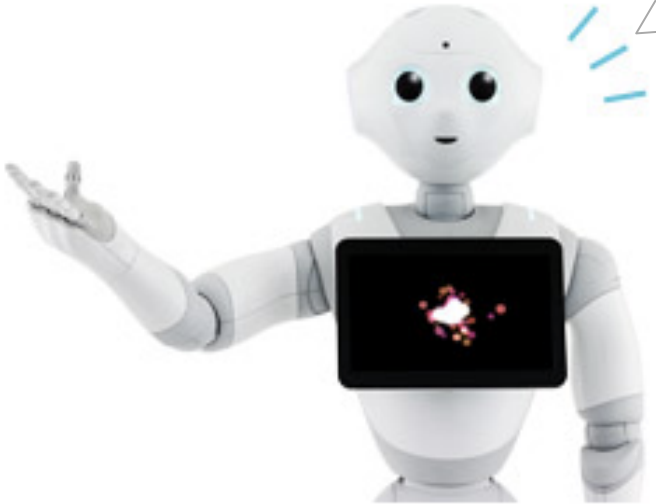
※ Tensorboard でモデルを確認可能（$ tensorboard --logdir=graphs）

物体認識結果の処理

- NodeLookup Class の node_lookup.id_to_string(ID) によりラベル名を取得

ALTextToSpeechによる結果の発話

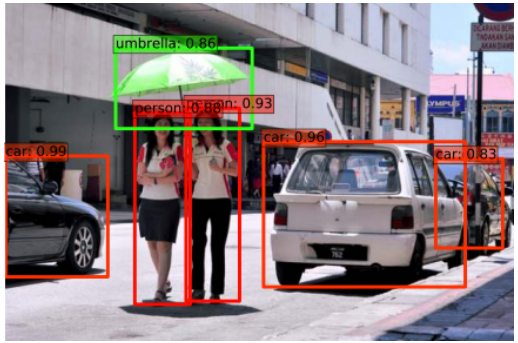
Keras+TensorFlowで
物体検出



Keras+TensorFlowで物体検出

- Keras+TensorFlowで学習済みの**物体検出モデル**を使って、
  Pepperのカメラ画像のリアルタイム解析結果を表示するプログラムを作ろう！

- アウトライン
  ① **Keras+TensorFlow**で学習済みのモデルの読み込み
  ② **PythonSDK**でALVideoDeviceによるPepperのカメラ情報のリモート取得
  ③ Deep Learningモデルによる物体検出の実行
  ④ 検出結果を数値から画像に書き込み
  ⑤ PC上に分析結果画像の表示 (→ 2. へ繰り返し)



[Liu+,2016,SSD: Single Shot MultiBox Detector]

objdetect_bypepper.py に学習済みモデルを対応

- 今回の使用モデル：「SSD (Single Shot MultiBox Detector)」
  - Pascal VOC 2007で学習された、画像から21種類の物体を検出するモデル
    - Github：https://github.com/rykov8/ssd_keras
    - 論文：https://arxiv.org/abs/1512.02325
  - 本WS資料では「ssd_keras/」にダウンロード済み

- objdetect_bypepper.py を一部変更（穴埋め形式）
  - ✓ L30 ：' ＊＊＊ ここを変更(1) ＊＊＊ ' → "background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"
  - ✓ L34 ：' ＊＊＊ ここを変更(2) ＊＊＊ ' → SSD300(input_shape, num_classes=num_classes)
  - ✓ L35 ：' ＊＊＊ ここを変更(3) ＊＊＊ ' → 'ssd_keras/weights_SSD300.hdf5'
  - 実行例：$ python objdetect_bypepper.py 192.168.100.1
  - 停止：ctrl-C

自分のPepperの
IPアドレス

objdetect_bypepper.py [1/3]

```
# --- coding: utf-8 ---
import sys
sys.path.append("ssd_keras")

import argparse
import numpy as np
import cv2

import keras
from keras.applications.imagenet_utils import preprocess_input
from keras.backend.tensorflow_backend import set_session
from keras.models import Model
from keras.preprocessing import image as imgprocess
from keras.utils.visualize_util import plot

from ssd import SSD300
from ssd_utils import BBoxUtility

from naoqi import ALProxy
```

必要ライブラリ読み込み
・ssd を読み込むため、PATHに ssd_keras フォルダを追加

```
# Pepper の IP アドレスを引数で取得
parser = argparse.ArgumentParser()
parser.add_argument('ip_ad', help='my IP address(ifconfig)')
args = parser.parse_args()
ip_ad = args.ip_ad
```

実行時引数からPepperのIPアドレスを取得、変数 ip_ad に保存
※実行例：python objdetect_bypepper.py 192.168.100.1

```
# 各種設定
input_shape = (300, 300, 3)
class_names = ["background", "aeroplane", "bicycle", "bird", "boat",
               "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
               "dog", "horse", "motorbike", "person", "pottedplant",
               "sheep", "sofa", "train", "tvmonitor"]; # SSD_kerasのクラスラベル名
num_classes = len(class_names)
```

入力する画像のサイズ、クラスラベル名、クラスラベルの数を設定

```
# 学習モデル読み込み
model = SSD300(input_shape, num_classes=num_classes)
model.load_weights('ssd_keras/weights_SSD300.hdf5')
bbox_util = BBoxUtility(num_classes)
plot(model, to_file='graphs/SSDmodel.png') # モデル描画
```

①

Keras+TensorFlow学習済みモデルの読み込み
「ssd_keras/weights_SSD300.hdf5」から学習済みパラメータを
「SSD300」で定義されたグラフに読み込み

※ graphs/SSDmodel.png にモデルのイメージを出力

objdetect_bypepper.py [2/3]

④

```
# バウンディングボックスの色設定
class_colors = []
for i in range(0, num_classes):
    # This can probably be written in a more elegant manner
    hue = 255*i/num_classes
    col = np.zeros((1,1,3)).astype("uint8")
    col[0][0][0] = hue
    col[0][0][1] = 128 # Saturation
    col[0][0][2] = 255 # Value
    cvcol = cv2.cvtColor(col, cv2.COLOR_HSV2BGR)
    col = (int(cvcol[0][0][0]), int(cvcol[0][0][1]), int(cvcol[0][0][2]))
    class_colors.append(col)
```

バウンディングボックス（BBOX）の色をラベルに割り当て

・HSV色空間で彩度・明度を固定して色相を均等に割り当て

④

```
# 出力をバウンディングボックスに変換して画像に描画する関数
def draw_bbox_from_results(image, results):
    if len(results) > 0 and len(results[0]) > 0:
        det_label = results[0][:, 0]
        det_conf = results[0][:, 1]
        det_xmin = results[0][:, 2]
        det_ymin = results[0][:, 3]
        det_xmax = results[0][:, 4]
        det_ymax = results[0][:, 5]

        top_indices = [i for i, conf in enumerate(det_conf) if conf >= 0.6] #0.6以上のものだけ取得
        top_conf = det_conf[top_indices]
        top_label_indices = det_label[top_indices].tolist()
        top_xmin = det_xmin[top_indices]
        top_ymin = det_ymin[top_indices]
        top_xmax = det_xmax[top_indices]
        top_ymax = det_ymax[top_indices]

        for i in range(top_conf.shape[0]):
            xmin = int(round(top_xmin[i] * image.shape[1]))
            ymin = int(round(top_ymin[i] * image.shape[0]))
            xmax = int(round(top_xmax[i] * image.shape[1]))
            ymax = int(round(top_ymax[i] * image.shape[0]))
            score = top_conf[i]
            class_num = int(top_label_indices[i])
            class_name = class_names[class_num]

            # 画像に BOX を描画
            cv2.rectangle(image, (xmin, ymin), (xmax, ymax),
                          class_colors[class_num], 2)
            text = class_name + " " + ('%.2f' % score)

            text_top = (xmin, ymin-10)
            text_bot = (xmin + 80, ymin + 5)
            text_pos = (xmin + 5, ymin)
            cv2.rectangle(image, text_top, text_bot, class_colors[class_num], -1)
            cv2.putText(image, text, text_pos, cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0,0,0), 1)
            print text

        return image
```

draw_bbox_from_resuls (元画像,結果) : 物体検出結果をBBOXとして描画する関数

Deep Learningからの出力結果 results は、以下のような内容

- results[sample][object, 0] : クラスラベルID
- results[sample][object, 1] : 確度
- results[sample][object, 2&3] : BBOXの左上の座標
- results[sample][object, 4&5] : BBOXの右下の座標

これを画像に実際に四角形として描画

objdetect_bypepper.py [3/3]

```
# NAOqi module の取得
videoDevice = ALProxy('ALVideoDevice', ip_ad, 9559)
videoDevice.unsubscribe('test')
```

②

```
# 上部カメラの監視
AL_kTopCamera = 0
AL_kQVGA = 2   # 0: kQQVGA (160x120), 1: kQVGA (320x240), 2: kVGA (640x480)
AL_kBGRColorSpace = 13   # 0: kYuv, 9: kYUV422, 10: kYUV, 11: kRGB, 12: kHSY, 13: kBGR
fps = 5   # 5, 10, 15, 30
nameID = 'test'
captureDevice = videoDevice.subscribeCamera(
    nameID, AL_kTopCamera, AL_kQVGA, AL_kBGRColorSpace, fps)

width = 640
height = 480
```

②

```
while True:
  result = videoDevice.getImageRemote(captureDevice)  # Pepperから画像取得
  image = np.zeros((height, width, 3), np.uint8)

  if result == None:
    print 'cannot capture.'
  elif result[6] == None:
    print 'no image data string.'
  else:
    values = map(ord, list(result[6]))
    # Pepperから得られる画像情報は一列なのでxy,RGBにマッピング
    i = 0
    for y in range(0, height):
      for x in range(0, width):
        image.itemset((y, x, 0), values[i + 0])
        image.itemset((y, x, 1), values[i + 1])
        image.itemset((y, x, 2), values[i + 2])
        i += 3

    image = cv2.resize(image, (300,300))
    #cv2.imwrite("input.jpg",frame)

    # kerasに与えるための画像の前処理
    input_image = [imgprocess.img_to_array(image)]
    input_image = preprocess_input(np.array(input_image))
```

②

```
    prediction = model.predict(input_image)  # 実行
    results = bbox_util.detection_out(prediction)  # BBOXとして出力を処理
    result_image = draw_bbox_from_results(image, results) # BBOXを画像に描画
```

③

④

```
    cv2.imshow("pepper-camera-ssd", image) # 画像表示
```

⑤

```
    k = cv2.waitKey(50);
    if k == ord('q'):  break;

videoDevice.unsubscribe(nameID)
```

繰り返し（q入力で終了）

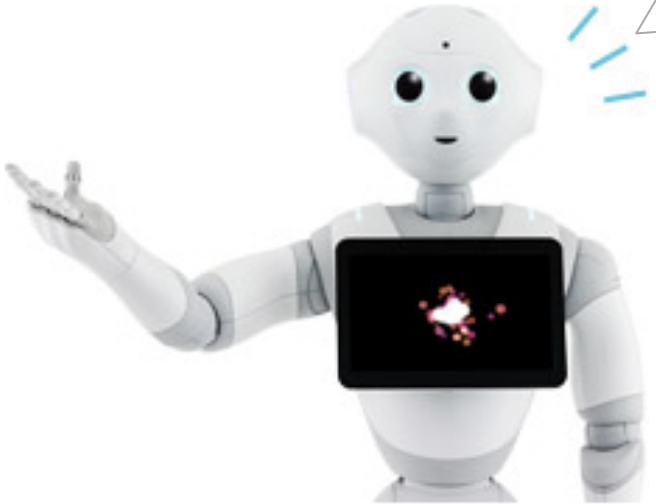
PythonSDKでAPI「ALVideoDevice」取得
引数：「API名」「PepperのIPアドレス」「ポート番号」

ALVideoDeviceによるカメラ情報取得のための設定
詳細は公式ドキュメントにて

Pepperから画像取得・処理
- result[6]に画像値が１次元配列で保存されているので、画像として配列し直す
- 学習されたモデルの入力300x300にリサイズ
- Keras で用意された前処理をかける

TensorFlow 学習モデルによる物体検出
物体検出結果の処理：関数 draw_bbox_from_resuls (元画像,結果) による結果の描画
物体検出結果を表すの画像の表示

おまけ



開発のための情報

Pepper デベロッパーポータル

「Pepper developer」で検索
https://developer.softbankrobotics.com/jp-ja

Pepperに関するデベロッパー向けの情報を集約したポータルサイト
・技術ドキュメント
・事例を共有するショーケース
・Pepper SDK for Android Studioのダウンロード
・最新ニュースの提供

開発のための情報

Pepper アトリエ秋葉原 with SoftBank

「アトリエ秋葉原 ブログ」で検索

・ペッパー開発に役立つ記事を見ることができる
・イベントの紹介とイベントのレポートが見ることができる
・tipsの項目から開発に便利なツールを手に入れることができる

開発のための情報

アトリエ秋葉原FBグループ

「アトリエ秋葉原　FB」で検索

・アトリエ秋葉原のFacebookグループです
・情報共有や質問ができます

開発のための情報

Qiita

「Qiita　pepper 」で検索

・エンジニアの情報交換サイト
・PepperタグでPepperに関する様々な技術情報がある

参考ページ

- TensorFlow，Keras
  - https://www.tensorflow.org/
  - https://keras.io/ja/
- pythonSDK(Macユーザー必見)
  - https://qiita.com/kyohara/items/ae0dd0f57413caa90674
- DeepLearningを活用した物体検出（SSD_Keras）を可能とするPepper
  - http://ai-coordinator.jp/pepper-deeplearning-ssd_keras
- SSD(Keras/TensorFlow)でディープラーニングによる動画の物体検出を行う
  - https://qiita.com/yampy/items/37c607fdf77a919cda5d
- R-CNNを「Tensorflow x Pepper」で実装する方法
  - http://ai-coordinator.jp/r-cnn-tensorflow-pepper
- NAOqi 2.5 ALVideoDevice API
  - http://doc.aldebaran.com/2-5/naoqi/vision/alvideodevice-api.html

おつかれさまでした！
これにてWS番外編は終わりになります。

タッチアンドトライでの質問もお待ちしています。