

組み込みでディープラーニング Cortex[®]-A9 ベアメタル実装

ARM

田中 隆治

アーム株式会社 シニア フィールドアプリケーションエンジニア

2017/07/28

©ARM 2017

Agenda

- 誰でもできる！ディープラーニング
- 組み込みでもできる？ディープラーニング
- ソフトウェア資産の保護

誰でもできる！
ディープラーニング

ディープラーニング開発環境

人気のある(※)ディープラーニングフレームワーク

(※)個人の見解を含みます

- Caffe
- Chainer
- Theano
- Tensorflow

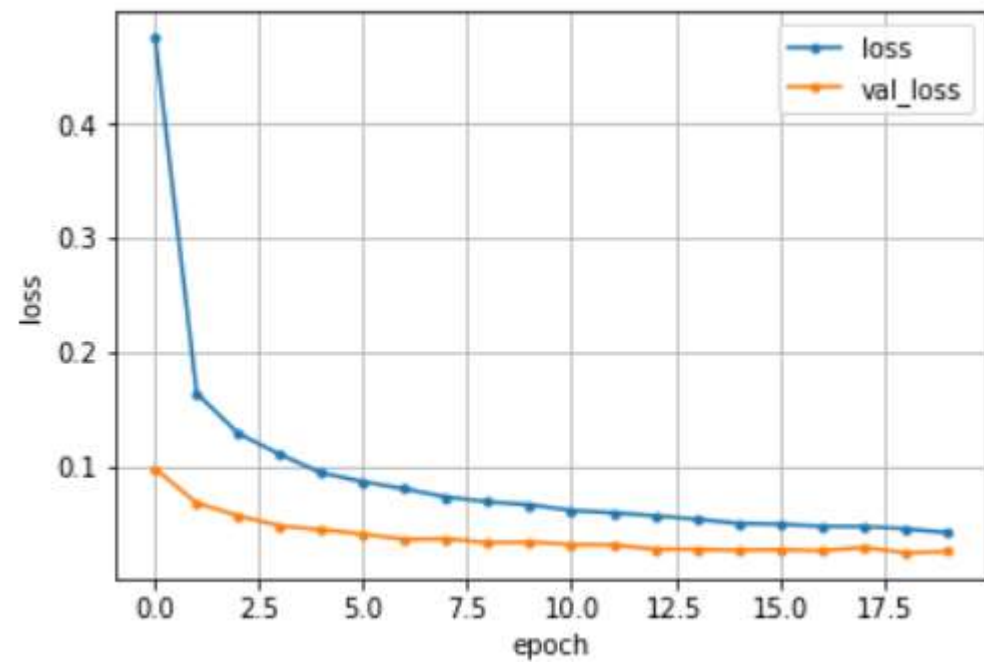
高度に抽象化されたラッパーライブラリ

- Keras (Theano, Tensorflowの上位ラッパーライブラリ)
- プログラミング言語としてPythonをサポート

Chainer®は、株式会社Preferred Networksの日本国およびその他の国における商標または登録商標です。
TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.
その他、記載されている会社名、製品名は、各社の登録商標または商標です。

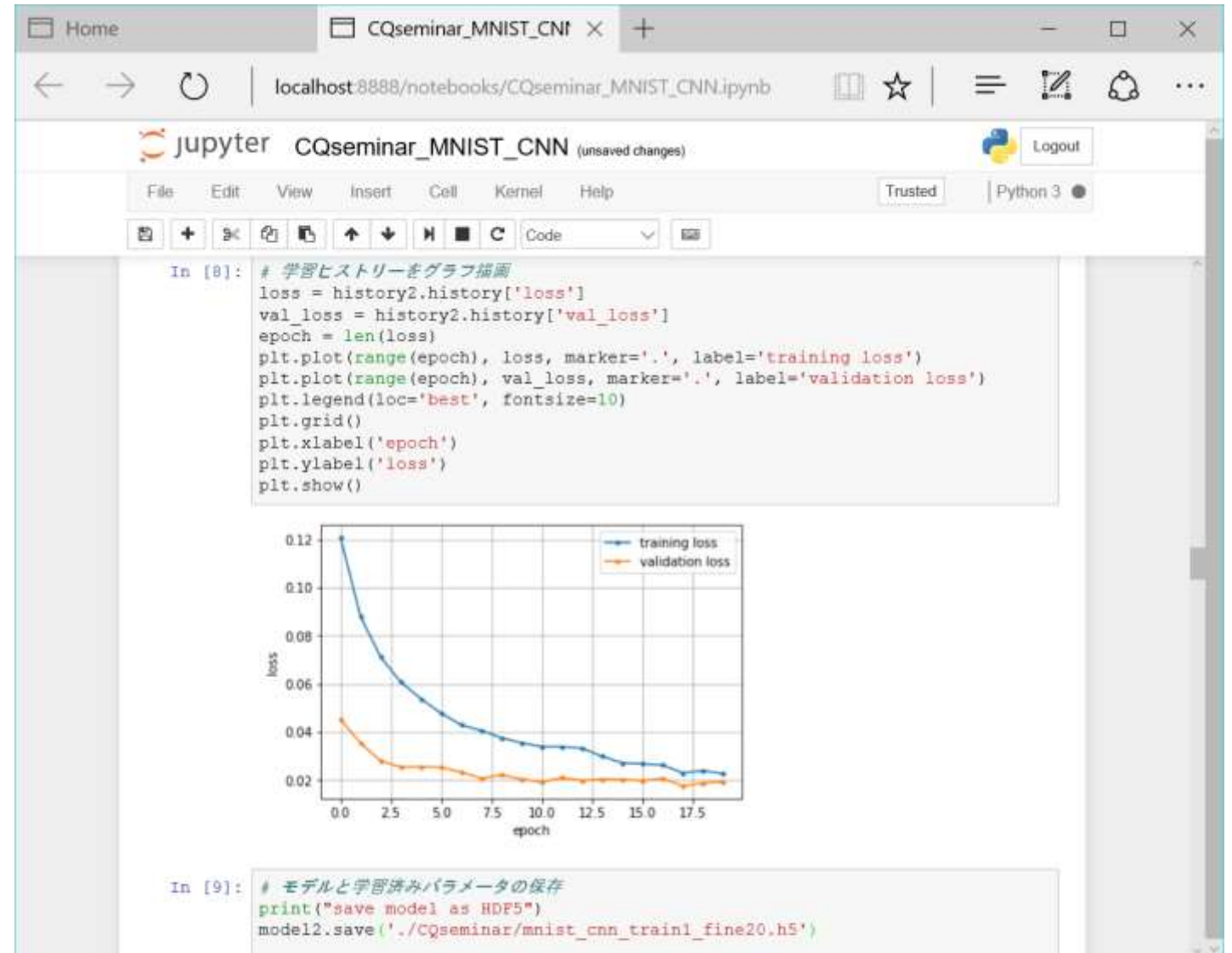
Python

- マルチプラットフォーム
- 動的型付け
- 豊富な外部ライブラリ
 - NumPy
 - 強力な配列の演算
 - Matplotlib
 - グラフや画像の描画
 - Jupyter notebook
 - ウェブブラウザ上でコーディング・実行

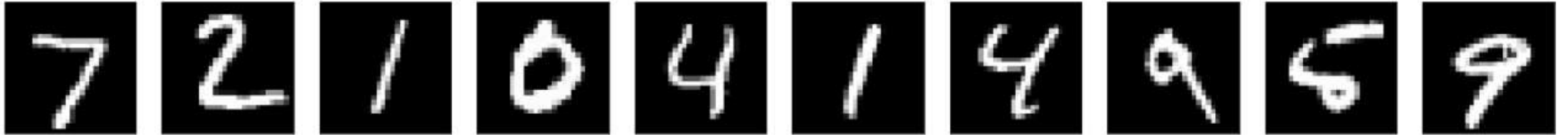


今回の開発環境

- Keras
- Tensorflow
- Jupyter Notebook
- Python



Hello World的なMNIST



MNISTデータセット

- 0～9の数字を手書きした画像と正解データのセット
- 学習用に60,000件、テスト用に10,000件と大規模
- 多くのディープラーニングフレームワークが Hello World 的なサンプルに利用

<http://yann.lecun.com/exdb/mnist>

Keras MNIST CNN

畳み込み層

5x5フィルター 16個

ReLU活性化関数

MAXプーリング層

2x2フィルター

畳み込み層

5x5フィルター 32個

ReLU活性化関数

MAXプーリング層

2x2フィルター

ドロップアウト

0.25

平滑化層

全結合層

ReLU活性化関数

ドロップアウト

0.5

全結合層

Softmax

```
# モデル設定
model = Sequential()
model.add(Conv2D(16, kernel_size=(5, 5),
                 activation='relu',
                 input_shape=input_shape)) # 畳み込み層 5x5x16 ReLU
model.add(MaxPooling2D(pool_size=(2, 2))) # MAXプーリング層 2x2
model.add(Conv2D(32, (5, 5), activation='relu')) # 畳み込み層 5x5x32 ReLU
model.add(MaxPooling2D(pool_size=(2, 2))) # MAXプーリング層 2x2
model.add(Dropout(0.25)) # ドロップアウト 0.25
model.add(Flatten()) # 平滑化層
model.add(Dense(128, activation='relu')) # 全結合層 ReLU
model.add(Dropout(0.5)) # ドロップアウト 0.5
model.add(Dense(num_classes, activation='softmax')) # 全結合層 Softmax
```

```
# Conv2Dのデフォルト値(https://keras.io/ja/layers/convolutional/)
# strides=(1, 1) : ストライド1
# padding='valid' : パディングしない(画像端の特徴は学習できないがMNISTでは問題ない)
```

```
# 学習設定
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              #optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

```
# 学習
model.fit(x_train, y_train,
        batch_size=batch_size, # 学習用データ
        epochs=epochs, # 勾配更新間隔
        verbose=1, # 学習回数
        validation_data=(x_test, y_test)) # プロGRESS表示
```

```
# 検証
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

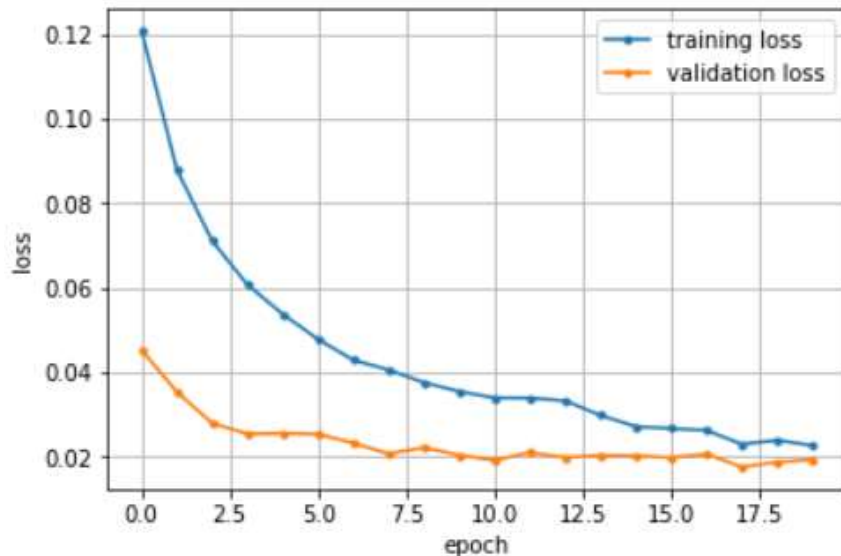
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
Epoch 1/1
60000/60000 [=====] - 36s - loss: 0.3959 - acc: 0.8749 - val_loss: 0.0688 - val_acc: 0.9772
Test loss: 0.068829282061
Test accuracy: 0.9772
```

1セットの学習で
テストデータ正解率: 0.9768

Keras MNIST CNN

- 学習済みモデルのセーブ・ロード
- 20セット追加学習
 - 合計21セット学習済み



```
from keras.models import load_model

# コンパイル済み、学習済みモデルをロード
model2 = load_model('./CQseminar/mnist_cnn_train1.h5')

# 再学習
history2 = model2.fit(x_train, y_train,
                      batch_size=batch_size,
                      epochs=20,           # 20回追加学習
                      verbose=1,
                      validation_data=(x_test, y_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/20
60000/60000 [=====] - 35s - loss: 0.1193 - acc: 0.9639 - val_loss: 0.0488 - val_acc: 0.9840

Epoch 2/20
60000/60000 [=====] - 36s - loss: 0.0872 - acc: 0.9740 - val_loss: 0.0354 - val_acc: 0.9878

⋮

Epoch 17/20
60000/60000 [=====] - 40s - loss: 0.0263 - acc: 0.9916 - val_loss: 0.0208 - val_acc: 0.9927

Epoch 18/20
60000/60000 [=====] - 42s - loss: 0.0231 - acc: 0.9926 - val_loss: 0.0177 - val_acc: 0.9946

Epoch 19/20
60000/60000 [=====] - 42s - loss: 0.0240 - acc: 0.9927 - val_loss: 0.0188 - val_acc: 0.9945

Epoch 20/20
60000/60000 [=====] - 40s - loss: 0.0228 - acc: 0.9925 - val_loss: 0.0194 - val_acc: 0.9936

20セット学習後
テストデータ正解率：0.9936

組み込みでもできる？
ディープラーニング

組み込み開発のハードル

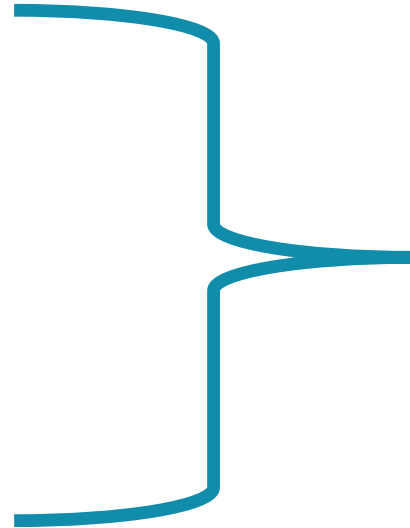
- ライブラリで抽象化された処理の理解



CQ出版 インターフェース
2017年8月号

- 学習済みモデルの移植

- 大規模GPU vs 組み込みプロセッサ



ARM DS-5
ソフトウェア
開発環境

学習済みモデル

- 学習済みパラメータの構成を解析
 - パラメータArray（重み、バイアス）の解析
CQseminar_MNIST_CNN.ipynb
 - 各層の出力（次層の入力）Arrayの解析
CQseminar_MNIST_CNN_2.ipynb

```
In [57]: # 各層の構造とパラメータ (名前、バイアス) を表示
for layer in model2.layers:
    g=layer.get_config()
    h=layer.get_weights()
    print('*** layer config ***')
    print(g)
    print('*** layer weights ***')
    print(h)
```

```
*** layer config ***
{'kernel_constraint': None, 'name': 'conv2d_5', 'activation': 'relu', 'use_bias': True, 'filters': 16, 'padding':
'valid', 'dtype': 'float32', 'trainable': True, 'kernel_size': (5, 5), 'dilation_rate': (1, 1), 'bias_constraint':
None, 'strides': (1, 1), 'batch_input_shape': (None, 28, 28, 1), 'bias_regularizer': None, 'bias_initializer': ('co
nfig': {}, 'class_name': 'Zeros'), 'activity_regularizer': None, 'kernel_initializer': {'config': {'scale': 1.0, 's
eed': None, 'distribution': 'uniform', 'mode': 'fan_avg'}, 'class_name': 'VarianceScaling'}, 'kernel_regularizer':
None, 'data_format': 'channels_last'}
*** layer weights ***
[[array([[[[ 9.17235091e-02,  5.82217146e-03, -2.12860093e-01,
  2.6158871e-02,  1.00777401e-02, -3.54313813e-02,
  4.35772277e-02, -4.21801358e-02,  7.40430206e-02,
  9.46836844e-02,  8.33052304e-03,  3.62434834e-02,
  1.24821335e-01,  3.36471051e-01,  6.69756881e-02,
  1.2265347e-01]],
  [[ 5.96388653e-02,  3.20913605e-02, -1.07701316e-01,
  6.31961823e-02, -9.35485065e-02, -4.71841171e-02,
  1.28566772e-01, -3.72202806e-02,  1.18012607e-01,
  8.36518407e-02,  5.34471348e-02, -1.02290995e-01,
  2.38167028e-01,  1.40018373e-01,  1.87823117e-02,
```

```
# 層間パラメータをJSONで書き出し
# 最大プーリング層(Channel:16, Figure Width:12, Figure Height:12)
# 畳み込み層2(Activation:ReLU, Channel:32, Filter Width:8, Filter Heights:8)
# 重み(Array[5][5][16][32]), バイアス(Array[32])
layidx = 2
params_list = model3.layers[layidx].get_weights()
weights_array = params_list[0]
biases_array = params_list[1]
dict = {}
dict['weights'] = weights_array.tolist()
dict['biases'] = biases_array.tolist()
file_path = filename % layidx
json.dump(dict,
          codecs.open(file_path, 'w', encoding='utf-8'),
          separators=(',', ':'),
          sort_keys=False,
          indent=4)
```

```
mnist_cnn_train121_params_layer0.json {
  1 {
  2   "biases": [
  3     -0.2305668592453003,
  4     -0.2242465317249298,
  5     -0.3020424246788025,
  6     -0.17507998645305634,
  7     -0.1673714518547058,
  8     -0.16205695271492004,
  9     -0.06616398692131042,
10    -0.02443808689713478,
11    -0.34706518054008484,
12    -0.46608760952949524,
13    0.10299401730298996,
14    -0.004913313779979944,
15    -0.1697794497013092,
16    -0.21209420263767242,
17    -0.05130063742399216,
18    -0.41009610891342163
```

ARM DS-5 ソフトウェア開発環境



DS-5 Development Studio



ARM コンパイラ

ARMプロセッサ向け最適化、
組み込み向けライブラリ、
ADASなど高度なシステムで
実績



Streamline Analyzer

CPU, GPU, OS, アプリケーション
などシステム全体のパフォーマンス
を解析



DS-5 デバッガ

Jythonデバッガスクリプトによる
高度な開発支援、
TrustZoneデバッグ対応

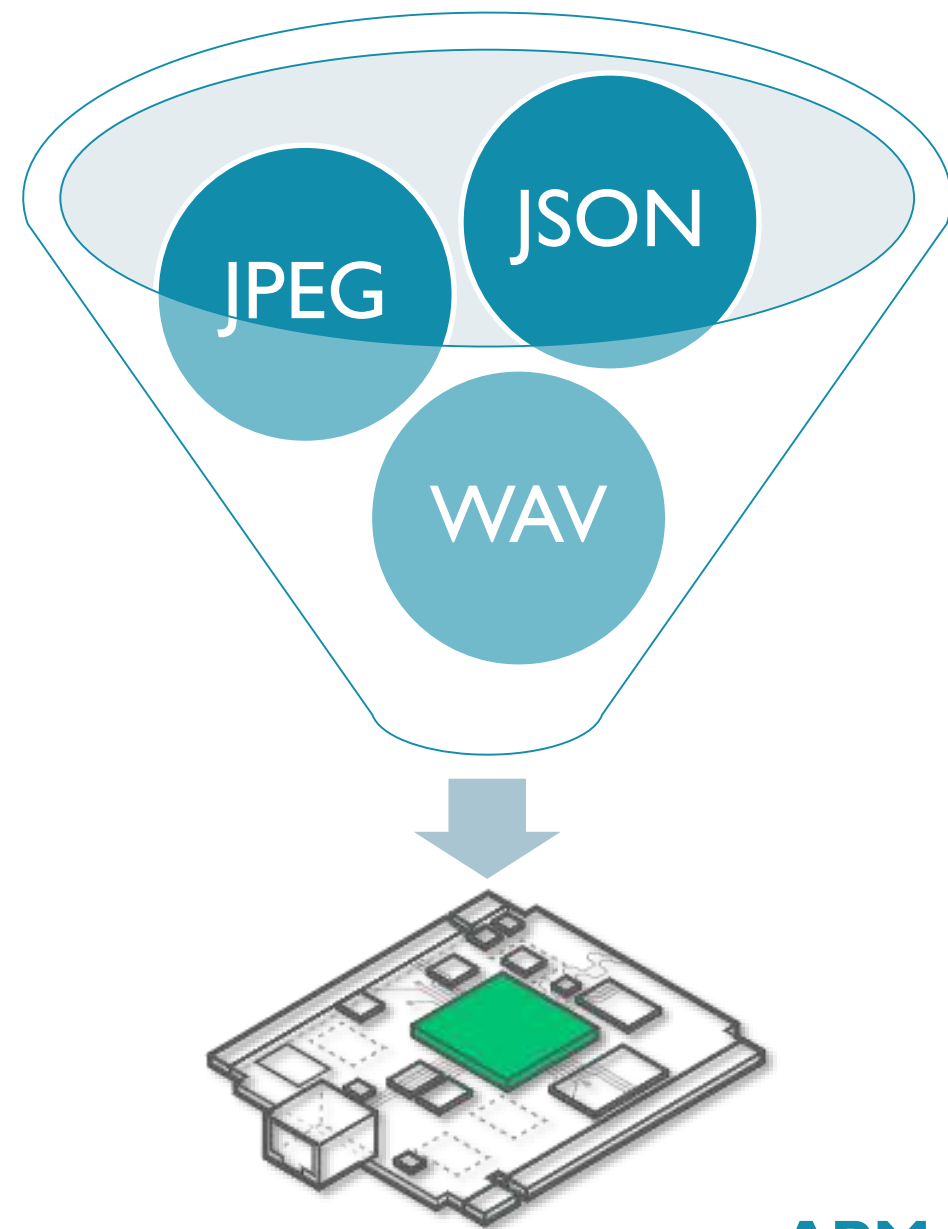


DS-5 IDE

EclipseベースIDEの採用による
PyDevなどの多彩なプラグイン

DS-5 Jythonデバッガスクリプト

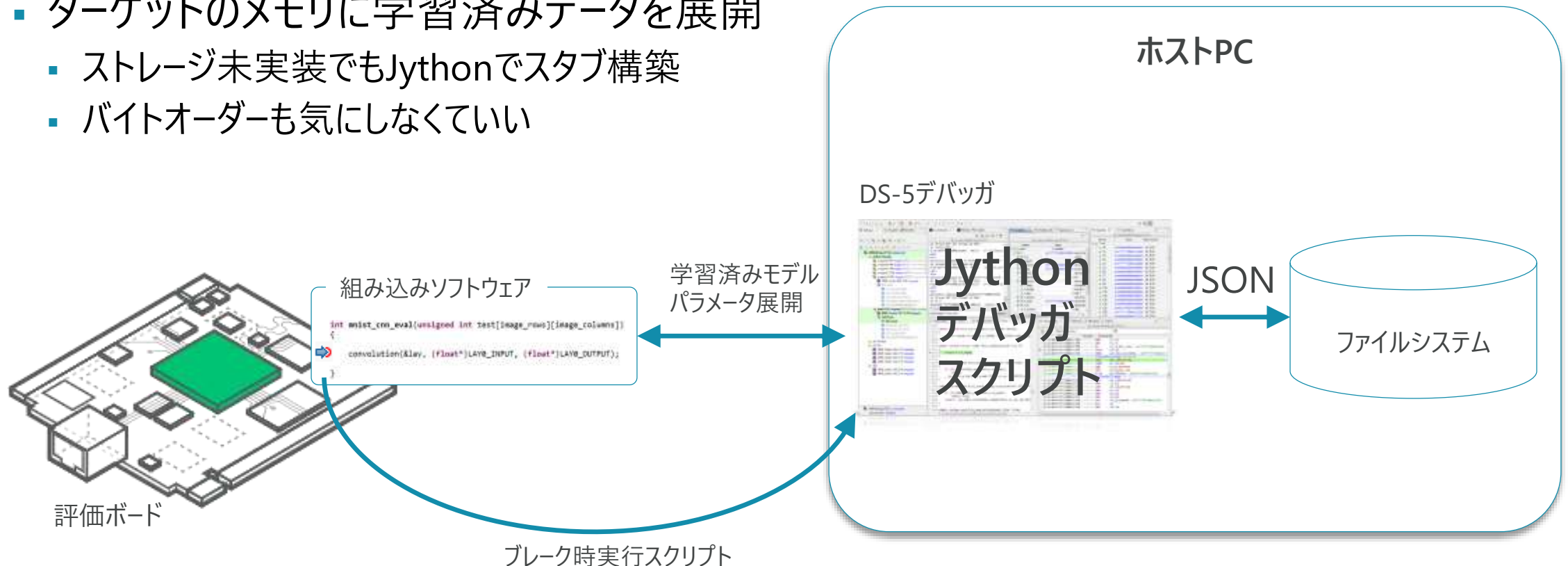
- 多様なデータを組み込みデバイスのメモリに展開
 - ニューラルネットワークのパラメータ
 - センサーデータ
 - 画像, 音声データ
- ペリフェラルの完成を待たずにプロトタイピング
 - デモではニューラルネットワークのパラメータとテスト画像のメモリ展開に利用しています。



スタブ構築：ストレージ

ターゲットデバイスを抽象化するDS-5デバッガのデバッグスクリプト

- JythonデバッグスクリプトでJSONデータを簡単読み込み
- ターゲットのメモリに学習済みデータを展開
 - ストレージ未実装でもJythonでスタブ構築
 - バイトオーダーも気にしなくていい



スタブ構築：ストレージ

- 学習済みパラメータをJSONから簡単読み込み
- デバッガコマンドでターゲットデバイスのメモリに格納

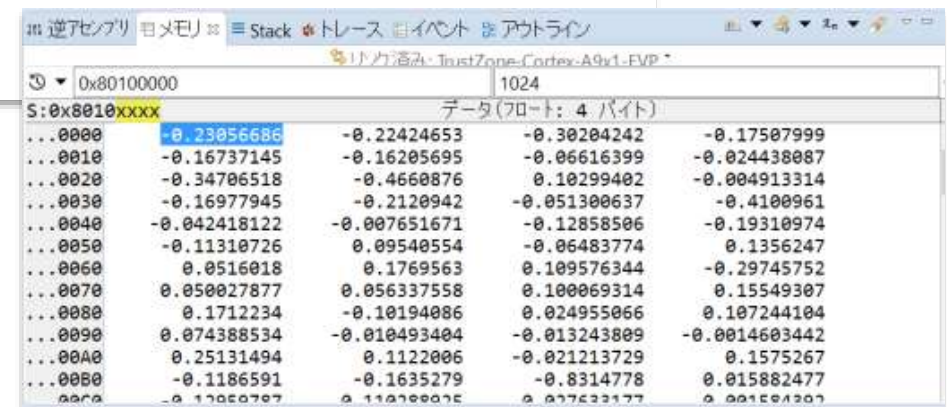
```
def loadParams(filepath):  
  
    fp = open(filepath, 'r')  
    jsonObj = json.load(fp)  
    fp.close()  
  
    return jsonObj
```

cnn_import.py

```
def storeParams(ec, store_adr, params, biases_max, weights_max0, weights_max1, weights_max2, weights_max3):  
  
    adr = store_adr  
  
    # biases  
    start_adr = adr  
    for i in range(0, biases_max):  
        dscmd = 'memory set_typed S:0x%08x (float) (float)(%.18f)' % (adr, params['biases'][i])  
        ec.executeDSCommand(dscmd)  
        adr += 0x4  
    print 'biases S:0x%08x - S:0x%08x' % (start_adr, adr)
```

- ターゲット上のメモリイメージをファイルにエクスポート
 - 2回目以降はバイナリ読み込み

restore "ds5_params.bin" binary S:0x80300000

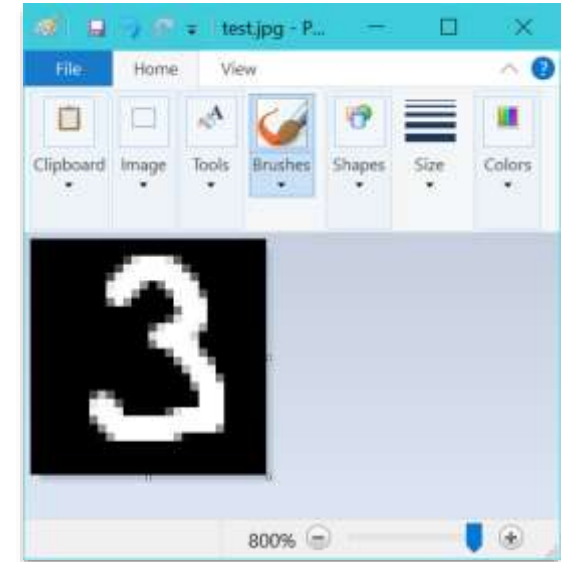


Address	Value (Hex)	Value (Dec)	Value (Hex)	Value (Dec)
0x80100000	0x23056686	-0.22424653	0x30204242	-0.17507999
0x80100001	0x16737145	-0.16205695	0x06616399	-0.024438087
0x80100002	0x34706518	-0.4660876	0x10299402	-0.004913314
0x80100003	0x16977945	-0.2120942	0x051300637	-0.4100961
0x80100004	0x042418122	-0.007651671	0x12858506	-0.19310974
0x80100005	0x11310726	0.09540554	0x06483774	0.1356247
0x80100006	0x0516018	0.1769563	0x109576344	-0.29745752
0x80100007	0x050027877	0.056337558	0x100069314	0.15549307
0x80100008	0x1712234	-0.10194086	0x024955066	0.107244104
0x80100009	0x074388534	-0.010493404	0x013243809	-0.0014603442
0x8010000A	0x25131494	0.1122006	0x021213729	0.1575267
0x8010000B	0x1186591	-0.1635279	0x0314778	0.015882477
0x8010000C	0x17050707	0.11070075	0x077633177	0.001501307

スタブ構築：LCD/タッチパネル

- ペイントツールで作成したJPEGファイルを簡単読み込み
- デバッガコマンドでターゲットデバイスのメモリに格納
image_import.py

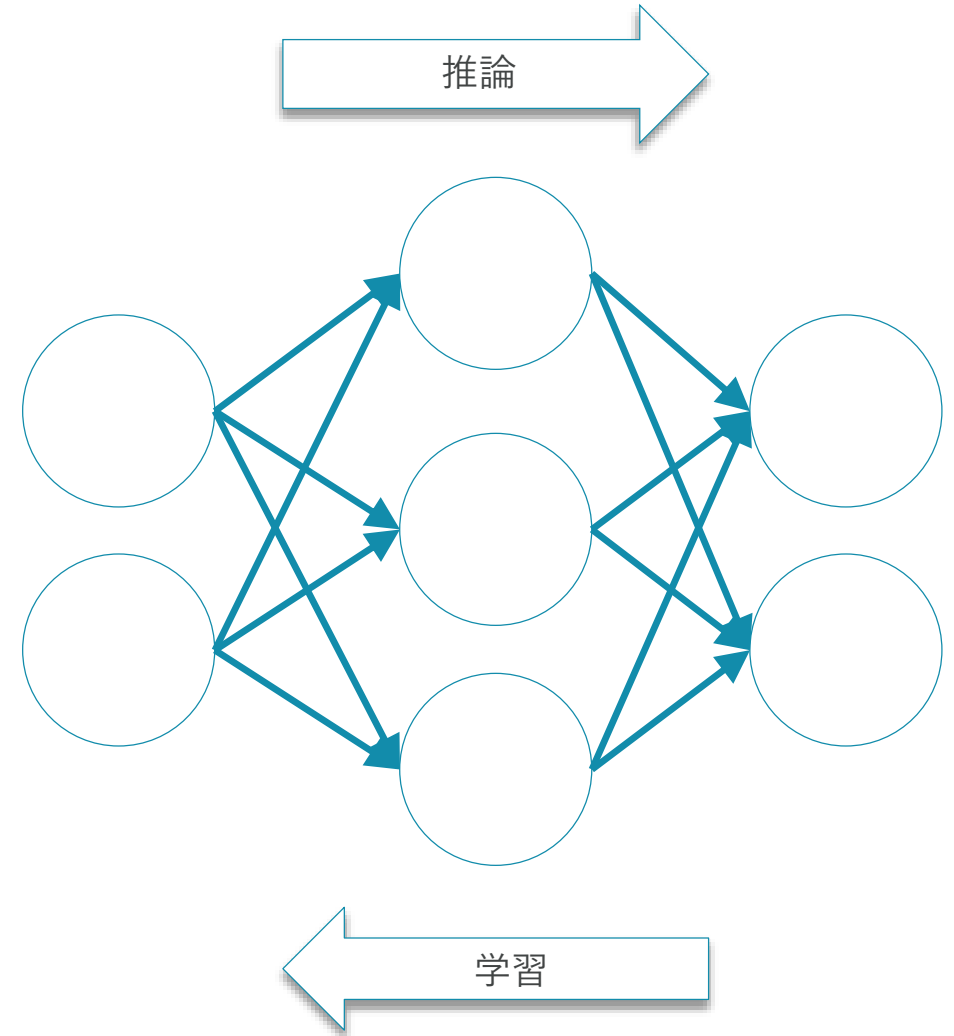
```
def main():  
    # Obtain the first execution context  
    debugger = Debugger()  
    ec = debugger.getCurrentExecutionContext()  
  
    # Assumed the image file is located on the project folder  
    imgDir = getImageDir(ec)  
  
    image = loadImage(imgDir + '/scripts/test.jpg')  
  
    #--- store image ---  
    adr = 0x80370000 # TESTDATA 0x80370000 to 0x80370C40 (size 0xC40)  
  
    adr = storeImage(ec, adr, image)
```



データ (フーント: 4 バイト)			
-0.23056686	-0.22424653	-0.30204242	-0.17507999
-0.16737145	-0.16205695	-0.06616399	-0.024438087
-0.34706518	-0.4660876	0.10299402	-0.004913314
-0.16977945	-0.2120942	-0.051300637	-0.4100961
-0.042418122	-0.007651671	-0.12858506	-0.19310974
-0.11310726	0.09540554	-0.06483774	0.1356247
0.0516018	0.1769563	0.109576344	-0.29745752
0.050027877	0.056337558	0.100069314	0.15549307
0.1712234	-0.10194086	0.024955066	0.107244104
0.074388534	-0.010493404	-0.013243809	-0.0014603442
0.25131494	0.1122006	-0.021213729	0.1575267
-0.1186591	-0.1635279	-0.8314778	0.015882477
-0.17050707	0.11070075	0.07633177	0.001501307

割り切った運用

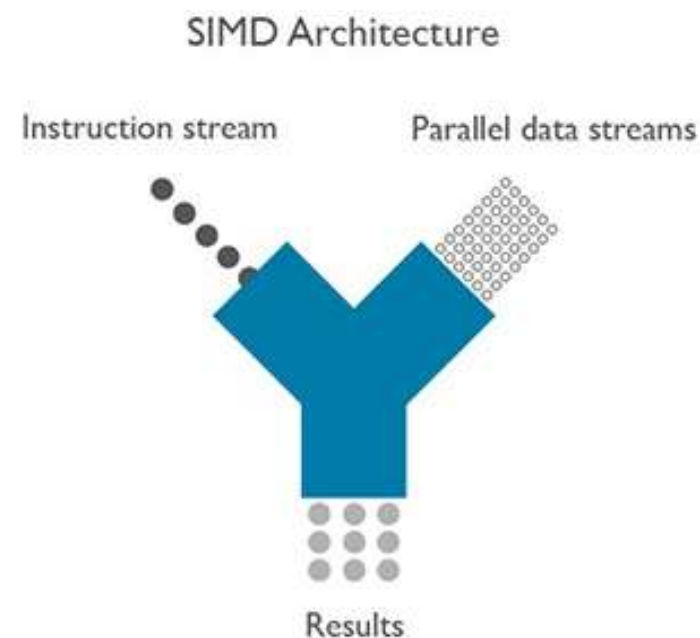
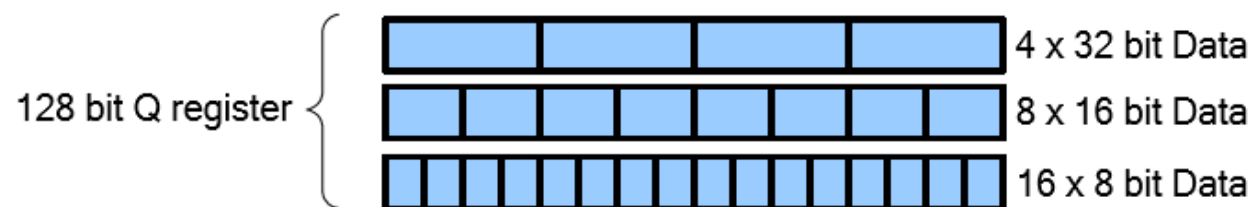
- 入力データの推論のみ
- 新しいデータの学習は行わない
- 必要な処理
 - 前処理(入力正規化)
 - 畳み込み層
 - 最大プーリング層
 - 全結合層
 - 後処理(最大スコアの要素取得)
 - ReLU活性化関数
- 実装しない処理
 - Adadelta最適化
 - Dropout
 - 平滑化層
 - Softmax



デモ

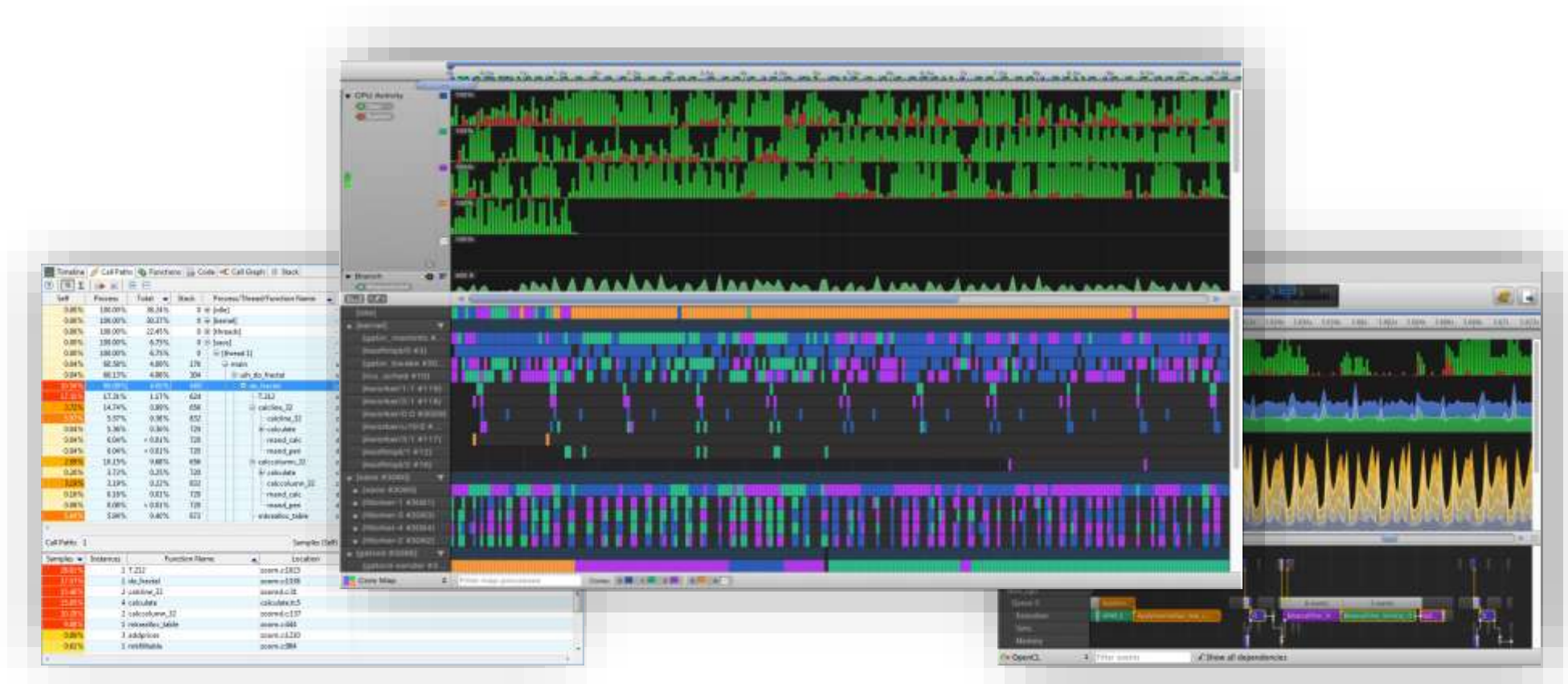
ARM Compiler

- NEON 自動ベクトル化コンパイラ
 - C言語のループを自動で並列化
 - 一度の命令で複数データを同時に処理



推論処理の最適化

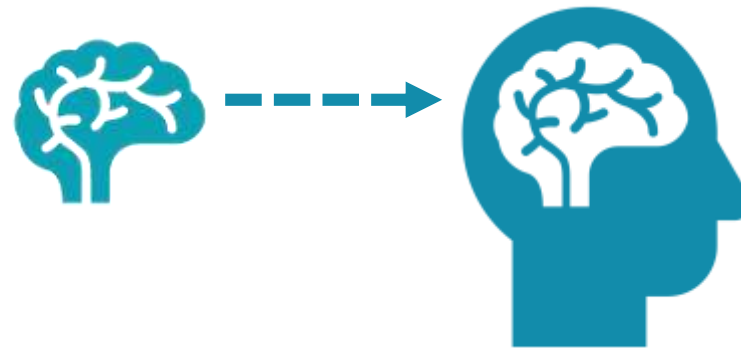
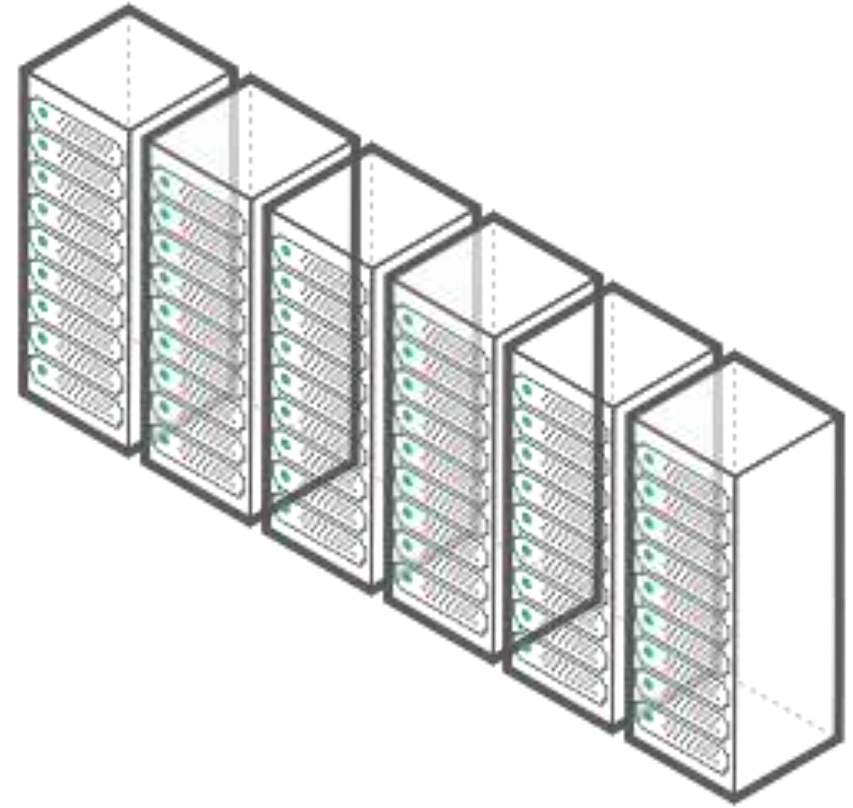
- Streamline ベアメタル
- ソフトウェアのホットスポットを解析



ソフトウェア資産の保護 TrustZone

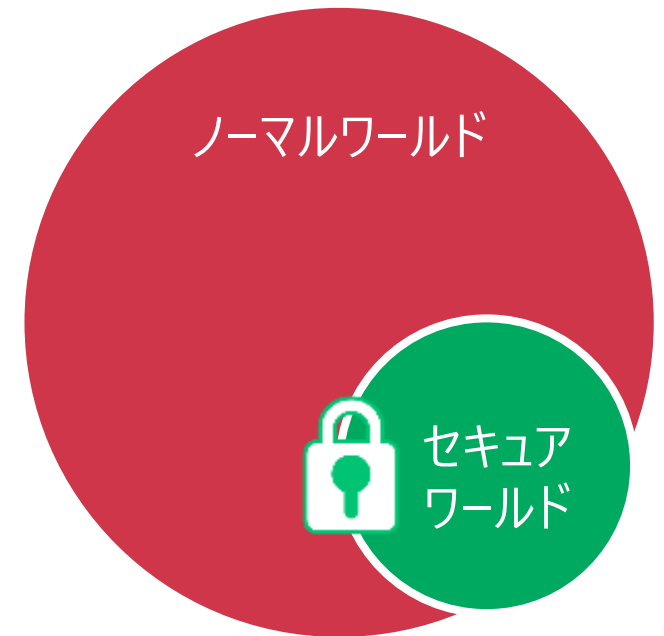
保護すべき学習済みモデル

- 学習にかかるコスト
 - 大量のデータを集めるコスト
 - 教師データを付与する分類作業コスト
 - 学習のためのコンピューティングリソースコスト
- 学習済みモデルの転移学習
 - 学習済みモデルをベースに再学習
 - 学習コストを大幅に削減
 - 盗用リスクも・・・



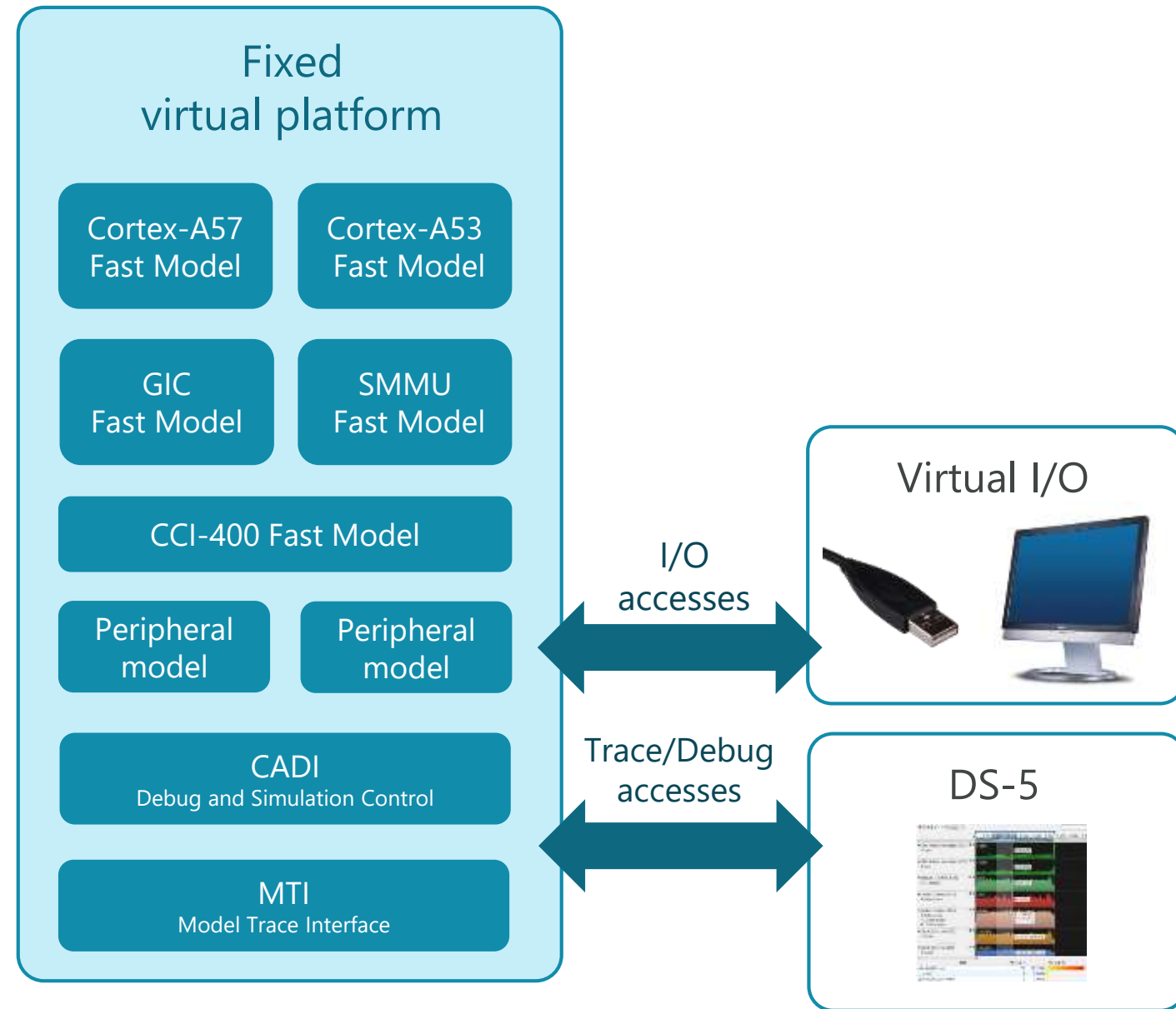
ARM® TrustZone®によるデータの保護

- TrustZone
 - Cortex-Aなどのアプリケーションプロセッサ向けセキュリティ技術
 - スマートフォンのDRM管理などに利用
- TrustZone for ARMv8-M
 - Cortex-M23およびCortex-M33向けセキュリティ技術
 - IoTへの応用が期待される



TrustZone

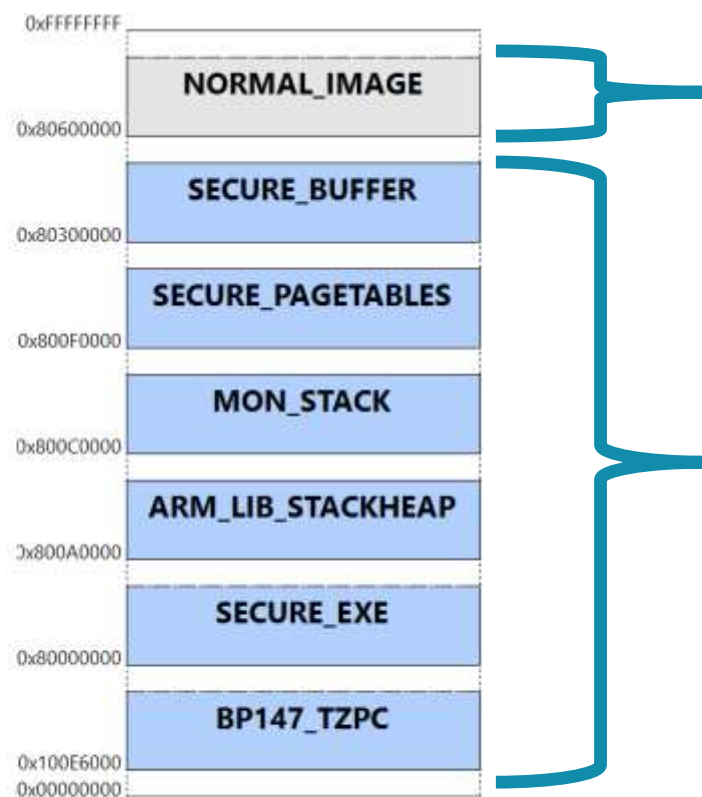
- FVP (Fixed Virtual Platforms)
- ARM自社製シミュレーションモデル
 - 非公開のRTL検証パターンでモデルを検証済み
- ホストPCのみでARM向け組み込みソフトウェア開発が可能
- TrustZoneシミュレーション可能



TrustZone セキュアワールド メモリマップ



ロード領域



実行領域

推論結果を利用するアプリケーションは
ノーマルワールドに配置

学習済みモデル・パラメータおよび
推論処理をセキュアワールドに配置

まとめ

- 敷居が下がったディープラーニング
- DS-5によるスタブ構築でプロトタイピング
- TrustZoneで資産保護
- サンプルコード
https://github.com/RyujiTanaka/ds5_keras_mnist

ARM

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

Copyright © 2017 ARM Limited

©ARM 2017