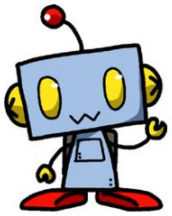
PASONA TECH

組込み向けDeep Learning フレームワーク コキュートスの紹介

HUMANWARE® changes the world



株式会社パソナテック
西日本支社
夏谷

2017/8/25

アジェンダ

PASONA TECH

- 自己紹介
- 業界動向
- 開発の経緯
- コキュートスの紹介
  - コキュートスとは
  - Kerasからの情報取得
  - Numpyフォーマット
  - データの型を変更する
  - 固定小数点対応
  - Kerasの標準でない機能について
- パソナテックのサービス

自己紹介

自己紹介

- 夏谷実
- 株式会社パソナテック
  - 株式会社パソナテック
  - エンジニアリング事業部
  - エンジニアマネージメントグループ
- TFUG KANSAI
- 最近は半導体関連の仕事が多い
- FPGAが好き
- プログラミングも好き
- Deep Learningも好き

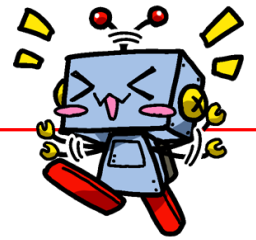
パソナテック応援キャラクター てくのたん

自由な働き方を模索する、
パソナテックのインターン生
Twitterを中心に情報発信中！

Twitter：@techno_tan

コミックマーケット92企業ブース出展や、
3Dコンテンツの配信を計画中★

10月にGMOインターネット様との共催技術系カンファレンスイベント”けーすた文化祭(仮)”を開催予定！



自己紹介

PASONA TECH

Hatena::Diary 日記 検索

ブログトップ 記事一覧 記事を書く 管理 ログアウト ヘルプ

ぱたへね

<前の5日分

2017-06-18 編集

[keras][コキュートス]VGG-YOLOを作りたい その1 データの転移

Tiny-YOLOがどうやっても普通の固定小数点化では精度がでない。もうひと工夫いるんだと思います。精度を上げるのに固定小数点化をいろいろ試すのではなく、CNNの部分をVGG16にしてみようと思いました。VGG16にすることで、バッチノーマライゼーションと、Leaky ReLUがなくなるので固定小数点にしたときの精度はコントロールしやすくなります。(多分)

最初の一歩として、Kerasのexampleに含まれているVGG16学習データを、自分のNNに移してみました。最終段の形が違うためそのままでは上手く行かず、結局レイヤー単位で、get_weightsとset_weightsを繰り返しました。多分、もっと良い方法があるはずです。

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, InputLayer, Dropout

# 入力サイズ等はここを変更
```

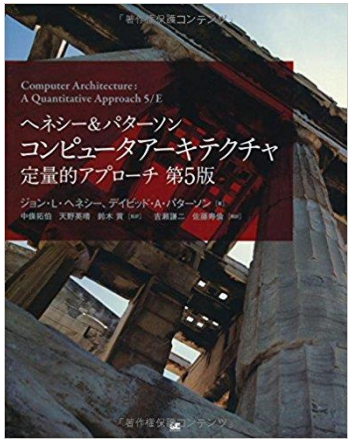
プロフィール

natsutan

YUKI.N

カテゴリー

- パタヘネ
- mips
- x86
- ARM
- sh
- sparc
- binary
- 目次
- FPGA
- book



自己紹介

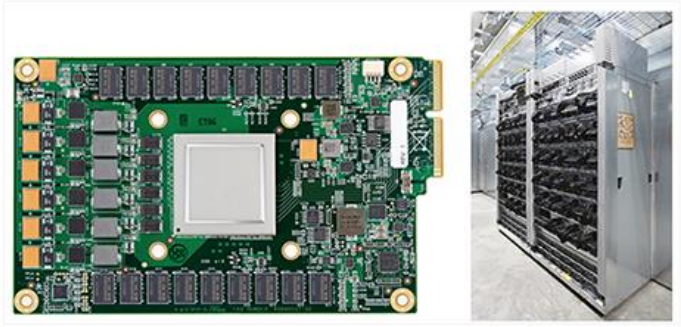
PASONA TECH

Googleのオフィシャルブログに名前載りました

An in-depth look at Google's first Tensor Processing Unit (TPU)

Friday, May 12, 2017

By Kaz Sato, Staff Developer Advocate, Google Cloud; Cliff Young, Software Engineer, Google Brain; and David Patterson, Distinguished Engineer, Google Brain



Google's first Tensor Processing Unit (TPU) on a printed circuit board (left); TPUs deployed in a Google datacenter (right)

Acknowledgement

Thanks to Zak Stone, Brennan Saeta, Minoru Natsutani and Alexandra Barrett for their insight and feedback on earlier drafts of this article.

https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

業界動向

エッジでのDeep Learning

PASONA TECH

Smartphone
・2017年1月　Snapdragonに実装されているDSP HexagonでGoogLeNetが動作。
・2017年3月　Google TensorFlow XLAの発表

組込向け
・2017年4月　ルネサスエレクトロニクスがe-AIを発表
・2017年6月　ソニーがNeural Network Librariesを発表
・2017年6月　マイクロソフトがEmbedded Learning Libraryを発表(Arduino対応)
・2017年7月　GoogleがTensorFlow Liteを発表(Ras Pi対応)
・2017年7月　XilinxがreVisionリリース

国内でも、Preferred Networks, Leap Mind, ディープインサイト等ベンチャー企業の動きも活発。業界的には、組込機器（エッジ）でのDeep Learningが注目を浴びている。

PASONA TECH

2017/3/31公正取引委員会　**データと競争政策に関する検討会　松尾先生**

日本なりのプラットフォーム戦略

- DLの技術はコモディティ化する。
  - 競争力をもつのは、データとハードウェア。
  - 早くDLの技術を取り入れてしまえばよい。
  - DLの技術とハードウェアのすりあわせになった瞬間、日本企業が再度、力を取り戻せる。
- 欧米のスタートアップ（とDL研究者）は、意外なほどハードウェアに対する抵抗感がある
  - そもそも、産業用ロボットの導入台数は日本が（ほぼ）トップ
  - また、ロボットに対する社会的抵抗感もある。
  - 米国は雇用を守らないといけない。日本は人手が足りない。
- ものが関連しないプラットフォームは無理
  - 英語圏でやったほうが絶対に強い。
  - 広告費規模でも10倍、ECの規模でも3倍以上

日本でもDLの専門家はハードウェアを知らない

https://www.google.co.jp/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwic3O6Czp7VAhWDe7wKHTtpBvMQFggnMAA&url=http%3A%2F%2Fwww.jftc.go.jp%2Fcprc%2Fconference%2Findex.files%2F170606data01.pdf&usg=AFQjCNE22edhkjcl-q3mswvN4vfgduR52w

現場の話は後で

開発の経緯

開発の経緯

PASONA TECH

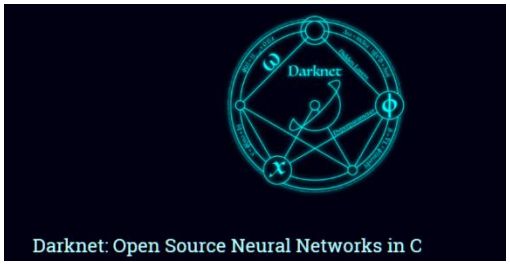
NPS-Verilog → NPU → コキュートス

データフローを定義したらFPGAで動くVerilogを作ろう

→そもそも機械学習やDeep Learningが分かってなく、挫折。

量子化されたデータを扱う専用プロセッサを作ろう。

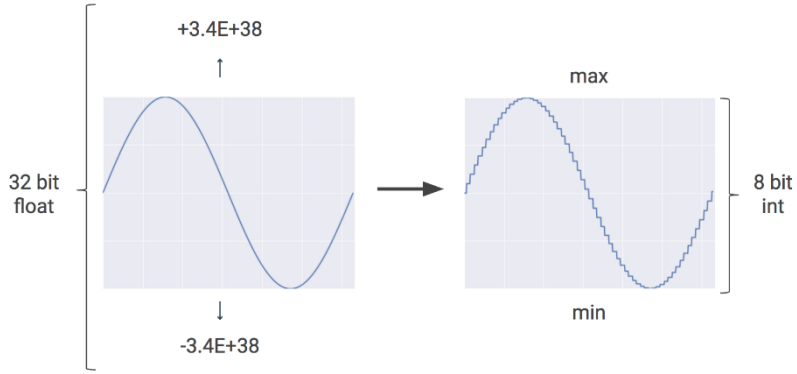
Cのリファレンスモデルがいるよね。



Darknetの影響は受けている

NPUのプロジェクトから、C言語生成ツールのみスピンアウト

Googleが使っている量子化の話（脱線）



浮動少数点数を、8bitに量子化する。データに対して、maxとminを紐付けして、その中を256等分する。

https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

量子化された加算

$C = A + B$とした時の、量子化された値A_{qt}, B_{qt}, C_{qt}の関係式

$$gain = \frac{(b_{max} - b_{min})}{(a_{max} - a_{min})}$$

$$c_{min} = a_{min} + b_{min}$$
$$c_{max} = a_{max} + b_{max}$$

Cのmaxとminは、A,Bそれぞれのmax, minの和

$$q_{param} = \frac{a_{max} - a_{min}}{c_{max} - c_{min}}$$

$$C_{qt} = (B_{qt} \times gain + A_{qt}) \times q_{param}$$

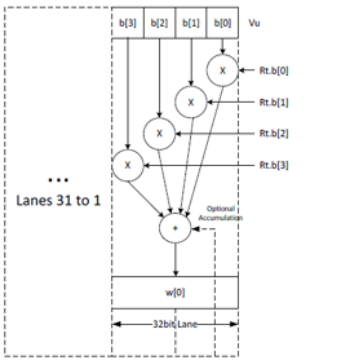
ここには負の数がでてこない。地味にVerilog実装が楽

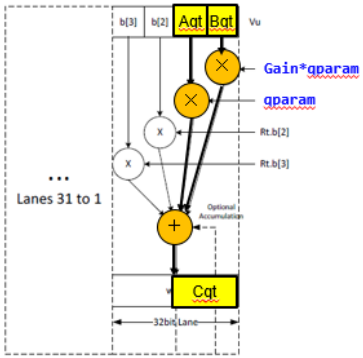
Bqtのスケールを Aqtに合わせて加算

加算結果をCqtのスケールに合わせる

量子化された加算

$$C_{qt} = (B_{qt} \times gain + A_{qt}) \times q_{param}$$





1命令で量子化された加算を実行可能

開発現場の苦悩

PASONA TECH

①ツールの問題
- 価格
- 使いたいNNがサポートされていない
- ブラックボックス、自分たちでカスタマイズできない
- モデルの圧縮等の最新手法がサポートされていない

②ボードの問題
- ツールが使いたいCPUをサポートされていない
- ファイルシステムやメモリ管理が必須。
- センサーが特殊

③人材の問題
- Deep Learning, Python, 組込みC, ボードに精通したメンバーがいない
- 研究所とのコミュニケーションギャップ
- 外部に出すと自社にノウハウがたまらない
- DLがわからない社内の組込み人材を有効活用したい

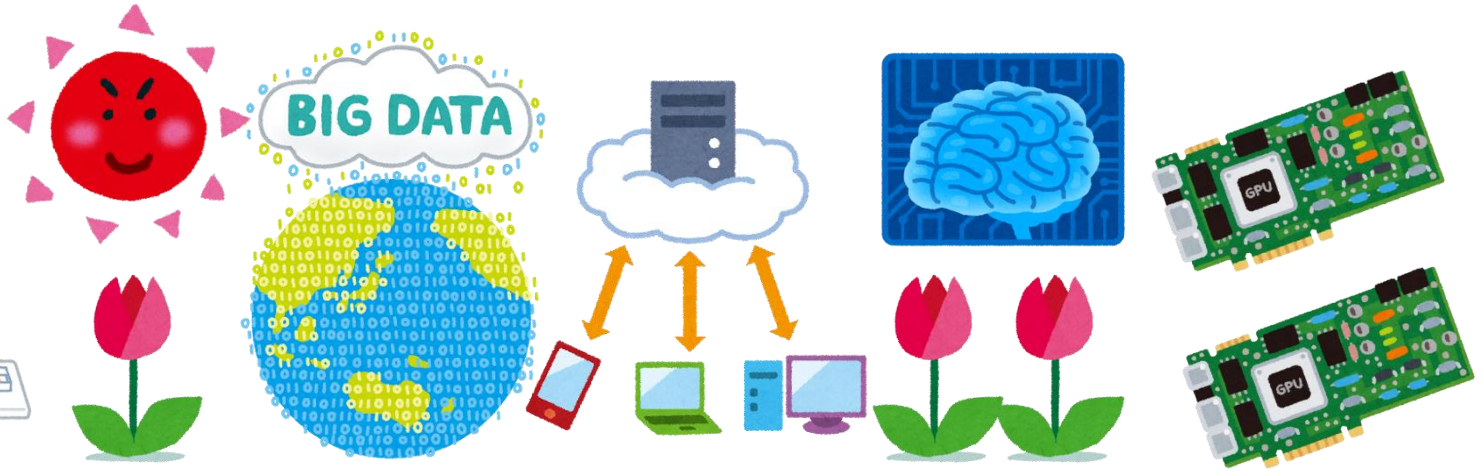
Wikipediaより、コキュートスとは

コーキュートス、または**コキュートス**（Cocytus, Kokytos, 希: κωκυτός）は、ギリシャ神話の冥府に流れる川。その名は「嘆きの川」を意味し、同じく地下の冥府を流れるアケローンに注ぎ込む。

コキュートス（嘆きの川）

地上
（クラウド、GPU)

AIサイエンティスト

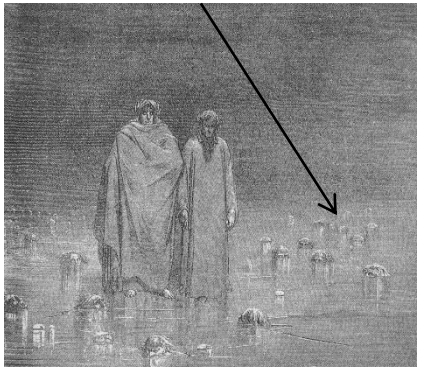


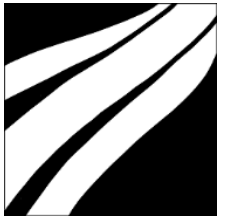
地獄
（組み込み業界）

電力が足りず、GPUやクラウドが使えない世界

層構造

組込みエンジニア達





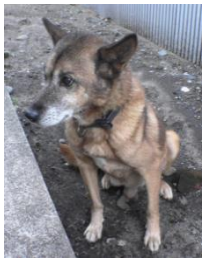
嘆きの川

PASONA TECH

①MNIST CNN
28x28の文字認識

あいうえお

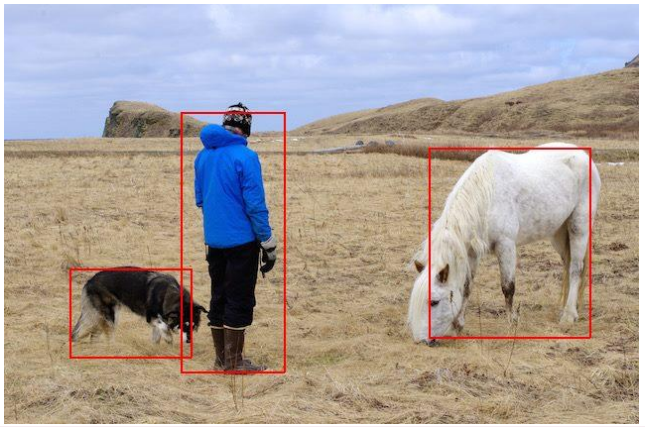
②VGG16



```
'Norwegian_elkhound', 0.6510464
'malinois', 0.20971657
'German_shepherd', 0.124572
'kelpie', 0.0050396309
'Border_terrier', 0.0034161564
```

③Tiny-YOLO

領域提案のニューラルネットワーク

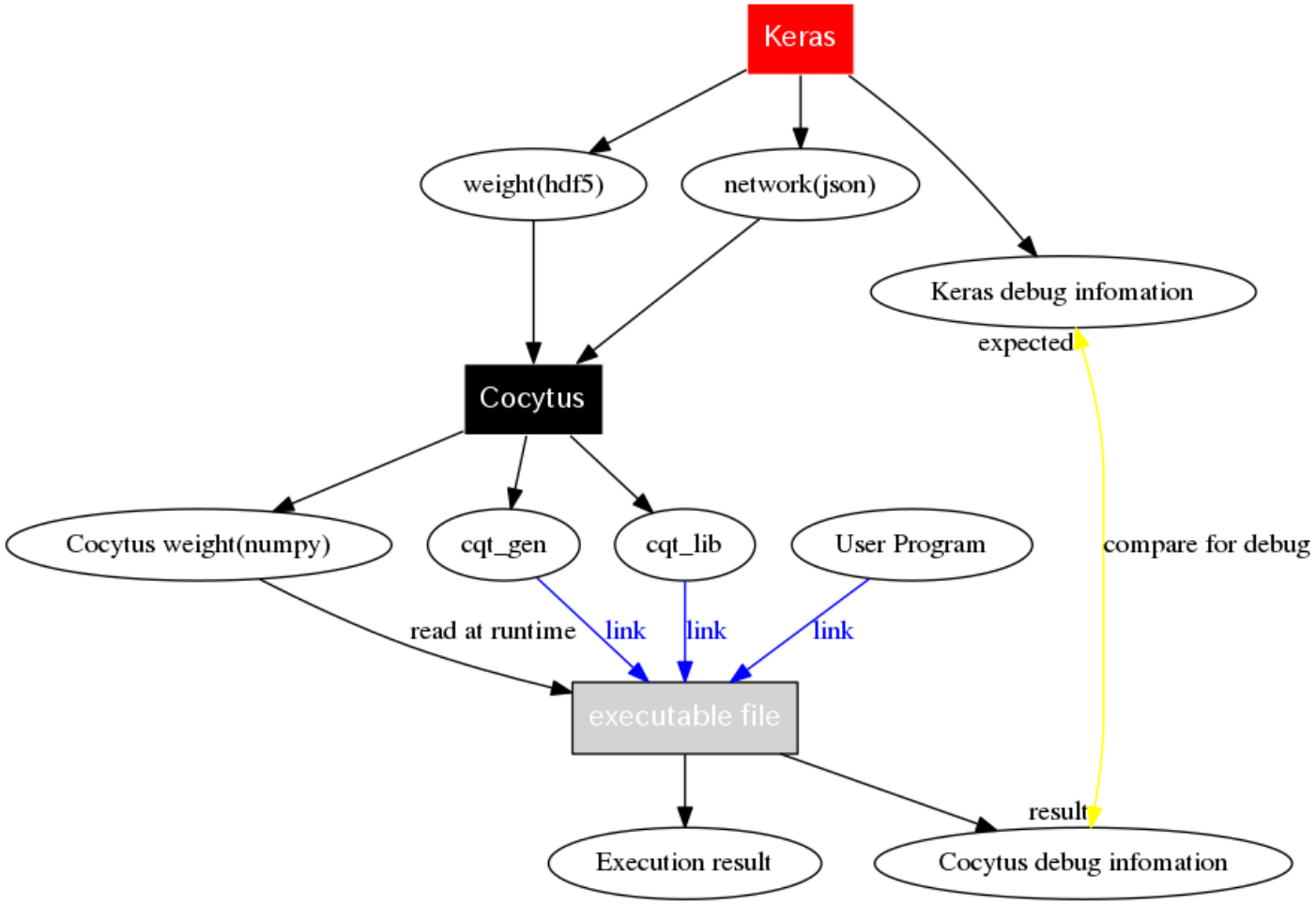


```
Person
Sheep
dog
```

VGG16はラズパイ動作（デフォルトでは速度が出ず）
VGG16のみ固定少数点動作

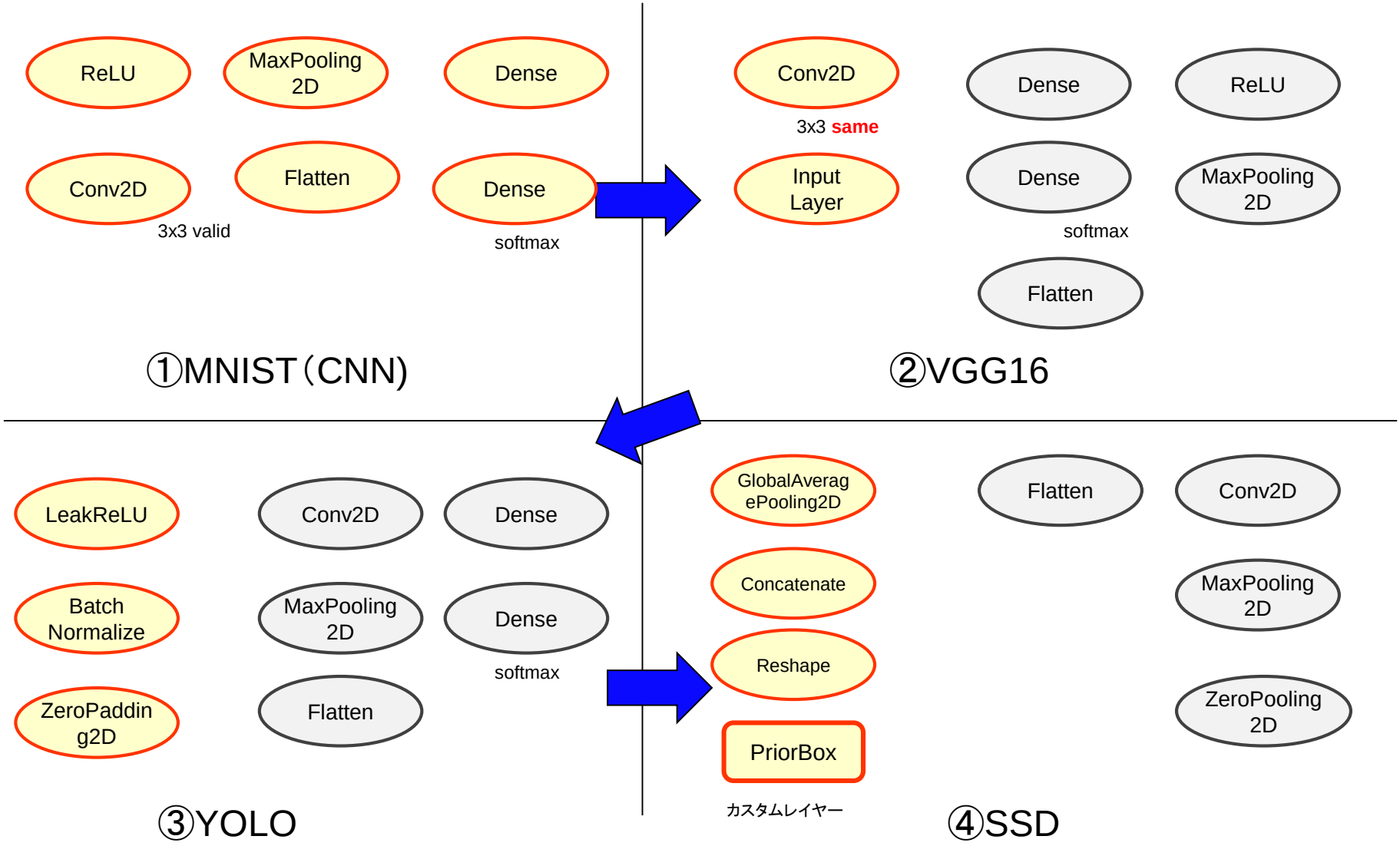
コキュートスの紹介

コキュートスとは



Kerasから、Cソースを半自動生成します。今の所学習はしません。

コキュートスのネット対応方法



コキュートスが出力するCソースの特徴

PASONA TECH

特徴1：どんなプアーな環境でも動くCソースを出力する。
・mallocやファイルシステムを前提としない。
・実行に必要なメモリ使用量が明確
・ブラックボックスを作らない
・必要なライブラリを最小にする。
・特別な処理は、直接Cソースを書き換える。
・環境に応じた最適化は別途行う。

特徴2：C言語が出来れば参加OKのフレームワークを目指す。
・組込機器でのデバッグしやすさを優先
・C++の機能は使わない。
・関数ポインタ、ポインタのポインタ、ポインターの配列は使わない。
・出来る限り、べたにループで回す。
・環境に応じた最適化は出来る人がする。

Pythonの知識が無くてもDeep Learningのプロジェクトに楽しく参加できる！

出力ソースの実例

PASONA TECH

```
//Layers
LY_InputLayer input_1;
LY_Conv2D block1_conv1;
LY_Conv2D block1_conv2;

//weights
NUMPY_HEADER nph_block1_conv1_W;
NUMPY_HEADER nph_block1_conv1_b;
float w_block1_conv1_W[64][3][3][3];
float w_block1_conv1_b[64];
NUMPY_HEADER nph_block1_conv2_W;
NUMPY_HEADER nph_block1_conv2_b;
float w_block1_conv2_W[64][64][3][3];
float w_block1_conv2_b[64];
NUMPY_HEADER nph_block2_conv1_W;
NUMPY_HEADER nph_block2_conv1_b;
float w_block2_conv1_W[128][64][3][3];
float w_block2_conv1_b[128];

//outputs
float input_1_output[3][224][224];
float block1_conv1_output[64][224][224];
float block1_conv2_output[64][224][224];
float block1_pool_output[64][112][112];
```

初期設定

```
strcpy(g_cqt_vgg16.layer[1].name, "block1_conv1");
g_cqt_vgg16.layer[1].type = LT_Conv2D;
block1_conv1.filters = 64;
block1_conv1.kernel_size[0] = 3;
block1_conv1.kernel_size[1] = 3;
block1_conv1.strides[0] = 1;
block1_conv1.strides[1] = 1;
block1_conv1.padding = PD_SAME;
block1_conv1.data_format = DF_CHANNELS_LAST;
block1_conv1.dilation_rate[0] = 1;
block1_conv1.dilation_rate[1] = 1;
block1_conv1.activation = ACT_RELU;
block1_conv1.use_bias = true;
```

データの並びを画処理のエンジニアが理解しやすい順番に変換

デバッグ中に確認したくなるであろう情報は、全てグローバル変数で宣言する。デバッガで止めたときにいつでも確認できる。

https://github.com/natsutan/cocytus/blob/master/example/vgg16/c/cqt_gen/cqt_gen.c

出力ソースの実例

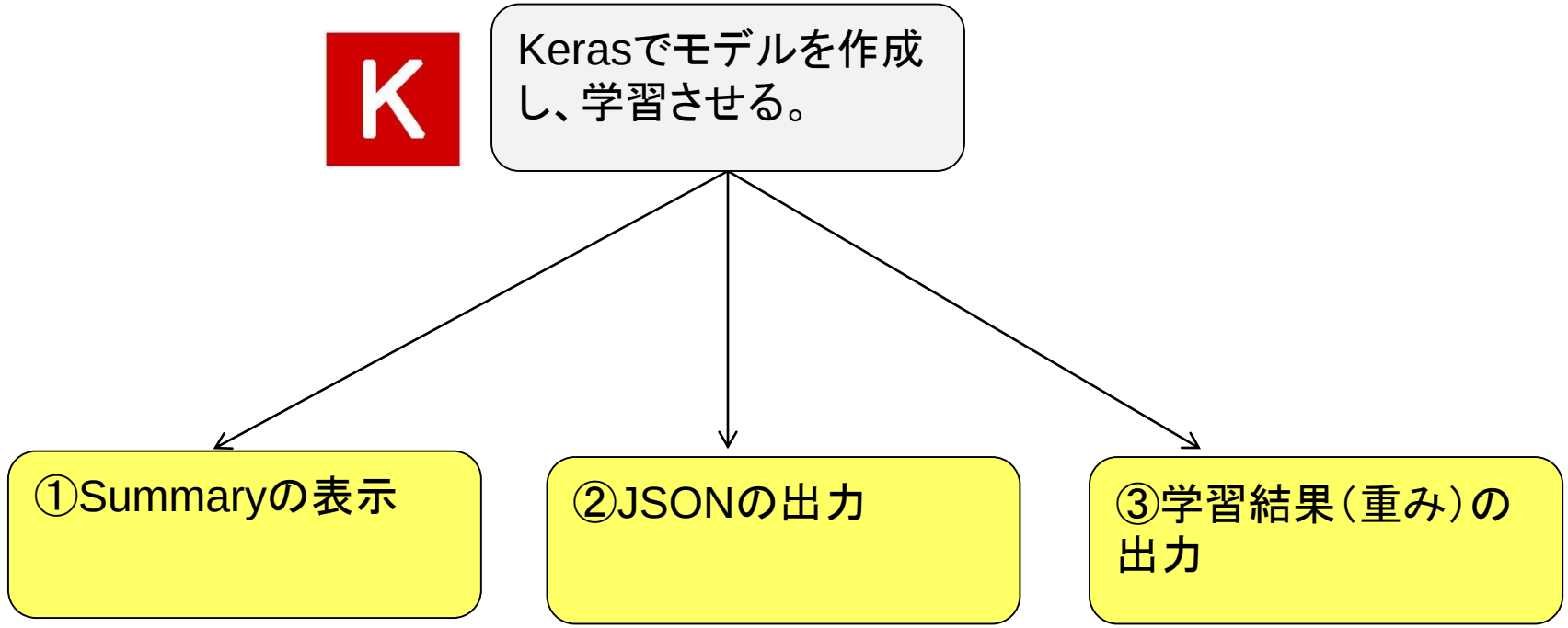
PASONA TECH

```
input_size_x = lp->cqt_input_shape[1];  //画像サイズ
input_size_y = lp->cqt_input_shape[2];  //画像サイズ
input_size_num = lp->cqt_input_shape[3]; //入力の数

for(n=0;n<input_size_num;n++) {
    beta = *((float *)bnp->beta_p + n);
    gamma = *((float *)bnp->gamma_p + n);
    mean = *((float *)bnp->moving_mean_p + n);
    var = *((float *)bnp->moving_variance_p + n);

    inv_denomin = 1.0 / sqrt(var + bnp->epsilon);

    for(y=0;y<input_size_y;y++) {
        for(x=0;x<input_size_x;x++) {
            idx_i = (n * input_size_y * input_size_x) + (y * input_size_x) + x;
            idx_o = idx_i;
            i_data = *(ip + idx_i);

            normalized_data = (i_data - mean) * inv_denomin;
            o_data = normalized_data * gamma + beta;
            *(op + idx_o) = o_data;
        }
    }
}
return CQT_RET_OK;
```

BachNormalization.c

Kerasからの情報取得

Kerasからの情報取得



①summaryの表示

PASONA TECH

model.summary() を使う　　学習データの個数

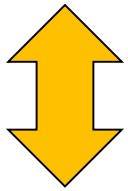
Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 26, 26, 32)	320	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 26, 26, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 24, 24, 32)	9248	activation_1[0][0]
activation_2 (Activation)	(None, 24, 24, 32)	0	convolution2d_2[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 12, 12, 32)	0	activation_2[0][0]
dropout_1 (Dropout)	(None, 12, 12, 32)	0	maxpooling2d_1[0][0]
flatten_1 (Flatten)	(None, 4608)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 128)	589952	flatten_1[0][0]

コンボリューション用の3x3のフィルターが32個あります。重み（学習済みデータ）の個数はいくつですか？3×3×32=288個じゃないの？

各フィルターにbias項があるので、32個増えて288+32 = 320個が正解

②jsonの出力

```
with open(‘output/cnn.json’, ‘w’) as fp:
    json_string = model.to_json()
    fp.write(json_string)
```



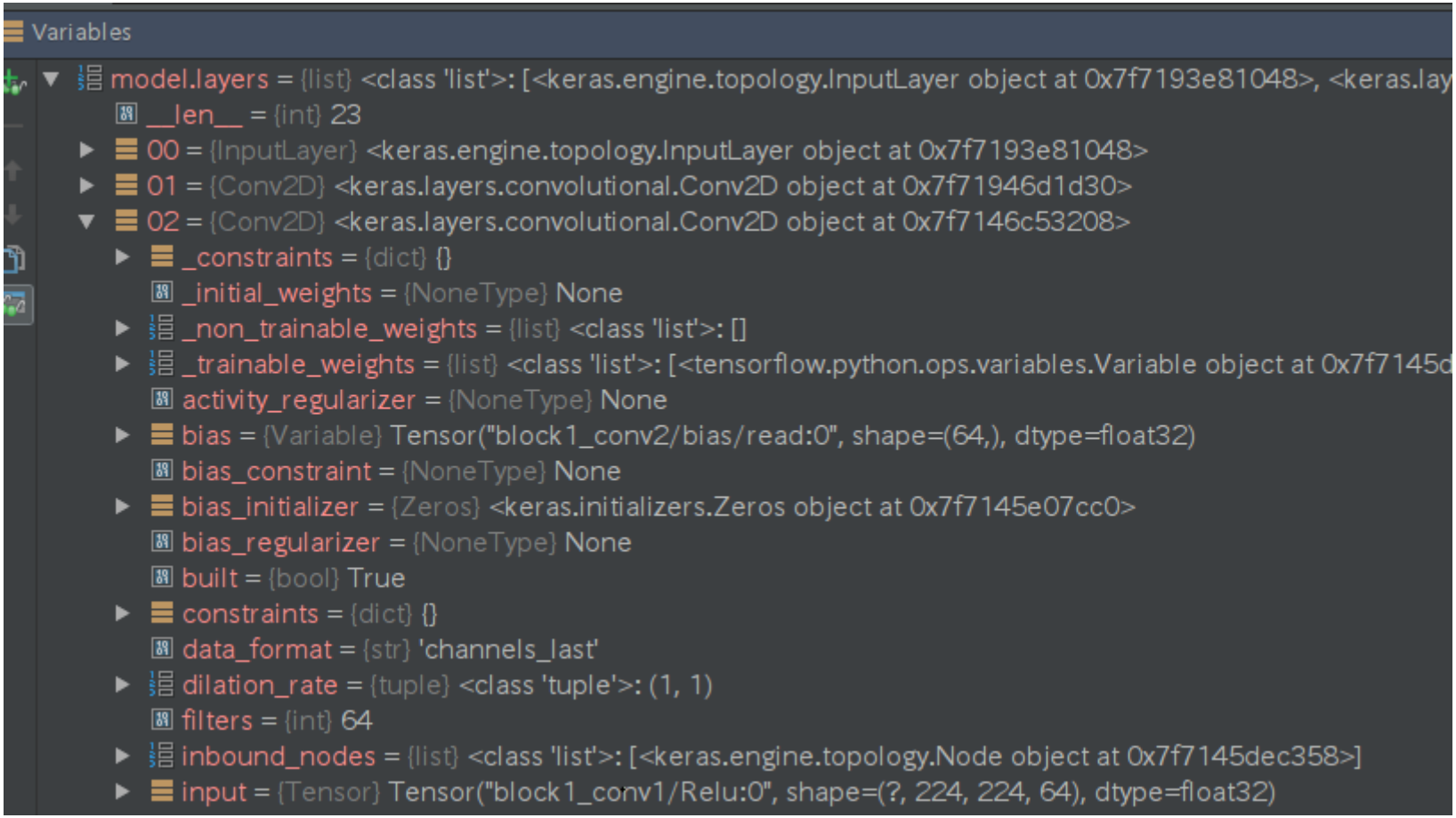
```
json_string = open(json_file, 'r').read()
model = model_from_json(json_string)
```


```
{"kernel_regularizer": null,
    "name": "block1_conv1",
    "trainable": true, "kernel_constraint": null,
    "filters": 64,
    "activity_regularizer": null,
    "bias_constraint": null,
    "strides": [1, 1],
    "bias_regularizer": null,
    "kernel_size": [3, 3],
    "activation": "relu",
```


Keras Layerのコンストラクタに与える引数がjsonに入っている。

カスタムレイヤーの場合は、json出力のために関数を定義する必要がある。

②jsonの出力

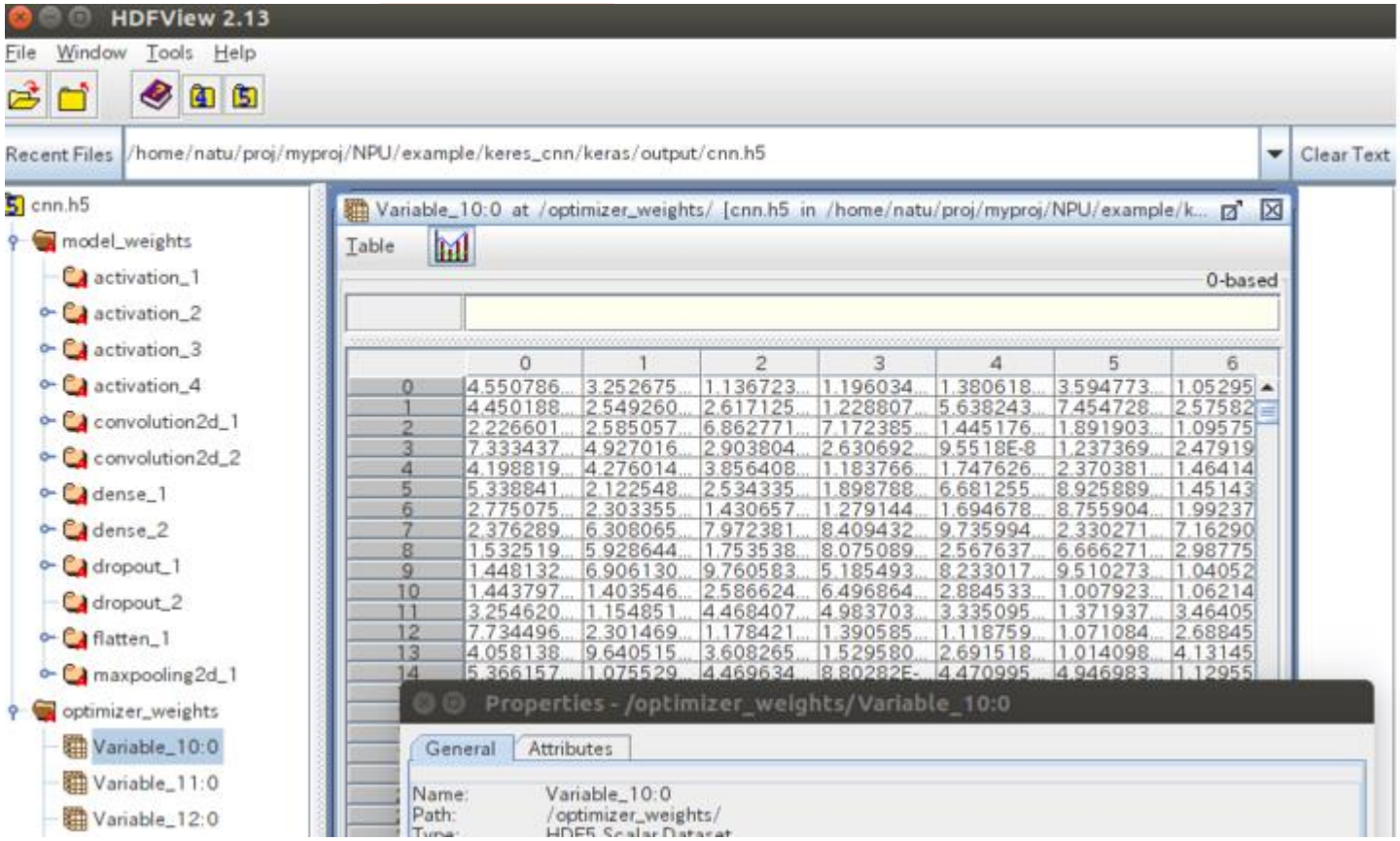


一度jsonファイルを読み込むと、model.layersから各層の詳細情報が取得できる。

③学習結果の出力

```
model.save('output/cnn.h5')
```

Hdf5形式で保存できる



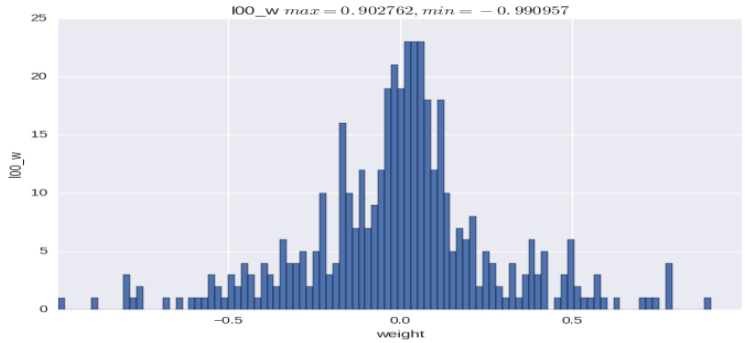
ディレクトリ構造を持ったファイル形式。
コキュートスでは、Kerasのソースコードを流用して、レイヤー単位のnumpy形式で保存している

Numpy形式について

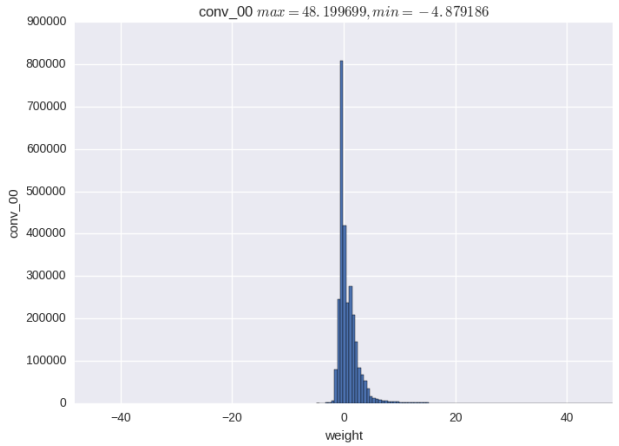
データをnumpy形式でやりとりする。

PASONA TECH

Numpy形式で保存することで、np.loadで簡単に読み込める。
Numpyの統計情報や、matplolibによる可視化が簡単。
デバッグ時に非常に役に立つ。



重みの分布を可視化



ある画像を入れたときの、特定の層の出力値の可視化

kerasで特定の層の出力を取り出す。

PASONA TECH

```
from keras import backend as K

# モデルの読み込み
model = VGG16(include_top=True, weights='imagenet', input_tensor=None,
input_shape=None)

# 入力データをxに設定
img = image.load_img(img_file, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

# K.funtionで第l層の出力を取り出す関数を作る。
l = 10
get_layer_output = K.function([model.layers[0].input, K.learning_phase()],
[model.layers[l].output])

# 出力の取り出し
layer_output = get_layer_output([x, 0])
```

取り出した出力を、np.saveを使ってnumpyフォーマットで保存する。

numpyフォーマット

PASONA TECH

```
        +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F  0123456789ABCDEF
00:000  0 93 4E 55 4D 50 59 01 00-46 00 7B 27 64 65 73 63  .NUMPY..F.{'desc
00:001 10 72 27 3A 20 27 3C 66 34-27 2C 20 27 66 6F 72 74  r': '<f4', 'fort
00:002 20 72 61 6E 5F 6F 72 64 65-72 27 3A 20 46 61 6C 73  ran_order': Fals
00:003 30 65 2C 20 27 73 68 61 70-65 27 3A 20 28 33 2C 20  e, 'shape': (3,
00:004 40 32 32 34 2C 20 32 32 34-29 2C 20 7D 20 20 20 0A  224, 224), }   .
00:005 50 B8 1E 05 41 B8 1E 75 41-A4 70 F5 C1 A4 70 FD C1  ...A..uA.p...p..
00:006 60 52 B8 12 C2 48 E1 5A C1-AE 47 55 42 D7 A3 AA 42  R...H.Z..GUB...B
```

numpyヘッダー
- magic number x93NUMPY
- major version, minor version
- HEADER_LEN
- python dictionary
  - 型 <f4 float32
  - 配列のサイズ (3, 244, 244)

```
float data[3][224][224];

//fseekでヘッダーを飛ばす
fseek(fp, 8+2+hsize, 0);
//freadでサイズ分読み込み
fread(&data, 4, DATA_NUM, fp);
```

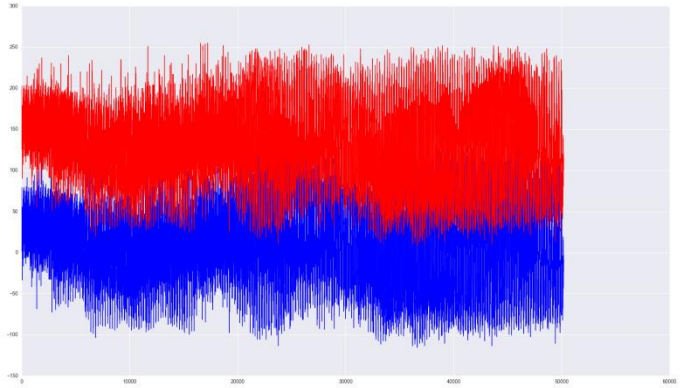
Cソースによるnumpy 読み込み

np.save(filename, data, allow_pickle=False)
を忘れずに

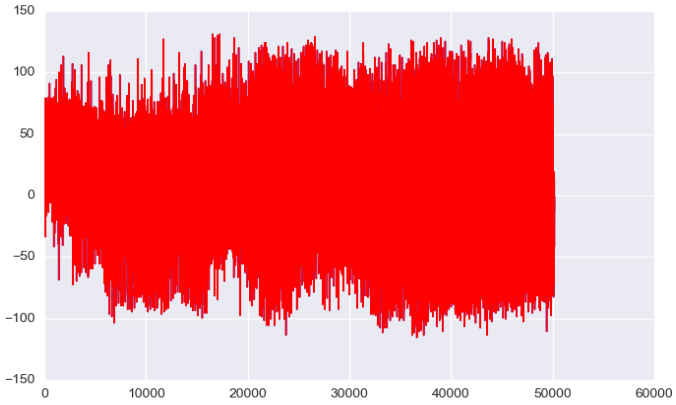
numpyを使ったデバッグ例①

VGG16に犬の画像を入れて、レイヤー0
（InputLayer)の値をnumpy出力して比較した。

青：Kerasの出力
赤：コキュートスの出力

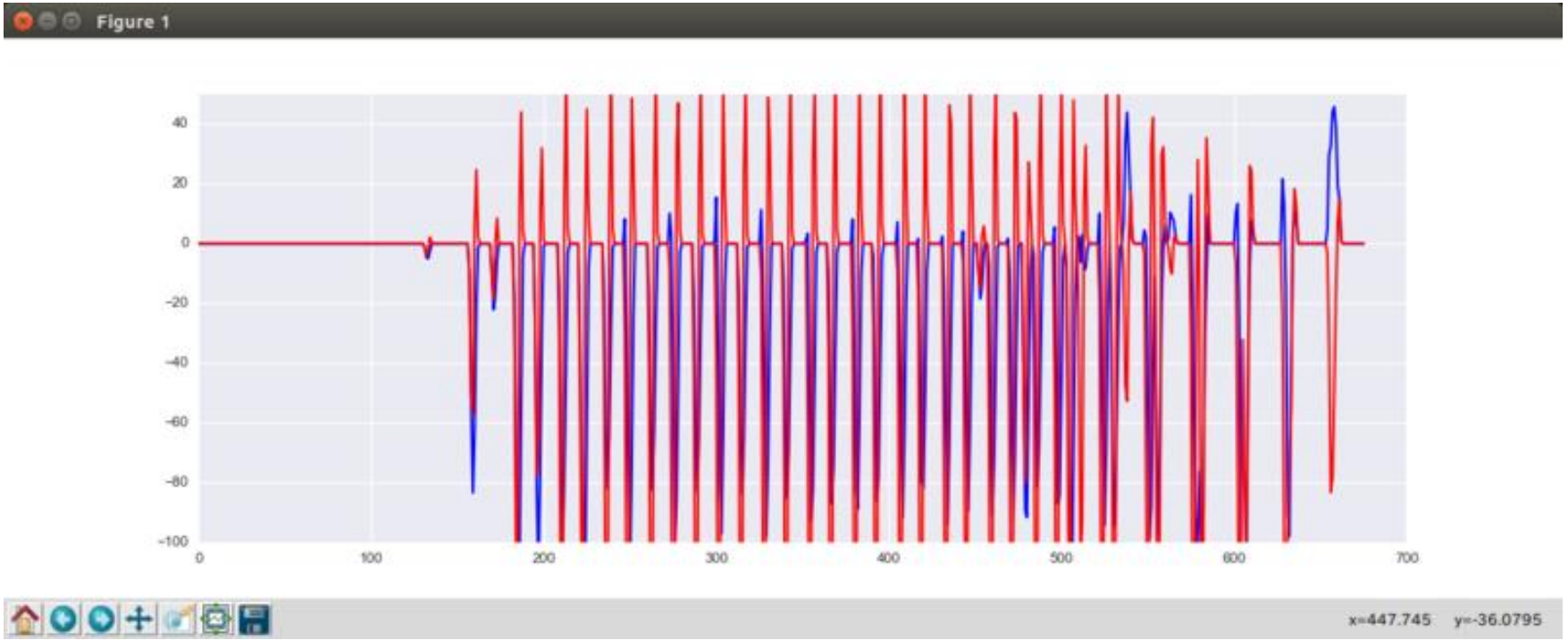


Kerasの入力が、全体的に小さい値になっている。調べてみると、VGG16では、RGBに対して(103.939, 116.779, 123.68)を引かないと行けなかった。



コキュートスの画像変換プログラムを修正
修正後には一致

numpyを使ったデバッグ例②

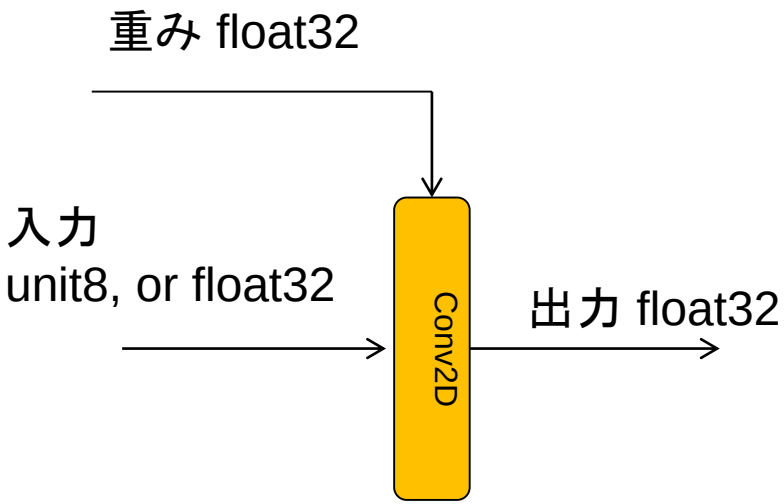


3x3のコンボリューションの例:
ニューロンの反応する場所はあっているけど、計算結果が違う。

→3x3のxとyの取り方が逆でした。

データの型を変更する

コキュートスの型変換方法



MNISTとVGG16では入力の型が違う。
同じソースで処理したいが、Cの枠組みでは扱えない。

将来的には、重みを8bitにしたり、柔軟な型に対応したい。

解決策1：C++の関数テンプレートを使う。

```
template<typename T,typename U> T Add(T a,U b){ return a+b; }
```

技術的には正しい選択だが、C++はいろいろ難易度が高い。

解決策2:ポインターをキャストしもって使う。
Cソースの難易度が上がって、手を入れられなくなる。

さて、どうしましょうか？

コキュートスの型変換方法

C++のテンプレート関数の仕組みを、コキュートス内部に実装。
C++テンプレート関数の恩恵を受けながらも、Cのソースを出力する。

```
int $func_name (CQT_LAYER *lp, void *inp, void *outp)
{
   $weight_type filter3x3[3][3];
   $input_type data3x3[3][3];
   $weight_type bias;

   LY_Conv2D *cnvp;
   cnvp = lp->param_p;

   $input_type *ip = ($input_type *)inp;
```

ライブラリ生成時に変換

```
int CQT_Conv2D_same_3x3_if_of (CQT_LAYER *lp, void *inp, void *outp)
{
   float filter3x3[3][3];
   float data3x3[3][3];
   float bias;

   LY_Conv2D *cnvp;
   cnvp = lp->param_p;

   float *ip = (float *)inp;
```

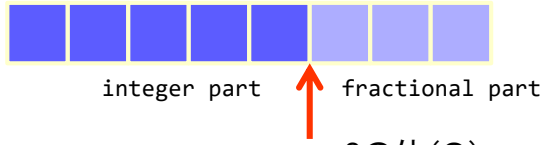
ソースを書き換えて型を指定。
関数名は一定のルールでマングリング
DLの計算は、四則演算+三角関数程度なので、
C言語の暗黙の型変換で十分対応できる。

将来的には、固定少数点数の桁数が違う関数や、特別な仕組みの量子化にも対応予定

固定少数点対応

固定少数点数対応/Quantization(量子化)

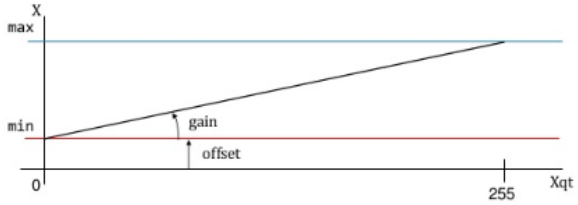
①Dynamic Precision Data Quantization



0の位（Q）

レイヤーによって精度を変える。

②Gain and Offset



Google　TPU

③対数

Bit-width β	Uniform	Gaussian	Laplacian	Gamma
1	1.0	1.596	1.414	1.154
2	0.5	0.996	1.087	1.060
3	0.25	0.586	0.731	0.796
4	0.125	0.335	0.456	0.540

Qualcomm Research

https://arxiv.org/pdf/1511.06393.pdf

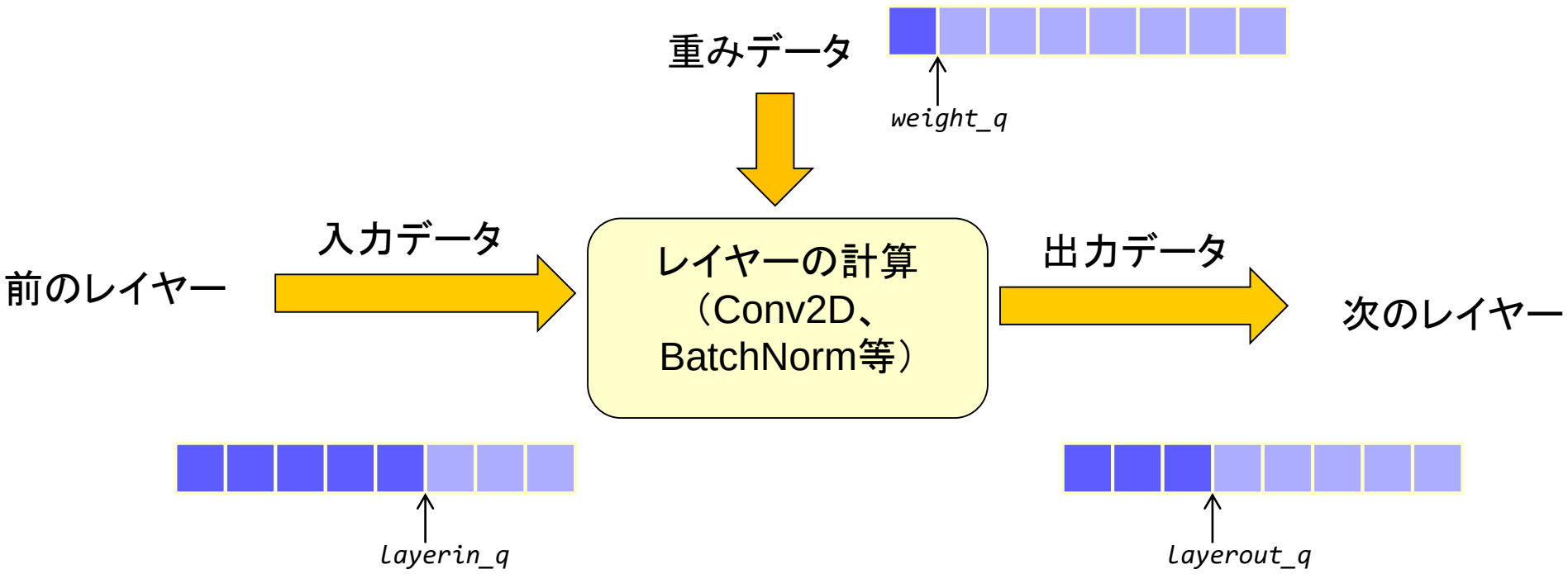
DSP向け

量子化のメリット
・データ数の削減
・FPUが不要になるケースがある。

④Weight Sharingとのあわせ技
→Deep Compression

https://arxiv.org/abs/1510.00149

Dynamic Precision Data Quantizationの計算



大まかに入力データ、重みデータ、出力データで、固定小数点数のフォーマットが違う
コキュートスでは、それぞれのQの位置をパラメータとして持つ。
パラメータに応じて自動的に上手く計算ができるようにしている。（ビットシフト）

重みの量子化方法

PASONA TECH

重み変換時にログが出力されます

```
convert conv2d Q = 14
WQ = 14, (max = 0.358668, min = -0.263980
save conv2d_6/kernel:0 to c_fix/weight/conv2d_6_kernel_z.npy(fix16)

batch_normalization_6
['batch_normalization_6/gamma:0', 'batch_normalization_6/beta:0',
'batch_normalization_6/moving_mean:0', 'batch_normalization_6/moving_variance:0']

WQ = 13, (max = 2.427672, min = 0.480331
save batch_normalization_6/gamma:0 to c_fix/weight/batch_normalization_6_gamma_z.npy(fix16) min =
0.480331, max = 2.427672
```

設定ファイルに記載

```
; レイヤー毎のカスタマイズ
[conv2d_1]
layerin_q=8
layerout_q=8
weight_q=14
```

← 手で指定

重み変換時に調整する。今は手で設定しているが、自動化は当然できる。

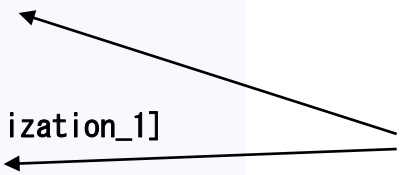
レイヤー出力の量子化手法

実行時にオーバーフローが発生したレイヤーを表示させる。これを元に最適なQの位置を調整する。

```
Overflow:l=2, batch_normalization_1, cnt = 292, inq = 8, outq = 9, wq = 8
Overflow:l=5, conv2d_2, cnt = 2272, inq = 8, outq = 8, wq = 8
Overflow:l=6, batch_normalization_2, cnt = 5, inq = 8, outq = 8, wq = 8
Overflow:l=26, batch_normalization_7, cnt = 27, inq = 10, outq = 10, wq = 10
```

設定ファイルに記載

```
; レイヤー毎のカスタマイズ
[conv2d_1]
layerin_q=8
layerout_q=8
weight_q=14

[batch_normalization_1]
layerin_q=8
layerout_q=8
weight_q=10
```



一度画像を入れて、オーバーフローのチェックをする。自動化は可能。

領域提案時の問題点(Tiny-YOLO使用)

PASONA TECH

識別(*Classification*)の場合

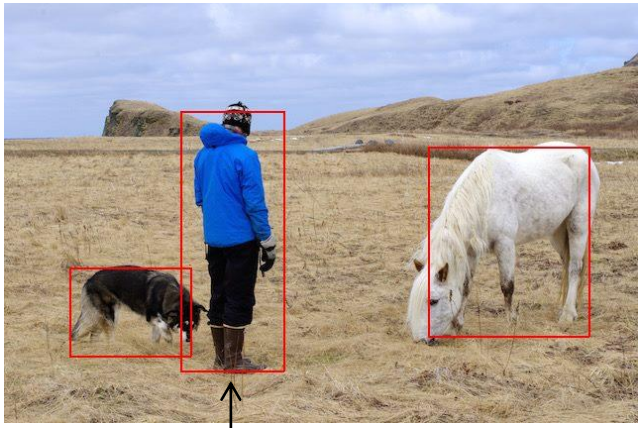
```
'Norwegian_elkhound', 0.6510464
'malinois', 0.20971657
'German_shepherd', 0.124572
'kelpie', 0.0050396309
'Border_terrier', 0.0034161564
```

一番大きな値を識別結果とする場合、
固定小数点数化による計算誤差の影響は少ない。

0.65が、0.99だろうが、0.30になろうが、
識別結果(一番大きな値)は同じ。

これがバイナリネット等ができる理由

領域提案の場合



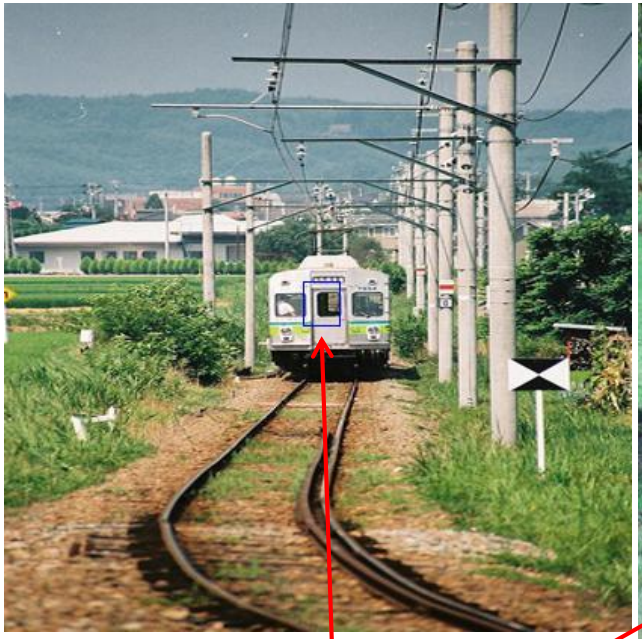
固定小数点化による計算誤差が、
そのまま領域に乗ってくる。
難易度が高い

領域提案時の問題点(Tiny-YOLO使用)

PASONA TECH

コキュートス版Tiny-YOLO（固定小数点数版）の出力例

計算した領域が
大きくなりすぎる



センターはあっているが、領域が小さすぎる。

領域提案における固定小数点数化は今後の課題

Kerasの標準でない機能について

YOLOの構成図

PASONA TECH

You Only Look Once:Unified, Real-Time Object Detection

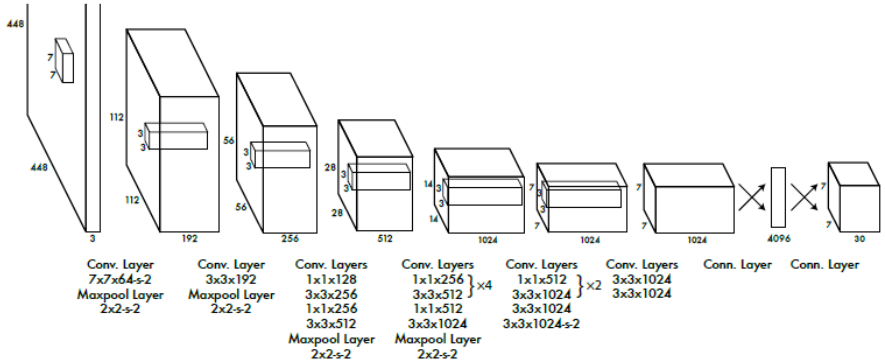
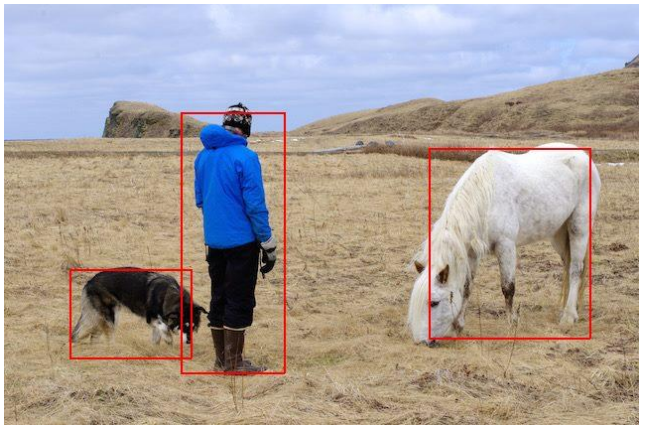


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

https://arxiv.org/pdf/1506.02640.pdf

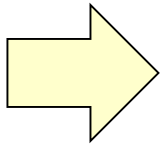


```
sheep 0.814217 (415, 145), (571, 336)
person 0.665931 (172, 109), (273, 371)
cow 0.438520 (64, 267), (182, 356)
```

ニューラルネットワークの出力から、領域提案までに、もう一手間が存在している。
YOLOに限らず、他の領域提案NNでも同様で、PythonやC++で記載されている。
ここは自力でCで実装しないといけない。

対応するCソース

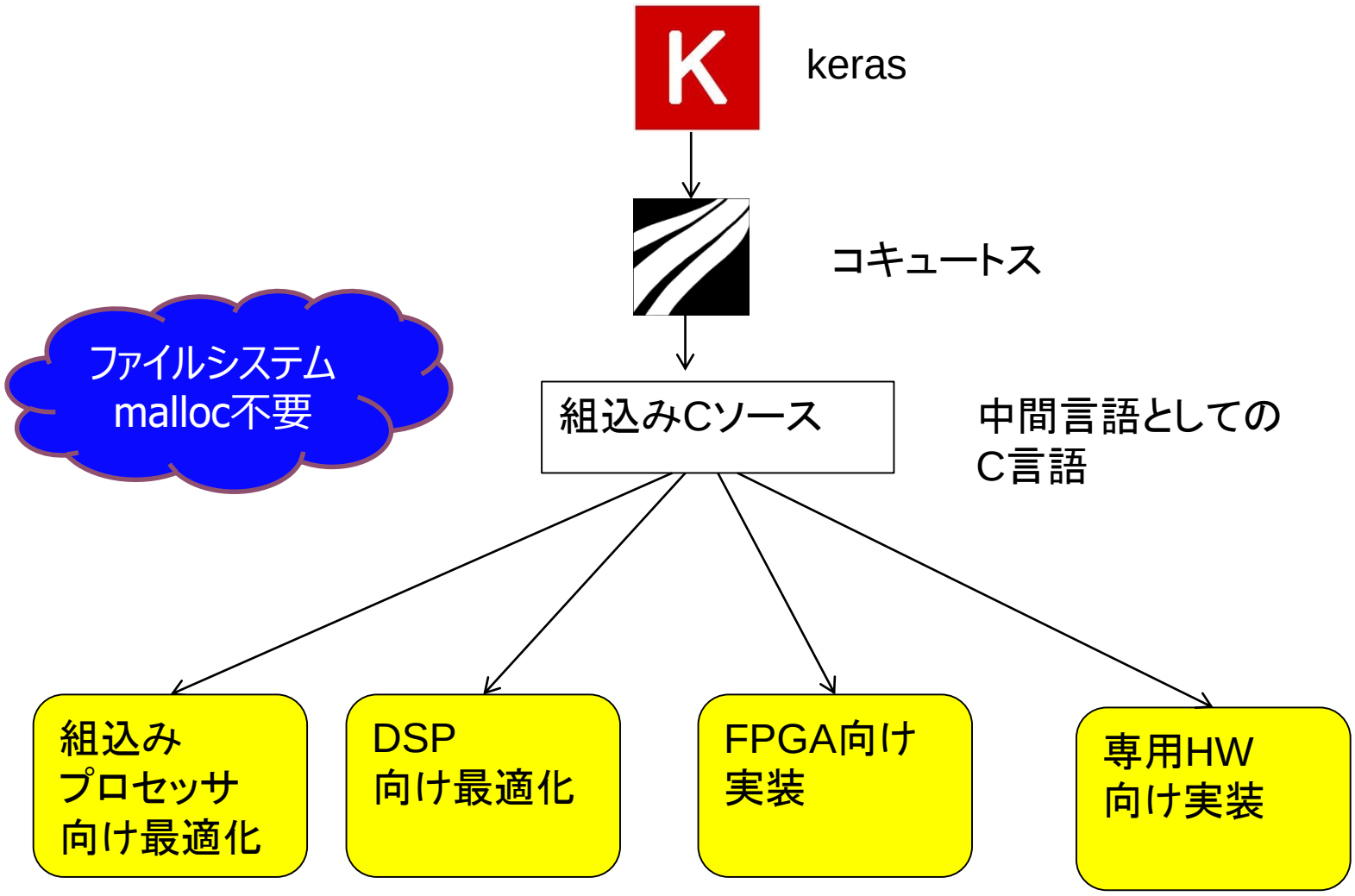
```python
def non_max_surpression(boxes, scores, tresh):
    """    img = Image.open(img_file)

    see http://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/
    :param boxes:
    :param scores:
    :param tresh:
    :return:
    """
    pick = []
    x1 = boxes[:,0]
    y1 = boxes[:,1]
    x2 = boxes[:,2]
    y2 = boxes[:,3]
    area = (x2 - x1 + 1) * (y2 - y1 + 1)
    idxs = np.argsort(scores)

    while len(idxs) > 0:
        last = len(idxs) - 1
        i = idxs[last]
        pick.append(i)

        idx_last = idxs[:last]
        xx1_tmp = x1[idx_last] #  x1[idxs[:last]]
```



```cpp
int non_max_surpression(int num, float iou_thresh)
{
    float area[YOLO_MAX_RESULT];
    bool remove_flg[YOLO_MAX_RESULT];
    int idxs_len = num;

    int i, j;
    int tmp;
    float a, b;
    float x1, y1, x2, y2;
    int last;
    int idxs[YOLO_MAX_RESULT];
    int idxs_work[YOLO_MAX_RESULT];
    float xx1, yy1, xx2, yy2;
    float w, h, overlap;

    int idwork_idx;
    int ret_idx = 0;

    //エリアの計算
    for(i=0;i<num;i++) {
        x1 = filtered_boxes[i].box.left;
        y1 = filtered_boxes[i].box.top;
        x2 = filtered_boxes[i].box.right;
        y2 = filtered_boxes[i].box.bottom;
        area[i] = (x2 - x1 + 1) * (y2 - y1 + 1);
    }

    //とりえあず、バブルソート
    for(i=0;i<idxs_len;i++) {
        idxs[i] = i;
    }
```

Kerasの標準の機能（標準のレイヤー）だけで対応できない箇所は、人手でCへ移植する。

まとめ


keras
コキュートス
ファイルシステム
malloc不要
組込みCソース
中間言語としての
C言語
組込み
プロセッサ
向け最適化
DSP
向け最適化
FPGA向け
実装
専用HW
向け実装

コキュートスの特徴

PASONA TECH

①どんな組み込み機器でも動作する

オープンソース
Github公開

②開発が簡単

C言語が少し出来たらOK
DLやPython、フレームワークの知識不要

AI分野に関するパソナテックのサービス

PASONA TECH

①大量のデータに関する業務
- ・データ収集
- ・データ入力
- ・データタグづけ、不正データ除去

他部署と連携し、大量データを安価にご提供します。
セキュリティ面もご相談ください。

②機械学習に関する業務
- ・学習環境の構築
- ・学習器の設計、評価
- ・パラメータ調整

amazon web services

お客様の目的に沿ったビジネスパートナーをご紹介します。

OPTiM®

③商品化への業務
- ・組込プログラミング
- ・機械設計

専門スキルを持った人材をご提供いたします。
委託契約、派遣契約どちらも可能です。

機械学習に関する業務に関して、どの工程でも必要なサービスをご提供致します。

ご静聴ありがとうございました。