連載タイトル　生成 AI×エッジ・デバイスで AI 画像認識（仮）

連載 3 回目
タイトル　カラーボール認識のアルゴリズム～シンプルな 6 層 CNN の仕組み（仮）
サブタイトル　いろいろな画像認識に使える畳み込みニューラルネットワーク！（仮）
著者　岩田利王（いわた・としお）

●生成 AI 時代のディープラーニング（セミナもある）
　前回までの連載では Google Colab（＊１）上で Python プログラムを走らせて次を行いました。
①生成 AI でカラーボールの画像セットを自作
②パソコンで学習。学習済みモデルを生成
③パソコンで推論。認識率や実用性を確認
　今回は②③のアルゴリズムや Python プログラムについて説明します。また①②③の内容を実践するセミナも用意しています（？ページ）

●エッジ AI の有力候補～ラズパイと FPGA（セミナもある）
　次回以降は次のように進めます（来月号特集の一部になる予定）。
④ラズベリーパイで推論。USB カメラからの動画のリアルタイム認識
⑤FPGA で推論。CMOS イメージセンサからの動画のリアルタイム認識
　エッジデバイス 2 種（ラズベリーパイ／FPGA）の画像認識性能や手順を比較します。また④⑤の内容を実践するセミナも用意しています（？ページ）

　CNN は畳み込みが 4 層、全結合が 2 層からなる

　本連載では、画像認識に適したアルゴリズムである CNN（Convolutional Neural Network、畳み込みニューラルネットワーク）を使用します。図１のように 6 層からなり、最初の 4 層は「畳み込み層」、その後の 2 層は「全結合層」といいます。

●本 CNN の入力は 3ch、30×30 画素
　1 層目の入力はカラー画像なので RGB、すなわち 3ch になります。また画素数は 30×30 になります（＊２）。畳み込みの場合、層が進むたびにチャネル数が徐々に増え、画素数は徐々に減っていきます。

●1～4 層目までは畳み込み層
　4 層目の出力ではチャネル数は 128ch まで増え、画素数は 4×4 まで減ります。そして全チャネルの全画素（128×4×4＝2048）が一列に並べられ、5 層目の入力となります。

●5～6 層目は全結合層

5 層目からは全結合層になり、入力は 2048 画素になります（チャネル数は 1ch)。全結合の場合、層が進むたびに画素数は減っていき、6 層目の出力は 4 画素となります。

●最終層の出力は 4 クラスの特徴量となる

4 画素のうち最大の値を持つものが「推論結果」となります。この CNN はカラーボールを認識するもので、クラス数（ボールの色）が４つ（赤／緑／青／黄）だからです。

このような手順で推論を行う～1 層目は畳み込み層

まずは 1 層目から説明します。図 1 に示すように 1 層目では 3ch→16ch と増え、画素数は 30×30→28×28 と一回り小さくなります。

●畳み込みという一種の 2 次元フィルタ演算

前回の連載で生成した「学習済みモデル」には、3×3 の要素を持つ重み係数（カーネル）が含まれます（図２)。畳み込みではカーネルを同図のように走査しながら積和演算を行います。このような作業を「畳み込み」といいます。

●横に 1 個ずらしながら積和。右端まで行ったら縦に 1 個ずらす

1 層目の入力は 30×30 画素なので（＊２)、カーネルが動ける範囲は 28×28 になります。したがって 1 層目の出力は一回り小さい 28×28 画素になります。

●入力 3ch それぞれ違う重み係数で積和→さらに足し込む

カラー画像なので入力チャネルは RGB の 3ch で、それぞれ固有のカーネルがあります。図３のように 3ch ぶんの積和結果がさらに加算されて一画素になります。それを 28×28 のエリアで繰り返し、出力の 1ch、28×28 画素が生成されます。

●出力 16ch の重み係数もそれぞれ違う

1 層目の出力は 16ch あり、それらにも固有のカーネルが存在します。すなわち全部で 3×16 個（入力 ch×出力 ch）のカーネルがあり（図 4)、それらで畳み込みを行う結果、16ch の画像が生成されます。

●バイアスは出力 ch と同じ 16 個ある

また学習済みモデルには「バイアス」も含まれます。同図のように畳み込みの後、バイアスが加算されます。出力チャネルそれぞれに固有のバイアス値が存在し、各チャネル 28×28 画素にその値が加算されます。

●層の最後は ReLU という非線形関数

その後は ReLU 関数が施されます。入力が負の数だったら０，正の数だったらそのまま出力する関

数です。すなわち 1 層目の出力は 0 または正の値になります。

●1 層目をまとめると次のようになります（表 1）

入力	30×30 画素	3ch
出力	28×28 画素	16ch
重み係数	3×16×9＝432 個	カーネル 3×3＝9
バイアス	16 個	

2 層目も畳み込み層

1 層目の出力は 2 層目の入力になります。2 層目では 16ch→32ch となり、画素数は 28×28→13×13 とぐっと減ります（図 1）。

●カーネルは全部で 16×32（入力 ch×出力 ch）

入力 16ch にはそれぞれ固有のカーネルがあり、それらにより出力 1ch が生成されます。出力は 32ch あり、それらにも固有のカーネルが存在します。カーネル数は全部で 16×32＝512 となり、それらを使って 32ch の画像が生成されます。

●畳み込みの後、バイアス加算と ReLU…ここまでは 1 層目と同じ

入力は 28×28 画素なので、カーネルの動く範囲は 26×26 になります。したがって畳み込みの結果、26×26 の画像が 32 枚生成されます（図５）。

畳み込みの後は 1 層目と同様にバイアスが加算され、さらに ReLU 関数が施されます。

●4 画素から最大値を 1 個ピックアップし縦横半分に

1 層目と違うのはその後の「プーリング」で、26×26 画素を間引いて小さくします。間引き方は「近傍の 4 画素のうち最大のものを残す」というもので、図６のように画素数は 13×13 となります。このようなプーリングがあるのは 2 層目と 4 層目のみです。

●2 層目をまとめると次のようになります（表 2）

入力	28×28 画素	16ch
出力	13×13 画素（プーリング後）	32ch
重み係数	16×32×9＝4,608 個	カーネル 3×3＝9
バイアス	32 個	

3、4 層目までは畳み込み層

3、4 層目でチャネル数は 32ch→64ch→128ch と増えていき、画素数は 13×13→11×11→4×4 と減っていきます（図 1）。

●カーネル数は 32×64（入力 ch×出力 ch）

3 層目の入力は 32ch、それぞれ固有のカーネルがあり、それらにより出力 1ch が生成されます。そして出力は 64ch あり、それらにも固有のカーネルが存在します。カーネル数は全部で 32×64＝2048 となり、それらを使って 64ch の画像が生成されます。

●畳み込みにより出力 64ch 生成、それぞれ 11×11 画素

入力は 13×13 画素なので、カーネルの動く範囲は 11×11 になります。したがって図 7 のように 11×11 の画像が 64 枚生成されます。前層までと同様に畳み込みの後、バイアス加算→ReLU 関数が施されて出力となります。

●3 層目をまとめると次のようになります（表 3）

入力	13×13 画素	32ch
出力	11×11 画素	64ch
重み係数	32×64×9＝18,432 個	カーネル 3×3＝9
バイアス	64 個	

●入出力 ch の多い 4 層目はカーネルも多くなる

同図のように 4 層目の入力は 64ch、出力は 128ch あり、カーネル数は全部で 64×128＝8192 となり、それらを使って 128ch の画像が生成されます。

●奇数画素なのでプーリングに注意

入力が 11×11 画素なので出力は 9×9 画素となります。畳み込みの後、バイアス加算→ReLU 関数が施されますが、2 層目と同様に 4 層目にもプーリングがあります。9×9 画素なので、図 8 のように右端と下端を捨てて 8×8 画素にした後、縦横半分にして画素数は 4×4 になります。

●4 層目をまとめると次のようになります（表 4）

入力	11×11 画素	64ch
出力	4×4 画素（プーリング後）	128ch
重み係数	64×128×9＝73,728 個	カーネル 3×3＝9
バイアス	128 個	

5, 6 層目は全結合層になる

以降は畳み込みではなく全結合層になり、チャネル数は 1ch となります。図 1 のように画素数は 2048→128→4 と減っていきます。

●2 次元的な画像を 1 次元に並び替える

4 層目の出力は 4×4 画素が 128ch あります。これらを図 9 に示すように一列に並び替えます。 同図における Zc,x,y の c はチャネル番号、x,y は縦横の座標になります。

●2048 画素の並べ方に注意！

すなわち最初に左上の画素を 128 チャネルぶん並べ、次は右隣の画素を 128 チャネルぶん並べます。これを 4×4＝16 画素において行うと、結果は同図のように 2048 個となり、これが 5 層目の入力になります。

●全結合層は行列を乗算する（畳み込みはしない）

5 層目以降は全結合層なのでカーネルという概念はありません。図 10 のように 2048×128 の行列を用意して入力に乗算します。その結果は 128 個になります。その後バイアス加算→ReLU 関数と続き、その結果が 5 層目の出力となります。

●5 層目をまとめると次のようになります（表 5）

入力	1024 画素	1ch
出力	128 画素	1ch
重み係数	2048×128＝262,144 個	5 層目が一番多い（コラム B）
バイアス	128 個	

●6 層目は行列乗算→バイアス加算で終わり（ReLU はなし）

5 層目の出力は 6 層目の入力となります。この層も全結合なので、図 11 のように 128×4 の行列と乗算します。バイアス加算した結果は 4 個になりますが、それらは赤／緑／青／黄色ボールの特徴量を表します。そして一番大きな値が「推論結果」となります。

●6 層目をまとめると次のようになります（表 6）

入力	128 画素	1ch
出力	4 画素	1ch
重み係数	128×4＝512 個	
バイアス	4 個	

クラスを増やす際の Python プログラムの変更点

前回の連載では PC(Google Colab)で画像認識の Python を走らせました。ここではそれらプログラムについて説明します。

●床の誤認識を減らしたのを思い出そう

リスト１は Python プログラムの一部です。同リスト(a)は赤／緑／青／黄色ボールの 4 クラスで判定を行うものですが、これでは床の誤認識が多くて実用的ではありませんでした。したがって同リスト(b)のように「床クラス」を加えて全部で 5 クラスとし、誤認識を劇的に減らしました（前回の連載の図１４→図１９参照)。

●CNN の model 定義の部分、最終層だけ 4→5 にする

リスト 2 も同様にクラスを増やす際の Python プログラム変更点です。同リストでは最終層（6 層目）の出力画素数（すなわちクラスの数）を 4 から 5 に変更しています。

●どの画像が床クラスに属するかはラベルリストに記述

リスト 3 は推論用プログラムの一部で、ラベルリストを 5 クラス用に変更しています。TestBalls_5.csv を開くとリスト 4 のように床クラスの ClassID と画像の Path が追加されています。この場所(val_1/floor)に推論用の床画像を置きます（学習用は train_1/4 に置く)。

●クラスを増やすときの変更点は”floor”をサーチで

Python プログラムは学習用、推論用（認識率確認)、推論用（実用性確認）の３つあります(balls_float.py, balls_float_pred.py, balls_still_pred_colab.py)。クラス数を増やすときは”Modify for floor detection”というコメントをサーチして適宜変更しましょう。

●いろいろな認識対象に使い回せる

これらの Python プログラムではカラーボールの認識を行いますが、全く別の認識、例えば MNIST（手書き数字認識）にも使えます（コラム A 参照)。その場合、学習用・推論用画像を MNIST のものと置き換え、クラスの数を 10（数字が 0～9 まで 10 種あるので）とします。またラベルリストも変更します。

□コラム A　世界中の AI 科学者が集うサイト Kaggle を参考にした

本連載で使用する Python プログラムは筆者が一から考えたものではありません。kaggle.com に行き、適当にサーチして「GTSRB - CNN (98% Test Accuracy)」を見つけます。すると Shivank Sharma 氏のコードが見つかり、それを参考にして私（岩田）が作成したものです。

GTSRB は German Traffic Sign Recognition Benchmark の略で、ドイツの道路標識の認識を行うものです。つまり認識対象は元々「道路標識」であり、それを「カラーボール認識」に転用したのが本連載の Python プログラムです。したがって MNIST に転用するのも可能ですし、それ以外、例えば信号機の認識、自動車/バイク/人の認識などへの転用も可能だと思います。

□コラム B　後々のハード化のために必要な Python の変更

リスト 2 に 5 層目の定義があり、出力画素数が 128 になっています。オリジナル（Shivank Sharma

氏のコード）では 512 でしたが、将来 FPGA に実装することを鑑みて私が 128 に変更したものです。

表 5 を見てみましょう。5 層目の重み係数が 262,144 個となっており、他層より突出して多くなっています。係数の数は入力画素数(2048)×出力画素数(128)になるので、出力が 512 のままだと 4 倍の 1,048,576 個となってしまいます。FPGA において係数は ROM(初期値付き RAM)に入れるので、この係数量がリソースを圧迫し、廉価な FPGA では回路が入り切らないことが予想されます。

そこで 5 層目の出力を減らして Python を走らせます。512→256→128 としてもさほど認識率は下がらず、64 にするとガクッと下がりました。このように後々のハード化を見据え、Python の時点で試行錯誤することも重要になります。

（本文終わり）いわた・としお

□お知らせ

〇**セミナ**のお知らせ：本連載記事に連動したセミナを 2 つ用意しています。

・生成 AI 時代のディープラーニング～画像生成してシンプルなニューラルネットワークで認識！：2/18（火）

・エッジ AI の有力候補～ラズベリーパイと FPGA～画像認識性能＆手順を比較！：2/27（木）

開催日は予定です。詳細は「CQ エレクトロニクスセミナ」のサイトで。

（＊注釈）

（＊１）Google Colab（Google Colaboratory）とはブラウザ（Google Chrome 推奨）上で動く Python 実行環境であり、GPU も無料で使用できる（使用量制限あり）。高価なグラフィックボードのないパソコンでも生成 AI が使えるようになる

（＊２）Python プログラム内で 30x30 画素にリサイズする（実際の画像より一回り小さくする）。