気象・雪氷分野向け Python3_CNN の導入(Keras+TensowFlow2 系)

Ver. 2.7b　2022/06/21　Nakai, S.

まえがき：　なぜこのようなドキュメントを作ろうとしたのか？

筆者は機械学習の専門家ではないし、工学分野の人間ですらない。しかし、降雪研究の一環として機械学習を用いたレーダー降水分布画像分類を行うことになった。実際に機械学習のモデルを導入、実行し、結果が出る所まで手を動かした結果、気象学・雪氷学のような分野の者にとってはあまり合理的でないハードルがあることに気がついた。であれば、そのハードルを多少なりとも下げられるなら、書く意味はあるだろうと考えたのが動機である。

そのハードルとは、

i. そもそもインストールや設定など導入がうまくいかない。
ii. 本やサイトを見ても用語が何言ってるのかよくわからない。
iii. 結果の説明で「これが効いたと思われる」、それ主観では？と思った。

である。3 番目はハードルと言うよりはモチベーションに対するブレーキと言った方が良いかもしれない。決定論的な解析に慣れている身にとっては、ブラックボックスの結果見てプロセスを想像するしかできない解析に対しては意義が見いだしにくく、その上に i や ii のハードルを越えるために時間をかけることには抵抗がある。しかし、i と ii をほぼ回避できて機械学習のモデルが中で何してるか多少なりとも理解できるなら、通常の降水分布特性の解析ツールの一つとして、機械学習という方法を手に入れることには魅力を感じる。

機械学習にも様々な種類があり、また進化は日進月歩と言われているが、その一方で参考書籍に書かれているような定着した方法、もしくは初学者向けによく説明されている方法も Web サイトなどで見ることができる。筆者がたまたまこれまでに手にしたのが CNN (Convolutional Neural Network; 畳み込みニューラルネットワーク;　機械学習の一種)による画像分類のスクリプトである。その内容を吟味整理しながら実行した結果、気象分野において広く存在する格子データについて、どのようなものでも画像化すれば使える汎用的なツールになりうると考えられた。ソースコードも Github 等に由来する広く使われているものを中心に構成されており、いずれ Web サイトでの公開も可能と思われる。となれば、

1) 作業工程をなるべく一般化してポータブルなソースに説明を付ける。
2) 用語について、気象・雪氷分野で使われていることばに翻訳する。
3) 結果がモデルや入力の何に依るかできるだけソースと関連づけて記述する。

の作業をすればよいのではないか？　まだ途上ではあるが、試みた結果をここで記述する。

謝辞：　ここで作成した処理系は長岡技術科学大学の畠山周愉氏が実務訓練として作成したもの(畠山ら, 2019)が基礎である。研究の立ち上げへの貢献に感謝したい。

畠山周愉, 陸旻皎, 中井専人 (2019): 深層学習の水文･気象学への応用研究の立ち上げ. 第 37 回土木学会関東支部新潟会研究調査発表会, II-315.

参考書：

ここに書いてあることよりさらに基礎的な知識は教科書や参考書を見た方が良い。筆者が参考にしたのは次の 2 冊である。

・実際の作業手順に沿った記述がある。

[1]チーム・カルポ, 2020: TensorFlow2 TensorFlow&Keras 対応 プログラミング実装ハンドブック. 秀和システム, 448pp.

・初学者向けながら系統的な記述がある。

[2]チーム･カルポ, 2021: 物体･画像認識と時系列データ処理入門[TensorFlow2／PyTorch 対応第 2 版] TensorFlow/Keras/TFLearn による実装ディープラーニング. 秀和システム, 583pp.

開発環境の選択：

OS はひとまず Windows 10 とした。Anaconda などツールの導入が容易そうだったのが理由である。しかし、最近になって Anaconda、Python、機械学習関係のライブラリのバージョンが上がり、情報も揃ってきて Linux でも Windows とあまり労力に差が無く導入可能になってきたと見られる。

機械学習のスクリプトは、Anaconda をインストールの後、Anaconda prompt (コンソール) から python *。py を入力して動かした。ものによっては*.py を動かして*.log にコンソール出力を保存するバッチ*.bat を作成し、以後は*`.bat を投入するようにした。Jupyter Notebook のような統合環境は利用していない。理由は、速いからと、このやり方の方が筆者にはなじんでいるからである。上記参考書や多くの Web サイトでは Jupyter Notebook などを利用している。どちらでも使いやすい方で良いと筆者は考えている。

では、がんばりましょう。

目次

既知の問題：　現在、*.py スクリプトの実行は正常にできるが次のような警告が出る。インストール関連の問題。

(tf253) G:¥DL¥procroot202204-2¥分類する>dGradCam3.py

WARNING:tensorflow:From C:¥Users¥python¥anaconda3¥envs¥tf253¥lib¥site-packages¥tensorflow¥python¥keras¥layers¥normalization.py:534: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

2022-05-21 12:07:49.776787: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-05-21 12:07:49.777773: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.

１．環境編

1-1 グラフィック関係のバージョンチェックと nVIDIA 関係のインストール

機械学習の中心的なライブラリである TensowFlow は関係するソフトウェアとのバージョン依存性がかなりあり、問題なく実行させるためには、Python(モデルの言語)、cuDNN(GPU 用ディープラーニングライブラリ)、CUDA(GPU 並列計算用開発環境)のバージョンを合わせる事が重要である。動作確認されたバージョンの組合せは https://www.tensorflow.org/install/source?hl=en#gpu に書かれている(参考：https://nixeneko.hatenablog.com/entry/2021/06/16/000000)。CUDA にはバージョンによって対応する GPU が異なるので、GPU の型番もチェックして適合するものを選ぶ必要がある。

今回は、TensowFlow2.5.3 にした。この場合、Python 3.6-3.9　cuDNN 8.1　CUDA 11.2.2 が対応するバージョンとなる。CUDA と cuDNN は GPU メーカーの NVIDIA が提供しているものであり、入手、インストールは次のようにした。

CUDA：

https://developer.nvidia.com/cuda-11.2.2-download-archive?target_os=Windows&target_arch=x86_64&target_version=10&target_type=exelocal　(実際は 1 行)

から、CUDA11.2.2 をダウンロード、インストーラを実行した。

cuDNN：

https://developer.nvidia.com/rdp/cudnn-archive

から、

Download cuDNN v8.1.1 (Feburary 26th, 2021), for CUDA 11.0,11.1 and 11.2

cuDNN Library for Windows (x86)

と進むと　NVIDIA Developer Program Membership　のアカウント作成を求められる。これは cuDNN には必須である。氏名や分野等を入力して submit すると cuDNN がダウンロードされる。zip を解凍してできた cuda 以下を

C:¥Program Files¥NVIDIA GPU Computing Toolkit¥CUDA¥v11.2

に全コピーした。これは、この下の bin include lib にファイルを追加することになる。

参考：https://okkamotto.blog.jp/archives/21386908.html

パス設定等は何もしなくて良かった。ここに書かれている画像拡大/補正ソフトの waifu2x-caffe で動作チェックして「cuDNN チェック」をクリックして「使えます。」が出ればＯＫ。

なおここでは　Visual Studio 2013 の Visual C++ 再頒布可能パッケージ　もインストールした。(おそらく不要)

他参考サイト：

https://www.kkaneko.jp/tools/win/cuda.html#S0

https://www.kkaneko.jp/tools/win/cuda10.html

TensowFlow とのバージョンの関係など、特にこれが良く整理されているが、必ずしも必要ない内容も含まれているように思われる。

https://nw.tsuda.ac.jp/lec/cuda/install_11-win/

インストールとバージョン選択のかなり実際的なことが書いてあり必見

1-2 Anaconda のインストールと TensowFlow などライブラリ・モジュールの準備

tips： エラーが出た場合は、TensowFlow や Keras のバージョンの違いによることも多い。既存のコードや Web サイトはよく使われる変数名などをそのまま使っていることも多く、エラーメッセージをそのまま Web 検索すると解決法にたどり着くことが多い。

tips： python ソースは UTF-8 で保存すること。SJIS にすると扱えないというメッセージが出て終わる。しかし、python ソースをバッチファイルで呼び出す場合、そのバッチファイルは SJIS で保存すること。そうしないと日本語ファイル名などが化けて動作しない。

1-2-1 まず、Anaconda の最新版を普通にインストールする。そのあと Anaconda Navigator からのライブラリのアップデートはしない(全然終わらない)。

1-2-2 次に、Anaconda Prompt から順次ライブラリをインストールする。Anaconda Powershell Prompt は activate による環境切り換えができなかったので使わなかった。

tips： なぜか Anaconda Prompt が即時異常終了する場合

Anaconda を一度アンインストール、再インストールなどすると発生することがある。

Windows10 では PowerShell で以下のコマンドを打つと解決する。

> Reg Delete "HKCU¥Software¥Microsoft¥Command Processor" /v AutoRun /f

参照： https://qiita.com/migihidari/items/9a07883f6ce03d6fd9c6

参考：https://nixeneko.hatenablog.com/entry/2021/06/16/000000

以下は Anaconda Prompt での作業内容である。なお、作業ディレクトリは必ずしも下記の通りでなくても良いはずだが，ここでは実際に動作したディレクトリをもとに記述する。

(base) C:¥Users¥{ユーザ名}>

(base) C:¥Users¥{ユーザ名}>conda create -n tf253 (「tf253」は任意の名称で良い。)

新しい Environment(Python の実行環境)を作る。

★これをやると余計なものがなくさくさく行く。

・作った Environment は Anaconda Navigator に反映される。

(base) C:¥Users¥{ユーザ名}>conda info -e

conda environments:

#

base　　*　C:¥Users¥{ユーザ名}¥anaconda3

tf253　　C:¥Users¥{ユーザ名}¥anaconda3¥envs¥tf253

設定されている環境の確認。＊が付いている「base」環境が今居る環境である。もし、作った環境がうまく行かなくなくて削除するときは

(base) C:¥Users¥{ユーザ名}> conda remove -n tf253 --all

(base) C:¥Users¥{ユーザ名}>activate tf253

機械学習用に作った「tf253」に移動

(tf253) C:¥Users¥{ユーザ名}>python --version　　←Python のバージョン確認

Python 3.6.13 :: Anaconda, Inc.

バージョン整合が取れることを確認する。GPU 関係ソフトウェアのバージョンは

Python3.6.13　cuDNN v8.1.1　CUDA 11.2.2

であった。この組み合わせだと　tensorflow-2.5.0, tensorflow-2.6.0 が使える。

参考：　https://www.tensorflow.org/install/source?hl=en#gpu

tips：環境に python も何もインストールされていないことがある。その場合

(tf253) C:¥Users¥{ユーザ名}>python --version　　と打つと

Python

または

'python' は、内部コマンドまたは外部コマンド、

操作可能なプログラムまたはバッチ ファイルとして認識されていません。

と表示される。こういうときは

(tf253) C:¥Users¥{ユーザ名}>conda install python==3.9.7

でまず python をインストールする。バージョンに注意。

深層学習関係のライブラリのインストールに入る。

TensowFlow が最も厄介そうなので、最初に入れる。

~~(tf253) C:¥Users¥{ユーザ名}>conda install tensorflow==2.5.0~~

(tf253) C:¥Users¥{ユーザ名}>conda install tensorflow-gpu==2.5.0

(2.5.3 としたらパッケージないと言われた。)

(python は 3.6-3.9 が整合性あるはずだが、python3.6 はダメと言われた。)

(tf253) C:¥Users¥{ユーザ名}>conda update python　→　3.9 に

tensorflow-2.8 も表ではいけそうだが、ひとまず 2.5 で様子を見る。

2.5、あっさり入った。

参考：　https://rupic.hatenablog.com/entry/2020/03/24/021455

TensowFlow-2.8 以降では"-gpu"は付けなくても GPU サポートが入る(ライブラリが統一された)。ここで使うのは 2.5.0 なので、"-gpu"を付けないと gpu サポートが入らない。

インストールした tensorflow が gpu を認識しているかのチェック

(tf253) C:¥Users¥{ユーザ名}>python -c "from tensorflow.python.client import device_lib; print(device_lib.list_local_devices())"

実行結果の中に「device_type: "GPU"」があれば GPU が認識されている。GPU が複数台あれば、”device:GPU:0”、”device: GPU:1”のように複数表示される。エラーメッセージが出ていないことを確認する。

参考：https://www.kkaneko.jp/tools/win/tensorflow2.html

残りのライブラリを入れる。２番目に面倒そうな matplotlib から実施した。

(tf253) C:¥Users¥{ユーザ名}>conda install matplotlib

案の定だめ

参照： https://qiita.com/yinutsuka/items/b3acb7af783c1639bd52

(tf253) C:¥Users¥{ユーザ名}>conda install -c conda-forge matplotlib

さくっと成功。

conda install -c conda-forge は insutall がうまく行かないときの定石らしい。

(tf253) C:¥Users¥{ユーザ名}>conda install Numpy

(tf253) C:¥Users¥{ユーザ名}>conda install pillow

os, glob, random ←これらは python 本体に入ってるので install 不要。

~~(tf253) C:¥Users¥{ユーザ名}>conda install keras~~

(tf253) C:¥Users¥{ユーザ名}>conda install pydot ←単独 keras を外したら必要になった。

学習の plot_model が呼び出している。graphviz(同じく呼び出される)も同時に入る。

(tf253) C:¥Users¥{ユーザ名}>conda install scikit-learn

(tf253) C:¥Users¥{ユーザ名}>conda install h5py ←既に全部あると言ってくるかも。

(tf253) C:¥Users¥{ユーザ名}>conda install opencv

ダメ

(tf253) C:¥Users¥{ユーザ名}>conda install -c conda-forge opencv

成功。インストールは opencv-python としてる記事が多いが、opencv で OK だった。

(tf253) C:¥Users¥{ユーザ名}>python

>>> import cv2 opencv が動作するかの確認、ＯＫ

(tf253) C:¥Users¥{ユーザ名}>conda install pandas

(tf253) C:¥Users¥{ユーザ名}>conda install seaborn

(tf253) C:¥Users¥{ユーザ名}>conda install fabric

(tf253) C:¥Users¥{ユーザ名}>conda install pyperclip

だめ

(tf253) C:¥Users¥{ユーザ名}>conda install -c conda-forge pyperclip

csv ←既にある。import csv で読める。pandas の一部？

(tf253) C:¥Users¥{ユーザ名}>conda install -c conda-forge python-pptx

「モデルアーキテクチャを図示」で pptx の図を描く場合に使用(今回は使ってない。)
参考：　https://github.com/yu4u/convnet-drawer

これでインストールは終了である。

２．準備編

2-1 ディレクトリとソースの用意

まず、作業やデータ置き場のディレクトリを次の通り用意する。ここでは解析のルートディレクトリを{procroot}として、その下に全て配置することにする。実際にはソースコードで指定すれば良いので、ディレクトリ配置に制限があるわけではない。なお、モデル構築後の解析用画像は{procroot}の外に置く。

ソースディレクトリ：　{procroot}¥1_学習する　←３．**学習編**のソース
　内容　a 画像リサイズと Numpy 形式保存.py
　　a 画像リサイズと Numpy 形式保存.bat　←ログ出力するためかぶせた bat
　　b 変換後 Numpy 画像の確認.py
　　c 学習.py
　　c 学習.bat　←c 学習.py の結果をファイル出力するためかぶせた bat
　　cnn_model.py　←モデル構造の記述
ソースディレクトリ：　{procroot}¥A_モデルアーキテクチャを図示
　　make_a_model_image.py
　　config.py
　　convnet_drawer.py
ソースディレクトリ：　{procroot}¥2_分類する　←４．**分類編(4.1,4.2)**のソース
　内容　g 任意の画像データセットの分類 3.py　←１枚を処理する
　　g 任意の画像データセットの分類 3.bat　←指定ディレクトリ内ループ
　　g 画像分類_resultedit.awk　←結果の短縮版を作る。
　　cnn_model.py　←モデル構造の記述
入力データディレクトリ：　{procroot}¥image0x　←学習・評価用
入力データディレクトリ：　Test, hseYYYYMMDD　←テスト、解析用
出力データディレクトリ：　{procroot}¥result
ログディレクトリ：　{procroot}¥logs
図表ディレクトリ：　{procroot}¥figs
ソースディレクトリ：　{procroot}¥3_InvestigateResult　←４．**分類編(4.3)**のソース

内容　g 任意の画像データセットの分類 3.py　←１枚を処理する
g 任意の画像データセットの分類 3.bat　←指定ディレクトリ内ループ
g 画像分類_resultedit.awk　←結果の短縮版を作る。
cnn_model.py　←モデル構造の記述

入力データディレクトリ：　{procroot}¥image0x　←学習・評価用
入力データディレクトリ：　Test, hseYYYYMMDD　←テスト、解析用
出力データディレクトリ：　{procroot}¥resul　←手作業で作っておく。
ログディレクトリ：　{procroot}¥logs　←手作業で作っておく。
図表ディレクトリ：　{procroot}¥figs　←手作業で作っておく。

2-2 データディレクトリに画像を用意する。

データ(a)ー学習用：　{procroot}¥image0x
学習画像＋評価画像 (2017/2018 冬季、5 class×500 枚)
クラス毎サブディレクトリ：　DM　LT　SY　VX　NO
各サブディレクトリに 500 枚ずつの画像(元画像そのまま)を入れておく。
これを人が上手に選ぶと目的に沿ったモデルになるかもしれない。
しかし上手に選びすぎると「過学習」して汎用に使えないかもしれない。

データ(b)ーテスト用：　{procroot}¥Test
学習用を除く分類対象の画像　(2017/2018 冬季、学習用除く 19190 枚)
ここも元画像を入れておけば良い。

この Test ディレクトリを変更することで、異なる冬季など別データの分類ができる。

2-3 実行環境を起動する。

インストールに続けて作業する場合はそのまま作業すれば良く、この項は skip する。
インストール済のＰＣで新たに作業を始める時には次のことだけやれば良い。
Anaconda prompt を起動する。
Tips：　Anaconda Powershell Prompt だと環境の activate がうまくいかなかった。
必要なライブラリをインストールした環境を適用する。

(base) C:¥Users¥{ユーザ名}>conda info -e
conda environments:
#
base　　*　C:¥Users¥{ユーザ名}¥anaconda3
tf253　　C:¥Users¥{ユーザ名}¥anaconda3¥envs¥tf253

(base) C:¥Users¥{ユーザ名}>activate tf253
(tf253) C:¥Users¥{ユーザ名}>

ソースーモデル構築(学習)　のディレクトリ　{procroot}¥学習する　に移動する。

３．学習編

3-1 学習前の前処理

tips：　ソースにパスを書くときバックスラッシュは"¥¥"と記述しないと、ディレクトリや変数の名前によって正しく読めないことがある。"¥"で読めてるのはたまたまエスケープコードでひっかからなかったから。

場所：　{procroot}¥学習する
分類済の学習＋バリデーション(評価)画像(上記(a))を Numpy 形式(.npz)に変換する。
バッチ：　**a 画像リサイズと Numpy 形式保存.bat**
ログ：　{procroot}¥logs¥a 画像リサイズと Numpy 形式保存.log
ソース：　**a 画像リサイズと Numpy 形式保存.py**
　パラメーター：　(現在はハードコーディングされている。)
　　保存するファイル名を決める：
　　outfile ="{procroot}¥¥image0x¥¥photos.npz"
　　各クラスの画像枚数，サイズ
　　max_photo = 500　←学習用に使う各クラスの画像の枚数
　　photo_size = 32　←分類のため粗くする画像のサイズ
処理内容：　画像ファイルを読み込んで Numpy 形式に変換.
　class 毎に読み込むとき random.shuffle(files)して順序をランダムにしている。
　　これを取ると毎回同じ順序の学習用データができる。
　　毎回結果が変わるのはおそらくひとつはここに起因する。
　データを入れるための空リストを用意する。
　　x = [ ] #　画像データ　　y = [ ] #　ラベルデータ
　ディレクトリ，ラベル番号 (ラベル番号は 0 から順番に)を指定してリストに追記
　あとは読んで変換して書き出すだけ。
　　リストは型制限がない→まとめてバイナリ 1 ファイルに書き出し

(tf253) {procroot}¥学習する> **a 画像リサイズと Numpy 形式保存.bat**

processing:　{procroot}¥image0x¥SY
processing:　{procroot}¥image0x¥VX
processing:　{procroot}¥image0x¥LT
processing:　{procroot}¥image0x¥DM
processing:　{procroot}¥image0x¥NO
保存しました: {procroot}¥image0x¥photos.npz 2500
(tf253) {procroot}¥学習する>

出力：　{procroot}¥image0x¥photos.npz

3-2 Numpy 形式で保存した画像を表示する

場所：　{procroot}¥学習する
ソース：　**b 変換後 Numpy 画像の確認.py**
処理内容：　Numpy 形式で保存した画像を確認
入力：　3-1 の出力 ({procroot}¥image0x¥photos.npz)
出力：　{procroot}¥figs¥photos.npz.sample.png　各クラスの画像例を 1 枚に
　出力画像を見ると、"NO"(クラスの番号は 4)カテゴリにかなりの面積の降水がある画像も一部含まれている。手作業分類時に過渡的な状態として"分類無し"をしたものを"NO"とみなしたと思われる。降水なしと広域の降水を含む画像はかなり異なるものであり、おそらくはこれが原因で、検証において一部クラスの F 値がかなり低くなることがある。学習用データを訓練用と検証用にランダムに分けるので、たまたま分けた画像の取り合わせによってこのようなことが起こると思われる。

3-3 モデルに分類済み画像を学習させる

場所：　{procroot}¥学習する
ソース：　**c 学習.py**　**cnn_model.py**　(同じディレクトリに置く)
実行バッチ：　**c 学習.bat**　(**python c 学習.py**　を実行する。)
　tips：　.py を python に関連付けしておくと c 学習.py と打つだけで実行はできるが、こうすると手元のマシンでは CPU が使われた。**python c 学習.py**　と打つと GPU が使われた。どちらが使われているかは画面表示に違いがあるのでわかる。バッチでは GPU が使われるよう **python c 学習.py** と書いた。両方試したところ、CPU では 205.9 秒、GPU では 26.4 秒と約 8 倍の速度差があった(CPU：Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz メモリ 32GB)、GPU:GeForce RTX 2080 Ti グラフィックメモリ 12GB)。
入力：　3-1 の出力 ({procroot}¥image0x¥photos.npz)　np.load で読める。

パラメーター：　(現状、ハードコーディングされている。)

im_rows = 32 # (リサンプリングした、以下同様)画像の縦ピクセルサイズ

im_cols = 32 # 画像の横ピクセルサイズ

im_color = 3 # 画像の色空間

in_shape = (im_rows, im_cols, im_color)

nb_classes = 5

出力：Accuracy と Loss の作図

figaccuracyfile = os.path.join(figdir, "fig_accuracy.png")

figlossfile = os.path.join(figdir, "fig_loss.png")

出力：モデルの可視化

modelfig = os.path.join(figdir, "high_precision_cnn_model.png")

出力：学習済み重みデータファイル名

LearnedWeightData = os.path.join(learndata, "photos-model-light.hdf5")

出力：作成されたモデル

modelzero = os.path.join(learndata, "modelzero.h5")

出力：混同行列の図

figconfusionmatrixfile = os.path.join(figdir, "ConfusionMatrix.png")

処理内容：　CNN モデルに分類済み画像を学習させ、評価してその結果およびチューニングされたモデルを出力する。以下、詳細をソースに従って記述する。

データを読み込む：　photos = np.load(inputfile)　←全画像とその分類ラベルが 1 ファイル

x = photos["x"]　←画像

y = photos["y"]　←分類ラベル(SY、LT など 5 種類に対応する 0 から 4)

学習用と評価用に分ける：　ScikitLearn の train_test_split を使う。

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8)

train_test_split では分ける画像の選択がデフォルトでランダム。固定もできる。

毎回結果が変わるのはおそらくひとつはここに起因する。

x には 5 つの種類(class)について各 500 枚のリサイズ済画像が含まれている。これを学習(train)用約 400 枚と評価(validation)用約 100 枚に分ける。

※評価結果を出力すると、各クラスの枚数がきっちり 400＋100 になっていない上、毎回少しずつ変わる。枚数指定やランダム種の固定もできるが、train_size=0.8 だけでは photos.npz に連結されている全画像から単にランダムに選んでいるだけかもしれない。そうなると、クラス毎に順番に並んでいても同数になるとは限らない。

※ここでやっている、単純に(ランダムに選んで)検証用データを分ける方法はホールドアウト検証(Hold-out Validation)と呼ばれる。もとの英語の意味は「(validation

分を)出さずにとっとく」ということに思える。これと対比して交差検証(Cross Validation)が説明されることが多い。交差検証は、データをいくつかのサブセットに分割してサブセットの一つを検証用とする解析を全サブセットについて繰り返して平均を取る、アンサンブル的な方法である。次のサイトの説明と図がわかりやすい。

参考：https://data-analysis-stats.jp/python/ホールドアウト検証と交差検証/

データ拡張(augmentation)：　回転、反転等により同一特徴の学習用画像を増やす。

ここでやっているのは、-30 度,-25 度, … , 25 度　まで 12 通りに回転×反転(2)。元画像から計 24 枚の画像を作り、置き換える。各クラス 400 枚→9600 枚、5 クラスで計 2000 枚→48000 枚。この処理を"水増し"と訳すサイトが多くあるが語感、意味とも適切に思えない。

使っているのは OpenCV(Open Source Computer Vision Library)というライブラリ(import cv2 する)で、画像や動画を処理するためのものであるが、回転処理の配列を数値で得たりもできて scipy や numpy など他のライブラリと組み合わせやすい。

参考：https://www.tech-teacher.jp/blog/python-opencv/

学習と評価を実行する：

モデルは **cnn_model.py** に記述されており、読み込むだけである。

```
model = cnn_model.get_model(in_shape, nb_classes)
```

学習と評価の実行はこの 1 文だけ。

```
hist = model.fit(x_train, y_train,
        batch_size=512,
        epochs=15,
        verbose=1,
        validation_data=(x_test, y_test))
```

※fit()については TensorFlow 公式サイトに *「これは、データを「batch_size」サイズの「バッチ」にスライスし、指定された数の「エポック」間にデータセット全体を繰り返し反復することによりモデルをトレーニングします。」* とあるが、そもそも用語に慣れてないと何言ってるのかわからない。

◆hist：model.fit の戻り値には学習状況が格納されるので保存する。

◆model.fit：model を x_train, y_train(学習用データ)に fit(学習)させるメソッド(データに結びつけられた関数)。fit メソッドのやってることはバックプロパゲーション(誤差逆伝播法)である。これは、損失(=真の値と出力値との誤差)をフィードバックすることで重みの値を少しずつ更新していく処理である。

◆batch_size：ミニバッチのデータ数。バッチ(ミニバッチ)とは学習データからランダムに抽出した部分データセットのことであり、batch size すなわちデータ数は慣習的に 2^n にされることが多い。この各ミニバッチに対して学習をする方法がミニ

バッチ法である。全ての学習データを一度に入力するより効率面も精度面も有効とされており、CNN(など多層パーセプトロンによる学習)では定番の手法である。1 回の学習につきデータの総数を batch size で割った回数(=ステップ数)だけ重みやパラメーターの更新処理を行い、全ての学習データが使われるようにする(参考書[1])。

epochs：全データを学習する回数

verbose=1：ログを出力する。

validation_data=(x_test, y_test))：validation_data という配列で評価データと正解を与える。こうすると返り値にの hist に epoch 毎の学習データでの評価と validation_data での評価、それぞれの結果が記録される。

このあたりの値の設定や cnn_model.py の記述は、”チューニング”と言われる作業になる。初期値を最適化したり、計算領域を設定したり、計算パラメーターを設定したりと、気象モデルであればモデルの設定とチューニングに相当すると思われる。

ログはコンソール出力される。Epoch 毎に

```
Epoch 15/15
 1/94 [..............................] - ETA: 14s - loss: 0.0427 - accuracy: 0.9922
 2/94 [..............................] - ETA: 13s - loss: 0.0368 - accuracy: 0.9912
 3/94 [..............................] - ETA: 12s - loss: 0.0427 - accuracy: 0.9896
      :
92/94 [============================>.] - ETA: 0s - loss: 0.0459 - accuracy: 0.9864
93/94 [============================>.] - ETA: 0s - loss: 0.0461 - accuracy: 0.9864
94/94 [==============================] - ETA: 0s - loss: 0.0461 - accuracy: 0.9864
94/94 [==============================] - 14s 145ms/step - loss: 0.0461 - accuracy: 0.9864 - val_loss: 0.1013 - val_accuracy: 0.9620
```

という形のログが出力される。その内容は次の通り

92/94(例)：バッチサイズ毎の処理が 94 回あるうちの 92 回目

ETA：estimated time of arrival　終了までの予測時間

accuracy, loss：学習データに対する accuracy,と loss

val_loss, val_accuracy：評価データに対する accuracy,と loss

accuracy,と loss の意味については『補足：cnn_model.py について』を参照

学習済みモデルの評価：

model_fit で validation_data が指定されているので評価済、やらなくて良い。

model_fit 最終行出力と同じものを単独で取り出せる。ここのソースではそれを改

めて print してるのでログとしては見やすい。

score = model.evaluate(x_test, y_test, verbose=1)

出力：　Accuracy= 0.9760000109672546 loss= 0.06677175313234329

Accuracy と Loss の推移を描画：

matplotlib の関数を並べている。→　本項の冒頭で指定したファイルに出力

学習済みモデル構造の書き出し：

model.summary()　→　コンソール→ログ出力

plot_model　→　本項の冒頭で指定したファイルに出力

学習済み重みデータの保存：　今ではこれは必要ない。

学習済みモデルの保存（画像分類に使用する）：　model.save()

出力：　$learndata/modelzero.h5　学習済みモデル

最新版 Keras では，構造，重みを含むパラメーターをまとめて保存可能になった。

混同行列と F 値を表示：　×混合行列　←まれに間違えてるサイトがある。

混同行列とは、２値(2 クラス)分類の場合は、気象学でよく使う予測と実況の(現象あり・なし)カテゴリー検証の分割表そのものである。それが多クラスに拡張されており、混同行列の図のマス目には分類された画像の枚数が入っている。多クラスの場合、下記指標値は対象のクラスを真、他のクラスを偽として計算する。

ただし、用語の定義の仕方が異なるので注意が必要である。

機械学習の混同行列

		class(学習データ)	
		あり (Positive)	なし (Negative)
予測	あり (Positive)	TP	FP
	なし (Negative)	FN	TN

※PとNは予測の現象有無を示す。

※TとFは当たり外れを示す。

←同じ→

←違う→

気象学の分割表

		実況	
		あり(Obser−vation)	なし (X:none)
予測	あり (Forecast)	FO	FX
	なし (X:none)	XO	XX

※FとXは予測の現象有無を示す。

※OとXは観測の現象有無を表す。

表１　機械学習の混同行列と気象予測で用いる分類表の比較

さらに、指標値について機械学習の用語と気象学の用語を比較すると、

機械学習		気象学
適合率(Precision) = TP / (TP+FP)	―	FO / (FO+FX)　(該当用語なし)
再現率(検出率, Recall) = TP / (TP+FN)	←	FO / (FO+XO) = 捕捉率
もしくは感度(Sensitivity)		
偽陽性率(False Positive Rate; FPR)		
= FP / (TN+FP)	←	FX / (FX+XX) = 誤検出率
(TP+FP) / (TP+FN)　(該当用語なし)	―	(FO+FX) / (FO+XO) = バイアススコア

TP / (TP+FP+FN)　(該当用語なし)　　－　　FO / (FO+FX+XO) = スレットスコア

正解率(Binary Accuracy)　　(該当用語なし)

= (TP+TN) / (TP+FP+TN+FN)　－　(FO+XX) / (FO+FX+XO+XX)

のように、対応するものとしないものがある。F 値(F 尺度, F_1score, F-Measures)は適合率(P)と再現率(R)の調和平均であり、

$$F\text{ 値 } = ((1/P+1/R)/2)^{-1} = 2PR/(P+R)$$
$$= 2TP^2 / (TP+FP)(TP+FN) \times (1/ (TP/(TP+FP) + TP/(TP+FN)))$$
$$= 2TP / (2TP+FP+FN)$$

となる。これは気象学分割票の表記では 2FO / (2FO+FX+XO) となり、数式としてはスレットスコアにやや近い。ただしスレットスコアが XX が非常に大きいまれな現象の予報精度を評価することが目的であるのに対して、F 値は FP や FN が大きいものを低く評価して検出し弾くことが目的である。

コンソール出力は、各クラスに対して適合率、再現率、F 値、画像の分類された枚数が出力される。F 値を見て、低ければ適合率と再現率を見て、空振りと見逃しのどちらが多かったか判断できる。

補足：cnn_model.py について

cnn_model.py は 1)Keras のインポート、2)Keras の関数呼び出しで CNN(Convolutional Neural Network; 畳み込みニューラルネットワーク; 機械学習(machine learning)のうち深層学習(deep learning)と呼ばれるものの一種)モデルの構造を記述、3)その記述に従い CNN モデルをコンパイルして返す、の 3 ステップを行っている。気象モデルで言えば、物理過程のスキームや同化のサイクルを決めてコンパイルするのに近い。実際には、親スクリプトから 3)が呼び出され、3)から 2)が呼び出される造りになっている。1)は cnn_model.py の冒頭に記述されている。cnn_model.py はよく似たコードが

https://colab.research.google.com/drive/11YwK8GDlt1KDxd3zBOkfcoqWsMfdvWTH

などに見られ、その大元のソースコードは Github

https://github.com/kujirahand/book-mlearn-gyomu/blob/master/src/ch6/photo_calorie

にあり、おそらく広く使われている。

2)のモデルの構造の書き方は, 使用する深層学習ライブラリによって異なる。2)のモデルの書き方が簡単になっているのは Keras を使用しているからであり、内容は

① model = Sequential() と記す.

② model.add() で層を追加していく.

※ 最初の層だけ input_shape が必要.

だけである。①の場所にはモデル構造の記述方法が記載される。選択肢は

Sequential API：一直線のモデルを構築可能

Functional API：複数の入出力、レイヤーを共有する等のモデルも構築可能

Subclassing API (Model Subclassing)：最も柔軟にモデルを構築可能

である。Sequential()と書くと Sequential API が呼び出され、その構造は

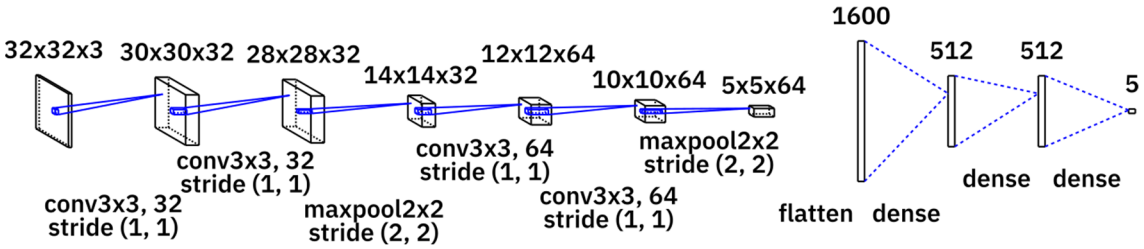

図１　ここで作成したモデルの主要構造

のように描かれる。青の円錐や点線が処理の各ステップ(”層”と呼ばれる)を意味しており、それぞれ②の model.add()で記述されている。直方体や四角は入出力のデータを意味する。各層では、例えばいくつかの入力に対して重み付けしてその和を取ったりしている。

conv*　畳み込み層：　画像なら 2 次元の重み付き移動平均をしている層

pool　プーリング層：　maxpool なら例えば 2x2→その最大値 1x1 に縮小

flatten：2 次元の画像を 1 次元配列に並べ替える。値の変化はない

dense　全結合層：　別名 Affine Layer、すなわち行列による線形写像

‥だったら最初からそう言えと思うのは私だけだろうか？その重みの決め方がキモなのだろうとは思うが。

conv*と書かれているところは Conv2D という関数として実装されている。これは畳み込み積分(要は重み付き移動平均)によって画像(配列データ)を特徴量マップ(feature map)に変換するもので、入力の配列データを図 1 なら 3x3 の 2 次元重み(こちらは機械学習では重みと言わずフィルタやカーネルと言う)による重み付き移動平均を行い、さらに別の重み(こちらを機械学習では「重み」と言う)を掛けて次の層の特徴量マップに出力する。使用するフィルタの数だけ特徴量マップができる。入力側に複数のマップ(チャネル)があれば、それは同じフィルタに別々の重みを掛けたものが足し合わされる。このあたりは次のサイトがわかりやすく(参考：https://www.timedia.co.jp/tech/cnn/)ここからリンクされてる下記 2 サイトも参考になる。

参　考　：　https://www.analyticsvidhya.com/blog/2016/04/deep-learning-computer-vision-introduction-convolution-neural-networks/　ポップアップがうざい。

参考：http://cs231n.stanford.edu/　本家スタンフォードの講義資料

model.add()の引数には呼び出す処理の種類とデータサイズやパラメーター以外にactivation='relu' という”活性化関数”の指定がある。活性化関数とはその層の出力に対するフィルタであり、負の値を 0 に丸めたり(relu；Rectified Linear Unit、ランプ関数；正の値はいじらない)、中間の値の変化を強く反映したり(Sigmoid function；シグモイド関数)、出力する全クラスの総和が 1 になるよう規格化したり(Softmax 関数；$p_i = exp(y_i)/\sum_{j=1}^{C} exp(y_j)$)する関数が用いられている。知らない名前が書かれていていても、形の決まった関数のフィルタ、という点に変わりはない。図 1 の stride という表記は 3x3 や 2x2 を言い換えているだけである。3)のコンパイル時の引数は決まり文句と思って良い。〇〇層という用語は下記サイトがわかりやすい。

参考：https://rightcode.co.jp/blog/information-technology/artificial-intelligence-machine-learning-word

図 1 は 1 枚の画像を学習するときの流れを示すと言えるが、実際の学習時にはミニバッチに含まれる全画像について処理したあと正解との誤差を取り、その誤差を中間の重みの誤差に分配して修正し(誤差逆伝播法)、学習することを繰り返し、誤差が最小となる各層の重みを求める。誤差を各層の出力に分配して出所を見るだけなら配列演算でできるらしいが、重みを修正しなければならないので、出力値と正解値から作られる「誤差関数(何種類かある)」を使用し、誤差関数の傾き(**勾配**)が最小になる補正値を求めて(勾配降下法)その値で重みを修正する。ただしたくさんの重みを整合性を取って補正するので、発散しないように補正値を小さくして反復して求める。詳細は教科書を参照した方が良い。

図 1 は次のスクリプトで描ける。見やすいが、主要な処理のみを表現した図なので、正確なものは「学習済みモデル構造の書き出し」での出力を参照すること。

場所：　{procroot}¥モデルアーキテクチャを図示

ソース：　make_a_model_image.py　convnet_drawer.py　config.py

cnn_model.py を簡略化して記述し、描画パラメーターを設定して、作画している。詳細は、この dir のファイルについて.txt　を参照のこと。

実行：　make_a_model_image.py を走らせる。

svg ファイルができる。png などラスターへの変換は次のようにしてできる。

テキストエディタで修正＋ブラウザで確認→スクショで png

Inkscape などで修正→エクスポートで png

3)ステップ目としてコンパイル済みの CNN モデルを返す次の関数が書かれている。モデルの構造など詳細は def_model を呼び出している。

```
def get_model(in_shape, nb_classes):
    model = def_model(in_shape, nb_classes)
    model.compile(
        loss='categorical_crossentropy',
        optimizer=Adam(),
```

```
            metrics=['accuracy'])
        return model
```

コンパイル時のオプションは次の通り

loss は損失関数(誤差関数)であり、モデルの訓練の backpropagation 時に使われる。ここで選択している categorical_crossentropy は教科書ではクロスエントロピー、交差エントロピーと書かれ、標準的なものと考えれば良さそうである。参考：書籍[2]

metrics はモデルの性能を測るために使われる”評価関数”であり、accuracy は既出の正解率のことである。binary_accuracy、categorical_accuracy というのは 2 クラス正解率、多クラス正解率を意味するだけである。正解率などで多クラスをどう扱うかは既出。

optimizer は上記の「勾配降下法」で重みを更新する際の計算方法の選択である。Adam は現在の標準的なものと思って良さそうである。

参考：https://work-in-progress.hatenablog.com/entry/2019/10/20/095009

参考：https://tensorflow.classcat.com/2016/03/29/keras-optimizers/

参考：https://atmarkit.itmedia.co.jp/ait/articles/1912/16/news026.html

参考：https://keras.io/ja/metrics/

４．分類編

4-1 学習済みモデルを用いて別の画像を分類する

場所：　{procroot}¥分類する

ソース：　g 任意の画像データセットの分類 3.py　処理本体

ソース：　g 任意の画像データセットの分類 3.bat　リダイレクト保存のためのバッチ

ソース：　cnn_model.py　モデルの読み込みに使用

ソース：　g 画像分類_resultedit.awk　出力の短縮版を作成する awk

入力：　g 分類する画像データセット.txt　入力データディレクトリフルパスのリスト

以下は現状ハードコーディング：

入力画像を変換する仕様(32×32 ピクセルにリサイズし, 5 class 分類)

```
im_rows = 32　#　画像の縦ピクセルサイズ
im_cols = 32　#　画像の横ピクセルサイズ
im_color = 3　#　画像の色空間
in_shape = (im_rows, im_cols, im_color)
```

分類のクラス

nb_classes = 5

LABELS = ["SY", "VX", "LT","DM","NO"]

学習済み CNN モデル(3-3 の出力)：　{procroot}¥image0x¥modelzero.h5

model = keras.models.load_model(modelzero,compile=False)

処理内容：　任意の画像セットを学習済モデルに従って分類する

testdata で指定したディレクトリ直下の全画像を走査して処理

各画像(path)について　def check_photo(path):　を呼び出す

もと画像を学習時と同じフォーマットにしてから数値データに変換

画像読み込み(NumPy array or a TensorFlow tensor にする。)

x = x.reshape(-1, im_rows, im_cols, im_color)　#4 次元配列にする必要

x = x / 255　#　値は 0-1 にする必要がある。

pre = model.predict([x])[0]　← # [0]は“データセットの最初の画像”を意味する。1 枚しか処理していないが次元としてこうなる。model.predict が”予測結果(分類結果)”として返すのは、その画像が分類しようとしているクラスのそれぞれに該当するかどうかの「確信度(=信頼度, confidence)」だけである。

参考：https://www.tensorflow.org/tutorials/keras/classification?hl=ja

idx = pre.argmax()　# idx は最も確信度が高いクラスの番号

per = int(pre[idx] * 100 + 0.5)　# per は idx の確信度を%表記したもの

for i in range(len(LABELS)):

print(LABELS[i], ">>>",pre[i]*100)

#　全クラスに対するこの画像の確信度を出力

実行すると少し待たされる。g 任意の画像データセットの分類.bat で動かした場合は、全部リダイレクトしてて終わるまで無言となる。

出力：　{procroot}¥result¥g_(入力データディレクトリ)_の分類_result.txt.gz

内容は次の通りで、入力データディレクトリ内の画像の分類結果と各クラスの確信度を列挙する。gz 圧縮されている。

：

****

SY >>> 0.7798598147928715

VX >>> 5.111904814839363

LT >>> 88.92003893852234

DM >>> 3.3602721989154816

NO >>> 1.8279245123267174

==========>> Image = LT　←確信度が一番大きいもの

==========>> 88 % confidence　←一番大きいものの値を丸めてる

==========>> < pr2hse20440.png >　←その画像ファイル名

：

のように、入力データディレクトリ内の画像の分類結果と各クラスの可能性を列挙する。gz 圧縮されている。

{procroot}¥result¥g_(入力データディレクトリ)_の分類_result.txt.gz

上の内容を分類結果、ファイル名、確信度のみの 1 行に短縮したもの。

処理速度：　現在のソースコードでは使用した PC において GPU 利用を on/off しても時間差はほぼなかった。約 19000 枚の分類処理おいて GPU 利用 on: 679.4 秒、GPU 利用 off: 687.6 秒であった。どちらも GPU を使っていないかもしれない。

4-2 分類結果の評価：混同行列と F 値

学習＋検証用データと同様にデータを作れば、これらも Keras の中で算出、作図できると思うが、今ここでは、学習済みのモデルを他の 10 年分のデータに適用したい。現時点では、それらに対して Keras 向け前処理をするよりは 4-1 の出力を Excel 手作業した方が早い。

入力：　{procdir}¥result¥g_* _の分類_short.txt

作業結果の例：　4-2 分類結果の評価：混同行列と F 値_例：_20172018.xls

混同行列や F 値は、分類結果が正答かどうかについてのものである。確信度は使っていない。

4-3 Grad-CAM で画像のどこが分類に強く反映したかを調査する

これについては、入力画像のディレクトリを指定して一括で描かせるスクリプトを示す。

ソース１：　dGradCam3.py　←heatmap を元画像に重ね書きしたものを出力する。

ソース２：　dGradCAMnumout.py　←heatmap の数値を出力する。

CAMnpzplot.py　←heatmap の数値を読み、元画像に重ね書き出力する。

ソース３：　CAMaveplot.py　　←heatmap の数値を読み、平均と標準偏差を描画出力する。

※これは元画像との重ね書きはしない。

パラメーター：　dGradCAM3_input.txt　入出力ディレクトリパス指定、dGradCam3.py 用

dGradCAMnum_input.txt　同上、dGradCAMnumout.py 用

CAMnpzplot_input.txt　同上、CAMnpzplot.py 用

CAMaveplot_input.txt　同上、CAMaveplot.py 用

入力は、指定したパス直下に全ての画像があるものとする。

出力：　tmp.dGradCAM3_datalist.txt　入力ファイルのリスト

出力：　CAM 画像(*CAMx.png)　x は分類するクラスの番号

このディレクトリ直下に全ての画像が出力される。

dGradCam3.py と dGradCAMnumout.py＋CAMnpzplot.py　の出力は一致することを確認している。dGradCAM3.py は、よくあるヒートマップと元図の重ね合わせスクリプトである。GradCAMnumout.py は、heatmap の数値をそのまま npz 保存するようにしたもので、CAMnpzplot.py はそれを単純に読みだしてプロットするスクリプトである。なお、CAMnpzplot.py では y 軸が上から下になっている。すなわち、画像から変換した配列は画像のピクセル配置(y 上→下)　を反映して並んでおり、dGradCAM3.py でもこれは同じである。

以上のソースは任意のディレクトリを指定できるが、サンプルデータについては目視解析結果があるので、それらをサブディレクトリに振り分けてから上記の処理を行った。振り分けのスクリプト等は次のものである。

ソース：　dGradCAMnumouk.bat　dGradCAMbunrui.awk

入力：　hse20152016.tgz　2015/2016 冬季の元画像　解凍してから作業する。

入力：　periods4_20152016_DL_bunrui.csv

←hse20152016.tgz の画像について、目視とモデルの結果を並べたリスト

出力：　入力画像を指定のとおり振り分けた別のディレクトリにコピーする。

処理内容：　Grad-CAM でモデルが画像のどの部分の特徴を拾い出したかを見る

以下、dGradCAM3.py について順に内容を述べる。dGradCAMnumout.py＋CAMnpzplot.py は数値出力が得られることが異なるのみであり、CAMaveplot.py はディレクトリ内全画像についての結果を min_max(np.mean(heatmap))と min_max(np.std(heatmap))しているのみである。

学習済みモデルを読み込む

model=load_model("{procdir}¥¥image0x¥¥modelzero.h5",compile=False)

保存されているのは、

モデルの構成(アーキテクチャ)

モデルに含まれるレイヤー、およびこれらのレイヤーの接続方法

モデルの重み値のセット（トレーニング時に学習される）

モデルのコンパイル情報（compile（）が呼び出された場合）

オプティマイザとその状態(存在する場合)

学習を途中から再開するために使用

である。保存形式が 2 種類あり、ここでは古いが軽量な”h5 形式”を使っている。

この形式では一部保存されない情報があるが、現時点では気にしなくて良い。

参考：　https://www.tensorflow.org/guide/keras/save_and_serialize?hl=ja

必要な数値を取り出すための関数を記述する。

モデルの最終出力を取り出す

model_output = model.output[:,0]

このモデルの最終出力の形状は(None, 5)で、末尾には分類するクラスの番号である 0 番目[:, 0]から 4 番目[:, 4]のどれかを指定する。クラスは LABELS = ["SY", "VX", "LT","DM","NO"]　のように決めてある。これを出力するモデルの最終層は

model.add(Dense(nb_classes, activation='softmax'))

と記述してあり、5 クラスのそれぞれに分類される可能性の高さの数値を全クラスの和が 1 になるよう規格化して、確信度として出力する。

最後の畳込み層を取り出す

last_conv = model.get_layer("conv2d_3")

層の名前は、c 学習.log の最後の方に出力してあるのを見れば良い。

last_conv はこのあと last_conv.output としてしか使っていない。すなわち、図 1 中程の”Conv3x3,64”と書かれている層の出力(形状は 10x10x64)を取り出しているのみである。この層は cnn_model.py ではいくつかある

model.add(Conv2D(64, (3, 3), activation='relu'))

の最後のものが対応する。要は、

conv2d_3 の 10x10x64 出力　と　モデル最終出力の確信度[0:4]

の勾配(次元は 5x10x10x64)を求めて加工している。最終出力の一つ前の dense ではなく conv2d_3 を使うのは、flatten で位置情報が失われるので「.. we can expect the last convolutional layers (最後の畳み込み層) to have the best compromise between high-level semantics (分類の特徴) and detailed spatial information.」とのこと。

Grad-CAM で何と何の間の勾配を計算しているかの数式は、

https://zenn.dev/iq108uni/articles/7269a1b72f42be

が一番わかりやすく、結果だけ書くと、

$$M_c(x,y) = \sum_k \alpha_c^k A_{xy}^k$$

$$\alpha_c^k = \frac{1}{Z} \sum_{x,y} \frac{\partial S_c}{\partial A_{xy}^k}$$

である。ソースコードと対応させると、S_c が model.output[:,0:4]、$A^k{}_{xy}$ が model.get_layer("conv2d_3").output、$\alpha^k{}_c$ が pooled_grads、$M_c(x,y)$が heatmap である(ただし、RELU 関数をかませて負値は 0 にする)。$\partial S_c / \partial A^k{}_{xy}$ が出力(分類のクラス)と特徴量マップの値との勾配である。この勾配は cnn_model の説明に記載したように誤差関数を用いて求められるようだが、このあたりをきちんと追って書いた資料が今見つからない。

なお、下記のサイトを参考にした。

参考：https://qiita.com/yakisobamilk/items/8f094590e5f45a24b59c

参考：https://tech.jxpress.net/entry/2018/12/12/130057

参考：https://zenn.dev/iq108uni/articles/7269a1b72f42be

参考：https://betashort-lab.com/データサイエンス/ディープラーニング/grad-cam/

この「勾配」を計算するため、dGradCam3.py ではこのあと

grads = K.gradients(model_output, last_conv.output)[0]

pooled_grads = K.mean(grads, axis=(0, 1, 2))

iterate = K.function([model.input],[pooled_grads, last_conv.output[0]])

のように何行かに分けて関数を定義している。K は

import tensorflow.keras.backend as K

で import しておいた「Keras バックエンド」という下位レベル関数呼び出しライブラリである。ここで使っている Keras バックエンドの関数は次の通り

function(inputs, outputs, updates=None)　* =以後はデフォルト値

inputs はテンソルのリストで、書かれた input に紐付けられた関数が実行され、outputs に指定されたテンソル(Numpy 配列)のリストを返す処理を定義する。inputs に書かれている[model.input]は仮引数で、呼び出し時には入力画像が指定される。ただし元画像ではなく c 学習.py の入力 photos.npz と同じ形まで変換したものを指定する。

outputs もテンソルのリストで、中身を書き下すと K.mean(K.gradients(ある分類クラスの確信度, Conv3x3,64”と書かれている層の出力)[0], axis=(0, 1, 2)) という内容になる。

updates は「更新する命令のリスト」。ここでは無指定なのでデフォルトの None。あまり使われていないかもしれない。

gradients(loss, variables)

variables の loss に関しての勾配テンソルを返す。loss は最小化するためのスカラーからなるテンソルとされている。変数の意味は必ずしも「損失(誤差)」でなくても良いように思う。

mean(x, axis=None, keepdims=False)

mean は入力テンソル x の平均を求めるもので、axis=で平均する軸の方向を指定できる。keepdims を指定していないので、平均した分次元の数も減ったテンソルを出力する。

以上定義した iterate = K.function が「勾配」の計算の全てである。その出力は

pooled_grads :　中身を書き下すと K.mean(K.gradients(ある分類クラスの確信度, “Conv3x3,64”と書かれている層の出力)[0], axis=(0, 1, 2)) となり、その配列形状は (64,) すなわち 1 次元 64 要素の配列である。

last_conv.output[0] :　“Conv3x3,64”層の出力。その配列形状は (10, 10, 64) 、すなわち 10x10 格子の特徴量マップ(feature map)64 種類となって

いる。特徴量 64 種類は 64 チャネルとも言う。雲や雪面を多波長の放射計で見ているようなものと思えば、理解できるかもしれない。

参考：

get_layer など Keras のメソッド(データに結びつけられた関数)はここにまとまっている。→ https://keras.io/ja/models/model/

Keras バックエンドの説明はここ → 参考：https://keras.io/ja/backend/

Grad-CAM の原著論文：https://arxiv.org/pdf/1610.02391v1.pdf

Web サイトにはやたらと「インスタンス」が出てきていろいろたとえで説明されているがよくわからない。「インスタンス」は「関数呼び出し」あるいはインタフェースと考えた方が良さそう。

heatmap を求める。

元画像を読み込んで c 学習.py の入力 photos.npz と同じ形のテンソルに変換する。

詳細前項。

入力画像に対する最終畳込み層出力の値と勾配を求める

pooled_grads_val, conv_output_val = iterate([img_tensor])

これ 1 行のみ。詳細は関数定義で書いてるので。前項参照。

Iterate は「あるクラスに対する確信度」を出力する。すなわち、モデルが回答として選択しなかったクラスの確信度も得られる。dGradCAM3.py ではそれも含めて全ての確信度について元画像のどの部分の特徴を拾い出したか作図する。

最終畳込み層出力の値に平均勾配をかけて、もう一度チャネル方向に平均

```
for i in range(64):
```

conv_output_val[:, :, i] *= pooled_grads_val[i]

最終畳込み層出力　　平均勾配

10x10 の特徴量配列 x64　1 次元 64 個

heatmap = np.mean(conv_output_val, axis=-1)　←これが求めるヒートマップ

データとしての heatmap はここまでで得られる。

ここまで見てくると、Web サイトで見た次の説明がだいたい理解できる。2)については「勾配」を誤差逆伝播で算出している旨書かれているが、同様な記述が他の資料になく、もう少し裏付けを取りたいところである。

Grad-CAM とは【Gradient-weighted Class Activation Mapping】の略で、一言で要約すると予測値に対する勾配を重み付けすることで、重要なピクセルを可視化する技術です。勾配が大きいピクセルは予測値に大きな影響を与える＝重要という発想です。(中略)　Grad-CAM の計算のワークフローは以下のようになります。

1)　分類のときと同様に画像を入力し、最終畳み込み層の出力と分類結果を取り出す

2) 分類結果から誤差逆伝播で確信度－最終畳み込み層間の勾配を計算する

3) 最終畳み込み層の勾配値を特徴量毎に平均したスカラー(GAP)にする。

4) GAP 重み付きの畳み込み層の和を求め、元の画像に合わせて重ねて描画する。

出典：https://dajiro.com/entry/2020/06/26/234720(元論文のリンクもある。)より改変

---

heatmap を後処理して元画像にスーパーインポーズしてファイルに書き出す

ここから先は可視化出力処理である。元画像に合わせてサイズ変更したり、0-255 にしたりしているが、変数の型変換を伴うので改修する場合は処理の順序など注意が必要である。

radCAM3.py では分類として選択されなかったクラスの確信度(結果としては捨てられる値)も描画しているので、2 番目に確信度の高かった分類ではどこの特徴を拾っていたか調べることも可能である。その場合は、別途出力された確信度と手作業で突き合わせる必要がある。ファイルハンドリングとしては、単一ディレクトリ直下に出力クラス番号を付加した全画像を出力している。あまり多量の画像の処理は(できるが)想定していない。予めピックアップする画像を選んで入力ディレクトリに置いて処理した方が良い。

処理速度： 現在のソースコードでは使用した PC において GPU 利用を on/off しても時間差はほぼなかった。dGradCAM3 で入力 77 枚(出力 77x5 枚)で計測比較したところ、GPU 利用 on: 719.6 秒、GPU 利用 off: 748.4 秒であった。どちらも GPU を使っていないかもしれない。

おわりに

このドキュメントは、とにかく最小限のディープラーニングを数十万円のデスクトップで何とか動かして、降水強度分布を題材になにがしかの結果を出して、それを曲がりなりにも理解するための操作及び処理内容の説明書である。ドキュメントとともに作成されたソースコードでは入力画像の解像度を落とした処理になっており、そこそこの速さのデスクトップであれば、練習として現実的な速さでひととおりの動作をさせることができる。ただし、CNN モデルの最適化はできていない。また、RNN や VGG といったより実用的そうなモデルについても書いていない。しかし、ソースコードが何をやっているか、また実際の落とし穴についても気がついた限り記載した。主に気象分野の読み手を想定しており、気象学でなじみのある用語を多用したり、具体的な比較をしたりもした。このドキュメントを見ながら一通り処理をすれば、少なくとも CNN に「なじむ」ことが比較的短時間でできると思う。導入～用語に慣れるまでに使う時間はできるだけ少なくしたいものである。

筆者は毎年気象庁全国合成レーダーデータを作図し、北信越地方の降雪系分類を手作業で行っていた。これは１冬季分やろうとすると、かなり慣れても正味でのべ２～３日間フルに必要であり、集中して作業できる時間をまとめてこれだけ確保するのは今では容易ではない。これを自動化したいというのが CNN を使い始めた動機の一つである。このような手

作業の自動化、あるいは解析の始めに当たりを付けるといった用途のツールとして機械学習の環境とスキルを手に入れれば、本来の気象学的な研究を加速できるのではないか、そう期待してもいる。

現時点でのこのドキュメントと連動したソースコードは、人手の解析の再現性としてはまだまだであるが、CNN の最適化とは別に、用意する画像の方をうまく作るのも再現性を上げるもうひとつの方法だろうとも思った。元はデータなので、よくある AI 画像分類のサンプルとは異なり、分類しやすいように画像を作ることも可能なはずである。

このドキュメントは不十分なところも多々あり、誤りなどもおそらくある。しかし、機械学習に取りかかるときのハードルを下げるために、また、機械学習を自分の研究ツールの一つとしてどう使うか検討するために、たたき台になれば幸いである。

筆　者