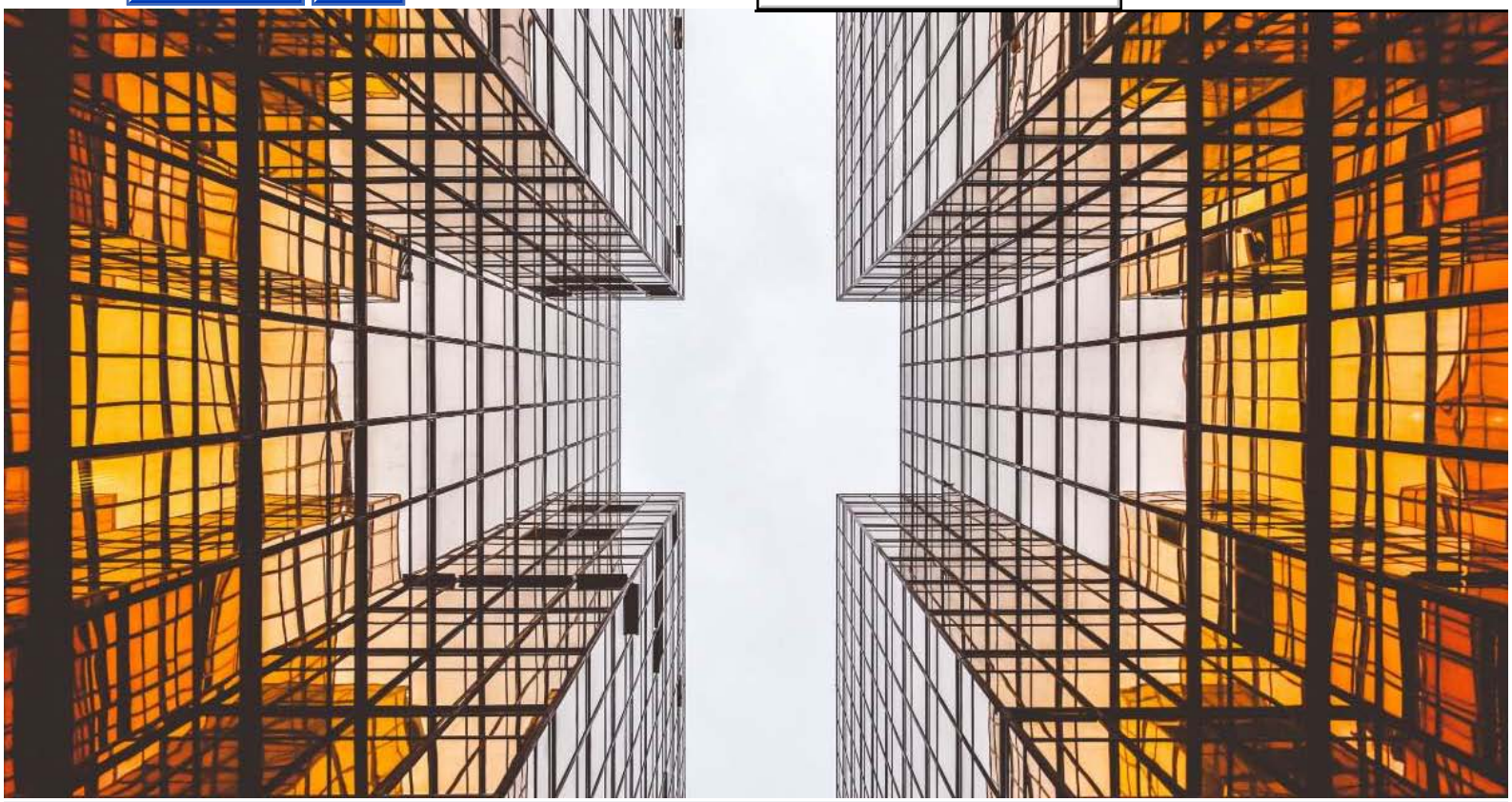
【高精度な画像分類器作りに挑戦！】(1)ゼロからCNN

tmtk



Topics

こんにちは。データサイエンスチーム tmtkです。
今回から、**写真を分類する機械学習モデルを作成**する手順を3回にわたってご紹介します。
この記事では、桜とコスモスを分類する**畳み込みニューラルネットワーク**(**CNN**)をゼロから作成します。

はじめに

以前の記事で、Microsoft AzureのCustom Vision Serviceを紹介しました。Custom Vision Serviceは、訓練データを自分で用意することで、画像の分類器を自動的に作成してくれるサービスです。記事では、Custom Vision Serviceに桜の写真とコスモスの写真を10枚ずつ訓練データとして与え、精度100%の分類器を作成しました。
この記事では、**Keras**や**TensorFlow**といったディープラーニング用ライブラリを使って、桜とコスモスの写真を分類する機械学習モデルを、ゼロから作成します。既成のサービスでよい精度が発揮できている場合で

も、原理・原則をふまえて自力で対処できるようになることで、既成のサービスでは対応できない問題も解決できるようになることが狙えます。
TensorFlowは最も有名なディープラーニング用フレームワークの一つであり、KerasはTensorFlow上の高レベルAPIとして使用できます。Kerasを使うことで、簡単に実験が進められます。

準備

セットアップを簡単にするため、この記事ではコンピューティング環境として**Amazon EC2**を使うことにします。
まず、Amazon EC2から、今回使うためのインスタンスを立ち上げます。AMIは「**Deep Learning AMI (Ubuntu) Version 6.0**」を選択し、インスタンスタイプは「**p2.xlarge**」を選択します。
P2インスタンスはディープラーニングなどGPUコンピューティング向けに設計されており、今回作成する機械学習モデルの学習を高速に行うことができます。Deep Learning AMIにはKerasやTensorFlowなどのソフトウェアがあらかじめインストールされています。
次に、立ち上げたインスタンスにSSH接続し、以下の手順でセットアップをおこないます。

訓練データの準備

以前の記事で使ったものと同じ桜とコスモスの写真を使います。これを仮想マシンにダウンロードしておきます。「**/home/ubuntu/sakura/**」と「**/home/ubuntu/cosmos/**」以下にそれぞれ15枚ずつ配置します。（桜1 , 桜2 , 桜3 , 桜4 , 桜5 , 桜6 , 桜7 , 桜8 , 桜9 , 桜10 , 桜11 , 桜12 , 桜13 , 桜14 , 桜15 ,コスモス1 , コスモス2 , コスモス3 , コスモス4 , コスモス5 , コスモス6 , コスモス7 , コスモス8 , コスモス9 , コスモス10 , コスモス11 , コスモス12 , コスモス13 , コスモス14 , コスモス15 ）

環境の有効化

KerasとTensorFlowをつかうため、**環境を有効化**します。

```
source activate tensorflow_p36
```

また、画像処理ライブラリ**Pillow**と機械学習ライブラリ**scikit-learn**をインストールします。PillowはKerasから画像を読み込むために、scikit-learnは訓練データとバリデーションデータの分割のために使います。

```
pip install pillow scikit-learn
```

学習

準備が整ったので、いよいよ機械学習モデルの作成に入ります。
IPythonを起動します。

```
ipython3
```

再現性確保のため、Kerasのドキュメント を参考にして乱数のシードを固定します。

```
import numpy as np
import tensorflow as tf
import random as rn
import os
```

```
from keras import backend as K
os.environ['PYTHONHASHSEED'] = '0'
np.random.seed(0)
rn.seed(0)
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
tf.set_random_seed(0)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

次に、データを読み込み、訓練データとバリデーションデータを準備します。

```
from keras.preprocessing import image
from sklearn.model_selection import train_test_split
import keras
import numpy as np
import os
input_shape = (224, 224, 3)
batch_size = 128
epochs = 100
num_classes = 2
x = []
y = []
for f in os.listdir("sakura"):
    x.append(image.img_to_array(image.load_img("sakura/"+f, target_size=input_shape[:2])))
    y.append(0)
for f in os.listdir("cosmos"):
    x.append(image.img_to_array(image.load_img("cosmos/"+f, target_size=input_shape[:2])))
    y.append(1)
x = np.asarray(x)
x /= 255
y = np.asarray(y)
y = keras.utils.to_categorical(y, num_classes)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state= 3)
```

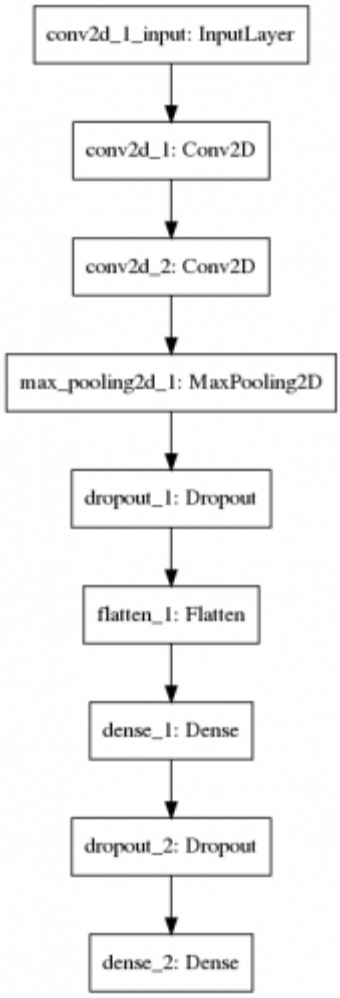
最後に、小さな畳み込みニューラルネットワークを構築し、訓練データを使って学習させます。このネットワークはKerasのMNIST用の例 を参考にしています。

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
```

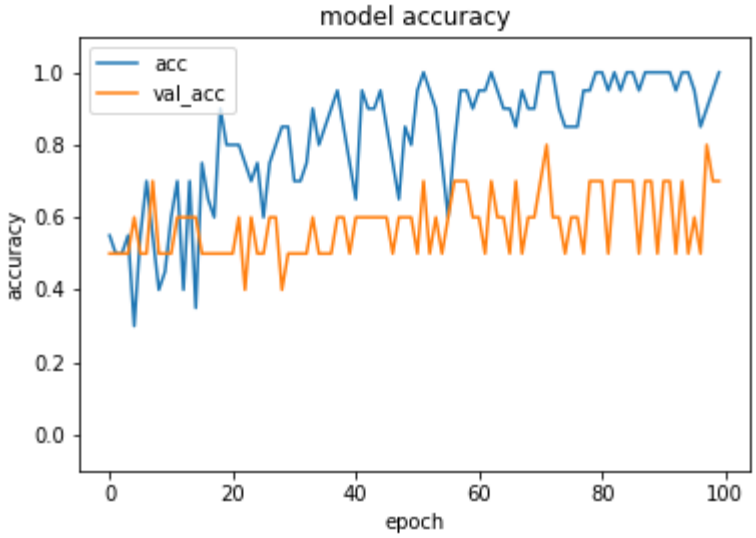
```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer="SGD",
metrics=['accuracy'])
history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

すると、以下のように学習が進みます。数分程度で学習は完了します。

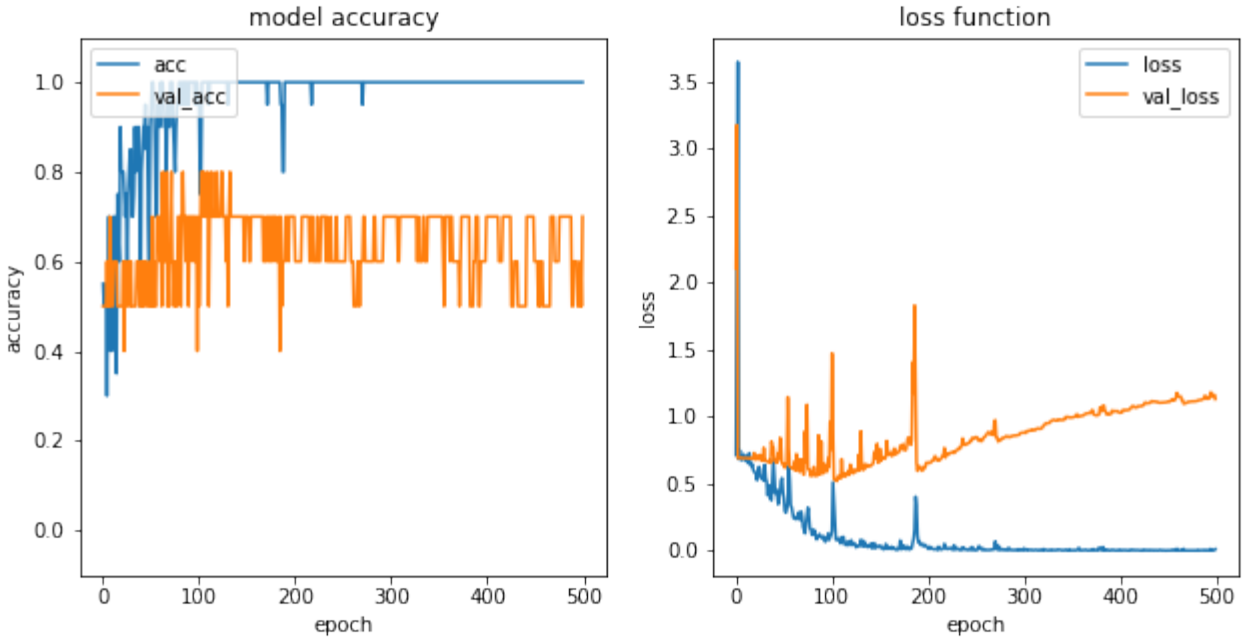
```
Train on 20 samples, validate on 10 samples
Epoch 1/100
20/20 [==============================] - 3s 147ms/step - loss: 0.7121 - acc: 0.5500 - val_loss: 2.105
6 - val_acc: 0.5000
Epoch 2/100
20/20 [==============================] - 0s 15ms/step - loss: 2.3287 - acc: 0.5000 - val_loss: 3.1728
- val_acc: 0.5000
Epoch 3/100
20/20 [==============================] - 0s 15ms/step - loss: 3.6446 - acc: 0.5000 - val_loss: 0.6903
- val_acc: 0.5000
...
Epoch 98/100
20/20 [==============================] - 0s 15ms/step - loss: 0.3436 - acc: 0.9000 - val_loss: 0.5973
- val_acc: 0.8000
Epoch 99/100
20/20 [==============================] - 0s 15ms/step - loss: 0.1847 - acc: 0.9500 - val_loss: 0.5408
- val_acc: 0.7000
Epoch 100/100
20/20 [==============================] - 0s 15ms/step - loss: 0.1024 - acc: 1.0000 - val_loss: 0.5448
- val_acc: 0.7000
```



（今回作成した畳み込みニューラルネットワークのネットワーク構造）
学習した結果、最終的に**訓練データに対する精度が100%、バリデーションデータに対する精度が70%**になっています。それぞれの精度の推移を可視化するとわかるように、**過学習の兆候が見られます**。これは訓練データが10枚/クラス × 2クラス = 20枚と、非常に少ないために起こっていることです。



（訓練データとバリデーションデータに対する精度の推移）
エポック数を5倍の500に増やすと、**過学習**の傾向がより顕著に観察できます。100エポック目では70%程度あったバリデーションデータに対する精度が、500エポックが経過するころには**60%前後に下がっています**。また、訓練データに対する損失関数の値は下がり続けていますが、バリデーションデータに対する**損失関数の値が**100エポックくらいを境に**逆に上がり始めています**。



（学習を続けると過学習が顕著になる）

まとめ

この記事では、桜とコスモスを分類する畳み込みニューラルネットワークを作成しました。
訓練データに対する精度は100%を達成しましたが、訓練データが非常に少ないために過学習を起こし、バリデーションデータに対する精度は70%ほどにとどまりました。
次回の記事では、ファインチューニングという技法を使って機械学習モデルを作成します。ファインチューニングによって、学習が高速化し、過学習が抑えられることを観察します。

参考

- だれでも簡単に画像の分類ができる！Microsoft AzureのCustom Vision Serviceとは？ | DATAHOTEL Tech Blog | NHN テコラス株式会社
- Keras Documentation
- TensorFlow
- Keras – Deep Learning AMI
- Pillow — Pillow (PIL Fork) 5.1.0.dev0 documentation
- scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation
- keras/mnist_cnn.py at master · keras-team/keras · GitHub

#Amazon EC2#Custom Vision Service#Keras#Microsoft Azure#Python#TensorFlow#ニューラルネッ

データ分析と機械学習とソフトウェア開発をしています。 アルゴリズムとデータ構造が好きです。

Recommends

こちらもおすすめ

インチューニングで高精度

2018.4.26

Data Science

S

Tech

GCPの利用料が安くなる|GCPの請求代行・運用代行・導入移行支援AWS
アの

2020.5.18

GCP2020.3