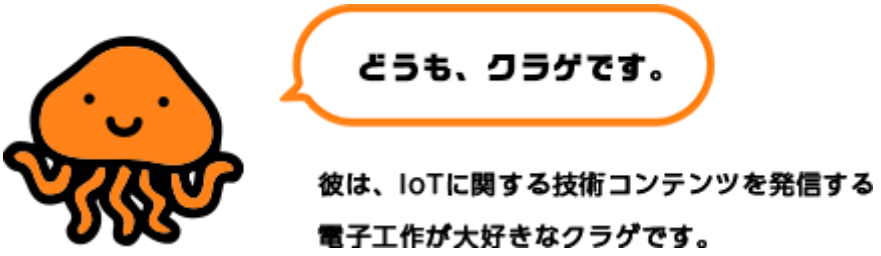
JELLYWARE

MENU

2019/07/22

NumPyプログラミングを学ぶ



どうも、ディープなクラゲです。

「OpenVINO™ でゼロから学ぶディープラーニング推論」シリーズの６回目記事です。

このシリーズは、ディープラーニング概要、OpenVINO™ツールキット、Neural Compute Stick、RaspberryPiの使い方、Pythonプログラミングをゼロから徹底的に学び、成果としてディープラーニング推論アプリケーションが理解して作れるようになることを目指します。

第６回目はNumPyを学びます。

「OpenVINO™ でゼロから学ぶディープラーニング推論」で使うNumPyに絞ってじっくりと説明してゆきます！

前半は配列としての基礎、後半は画像データとの関連について学びます。

【 目次 】

NumPyとは

NumPyとは数値計算を効率的に行うための拡張モジュールです。簡単に言うと、ベクトルや行列などの多次元配列を高速に計算するためのライブラリです。また画像データとも密接な関係があります。
OpenCVと同様に、NumPyを使用するためには`numpy`モジュールを`import`する必要があります。

```
import numpy
```

上記の`import`でも全く問題ないのですが、慣例的に `np` という別名を付けることが一般的です。

```
import numpy as np
```

ndarray

NumPyで扱う多次元配列は ndarray という名称です。（N-dimensional array の略）
ndarrayはPython基礎で習ったリストと関連性があります。リストと表

記が似ていて混同しやすいので注意しましょう。

ちなみに、プログラムのソースコードには ndarray という関数は出てきません。

```
import numpy as np

a = [10, 20, 30]
b = np.array(a)
print(b)

# 実行結果
# [10 20 30]
```

上記のソースコードの解説です。

変数 `a` はPython基礎で習ったリストです。

変数 `b` がndarrayです。 `np.array` という関数を使ってリストをndarrayに変換しています。

最後に `print` で表示していますが、ndarrayはリストと異なり、カンマが無いのが特徴です。

初期化

ndarrayの初期化方法です。既に先ほどのコードで出ていますが、 `np.array` を使って行います。

ここでは、具体的にリストとndarrayの違いを比較しながら確認してゆきましょう。

実行結果を見ると分かりますが、リストにはカンマがついていますが、ndarrayにはカンマが無いのが確認できます。

```
import numpy as np

#リスト
a = [10, 20, 30]
print(a)

#ndarray
b = np.array([10, 20, 30])
print(b)

# 実行結果
# [10, 20, 30]
```

```
# [10 20 30]
```

２次元の場合の比較です

ndarrayの結果は途中で改行が入っています。

```
import numpy as np

#リスト
a = [[10, 20, 30], [40, 50, 60]]
print(a)

#ndarray
b = np.array([[10, 20, 30], [40, 50, 60]])
print(b)

# 実行結果
# [[10, 20, 30], [40, 50, 60]]
# [[10 20 30]
# [40 50 60]]
```

要素参照

要素参照についても、具体的にリストとndarrayの違いを比較しながら確認してゆきましょう。

１次元の場合の要素参照はリストもndarrayも同じです。

```
import numpy as np

#リスト
a = [10, 20, 30]
print(a[1])

#ndarray
b = np.array([10, 20, 30])
print(b[1])

# 表示結果
# 20
# 20
```

２次元の場合、表示結果は同じですが、参照方法が異なることに注目してください。

実はndarrayも `b[1][2]` という書き方でもOKなのですが、ndarrayの場合は慣例的に `b[1, 2]` という書き方をします。

```
import numpy as np

#リスト
a = [[10, 20, 30], [40, 50, 60]]
print(a[1][2])

#ndarray
b = np.array([[10, 20, 30], [40, 50, 60]])
print(b[1, 2])

# 表示結果
# 60
# 60
```

最大値のインデックス

`np.argmax` は 配列（リストやndarray）の中で最大値の要素のインデックスを値で返します。

```
import numpy as np

a = np.array([10, 50, 40, 30, 20])
b = np.argmax(a)
print(b)

# 表示結果
# 1
```

配列の中の最大値は `50` ですが、そのインデックス `1` であるため、このような実行結果になります。最大値そのものではなく、最大値があるインデックスを返すというのがポイントです。

形状

ndarrayは「次元」以外の要素として「形状」があります。形状は各次元における要素の数のことです。

`shape` を使うと、形状を表示させることができます。

まずは 1 次元の例です。

実行してみると、中途半端なカンマで終わっていますが、要素数が `4`

ということで合っていると思います。１次元の場合はこのような `(x,)` という中途半端なカンマで表示されます。

```
import numpy as np

a = np.array([1, 2, 3, 4])
print(a.shape)

# 表示結果
# (4,)
```

今度は２次元の例です。実行してみると `(3, 4)` ということで一致しています。

```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(a.shape)

# 表示結果
# (3, 4)
```

次に３次元の例。長いので途中で改行しました。結果は `(2, 3, 4 )` で一致しています。

```
import numpy as np

a = np.array([[[ 1,  2,  3,  4], [ 5,  6,  7,  8], [ 9, 10, 11, 12]],
              [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]])
print(a.shape)

# 表示結果
# (2, 3, 4)
```

ndarrayと画像データ

冒頭で述べたようにndarrayと画像データは密接な関係があります。これまで使ってきた変数と画像の型を比較したいと思います。

`type` 関数を使うと、変数の型を見ることができます。

```
import numpy as np

a = 100
```

```
b = 3.14
c = 'jellyfish'
d = [10, 20, 30]
e = np.array([10, 20, 30])

print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))

# 表示結果
# <class 'int'>
# <class 'float'>
# <class 'str'>
# <class 'list'>
# <class 'numpy.ndarray'>
```

整数、小数、文字列、リスト、ndarrayと表示されていると思います。

次にOpenCVで使った `cat.png` を `cv2.imread` して型と形状を見てみましょう

```
import cv2

image_file = 'cat.png'
img = cv2.imread(image_file)

print(type(img))
print(img.shape)

# 表示結果
# <class 'numpy.ndarray'>
# (600, 800, 3)
```

実は `cv.imread` で読み込まれた画像はndarrayに変換されていました。

そして形状をみると `(600, 800, 3)` であり、(高さ, 幅, カラーチャンネル)を示しています。カラーチャンネルは 青と緑と赤の3色です。

個別に画像の幅を取得したい場合は形状の0番目の要素、高さを取得したい場合は形状の1番目の要素を指定すれば得られます。

```
import cv2

image_file = 'cat.png'
img = cv2.imread(image_file)

print(img.shape[0])
print(img.shape[1])
```

```
# 表示結果
# 600
# 800
```

スライス

ndarrayにて `[n:m]` を使うとndarrayから条件にあった要素全てを取り出すことができます

取り出される要素は `n` ～ `m-1` 番目です。 `m-1` というのがちょっとややこしいですね。

実際のプログラムで確かめてみましょう。

```
import numpy as np

a = np.array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
print(a[3:8])

# 表示結果
# [30 40 50 60 70]
```

先程のコードは1次元のスライスでしたが、2次元や3次元でも同様に使うことができます。

スライスを画像データに適応すると、画像の1部分だけの取得が可能になります。

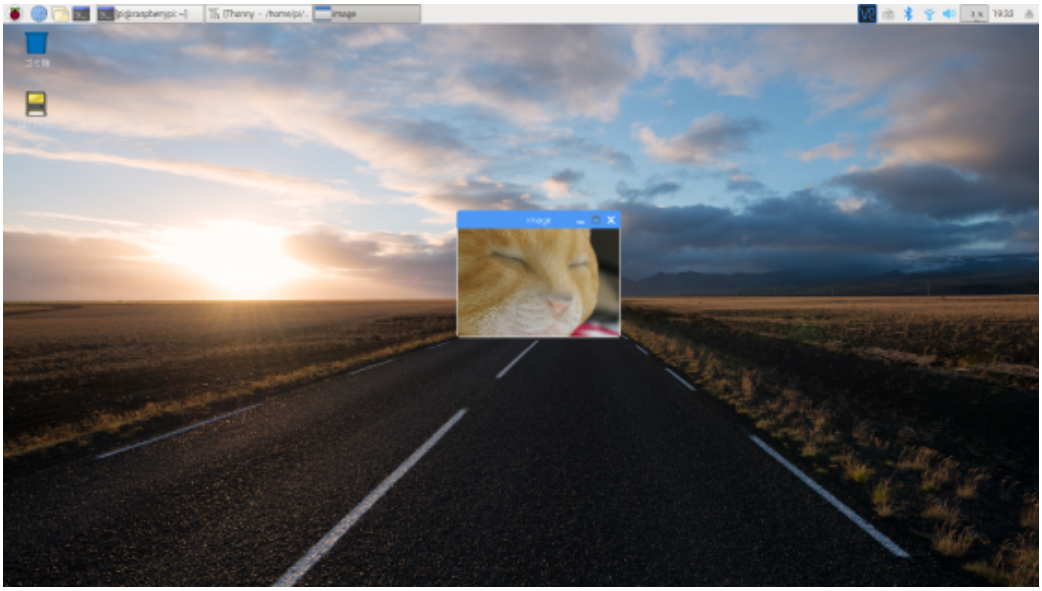
```
import cv2

image_file = 'cat.png'
img = cv2.imread(image_file)

cv2.imshow('image', img[200:400, 300:600])

cv2.waitKey(0)
cv2.destroyAllWindows()
```

猫の顔の部分だけが表示されたと思います。



ndarrayの順番は `[高さ方向，幅方向]` であることに注意してください。

次元

これまでも、1次元、2次元、3次元と次元を扱ってきましたが、ここでは次元を参照・変更するという関数について説明します。

次元数の参照：ndim

`ndim` はndarrayの次元数を参照することができます。

```
import numpy as np

a = np.array([1, 2, 3, 4])
print(a.ndim)

b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(b.ndim)

c = np.array([[[ 1,  2,  3,  4], [ 5,  6,  7,  8], [ 9, 10, 11, 12]],
              [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]])
print(c.ndim)

# 表示結果
# 1
# 2
# 3
```

次元の入れ替え：transpose

`transpose`は引数で指定した順番通りに軸を入れ替える関数です

実際のコードで見た方が理解しやすいです

```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

print(a)
print('----------')

b = a.transpose(1, 0)
print(b)

# 表示結果
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
# ----------
# [[ 1  5  9]
#  [ 2  6 10]
#  [ 3  7 11]
#  [ 4  8 12]]
```

`print('----------')`は表示結果を見やすくするために入れています。

`a.transpose(0, 1)`と書いた場合は変化はありません。2次元の場合は`a.transpose(1, 0)`と書くことで軸が入れ替わります。

3次元である画像に`transpose`を適用した例を見てみましょう

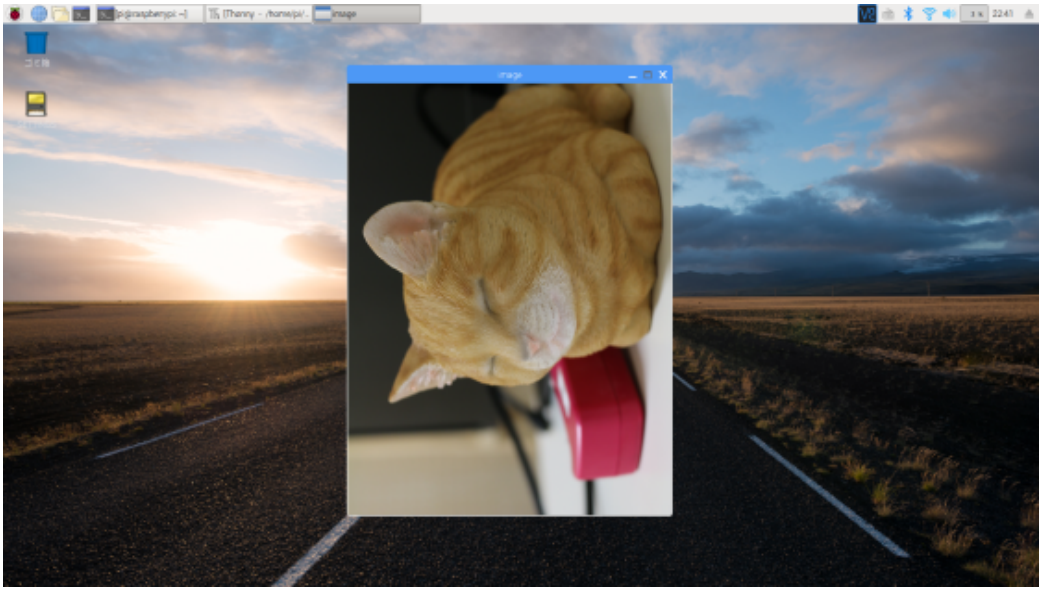
```
import cv2

image_file = 'cat.png'
img = cv2.imread(image_file)

cv2.imshow('image', img.transpose(1, 0, 2))

cv2.waitKey(0)
cv2.destroyAllWindows()
```

これは分かりやすい結果だと思います。



画像データは(高さ, 幅, カラーチャンネル)という並びであることを先ほど言いましたが、これを英語で書くと(Hight, Width, Channel)になります。この頭文字を取ってHWCと略します。
上記のコードではHWCからWHCへの変換したデータを表示しているということです。
HCW,WCH,CHW,CWHへの変換自体は可能です。ただし、それらのデータで表示を行うとエラーになりますので気を付けて下さい。

次回取り扱う推論エンジンの画像フォーマットはCHWです。そのためこのようなコードが必要になります

```
img = img.transpose((2, 0, 1)) # HWC > CHW
```

次元の削減：squeeze

`squeeze`は大きさが1である次元を削除する関数です。これも実例をみて確かめてみましょう
まずは、大きさが1である次元の例を書いてみます。

```
import numpy as np

a = np.array([1, 2, 3, 4])
print(a)
print(a.ndim)
print(a.shape)
print('----------')
```

```
b = np.array([[1, 2, 3, 4]])
print(b)
print(b.ndim)
print(b.shape)
print('----------')

c = np.array([[[1, 2, 3, 4]]])
print(c)
print(c.ndim)
print(c.shape)

# 表示結果
# [1 2 3 4]
# 1
# (4,)
# ----------
# [[1 2 3 4]]
# 2
# (1, 4)
# ----------
# [[[1 2 3 4]]]
# 3
# (1, 1, 4)
```

変数`a`は1次元、`b`は2次元、`c`は3次元のndarrayです

`a`は今まで通りのndarrayですが、`b`や`c`は余分に`[ ]`が付いているのが分かるかと思います。

これが大きさが1である次元を含んでいる状態です。

`squeeze`は簡単に言うと、この余分な`[]`を取り除き、次元を減らす関数です。

以下のコードでは先の全てのndarrayに対して、`squeeze`を加えてみた例です。

```
import numpy as np

a = np.array([1, 2, 3, 4])
a = np.squeeze(a)
print(a)
print(a.ndim)
print(a.shape)
print('----------')

b = np.array([[1, 2, 3, 4]])
b = np.squeeze(b)
print(b)
print(b.ndim)
print(b.shape)
print('----------')
```

```
c = np.array([[[1, 2, 3, 4]]])
c = np.squeeze(c)
print(c)
print(c.ndim)
print(c.shape)

# 表示結果
# [1 2 3 4]
# 1
# (4,)
# ----------
# [1 2 3 4]
# 1
# (4,)
# ----------
# [1 2 3 4]
# 1
# (4,)
```

`a`は元々、大きさ1の次元はないため変化ありませんが、`b`と`c`は余分な`[]`が消えているのが分かるかと思います。

次元の追加：expand_dims

`expand_dims`は逆に大きさ１の次元を追加することができます。

キーワード引数として`axis=0`を書くことにより次元の追加先を0番目の軸（一番外側の次元）に指定することができます。

```
import numpy as np

a = np.array([1, 2, 3, 4])
a = np.expand_dims(a, axis=0)
print(a)
print(a.ndim)
print(a.shape)
print('----------')

b = np.array([[1, 2, 3, 4]])
b = np.expand_dims(b, axis=0)
print(b)
print(b.ndim)
print(b.shape)
print('----------')

c = np.array([[[1, 2, 3, 4]]])
c = np.expand_dims(c, axis=0)
print(c)
print(c.ndim)
print(c.shape)
```

```
# 表示結果
# [[1 2 3 4]]
# 2
# (1, 4)
# ----------
# [[[1 2 3 4]]]
# 3
# (1, 1, 4)
# ----------
# [[[[1 2 3 4]]]]
# 4
# (1, 1, 1, 4)
```

大きさ１の次元が追加されているのが分かるかと思います。

このようにいくらでも次元を増やすことができます。一見意味無いように思えますが、次元数のフォーマットを合わせる際に使う場合があります。

以上、「NumPyプログラミングを学ぶ」でした。

Newsletter

ご登録いただくと、新商品やイベント情報をお知らせします。

メールアドレスを入力してください 登録

個人情報について

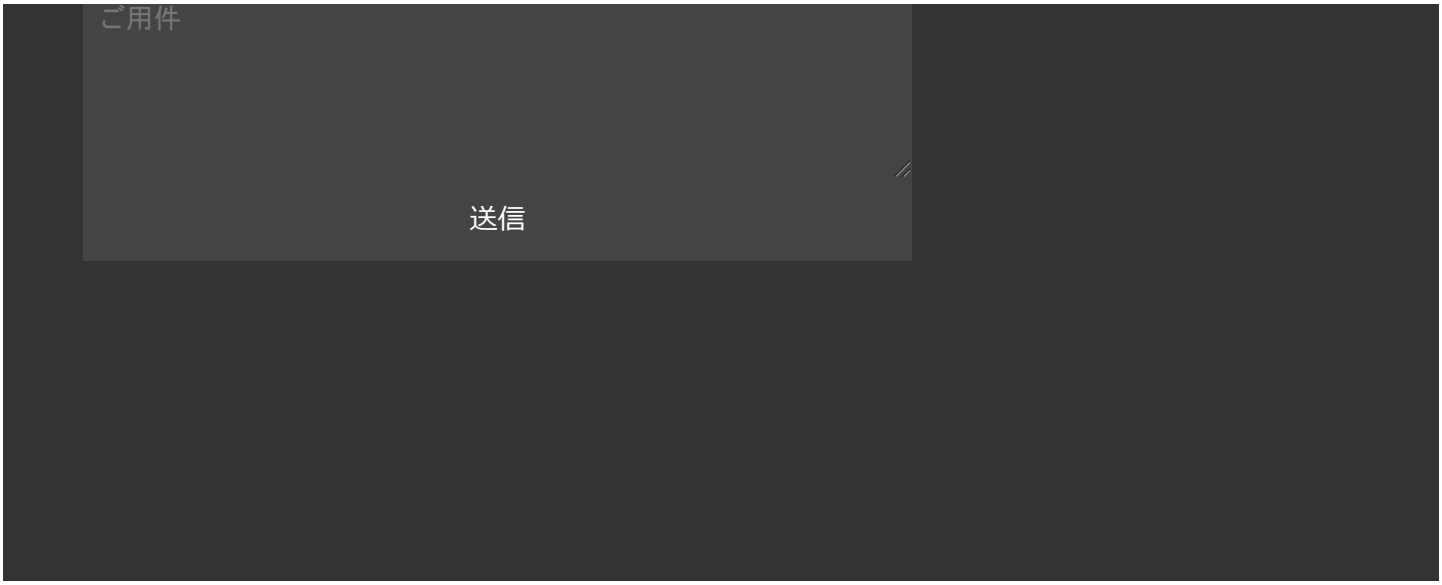
JELLYWARE

〒160-0004 東京都新宿区四谷2-3-6 パルム四谷702号室
03-6273-0758
info@jellyware.jp

個人情報について

CONTACT

お名前
メールアドレス
メールアドレス確認
お電話番号（任意）
ご件名


ご用件
送信