

# 知覚ハッシュの生成を目的とした Keras中間層の確率分布の解析

木下研究室 201603744 小澤涼平

# 研究背景

- 近年、ネットワーク上のデジタルコンテンツの流通の普及に伴い海賊版サイトによる著作権法違反やコミックマーケットにおけるコンテンツの二次利用の問題などが存在し、コンテンツの著作権保護の担保が必要不可欠となってきている。
- この解決策の1つとして、電子透かしが用いられている。



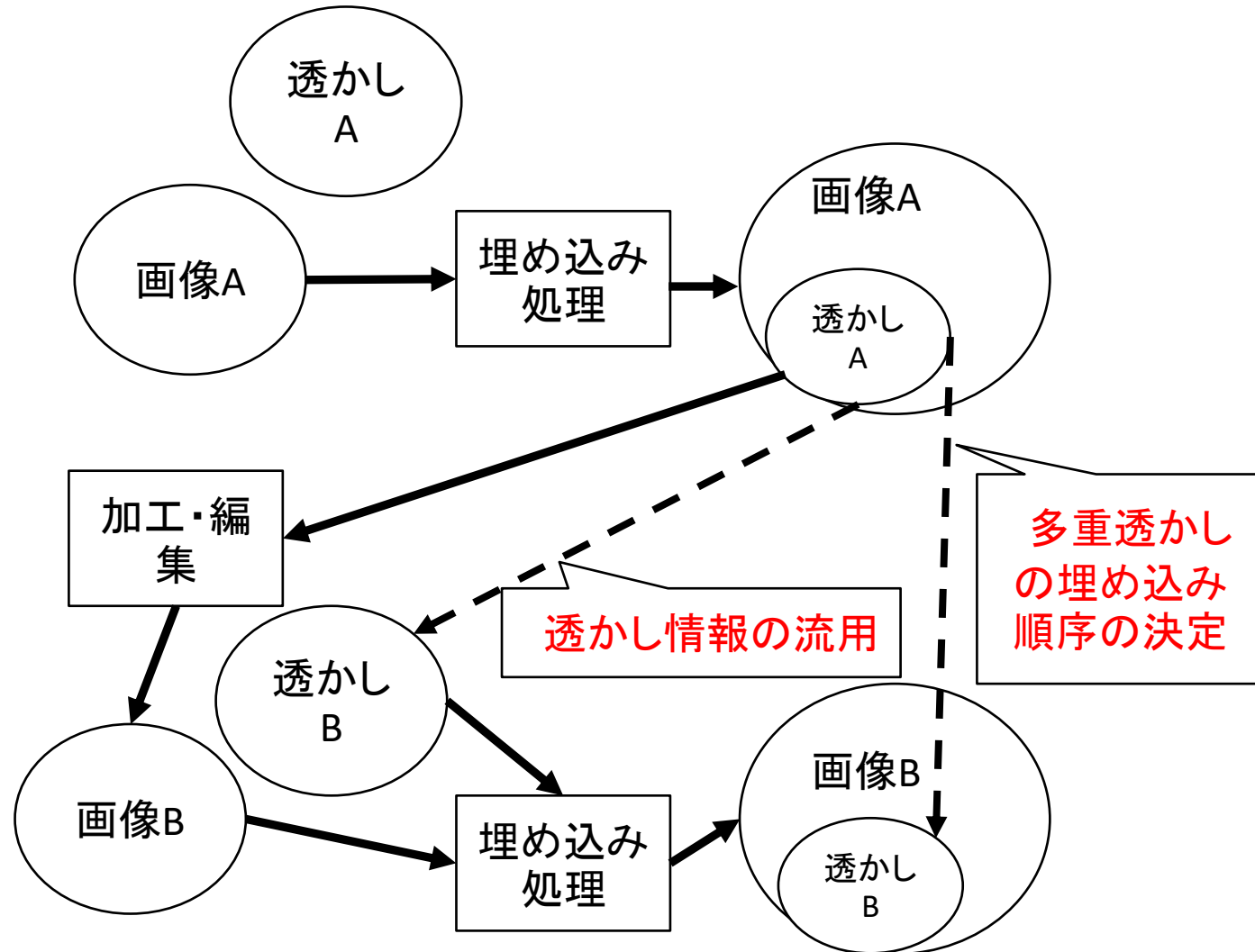
普段、電子透かしは見えないが



実は著作権情報が埋め込まれている

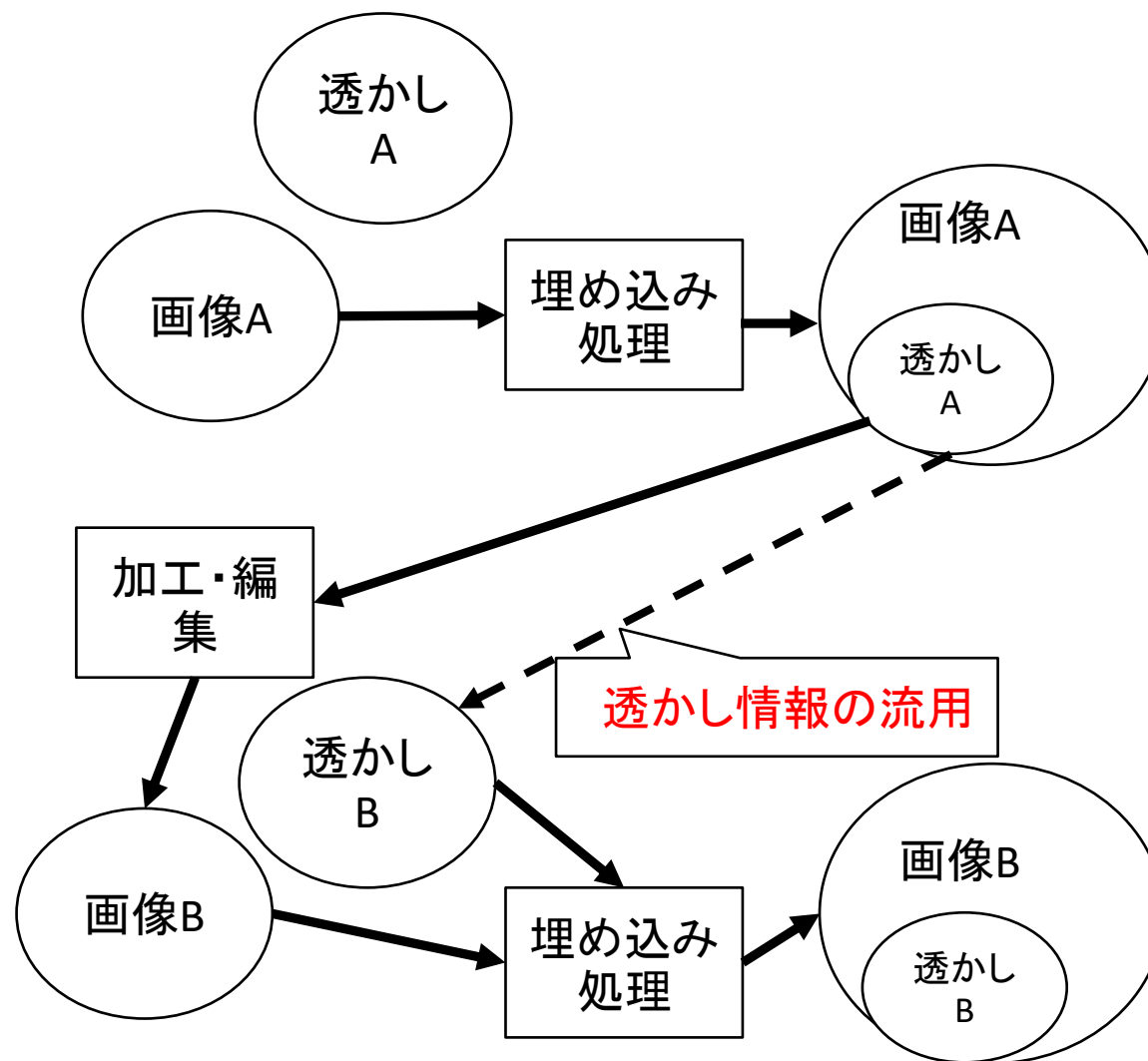
# 電子すかしの問題点

- すかし情報がコピーされて他のコンテンツに流用される可能性がある。
- 二次利用により多重に埋め込まれた電子すかしの優先順位や加工内容との対応関係を明確にできない。



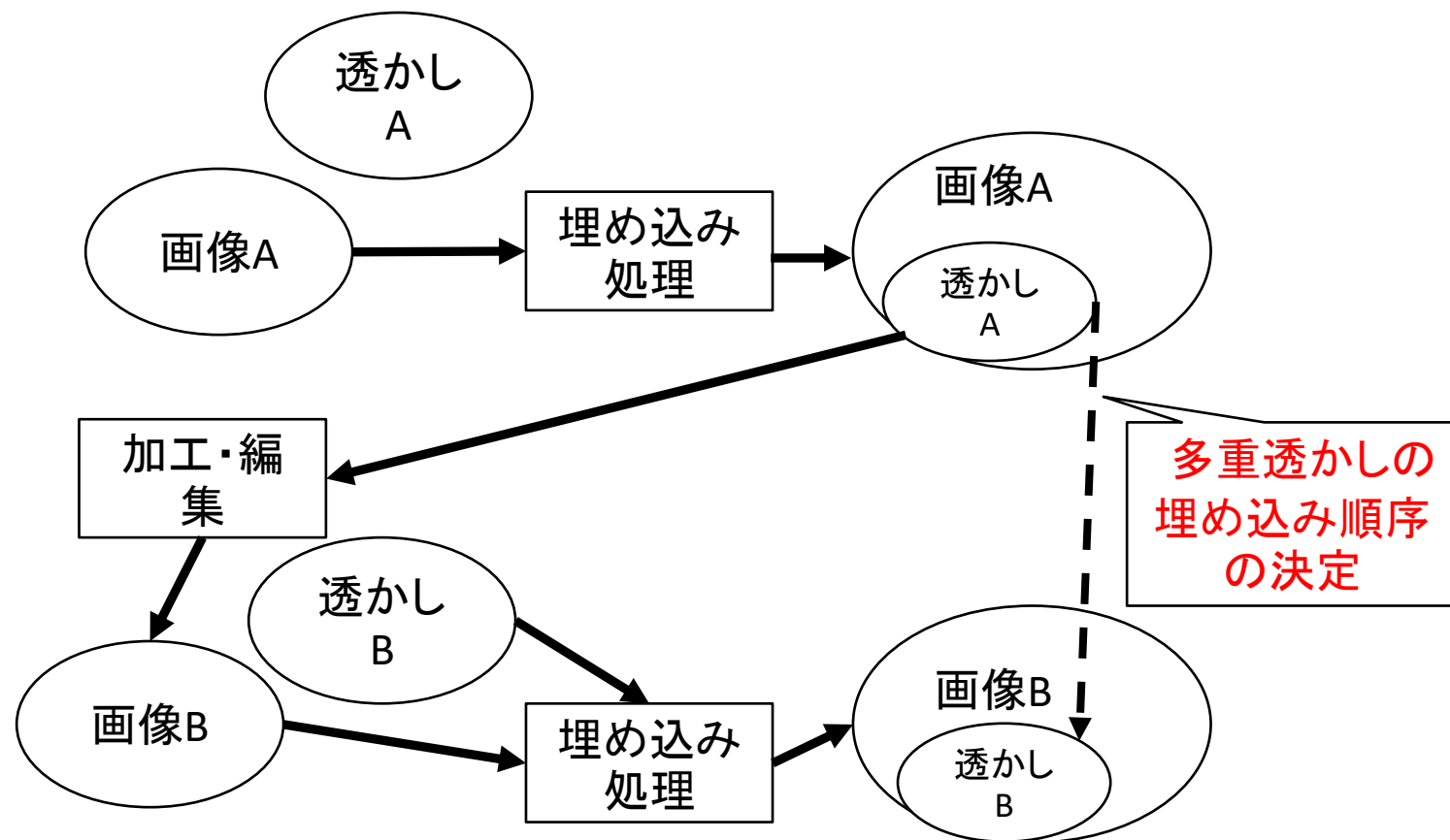
# 電子透かしの問題点の解決法

- 透かし情報は原画像に依存しなければならない
- 暗号的ハッシュ関数は加工編集の影響を受けるので不向き
- 加工編集の影響を受けない知覚ハッシュを利用



# 電子透かしの問題点の解決法

- 多くの人が順序よくすかし情報を利用しなければならない。
- 多重すかしの埋め込み順序を保証するために信頼できる第三者を仮定することはプライバシー保護やサービスの継続の面から望ましくない
- **ブロックチェーンによる透かし情報の管理、知覚ハッシュによる電子透かしの生成が提案される。**



# 従来の知覚ハッシュの問題点

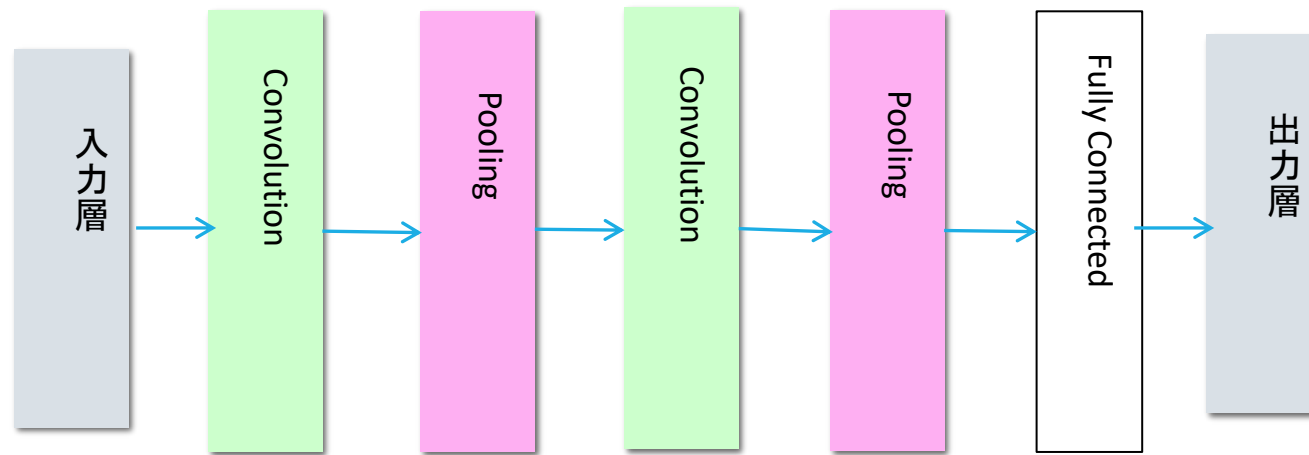
- 従来の知覚ハッシュの場合、画像の変形、回転、反転が施されると出力されるハッシュ値が変わる欠点がある。
- 加工編集の種類に応じて最適なアルゴリズムが異なる。

# CNNを利用した知覚ハッシュの提案

- CNNの中間層には構造など画像に固有の情報が含まれているので原画像に依存した透かし情報を生成することができる。
- 画像に依存 → 機械学習 (CNN(Convolutional Neural Network))
- 前の発表(小藤田)は中間層をパシステントホモロジーを利用して位相幾何学的構造を抽出した。
- 本研究は、CNNの中間層のノードを主成分分析し、確率分布を出力することで知覚ハッシュを生成する要素を抽出する。

# KerasのCNN(畳み込みニューラルネットワーク)

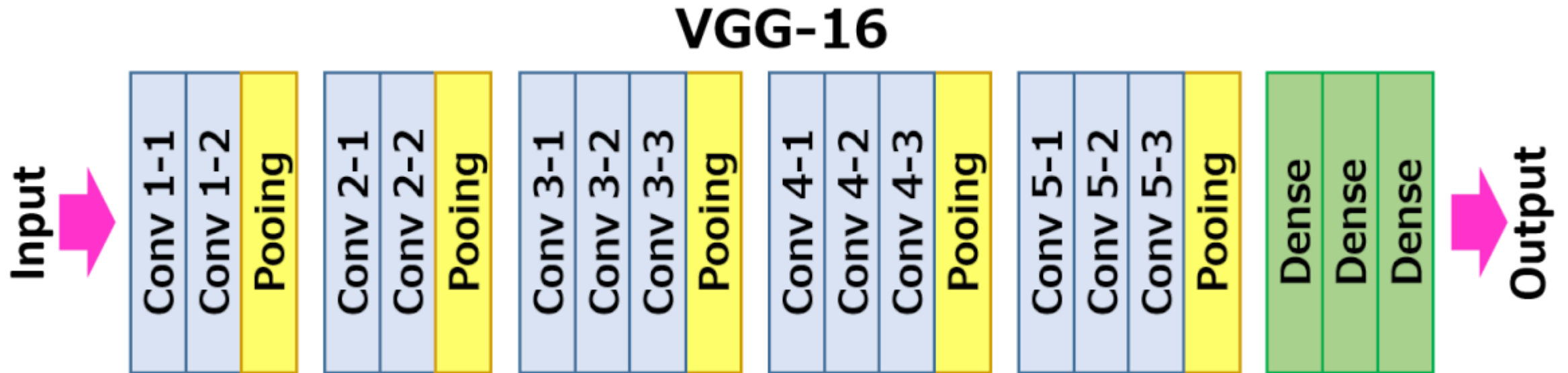
- ・CNNはニューラルネットワークの中間層の部分に畳み込み層(Convolution層)とプーリング層(Pooling層)を組み込んで構成されるニューラルネットワーク。



Pooling層で特徴量を圧縮することで、画像の移動や変形に対して影響を受けにくくし、計算量を下げる。

# VGG16

- ◆ 畳み込み13層 + 全結合層3層 = 16層からなるニューラルネットワークのモデル。



# 学習済みVGG16 Fine-tuning

## ◆ 学習済みVGG16

元々1000種類にラベル付けされた100万枚以上の画像の集まりであるImageNetという大規模データセットで学習済みのモデルであり、出力は1000クラスの分類である。

## ◆ Fine tuning

転移学習のひとつであり、学習データに合わせて各層の重みが変わることである。学習済みモデルを利用するのは1からCNNを設計する手間を省けるためである。さらには、既に大量のデータで学習済みである画像の分類に特化したモデルを利用することで、本研究用の少ないデータセットでも高い分類精度を実現できる。

元々学習済みのVGG16をFine tuningすることによってより精度の高い結果を得られるモデルを用いて実験が行える。

# 提案手法

## VGG-16



<https://neurohive.io/en/popular-networks/vgg16/>

- ◆ VGG16を用いてそれぞれの画像の5つ目のPooling層から特徴量を抽出する。
- ◆ 出力に最も近いPooling層を選ぶことによって原画像に依存した特徴量を抽出できる。

出力される512個のノード

## 主成分分析

知覚ハッシュを生成する要素を抽出する。

# 実験で用いる画像

- ◆ 256 × 256の原画像に対して回転や反転を行い1000枚の画像を作る。
- ◆ パラメータは、回転角度を左右それぞれ最大180度、左右上下に最大50%のスライド、元のサイズに対して50%から200%の縮小・拡大を設定。の設定は、主観評価で原画像の特徴が残る程度にしている。画像1枚につき先ほどのパラメータの範囲でランダムに生成。



- ◆ 主成分分析の対象として約 1000 枚の画像の中から無作為に 3 枚選んだ。

# 中間層の出力

- 学習済みのVGG16 Fine-tuningによって各画像の5つ目のPooling層を出力する。
- 5つ目のPooling層は各画像それぞれ512個のCSVファイルとして出力された。

```
# -*- encoding: utf-8 -*-
from keras.layers import MaxPooling2D
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing import image
from keras import models
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from keras.models import model_from_json
from keras.utils import np_utils
import matplotlib.pyplot as plt
import os
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model
from keras.layers import Input, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
import numpy as np
import time
import tensorflow as tf
import sys
import glob

model = model_from_json(open('/home/ryohei/ダウンロード/VGG16-Finetuning-model.json').read())
model.load_weights('/home/ryohei/ダウンロード/VGG16-Finetuning-weight.h5')

file_list = glob.glob('/home/ryohei/ダウンロード/中間層/pic2/*.bmp')
for x, file in enumerate(file_list):
    img = image.load_img(file, target_size=(150, 150))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    print("IMAGE: %s" % str(img.shape))

    # ImageNet データセットでFine-tuning済みの VGG16 をロード
    layers = model.layers[1:19] # ← ここで出力する層を決める
    layer_outputs = [layer.output for layer in layers]
    activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
    activation_model.summary()

    activations = activation_model.predict(img)

    # Pooling層のみを行列で出力
    activations = [activation for layer, activation in zip(layers, activations) if isinstance(layer, MaxPooling2D)]

    for i, activation in enumerate(activations):
        num_of_image = activation.shape[3]
        for j in range(0, num_of_image):
            np.savetxt("/home/ryohei/ダウンロード/中間層4/%d_%d.csv" % (x+1, i+1, j+1), activation[0, :, :, j])
```

# 出力ファイルの整理

- 主成分分析を行うため出力された5つ目のPooling層の要素である512個のCSVファイルを1個のCSVファイルにまとめた。

```
import numpy as np
```

```
import os  
import glob
```

```
cnn=glob.glob('/home/ryohei/デスクトップ/Pooling5.2/*.csv')
```

```
d=[]  
for a in cnn:  
    b=np.loadtxt(a)  
    c=np.reshape(b,(16,))  
    d.append(c)
```

```
print(len(d))
```

```
512
```

```
raw_data= np.array(d)
```

```
raw_data.shape
```

```
(512, 16)
```

```
np.savetxt('/home/ryohei/デスクトップ/Pooling_pca/pooling5.2/pooling5.2.csv',raw_data)
```

# 主成分分析

- SklearnのPCAをインポートする。
- 3枚の画像の5つ目のPooling層について同時に主成分分析を行い1つの確率分布にまとめた。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA

def main():

    a_1=np.loadtxt('/home/ryohei/デスクトップ/Pooling_pca/pooling5/pooling5.csv')
    b_1=np.loadtxt('/home/ryohei/デスクトップ/Pooling_pca/pooling5.1/pooling5.1.csv')
    c_1=np.loadtxt('/home/ryohei/デスクトップ/Pooling_pca/pooling5.2/pooling5.2.csv')

    # y = 2x

    # グラフ描画サイズを設定する
    plt.figure(figsize=(12, 4))

    # 主成分分析する
    pca = PCA(2)
    pca.fit(a_1)

    # 分析結果を元にデータセットを主成分に変換する
    a = pca.fit_transform(a_1)
    b = pca.fit_transform(b_1)
    c = pca.fit_transform(c_1)

    # 主成分をプロットする
    plt.subplot(1, 2, 2)
    plt.scatter(a[:, 0], a[:, 1]) # 青が全編集画像
    plt.scatter(b[:, 0], b[:, 1]) # 黄色が原画像
    plt.scatter(c[:, 0], c[:, 1]) # 緑が任意の編集画像

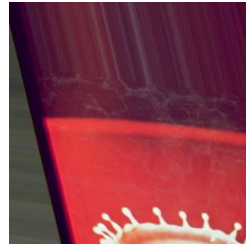
    plt.title('principal component')
    plt.xlabel('pc1')
    plt.ylabel('pc2')

    # 主成分の次元ごとの寄与率を出力する
    print(pca.explained_variance_ratio_)

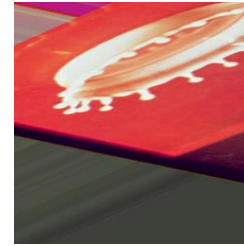
    # グラフを表示する
    plt.show()

if __name__ == '__main__':
    main()
```

# 実験結果



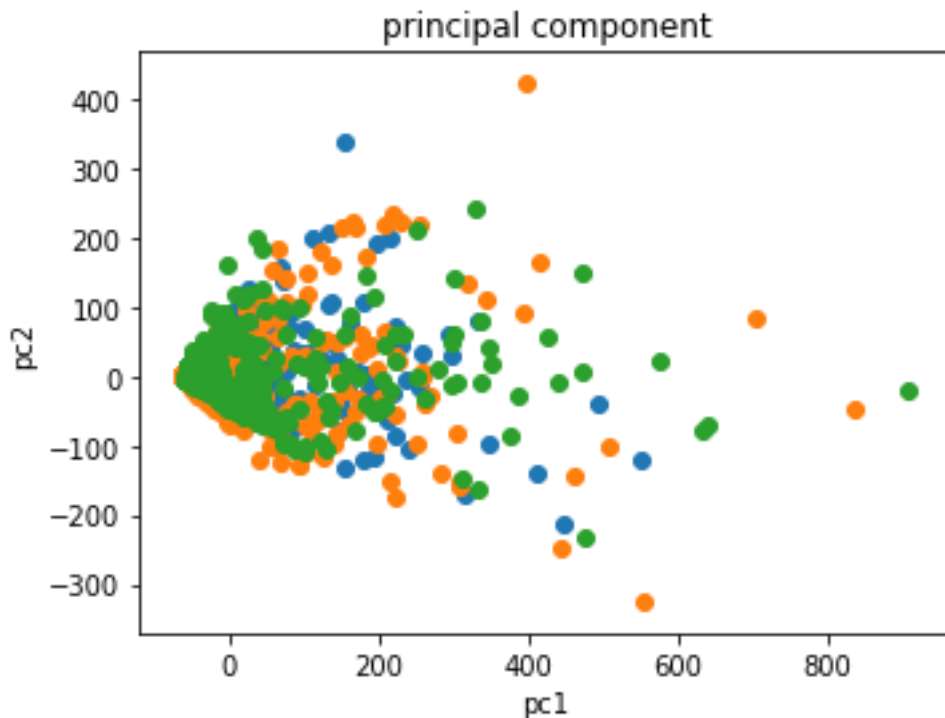
...青



...黄色



...緑



- 各色ごとに4×4のテンソルが512個抽出され、ノードとして表される。
- 各画像でばらつきのあるノードは加工編集による影響(確率変数は各ノード)
- 図のpc1軸0~200、pc2軸-100~100の範囲に集中しているノードが加工編集に影響されないノードの集合と考えられる。

# 考察と今後の課題

- 集中しているノードの集合のベクトルに依存して、知覚ハッシュアルゴリズムを構成すれば、加工編集に対して耐性のある知覚ハッシュ値を得ることができると考えられる。
- 今後の課題としては集中しているノードから加工編集に影響されない知覚ハッシュ値を生成する方法の確立が課題である。

# 研究環境

使用言語 Python3.6

機械学習ライブラリとしてKerasを使用

- 比較的短いソースコードで実装することが可能。
- ネットワークの構築が簡略化されている。

統計分析ではRを用いる

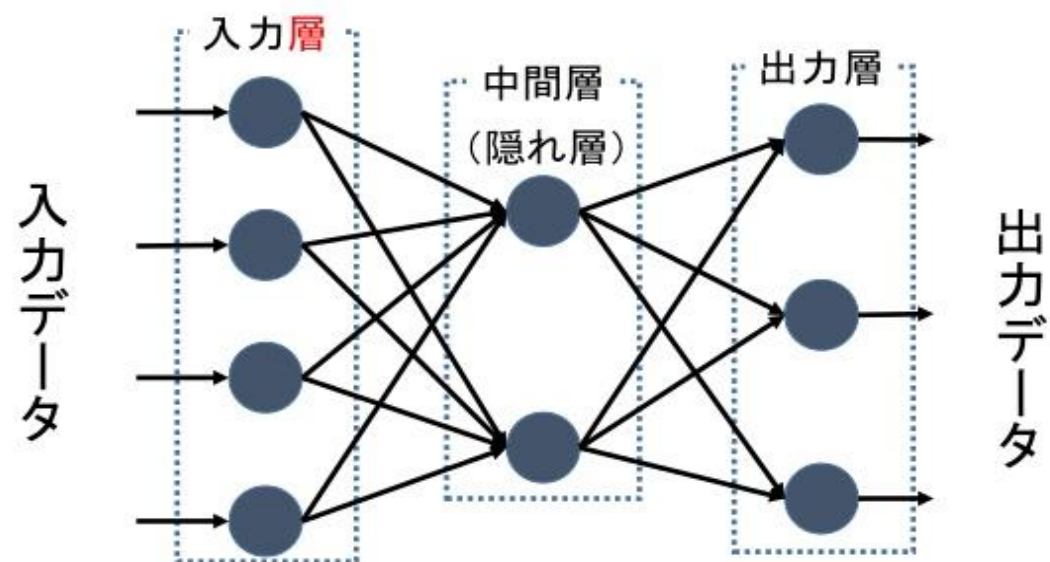
- 統計処理などに特化していてかつ便利

これらはオープンソースであるAnacondaのライブラリーであるため、一括して開発はAnacondaで行っている。

# ニューラルネットワーク

- 機械学習と呼ばれる手法の1つであり、人間の脳内にある神経細胞（ニューロン）とそのつながり、つまり神経回路網を人工ニューロンという数式的なモデルで表現したもの。

## ニューラルネットワークの基本構造



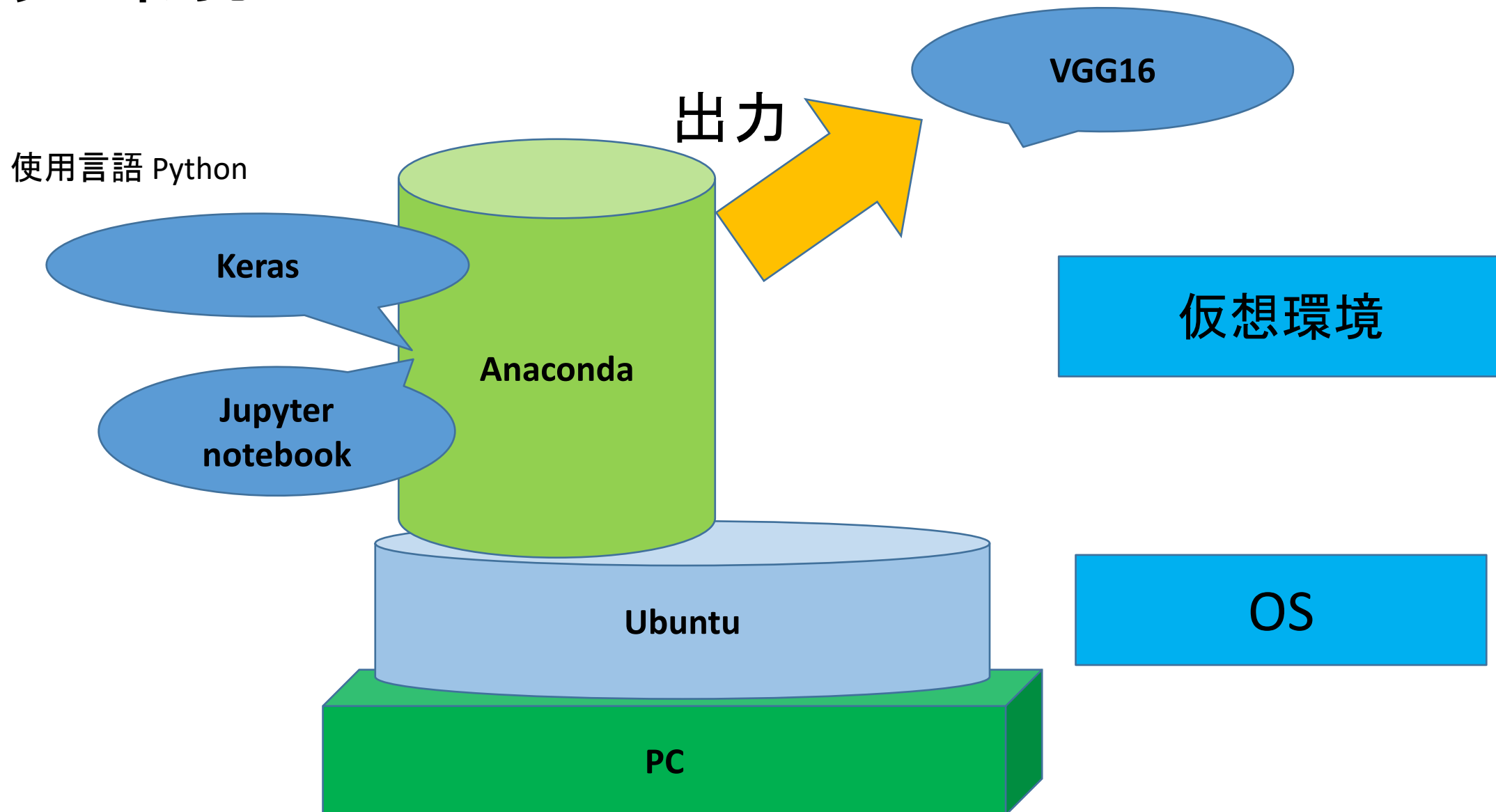
- \* 中間層は複数あっても良いが、多層になると学習が進みにくいという特徴がある。
- \* 中間層のユニット数は自由に調整できるが、多すぎても少なすぎてもダメだ。

# Keras

- 機械学習のライブラリであり、Python(言語)で書かれたTensorFlow(様々な機械学習の分野で使用するためのオープンソフトウェアライブラリ)などで実行可能なニューラルネットワークライブラリである。

→比較的短いソースコードで実装することが可能。

# 研究環境

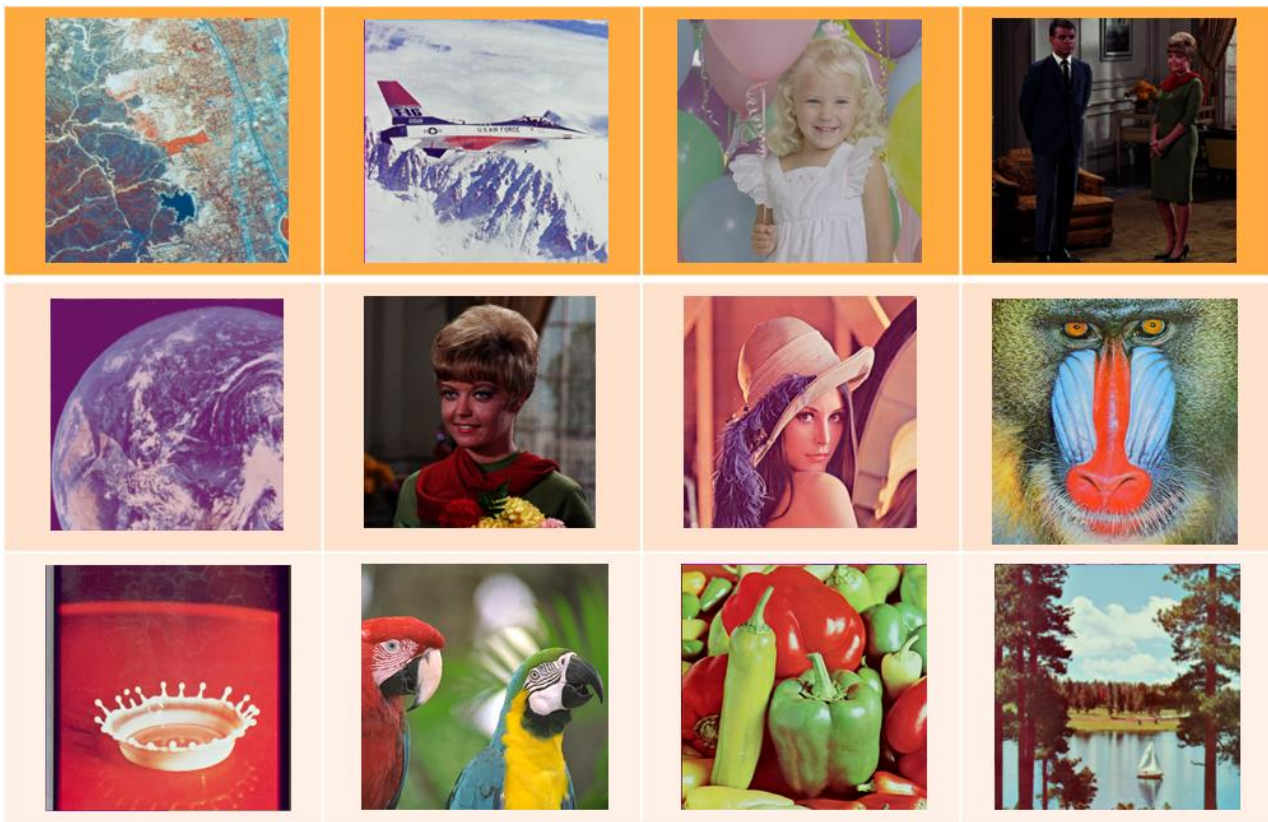


# 知覚ハッシュの種類

- 知覚ハッシュアルゴリズムには以下のようなものがある。

| aHash              | pHash                                              | dHash          | wHash                                |
|--------------------|----------------------------------------------------|----------------|--------------------------------------|
| 画像の平均輝度からの差分を使って算出 | 画像を離散コサイン変換(DCT)し周波数領域に変換後、低周波領域に対してaHashと同様の手法で算出 | 隣接領域との差分を使って算出 | pHashのDCTの代わりに離散ウェーブレット変換(DWT)を用いて算出 |

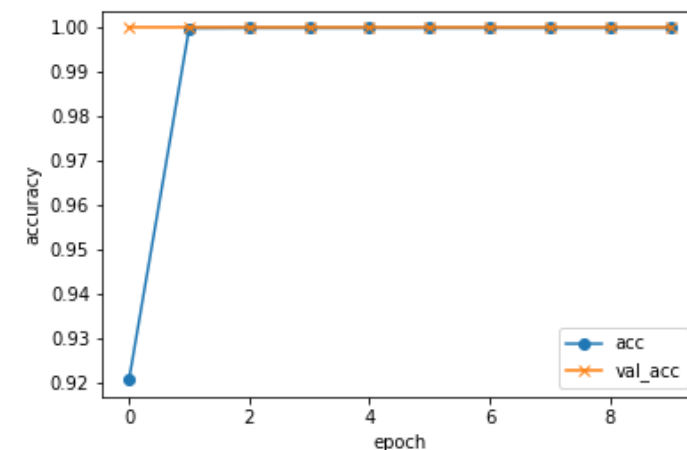
# データセット



・データセットとして画像処理の分野で広く利用されている12枚のカラー画像を使用する。

# 転移学習

| Layer (type)               | Output Shape         | Param # | block3_pool (MaxPooling2D)      | (None, 18, 18, 256) | 0       |
|----------------------------|----------------------|---------|---------------------------------|---------------------|---------|
| input_1 (InputLayer)       | (None, 150, 150, 3)  | 0       | block4_conv1 (Conv2D)           | (None, 18, 18, 512) | 1180160 |
| block1_conv1 (Conv2D)      | (None, 150, 150, 64) | 1792    | block4_conv2 (Conv2D)           | (None, 18, 18, 512) | 2359808 |
| block1_conv2 (Conv2D)      | (None, 150, 150, 64) | 36928   | block4_conv3 (Conv2D)           | (None, 18, 18, 512) | 2359808 |
| block1_pool (MaxPooling2D) | (None, 75, 75, 64)   | 0       | block4_pool (MaxPooling2D)      | (None, 9, 9, 512)   | 0       |
| block2_conv1 (Conv2D)      | (None, 75, 75, 128)  | 73856   | block5_conv1 (Conv2D)           | (None, 9, 9, 512)   | 2359808 |
| block2_conv2 (Conv2D)      | (None, 75, 75, 128)  | 147584  | block5_conv2 (Conv2D)           | (None, 9, 9, 512)   | 2359808 |
| block2_pool (MaxPooling2D) | (None, 37, 37, 128)  | 0       | block5_conv3 (Conv2D)           | (None, 9, 9, 512)   | 2359808 |
| block3_conv1 (Conv2D)      | (None, 37, 37, 256)  | 295168  | block5_pool (MaxPooling2D)      | (None, 4, 4, 512)   | 0       |
| block3_conv2 (Conv2D)      | (None, 37, 37, 256)  | 590080  | Total params: 14,714,688        |                     |         |
| block3_conv3 (Conv2D)      | (None, 37, 37, 256)  | 590080  | Trainable params: 7,079,424     |                     |         |
|                            |                      |         | Non-trainable params: 7,635,264 |                     |         |



- ここでは上図に示す全ての畳み込み層とプーリング層の重みを可変にすることで、より自前のデータセットに特化したモデルを作れる。
- 右図に示すように分類精度は100%となった。

# 転移学習の利用

- 転移学習とは、ある設定（例えば分布 $P_1$ ）で学んだことを別の設定（例えば分布 $P_2$ ）の汎化能力向上に役立てようとする状況を指す言葉である。
- 転移学習のメリットはわざわざ1から自分で作成することなくCNNを利用できる。  
（CNNの設計には層の深さや学習率の設定といった調整が多い）
- すでに画像の分類に特化したモデルを利用できる。（VGG16はImageNetという100万枚以上の大規模画像データセットで学習済みのモデルであり、出力は1000クラスからなる画像の分類に特化したモデルであるため）
- 少ないデータセットでも高い分類精度を実現できる。（すでに大量のデータで学習済みなため、追加のデータは少量で済む）

# 加工編集の種類

- ここで行う編集は、二次利用を想定した非可逆圧縮、ノイズ付加、アフィン変換である。
- 画像1枚につき3000枚の画像セットを生成する。
- このように学習データを増やす作業をデータオーグメンテーションという。

|       |               |
|-------|---------------|
| 非可逆圧縮 | JPEG方式を採用     |
| ノイズ   | 画像にランダムノイズを付加 |
| 回転角度  | 左右に最大180度     |
| スライド  | 左右上下に最大4ピクセル  |



# 使用するCNNのモデルと各層におけるパラメータの数

| Layer (type)                  | Output Shape          | Param #   |
|-------------------------------|-----------------------|-----------|
| input_1 (InputLayer)          | (None, 224, 224, 3)   | 0         |
| block1_conv1 (Conv2D)         | (None, 224, 224, 64)  | 1792      |
| block1_conv2 (Conv2D)         | (None, 224, 224, 64)  | 36928     |
| block1_pool (MaxPooling2D)    | (None, 112, 112, 64)  | 0         |
| block2_conv1 (Conv2D)         | (None, 112, 112, 128) | 73856     |
| block2_conv2 (Conv2D)         | (None, 112, 112, 128) | 147584    |
| block2_pool (MaxPooling2D)    | (None, 56, 56, 128)   | 0         |
| block3_conv1 (Conv2D)         | (None, 56, 56, 256)   | 295168    |
| block3_conv2 (Conv2D)         | (None, 56, 56, 256)   | 590080    |
| block3_conv3 (Conv2D)         | (None, 56, 56, 256)   | 590080    |
| block3_pool (MaxPooling2D)    | (None, 28, 28, 256)   | 0         |
| block4_conv1 (Conv2D)         | (None, 28, 28, 512)   | 1180160   |
| block4_conv2 (Conv2D)         | (None, 28, 28, 512)   | 2359808   |
| block4_conv3 (Conv2D)         | (None, 28, 28, 512)   | 2359808   |
| block4_pool (MaxPooling2D)    | (None, 14, 14, 512)   | 0         |
| block5_conv1 (Conv2D)         | (None, 14, 14, 512)   | 2359808   |
| block5_conv2 (Conv2D)         | (None, 14, 14, 512)   | 2359808   |
| block5_conv3 (Conv2D)         | (None, 14, 14, 512)   | 2359808   |
| block5_pool (MaxPooling2D)    | (None, 7, 7, 512)     | 0         |
| flatten (Flatten)             | (None, 25088)         | 0         |
| fc1 (Dense)                   | (None, 4096)          | 102764544 |
| fc2 (Dense)                   | (None, 4096)          | 16781312  |
| predictions (Dense)           | (None, 1000)          | 4097000   |
| Total params: 138,357,544     |                       |           |
| Trainable params: 138,357,544 |                       |           |
| Non-trainable params: 0       |                       |           |

- CNNの実装には、Pythonの機械学習ライブラリのKerasで用意されている学習済みモデルのVGG16を使用する。
- このモデルは畳み込み13層＋全結合層3層＝16層のニューラルネットワークで構成されている。

# 主成分分析

多次元のデータを低次元データに縮約する方法のことである。

高次元データを2次元か3次元に落とすことで人間が理解しやすい形式に変換するために行う。

データの概要を掴みやすく、分散を要約しやすい。

→データを要約しやすいため、特徴を判断しやすい。