**２０２１　年度　　卒業論文**

卒業論文題目

|  |
| --- |
| 画像検索のための画像特徴ベクトルの次元数に着目した認識精度と計算コストの関係性の評価・実験 |

　指導教員　　　　　　鷹野　孝典　教授

神奈川工科大学　情報工学科

　　　　　学籍番号　　　　1821005

　　　　　学生氏名　　　　　 吉岡　拓郎

　提出日　２０２１年　１２月２３日　指導教員　　印

　受理日　２０２１年　１２月２３日　情報工学科長　　印

# **論文要旨**

内容

[**論文要旨** 2](#_Toc89781564)

[**1.** **まえがき** 2](#_Toc89781565)

[1.1 はじめに 2](#_Toc89781566)

[1.2 本論文の構成 2](#_Toc89781567)

[**2.** **関連研究** 3](#_Toc89781568)

[**3.** **提案方式** 4](#_Toc89781569)

[3.1 提案方式の概要 4](#_Toc89781570)

[3.2 提案方式の手順 4](#_Toc89781571)

[3.2.1 複数モデルの作成について 5](#_Toc89781572)

[3.2.2 特徴ベクトルの抽出 5](#_Toc89781573)

[3.2.3 実験・評価について 6](#_Toc89781574)

[**4.** **実装** 7](#_Toc89781575)

[4.1 実装システム概要 7](#_Toc89781576)

[4.2 実装環境 7](#_Toc89781577)

[4.2.1 環境の概要 7](#_Toc89781578)

[4.2.2 プログラミング言語 7](#_Toc89781579)

[4.2.3 仮想環境 8](#_Toc89781580)

[4.2.4 Pythonライブラリ 8](#_Toc89781581)

[4.3 モデルの作成方法 8](#_Toc89781582)

[4.3.1 AlexNetについて 8](#_Toc89781583)

[4.3.2 モデルの詳しい説明 9](#_Toc89781584)

[4.4 特徴ベクトルの抽出方法 12](#_Toc89781585)

[4.5 評価・実験の方法について 13](#_Toc89781586)

[**5.** **実験** 20](#_Toc89781587)

[5.1 20](#_Toc89781588)

[**6.** **むすび** 21](#_Toc89781589)

[6.1 21](#_Toc89781590)

[**参考文献** 22](#_Toc89781591)

[その他 23](#_Toc89781592)

# **まえがき**

## はじめに

ソーシャルネットワーキングサービスの普及に伴い，写真や画像の投稿が盛んに行われるようになり，大量の画像や写真が蓄積されている．そこで，ユーザが目的の写真や画像にアクセスする手段として，画像検索機能の重要性が増している．又，画像認識の性能向上のため，多くの研究が推進されている．

画像検索機能の画像認識において，画像の特徴量が高次元になるほど，認識性能が高くなる．しかし，特徴量が高次元になると特徴ベクトルの算出コスト，認識処理の計算コストが大きく増加してしまい，解析が困難になってしまう．この現象を次元の呪いという．次元の呪いとは，データの次元数が大きくなり過ぎると，そのデータで表現できる組み合わせが飛躍的に多くなってしまい，その結果，手元にある有限なサンプルデータでは十分な学習結果が得られなくなる.また，計算コストが莫大となってしまう問題が発生する．次元数が小さくなり過ぎると，意味情報が損失し，認識性能が低くなるが，次元数が少ないために，処理時間が減少するといったことが起こる．

上記のことを念頭に入れて，本研究では，画像認識が有効に行える範囲を調査する．ここで記載する意味情報とは，画像を認識する際に，その判断材料となる情報のことを指す．

画像認識の性能を向上させるために，画像認識に有効な特徴ベクトルの次元数とかかる計算コストで処理できるのか計測する．それらの結果を踏まえて評価・考察をする．計算コストを抑えた画像認識手法を調査することで，画像検索システムを成り立たせるうえで，認識性能が正常に扱える有効な範囲はどこなのか明確にする．

## 本論文の構成

本論文は全６章で構成されている．本章では研究の背景や動機，目的について述べる．第２章では，モデルの作成手法，それらにまつわる関連研究について述べる．第３章では，本研究の提案方式とモデルの内容，学習データ，実験環境について述べる．第４章では，提案方式を構築した本システムの設計及び実験システムの実装について述べる．第５章では，AlexNetをもとに作成したモデルを使用して，性能について比較実験をする．実験の概要と，比較実験後の結果を述べる．第６章では，結論と今後の展望について述べ

# **関連研究**

# **提案方式**

## 提案方式の概要

本研究では，画像検索システムにおいて，画像処理が有効に扱える範囲を調査するための画像検索評価システムを提案する．学習モデルを自作し，次元数を操作して画像検索システムとして機能するかどうか実験・評価をする．

CNN　(Convolutional Neural Network)　を用いて抽出した画像の特徴ベクトルは，画像の意味情報が保持されていると仮定する．(仮定するのでなく、決めつけられないか、参考文献を探す)ここでの意味情報とは，画像を認識する際に，その判断材料となる情報のことを指す．特徴ベクトルの生成に，CNNを利用する．CNNを用いた特徴ベクトルの生成方法としては，識別層の一つ二つ手前の全結合層を用いる．また，次元数を変化させる層は，特徴ベクトルを生成する層と同じく，識別層から二つ手前の全結合層の値を変化させる．

## 提案方式の手順

　この提案方式では，画像認識の認識性能を高く保ち，計算コストを抑える最適な画像特徴ベクトルの次元数を調査する．画像集合を用意し，CNNを用いて特徴ベクトルを生成する．

初めに，学習済みの分類モデルを作成する．作成した学習済みモデルを使用して，特徴ベクトルの抽出を行う．抽出した特徴ベクトルを保存する．その後，抽出した特徴ベクトルを用いて，特徴ベクトル間の距離を計算する．そして，距離が近い順に類似度が高いとして，評価を行う．そのときの計算コストと精度について評価する．具体的な流れを下記に記入する．また，提案システム図を図3-1に示す．

STEP-1: 学習済みのモデルを作成する．

STEP-2: STEP-1にて作成したモデルを使用して，特徴ベクトルの抽出を行う．

STEP-3: STEP-2で抽出した特徴ベクトルを保存する．

STEP-4: STEP-3で抽出した特徴ベクトルを用いて，実験・評価を行う．

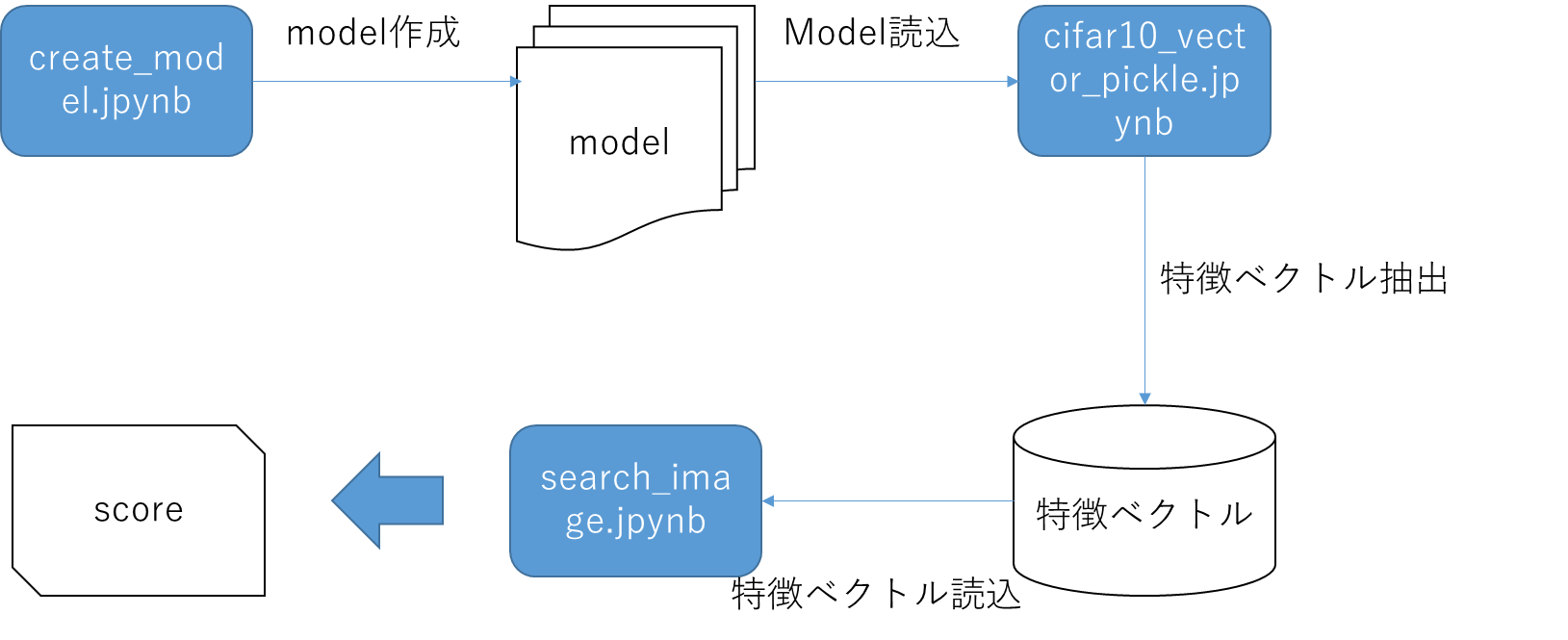


図3-1　提案システム図

### 複数モデルの作成について

本研究では，次元数を変化させて最適な画像検索システムを成り立たせるうえで，認識精度に有効な範囲を調査する．そのため，次元数を変化させたモデルを複数作成する必要がある．本研究では，識別層(softmax)の一つ二つ手前の全結合層の次元数の値を変化させえてモデルを作成する．(図3-2)

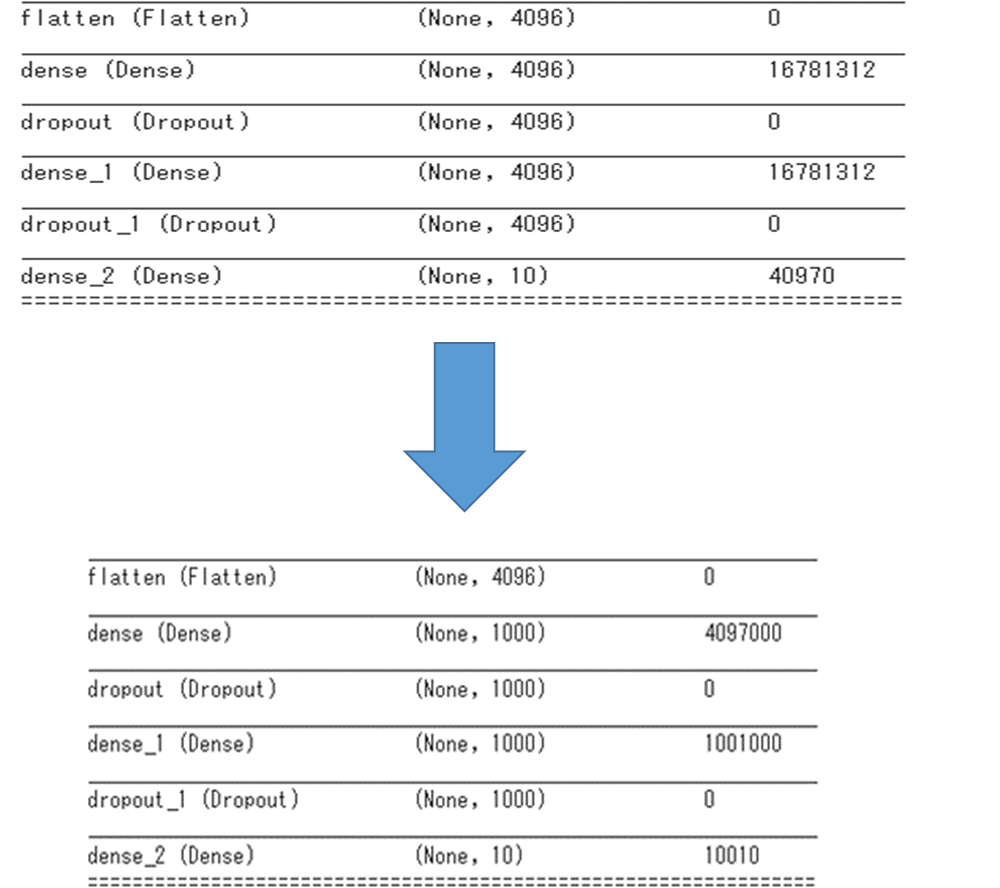


図3-2　モデル変更箇所の例

### 特徴ベクトルの抽出

本研究では，17層のモデルを使用して，特徴ベクトルの抽出を行う．実際に，特徴ベクトルを抽出する層は，15層目のDense層から特徴ベクトルを抽出する．例として，15層目の次元数が4096の場合の特徴ベクトルの一部を表3-1に示す．

表3-1　次元数4096の特徴ベクトル一部

|  |
| --- |
| array([[ 0.90278023, 0.0702083 , 0.67032325, ..., 0.58752143,  0.8521021 , 0.9570551 ],  [ 0.94941396, 0.26129064, 0.85260475, ..., 0.46636236,  0.7725632 , 0.8928799 ],  [ 0.962252 , 0.26241592, 0.78311104, ..., 0.56774634,  0.88178414, 0.88763523],  ...,  [ 0.89410365, 0.8571504 , 0.87597483, ..., 0.42534053,  0.6453284 , 0.79393625],  [ 0.89593625, 0.62443584, -0.02818177, ..., 0.6658653 ,  0.7585609 , 0.9019706 ],  [ 0.950409 , 0.57382387, 0.6239227 , ..., 0.3752147 ,  0.6969327 , 0.9168432 ]], dtype=float32) |

### 実験・評価について

3章では，モデルの作成，特徴ベクトルの抽出について説明を行った．次章では，本研究の画像検索における認識性能について実験，評価を行う．実験では，特徴ベクトル間の距離を測り，それをもとに距離が近い順に類似度が高いとして，列挙する．評価方法は，画像の正解ラベルを確認して，元の画像と同じ場合は正解として，全体の正答数と正答率を求めて，認識性能を確認する．

# **実装**

## 実装システム概要

実装システムでは，画像検索において，画像処理が有効に扱える範囲を調査するために，Google Colabolatoryの仮想環境を使用して，pythonを用いて実装をした．検索の対象は，cifar10データセット内のテストデータ1万件を使用して，特徴ベクトルの抽出を行う．その後，抽出した特徴ベクトルを用いて，類似度，計算コストを測り，評価を行う．

## 実装環境

### 環境の概要

実装を行ったPC環境を表4-1に，プログラミング言語などの開発，実行環境を表4-2に，実装の際に，使用したPythonライブラリを表4-3に示す．

表 4‑1　PC環境

|  |  |
| --- | --- |
| 名称 | 種類 |
| PC |  |
| OS |  |
| GPU |  |

表 4‑2　開発・実行環境

|  |  |
| --- | --- |
| 名称 | バージョン |
| Google Colaboratory |  |
| Python |  |

表 4‑3　Pythonライブラリ

|  |  |  |
| --- | --- | --- |
| 名称 | バージョン | 説明 |
| Numpy | 1.19.5 |  |
| TensorFlow | 2.7.0 |  |
| Matplotlib |  |  |
| Pickle |  |  |
| keras | 2.7.0 |  |

### プログラミング言語

本研究の実装で扱うプログラミング言語は，Pythonを用いた．Pythonは，比較的にソースコードを書く量が少ないため，コードが読みやすい点がある．そして，Pythonは，コンパイルを行い最適化しなくても実行できるため，プログラムを実行するまでの時間を短縮できる．また，Google Colaboratoryという仮想環境が提供されていたためである．ランタイムがリセットされてしまうとファイル等が消えてしまうが，ノートブック形式でプログラムが実行することができるからである．

### 仮想環境

実装では，Pythonの仮想環境として，Google Colaboratoryを使用する．Google　Colaboratoryは，機械学習の普及を目的として作られた．Webブラウザで利用できる無料のPython実行環境である．見た目と操作方法がJupyter notebookに非常に似ている．Google Colaboratoryのメリットとしては，パッケージのインストールなどの環境構築が必要ないこと，GPU・TPUといったものが無償で使用することができること，機種に依存せずに使用することができることが挙げられる．デメリットとしては，PCのファイルを扱うことができないこと，ランタイムがリセットされると実行結果，ファイル等が消えてしまうことが挙げられる．ランタイムがリセットされても平気なように，作成した都度，自身のPC内に保存することが必要である．

### Pythonライブラリ

本研究で実装・実験で，大きく関わってくるPythonライブラリについて説明する．

## モデルの作成方法

モデルの作成では，Google Colaboratoryを利用し，pythonでシステムの開発を行う．ここでは5との次元数の異なるモデルを作成する．それぞれ13層と15層のdence層の部の次元数を変化させてモデルを作成していく．以下に作成したモデルについて表に示す．

表 4-1　作成したモデル一覧表



### AlexNetについて

　2012年のImageNetを用いた画像認識コンペILSVRCでチャンピオンに輝き，Deep Learningの火付け役となったモデルです．5つの畳み込み層，3つの全結合層などから構成されています．具体的な構成は以下の表に示す．

表4-2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer | Kernel\_size | Filters | Strides | Padding | Output\_size |
| Conv\_1 | (11, 11) | 96 | (4, 4) | 0 | (None, 55, 55, 96) |
| Max\_pool\_1 | (3, 3) | - | (2, 2) | - | (None, 27, 27, 96) |
| Conv\_2 | (5, 5) | 256 | (1, 1) | 2 | (None, 27, 27, 256) |
| Max\_pool\_2 | (3, 3) | - | (2, 2) | - | (None, 13, 13, 256) |
| Conv\_3 | (3, 3) | 384 | (1, 1) | 1 | (None, 13, 13, 384) |
| Conv\_4 | (3, 3) | 384 | (1, 1) | 1 | (None, 13, 13, 384) |
| Conv\_5 | (3, 3) | 256 | (1, 1) | 1 | (None, 13, 13, 256) |
| Max\_pool\_5 | (3, 3) | - | (2, 2) | - | (None, 6, 6, 256) |
| FC\_6 | - | - | - | - | (None, 4096) |
| FC\_7 | - | - | - | - | (None, 4096) |
| FC\_8 | - | - | - | - | (None, 1000) |

　AlexNetのモデルでは，Cifar-10データセットの画像入力サイズが違うため，モデルを変更します．上記の表4-2では，ILSVRCコンテスト用なので，入力が3\*224\*224となっている．Cifar-10は，3\*32\*32であるので，修正が必要である．最初の層のConv2d(Conv‗1)の設定を[kernel\_size=11→3，stride=4→1，padding=2→1]に変更して画像サイズが変更しないようにする．そして，3つあるMaxPool2d(Max\_pool\_1，Max\_pool\_2，Max\_pool\_5)の設定を[karnel\_size=3→2]に変更する．このようにすることで，画像サイズを変更するのがMaxPool2dの3回だけになり，[32\*32→16\*16→8\*8→4\*4]と全結合層の直前では4\*4となる．本研究で，作成した構成について以下の表4-3に示す．また，変更箇所は赤字で示す．

表4-3　変更した構成

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer | Kernel\_size | Filters | Strides | Padding | Output\_size |
| Conv\_1 | (3, 3) | 96 | (1, 1) | 1 | (None, 32, 32, 96) |
| Max\_pool\_1 | (2, 2) | - | (2, 2) | - | (None, 16, 16, 96) |
| Conv\_2 | (5, 5) | 256 | (1, 1) | 2 | (None, 16, 16, 256) |
| Max\_pool\_2 | (2, 2) | - | (2, 2) | - | (None, 8, 8, 256) |
| Conv\_3 | (3, 3) | 384 | (1, 1) | 1 | (None, 8, 8, 384) |
| Conv\_4 | (3, 3) | 384 | (1, 1) | 1 | (None, 8, 8, 384) |
| Conv\_5 | (3, 3) | 256 | (1, 1) | 1 | (None, 8, 8, 256) |
| Max\_pool\_5 | (2, 2) | - | (2, 2) | - | (None, 4, 4, 256) |
| FC\_6 | - | - | - | - | (None, 4096) |
| FC\_7 | - | - | - | - | (None, 4096) |
| FC\_8 | - | - | - | - | (None, 10) |

### モデルの詳しい説明

本研究のモデルを作成するにあたって，ソースコードの説明を加える．

Sequentialは，モデル層を積み重ねる形式の記述方法である．Addメゾットで簡単に層を追加することが可能である．以下にソースコードを示す．

表4-4　Sequential

|  |
| --- |
| model = Sequential() |

Cond2Dは，2次元畳込み層である．Cond２D(16，(3，3)の場合，「3\*3」の大きさのフィルタを16枚使うという意味である．「5\*5」「7\*7」のような中心を求めやすい奇数を用いられることが多い．フィルタ数は，「16・32・64・128・256・512」などが使われる傾向がある．本研究では，初めに96枚を使用する．フィルタ数に関しては，複雑な問題であればフィルタ数を多くして，簡単な問題であればフィルタ数を少なくして試していくという形をとる．input\_shapeは，入力画像のサイズとチャンネル数を表している．本実装では，「32\*32\*3」の形をとる．以下にソースコードを示す．

表4-5　Cond2D

|  |
| --- |
| model.add(conv2d(96, 3, bias\_init=0, input\_shape=(image\_size, image\_size, channel))) |

Conv2Dの所では，「padding=’same’」を使用することによって，出力画像のサイズが変わらないようにパディングしている．フィルタを適用前に0などの要素で周囲を増やしている．これをゼロパディングという．その他にも，「stride=(1，1)」とすることで横に1ピクセル，縦に1ピクセルずつフィルタを適用させることも可能である．指定しない場合は，縦横Ⅰピクセルずつフィルタが適用される．「activation=’relu’」は，活性化関数「ReLU(Rectified Linear Unit)」ランプ関数という．フィルタ後の画像に実施する．入力が0以下の時には出力を0，入力が0より大きい場合はそのまま出力する．以下にソースコードを示す．trunc，cnstは，重み平均0，標準偏差0.01としたガウス分布で初期化，バイアスを2，4，5番目の層の畳込み層及び全結合層では1で，それ以外の層は0で初期化することを行っている．

表4-6　Cond2D-2

|  |
| --- |
| def conv2d(filters, kernel\_size, strides=(1, 1), padding='same', bias\_init=1, \*\*kwargs):      trunc = TruncatedNormal(mean=0.0, stddev=0.01)      cnst = Constant(value=bias\_init)      return Conv2D(          filters, kernel\_size, strides=strides, padding=padding,          activation='relu', kernel\_initializer=trunc, bias\_initializer=cnst, \*\*kwargs      ) |

Max\_pooling2Dは，「2\*2」の大きさの採泥プーリング層である．入力画像内の「2\*2」の領域で最大の数値を出力する．「strides=(2，2)」は，縦横2ピクセルずつフィルタを適用している．以下にソースコードを示す．

表4-7　Max\_pooling2D

|  |
| --- |
| model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2,2))) |

　AlexNetでは，元々BatchNormalizationではなく，Local Response Normalization(LRN)が用いられていた．しかし，Kerasライブラリから削除されてしまったため，BatchNormalizationを用いる．また，LRNは効果が薄く，フィルタサイズも大きくなってしまうので，使用されていない．BatchNoramalizationは，勾配消失・爆発を防ぐための手法である．ネットワークの学習プロセスを全体的に安定化させて学習速度を高めることに成功している．以下にソースコードを示す．

表4-8　BatchNormalization

|  |
| --- |
| model.add(BatchNormalization()) |

　Flattenは，平坦化することである．つまり，1次元ベクトルの形に変換することをいう．以下にソースコードを示す．

表4-9　Flatten

|  |
| --- |
| model.add(Flatten()) |

　Denseは，全結合層のことである．以下の表4-10では，出力が4096となっている．以下にソースコードを示す．

表4-10　Dense

|  |
| --- |
| model.add(dense(4096)) |

　Dropoutは，過学習を予防するためのものである．以下の表4-11では，全結合層とのつながりを「50％」無効化している．以下にソースコードを示す．

表4-11　Dropout

|  |
| --- |
| model.add(Dropout(0.5)) |

　以下の表4-12は，最後の全結合層である．本研究では，「num\_classes=10」となっているので，この全結合層の出力は10となっている．Dense層の所で，CNNの最終的な全結合層の出力次元数の決め方は，判定するクラスの数を指定する．本研究の場合は，「airplane，automobile，bird，cat，deer，dog，frog，horse，ship，truck」の10個を判定するため，全部で１０クラスになる．Softmax関数で総和が1となるように各出力の予測確率を計算する．以下にソースコードを示す．

表4-12　Dense-2

|  |
| --- |
| model.add(dense(num\_classes, activation='softmax')) |

　4.3.2節では，モデルを作成する際に用いたソースコードについての説明をしてきた．基本的には，上記のような流れでモデルの作成を行う．以下の表に，モデル作成の主要となっている部分を示す．

表4-13　create\_model.jpynbの一部

|  |
| --- |
| def conv2d(filters, kernel\_size, strides=(1, 1), padding='same', bias\_init=1, \*\*kwargs):      trunc = TruncatedNormal(mean=0.0, stddev=0.01)      cnst = Constant(value=bias\_init)      return Conv2D(          filters, kernel\_size, strides=strides, padding=padding,          activation='relu', kernel\_initializer=trunc, bias\_initializer=cnst, \*\*kwargs      )  def dense(units, activation='tanh'):      trunc = TruncatedNormal(mean=0.0, stddev=0.01)      cnst = Constant(value=1)      return Dense(          units, activation=activation,          kernel\_initializer=trunc, bias\_initializer=cnst,      )  def AlexNet(image\_size, channel, num\_classes):      model = Sequential()      #conv1      model.add(conv2d(96, 3, bias\_init=0, input\_shape=(image\_size, image\_size, channel)))      #pool1      model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2,2)))      model.add(BatchNormalization())      #conv2      model.add(conv2d(256, 5))      #pool2      model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2,2)))      model.add(BatchNormalization())      #conv3      model.add(conv2d(384, 3, bias\_init=0))      #conv4      model.add(conv2d(384, 3))      #conv5      model.add(conv2d(256, 3))      #pool5      model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2,2)))      model.add(BatchNormalization())      #fc6      model.add(Flatten())      model.add(dense(4096))      model.add(Dropout(0.5))      #fc7      model.add(dense(4096))      model.add(Dropout(0.5))      #fc8      model.add(dense(num\_classes, activation='softmax'))        return model |

　以上が，本研究で扱うモデルについての説明である．最後に，作成したモデルの内容について，下記の表4-14に示す．

表4-14　alexnet\_cifar10\_4096.h5のモデル(きれいに見やすく変更します)

|  |
| --- |
| Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d (Conv2D) (None, 32, 32, 96) 2688    max\_pooling2d (MaxPoolin　 (None, 16, 16, 96) 0    batch\_normalization (Bat (None, 16, 16, 96) 384    conv2d\_1 (Conv2D) (None, 16, 16, 256) 614656    max\_pooling2d\_1 (MaxPool (None, 8, 8, 256) 0    batch\_normalization\_1 (B (None, 8, 8, 256) 1024    conv2d\_2 (Conv2D) (None, 8, 8, 384) 885120    conv2d\_3 (Conv2D) (None, 8, 8, 384) 1327488    conv2d\_4 (Conv2D) (None, 8, 8, 256) 884992    max\_pooling2d\_2 (MaxPool (None, 4, 4, 256) 　　　0    batch\_normalization\_2 (B (None, 4, 4, 256) 1024    flatten (Flatten) (None, 4096) 0    dense (Dense) (None, 4096) 16781312    dropout (Dropout) (None, 4096) 0    dense\_1 (Dense) (None, 4096) 16781312    dropout\_1 (Dropout) (None, 4096) 0    dense\_2 (Dense) (None, 10) 40970    =================================================================  Total params: 37,320,970  Trainable params: 37,319,754  Non-trainable params: 1,216  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

## 特徴ベクトルの抽出方法

特徴ベクトルを抽出するのは，cifar10\_vector\_pickle.jpynbによって行う．このプログラムでは，すでに学習されたモデルから特徴ベクトルを抽出する．

　まず，モデルをロードする．「alexnet\_cifar10\_4096.h5」は，特徴ベクトルを抽出するモデルが保存されているファイルを指定している．以下にコードを示す．

表4-15　モデルのロード

|  |
| --- |
| model = load\_model('alexnet\_cifar10\_4096.h5')  model.summary() |

cifar10データセット内のテストデータ一万件を特徴ベクトル抽出に用いるため，Cifar-10データセットをロードします．以下にコードを示す．

表4-16　cifar10データセットのロード

|  |
| --- |
| from keras.datasets import cifar10  (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data() |

　表4-16のコードまでで，特徴ベクトルを取得できる準備は整っている．特徴ベクトルを抽出する．以下にコードを示す．

表4-17　特徴ベクトルの抽出

|  |
| --- |
| dense =models.Model(inputs=model.input, outputs=model.layers[-3].output)  dense\_out = dense.predict(x\_test) |

一番上の行で，特徴ベクトルを抽出したい層を指定している．本研究では，上から15番目の層から特徴ベクトルの抽出を行う．次の行の所で，特徴ベクトルの抽出を行っている．これで，一万件の特徴ベクトルの抽出をすることができる．抽出した特徴ベクトルの保存については，pickleを用いる．理由としては，numpy配列として扱うことができるため，画像検索評価システムに落とし込みやすいからである．又，保存方法としてcsvも考えていたが，文字列として保存されてしまうので，扱える形に合うるのが少々手間である観点から使用していない．しかし，一万件のデータから一つ一つ取り出すといった作業は，pickleではできないため，必要に応じて使い分ける必要がある．以下に示すコードは，pickleでの保存，ロードの方法である．

表4-18

|  |
| --- |
| with open('cifar10\_vecs4096.pickle', 'wb') as f:    pickle.dump(dense\_out, f) |
| with open('cifar10\_vecs4096.pickle', 'rb') as f:    b = pickle.load(f) |

　上記のように，簡単なコードで一万件の特徴ベクトルを保持することができる． 以上が，特徴ベクトルの抽出方法である．

## 実験・評価の方法について

　実験・評価をするにあたって必要となる特徴ベクトルを抽出して，保存することができた．ここから本研究の本題である画像検索における，認識性能について評価・実験を行う．

　本研究では，特徴ベクトル同士の距離を測る際にユークリッド距離を用いる．

以下にソースコードを示す．

表4-19　ユークリッド距離計算

|  |
| --- |
| def distance(x, y):    return np.sqrt(np.sum((x - y) \*\* 2, axis=-1)) |

　次に，特徴ベクトルを距離の近い順に並び替えをする．ここでは，上位num件の特徴ベクトルを取得する．「argsort」は，出力がインデックスになっている．以下にソースコードを示す．

表4-20　ソート

|  |
| --- |
| def top\_index(v, vecs, num):    return np.argsort(distance(v, vecs))[:num] |

　そして，評価についてである．評価については，まず，上位num件を選択して，取り出した特徴ベクトルの画像のラベルと同じものかを確認する．同じものの場合は，カウントしてresultに１ずつ足していく．以下にソースコードを示す．

表4-21　評価

|  |
| --- |
| def evaluate\_sum(vecs):    num = 30    result = 0    for j in range(0,10000):      top5 = top\_index(vecs[j], vecs, num)      for index in top5:        if y\_test[j] == y\_test[index]:          result += 1    return result |

　最後に，評価をしたい次元数の特徴ベクトルが保存しているファイルを読み込んでくる．例として，4096次元の特徴ベクトルの評価方法について以下にソースコードを示す．また，他の次元数の特徴ベクトルについても同様の方法で評価することができる．

表4-22　4096次元ベクトルの評価

|  |
| --- |
| with open('cifar10\_vecs4096.pickle', 'rb') as f:    vecs = pickle.load(f)  score = evaluate\_sum(vecs)  (score-10000) /(50000-10000) \*100 |

# **実験**

## 

|  |  |  |
| --- | --- | --- |
| 1 | 3 |  |
| 2 | 4 |  |
| 3 | 6 |  |
| 4 | 3 |  |
| 5 | 4 |  |
| 6 | 7 |  |
| 7 | 7 |  |
| 8 | 9 |  |
| 9 | 5 |  |
| 10 | 4 |  |
| 11 | 6 |  |
| 12 | 9 |  |
| 13 | 2 |  |
| 14 | 3 |  |
| 15 | 1 |  |
| 16 | 2 |  |
| 17 | 7 |  |
| 18 | 3 |  |
| 19 | 7 |  |
| 20 | 5 |  |
| 21 | 1 |  |
| 22 | 6 |  |
| 23 | 5 |  |
| 24 | 4 |  |
| 25 | 6 |  |
| 26 | 7 |  |
| 27 | 6 |  |
| 28 | 0 |  |
| 29 | 2 |  |
| 30 | 1 |  |

# **むすび**

## 

# **参考文献**

# その他