**２０２１　年度　　卒業論文**

卒業論文題目

|  |
| --- |
| 深層学習モデルから抽出した特徴ベクトルの画像検索精度と計算時間に関数する評価 |

　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**

目次

[**論文要旨** 2](#_Toc90638517)

[**1.** **まえがき** 3](#_Toc90638518)

[1.1 はじめに 3](#_Toc90638519)

[1.2 本論文の構成 3](#_Toc90638520)

[**2.** **関連研究** 4](#_Toc90638521)

[**3.** **提案方式** 5](#_Toc90638522)

[3.1 提案方式の概要 5](#_Toc90638523)

[3.2 提案方式の手順 5](#_Toc90638524)

[3.2.1 複数モデルの作成について 6](#_Toc90638525)

[3.2.2 特徴ベクトルの抽出 6](#_Toc90638526)

[3.2.3 実験・評価について 7](#_Toc90638527)

[**4.** **実装** 8](#_Toc90638528)

[4.1 実装システム概要 8](#_Toc90638529)

[4.2 実装環境 8](#_Toc90638530)

[4.2.1 環境の概要 8](#_Toc90638531)

[4.2.2 プログラミング言語 8](#_Toc90638532)

[4.2.3 仮想環境 9](#_Toc90638533)

[4.2.4 Pythonライブラリ 9](#_Toc90638534)

[4.3 モデルの作成方法 9](#_Toc90638535)

[4.3.1 AlexNetについて 9](#_Toc90638536)

[4.3.2 モデルの詳しい説明 10](#_Toc90638537)

[4.4 特徴ベクトルの抽出方法 14](#_Toc90638538)

[4.5 実験・評価の方法について 15](#_Toc90638539)

[**5.** **実験** 17](#_Toc90638540)

[5.1 実験目的 17](#_Toc90638541)

[5.2 実験環境 17](#_Toc90638542)

[5.3 実験方法 17](#_Toc90638543)

[5.3.1 実験1 17](#_Toc90638544)

[5.3.2 実験2 17](#_Toc90638545)

[5.4 実験1結果 17](#_Toc90638546)

[5.5 実験2結果 18](#_Toc90638547)

[5.5.1 ラベル別の正答回数 18](#_Toc90638548)

[5.5.2 類似度の近い画像表示 21](#_Toc90638549)

[5.6 考察 25](#_Toc90638550)

[**6.** **むすび** 27](#_Toc90638551)

[謝辞 28](#_Toc90638552)

[**参考文献** 29](#_Toc90638553)

[その他 30](#_Toc90638554)

# **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．また，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うえに，計算コストが莫大となってしまう問題が発生する．この現象を次元の呪いという．また，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．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．これらを念頭に入れて，本研究では，画像認識が有効に行える範囲を調査する．

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る．

# **関連研究**

## AlexNet

AlexNetの構成の提案は，Alex Krizhevsky，Iiya Sutskever，Geoffery E. Hintonらのチームによって開発された物体認識のためのアーキテクチャである．2012年のILSVRCコンペティションにおいて飛躍的な性能を示した．2012年前までのILSVRCでは，画像から特徴量を抽出する際に，人が物体の色・輝度・形等の特徴量を設計していた．しかし，AlexNetの登場により，人が特徴量を設計せずとも，十分なデータが存在すれば，機械自身が特徴量を見る毛出すことができることが示された．

AlexNetは，次の図にあるように3つの畳込み層，2つのプーリング層及び3つの全結合層から構成されている．入力画像は，224\*224サイズの正方形で，出力は1000個の要素を持つ1次元ベクトルであり，それぞれ1000個の分類クラスを表している．本研究では，CIFER-10での学習モデルを作成したかったため，入力画像サイズと出力の分類クラスを変更して，モデルの作成を行った．

引用の図を入れる．

## 特徴ベクトルの距離の測り方

様々な研究で特徴ベクトルの類似性を測る際の手段の1つとして，ユークリッド距離が用いられている．～らの研究では，花押類似検索を行う中で，中間層から画像の特徴抽出を行い，その特徴ベクトルの類似性を確認するためにユークリッド距離を用いている．また，２．１．１に記載した～の研究でも中間層から抽出した特徴ベクトルに対し，ユークリッド距離が短いものは，類似性があるとして評価を行っている．本研究でも，それぞれの次元数の異なる特徴ベクトルから画像検索をする際に、ユークリッド距離を用いて距離の近い画像は，類似性が高いとして実験・評価を行った．また，特徴ベクトルの類似性を測る際にユークリッド距離が有効なものなのか考察を行った．

## 次元の呪いについての研究

# **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，距離が近い順に画像の持っている意味情報の類似度が高いと仮定して，評価を行う．そのときの計算時間と検索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．

1. 学習済みのモデルを作成する
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．
3. STEP-2で抽出した特徴ベクトルを保存する
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．

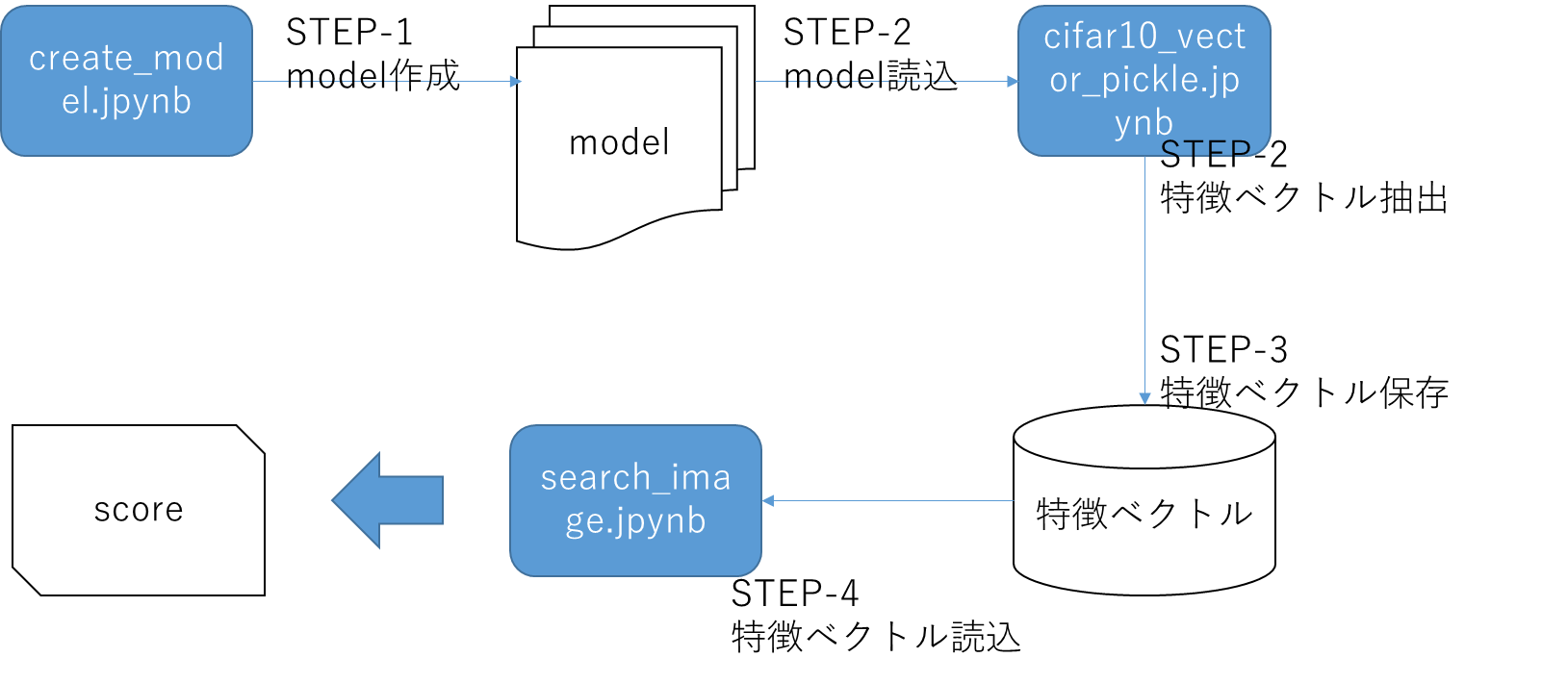


図 3.1　提案方式図

### 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てモデルを作成する．(図 3.2)



図 3.2　次元数を変化させた箇所

### 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

最後に，本研究の画像検索における精度について実験，評価を行う．実験の提案としては，実験1と実験2の2つに分けて行おうと考えている．まず，作成した学習済みモデルから特徴ベクトルを抽出する．その特徴ベクトルを実験・評価プログラムに読み込む．そこで特徴ベクトル間の距離を測る．距離の近い画像を類似度が高いとして上位何件かを選択して取得する．ここまでは，実験1，2ともに一緒の手順で進めていく．

実験1では，取得した画像について，基準となる特徴ベクトルのラベルと同じラベルが何件あるかを数える．そのあっていた数を正答回数として，正答率を求める．以下の図 3.3に評価の流れを示す．

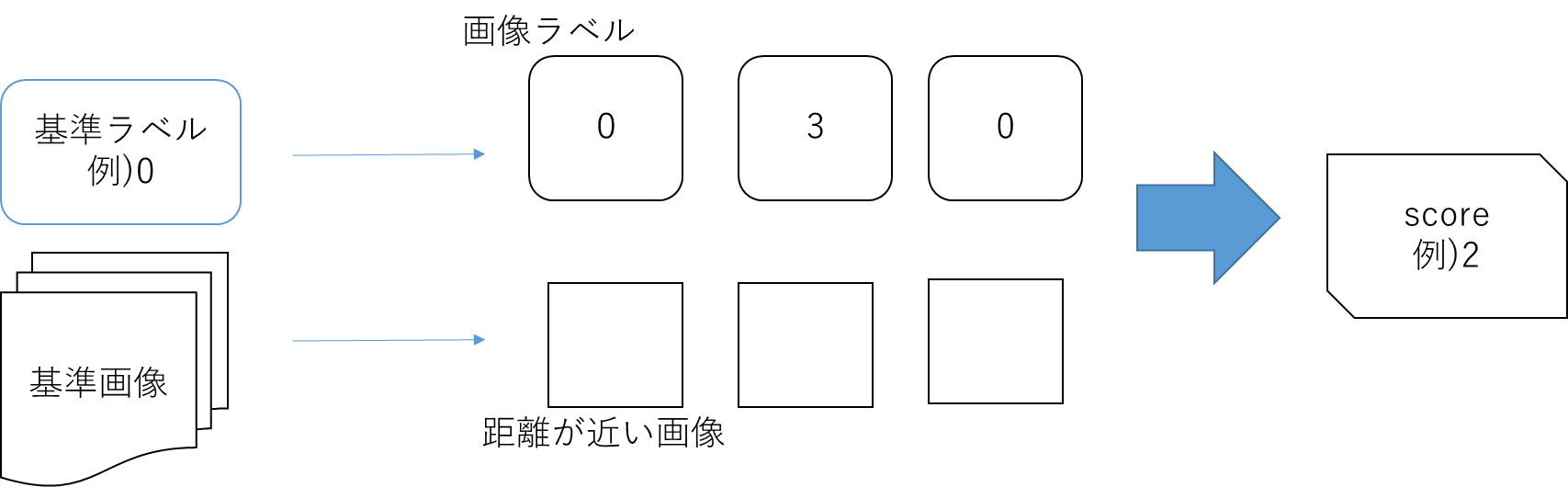


図 3.3　実験1評価提案図

実験2では，画像を取得する際に，ラベルごとの正答回数を数えて，ラベル別の正答率に違いがないかを評価する．また，取得した画像を表示して視覚的に画像を確認することで，距離が近い画像について類似点や共通点がないかを考察する．以下の図 3.4に評価の流れを示す．

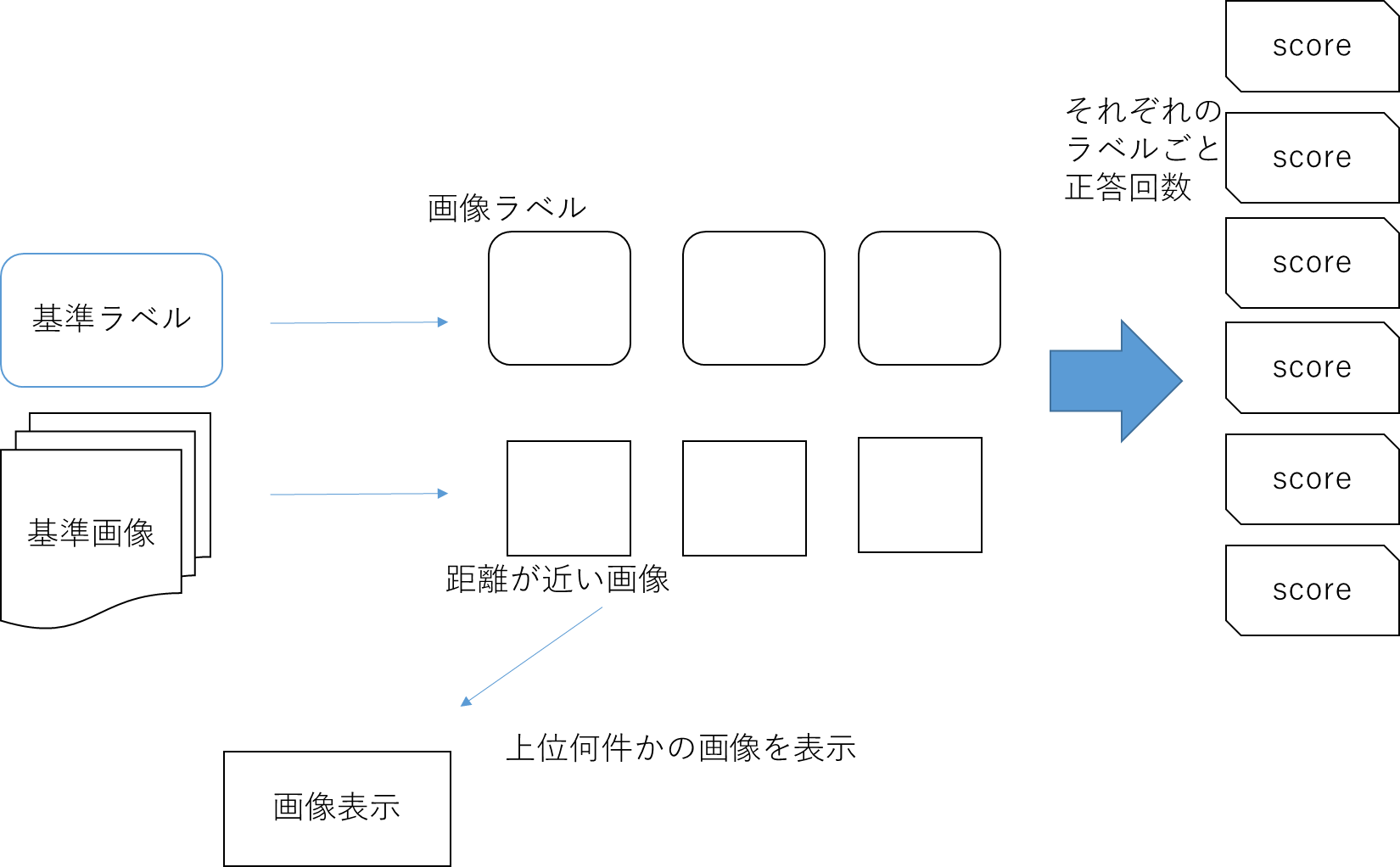


図 3.4　実験2評価提案図

# **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　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 | 数値計算ライブラリ |
| TensorFlow | 2.7.0 | 機械学習ライブラリ |
| Matplotlib | 3.2.2 | グラフ描画ライブラリ |
| 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．数値計算ライブラリとしてNumpyを利用した．Numpyはベクトルや行列の演算を行うのに便利なライブラリとなっており，高速かつ効率的に数値計算を行うことができる．1)機械学習計算のフレームワークであるTensorflowライブラリを利用する．

Tensorflowをそのまま深層学習で利用するよりもKerasを通して利用する方が，深層学習のコードを子水準で短くわかりやすくに記述できる利点がある．2)

Matplotlibはグラフ描画ライブラリである．折れ線グラフや棒グラフ，円グラフなど2次元，3次元のグラフも表示することができる．

PickleはPythonオブジェクトをファイルとして保存・復元するために利用する．3)

## 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．以下の表 4.4に作成したモデルについて表に示す．

表 4.4　作成したモデル

|  |  |  |  |
| --- | --- | --- | --- |
| model | 次元数 | 評価値 | 正答率(%) |
| Alexnet\_cifar10\_10 | 10 | 9583 | 17 |
| Alexnet\_cifar10\_100 | 100 | 24246 | 34 |
| Alexnet\_cifar10\_500 | 500 | 20768 | 25 |
| Alexnet\_cifar10\_1000 | 1000 | 26639 | 38 |
| Alexnet\_cifar10\_2000 | 2000 | 25324 | 34 |
| Alexnet\_cifar10\_3000 | 3000 | 25773 | 35 |
| Alexnet\_cifar10\_4096 | 4096 | 24785 | 34 |

### AlexNetについて

　2012年のImageNetを用いた画像認識コンペティション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ある．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る．具体的な構成は以下の表 4.5に示す．

表 4.5　AlexNetの構成

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 4.5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 4.6に示す．また，変更箇所は赤字で示す．

表 4.6　AlexNetを変更した構成

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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．以下の表 4.7にソースコードを示す．

表 4.7　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．以下の表 4.8にソースコードを示す．

表 4.8　Conv2D

|  |
| --- |
| 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．以下の表 4.9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 4.9　Conv2Dの中身

|  |
| --- |
| 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．以下の表 4.10にソースコードを示す．

表 4.10　Maxpooling2D

|  |
| --- |
| 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．以下の表 4.11にソースコードを示す．

表 4.11 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．以下の表 4.12にソースコードを示す．

表 4.12　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．以下の表 4.13にソースコードを示す．

表 4.13 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 4.14では，全結合層とのつながりを「50％」無効化している．以下の表 4.14にソースコードを示す．

表 4.14 Dropout

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

　以下の表4-15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．以下の表 4.15にソースコードを示す．

表 4.15 Dense(識別層)

|  |
| --- |
| 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．以下表 4.16に，モデル作成の主要となっている部分を示す．

表 4.16 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 4.17に示す．

表 4.17　alexnet\_cifar10\_4096.h5のモデル構成

|  |  |  |
| --- | --- | --- |
| 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．以下の表 4.18にソースコードを示す．

表 4.18　モデルのロード

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

CIFAR-10データセット内のテストデータ1万件を特徴ベクトル抽出に用いるため，CIFER-10データセットをロードします．以下の表 4.19にソースコードを示す．

表 4.19　CIFER-10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 4.18から表 4.19のプログラムまでで，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．以下の表 4.20にソースコードを示す．

表 4.20　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，1万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，扱える形に合うが少々手間である観点から使用していない．しかし，1万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．以下の表 4.21に示すコードは，pickleでの保存，ロードの方法である．

表 4.21　保存と読込

|  |
| --- |
| 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1万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．ユークリッド距離の計算式は以下の通りである．

以下の表 4.22にソースコードを示す．

表 4.22　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．以下の表 4.23にソースコードを示す．

表 4.23　ソート方式

|  |
| --- |
| 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．以下の表 4.24にソースコードを示す．

表 4.24　評価方法

|  |
| --- |
| 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の表 4.25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 4.25　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**

## 実験目的

　本研究では，作成した次元数の異なるモデルの認識性能についての評価として，距離と正解ラベル，そして，処理にかかった時間を計測することで，認識処理に有効なのかを調査する．本章では，これらの評価について結果を確認することで，提案方法の評価，考察をすることを目的とする．

## 実験環境

　4章の実装環境と同様．

## 実験1

### 実験目的

### 実験方法

実験1では，それぞれの次元数の特徴ベクトルの検索精度とプログラムを実行するのにかかった計算時間を計測する．操作は以下の手順で進める．

1. 用意した特徴ベクトルを実験・評価プログラムに読み込む．
2. 読み込んだ特徴ベクトルを用いて，ユークリッド距離を測る.
3. 距離の近い順に並び替えをする．この際，特徴ベクトルの配列のインデックス順に並べる．
4. 上位20件を選択して取得する．
5. 取得した上位20件の画像について，基準となる特徴ベクトルのラベルと同じラベルが何件あるのかを数える．
6. 数えたものをもとに正答率を計算し，検索精度をみる．
7. ここまでにかかった計算時間を測る．

### 実験結果

5.3.1で示した流れで実験を行った．実験1では，それぞれ次元数の異なる特徴ベクトルでユークリッド距離を測り，近い画像の中にいくつ同じ正解ラベルがあるかを数え，正答率を求めた．以下の図 5.1に示す．また，図中のaccは正答率をcostはプログラムの計算時間を表している．



図 5.1　次元数別　正答率と計測時間

　実験1の結果としては，次元数8192の特徴ベクトルが最も良い検索精度となった．計算時間は，次元数が増えるのと比例して時間がかかるようになった．

### 考察

実験１の結果としては，特徴ベクトルに含まれている意味情報は，全結合層の前の層の所では，有効に扱えるとは確認できなかった．まず，考えられる理由としては，モデルを作成する際に，変更した次元数が的確ではなかったことから問題が発生した可能性がある．しかし，次元数を変化させていない4096次元のモデルの特徴ベクトルに関しても認識性能の結果が良くなかったことからその可能性は低いと考えられる．なので，転移学習を行っていない本研究では，低い認識性能しか出すことができなかったのではないかと考えられる．

次に考えられる理由としては，特徴ベクトルについて評価をするときに，特徴ベクトル間のユークリッド距離を求めて，その距離が近いものを画像の持つ意味情報の類似度が高いとしていたが，ユークリッド距離の近さでは，本研究で期待していた検索精度の向上を促すための類似度を測ることが出来ていなかったということが考えられる．

## 実験2

### 実験目的

実験1の結果を見たが，確かな検索精度を確認することが出来なかった．なので，実験2を行い，類似度の高いとされた上位20件の画像を表示した．そして，視覚的な特徴の類似している点があるかを評価した．その過程で，ラベルによって検索精度に大きく違いが出ていることが確認できたので，ラベル別の正答回数についても確認しようと考えた．

5.3.2で示した流れで実験を行った．実験2では，それぞれの特徴ベクトルのラベルすべての正答率ではなく，ラベルごとの正答率について評価を行った．また，類似度の近い画像に視覚的な特徴はないかを調査した．

### 実験方法

実験2では，画像を表示した際に視覚的な類似度がないかとラベルごとの正答回数がそれぞれいくつなのか計測する．操作は以下の手順で進める．

1. 実験1の4)までの手順を実行する．
2. 実験2の1)の時に，ラベルごとの正答回数を数える．
3. 取得した20件の画像をユークリッド距離の近い順に表示させる

### 実験結果

ラベル別正答回数

次元数100～4096の特徴ベクトルのラベルごとの正答回数についての図を以下のそれぞれ図 5.2，図 5.3，図 5.4，図 5.5，図 5.6，図 5.7，図 5.8に示す．

図 5.2　次元数100　ラベル別正答回数

図 5.3　500次元　ラベル別正答回数

図 5.4　次元数1000　ラベル別正答回数

図 5.5　次元数2000　ラベル別正答回数

図 5.6　次元数3000　ラベル別正答回数

図 5.7　次元数4096　ラベル別正答回数

図 5.8　次元数8192　ラベル別正答回数

　特徴ベクトルごとに多少ラベルごとの検索精度に違いはありますが，全体的に見るとラベル1「automobile」が1番良い検索精度が出せていると見受けられる．また，ラベル3「cat」が1番低い検索精度が出ていると見受けられた．このようにラベルによって正答率に大きな差が出ていることが確認できた．以下の図 5.9に図 5.2～図 5.8までの正答回数の平均を示す．

図 5.9　各次元数　ラベル別正答回数　平均

　5.5.2では，5.5.1の結果を踏まえて，実際に正答回数が1番良いラベルの上位20件と1番悪いラベルの上位20件の画像を表示させる．

実験2

　大量の画像とそのラベルを見ている際に確認できたことは，ラベルによって検索精度に大きな差があったことである．検索精度の低いラベルに関しては，全く違う画像が表示されることが多々確認された．また，検索精度の高いラベルに関しては，ラベルが同一の画像もしくは，基準となる画像に近しいものが表示されていた．例えば，ラベル9「truck」が基準の時にラベル1「automobile」が多く見受けられた．以下の図 5.10～図 5.23にそれぞれの次元数ごとのラベル1「automobile」とラベル3「cat」の画像について，No.0を基準の画像として類似度の高いとされた上位20件の画像を示す．



図 5.10　次元数100　ラベル1「automobile」[37]



図 5.11 次元数100　ラベル3「cat」[0]



図 5.12 次元数500　ラベル1「automobile」[37]



図 5.13 次元数500　ラベル3「cat」[0]



図 5.14 次元数1000　ラベル1「automobile」[37]



図 5.15 次元数1000　ラベル3「cat」[0]



図 5.16 次元数2000　ラベル1「automobile」[37]



図 5.17 次元数2000　ラベル3「cat」[0]



図 5.18 次元数3000　ラベル1「automobile」[37]



図 5.19 次元数3000　ラベル3「cat」[0]



図 5.20 次元数4096　ラベル1「automobile」[37]



図 5.21 次元数4096　ラベル3「cat」[0]



図 5.22 次元数8192　ラベル1「automobile」[37]



図 5.23 次元数8192　ラベル3「cat」[0]

　上記の結果からも分かる通り，ラベル1では，同一のラベルの画像とトラック等の車と似た画像が表示されていることが多かった．ラベル3では，猫以外の写真が出ることが多かった．両方ともに言えることは，次元数が増えるごとに表示結果がより基準となる画像に近しいものが出るようになった．次元数1000を超えたあたりから良くなっている．

### 考察

　実験2の結果としては，実験1ではあまり芳しくない結果が出ていたため，想像以上に良い結果を確認することができた．ラベルごとに画像を確認していくと，

ラベル0「airplane」は，平均的に見ても検索精度は良い方であった．検索結果の中には，ラベル2「bird」が多く含まれていることが確認できた．

ラベル1「automobile」は，実験結果でも述べているが，最も検索精度が良かった．検索結果の中には，ラベル9「truck」が多く含まれていた．2つの写真は，形状や写真の構図の特徴が似ていることから検索上位に表示されているのではないかと考えられる．また，その反対のことも言えて，ラベル9が基準の画像だった場合，ラベル1が多く含まれているように見受けられた．

ラベル2「bird」は，あまり良くない検索精度だった．先ほどラベル0の時は，ラベル2が含まれることが多かったと述べたが，ラベル2では，ラベル0をそこまで確認することが出来なかった．ラベル2の画像の中には，羽を広げている写真だけでなく，羽をしまっている写真や顔がアップで写っている写真も多く見受けられたので，ラベル0の画像とは類似度が低いとされたと考えられる．

ラベル3「cat」は，実験結果でも述べたが，最も検索精度が低かった．考えられる理由としては，他のラベルの画像である「犬」，「馬」等の動物が検索結果に表示されていることが多かった．ラベル4「deer」に関しても同じことが言えた．形状や写真の構図等の特徴が似ていることから類似度が高いとされたと考えられる．

ラベル5「dog」，ラベル６「frog」，ラベル7「horse」

全体を通してみると，検索精度は低いものの抽出した特徴ベクトルには，画像の意味情報が特徴として含まれていると考えられる．また，ラベルによって検索精度が異なることも確認することができた．検索結果として出された画像は，対象物か背景の色が近しい画像が多く選ばれていたように見受けられた．そのことから考えるに対象物がはっきりと形状がわかる画像，そして，色が識別しやすく，単一な画像であれば同一のラベルとして検索上位に画像が表示されるのではないかと考えられる．

# **むすび**

　本研究では，モデルの次元数を変化させることで，画像検索を正常に行える有効な範囲を調査し，評価した．全結合層の2つ手前の次元数が異なるモデルから特徴ベクトルを抽出する．抽出した特徴ベクトルが画像検索のできる情報を持っているのか調査した．次元数が増えるほど計算時間がかかることが確認できた．しかし，画像検索精度に関しては，次元数が増えることで検索精度が上がるということはなかった．

# 謝辞

　本研究を行うにあたり，多くの助言をくださった鷹野孝典教授に心から感謝いたします．

研究を一緒に進めながら様々な意見を出し合い，本研究の方向性の確立にご協力くださった鷹野研究室の４年生のメンバー1)に深く感謝いたします2)．

# **参考文献**

1. PythonのNumpyの使い方をわかりやすく解説！　<https://www.tech-teacher.jp/blog/python-numpy/(2021/12/18)>
2. ディープラーニングを簡単に手を動かして学ぼう　　<https://algorithm.joho.info/machine-learning/python-keras-nyumon/>
3. Pickleでオブジェクトを保存する方法を解説！　<https://www.sejuku.net/blog/31480>

# その他