**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**

ユーザが目的の画像に辿り着くための画像検索機能の重要性が増している．このような状況において，深層学習によるCNNの登場により画像検索機能は劇的に向上した．そこで，深層学習モデルの中間層から抽出できる特徴に目が向けられている．特徴量は，高次元になるほど検索精度が良くなるが計算時間が増加する．反対に低次元にすると計算時間は早くなるが検索精度が落ちてしまうことがわかっている．なので，本研究では学習済みモデルを複数用意し，それぞれ次元数の異なる特徴ベクトルを抽出して画像検索精度および計算時間を最適化するための特徴ベクトル次元数について評価する．実験方法としては，学習済みモデルを特徴抽出器として，任意の中間層から画像の特徴ベクトルを抽出し，ベクトル間のユークリッド距離を測ることで評価する．ユークリッド分離が小さいほど類似性が高いとする．結果は，各次元数の特長ベクトルの正答率は決して良い結果ではなかった．しかし，各ラベルの正答回数，類似画像を視認することで特徴ベクトルが形状やエッジのポイントを特徴として持っているのでは考えられた．結果としては，次元数1000の特徴ベクトルが画像検索精度と計算時間の観点から見て最も優れていたと結論付けた．

目次

[**1.** **まえがき** 3](#_Toc90986574)

[1.1 はじめに 3](#_Toc90986575)

[1.2 本論文の構成 4](#_Toc90986576)

[**2.** **関連研究** 5](#_Toc90986577)

[2.1 AlexNet 5](#_Toc90986578)

[2.2 特徴ベクトル抽出 5](#_Toc90986579)

[2.3 特徴ベクトルの距離の測り方 6](#_Toc90986580)

[2.4 次元の呪いについての研究 6](#_Toc90986581)

[**3.** **提案方式** 7](#_Toc90986582)

[3.1 提案方式の概要 7](#_Toc90986583)

[3.2 提案方式の手順 7](#_Toc90986584)

[3.2.1 複数モデルの作成について 8](#_Toc90986585)

[3.2.2 特徴ベクトルの抽出 9](#_Toc90986586)

[3.2.3 実験・評価について 9](#_Toc90986587)

[**4.** **実装** 11](#_Toc90986588)

[4.1 実装システム概要 11](#_Toc90986589)

[4.2 実装環境 11](#_Toc90986590)

[4.2.1 環境の概要 11](#_Toc90986591)

[4.2.2 プログラミング言語 11](#_Toc90986592)

[4.2.3 仮想環境 12](#_Toc90986593)

[4.2.4 Pythonライブラリ 12](#_Toc90986594)

[4.3 モデルの作成方法 12](#_Toc90986595)

[4.3.1 AlexNetについて 12](#_Toc90986596)

[4.3.2 モデルの詳しい説明 14](#_Toc90986597)

[4.4 特徴ベクトルの抽出方法 17](#_Toc90986598)

[4.5 実験・評価の方法について 18](#_Toc90986599)

[**5.** **実験** 20](#_Toc90986600)

[5.1 実験目的 20](#_Toc90986601)

[5.2 実験環境 20](#_Toc90986602)

[5.3 実験1 20](#_Toc90986603)

[5.3.1 実験目的 20](#_Toc90986604)

[5.3.2 実験方法 20](#_Toc90986605)

[5.3.3 実験結果 20](#_Toc90986606)

[5.3.4 考察 21](#_Toc90986607)

[5.4 実験2 21](#_Toc90986608)

[5.4.1 実験目的 21](#_Toc90986609)

[5.4.2 実験方法 22](#_Toc90986610)

[5.4.3 実験結果(ラベル別正答回数) 22](#_Toc90986611)

[5.4.4 実験結果(画像表示) 26](#_Toc90986612)

[5.4.5 考察 29](#_Toc90986613)

[**6.** **むすび** 31](#_Toc90986614)

[謝辞](#_Toc90986615)

[**参考文献**](#_Toc90986616)

# **まえがき**

## はじめに

ソーシャルネットワーキング(SNS)や写真共有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．このような状況において，ユーザが目的の写真や画像にサ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．従来研究において画像検索機能に関して多くの研究が推進されているが，深層学習による畳み込みニューラルネットワーク(Convolutional Neural Network，CNN)の登場により，画像認識性能が劇的に向上した．

CNNは，2012年の画像認識コンテストILSVRCの結果において優れた画像認識精度を達成した1)，それ以降CNNとその応用は，深層学習による画像分類モデルとして代表的な手法となっている．一方で，事前学習した深層学習画像分類モデルの中間層から抽出した画像の特徴ベクトルは，従来の内容に基づく検索(Content-based Information Retrieval, CBIR)における画像検索システムに適用可能であり，十分な規模数の学習データで学習した深層学習分類モデルの利用により，従来よりも高精度の画像検索が実現できることが期待されている．本研究においても，深層学習画像分類モデルの中間層から抽出した画像の特徴ベクトル2)を用いて，深層学習がどのような情報をもとに画像検索を行っているかを考察することにより，より良い画像検索サービスが作れる実現できるのではないかと考えている．ここで，画像特徴ベクトルを用いた画像検索機能において，画像の特徴量が高次元になるほど，画像検索精度は高くなる．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，画像検索の計算時間が大きく増加してしまい，解析が困難になってしまう上に，プログラムの実行時間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して知られている．また，次元数が小さくなり過ぎると，意味情報が損失し，画像検索精度が低くなるが，次元数が少ないために，計算時間が減少するといったことが起こる．本研究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．本研究では，このような考察に基づいて，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する評価を行うための手法を提案する．提案手法では，抽出した画像特徴ベクトルの次元数と画像検索精度の関係性を調べるために，ベクトル次元数を数十～数千次元に変化させて抽出する．このために，深層学習モデルのSoftmax層の直前の中間層の次元数を変化させて，目的の次元数の画像特徴ベクトルを抽出する．画像特徴ベクトルの類似性を図るためにユークリッド距離を用いる．1)3)ユークリッド分離が小さいほど類似性があるとして評価する．抽出した特徴ベクトルがどのような意味情報を持っているのか評価するために，画像検索による検索精度を評価する．画像検索精度と実行時間を評価することにより，両者を最適化する画像特徴ベクトル次元数を導出することが可能となる．

実験は，プログラミング言語としてPythonを使用し，パッケージなどのインストールが必要ないことやGPU・TPUが無償で使用することができるGoogle Colaboratoryを使用する．実験プログラムは，モデル作成，特徴ベクトル抽出，画像検索評価の3つのプログラムを作成する．また，評価用の画像データセットとしてCIFER-10を用いて，深層学習モデルであるAlexNetのニューラルネットワーク構造で事前学習済みモデルを作成した．10種類の各ラベルを評価することで，画像検索に有効な特徴ベクトルの次元数と計算時間の関係性を導出し，画像検索精度および計算時間を最適化するための画僧特徴ベクトル次元数について考察する．

## 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，実験・評価方法，それらにまつわる関連研究について述べる．第３章では，本研究で提案する，深層学習モデルの作成，特徴ベクトルの抽出，画像検索精度の評価方法について述べる．第４章では，実験システムの実装について述べる．第５章では，実験概要と，実験結果について述べる．第６章では，結論と今後の展望について述べる．

# **関連研究**

第２章では，モデルの作成手法，実験・評価方法，それらにまつわる関連研究について述べる．

## 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年の画像認識コンペティションILSVRCにおいて飛躍的な性能を示した．2012年前までのImageNet Large Scale Visual Recognition Challenge(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つけ出すことができることが示された．

AlexNetは，図 2.1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．



図 2.1　出典:[Alex2012]p.51)

## 特徴ベクトル抽出

深層学習での特徴ベクトルの抽出は，主に学習モデルの適当な中間層から抽出される．学習済みネットワークを固定し，特徴抽出器として利用する．これは，CNNの最も簡単な利用方法である．つまり，入力画像をフィードフォワードし，適当な中間層の出力する値をそのまま特徴ベクトルとして用いることで，深層学習の知識やCNNに関する知識がなくとも手軽に利用することができると中山英樹は述べている．本研究でも，識別層の1つ手前の全結合層から特徴ベクトルの抽出を行い，実験に利用している．以下の図 2.2は，学習済みモデルから特徴ベクトルを抽出する位置を示した．

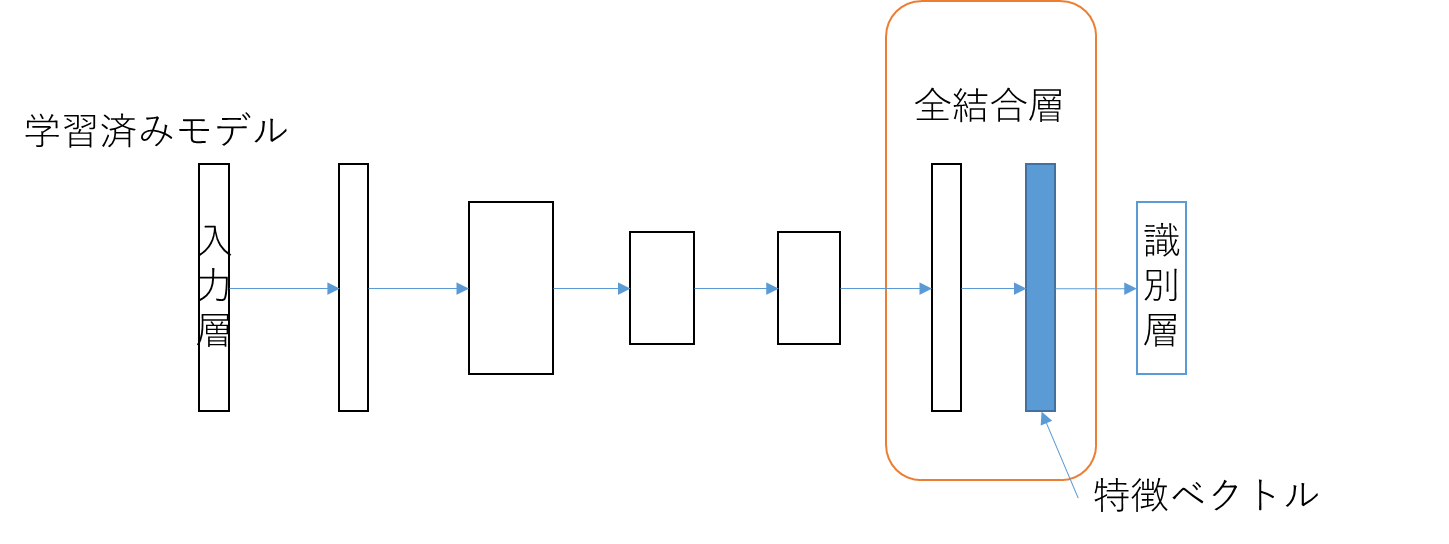


図 2.2　学習済みモデルを用いた特徴抽出

## 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．鬼塚洋輔らの研究では，花押類似検索を行う中で，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に記載したAlex Krizhevskyら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

　機械学習においてクラスタリングはラベルなしのデータ分類を行う教師なし学習として重要である．しかし，高次元データのクラスタリングでは次元の呪いが生じるため，多くの場合は，前処理として次元削除を行うことが必要になると高橋春輝，竹林高志らは述べている．本研究では，CIFER-10データセットを学習等に用いるため次元数が高次元になりすぎることは想定しづらいが，実際の画像検索には，もっと高次元の画像が入ることが想定されるので，本実験でも次元の呪いの観点を考慮して実験・評価を行う．

# **提案方式**

第３章では，本研究で提案する，深層学習モデルの作成，特徴ベクトルの抽出，画像検索精度の評価方法について述べる．

## 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では，入力に近い層から識別層に近づくにつれ，低次元の視覚的特徴からデータセットに特化した意味的な特徴に構造化されることが知られている．2)つまり，CNNの中間層から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．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1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．

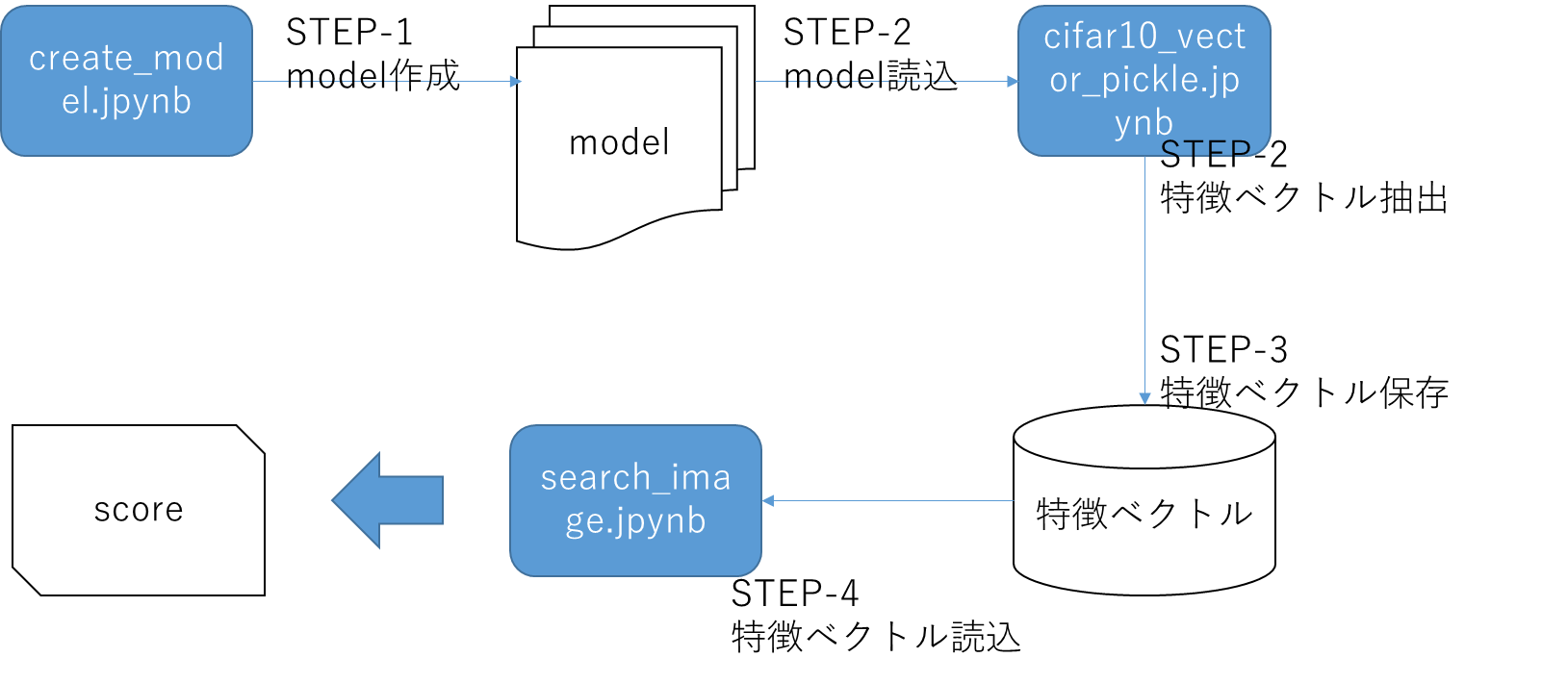


図 3.1　提案方式図

### 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．

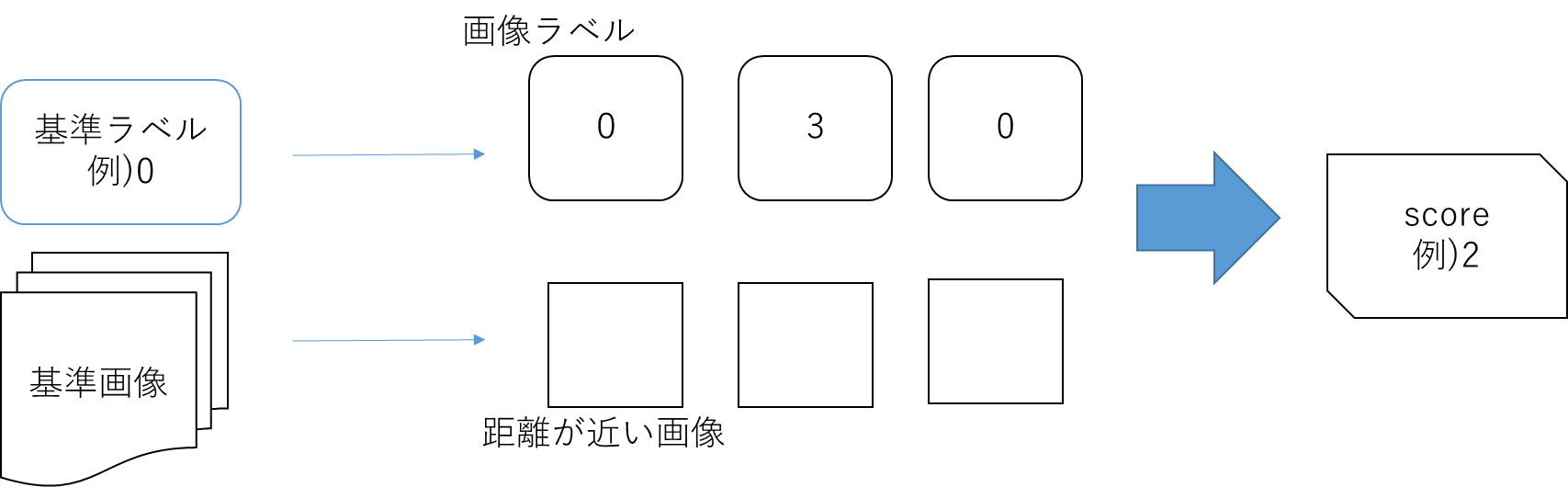


図 3.3　実験1評価提案図

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．

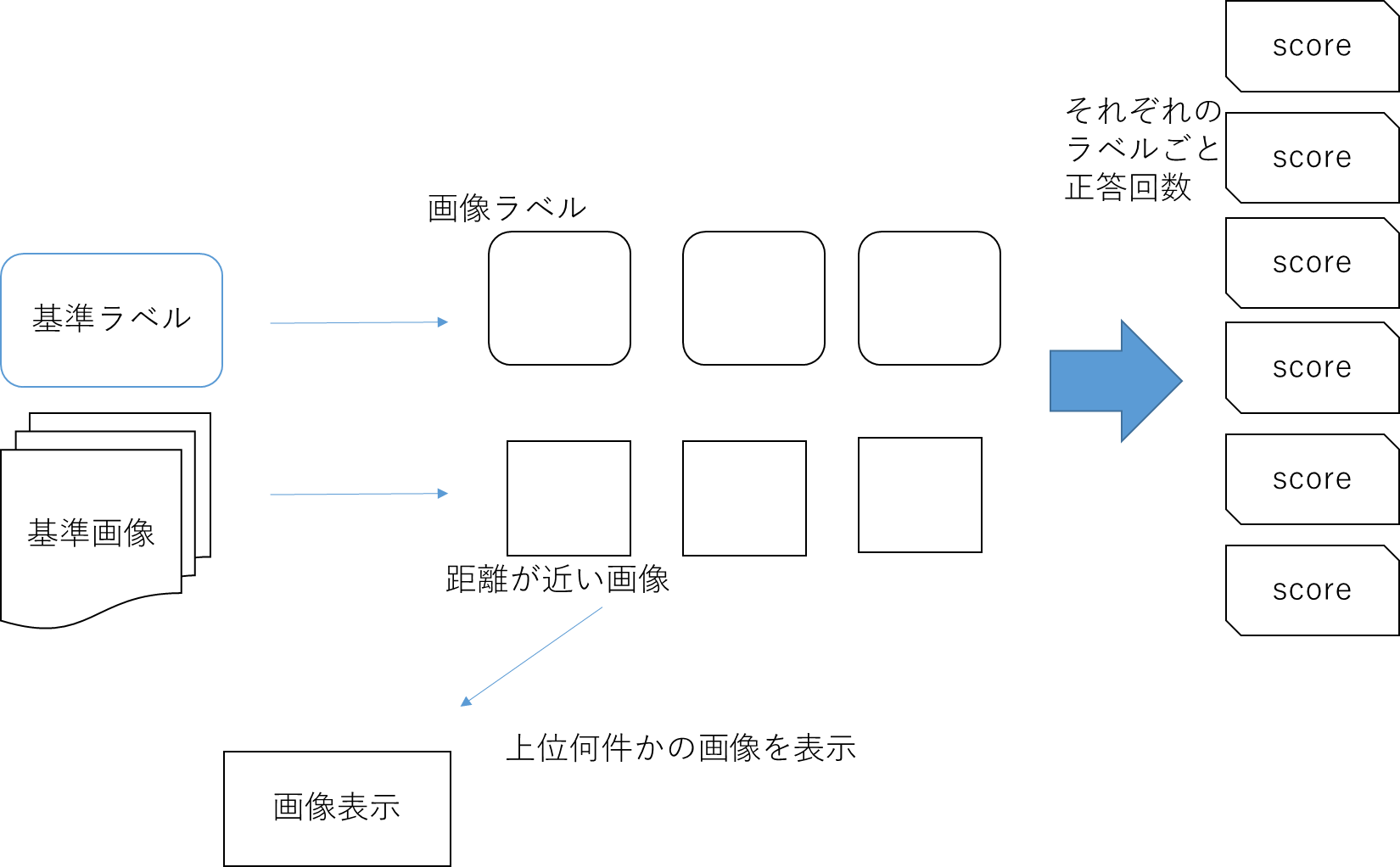


図 3.4　実験2評価提案図

# **実装**

第４章では，実験システムの実装について述べる．

## 実装システム概要

実装システムでは，画像検索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，CIFER-10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，計算時間を測り，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 | 20H2 |
| GPU | Tesla P100-PCIE-16GB |

表 4.2　開発・実行環境

|  |  |  |
| --- | --- | --- |
| 名称 | バージョン | 説明 |
| Python | 3.7.12 | プログラミング言語 |
| Google Colaboratory |  | 実行環境 |

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 |
| keras | 2.7.0 |  |
| Matplotlib | 3.2.2 | グラフ描画ライブラリ |
| Pickle |  |  |

### 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．

### 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．

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．4)

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．6)

## 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dense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\_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\_cifar10\_8192 | 8192 | 80697 | 42 |

### AlexNetについて

2012年のImageNetを用いた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．1)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．4)

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 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，作成した識別層の直前の層が4096次元の時のモデルについて，以下の表 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．4)

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データセットを読み込む．以下の表 4.19にソースコードを示す．

表 4.19　CIFER-10データセットの読み込み

|  |
| --- |
| 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) |

1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，検索精度について評価・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numSameLabel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\_ratio(vecs, num\_top=5):    numSameLabel = 0    numSameindex = 0    for i, vec in enumerate(vecs):      top\_index\_list = top\_index(vec, vecs, num\_top)      for index in top\_index\_list:        if index != i and y\_test[i] == y\_test[index]:          numSameLabel += 1        if index == i:          numSameindex += 1      numEvaluate = len(vecs) \* num\_top - numSameindex    return numSameLabel / numEvaluate |

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)  evaluate\_raito(vecs) |

実験2の評価方法の1つである画像を視認で確認するために，取得した上位num件の画像の表示について以下の表 4.26にソースコードを示す．下記では，20件の画像を表示するように設定している．

表 4.26　上位num件の画像表示

|  |
| --- |
| top5 = top\_index(vecs4096[bigin], vecs4096, 20)  print(top5)  i = 0  for index in top5:    plt.figure()    plt.imshow(x\_test[index])    print(y\_test[index]) |

# **実験**

第５章では，実験概要と，実験結果について述べる．

## 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4.2に記載してある環境を参照．

## 実験1

### 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

図 5.1から次元数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．そのため次元数8192は，最も検索精度が良いが計算時間は，最もかかることが確認できる．

### 実験目的

実験1の目的は，本研究で作成したモデルから抽出した特徴ベクトルが画像検索をする際に，その検索精度がどこまで有効なものが出るのかを検証・評価をするために行う．

### 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，識別層の直前の全結合層の位置では，有効に扱えるとは確認することができなかった．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次元数4096のモデルの特徴ベクトルに関しても検索精度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，低い検索精度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，ユークリッド分離が小さい画像は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できていなかったということが考えられる．または，特徴ベクトルが持つ意味情報は，全体の画像では十分な正答率を得ることはできなかったが，実際に類似度が高いとされた画像を表示して確認することで，ユークリッド距離が近いことでの共通点がないかを確認しようと考えた．そのことから実験2を行うことにした．

## 実験2

### 実験目的

実験1の結果から，確かな検索精度を確認することができなかった．したがって，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，各ラベルの正答回数についても確認しようと考えた．

実験の流れは以下の5.4.2に示した．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回数ではなく，各ラベルの正答回数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100～8192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．(何個かは付録行きですかね)

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

各ラベルの検索精度には，特徴ベクトルごとに多少の違いはあるが，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

5.4.4では，5.4.3の結果を踏まえて，実際に正答回数が1番良いラベル1の上位20件と1番悪いラベル3の上位20件の画像を表示させる．

### 実験結果(画像表示)

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，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，ラベルによって正答回数に差があることが確認できた．また，次元数が異なっていても正答回数の良いラベル，悪いラベルは基本的には変化がなく，同じ結果となっていた．このことから特徴ベクトル間のユークリッド距離から得た類似度は何かしらの意味情報を持っていると考えられる．ここからCIFER-10に含まれる各ラベルについて考察していく．

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」が多く含まれていた．しかし，ラベル2「bird」は，あまり良くない検索精度だった．ラベル2では，ラベル0を多くは確認することができなかった．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画像が多数あったと考えられる．

ラベル1「automobile」は，実験結果でも述べたが，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．これらの写真は，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．ラベル8「ship」も検索結果の中に多数含まれていた．また，その反対のことも言えて，ラベル8やラベル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」は，正答回数がほぼ同じであった．共通して言える点は，形状がはっきりとしている写真が多かったことが言える．その中でも色が単一の物は形状が捉えたやすいのか検索精度はよかった．

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．車やトラック等の乗り物類が検索精度としては良かった．乗り物の写真に共通して言えることは，形状がはっきりと示されていて，CIFER-10の画素が荒い画像でも人が読み取れる画像が多かった．ユークリッド距離の近さで見た特徴ベクトルでは，画像の形状，エッジが主に特徴として含まれていて，その類似度が高い画像が検索上位に出されているのではないかと考えられる．また，色に関しての情報も含まれているのではと考えていたが，色が単一なものは形状がわかりやすく，色がはっきりとわかっていても検索精度が良くない写真もあったことからユークリッド距離での類似度検索では，特徴ベクトルに色の情報が含まれるとは考えられなかった．

# **むすび**

本研究では，3章で提案手法を示し，それを実装するための手順や説明を4章に記載した．本研究は，全結合層の次元数が異なる学習済みモデルを7つ作成し，それぞれのモデルの識別層から1つ手前の全結合層より特徴ベクトルを抽出した．そして，各特徴ベクトルの検索精度を検証するため，ユークリッド距離を利用して，ベクトルの持つ類似性を調査し，それを正答率として評価する方法を行った．

この方法を行った結果，まず，本研究で作成した各特徴ベクトルの正答率は，画像検索が正確に行えると位置づけられる範囲には及ばなかったといえる．最も良かった次元数は8192で正答率は約42%となった．しかし，計算時間の観点も考えると次元数8192は時間がかかり過ぎている．このことから次に正答率の良い次元数1000が本研究では，次元数，計算時間の両方の観点から最も良かったと考えられる．次に，各ラベルで正答率に差がないかを調査した．結果，ラベルによって正答率が良いラベルと悪いラベルがあることが判明した．各特徴ベクトルによってのバラつきはなく，多少の誤差はあるものの同じ結果のグラフが出来上がった．すべての平均を求めて，グラフに示した結果，ラベル1が1番良い結果となり，ラベル3が悪い結果となった．このことから特徴ベクトルは特定の意味情報を持っており，ユークリッド距離で測れる類似度があると考えた．それを確認するために，類似度が高いとされた上位20件の画像を表示して，視覚的な特徴の観点から共通点はないかの調査を行った．その結果，正答率が良い画像には，いくつかの共通点があると見受けられた．1つは，形状がはっきりしている画像であること．2つ目は，背景がシンプルもしくは単一な色で表されている画像であることが考えられた．結果から，ユークリッド距離で測ることのできる特徴ベクトルは，主に形状やエッジのポイントといった情報を持っていると結論付けられた．

今後の展望として，本研究の提案手法では，画像検索システムを最も効果的に計算時間を早くすることを決定づけることはできなかった．しかし，ユークリッド距離で類似性を測った際の特徴ベクトルの持っている意味情報について調査し，評価することができたと考えている．本研究で行った提案手法を応用させる．ユークリッド距離以外での類似性の評価，異なる深層学習モデルを使用した特徴ベクトル抽出を行うことで，得られる意味情報の調査をすることができるのではないかと考えている．また，本提案手法を用いることで，処理速度の速い画像検索システム作成に貢献できることを期待している．

# 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年生のメンバーに深く感謝いたします．

# **参考文献**

1. [Alex2012] Alex Krizhevsky，Ilya Sutskever，Geoffrey E. Hinton：ImageNet Classification with Deep ConvolutionalNeural Networks（2012）
2. [中山2015]　中山英樹：深層畳み込みニューラルネットワークによる画像特徴抽出と転移学習（2015）
3. [鬼塚2018] 鬼塚洋輔，山田太造，井上聡，内田誠一：花押類似検索のための畳み込みオートエンコーダによる画像特徴抽出（2018）
4. フランソワ・ショア，巣籠悠輔　監訳，株式会社クイープ　訳：PythonとKerasによるディープラーニング．p32~3539~41，p124~186　株式会社マイナビ出版（2018）
5. [高橋2020] 高橋春輝，竹川高志：ラベル情報の一般化によるLaplacian EigenmapsとLinear Discriminant Analysisの体系化（2020）
6. [新井2020]　新井悠介，天方大地，原隆治，藤田澄夫：高次元データに対するグラフインデックスを用いた近似範囲検索アルゴリズム．（2020）
7. [岡本2020]　岡本弘野，鈴木雅大，松尾豊：深層生成モデルを用いた高次元特徴量のための分布外検知．（2020）
8. GitHub: [uran110/AlexNet-Cifar10: AlexNet experiment on Cifar-10 (github.com)](https://github.com/uran110/AlexNet-Cifar10) 2021/12/21
9. Pickleでオブジェクトを保存する方法を解説！　<https://www.sejuku.net/blog/31480>