# 知能情報実験 III（データマイニング班）
# 指紋認証を用いた CNN の手法についての模索

**185710A 金城海斗**
**185714C 石橋竜弥**
**185745C 上間翔**
**185752F 新垣裕二**
**185763B 草薙幸菜**

提出日：2021 年 2 月 9 日

# 目次

# 1　はじめに

データマイニングとはデータの中に埋め込まれている有用な知識を発掘することである。別の言い方では、データマイニングは、より良い意思決定をするために履歴データをうまく使って一般的な規則性を発見しよとする研究分野である。今回私たちのグループ 3 では、機械学習の基本的な考え方を実装、体験を通して学んだ。そしてその応用として、既存に存在する指紋データを使った指紋認証のプログラムの分析から、畳み込みニューラルネットワークの手法についての模索、改善を行い、結果の精度向上を目指し、考察したことについて報告する。

# 2　概要

本グループでは機械学習の基本的な考えを学びそれを応用するために、指紋を畳み込みニューラルネットワークを用いて指紋の持ち主や左右どの指であるかを識別を行なった。畳み込みニューラルネットワークとは、ある特徴を持つ画像をフィルタに畳み込んでいくことによってその画像の特徴を抽出する識別手法である。指紋の識別を対象にした理由としては、ラベルが明確であることと特徴が複雑であると考えたからである。本実験の目的は、CNN のパラメータを変更し識別の精度を向上させ、得られた結果を考察し、CNN について理解を深めることである。具体的な識別の内容としてはある指紋が、「誰のものであるか」「左右十本のうちどの指であるか」の 2 種類の識別を行う。実験結果としては、エポック数、バッチサイズ、活性化関数などを変更したが、どの場合についても識別率が 99.8% から大きく変わることはなかった。エポック数による精度の向上はやや見られたが、活性化関数が識別にどのような効果をもたらしているのかは読み取ることができなかった。実験全体を通し、当初予定していた計画通りに進めることができた。しかし、バージョン管理やユニットテストについては、多くは行えなかったためそれらが今後な課題となる。

## 2.1　Convolutional Neural Network:CNN

Convolutional Neural Network （これより CNN と呼ぶ）は畳み込みニューラルネットワークという意味であり、機械学習で画像の深層学習といえば CNN であるというほどよく使われている識別手法である。これは、ニューラルネットワークに畳み込みという操作を導入したものである。CNN について、簡単な手順を記述する。まず手順 1 として、**画像から特徴を抽出**する。フィルタを使い、入力層データの中で位置を変えながらスキャンした部分のデータと、フィルター自身の持つデータとの差異を畳み込みの結果として畳み込みそうに書き込んだものを特徴量といい、入力層の全データをスキャンしてできた畳み込み結果の値の集まりを特徴マップという。複数のフィルタを用意することで、入力層のデータ特徴を捉えやすくしている。**図 1** は用意した複数フィルタのうち、一つが完全に入力層のデータの一部と同じであることを示す。

手順 2 として、**画像を畳み込み**する。入力層のデータをフィルターのデータとピクセル毎に比較

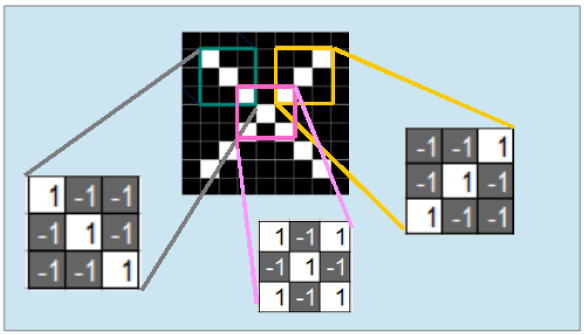

図 1　CNN 解説手順 1 (参考サイト [1] より引用)

することで、畳み込み層にその類似度（特徴量）を書き込む。**図 2** はフィルタを利用して特徴量を抽出し、特徴マップを作成した例である。

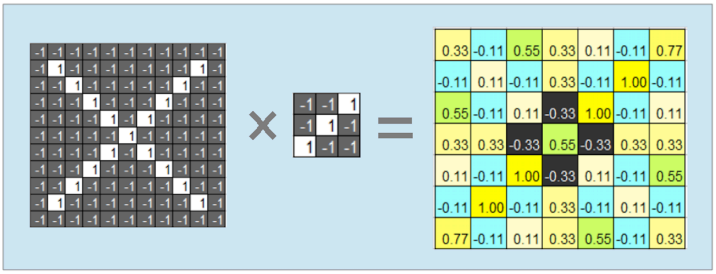

図 2　CNN 解説手順 2 (参考サイト [1] より引用)

手順 3 として、**画像をプーリング**する。畳み込みの層の情報はプーリング層で集約する。出力に関しては、プーリング層のユニット全てと全結合し、計算結果を利用して、フィルタ、重み、バイアスを更新していく。(**図 3**)

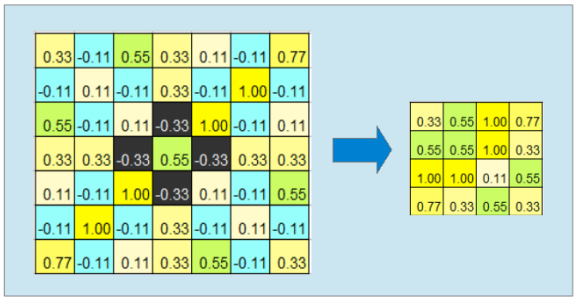

図 3　CNN 解説手順 3 (参考サイト [1] より引用)

## 2.2 テーマ指紋認証とは

本グループでは、授業の中でデータマイニングについて学び、それらの応用実験として、画像認識について分析しようと考えた。そして、データマイニングを行う識別手として CNN に目をつけ、データの分類ラベルがはっきりしており、複雑である指紋認証の分析、精度改善を行うこととなった。CNN は畳み込み処理を利用したニューラルネットワークであり、どのくらい畳み込み処理を行うのか、どのくらいニューラルネットワークを深くするのかは定義されていない。

# 3 実験方法

## 3.1 実験目的

本実験では大きく 3 つを目的にあげる。1 つ目に、指紋データセットを用いて指紋認証を CNN というアルゴリズムを使って行っているが、CNN というデータマイニングの画像認識の手法の一つを理解し、実行することを目的とする。二つ目に、CNN のパラメータ調整をいくつか行っていくことで、指紋認証の識別率の改善、結果向上も目的にある。さらに、三つ目として、各々パラメータ調整によってどのような違いがうまれるのか等を可視化を通しながら考察していくことを目的とする。

## 3.2 データセット構築

Kaggle より、指紋のデータセット [1] を利用している。

## 3.3 モデル選定

アルゴリズムは序章で述べたとおり、CNN を利用しており、SubjectID&Finger_CNNRecognizer [3] のコードを参考にしている。

本アルゴリズムの運用において、指紋認証の正答率が十分に高い点、可読性に優れており改善手法を模索しやすい点に特に優れているため使用することにした。CNN のネットワーク図を以下の図 4 に示す。

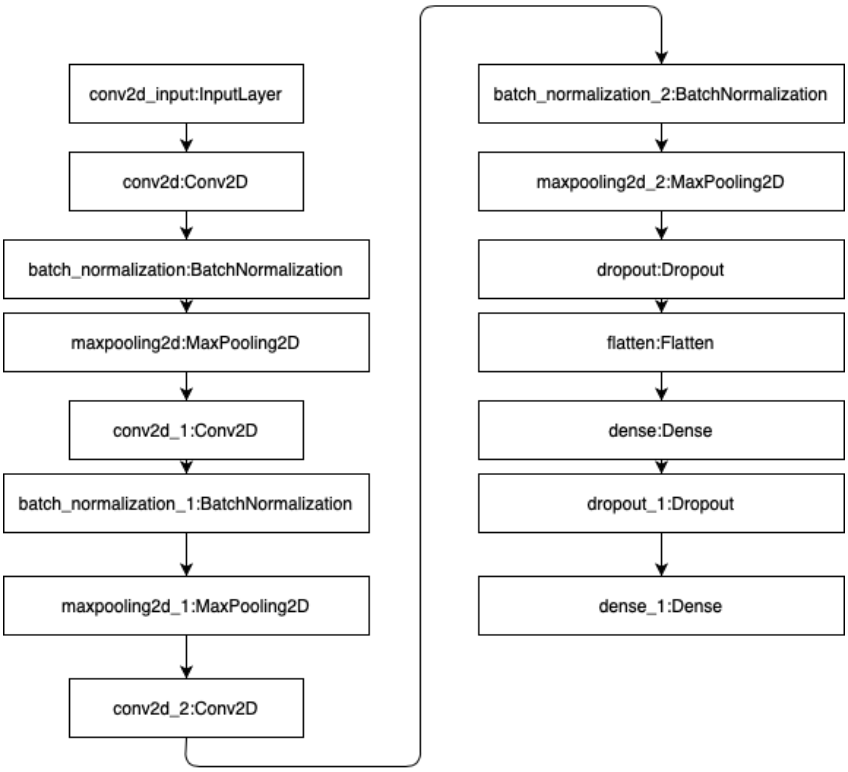


図 4　ネットワーク図

## 3.4　パラメータ調整

CNN のパラメータ調整では、epoch 数・画像データ数・batchsize や活性化関数である LeakyReLU を変更していった。以下にコード変更部分を記載する。

```
#CNN
nets = 2 #人を特定するnetと指を特定するnetの2つ用意
model = [0] * nets
final_Dense_units = [600, 10]
model_name = ['SubjectID_Mod', 'FingerNum_Mod']
for i in range(nets):
    model[i] = Sequential(name=model_name[i])
    #Sequential-モデル層を積み重ねる形式の記述方法。.addメソッドで簡単に層を追加できる。

    model[i].add(layers.Conv2D(32, (5, 5), activation='relu', kernel_regularizer=regularizers.l2(0.001), input_shape = (64, 64, 1)))
    #Conv2D 2次元畳み込み層：5×5の大きさのフィルタを32枚使う
    #activation= "relu":活性化関数ReLUはフィルタ後の入力データが
```

```
    0以下の時は出力0入力が0より大きい場合はそのまま出力
13  #kernel_regularizer:kerasのConvolution層(畳み込み層)に正則化を入れる
14  #正則化:過学習の回避
15  model[i].add(layers.BatchNormalization())
16  #各バッチ毎に、前の層の出力(このレイヤーへの入力)を正規化する。平均を0,標準偏差値を1に近づける変換を適用。
17  model[i].add(layers.MaxPool2D((2, 2)))
18  model[i].add(layers.Conv2D(64,(5, 5), activation='relu', kernel_regularizer=regularizers.l2(0.001)))
19  model[i].add(layers.BatchNormalization())
20
21  model[i].add(layers.MaxPool2D((2, 2)))
22  #MaxPool2D:最大プーリング層:2×2の領域で最大の数値を出力する。
23  model[i].add(layers.Conv2D(128,(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.001)))
24  model[i].add(layers.BatchNormalization())
25  model[i].add(layers.MaxPool2D((2, 2)))
26  model[i].add(layers.Dropout(0.3))
27  #Dropout:ドロップアウトー過学習防止予防、全結合の層との繋がりを30%無効化している。
28  model[i].add(layers.Flatten())
29  #Flatten:平坦化(次元削除)ー1次元ベクトルに変換する。
30  model[i].add(layers.Dense(256, activation='relu'))
31  #Dense:全結合層、出力256,活性化関数ReLU
32  model[i].add(layers.Dropout(0.4))
33  #全結合層とのつながりを40%無効化
34  model[i].add(layers.Dense(final_Dense_units[i], activation='softmax'))
35  #全結合層、出力はここではfinal_Dense_unitsで与えている600と10
36  #Softmax関数で挿話が1となるように、各出力の予測確率を計算。
37
38  # モデルのコンパイル
39  model[i].compile(optimizer=optimizers.Adam(learning_rate=0.0001), loss="categorical_crossentropy", metrics=["accuracy"])
40  #loss="categorical_crossentropy"は損失化関数ー交差エントロピー誤差を指定している。識別問題などで使用される。
41  #optimizer=Adam()は最適化関数ーAdamを指定している。学習の最適化手法の一つ。
42  model[i].summary() #モデルの要約を出力
```

Listing 1　パラメーター調整 code1

```
1  #学習の実行の準備
```

```
history = [0] * nets
CallBack = [0] * nets
ReduceLR_minlr = [1e-9, 1e-7]
epochs = 20
#学習する回数
batch_size = 64
for i in range(nets):
    CallBack[i] = [       #Callback:学習率の調整コード
        callbacks.EarlyStopping(monitor='val_accuracy', patience
          =10, mode='max', verbose=1),
        #監視する値の変化が停止した時に訓練を終了する。
        #monitor:監視する値、ここでは'val_accurary'
        #patience:ここで指定したエポック数の間(監視する値に)改善
          がないと訓練が停止
        #mode:{auto,min,max}の内、一つ選択。
        #minは監視する値の減少が停止した際に訓練終了。maxは監視す
          る値の増加が停止した際に訓練を終了。autoは自動的に監視
          されいている値から推測。
        callbacks.ReduceLROnPlateau(factor=0.1, patience=1,
          min_lr=ReduceLR_minlr[i], verbose=1),
        #評価値の改善が止まった時に学習率を減らす。
        #factor:学習率を減らす割合,
        #patience:何エポック改善が見られなかったら学習率の削減を
          行うか,
        #min_lr:学習率の下限
        callbacks.TensorBoard(log_dir="./log_dir/"+model_name[i])
          ]
        #tensorboardによる基本的な可視化:Tensorboardのログを出
          力。異なる層への活性化ヒストグラムと同様に、訓練とテス
          トの評価値を動的にグラフ化し、可視化できる。

#学習開始!
history[0] = model[0].fit(X_SubjectID_train, y_SubjectID_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    validation_data = (X_SubjectID_val,
                      y_SubjectID_val),
                    verbose = 1, callbacks= CallBack[0])
#history = model.fit():訓練データで学習を開始
#X_SubjectID_train,y_SubjectID_train:訓練データ
#batch_sizeはバッチサイズ
#epochs:学習する回数
#validation_data:評価用データの指定ここでは(X_SubjectID_val,
  y_SubjectID_val)
#verbose:ログ出力の指定。0だとログ無し。
```

```
36 #callback:学習率を調整
```

Listing 2　パラメーター調整 code2

# 4　実験結果

実験結果を以下の表 1 から表 3 に示す。今回は、指紋の持ち主の識別と、左右のどの指かの識別の 2 種類の識別を行った。そのため、指紋の持ち主の識別率を subjectID accracy、左右のどの指かの識別率を fingerNum accracy で表している。

表 1　エポック数を 1 から 30 までの値に変更

| epoch | subjectID accracy | fingerNum accracy |
|---|---|---|
| 1 | 01.533 % | 63.016 % |
| 3 | 61.283 % | 89.483 % |
| 5 | 96.350 % | 98.150 % |
| 10 | 99.716 % | 99.716 % |
| 20 | 99.733 % | 99.883 % |
| 30 | 99.733 % | 99.900 % |

表 2　epoch を 20 に固定し、batch_size を変更

| batch_size | subjectID accuracy | fingerNum accuracy |
|---|---|---|
| 32 | 99.733 % | 99.900 % |
| 64 | 99.733 % | 99.883 % |
| 128 | 99.716 % | 99.883 % |

表 3　活性化関数を LeakyReLU、sigmoid、tanh に変更

| 活性化関数 | subjectID accracy | fingerNum accracy |
|---|---|---|
| LeakyReLU (alpha=−0.5) | 99.433 % | 99.883 % |
| LeakyReLU (alpha=0.3) | 99.716 % | 99.733 % |
| LeakyReLU (alpha=0.5) | 99.699 % | 99.633 % |
| sigmoid | 99.733 % | 99.866 % |
| tanh | 99.733 % | 99.883 % |

# 5　考察

初めに、元のサンプルコードの条件はエポック数 20 のバッチサイズ 64、活性化関数は中間層で ReLU 関数を用いて、出力層に対して softmax 関数を用いている。結論から言うと、どの条件下においても大きな変化は見られなかった。精度を向上させることができた条件は、エポック数を 20 から 30 に増やすこと、バッチサイズを 64 から 32 に変更することの 2 つであった。また、両者とも精度が上がったのは fingerNum accracy のみであった。この原因と、その他の条件において精度が上昇しなかった原因について考える。

初めに、fingerNum accracy のみ精度が向上したことについて考える。エポック数とは学習する世代のことであり、これを多くしていくことで細かい識別が可能になる。今回の結果から考えると、fingerNum accracy の方が識別により細かい特徴を必要とすると考える。

次に、活性化関数を変更した場合について考える。今回は画像の識別であるため、活性化関数も多くの値を表現できるものが良いと考えた。0 と 1 だけで識別するのと、1 から 100 までを用いて識別するのは後者の方がより細かいものを表現できるはずである。実際、元のサンプルコードは負の値は 0 にし、正の値はそのまま使用する ReLU 関数を用いていた。そこで、負の値も一定の割合で使用する LeakyReLU 関数を用いたが結果は精度がやや落ちる程度だった。また、sigmoid 関数と tanh 関数も用いた。これらは入力された値を、0 から 1、-1 から 1 の実数にする関数である。これまでの考えだと精度は下がると予測したが、LeakyReLU 関数を用いた結果とあまり大差はなかった。これらのことを見ると活性化関数に精度との相関が見られない。活性化関数の他に精度をあげている要因があると考えられる。今回の実験では最適化関数について変更を行っていため、最適化関数に原因があるのではないかと考える。

以上より、エポック数による精度の向上はやや見られたが、活性化関数が識別にどのような効果をもたらしているのかは読み取ることができなかった。

## 6　意図していた実験計画との違い

当初予定していた実験計画は以下の通りとなる。下記の予定は 11 月 17 日に作成されたものである。後々詳しく決められた締め切り予定には (追加内容:) として記述する。

| 講義時間 (日次) | 実験計画 | 実際に行った内容 |
|---|---|---|
| 6 週目 (11/17) | テーマ決め<br>データセット探し<br>など | テーマ決定 (画像認証/fingerprint)<br>データセット探し<br>画像認証に対しての学習 |
| 7 週目 (11/24) | 特徴ベクトルの抽出や<br>コード探して理解する | 指紋の特徴量の分析<br>CNN の画像認証コードの検索 |
| 8 週目 (12/1) | 画像認識をコードに落とし込む | 参考する CNN コードの決定<br>コードの内容の分析・実行 |
| 9 週目 (12/8) | 実験開始<br>出来上がったコードを動かす | コードの内容の分析<br>実行 2nd |
| 10 週目 (12/15) | コードや特徴ベクトルの調整 | コードの内容の分析 3rd<br>amane による実行開始<br>実験目的の草案作成 |
| 11 週目 (12/22) | 未決定 | コードの内容の分析 4th・実行 2nd<br>amane で画像データ作成は難しいと判断<br>来年に向けての引継ぎ |
| 12 週目 (1/5) | 未決定 | コードの内容の分析 5th<br>epoch 数を変えた大規模実験<br>Python/Keras 環境の統一 |
| 13 週目 (1/12) | 改善実験 | コードの内容の分析 6th<br>batch_size や LeakyReLU を変更した実験 |
| 14 週目 (1/19) | 未決定 | レポートの作成<br>amane による実験 |
| 15 週目 (1/26) | レポート・プレゼン資料作成<br>(追加内容:レポート初期版の提出日) | レポート・プレゼン資料作成<br>レポート初期版の作成 |
| (1/26) | | レポート・プレゼン資料作成 |
| 期末日 (2/2) | 最終発表 | 最終発表 |
| (2/16) | (追加内容:Github 公開) | 公開済み |

実験計画を作成した段階では後半の進捗の進み具合の判断が困難だったため、前半の予定と締め切りに合わせた大まかな目標を設定し、他は未定とした。未定の欄を除いて概ね実験計画と実際の

進捗に大きな乖離はなく、順調に進んだと判断できる。

## 7　まとめ

本実験のデータマイニング班の達成目標を大まかにまとめると、バージョン管理やユニットテストを用いながら機械学習やデータセットの可視化や構築をできるようになるというものである。

今回、本グループはサンプルコードを改良し結果を向上させる形で実験に取り組んだ。そのため、データセットについては既に構築されており、コードについても jupyter で編集する形であったため、バージョン管理やユニットテストについては多くは行わなかった。しかし、開発に時間を割かなかった分、本実験で使用した CNN の畳み込みやプーリング、活性化関数といった構造を理解することができた。そこから CNN による識別率の向上に寄与する要因を考察し手を加え、さらに得られた結果から自分たちが修正したものがどのような影響を及ぼしたのかについても考察することができた。また作業においても役割を分担し、計画通りに進めることができた。

## 8　振り返り

今後の課題としてはまとめで述べたように、バージョン管理、ユニットテスト、自力でのコーディングが行えなかったので、自分たちでコーディングできるようになり、そのコードに対しバージョン管理、ユニットテストを行えるようになることが考えられる。大まかにスケジュールを組むならば、用いるライブラリの打ち合わせ、用いる学習器の決定、その後コーディングに入るとスムーズにいくと考えられる。

## 参考文献


[1]「畳み込みニューラルネットワークとは？手順も丁寧に解説」,
<https://udemy.benesse.co.jp/data-science/ai/convolution-neural-network.html>2021/02/09.

[2] ruizgara「Sokoto Coventry Fingerprint Dataset (SOCOFing)」
<https://www.kaggle.com/ruizgara/socofing>2021/02/09.

[3] Brian Zhang「SubjectID&Finger_CNNRecognizer」,
<https://www.kaggle.com/brianzz/subjectid-finger-cnnrecognizer>2021/02/09.


## 謝辞


データマイニングの担当教員である當間愛晃教授には、本実験の指導を通じて多くの知識や示唆をいただき心より感謝いたします。また、サンプルコードを書いてくださった Mr.Brian Zhang に深く感謝いたします。