

情報工学実験 I・II 報告書

実験題目	パターン認識と機械学習	
実施日	2022 年 11 月 29 日	2022 年 12 月 6 日
	2022 年 12 月 13 日	2022 年 12 月 20 日
	2023 年 1 月 17 日	2023 年 1 月 24 日
報告書作成日	2023 年 1 月 31 日	
報告者	421858 若山陽向	

三重大学 工学部 総合工学科 情報工学コース

目 次

1	目的	3
2	このレポートの構成	3
3	実験方法	3
4	実験結果、動作試験、およびプログラムの詳細解説	3
4.1	畳み込みフィルタ	3
4.2	最急降下法	8
4.2.1	基礎	8
4.2.2	発展	11
4.3	手書き文字認識	13
5	自己設定課題	16
5.1	導入	16
5.2	修正点	17
5.3	まとめ	24

1 目的

今回の目的は、Python を使いパターン認識と機械学習の基礎がどのように行われているのかわかり人工知能関連の研究に対するイメージを付けることと、個別課題を通して問題解決する力を身に付ける事である。そのために、Jetson Nano を使用してプログラム作成を行う。

2 このレポートの構成

プログラムの結果や解説は第 4 章に載せてある。その際、まずはコードを全て載せ、その後に随時解説をしている。

このようにコードを説明するときは一度に載せるのではなく、分割して載せ、その度に説明することで、可読性を向上させた。

3 実験方法

Jetson Nano に linux ディストリビューションの Debian 系である Ubuntu をインストールして行った。GUI なので使いやすかった。コーディングには主に VSCode を使った。理由としては、コーディングが圧倒的に VSCode がやりやすく、また、Jupyter を使うことも考えたが、全体のコードを分割して入力していく方式だと、後々コードを提出することを考えた時に煩わしいので、一度に全部を入力できる VSCode を使用した。

なお、Python のプログラムを実行するときは、動作などは main 関数や他の関数を用意して書いたが、プログラムの最後には必ず

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

を付けた。

これをつけることで、python ファイル名.py ではなく、単にそのプログラムが import されただけの時に処理が実行されることを防ぐことができる。

4 実験結果、動作試験、およびプログラムの詳細解説

4.1 畳み込みフィルタ

```
1 # coding: UTF-8  
2 import cv2  
3 import numpy as np  
4 import time  
5  
6 def main():  
7  
8     #ここを変えることで他の画像も使える  
9     before_picture = 'ducks.jpg'  
10  
11     #SobelYを定義  
12     SobelY = np.array([[ -1, -2, -1], [ 0, 0, 0], [ 1, 2, 1]])  
13  
14     #SobelXを定義  
15     SobelX = np.array([[ -1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  
16  
17     #Gaussian3を定義  
18     Gaussian3 = np.array([[ 1/16, 2/16, 1/16], [ 2/16, 4/16, 2/16], [ 1/16, 2/16, 1/16]])  
19
```

```

20 #Gaussian5を定義
21 Gaussian5 = np.array([[1/256, 4/256, 6/256, 4/256, 1/256], [4/256, 16/256, 24/256,
22                      16/256, 4/256], [6/256, 24/256, 36/256, 24/256, 6/256],
23                      [4/256, 16/256, 24/256, 16/256, 4/256], [1/256,
24                      4/256, 6/256, 4/256, 1/256]])
25
26 img = cv2.imread(before_picture, cv2.IMREAD_GRAYSCALE) #モノクロで読み取る
27
28 print('処理中です')
29 time_begin = time.perf_counter()
30
31 cv2.imwrite('convoluted.jpg', convolution(img, SobelY))
32
33 time_end = time.perf_counter()
34 print('終了しました\nconvoluted.jpgに結果を保存しました')
35 calc_sec = time_end - time_begin
36 print('かかった時間 : ', calc_sec, '秒')
37
38 def convolution(img, kernel):
39     img_xsize = img.shape[0]
40     img_ysize = img.shape[1]
41     #グレイスケールなのでz成分はない
42
43     kernel_xsize = kernel.shape[0]
44     kernel_ysize = kernel.shape[1]
45
46     #最終的にreturnする画像はappended_img
47     appended_img = np.zeros((img_xsize + 4, img_ysize + 4))
48
49     #imgからkernelのサイズ分抜き出した画像の要素の行列
50     picked_img = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
51
52     temp = np.zeros((kernel_xsize, kernel_ysize))
53
54     for i in range(img_xsize):
55         for j in range(img_ysize):
56             appended_img[i+2, j+2] = img[i, j]
57             #appended_imgの内部はimgと同じだが、周囲が0でパディングされているところだけ違う
58
59     for i in range(img_xsize):
60         for j in range(img_ysize):
61             picked_img = appended_img[i: i+kernel_xsize, j: j+kernel_ysize] #
62             #imgからkernelサイズ分抜き出す
63             temp = np.multiply(picked_img, kernel) #
64             #picked_imgとkernelのアダマール積を求める
65             appended_img[i, j] = np.sum(temp) #先程のtempの全成分の和を代入する
66
67     return appended_img
68
69 if __name__ == '__main__':
70     main()

```

まずは、次のようにカーネルを定義した。

```

1 #SobelYを定義
2 SobelY = np.array([[ -1, -2, -1], [0, 0, 0], [1, 2, 1]])
3
4 #SobelXを定義
5 SobelX = np.array([[ -1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
6
7 #Gaussian3を定義
8 Gaussian3 = np.array([[1/16, 2/16, 1/16], [2/16, 4/16, 2/16], [1/16, 2/16, 1/16]])
9
10 #Gaussian5を定義
11 Gaussian5 = np.array([[1/256, 4/256, 6/256, 4/256, 1/256], [4/256, 16/256, 24/256, 16/256,
12                      4/256], [6/256, 24/256, 36/256, 24/256, 6/256],
13                      [4/256, 16/256, 24/256, 16/256, 4/256], [1/256,
14                      4/256, 6/256, 4/256, 1/256]])

```

次に、画像をモノクロで読み取り、

```

1 img = cv2.imread(before_picture, cv2.IMREAD_GRAYSCALE)

```

関数 `convolution()` を呼び出し計算させた。また、関数 `time.perf_counter()` を用いて計算時間を算出した。フィルターをかけた後の画像は `convoluted.jpg` に保存した。

```

1 print('処理中です')
2 time_begin = time.perf_counter()
3
4 cv2.imwrite('convoluted.jpg', convolution(img, SobelY))
5
6 time_end = time.perf_counter()
7 print('終了しました\nconvoluted.jpgに結果を保存しました')
8 calc_sec = time_end - time_begin
9 print('かかった時間 : ', calc_sec, '秒')

```

ここから、関数 `convolution()` の説明をする。まずはカーネルのサイズを `shape` で取得する。
x 方向の大きさは `kernel.shape[0]` に、y 方向の大きさは `kernel.shape[1]` にあるので、それぞれ取得した。

```

1 kernel_xsize = kernel.shape[0]
2 kernel_ysize = kernel.shape[1]

```

最終的に `return` する画像、つまりフィルタをかけた後の画像は `appended_img` であるが、そのサイズは、x 方向 y 方向ともに 4 ずつ伸ばした。今回使うカーネルの最大は Gaussian5 の 5×5 であり、畳み込み演算を行う時に使うサイズは 3×3 、よって 2 足りない。この足りない 2 が x 方向だと左右、y 方向だと上下にあるので、足りない 2 に $\times 2$ をして 4 となるから、それらを追加したサイズとした。

```

1 appended_img = np.zeros((img_xsize + 4, img_ysize + 4))

```

次の処理は、元々の `img` のままだと、フィルタをかける時に端までかけることができないので、元々の画像の周囲を 0 でパディングしている。

```

1 for i in range(img_xsize):
2     for j in range(img_ysize):
3         appended_img[i+2, j+2] = img[i, j]
4         # appended_img の内部は img と同じだが、周囲が 0 でパディングされているところだけ違う。

```

ここからは計算部分である。

```

1 for i in range(img_xsize):
2     for j in range(img_ysize):
3
4         # img から kernel サイズ分抜き出す
5         picked_img = appended_img[i: i+kernel_xsize, j: j+kernel_ysize]
6
7         # picked_img と kernel のアダマール積を求める
8         temp = np.multiply(picked_img, kernel)
9
10        # 先程の temp の全成分の和を代入する
11        appended_img[i, j] = np.sum(temp)
12
13    return appended_img

```

ここで、アダマール積の説明をする。アダマール積とは、行列の同じ部分の要素のみ掛け合わせる演算である。

例えば、 $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \odot \begin{bmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{bmatrix}$ のような計算を考えてみる。 (\odot) はアダマール積の演算を示す。)

答への (1,1) 成分は 1×10 なので 10、(1,2) 成分は 2×11 なので 22 という様に考えるので、答えは $\begin{bmatrix} 1 \times 10 = 10 & 2 \times 11 = 22 & 3 \times 12 = 36 \\ 4 \times 13 = 52 & 5 \times 14 = 70 & 6 \times 15 = 90 \\ 7 \times 16 = 112 & 8 \times 17 = 136 & 9 \times 18 = 162 \end{bmatrix}$ となる。

以上を踏まえると、この部分ではまず先程パディングした `appended_img` をカーネルのサイズで切り出し、それとカーネルとのアダマール積を求めている。あとはアダマール積の演算でできた配列 `temp` の全要素の和を算出すれば、求めたい要素の値になる。分かりやすくするために、参考にしたサイトを書き込んだ画像を貼り付ける。(出典：[1, 畳み込み演算])

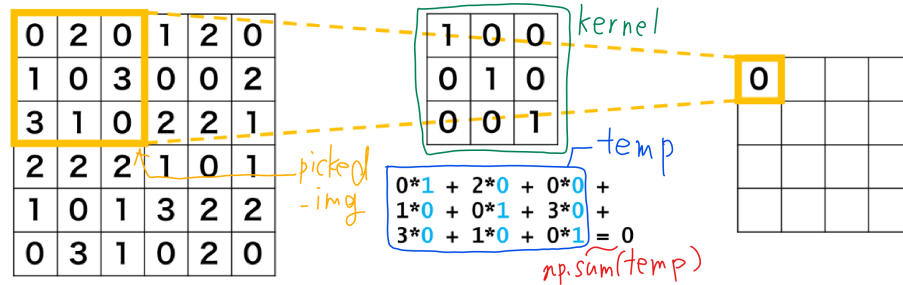


図 1: 参考図

ここから、各カーネルの結果を示す。まずは、元々の画像を貼る。



図 2: フィルタ前の画像

ここからはフィルタをかけている画像である。

1. SobelY



図 3: SobelY の画像

2. SobelX



図 4: SobelX の画像

3. Gaussian3



図 5: Gaussian3 の画像

4. Gaussian5



図 6: Gaussian5 の画像

確かに、スライドと同じようなフィルタを通した後の結果が得られたことが分かる。

4.2 最急降下法

4.2.1 基礎

```

1  #!/usr/bin/python3
2  # coding: UTF-8
3  import cv2
4  import math
5  import time
6
7  def main():
8      check_f1()
9      check_f2()
10
11  def check_f1():
12      eta = pow(10, -3)  #収束速度定数は $10^{-3}$ にした
13      exist_minimum = True  #最小値が存在するか(存在しないのならFalse)
14
15      f1_minimum_list = []  #f1の最小値を格納するリスト
16      minimum_x_list = []  #xの最小値を格納するリスト
17      #複数の初期値からなので、 $-10 \sim 10$ までで試す
18      for begin_init in range(-10, 10):
19          x = begin_init
20          for i in range(10000):
21              x -= eta * d_f1(x)
22              if (f1(x) > pow(2, 31)):  # $2^{31}$ を超えると流石にでかすぎて答えとして不適なので
23                  break  #breakする
24              if (f1(x) < -pow(2, 31)):  # $-(2^{31})$ より小さいと最小値なしと判定
25                  exist_minimum = False
26                  break
27              f1_minimum_list.append(f1(x))  #結果をlistに付け加えておく
28              minimum_x_list.append(x)  #結果をlistに付け加えておく
29
30      if (exist_minimum == True):
31          print('2x^2+8x-16の最小値は ', min(f1_minimum_list))  #listの最小値を出す
32          min_index = f1_minimum_list.index(min(f1_minimum_list))  #minの時の添字を出す
33          print('その際のx = ', minimum_x_list[min_index], '\n')
34      else:
35          print('2x^2+8x-16の最小値は存在しません\n')
36
37
38
39  def check_f2():
40      eta = pow(10, -3)  #収束速度定数は $10^{-3}$ にした
41      exist_minimum = True  #最小値が存在するか(存在しないのならFalse)
42
43      f2_minimum_list = []  #f1の最小値を格納するリスト
44      minimum_x_list = []  #xの最小値を格納するリスト
45      exist_minimum = True
46      #複数の初期値からなので、 $-10 \sim 10$ までで試す

```



```

47     for begin_init in range(-10, 10):
48         x = begin_init
49         for i in range(10000):
50             x -= eta * d_f2(x)
51             if ( f2(x) > pow(2, 31)):  #2^31を超えると流石にでかすぎて答えとして不適なので
52                 breakする
53             if ( f2(x) < -pow(2,31)):  #-(2^31)より小さいと最小値なしと判定
54                 exist_minimum = False
55                 break
56             f2_minimum_list.append(f2(x))  #結果をlistに付け加えておく
57             minimum_x_list.append(x)  #結果をlistに付け加えておく
58
59         if( exist_minimum == True ):
60             print('2x^2+8x-16の最小値は ', min(f2_minimum_list))  #listの最小値を出す
61             min_index = f2_minimum_list.index(min(f2_minimum_list))  #minの時の添字を出す
62             print('その際のx = ', minimum_x_list[min_index], '\n')
63         else:
64             print('2x^2+8x-16の最小値は存在しません')
65
66
67
68     def f1(x):
69         return 2*pow(x, 2) + 8*x - 16
70
71     def d_f1(x):
72         return 4*x + 8
73
74     def f2(x):
75         return pow(x, 3)- 4*pow(x, 2) + x - 2
76
77     def d_f2(x):
78         return 3*pow(x, 2) - 8*x + 1
79
80     if __name__ == '__main__':
81         main()

```

次に、再急降下法について考える。まず、収束速度定数は 10^{-3} にした。

```

1  #収束速度定数は10^-3にした
2  eta = pow(10, -3)

```

続いて bool 型の変数 `exist_minimum` を定義し、最小値が存在するかを判定させた。どのように判定するかは後述する。

```

1  #最小値が存在するか(存在するならTrue)
2  exist_minimum = True

```

初期値に対する依存性を調べるため、複数の初期値から開始しているのが次の部分である。

```

1  #複数の初期値からなので、-10~10までで試す
2  for begin_init in range(-10, 10):
3      x = begin_init

```

再急降下法は 1 次収束であり、局所的最適解にたどり着くまでに必要な反復回数が多くなりやすいことが知られている。(出典：[2, 再急降下法])

そのため、ニュートン法と比較すると収束までが遅い。よって、反復回数は 10000 回と多めに設定した。

```

1  for begin_init in range(-10, 10):
2      x = begin_init
3      for i in range(10000):

```

続いて、定義式

$$x \leftarrow x - \eta \frac{df(x)}{dx} \quad (1)$$

とあるように、4 行目の `x -= eta * d_f1(x)` でその計算を行っている。

5 行目以降は極端な値になった時の処理である。特に、下の if 文は、 $-(2^{31})$ より小さいと最小値なしと判定し、最小値の有無を示す変数 `exist_minimum` を `False` にしている。これにより、最小値が存在しない事を記録することができる。

```

1 for begin_init in range(-10, 10):
2     x = begin_init
3     for i in range(10000):
4         x -= eta * d_f1(x)
5         if( f1(x) > pow(2, 31)): #2^31を超えると流石にでかすぎて答えとして不適なので
6             breakする
7             break
8         if( f1(x) < -pow(2,31)): #- (2^31)より小さいと最小値なしと判定
9             exist_minimum = False
10            break

```

また、その時の $f1(x)$, x の値は `append` を使用して記録しておく。

```

1 for begin_init in range(-10, 10):
2     x = begin_init
3     for i in range(10000):
4         x -= eta * d_f1(x)
5         if( f1(x) > pow(2, 31)): #2^31を超えると流石にでかすぎて答えとして不適なので
6             breakする
7             break
8         if( f1(x) < -pow(2,31)): #- (2^31)より小さいと最小値なしと判定
9             exist_minimum = False
10            break
11    f1_minimum_list.append(f1(x)) #結果をlistに付け加えておく
12    minimum_x_list.append(x) #結果をlistに付け加えておく

```

最後に、変数 `exist_minimum` の値によって最小値を表示するかどうかを判定している。その際、`index` を用いて添え字を取得した。

```

1 if( exist_minimum == True ):
2     print('2x^2+8x-16の最小値は ', min(f1_minimum_list)) #listの最小値を出す
3     min_index = f1_minimum_list.index(min(f1_minimum_list)) #minの時の添字を出す
4     print('その際のx = ', minimum_x_list[min_index], '\n')
5 else:
6     print('2x^2+8x-16の最小値は存在しません\n')

```

$f2$ については、 $f1$ と考え方は全く同じなので詳細な解説は省略する。変更点は、 $f1$ を $f2$ にしたところと、 $f1$ における

```

1 if( exist_minimum == True ):
2     print('2x^2+8x-16の最小値は ', min(f1_minimum_list)) #listの最小値を出す
3     min_index = f1_minimum_list.index(min(f1_minimum_list)) #minの時の添字を出す
4     print('その際のx = ', minimum_x_list[min_index], '\n')
5 else:
6     print('2x^2+8x-16の最小値は存在しません\n')

```

を

```

1 if( exist_minimum == True ):
2     print('x^3-4x^2+x-2の最小値は ', min(f2_minimum_list)) #listの最小値を出す
3     min_index = f2_minimum_list.index(min(f2_minimum_list)) #minの時の添字を出す
4     print('その際のx = ', minimum_x_list[min_index], '\n')
5 else:
6     print('x^3-4x^2+x-2の最小値は存在しません\n')

```

に変更したところである。

また、 $f1()$, $d_f1()$, $f2()$, $d_f2()$ は次のように定義している。

```

1 def f1(x):
2     return 2*pow(x, 2) + 8*x - 16
3
4 def d_f1(x):
5     return 4*x + 8
6
7 def f2(x):
8     return pow(x, 3) - 4*pow(x, 2) + x - 2
9
10 def d_f2(x):
11     return 3*pow(x, 2) - 8*x + 1

```

要するに、 $d_f1(x)$ には $f1(x)$ を x について微分した値が入っていて、 $d_f2(x)$ には $f2(x)$ を x について微分した値が入っている。

このプログラムの実行結果を以下に示す。

$2x^2+8x-16$ の最小値は -24.0

その際の $x = -2.0000000000000055$

$2x^2+8x-16$ の最小値は存在しません

4.2.2 発展

```
1  #!/usr/bin/python3
2  # coding: UTF-8
3  import cv2
4  import math
5  import time
6
7
8  def main():
9      eta = pow(10, -3) #収束速度定数は $10^{-3}$ にした
10     exist_minimum = True #最小値が存在するか(存在しないのならFalse)
11
12     minimum_list = [] #最小値を格納するリスト
13     minimum_x_list = [] #xの最小値を格納するリスト
14     minimum_y_list = [] #yの最小値を格納するリスト
15
16     #複数の初期値からなので、 $-10 \sim 10$ までで試す
17     for begin_init in range(-10, 10):
18         x = begin_init
19         y = begin_init
20         for i in range(10000):
21             x -= eta * d_f(x, y)
22             y -= eta * d_f(x, y)
23             if (x > pow(2, 31) or y > pow(2, 31)): # $2^{31}$ を超えると流石に大きすぎて答えとして不適なのでbreakする
24                 break
25             if (f(x, y) < -pow(2, 31)): # $-(2^{31})$ より小さいと最小値なしと判定
26                 exist_minimum = False
27                 break
28             minimum_list.append(f(x, y)) #結果をlistに付け加えておく
29             minimum_x_list.append(x) #結果をlistに付け加えておく
30             minimum_y_list.append(y) #結果をlistに付け加えておく
31
32     if (exist_minimum == True):
33         print('(1-x)^2 + 100(y-x^2)^2の最小値は', min(minimum_list)) #listの最小値を出す
34         min_index = minimum_list.index(min(minimum_list)) #minの時の添字を出す
35         print('その際のx = ', minimum_x_list[min_index])
36         print('その際のy = ', minimum_y_list[min_index])
37     else:
38         print('(1-x)^2 + 100(y-x^2)^2の最小値は存在しません\n')
39
40
41 def f(x, y):
42     return pow((1-x), 2) + 100*pow(y - pow(x, 2), 2) #つまり、 $(1-x)^2 + 100(y - x^2)^2$ 
43
44 def d_f(x, y):
45     return 400*pow(x, 3) - 200*pow(x, 2) + 2*x - 400*x*y - 2 + 200*y
46
47 if __name__ == '__main__':
48     main()
```

基本的な考え方は 4.2.1 と同じだが、2 変数になったので、若干コード量が増えている。例えば、基礎課題 2 の $f1$ の方は

```
1  for begin_init in range(-10, 10):
2      x = begin_init
3      for i in range(10000):
4          x -= eta * d_f1(x)
```

```

5         if( f1(x) > pow(2, 31)): #2^31を超えると流石にでかすぎて答えとして不適なので
6             break
7         if( f1(x) < -pow(2,31)): #- (2^31)より小さいと最小値なしと判定
8             exist_minimum = False
9             break
10        f1_minimum_list.append(f1(x)) #結果をlistに付け加えておく
11        minimum_x_list.append(x) #結果をlistに付け加えておく

```

となっていたが、この部分は

```

1 #複数の初期値からなので、-10~10までで試す
2 for begin_init in range(-10, 10):
3     x = begin_init
4     y = begin_init
5     for i in range(10000):
6         x -= eta * d_f(x, y)
7         y -= eta * d_f(x, y)
8         if( x > pow(2, 31) or y > pow(2, 31)): #2^31を超えると流石に大きすぎて答えとして不適
          適なのでbreakする
9             break
10        if( f(x, y) < -pow(2,31)): #- (2^31)より小さいと最小値なしと判定
11            exist_minimum = False
12            break
13        minimum_list.append(f(x, y)) #結果をlistに付け加えておく
14        minimum_x_list.append(x) #結果をlistに付け加えておく
15        minimum_y_list.append(y) #結果をlistに付け加えておく

```

とあるように、変数 y に関するコードを増やした。また、基礎課題 2 の f1 で最小値を出す部分は

```

1 if( exist_minimum == True ):
2     print('2x^2+8x-16の最小値は ', min(f1_minimum_list)) #listの最小値を出す
3     min_index = f1_minimum_list.index(min(f1_minimum_list)) #minの時の添字を出す
4     print('その際のx = ', minimum_x_list[min_index], '\n')
5 else:
6     print('2x^2+8x-16の最小値は存在しません\n')

```

となっていたのに対し、発展課題 2 は

```

1 if( exist_minimum == True ):
2     print('(1-x)^2 + 100(y-x^2)^2の最小値は', min(minimum_list)) #listの最小値を出す
3     min_index = minimum_list.index(min(minimum_list)) #minの時の添字を出す
4     print('その際のx = ', minimum_x_list[min_index])
5     print('その際のy = ', minimum_y_list[min_index])
6 else:
7     print('(1-x)^2 + 100(y-x^2)^2の最小値は存在しません\n')

```

と、これも変数 y に関するコードを増やしている。

次に、関数 f(), d_f() についてだが、まず f() については

```

1 def f(x, y):
2     return pow((1-x), 2) + 100*pow(y - pow(x, 2), 2) #つまり、(1-x)^2 + 100(y - x^2)^2

```

とあるように、問題文の式

$$f(x, y) = (a - x)^2 + b(y - x^2)^2 = (1 - x)^2 + 100(y - x^2)^2 \quad (2)$$

に $a = 1$, $b = 100$ を代入している。

また、d_f() については 2 変数関数の微分公式

$$\nabla f(x, y) = \frac{df(x, y)}{dx} + \frac{df(x, y)}{dy} \quad (3)$$

を用いて計算できるから、

$$\begin{aligned}\frac{df(x,y)}{dx} &= -2 + 2x + 100(-4xy + 4x^3) \\ \frac{df(x,y)}{dy} &= 100(2y - 2x^2)\end{aligned}\tag{4}$$

を代入して整理すると、

$$\begin{aligned}\nabla f(x,y) &= -2 + 2x + 100(-4xy + 4x^3) + 100(2y - 2x^2) \\ &= 400x^3 - 200x^2 + 2x - 400xy - 2 + 200y\end{aligned}\tag{5}$$

となる。したがって、関数 `d_f()` は

```
1 def d_f(x, y):
2     return 400*pow(x, 3) - 200*pow(x, 2) + 2*x - 400*x*y - 2 + 200*y
```

のように書ける。

このプログラムの実行結果を以下に示す。

$(1-x)^2 + 100(y-x^2)^2$ の最小値は 0.0

その際の $x = 1.0$

その際の $y = 1.0$

4.3 手書き文字認識

```
1  #!/usr/bin/python3
2  # coding: UTF-8
3  import cv2
4  import tensorflow.keras
5  from tensorflow.keras.datasets import cifar10
6  from tensorflow.keras.models import Sequential
7  from tensorflow.keras.layers import Dense, Dropout, Flatten
8  from tensorflow.keras.layers import Conv2D, MaxPooling2D
9  from tensorflow.keras import backend as K
10
11 def main():
12     (x_train, y_train), (x_valid, y_valid) = cifar10.load_data()
13
14     x_train = x_train.astype('float32')
15     x_train /= 255
16     x_valid = x_valid.astype('float32')
17     x_valid /= 255
18
19     #x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
20     x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 3)
21     #x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 1)
22     x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 3)
23
24     y_1hot_train = tensorflow.keras.utils.to_categorical(y_train, 10)
25     y_1hot_valid = tensorflow.keras.utils.to_categorical(y_valid, 10)
26
27     model = Sequential()
28     model.add(Conv2D(32, kernel_size = (3,3),
29                     activation = 'relu',
30                     input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3])
31                     )))
32     model.add(Conv2D(16, (3,3), activation = 'relu'))
33     model.add(MaxPooling2D(pool_size = (2, 2)))
34     model.add(Dropout(0.5))
35     model.add(Flatten())
36     model.add(Dense(256, activation = 'relu'))
37     model.add(Dropout(0.2))
38     model.add(Dense(64, activation = 'relu'))
39     model.add(Dropout(0.3))
```

```

39     model.add(Dense(32, activation = 'relu'))
40     model.add(Dropout(0.3))
41     model.add(Dense(10, activation = 'softmax'))
42
43     model.compile(
44         loss=tensorflow.keras.losses.categorical_crossentropy,
45         optimizer=tensorflow.keras.optimizers.Adam(),
46         metrics=['accuracy'])
47
48     model.fit(x_train, y_1hot_train,
49             batch_size=128,
50             epochs=32,
51             verbose=1,
52             validation_data=(x_valid, y_1hot_valid))
53
54     score = model.evaluate(x_valid, y_1hot_valid, verbose=0)
55     print('Loss :', score[0])
56     print('Accuracy :', score[1])
57
58 if __name__ == '__main__':
59     main()

```

まずは次のように

```

1  import cv2
2  import tensorflow.keras
3  from tensorflow.keras.datasets import mnist
4  from tensorflow.keras.models import Sequential
5  from tensorflow.keras.layers import Dense, Dropout, Flatten
6  from tensorflow.keras.layers import Conv2D, MaxPooling2D
7  from tensorflow.keras import backend as K

```

色々と必要な物を import しておく。続いて、関数 `mnist.load_data()` を用いて値を読み込んだ後に、`x_train` と `x_valid` を 255 で除算しておく。

```

1  (x_train, y_train), (x_valid, y_valid) = mnist.load_data()
2
3  x_train = x_train.astype('float32')
4  x_train /= 255
5  x_valid = x_valid.astype('float32')
6  x_valid /= 255

```

次に、スライド 24 枚目の `images = images.reshape(img_num, img_rows, img_cols, 1)` の部分は、

```

1  x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
2  x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 1)

```

のように、データ数が `shape[0]`、画像の高さが `shape[1]`、画像の幅が `shape[2]` に対応している。

続いて、`y_train` や `y_valid` を one-hot-vector 形式に変換する。変換した後の変数は分かりやすいように、`1hot_` を変数名に追加した。

```

1  y_1hot_train = tensorflow.keras.utils.to_categorical(y_train, 10)
2  y_1hot_valid = tensorflow.keras.utils.to_categorical(y_valid, 10)

```

次の部分は MNIST 識別用モデルを構築しているところである。

```

1  model = Sequential()
2      model.add(Conv2D(4, kernel_size = (3,3),
3                      activation = 'relu',
4                      input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3]
5                      )))
6      model.add(Conv2D(8, (3, 3), activation = 'relu'))
7      model.add(MaxPooling2D(pool_size = (2, 2)))
8      model.add(Dropout(0.25))
9      model.add(Flatten())
10     model.add(Dense(32, activation = 'relu'))
11     model.add(Dropout(0.5))
12     model.add(Dense(10, activation = 'softmax'))

```

9 行目の `Dense()` の第 1 引数は 16 ですと正答率が悪かったので、32 に変更した。また、11 行目の `num_classes` は今回は 10 である。

続いて、学習方法を設定した。

```
1 model.compile(  
2     loss=tensorflow.keras.losses.categorical_crossentropy,  
3     optimizer=tensorflow.keras.optimizers.Adam(),  
4     metrics=['accuracy'])
```

SGD としてスライドでは Adadelta が使用されていたが、正答率が 17% と非常に悪かった。後述する `epochs` の数を 16 に増やしても 32% とそれほど向上しなかった。よって、次の資料を参考に、Adam を選択した。(出典：[3, Adam を選択した理由])

これにより、正答率が 96% と大幅に向上した。

そして、学習を実行させた。

```
1 model.fit(x_train, y_1hot_train,  
2         batch_size=128,  
3         epochs=4,  
4         verbose=1,  
5         validation_data=(x_valid, y_1hot_valid))
```

`y` の方の変数は `x` とは違い、one-hot-vector 形式の方を使うことに注意した。

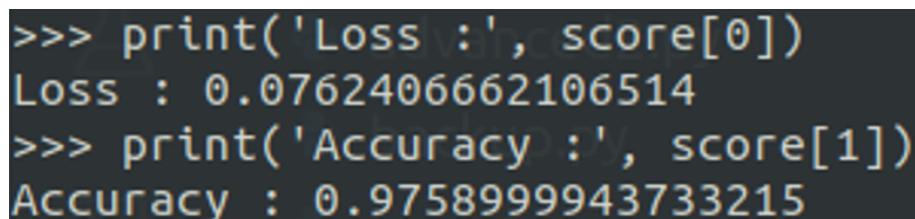
また、`batch_size` は 4 だと小さすぎる。これだと 1 回に計算するデータの数が少なすぎて、各データに最適化されてしまうので 128 にした。`epochs` の値は、何回学習させるかを示しているが、課題 3 ではそれほど大きくしなくても十分な正答率が得られたので、少なめの値にしている。

最後に、精度を評価した。

```
1 score = model.evaluate(x_valid, y_1hot_valid, verbose=0)  
2 print('Loss :', score[0])  
3 print('Accuracy :', score[1])
```

評価用データ `images` は `x_valid` に、正解ラベル `labels` は `y_1hot_valid` にそれぞれ入っている。

このプログラムを実行した結果を以下に示す。



```
>>> print('Loss :', score[0])  
Loss : 0.0762406662106514  
>>> print('Accuracy :', score[1])  
Accuracy : 0.9758999943733215
```

図 7: 手書き文字認識 結果

確かに、`Accuracy` の値が 90% 以上になっている事が分かる。
課題の条件である

- 畳み込み層を含むニューラルネットワークを用いている。
- Jetson Nano 上で GPU を用いて学習が動作する。
- 汎化性能 (`Accuracy` の値が 90% 以上を達成している)。

は全て満たしている事が分かる。

5 自己設定課題

5.1 導入

```
1  #!/usr/bin/python3
2  # coding: UTF-8
3  import cv2
4  import tensorflow.keras
5  from tensorflow.keras.datasets import cifar10
6  from tensorflow.keras.models import Sequential
7  from tensorflow.keras.layers import Dense, Dropout, Flatten
8  from tensorflow.keras.layers import Conv2D, MaxPooling2D
9  from tensorflow.keras import backend as K
10
11 def main():
12     (x_train, y_train), (x_valid, y_valid) = mnist.load_data()
13     (x_train, y_train), (x_valid, y_valid) = cifar10.load_data()
14
15     x_train = x_train.astype('float32')
16     x_train /= 255
17     x_valid = x_valid.astype('float32')
18     x_valid /= 255
19
20     #x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
21     x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 3)
22     #x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 1)
23     x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 3)
24
25     y_1hot_train = tensorflow.keras.utils.to_categorical(y_train, 10)
26     y_1hot_valid = tensorflow.keras.utils.to_categorical(y_valid, 10)
27
28     model = Sequential()
29     model.add(Conv2D(4, kernel_size = (3,3),
30                     activation = 'relu',
31                     input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3]
32                     )))
33     model.add(Conv2D(8, (3, 3), activation = 'relu'))
34     model.add(MaxPooling2D(pool_size = (2, 2)))
35     model.add(Dropout(0.25))
36     model.add(Flatten())
37     model.add(Dense(32, activation = 'relu'))
38     model.add(Dropout(0.5))
39     model.add(Dense(10, activation = 'softmax'))
40
41     model.compile(
42         loss=tensorflow.keras.losses.categorical_crossentropy,
43         optimizer=tensorflow.keras.optimizers.Adam(),
44         metrics=['accuracy'])
45
46     model.fit(x_train, y_1hot_train,
47             batch_size=128,
48             epochs=4,
49             verbose=1,
50             validation_data=(x_valid, y_1hot_valid))
51
52     score = model.evaluate(x_valid, y_1hot_valid, verbose=0)
53     print('Loss :', score[0])
54     print('Accuracy :', score[1])
55
56 if __name__ == "__main__":
57     main()
```

スライドの 29 枚目の「課題例：より難しいデータセットへの適用」にあるように、CIFAR-10 を含むより難易度の高いデータセットに対し CNN を適用する課題に挑戦する。大まかなプログラムは 4.3 で示したものと似ているので、ところどころ省略して説明する。

まずは、

```
1  #(x_train, y_train), (x_valid, y_valid) = mnist.load_data()
2  (x_train, y_train), (x_valid, y_valid) = cifar10.load_data()
```

とあるように、mnist を cifar10 に変更した。これにより、CIFAR-10 を使用できるようになる。上のコメントアウトされている行は 4.3 の手書き文字認識でのプログラムである。

次に、

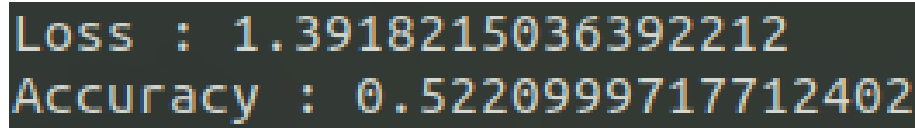
```
1 #x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
2 x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 3)
3 #x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 1)
4 x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 3)
```

とあるが、4.3 ではグレースケールであることからチャンネル数を 1 としていたが、今回は画像が RGB 画像であるため、3 にした。

MNIST 識別用モデルを構築し、学習方法を設定し、学習を実行させ、精度を評価する部分は 4.3 からとりあえず何も変更せずに実行する。

つまり、全体のプログラムとしては仮ではあるが、このようになる。

プログラムの実行結果を以下に示す。



```
Loss : 1.3918215036392212
Accuracy : 0.5220999717712402
```

図 8: 自己設定課題 始めの結果

4.3 では 90%を超えていたが、5.1 では難易度が高い画像を選んだことで、52%とかなり正答率が下がった事がわかる。これを改善していく。

5.2 修正点

```
1 #!/usr/bin/python3
2 # coding: UTF-8
3 import cv2
4 import tensorflow.keras
5 from tensorflow.keras.datasets import cifar10
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Dense, Dropout, Flatten
8 from tensorflow.keras.layers import Conv2D, MaxPooling2D
9 from tensorflow.keras import backend as K
10
11 from tensorflow.keras.callbacks import EarlyStopping
12 from tensorflow.keras.callbacks import ReduceLROnPlateau
13
14 early_stopping = EarlyStopping(
15     monitor='val_loss',
16     min_delta = 0.0,
17     patience = 10,
18 )
19
20 reduce_lr = ReduceLROnPlateau(
21     monitor = 'val_loss',
22     factor = 0.5,
23     patience = 2,
24     min_lr = 0.0001,
25 )
26 (x_train, y_train), (x_valid, y_valid) = cifar10.load_data()
27
28 def main():
29     x_train = x_train.astype('float32')
30     x_train /= 255
31     x_valid = x_valid.astype('float32')
32     x_valid /= 255
33
34     #x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
35     x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 3)
36     #x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 1)
37     x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_valid.shape[2], 3)
38
```

```

39 y_1hot_train = tensorflow.keras.utils.to_categorical(y_train, 10)
40 y_1hot_valid = tensorflow.keras.utils.to_categorical(y_valid, 10)
41
42 model = Sequential()
43 model.add(Conv2D(32, kernel_size = (3,3),
44                 activation = 'relu',
45                 input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3]
46                               )))
47 model.add(Conv2D(16, (3,3), activation = 'relu'))
48 model.add(MaxPooling2D(pool_size = (2, 2)))
49 model.add(Dropout(0.5))
50 model.add(Flatten())
51 model.add(Dense(256, activation = 'relu'))
52 model.add(Dropout(0.2))
53 model.add(Dense(64, activation = 'relu'))
54 model.add(Dropout(0.2))
55 model.add(Dense(32, activation = 'relu'))
56 model.add(Dropout(0.3))
57
58 model.add(Dense(10, activation = 'softmax'))
59
60 model.compile(
61     loss=tensorflow.keras.losses.categorical_crossentropy,
62     optimizer=tensorflow.keras.optimizers.Adam(),
63     metrics=['accuracy'])
64
65 model.fit(x_train, y_1hot_train,
66         batch_size=128,
67         epochs=64,
68         verbose=1,
69         validation_data=(x_valid, y_1hot_valid),
70         callbacks = [early_stopping, reduce_lr],)
71
72 score = model.evaluate(x_valid, y_1hot_valid, verbose=0)
73 print('Loss :', score[0])
74 print('Accuracy :', score[1])
75
76 if __name__ == '__main__':
77     main()

```

まずは

```

1 model.add(Conv2D(32, (3, 3), activation = 'relu'))

```

とあるように、Conv2D() の第 1 引数を 8 から 32 に変更した。すると次のようになった。

```

Loss : 1.300557017326355
Accuracy : 0.5649999976158142

```

図 9: 自己設定課題 2 回目の結果

多少正答率が上がった。

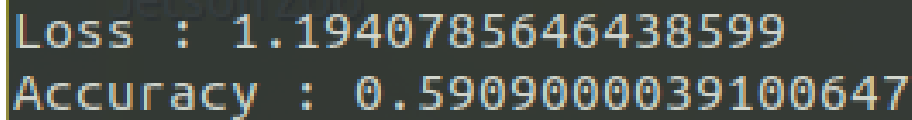
次に、

```

1 model.fit(x_train, y_1hot_train,
2         batch_size=128,
3         epochs=16,
4         verbose=1,
5         validation_data=(x_valid, y_1hot_valid))

```

とあるように、epochs の数を 4 から 16 に増やした。すると、次のようになった。



```
Loss : 1.1940785646438599
Accuracy : 0.5909000039100647
```

図 10: 自己設定課題 3 回目の結果

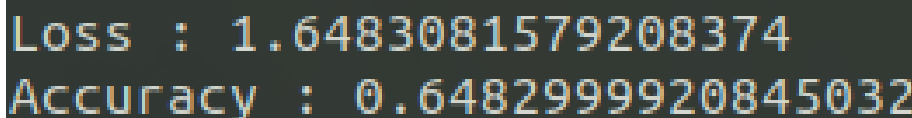
続いて、

```
1 model = Sequential()
2 model.add(Conv2D(32, kernel_size = (3,3),
3                 activation = 'relu',
4                 input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3] )))
5 model.add(Conv2D(16, (3,3), activation = 'relu'))
6 model.add(MaxPooling2D(pool_size = (2, 2)))
7 #model.add(Dropout(0.25))
8 model.add(Flatten())
9 model.add(Dense(128, activation = 'relu'))
10 #model.add(Dropout(0.5))
11 model.add(Dense(64, activation = 'relu'))
12 #model.add(Dropout(0.5))
13 model.add(Dense(32, activation = 'relu'))
14 #model.add(Dropout(0.5))
15 model.add(Dense(10, activation = 'softmax'))
```

とあるように MNIST 識別用モデルに層を追加した。Dropout() は一旦コメントアウトして考える。ここでは、追加した項目が多いので、比較用に変更前のプログラムも貼り付ける。

```
1 model = Sequential()
2     model.add(Conv2D(32, kernel_size = (3,3),
3                     activation = 'relu',
4                     input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3]
5                                     )))
6     model.add(Conv2D(16, (3, 3), activation = 'relu'))
7     model.add(MaxPooling2D(pool_size = (2, 2)))
8     model.add(Dropout(0.25))
9     model.add(Flatten())
10    model.add(Dense(32, activation = 'relu'))
11    model.add(Dropout(0.5))
12    model.add(Dense(10, activation = 'softmax'))
```

実行結果を以下に示す。



```
Loss : 1.6483081579208374
Accuracy : 0.6482999920845032
```

図 11: 自己設定課題 4 回目の結果

ここで、正答率は確かに向上しているが、Loss が増えてしまった。これについて考える。グラフを以下に示す。(出典: [4, ドロップアウト])

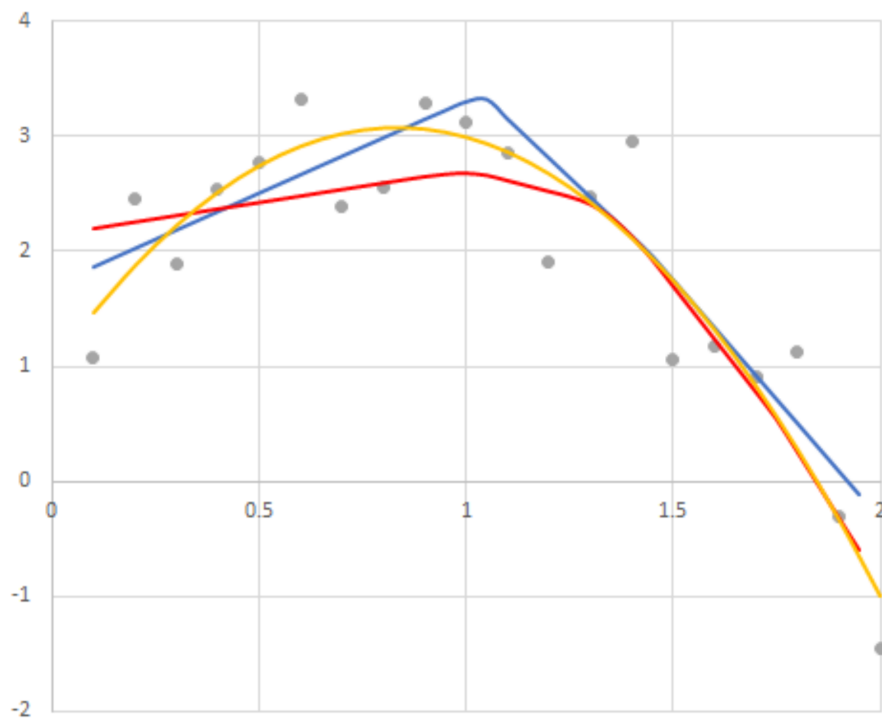


図 12: ドロップアウトの参考グラフ

ここで、各グラフの説明をすると、

- 黄色：2 次関数
- 赤色：ドロップアウトあり
- 青色：ドロップアウトなし

である。このグラフから分かる通りドロップアウトが無いと、カクンと折れ曲がった少し急なグラフとなる。それにより、極端な学習が行われてしまい、Loss(損失値・誤差) は大きくなったと考えられる。

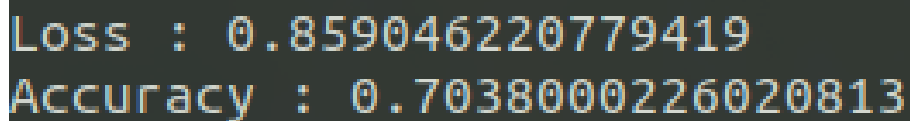
ということで、Dropout() に適切な引数を渡して実行してみる。

```

1 model.add(Conv2D(32, kernel_size = (3,3),
2   activation = 'relu',
3   input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3] )))
4 model.add(Conv2D(16, (3,3), activation = 'relu'))
5 model.add(MaxPooling2D(pool_size = (2, 2)))
6 model.add(Dropout(0.3))
7 model.add(Flatten())
8 model.add(Dense(128, activation = 'relu'))
9 model.add(Dropout(0.5))
10 model.add(Dense(64, activation = 'relu'))
11 model.add(Dropout(0.5))
12 model.add(Dense(32, activation = 'relu'))
13 model.add(Dropout(0.5))
14 model.add(Dense(10, activation = 'softmax'))

```

結果は次のようになった。



```
Loss : 0.859046220779419
Accuracy : 0.7038000226020813
```

図 13: 自己設定課題 5 回目の結果

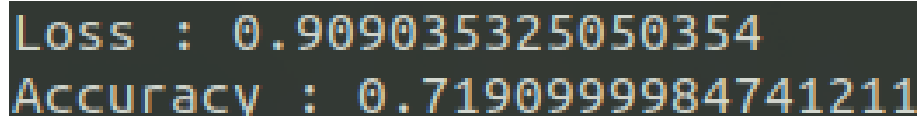
先程の図 11 の結果と比較して、Loss の値がおおよそ 1.65 から 0.859 になった。改善したことが分かる。

また、Accuracy も 70%を越えている。

更に Accuracy を向上させるために、Dropout() の引数を色々に変更して試してみる。しかし、最高で 71%とあまり改善しなかった。その時のプログラムを以下に示す。

```
1 model.add(Conv2D(32, kernel_size = (3,3),
2   activation = 'relu',
3   input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3] )))
4 model.add(Conv2D(16, (3,3), activation = 'relu'))
5 model.add(MaxPooling2D(pool_size = (2, 2)))
6 model.add(Dropout(0.5))
7 model.add(Flatten())
8 model.add(Dense(256, activation = 'relu'))
9 model.add(Dropout(0.2))
10 model.add(Dense(64, activation = 'relu'))
11 model.add(Dropout(0.3))
12 model.add(Dense(32, activation = 'relu'))
13 model.add(Dropout(0.3))
14 model.add(Dense(10, activation = 'softmax'))
```

後はこの条件下で epochs の値を過学習にならない程度に上げて、更なる Accuracy の改善を目指す。epochs = 64 で試す。結果は次のようになった。



```
Loss : 0.909035325050354
Accuracy : 0.7190999984741211
```

図 14: 自己設定課題 6 回目の結果

この数値になった過程について、Excel で accuracy のグラフをプロットしたのでその結果も貼る。



図 15: val_accuracy の結果を Excel でプロット

72%で頭打ちになっている事が分かる。また、loss のグラフも書いたので貼る。



図 16: val_loss の結果を Excel でプロット

図 16 から分かるように、val_loss が全然減っていない。むしろ増えているようにも見える。これは、過学習が原因だと考えられる。つまり、特定のデータにのみ強くなってしまい、その他のデータに対応する力が低下していることである。(出典：[5, 過学習])

ここで、過学習対策として、`tensorflow.keras.callbacks` を使用した。この関数を使用することで、何エポック改善されなかったら学習率を下げるといったことができ、過学習を抑制でき

る。

この関数を使用するため、プログラムの import 部分で

```
1 from tensorflow.keras.callbacks import EarlyStopping
2 from tensorflow.keras.callbacks import ReduceLROnPlateau
```

と書いた、また関数 `early_stopping()`、`reduce_lr()` も追加した。

```
1 early_stopping = EarlyStopping(
2     monitor='val_loss',
3     min_delta=0.0,
4     patience=10,
5 )
6
7 # val_lossの改善が2エポック見られなかったら、学習率を0.5倍する。
8 reduce_lr = ReduceLROnPlateau(
9     monitor='val_loss',
10    factor=0.5,
11    patience=2,
12    min_lr=0.0001
13 )
```

更に、

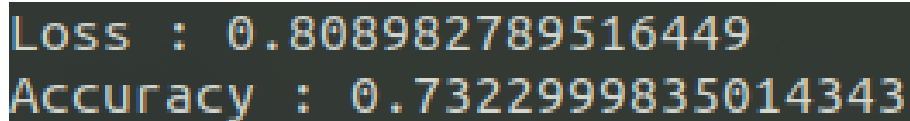
```
1 model.fit(x_train, y_1hot_train,
2     batch_size=128,
3     epochs=32,
4     verbose=1,
5     validation_data=(x_valid, y_1hot_valid))
```

に対して

```
1 model.fit(x_train, y_1hot_train,
2     batch_size=128,
3     epochs=32,
4     verbose=1,
5     validation_data=(x_valid, y_1hot_valid),
6     callbacks=[early_stopping, reduce_lr],)
```

のように変更した。つまり、`callbacks` の部分を追加した。この関数を定義する上で参考にしたサイトはこちらである。(出典：[6, 学習率を減らす方法])

このプログラムを実行すると、以下のような結果となった。



```
Loss : 0.808982789516449
Accuracy : 0.7322999835014343
```

図 17: 自己設定課題 7 回目の結果

Excel で過程をプロットした。

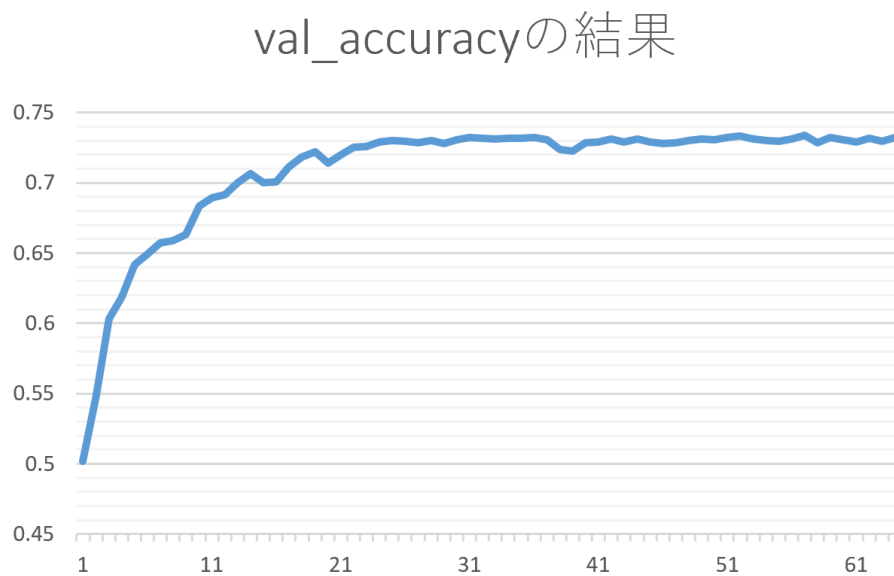


図 18: val_accuracy の結果を Excel でプロット



図 19: val_loss の結果を Excel でプロット

過学習が抑制されていることが分かる。

5.3 まとめ

最後の課題である自己設定課題は、より難しいデータセットへの適用にして、CIFAR-10 について行った。その際課題 3(手書き文字認識) から変更したところは、

1. epochs の数を 4 から 16 に増やした。
2. MNIST 識別用モデルに層を追加した。

3. Dropout に適切な引数を渡した。

4. 過学習を防ぐため、`tensorflow.keras.callbacks` を使用した。

である。最終的な結果は

```
Loss : 0.808982789516449
Accuracy : 0.7322999835014343
```

図 20: 自己設定課題 7 回目の結果

であり、70%を越える事ができた。また、



図 21: val_loss の結果を Excel でプロット

のグラフから分かる通り、過学習を抑制することもできた。

ただ、過学習の抑制はあくまでグラフのブレを抑制するだけであり、80%を越えるなどといった高い正答率は得られなかった。もっと高い正答率を得ようとするのならば、

```
1 x = Conv2D(64,(3,3),padding = "SAME",activation= "relu")(inputs)
2 x = Conv2D(64,(3,3),padding = "SAME",activation= "relu")(x)
3 x = BatchNormalization()(x)
4 x = Conv2D(64,(3,3),padding = "SAME",activation= "relu")(x)
5 x = MaxPooling2D()(x)
6 x = Dropout(0.25)(x)
7
8 x = Conv2D(128,(3,3),padding = "SAME",activation= "relu")(x)
9 x = Conv2D(128,(3,3),padding = "SAME",activation= "relu")(x)
10 x = BatchNormalization()(x)
11 x = Conv2D(128,(3,3),padding = "SAME",activation= "relu")(x)
12 x = MaxPooling2D()(x)
13 x = Dropout(0.25)(x)
14
15 x = Conv2D(256,(3,3),padding = "SAME",activation= "relu")(x)
16 x = Conv2D(256,(3,3),padding = "SAME",activation= "relu")(x)
17 x = BatchNormalization()(x)
18 x = Conv2D(256,(3,3),padding = "SAME",activation= "relu")(x)
19 x = Conv2D(256,(3,3),padding = "SAME",activation= "relu")(x)
20 x = Conv2D(256,(3,3),padding = "SAME",activation= "relu")(x)
21 x = BatchNormalization()(x)
22 x = Conv2D(512,(3,3),padding = "SAME",activation= "relu")(x)
```

```
23 x = Conv2D(512,(3,3),padding = "SAME",activation= "relu")(x)
24 x = GlobalAveragePooling2D()(x)
```

(出典：[7, 正答率の向上])

のように層をもっと増やす方法などが考えられるが、これはメモリなどの関係上厳しそうである。

画像をフィルタに通したり、深層学習の「収束」に当たる再急降下法を学んだり、画像を機械に認識させて何の画像かを当てることを通して、機械学習の基礎がどのように行われているのかが分かった。更に、5.1の自己設定課題で `tensorflow.keras.callbacks` を使って過学習を抑制するなど、問題解決の力も付いた。

参考文献

[1] 畳み込み演算

<https://axa.biopapyrus.jp/deep-learning/cnn/convolution.html>
2023/1/29 閲覧

[2] 再急降下法

https://kamino.hatenablog.com/entry/steepest_gauss
2023/1/29 閲覧

[3] Adam を選択した理由

<https://postd.cc/optimizing-gradient-descent/#whichoptimizertochoose>
2023/1/29 閲覧

[4] ドロップアウト

http://marupeke296.com/IKDADV_DL_No11_dropout.html
2023/1/30 閲覧

[5] 過学習

<https://www.tryeting.jp/column/6037/>
2023/1/30 閲覧

[6] 学習率を減らす方法

<https://analytics-note.xyz/machine-learning/reduce-lr-on-plateau/>
2023/1/30 閲覧

[7] 正答率の向上

<https://qiita.com/yy1003/items/c590d1a26918e4abe512>
2023/1/30 閲覧