def \_\_init\_\_(self ,data ,width ,height):

self.data = data

self.w = width

self.h = height

self.x = data.reshape(-1 ,self.w ,self.h )

self.ew = np.array([])

self.eh = np.array([])

self.ew1 = np.zeros((self.x.shape[0] ,width ,height))

self.eh1 = np.zeros((self.x.shape[0] ,width ,height))

def compress(self ,x):

a = 1

b = 0

c = 1

d = 0

for i in range(x.shape[0 ] ):

for j in range(x.shape[1 ] ):

for k in range(x.shape[2 ] - 1 ):

if x[i ,j ,k ] == x[i ,j ,k + 1 ]:

a = a + 1

if x[i ,j ,k ] != x[i ,j ,k + 1 ]:

self.ew1[i ,j ,b] = a

a = 1

b = b + 1

self.ew1[i ,j ,b] = a

a = 1

b = 0

for l in range(x.shape[0 ] ):

for n in range(x.shape[2 ] ):

for m in range(x.shape[1 ] - 1 ):

if x[l ,m ,n ] == x[l ,m + 1 ,n ]:

c = c + 1

if x[l ,m ,n ] != x[l ,m + 1 ,n ]:

self.eh1[l ,d ,n] = c

c = 1

d = d + 1

self.eh1[l ,d ,n] = c

print("ew1 :" + str(self.ew1) +"　a :" + str(c) )

c = 1

d = 0

self.ew = np.array(self.ew1 )

self.eh = np.array(self.eh1 )

return (self.ew ,self.eh )

[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

[[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[15. 5. 8. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[14. 6. 8. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[13. 9. 6. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[11. 11. 6. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[11. 11. 6. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[10. 12. 6. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 9. 9. 2. 3. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 8. 6. 6. 3. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 7. 7. 6. 3. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 7. 4. 9. 3. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 7. 3. 10. 3. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 4. 10. 3. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 4. 8. 5. 5. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 3. 8. 5. 6. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 3. 7. 4. 8. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 3. 6. 4. 9. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 13. 9. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 11. 11. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 6. 9. 13. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[ 7. 7. 14. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[28. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

元データ

train acc, test acc | 0.09928333333333333, 0.101

train acc, test acc | 0.7766333333333333, 0.7812

train acc, test acc | 0.8762333333333333, 0.8801

train acc, test acc | 0.8988666666666667, 0.9018

train acc, test acc | 0.90795, 0.9126

train acc, test acc | 0.9138833333333334, 0.9172

train acc, test acc | 0.9190833333333334, 0.9226

train acc, test acc | 0.9229333333333334, 0.9244

train acc, test acc | 0.9274, 0.9285

train acc, test acc | 0.93025, 0.9302

train acc, test acc | 0.9330166666666667, 0.9323

train acc, test acc | 0.9351166666666667, 0.9361

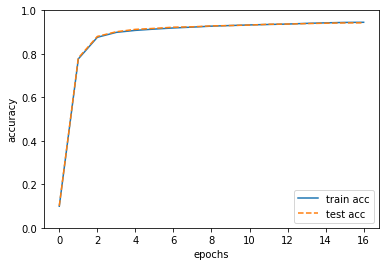
train acc, test acc | 0.9370166666666667, 0.937

train acc, test acc | 0.9401833333333334, 0.9391

train acc, test acc | 0.9426, 0.941

train acc, test acc | 0.9445333333333333, 0.9422

train acc, test acc | 0.94515, 0.9426



横圧縮

train acc, test acc | 0.09863333333333334, 0.0958

train acc, test acc | 0.8303, 0.8403

train acc, test acc | 0.9008, 0.9026

train acc, test acc | 0.9172, 0.9163

train acc, test acc | 0.92455, 0.9225

train acc, test acc | 0.9199, 0.9186

train acc, test acc | 0.93175, 0.9308

train acc, test acc | 0.92555, 0.9239

train acc, test acc | 0.9353, 0.9362

train acc, test acc | 0.93095, 0.9264

train acc, test acc | 0.9428166666666666, 0.9419

train acc, test acc | 0.93825, 0.9354

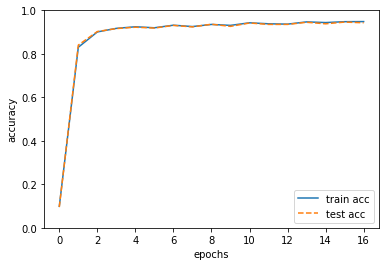
train acc, test acc | 0.9366666666666666, 0.9354

train acc, test acc | 0.9469666666666666, 0.9452

train acc, test acc | 0.9438833333333333, 0.9382

train acc, test acc | 0.9479166666666666, 0.9461

train acc, test acc | 0.9484166666666667, 0.9429



横圧縮＋縦圧縮

train acc, test acc | 0.10441666666666667, 0.1028

train acc, test acc | 0.92265, 0.9253

train acc, test acc | 0.93825, 0.9392

train acc, test acc | 0.9424, 0.9404

train acc, test acc | 0.9374, 0.9337

train acc, test acc | 0.9496666666666667, 0.9486

train acc, test acc | 0.95055, 0.9486

train acc, test acc | 0.9501333333333334, 0.9463

train acc, test acc | 0.9532333333333334, 0.9505

train acc, test acc | 0.9499, 0.9485

train acc, test acc | 0.9436166666666667, 0.9414

train acc, test acc | 0.9554333333333334, 0.9522

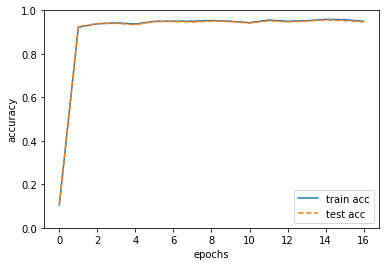
train acc, test acc | 0.9498, 0.947

train acc, test acc | 0.9523833333333334, 0.9506

train acc, test acc | 0.9585666666666667, 0.9557

train acc, test acc | 0.9572166666666667, 0.9526

train acc, test acc | 0.94955, 0.9475



横圧縮＋縦圧縮＋元データ

train acc, test acc | 0.09768333333333333, 0.0983

train acc, test acc | 0.92035, 0.9198

train acc, test acc | 0.9330166666666667, 0.9355

train acc, test acc | 0.94515, 0.9447

train acc, test acc | 0.9422166666666667, 0.9438

train acc, test acc | 0.9405333333333333, 0.9405

train acc, test acc | 0.95025, 0.9481

train acc, test acc | 0.9553666666666667, 0.9527

train acc, test acc | 0.94345, 0.9421

train acc, test acc | 0.95175, 0.9519

train acc, test acc | 0.9575166666666667, 0.9555

train acc, test acc | 0.95675, 0.9537

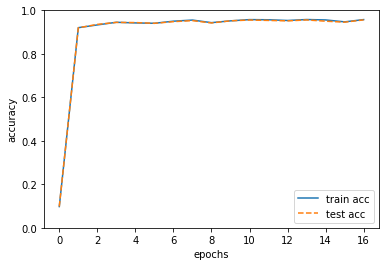
train acc, test acc | 0.95345, 0.9516

train acc, test acc | 0.95785, 0.9559

train acc, test acc | 0.9559666666666666, 0.9502

train acc, test acc | 0.9471166666666667, 0.9457

train acc, test acc | 0.9575333333333333, 0.9563



本論文の主旨は機械学習において前処理と呼ばれる部分を対象としている。そのため、主旨の前に機械学習の手法として用いたニューラルネットワークについて、ニューラルネットワークの基となったパーセプトロンと併せて簡単に説明する。

ニューラルネットワークは、パーセプトロンの出力値を求める関数を、入力値によって段階的に離散した値を返すSTEP関数から出力が0-1に値が収まるソフトマックス関数などの微分可能な関数に変えたものである（多層化する場合接続層には同じく微分可能なシグモイド関数などを用いる）。この変更によって得られる利益を述べる前にパーセプトロンの学習手順を説明する。

パーセプトロンは各独立変数に重み付けを行い、総和を求めることで入力されたデータがどの領域に属するかを判断する。その後、間違っていた場合のみ学習係数（入力値のみだと修正幅が大きすぎるため修正値の影響を下げる）と入力データとの積を修正値として既存の重みから引くことで正しい重みを学習していく。

それに対してニューラルネットワークは修正値に微分された式を用いる。微分された式は各独立変数が僅かに変化した時、従属変数がどれだけ変化するかを表す。よって、修正値の性質上パーセプトロンの学習方法と比べてより正しい重みを求めやすい。この点においてニューラルネットワークは優れていると言える。

ただし、ニューラルネットワークは文字認識という分野において多層化しなければ罠に陥りやすいと私は考える。先に述べたようにニューラルネットワークは出力関数を除いてパーセプトロンと同じ方式であるため、各独立変数と重み変数を掛け合わせて総和を取る。文字認識は色を必要とせず、グレースケールや単に０と１のみで構成される２値画像データで事足りる。この２値画像データを入力した場合出力はただ単に重みを足し算しただけとなり、重みの総和がある程度の値を持てば人間が文字と認識できないデータも文字と認識し得る。グレースケールに関してもこの問題点は変わらない。学習に用いたデータが筆圧の弱いもの中心で構成されていた場合はより簡単に、文字に共通しそうな点を塗りつぶしただけで文字とは言えないデータも文字として認識させられる。これは外れ値を除外する機能が認識可能な文字と文字の境界に設けられていたとしても重みの小さいことが多い四隅など用いてを調整すれば十分可能である。また、そんなことをしなくても文章から各文字を取り出す前処理の結果によっては偶然そのような誤認識が発生する可能性は十分あると思われる。

この問題を解決するために私はまず人間は文字と柄をどのように区別しているのだろうかを考えた。例えばモノクロ模様や迷彩柄の一部を文字として捉える人はいないだろう。しかし、文字を基にした企業の商標は文字と認識できるし、少なくとも私の属する日本人は中国語の簡体字や繁体字を漢字と認識できると思われる。この違いを考えた時、私は文字には連続性が存在することに着目した。文字はミクロの視点で見たとき連続して点を取り、多くの場合複数の線が交差して構成される。これは人間が普段文字を描く際に筆記具を擦って痕を残す方法を取っているためだ。これゆえに文字は点だけで構成されることは少なく、塗りつぶされた図形のように面を持つことがない。

そこで私はその連続性を強調したデータ形式ならこの問題は生まれにくいのではないかと考えた。文字が連続性という特徴を持つのであれば、重みにも各文字に特徴的な連続性を示す箇所に重みが大きく設定される。そのため、連続性を示す箇所の数値が増加するに連れて出力もより大きくなる。また、文字が中央に中心がなく左右上下に偏って描写されていても従来ではそのような偏りがあるデータも考慮して学習される。しかしそのような偏りがあるデータをある程度含めば基本的に白である端を示す変数がよく変化し、重みが小さくなる。よって、出力には大した影響を及ぼさない。そこでそのような連続性を強調できるデータ形式はないかを探し、画像データ圧縮に用いられるランレングス符号化という手法を採用することにした。理由としてはこの手法が簡単な構成であるためにデータを追いやすく連続性も軸は限定されるものの強調される点を評価したためだ。

私は問題点を説明した際に多層化しなければと条件を付け加えたが、これは多層化されることによって以下のように解決される。まずニューラルネットワークは多層化されることによって、１層目で各ピクセルに重みが割り振られているパターンを２層目以降では各層の積という多くの数検証しているイメージとして捉えてもらって構わない。そして私は文字には連続性という特徴があると述べたが、この特徴はその連続したピクセルに対して同じように大きいまたは小さい重みが設定されていればよい。そのため、そのような極端なパターンに対して２層目以降で大きく重みが振られていれば連続性に対しても処理時間という代償を払えば上手く特徴を捉えることができる。そして文字の位置に偏りがあるデータでもずれた位置に適したパターンが存在すればそのパターンの影響が大きくなり、しっかりと正答できる。

ここでこのランレングス符号化が具体的にどのような操作なのかを説明する。２値画像データにおいて、画像データが白か黒かの２パターンしか情報を持たないために隣り合わせの値が連続することが多い。そして値が変わる時、元が白であった場合は黒、元が黒であった場合は白となる。つまり隣の値は連続しているか反転するかで表せる。そのため、連続している場合カウントし、反転した場合カウントを終了して別のカウントを開始する。この時開始時の色が決まっていれば圧縮されたデータも元の画像データに戻すことができる。文字データは多くの場合四隅にまで記入することは無いため、白として考えてよいと思われる。

ランレングス符号化されたデータを機械学習に用いることによって生まれる問題点として次が想定される。ランレングス符号化前の元データを学習に用いると各位置による重みの情報が含まれる。しかし、ランレングス符号化されたデータは各ピクセルが凝縮され、位置情報が失われる。一見特に問題がないように思われるが、これは重みを見れば理解できる。元データでは各ピクセルに重みが割り振られるが、ランレングス符号化されたデータで学習された重みを元データに直して考えると同じ重みが連続して与えられる。その結果両隣が異なる色でない限り各文字で特に特徴的なピクセルに大きく割り振られるはずの重みは隣と同じ大きさとなる。また、このランレングス符号化は縦のみや横のみといった一方向での連続した情報しか持たないため、別方向の特徴的な連続は一切考慮されない。