

Fase 3: Object Classification

Vision C++ voor Gevorderden

Simcha van Helvoort (2132205)

Loes Modderman (2132188)

Eva Mol (2048715)

Minor Vision and Robotics

Avans Hogeschool

Voorwoord

De afgelopen twee weken hebben wij met zijn drieën gewerkt aan het opzetten van een werkend neurale netwerk. Voor het classificeren van objecten hebben we gekozen om handgeschreven getallen te classificeren.

Omdat wij aan het begin van fase 2 ervoor gekozen hebben om verder te gaan in python, was het voor ons noodzakelijk om de aangegeven code, `avansvisionlib20.cpp`, om te schrijven naar een python bestand. Dit was nodig voor ons om de eerste opdrachten af te kunnen ronden.

Inhoudsopgave

1	Opdracht 1	4
1.1	Opdrachtoomschrijving	4
1.2	Functiecode	4
1.3	Resultaat	7
2	Opdracht 2	8
2.1	Opdrachtoomschrijving	8
2.2	Functiecode	8
2.3	Testcode	14
2.4	Resultaat	16
3	Opdracht 3	17
3.1	Opdrachtoomschrijving	17
3.2	Features	17
3.3	Neuraal Netwerk	17
3.3.a	Initialiseer netwerk	18
3.3.b	Forward propagate	18
3.3.c	Back propagate error	18
3.3.d	Train netwerk	19
3.3.e	Voorspellen	19
3.4	Code	20
3.4.a	Uitvoerende code	20
3.4.b	Traincode	20
3.4.c	Testcode	23
3.4.d	Functiecode	24
3.5	Resultaat	29
3.5.a	Confusion Matrix	30
	Bibliografie	31

1 | Opdracht 1

Perceptron

1.1 Opdrachtschrijving

Het perceptron is, binnen de machine learning, een algoritme dat binaire klassen traint en leert. Dit houdt in dat perceptron-functies bepalen of een input, een vector, behoort tot een klasse, ja of nee (binair 1 of 0). Het is een lineaire classificatiemethode, wat wil zeggen dat het algoritme een voorspelling maakt door gewichten toe te kennen aan een classificatievector. Het algoritme verwerkt elementen één voor één.

Het perceptron algoritme, aangeleverd door Avans, werkt met random gegenereerde gewichten, en een threshold. De paarsgewijze vermenigvuldigde som van de gewichten maal de waarde van de variabele, geeft een indicatie voor de output.

Het verschil tussen de werkelijke output, en de voorspelde output laat het algoritme leren de threshold en de respectievelijke gewichten te verhogen, tot het verschil kleiner is dan ε .

In deze opdracht is het de bedoeling dat wij, met een kleine aanpassing in de aangeleverde code vanuit Avans, een perceptron kunnen trainen voor de functie: X1 AND X2 OR X3.

1.2 Functiecode

```
1 import numpy as np
2 import pandas as pd
3 from random import randint
4
5 def perceptronOutput(row, weight, threshold):
6     # bereken input
7
8     input = np.dot(weight, row)
9
10    # bepaal output
11    if (input > threshold):
12        return 1.0
```

```

13     else:
14         return 0.0
15
16     # waarheidstabel AND functie
17     tabelX1ANDX2 = [[ 0.0, 0.0, 0.0 ],
18                     [ 0.0, 1.0, 0.0 ],
19                     [ 1.0, 0.0, 0.0 ],
20                     [ 1.0, 1.0, 1.0 ]]
21
22     #waarheidstabel AND OR functie
23     tabelX1ANDX2ORX3 = [[ 0.0, 0.0, 0.0, 0.0 ],
24                          [ 0.0, 0.0, 1.0, 0.0 ],
25                          [ 0.0, 1.0, 0.0, 1.0 ],
26                          [ 1.0, 0.0, 0.0, 0.0 ],
27                          [ 1.0, 1.0, 0.0, 0.0 ],
28                          [ 1.0, 0.0, 1.0, 1.0 ],
29                          [ 0.0, 1.0, 1.0, 1.0 ],
30                          [ 1.0, 1.0, 1.0, 1.0 ]]
31
32     variableNames = ['X1', 'X2', 'X3', 'output']
33
34     # boolean flag wordt gekeken als output meer dan EPSILON afwijkt van ...
verwachte output
35     flag = False
36     EPSILON = 0.000001
37
38
39     # waarheidstabel afdrukken
40     df = pd.DataFrame(tabelX1ANDX2ORX3)
41     df.columns = variableNames
42
43     numberOfCols = len(df.columns)
44     numberOfXs = len(df.columns) - 1
45     numberOfWeights = numberOfCols - 1
46
47     #startwaarden van de weegfactoren (willekeurig)
48     weights = []
49
50     for w in range(numberOfWeights):
51         weights.append(randint(0, 1500))
52
53     # startwaarde van de threshold (willekeurig)
54     threshold = randint(0,255)
55
56
57     aantalRuns = 0
58     outputList = []
59

```

```
60 if len(df) > 0:
61     #voer een run uit
62     for row in range(len(df)):
63         print("Evaluating Row Nr \t:", row + 1)
64         rowInput = df.iloc[row][0:numberOfXs]
65         # bereken actuele output van het nog ongetrainde perceptron
66         output = perceptronOutput(rowInput, weights, threshold)
67
68         # bereken de afwijking van de actuele output t.o.v. verwachte output
69         DeltaOutput = (output - df.iloc[row]['output'])
70
71         while (abs(DeltaOutput) > EPSILON):
72
73             # berekening aanpassing W1 en W2
74             DeltaW = -DeltaOutput*df.iloc[row][0:numberOfXs]
75
76             # threshold, W1 en W2 aanpassen
77             threshold += DeltaOutput
78             weights += DeltaW
79
80             output = perceptronOutput(rowInput, weights, threshold)
81             # bereken de afwijking van de actuele output t.o.v. ...
82             verwachte output
83             DeltaOutput = (output - df.iloc[row]['output'])
84             aantalRuns += 1
85
86             aantalRuns += 1
87             print("Runs ", aantalRuns, "   Threshold = ", threshold)
88
89             outputList.append(output)
90
91 outputdf = pd.DataFrame(outputList)
92 outputdf.columns = ['output Perceptron']
93 df[outputdf.columns] = outputdf
94 print(df)
```

1.3 Resultaat

Het resultaat bij deze opdracht geeft telkens de tussentijdse aanpassing van de thresholdwaarde weer, zie 1.1. Het programma controleert bij elke run of de berekende output gelijk is aan de verwachte output. Als dit niet het geval is, zal het de threshold één verhogen of verlagen.

Zodra de outputs wel gelijk aan elkaar zijn, gaat het programma door naar de volgende rij en zal het vanaf de huidige thresholdwaarde dezelfde processen uitvoeren.

Door telkens de thresholdwaarde met één aan te passen, kan de thresholdwaarde nooit teveel veranderen per run.

Aan het einde van het proces, geeft het programma een tabel terug met de ingegeven inputwaarden, outputwaarden en de berekende outputwaarde van de perceptron.

```
Evaluating Row Nr : 1
Runs 1 Threshold = 73
Evaluating Row Nr : 2
Runs 128 Threshold = 199.0
Evaluating Row Nr : 3
Runs 166 Threshold = 162.0
Evaluating Row Nr : 4
Runs 595 Threshold = 590.0
Evaluating Row Nr : 5
Runs 650 Threshold = 644.0
Evaluating Row Nr : 6
Runs 651 Threshold = 644.0
Evaluating Row Nr : 7
Runs 765 Threshold = 531.0
Evaluating Row Nr : 8
Runs 766 Threshold = 531.0
X1 X2 X3 output output Perceptron
0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 1.0 0.0 0.0
2 0.0 1.0 0.0 1.0 1.0
3 1.0 0.0 0.0 0.0 0.0
4 1.0 1.0 0.0 0.0 0.0
5 1.0 0.0 1.0 1.0 1.0
6 0.0 1.0 1.0 1.0 1.0
7 1.0 1.0 1.0 1.0 1.0
```

(a)

```
Evaluating Row Nr : 1
Runs 1 Threshold = 115
Evaluating Row Nr : 2
Runs 473 Threshold = 586.0
Evaluating Row Nr : 3
Runs 699 Threshold = 361.0
Evaluating Row Nr : 4
Runs 837 Threshold = 498.0
Evaluating Row Nr : 5
Runs 959 Threshold = 619.0
Evaluating Row Nr : 6
Runs 960 Threshold = 619.0
Evaluating Row Nr : 7
Runs 961 Threshold = 619.0
Evaluating Row Nr : 8
Runs 962 Threshold = 619.0
X1 X2 X3 output output Perceptron
0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 1.0 0.0 0.0
2 0.0 1.0 0.0 1.0 1.0
3 1.0 0.0 0.0 0.0 0.0
4 1.0 1.0 0.0 0.0 0.0
5 1.0 0.0 1.0 1.0 1.0
6 0.0 1.0 1.0 1.0 1.0
7 1.0 1.0 1.0 1.0 1.0
```

(b)

```
Evaluating Row Nr : 1
Runs 1 Threshold = 138
Evaluating Row Nr : 2
Runs 22 Threshold = 158.0
Evaluating Row Nr : 3
Runs 23 Threshold = 158.0
Evaluating Row Nr : 4
Runs 376 Threshold = 510.0
Evaluating Row Nr : 5
Runs 808 Threshold = 941.0
Evaluating Row Nr : 6
Runs 1045 Threshold = 705.0
Evaluating Row Nr : 7
Runs 1046 Threshold = 705.0
Evaluating Row Nr : 8
Runs 1047 Threshold = 705.0
X1 X2 X3 output output Perceptron
0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 1.0 0.0 0.0
2 0.0 1.0 0.0 1.0 1.0
3 1.0 0.0 0.0 0.0 0.0
4 1.0 1.0 0.0 0.0 0.0
5 1.0 0.0 1.0 1.0 1.0
6 0.0 1.0 1.0 1.0 1.0
7 1.0 1.0 1.0 1.0 1.0
```

(c)

Figuur 1.1: Screenshots van de resultaten

2 | Opdracht 2

Back Propagation Neural Network

2.1 Opdrachtschrijving

Het perceptron-algoritme geeft niet altijd de juiste output, een voorbeeld hiervan is de XOR-functie.

Back Propagation is een methode die gebruik maakt van de bijdrage van de fout in elke neuron, nadat een hoeveelheid data is verwerkt. Net als bij het perceptron-algoritme worden ook hier de gewichten aangepast tot de fout onder de maximaal gegeven fout komt.

Bij deze opdracht is het de bedoeling om een trainingsset te maken voor de XOR-functie. Hiermee kan het neurale netwerk getraind worden. Daarna kan het neurale netwerk getest worden met behulp van een testset.

2.2 Functiecode

```
1 import numpy as np
2 import avansvisionlib as avl
3
4 # // TRAININGSET: I0 because of bias V0
5 # //
6 # // setnr      I0      I1      I2      I3      I4      O1      O2
7 # // 1          1.0     0.4     -0.7    0.1     0.71    0.0     0.0
8 # // 2          1.0     0.3     -0.5    0.05    0.34    0.0     0.0
9 # // 3          1.0     0.6     0.1     0.3     0.12    0.0     1.0
10 # // 4          1.0     0.2     0.4     0.25    0.34    0.0     1.0
11 # // 5          1.0    -0.2     0.12    0.56    1.0     1.0     0.0
12 # // 6          1.0     0.1    -0.34    0.12    0.56    1.0     0.0
13 # // 7          1.0    -0.6     0.12    0.56    1.0     1.0     1.0
14 # // 8          1.0     0.56    -0.2     0.12    0.56    1.0     1.0
15
16
17 def loadTrainingSet1():
```



```

18     # input of trainingsset
19     # number of columns = number of inputneurons of the BPN
20     ITset = np.array([1, 0.4, -0.7, 0.1, 0.71,
21                       1, 0.3, -0.5, 0.05, 0.34,
22                       1, 0.6, 0.1, 0.3, 0.12,
23                       1, 0.2, 0.4, 0.25, 0.34,
24                       1, -0.2, 0.12, 0.56, 1.0,
25                       1, 0.1, -0.34, 0.12, 0.56,
26                       1, 0.6, 0.12, 0.56, 1.0,
27                       1, 0.56, -0.2, 0.12, 0.56])
28     ITset.resize(8, 5)
29
30     # output of trainingset
31     # number of columns = number of outputneurons of the BPN
32     OTset = np.array([0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1])
33     OTset.resize(8, 2)
34
35     return [ITset, OTset]
36
37 # // TRAININGSET binary function 01 = (I1 OR I2) AND I3
38 # // without bias
39 # // setnr    I1    I2    I3    01
40 # // 1        0    0    0    0
41 # // 2        0    0    1    0
42 # // 3        0    1    0    0
43 # // 4        0    1    1    1
44 # // 5        1    0    0    0
45 # // 6        1    0    1    1
46 # // 7        1    1    0    0
47 # // 8        1    1    1    1
48
49 def loadBinaryTrainingSet1():
50     # input of trainingset (without bias)
51     ITset = np.array([0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
52                       1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1])
53     ITset.resize(8, 3)
54
55     # output of trainingset
56     OTset = np.array([0, 0, 0, 1, 0, 1, 0, 1])
57     OTset.resize(8, 1)
58
59     return [ITset, OTset]
60
61 def loadTrainingXOR():
62     # input of trainingsset
63     # number of columns = number of inputneurons of the BPN
64     # zonder bias
65

```

```

66     ITset = np.array([0, 0,
67                       0, 1,
68                       1, 0,
69                       1, 1])
70     ITset.resize(4, 2)
71
72     # output of trainingset
73     # number of columns = number of outputneurons of the BPN
74     OTset = np.array([0,
75                       1,
76                       1,
77                       0])
78     OTset.resize(4, 1)
79
80     return [ITset, OTset]
81
82
83 def initializeBPN(inputNeurons, hiddenNeurons, outputNeurons):
84     inputNeurons = int(inputNeurons)
85     hiddenNeurons = int(hiddenNeurons)
86     outputNeurons = int(outputNeurons)
87
88     # Set all weightfactors to a random value
89     V0 = [np.random.random() for ii in range(hiddenNeurons) for jj in ...
90           range(inputNeurons)]
91     V0 = np.array(V0)
92     V0.resize(hiddenNeurons, inputNeurons)
93
94     W0 = [np.random.random() for ii in range(outputNeurons) for jj in ...
95           range(hiddenNeurons)]
96     W0 = np.array(W0)
97     W0.resize(outputNeurons, hiddenNeurons)
98
99     # Initial modification of the weightfactors
100    dV0 = [[None for row in range(inputNeurons)] for col in ...
101            range(hiddenNeurons)]
102    dW0 = [[None for row in range(hiddenNeurons)] for col in ...
103            range(outputNeurons)]
104
105    dV0 = avl.setValue(dV0, 0)
106    dW0 = avl.setValue(dW0, 0)
107
108    return V0, W0, dV0, dW0
109
110
111 # def testBPN():
112 #     IT = np.array([0.4, -0.7, 0.3, -0.5, 0.6, 0.1, 0.2, 0.4, 0.1, -0.2])
113 #     IT.resize(5, 2)

```

```

110 #
111 #     OT = np.array([0.1, 0.05, 0.3, 0.25, 0.12])
112 #     OT.resize((5, 1))
113 #
114 #     VO = np.array([0.1, 0.4, -0.2, 0.2])
115 #     VO.resize(2, 2)
116 #
117 #     WO = np.array([0.2, -0.5])
118 #     WO.resize((2, 1))
119 #
120 #     dVO = np.array([0, 0, 0, 0])
121 #     dVO.resize(2, 2)
122 #
123 #     dWO = np.array([0, 0])
124 #     dWO.resize(2, 1)
125 #
126 #     return [IT, OT, VO, WO, dVO, dWO]
127
128
129 def calculateOutputHiddenLayer(input_inputLayer, weightfactorV):
130     # STEP 1: output_inputLayer = input_inputLayer
131     input_inputLayer.resize(len(input_inputLayer),1)
132     OI = input_inputLayer
133     # STEP 2: initializing weights (already done)
134
135     # STEP 3: calculate input_hiddenLayer
136     # Vt = np.transpose(weightfactorV)
137     IH = np.dot(weightfactorV, OI)
138
139     # STEP 4: calculate output_hiddenLayer
140     hiddenNeurons = weightfactorV.shape[0]
141     OH = np.zeros(hiddenNeurons)
142     for row in range(hiddenNeurons):
143         value = 1 / (1 + np.exp(-IH[row]))
144         OH[row] = value
145     OH = np.array(OH)
146     OH.resize(hiddenNeurons,1)
147     return OH
148
149
150 def calculateOutputBPN(OH, W):
151     # STEP 5: calculate input_outputLayer
152     # Wt = np.transpose(W)
153     IO = np.dot(W, OH)
154
155     # STEP 6: calculate output_outputLayer
156     outputNeurons = W.shape[0]
157     OO = []

```

```

158     for row in range(outputNeurons):
159         value = 1 / (1 + np.exp(- IO[row]))
160         OO.append(value)
161     OO = np.array(OO)
162     OO.resize(outputNeurons, 1)
163
164     return OO
165
166
167 def calculateOutputBPNErrror(OO, OT):
168     # STEP 7: calculate the error
169     sumSqrErr = 0
170     diff = 0
171     for row in range(OT.shape[0]):
172         diff = OT[row] - OO[row]
173         sumSqrErr += (diff * diff)
174     outputError = 0.5 * sumSqrErr
175
176     return outputError
177
178 # def getMultiplication(firstMatrix, matrixToMultiplyWith, outputShape):
179 #     rowsFM = firstMatrix.shape[0]
180 #     rowsMTMW = outputShape.shape[0]
181 #     newMatrix = [[0 for row in range(rowsFM)] for col in range(rowsMTMW)]
182 #     for col in range(rowsMTMW):
183 #         for row in range(rowsFM):
184 #             entrydStarT = np.dot(firstMatrix[row], ...
matrixToMultiplyWith[0][col])
185 #             newMatrix[[col][0]][row] = entrydStarT
186 #     newMatrix = np.array(newMatrix)
187 #     newMatrix.resize(rowsMTMW, rowsFM)
188
189
190
191 def adaptVW(OT, OO, OH, OI, W0, dW0, V0, dV0):
192     # adapt weightfactors W
193     # STEP 8:
194     ALPHA = 1
195     ETHA = 0.6
196     nrOutPutTrSet = OT.shape[0]
197     OT = np.array(OT)
198     OT.resize(nrOutPutTrSet, 1)
199     OOerror = OT - OO
200
201     d = []
202     for row in range(nrOutPutTrSet):
203         di = (OT[row, 0] - OO[row, 0]) * OO[row, 0] * (1 - OO[row, 0])
204         d.append(di)

```

```

205     d = np.array(d)
206     #d.resize(nrOutPutTrSet, 1)
207
208     Y = [[0 for r in range(OH.shape[0])] for c in range(nrOutPutTrSet)]
209     for j in range(nrOutPutTrSet):
210         for i in range(OH.shape[0]):
211             entry = np.dot(OH[i],d[j])
212             Y[[j][0]][i] = entry
213     Y = np.array(Y)
214     Y.resize(nrOutPutTrSet, OH.shape[0])
215
216
217     # STEP 9:
218     dW = ALPHA * dW0 + ETHA * Y
219
220     # adapt weightfactors V
221     # STEP 10:
222     OError = np.dot(W0.transpose(),d)
223     OError.resize(OH.shape[0], 1)
224
225     # STEP 11:
226     dStar = [0 for r in range(OH.shape[0])]
227     for row in range(OH.shape[0]):
228         dStari = OError[row, 0] * OH[row, 0] * (1 - OH[row, 0])
229         dStar[row] = dStari
230     dStar = np.array(dStar)
231     dStar.resize(OH.shape[0], 1)
232
233     # STEP 12:
234     dStarT = np.transpose(dStar)
235     X = [[0 for r in range(OI.shape[0])] for c in range(OH.shape[0])]
236     for col in range(OH.shape[0]):
237         for row in range(OI.shape[0]):
238             entrydStarT = np.dot(OI[row], dStarT[0][col])
239             X[[col][0]][row] = entrydStarT
240     # X = OI * dStarT
241     X = np.array(X)
242     X.resize(OH.shape[0], OI.shape[0])
243
244     # STEP 13
245     dV = ALPHA * dV0 + ETHA * X
246
247     # Update matrices with weightfactors
248     # STEP 14:
249     V = V0 + dV
250     W = W0 + dW
251
252     return V, W

```

```

253
254
255 def BPN(II, V, W):
256     OH = calculateOutputHiddenLayer(II, V)
257     OO = calculateOutputBPN(OH, W)
258     return OO

```

2.3 Testcode

```

1  import avlBPNEvaWorking as BPN
2  import avansvisionlib as avl
3  import pandas as pd
4
5  # Maximale fout die toegestaan wordt in de output voor de training input
6  MAX_OUTPUT_ERROR = 1E-10
7  # maximaal aantal runs dat uitgevoerd wordt bij het trainen
8  MAXRUNS = 10000
9
10
11 print("\n Testset laden... \n \n")
12
13 ITset, OTset = BPN.loadTrainingXOR()
14
15 originalSet = pd.DataFrame(ITset)
16 originalSet.columns = ["I0", "I1"]
17 outputSet = pd.DataFrame(OTset)
18 outputSet.columns = ["Expected Output"]
19 originalSet[outputSet.columns] = outputSet
20 print(originalSet)
21
22 print("BPN format: ")
23 print("\t BPN Inputlayer = " + str(ITset.shape[1]) + " neurons")
24 print("\t BPN Outputlayer = " + str(OTset.shape[1]) + " neurons")
25 hiddenNeurons = int(input("Please choose a number of hidden neurons: "))
26 print("Thank You!!")
27
28 print("Initialize BPN... ")
29
30 V0, W0, dV0, dW0 = BPN.initializeBPN(ITset.shape[1], hiddenNeurons, ...
    OTset.shape[1]);
31
32 print("initial values of weight matrices V0 and W0 ...
    \n*****")
33 avl.printMatrix(V0)
34 avl.printMatrix(W0)

```

```

35
36 input("==> PRESS ENTER")
37
38 outputError0 = MAX_OUTPUT_ERROR + 1
39 outputError1 = MAX_OUTPUT_ERROR + 1
40 sumSqrDiffError = MAX_OUTPUT_ERROR + 1
41
42 runs = 0
43
44 while ((sumSqrDiffError > MAX_OUTPUT_ERROR) & (runs < MAXRUNS)):
45     sumSqrDiffError = 0
46     for inputSetRowNr in range(ITset.shape[0]):
47         IT = ITset[inputSetRowNr]
48         OT = OTset[inputSetRowNr]
49         OH = BPN.calculateOutputHiddenLayer(IT, V0)
50         OO = BPN.calculateOutputBPN(OH, W0)
51         [V1, W1] = BPN.adaptVW(OT, OO, OH, IT, W0, dW0, V0, dV0)
52         outputError0 = BPN.calculateOutputBPNErrors(OO, OT)
53         outputError1 = BPN.calculateOutputBPNErrors(BPN.BPN(IT, V1, W1), OT)
54         sumSqrDiffError += (outputError1 - outputError0) * (outputError1 ...
            - outputError0)
55         V0 = V1
56         W0 = W1
57     print("sumSqrDiffError = " + str(sumSqrDiffError))
58     runs += 1
59
60 outputVectorBPN = OTset.copy()
61 for inputSetRowNr in range(ITset.shape[0]):
62     for inputSetColNr in range(OTset.shape[1]):
63         inputVectorTrainingSet = ITset[inputSetRowNr]
64         outputVectorBPN[inputSetRowNr][inputSetColNr] = ...
            round(round(BPN.BPN(inputVectorTrainingSet, V0, ...
                W0)[inputSetColNr][0]),1)
65
66 print("BPN Training is ready!")
67 print("Runs = " + str(runs))
68 outputTraining = pd.DataFrame(outputVectorBPN)
69 outputTraining.columns = ["Output BPN"]
70 originalSet[outputTraining.columns] = outputTraining
71 print(originalSet)

```

2.4 Resultaat

In afbeelding 2.1.a is een screenshot gegeven van de input van de functie. De input en de output wordt al in de code zelf bepaald. Alleen de gebruiker moet nog zelf bepalen hoeveel hidden neurons hij wilt dat er gebruikt wordt. Hierna begint het programma met het initialiseren van het BPN en van de weegfactoren. Zodra men het programma start door op Enter te drukken, wordt het BPN getraind op de ingegeven trainingsset.

In afbeelding 2.1.b is een screenshot van de output gegeven. Zoals te zien is, geeft het programma aan als de training afgerond is. Verder geeft het meteen weer hoeveel 'runs' het programma erover heeft gedaan om zichzelf te trainen. Het programma stopt met trainen zodra het een van de twee voorwaarde heeft bereikt:

1. Het programma heeft het maximale aantal runs bereikt.
2. De berekende error, die het programma bij de laatste run heeft berekend, is kleiner dan 10^{-10}

In het geval bij 2.1.b stopt het programma omdat het voorwaarde 2 heeft bereikt. Hierna geeft het programma de gegeven input, verwachte output en berekende output weer in een tabel. Hiermee kan de gebruiker voor zichzelf visueel controleren of het programma correct gewerkt heeft.

```
Testset laden...

  I0  I1  Expected Output
0  0  0                0
1  0  1                1
2  1  0                1
3  1  1                0
BPN format:
  BPN Inputlayer = 2 neurons
  BPN Outputlayer = 1 neurons
Please choose a number of hidden neurons: 3
Thank You!!
Initialize BPN...
initial values of weight matrices W0 and W1
*****
==> PRESS ENTER
sumSqrDiffError = [ 0.00081713]
sumSqrDiffError = [ 0.00083208]
sumSqrDiffError = [ 0.00083609]
sumSqrDiffError = [ 0.0008337]
sumSqrDiffError = [ 0.00082814]
sumSqrDiffError = [ 0.0008214]
```

(a)

```
sumSqrDiffError = [ 1.00593412e-10]
sumSqrDiffError = [ 1.00482235e-10]
sumSqrDiffError = [ 1.00371212e-10]
sumSqrDiffError = [ 1.00260343e-10]
sumSqrDiffError = [ 1.00149627e-10]
sumSqrDiffError = [ 1.00039065e-10]
sumSqrDiffError = [ 9.99286559e-11]
BPN Training is ready!
Runs = 4097
  I0  I1  Expected Output  Output BPN
0  0  0                0        0
1  0  1                1        1
2  1  0                1        1
3  1  1                0        0
```

(b)

Figuur 2.1: Screenshot van de Input en de Output van BPN

3 | Opdracht 3

Classificatie van objecten

3.1 Opdrachtoomschrijving

In deze opdracht wordt alle opgedane kennis over pre-processing en over feature-detection toegepast op een reeks objecten. Deze objecten worden gebruikt om een neurale netwerk zo te trainen, dat het ingeven van een soortgelijke testset een correcte output teruggeeft.

3.2 Features

De features die gekozen zijn om te gebruiken in ons neurale netwerk is de Histogram of Oriented Gradients, HOG.[1]

3.3 Neuraal Netwerk

Het neurale netwerk dat gekozen is om de classificatie van objecten uit te voeren, heet het Backwards Propagation Neural Network, ookwel BPN genoemd in onze code. BPN is gebaseerd op het verwerken van informatie door neurale cellen, neuronen genaamd, bijvoorbeeld in het menselijk lichaam. Een neuron krijgt hier een input signaal uit de synapsen. Synapsen zijn de connecties tussen een axon van de ene cel en een dendriet van de andere cel.

De bedoeling van de backpropagation methode is om een model te maken waarin de wegen, die gekoppeld zijn aan de input signalen, worden aangepast, om voor het gewenste output signaal te zorgen. De wegen worden aangepast, zodra het systeem observeert dat de berekende output afwijkt van de gewenste output. Hierna loopt het systeem nogmaals de berekening door. Dit blijft doorgaan totdat de berekende output overeen komt met de gewenste output. Het systeem traint zichzelf met behulp van deze methode.

Om een BPN op te zetten, zijn er verschillende stappen nodig [2]:

1. Initialiseer netwerk
2. Forward propagate
3. Back propagate error
4. Train netwerk
5. Voorspellen

3.3.a Initialiseer netwerk

Het initialiseren van het netwerk houdt in dat de weegfactoren geïnitieerd moeten worden. Elk neuron heeft een aantal bijbehorende weegfactoren. Eén weging voor elke connectie tussen de input- en de hiddenlayer en één weging voor de 'bias'.

Een netwerk is verdeeld in verschillende van zulke lagen. Er is een input-, hidden- en een outputlayer. De input- en outputlayer is altijd één laag. De hiddenlayer kan echter bestaan uit meerdere lagen. De inputlayer is een rij met alle data van de gebruikte trainingset. Alle hiddenlayers kun je zien als een soort black box. De grootte van een hiddenlayer kan de gebruiker dan ook zelf bepalen. De outputlayer is een laag met een grootte van een neuron voor elke klasse.

Het aantal weegfactoren dat geïnitieerd moeten worden, heeft te maken met de grootte van de layers. Er bestaat namelijk voor elke connectie, tussen alle neuronen in de input- en hiddenlayer, een weegfactor. Hetzelfde geldt voor de weegfactoren tussen de hidden- en de outputlayer.

3.3.b Forward propagate

Met behulp van de geïnitieerde weegfactoren en het neurale netwerk, kan er een output berekend worden, door een inputsignaal in te geven en dit signaal door elke laag heen te laten gaan. Deze techniek heet forward propagation.

3.3.c Back propagate error

Met forward propagation berekend het programma een output voor een bepaalde input. Op dit moment kan het programma controleren of de berekende output gelijk is aan de gewenste output, of niet. Deze berekende error koppelt het programma terug naar de hiddenlayers. Hier kan het aangeven dat de output niet correct is en dat de weegfactoren dus aangepast moeten worden. Dit heet de back propagate error.

3.3.d Train netwerk

Het neurale netwerk wordt getraind door meerdere keren over het programma heen te gaan. Dat wil zeggen de forward propagation, de back propagation error en het, aan de hand van deze informatie, het updaten van de weegfactoren in het netwerk.

3.3.e Voorspellen

Nadat het neurale netwerk op een correcte manier getraind is, kunnen er voorspellingen mee gemaakt worden. In dit geval heeft het neurale netwerk met het trainen bepaald wat de weegfactoren in alle gevallen moeten zijn en deze staan nu vast.

Om een voorspelling te maken, geeft de gebruiker de data mee van de testset. Let hierbij op dat de testset bestaat uit andere data dan de trainingsset. Deze data wordt aan het neurale netwerk meegegeven als inputlayer. Het enige wat het neurale netwerk nu hoeft uit te voeren is forward propagation. Dit geeft een output, wat een voorspelling maakt voor de input. De reden dat enkel forward propagation in dit geval genoeg is, is omdat het programma hiervoor al getraind is op deze klassen.

3.4 Code

3.4.a Uitvoerende code

```

1  import trainingProgram as trainp
2  import testProgram as testp
3  import extractFeatures as ef
4
5  # ===== GEEF HIER HET BIJBEHORENDE PAD OP ...
6  # =====
7  imageWDTrain = 'C://VisionPlaatje//trainSmall//'
8  imageWDTest = 'C://VisionPlaatje//testSmall//'
9  # ...
10 # =====
11
12 (VO,W0) = trainp.trainHandwrittenNumbers(imageWDTrain)
13 testp.testHandwrittenNumbers(imageWDTest,VO,W0)
14
15 # ===== TEST DE WEGINGSFACTOREN ...
16 # =====
17
18 # IT = np.array(ef.extractFeatures(binaryImage))
19 # BPN.BPN(IT,VO,W0)
20 # ...
21 # =====

```

3.4.b Traincode

```

1  import cv2
2  import numpy as np
3  import avlBPNEvaWorking as BPN
4  import avansvisionlib as avl
5  import pandas as pd
6  import os
7  import extractFeatures as ef
8
9  def trainHandwrittenNumbers(imageWD):
10
11     # Maximale fout die toegestaan wordt in de output voor de training input
12     MAX_OUTPUT_ERROR = 1E-8
13     # maximaal aantal runs dat uitgevoerd wordt bij het trainen
14     MAXRUNS = 10000
15
16     imageCodes = []

```

```

17
18     for file in os.listdir(imageWD): # +folder
19         if file != ".DS_Store":
20             numberOfInputs = len(os.listdir(imageWD))
21             imageCodes.append(file)
22             image = ef.makeBinaryImage(imageWD + file)
23
24     # print(imageCodes)
25     # avl.show16SImageStretch(image, "image")
26
27     numberOfFeatures = image.size + 1
28     numberOfOutputs = 4
29
30     print("\n Testset laden... \n \n")
31     print("\t BPN Inputlayer = " + str(numberOfInputs) + " neurons")
32     print("\t BPN Outputlayer = " + str(numberOfOutputs) + " neurons")
33     hiddenNeurons = int(input("Please choose a number of hidden neurons: "))
34     print("Thank You!!")
35
36     print("Initialize BPN... ")
37
38     VO, WO, dVO, dWO = BPN.initializeBPN(numberOfFeatures, ...
39         hiddenNeurons, numberOfOutputs)
40
41     #perimeterMax, areaMax = ef.memoriseLargest(imageWD)
42     counter = 0
43     totalImg = len(imageCodes)
44     # Haal de afbeeldingen uit de map.
45     for i in range(totalImg):
46         # TODO: random aanpassen
47         counter += 1
48         print("image {} out of {}".format(counter, totalImg))
49         indexnummer = np.random.randint(len(imageCodes))
50         filename = imageCodes.pop(indexnummer)
51         print("load image {}".format(filename))
52         binaryImage = ef.makeBinaryImage(imageWD + filename)
53
54         runs = 0
55
56         sumSqrDiffError = MAX_OUTPUT_ERROR + 1
57         # looping over afbeeldingen met eenen, twee, drie, vier, ...
58         # vijfen, zessen enz
59         # Voor elke feature worden de weights bepaald
60
61         # bepaal de input en output van de traindata
62         IT = np.array(ef.extractFeatures(binaryImage))
63         OT = np.array(ef.outputHandwrittenNumbers(filename))
64         ITold = np.zeros(IT.shape)

```

```

63     print(np.sum(ITold-IT))
64
65     ITold = IT
66
67
68     while ((sumSqrDiffError > MAX_OUTPUT_ERROR) & (runs < MAXRUNS)):
69         sumSqrDiffError = 0
70         #     for inputSetRowNr in range(ITset.shape[0]): # ...
71             Afbeeldingen in de map
72         # for inputSetRowNr in range(binaryImage.shape[0]):
73
74         OH = BPN.calculateOutputHiddenLayer(IT, VO)
75         OO = BPN.calculateOutputBPN(OH, WO)
76         [V1, W1, dV1, dW1] = BPN.adaptVW(OT, OO, OH, IT, WO, dWO, ...
77             VO, dVO)
78
79         #calculate model error
80         outputError0 = BPN.calculateOutputBPNErrror(OO, OT)
81         outputError1 = BPN.calculateOutputBPNErrror(BPN.BPN(IT, V1, ...
82             W1), OT)
83         sumSqrDiffError += (outputError1 - outputError0) * ...
84             (outputError1 - outputError0)
85
86         #save weights
87         VO = V1
88         WO = W1
89         dVO = dV1
90         dWO = dW1
91
92         #print("sumSqrDiffError = " + str(sumSqrDiffError))
93         runs += 1
94     print("Runs = " + str(runs))
95     print()
96     # Print de output
97     # outputVectorBPN = OTset.copy()
98     # for inputSetRowNr in range(ITset.shape[0]):
99     #     for inputSetColNr in range(OTset.shape[1]):
100     #         inputVectorTrainingSet = ITset[inputSetRowNr]
101     #         outputVectorBPN[inputSetRowNr][inputSetColNr] = ...
102     round(round(BPN.BPN(inputVectorTrainingSet, VO, ...
103     WO)[inputSetColNr][0]), 1)
104
105     #
106     print("BPN Training is ready!")
107     print(VO)
108     print(WO)
109     return VO, WO
110
111     # outputTraining = pd.DataFrame(outputVectorBPN)

```

```

105     # outputTraining.columns = ["Output BPN"]
106     # originalSet[outputTraining.columns] = outputTraining
107     # print(originalSet)

```

3.4.c Testcode

```

1  import cv2
2  import numpy as np
3  import avlBPNEvaWorking as BPN
4  import avansvisionlib as avl
5  import pandas as pd
6  import os
7  import extractFeatures as ef
8  import re
9
10 def testHandwrittenNumbers(imageWD, V0, W0):
11     imageCodes = []
12
13     for file in os.listdir(imageWD): # +folder
14         if file != ".DS_Store":
15             numberOfInputs = len(os.listdir(imageWD))
16             imageCodes.append(file)
17
18     O0list = []
19     filenameList = []
20
21     for i in range(len(imageCodes)):
22         indexnumber = np.random.randint(len(imageCodes))
23         filename = imageCodes.pop(indexnumber)
24         filenameList.append(filename)
25         binaryImage = ef.makeBinaryImage(imageWD + filename)
26         IT = np.array(ef.extractFeatures(binaryImage))
27         O0raw = np.array(BPN.BPN(IT,V0,W0))
28         O0 = np.array(BPN.BPN(IT, V0, W0))
29         O0 = np.round(np.round(O0,1))
30         O0list.append(O0)
31         print(O0raw)
32     print(confusionMatrix(O0list, filenameList))
33
34
35 def confusionMatrix(O0list, filenameList):
36     numberOutput = []
37     for i in range(len(O0list)):
38         O0number = ef.outputToNumber(O0list[i])
39         numberOutput.append(O0number)

```

```

40
41     combinedList = []
42     for i in range(len(filenameList)):
43         combinedList.append([filenameList[i], numberOutput[i]])
44
45     filenameList.sort(key=lambda var: [int(x) if x.isdigit() else x for ...
46         x in re.findall(r'[^0-9]|[0-9]+', var)])
47
48     newCombinedList = []
49     outputTest = []
50     for files in filenameList:
51         for i in range(len(filenameList)):
52             if files == combinedList[i][0]:
53                 newCombinedList.append(combinedList[i])
54                 outputTest.append(combinedList[i][1])
55
56     realOutput = [5,9,1,4,2,6,3,1,9,8,7,5,9,3, ...
57         3,5,6,2,3,2,6,4,8,4,8,3,5,2,9,3,3,7,0,2,9,8,8,4,7,1]
58
59     confusionMat = np.zeros((10,10))
60
61     for i in range(len(realOutput)):
62         confusionMat[realOutput[i],outputTest[i]] += 1
63
64     accuracy = []
65     confusionDiagonal = confusionMat.diagonal()
66     for i in range(10):
67         if sum(confusionMat[i]) == 0:
68             accuracyPerRow = 0
69         else:
70             accuracyPerRow = confusionDiagonal[i]/(sum(confusionMat[i]))
71         accuracy.append(accuracyPerRow)
72
73     confusiondf = pd.DataFrame(confusionMat)
74     accuracydf = pd.DataFrame(accuracy)
75     accuracydf.columns = ["Accuracy"]
76     confusiondf[accuracydf.columns] = accuracydf
77
78     return confusiondf

```

3.4.d Functiecode

```

1 import numpy as np
2 import avansvisionlib as avl
3 import cv2

```



```

4  # from IPython import get_ipython
5  # ipython = get_ipython()
6  from skimage import measure
7  import os
8  import matplotlib.pyplot as plt
9  from skimage.feature import hog
10 from skimage import exposure
11
12
13 def extractFeaturePerimeter(binaryImage):
14     # function returns an integer with the perimeter (length of the contour)
15     contourVec = avl.makeContourImage(binaryImage)[1]
16     perimeter = 0
17     for i in range(len(contourVec)):
18         perimeter += len(contourVec[i])
19
20     return perimeter
21
22 def extractFeatureArea(binaryImage):
23     # function returns an integer with the area size of the region of ...
24     the object
25     labeledImage = measure.label(binaryImage, background=0)
26
27     area = 0
28     for col in range(len(labeledImage[0])):
29         for row in range(len(labeledImage)):
30             if labeledImage[row, col] == 1:
31                 area += 1
32
33     return area
34
35 def extractFeatureNumberOfHoles(binaryImage):
36     # function returns an integer with the number of holes in the object
37     # contours
38
39     contourVec = avl.makeContourImage(binaryImage)[1]
40     NumberOfContours = len(contourVec)
41
42     # blobs
43     labeledImage = measure.label(binaryImage, background=0)
44     NumberOfBlobs = np.max(labeledImage)
45
46     NumberOfHoles = NumberOfContours - NumberOfBlobs
47
48     if NumberOfHoles == 0:
49         return 0
50     else:
51         return (1 / NumberOfHoles)

```

```

51
52
53 def extractFeatureCircularity(perimeter, area):
54     #  $(4\pi * Area) / (Perimeter)$ 
55     circularity = (perimeter**2)/(4*np.pi*(area))
56
57     return circularity
58
59 def retrieveHOG(inputMatrix, doPlot=False):
60     image = inputMatrix.copy()
61     fd, hogImage = hog(image, orientations=8, pixels_per_cell=(96/14, ...
62                        96/14),
63                        cells_per_block=(1, 1), visualise=True)
64
65     hogVector = [0]*hogImage.size
66     nRows = hogImage.shape[0]
67     nCols = hogImage.shape[1]
68     cv2.imshow('HOG Image', hogImage)
69     cv2.waitKey(0)
70     cv2.destroyAllWindows()
71     index = 0
72     for cols in range(nCols):
73         for rows in range(nRows):
74             hogVector[index] = hogImage[rows][cols]
75             index += 1
76
77     return hogVector
78
79 def pixelsOnly(binaryImage):
80     imgVector = [0]*binaryImage.size
81     nRows = binaryImage.shape[0]
82     nCols = binaryImage.shape[1]
83     index = 0
84     for cols in range(nCols):
85         for rows in range(nRows):
86             imgVector[index] = binaryImage[rows][cols]
87             index += 1
88
89     return imgVector
90
91 def extractFeatures(binaryImage):
92     # ...
93     =====
94     #     perimeter = extractFeaturePerimeter(binaryImage)
95     #     area = extractFeatureArea(binaryImage)
96     #     nrHoles = extractFeatureNumberOfHoles(binaryImage)
97     # ...
98     =====
99     hogVector = retrieveHOG(binaryImage, doPlot=False)
100    IT = [1]

```

```

96     IT.extend(hogVector)
97     return IT
98
99 def outputHandwrittenNumbers(filename):
100     #filename = str(filename)
101     if "zero" in filename:
102         OT = [0,0,0,0]
103     elif "one" in filename:
104         OT = [0,0,0,1]
105     elif "two" in filename:
106         OT = [0,0,1,0]
107     elif "three" in filename:
108         OT = [0,0,1,1]
109     elif "four" in filename:
110         OT = [0,1,0,0]
111     elif "five" in filename:
112         OT = [0,1,0,1]
113     elif "six" in filename:
114         OT = [0,1,1,0]
115     elif "seven" in filename:
116         OT = [0,1,1,1]
117     elif "eight" in filename:
118         OT = [1,0,0,0]
119     elif "nine" in filename:
120         OT = [1, 0, 0, 1]
121     else:
122         print("Could not correctly classify object.")
123     return OT
124
125
126 def outputToNumber(OO):
127     OO.resize(1,len(OO))
128     if (sum(sum(OO == np.array([0,0,0,0])))) == 4:
129         numberRecognized = 0
130     elif (sum(sum(OO == np.array([0,0,0,1])))) == 4:
131         numberRecognized = 1
132     elif (sum(sum(OO == np.array([0,0,1,0])))) == 4:
133         numberRecognized = 2
134     elif (sum(sum(OO == np.array([0,0,1,1])))) == 4:
135         numberRecognized = 3
136     elif (sum(sum(OO == np.array([0,1,0,0])))) == 4:
137         numberRecognized = 4
138     elif (sum(sum(OO == np.array([0,1,0,1])))) == 4:
139         numberRecognized = 5
140     elif (sum(sum(OO == np.array([0,1,1,0])))) == 4:
141         numberRecognized = 6
142     elif (sum(sum(OO == np.array([0,1,1,1])))) == 4:
143         numberRecognized = 7

```

```
144     elif (sum(sum(00 == np.array([1,0,0,0])))) == 4:
145         numberRecognized = 8
146     elif (sum(sum(00 == np.array([1,0,0,1])))) == 4:
147         numberRecognized = 9
148     else:
149         print("Could not correctly classify object.")
150     return numberRecognized
151
152 def makeBinaryImage(path):
153     image = cv2.imread(path)
154     grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
155     binaryImage = cv2.threshold(grayImage, 140, 1, cv2.THRESH_BINARY_INV)[1]
156     return binaryImage
157
158 def memoriseLargest(imageWD):
159     perimeterMax = 0
160     areaMax = 0
161     for file in os.listdir(imageWD):
162         if file != ".DS_Store":
163             binaryImage = makeBinaryImage(imageWD + file)
164             featureArray = np.array(extractFeatures(binaryImage))
165             perimeterCurrent = featureArray[1]
166             areaCurrent = featureArray[2]
167             if perimeterCurrent > perimeterMax:
168                 perimeterMax = perimeterCurrent
169             if areaCurrent > areaMax:
170                 areaMax = areaCurrent
171     return perimeterMax, areaMax
```

3.5 Resultaat

Onderstaande afbeeldingen, 3.1, 3.2.a en 3.2.b, geven het resultaat van de uitvoerende code weer.

Afbeelding 3.1 geeft aan dat de training voltooid is. Het programma geeft dan tevens de twee matrices van de weegfactoren weer. Zo is eerst de weegfactor tussen de inputlayer en de hiddenlayer te zien. Hierna is de weegfactor tussen de hidden- en de outputlayer weergegeven.

Vervolgens is in afbeelding 3.2.a en 3.2.b het resultaat van het testprogramma afgebeeld. Het programma geeft ten eerste aan welke afbeeldingen het gebruikt om te testen. De namen van deze afbeeldingen laat het programma zien in een string. Hiermee kan de gebruiker controleren of de correcte testset gebruikt is. Uiteindelijk geeft het programma per testobject weer wat de uitkomst is van het BPN.

```
BPN Training is ready!
Average number of runs: 19.0
[[ 0.7509399  0.92074779 -0.1414021 ..., -0.22311842  0.1206565
  0.09598881]
 [ 0.91962018  0.15956767  1.01896955 ...,  0.45112672  0.70110016
 -0.0596299 ]
 [ 0.14901208  0.67563397  0.54009311 ...,  1.25894429 -0.14881343
  0.88878414]
 ...,
 [ 0.72718213 -0.21554922  0.46071248 ...,  0.73124388  0.19963004
 -0.03337441]
 [ 0.79181327  1.04476221  0.45850026 ...,  0.74570659 -0.09913342
  0.50074677]
 [ 0.44189317  0.68607486  0.61915807 ..., -0.05646827  1.065467
 -0.25769617]]
[[ -0.29481964 -0.04732087 -0.05901706  0.15369637 -1.01311802  0.61084566
  0.03673494  0.14682939 -0.76557336 -0.52737817  0.58609325 -1.09419132
 -0.36200737 -0.39671862 -0.7156017  0.26655543]
 [ -0.22029692 -0.5029592  -0.43571142 -0.59595925  0.72109828 -0.81121834
 -0.42385312  0.38264106 -0.94595523 -0.0404181  -0.24503018  0.09223316
  0.51385153 -0.04677338 -0.0413189  -0.35382127]
 [ 0.32281148 -0.11517442  0.81660871  0.30883492  0.66068396  0.5447433
 -0.11307592  0.23433709 -0.39760472  0.40327909 -0.20935124  0.65493715
  0.35550985 -0.19315846  0.68018364 -0.51219019]
 [ -0.43238016  0.14071676 -0.51034401  0.29669831 -0.69290863 -0.59894382
 -0.37948834 -0.30097488 -0.63778876 -0.17015573  0.51187126 -0.76943305
  0.46325555 -0.3744929  -0.69113108  0.67059597]]

total time trained: 36 minuten en 4 seconden
```

Figuur 3.1: Resultaat van de uitvoerende code (1)

test object	output
numbersTest_17.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_31.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_11.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_5.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_28.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_32.jpg	0.0300 0.0496 0.9690 0.0300

(a)

numbersTest_36.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_2.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_34.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_14.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_29.jpg	0.0300 0.0496 0.9690 0.0300
numbersTest_20.jpg	0.0300 0.0496 0.9690 0.0300

(b)

Figuur 3.2: Resultaat van de uitvoerende code (2)

3.5.a Confusion Matrix

Onderstaand de verwarringsmatrix, afbeelding 3.3, van het Neuraal Netwerk, welke aangeeft hoe precies de bepaling is van de uitkomst.

De verwarringsmatrix is een belangrijk instrument bij Machine Learning, aangezien het een maat is voor de precisie van de gebruikte methode.

	0	1	2	3	4	5	6	7	8	9	Accuracy
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0
7	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	1.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0

Figuur 3.3: Verwarringsmatrix BPN

De verwarringsmatrix in afbeelding 3.3 dient als volgt te worden gelezen: de rij-as stelt de gewenste uitkomst voor, de kolom-as de berekende uitkomst. Wanneer deze aan elkaar gelijk zijn, zal dit terug te zien zijn op diagonaal, alles daarbuiten is een foute schatting.

In de laatste kolom, Accuracy, is af te lezen wat de accuracy per klasse is, van het neurale netwerk. Om de accuracy van het totale neurale netwerk te berekenen, hoeft men enkel een gemiddelde te nemen van de accuracies van alle klassen.

Bibliografie

- [1] Mallick, S. (2016) *Histogram of Oriented Gradients*. Geraadpleegd op 20 november 2017, van <https://www.learnopencv.com/histogram-of-oriented-gradients/>
- [2] Brownlee, J. (2016) *How to implement the backpropagation algorithm from scratch in Python*. Geraadpleegd op 11 december 2017, van <https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>