**Deep Learning EX1**

**Names:**

**Itai Danielli -**

**Maya Reiner- 200734648**

1. **Take the FizzBuzz example and make it work.**

The source code had been refactored making it usable and comprehendible.

**def** fizz\_buzz\_encode(i):

**def** fizz\_buzz\_cls(num):

**def** fizz\_buzz\_name(i, prediction):

1. **Add to the code a function that analyzes the results.**
   1. **Print the accuracy of the classifier.**

accuracy = metrics.accuracy\_score(expected\_vec, predicted\_vec)  
print(**'Accuracy : '** + str(accuracy));

* 1. **Generate a confusion matrix.**

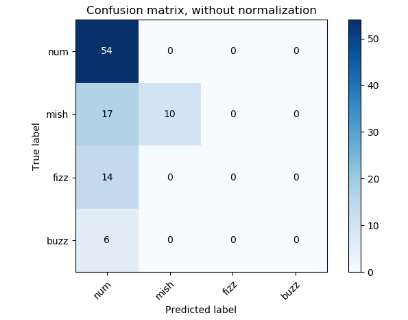
conf\_matrix = metrics.confusion\_matrix(expected\_vec, predicted\_vec)  
print(**'Confusion matrix : '** + str(conf\_matrix))  
plot(y\_true=expected\_vec, y\_pred=predicted\_vec, classes=[**"num"**, **"fizz"**, **"buzz"**, **"fizzbuzz"**])

1. **Run it once and put the results in a word file.**

These numbers are deceiving since most numbers are not fizz or buzz or fizzbuzz the classifier is more like a random classifier choosing the number most of the time.

Accuracy : 0.6336633663366337

Confusion matrix: [[54 0 0 0] [17 10 0 0] [14 0 0 0] [ 6 0 0 0]]



1. **Rerun the algorithm 10 times and see if there are differences in the accuracy.**

The accuracy changes randomly

Accuracy : 0.6534653465346535

Accuracy : 0.7326732673267327

Accuracy : 0.693069306930693

Accuracy : 0.5544554455445545

Accuracy : 0.7326732673267327

Accuracy : 0.693069306930693

Accuracy : 0.7623762376237624

Accuracy : 0.801980198019802

Accuracy : 0.693069306930693

Accuracy : 0.7326732673267327

1. **Change the algorithm to also deal with the number 2 besides the numbers 3 and 5 with all the relevant combinations (more classes2,3,5).**

**def** mish\_buzz\_encode(i):

**def** mish\_buzz\_cls(num):

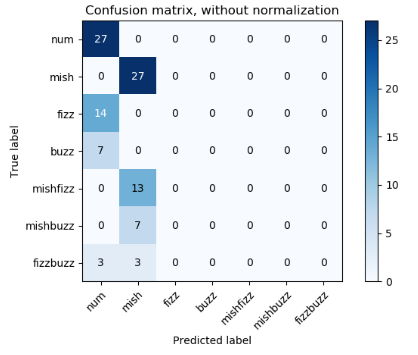
**def** mish\_buzz\_name(i, prediction):

1. **Run it once and put the results in a word file.**

Accuracy : 0.5346534653465347

Confusion matrix : [[27 0 0 0 0 0 0] [ 0 27 0 0 0 0 0] [14 0 0 0 0 0 0]

[ 7 0 0 0 0 0 0] [ 0 13 0 0 0 0 0] [ 0 7 0 0 0 0 0] [ 3 3 0 0 0 0 0]]



1. **Try to improve the results by more training, change the network etc.**

The network is increased to 5000 nodes reaching batch accuracy of about 0.88.

And the number of sample to test increased from 1000 to 128\*20=2560.

Reaching batch accuracy of nearly 1.0 and confusion matrix accuracy of 1.0:

Accuracy : 1.0

Confusion matrix :

[ [27 0 0 0 0 0 0]

[ 0 27 0 0 0 0 0]

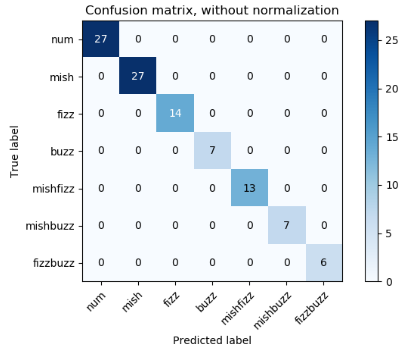
[ 0 0 14 0 0 0 0]

[ 0 0 0 7 0 0 0]

[ 0 0 0 0 13 0 0]

[ 0 0 0 0 0 7 0]

[ 0 0 0 0 0 0 6] ]



1. **Change the representation from binary to prime based(encode the numbers as prime number multiplayer). That means each number is coded by how many time each prime appears in the product. For large primes you can put them all in one bucket.**

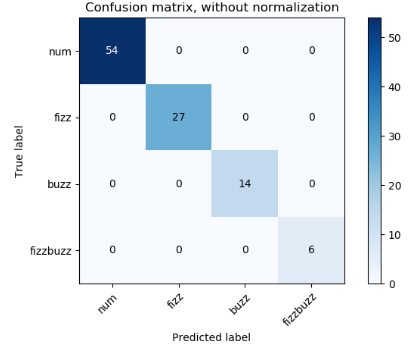
**Example: 24 is coded as 2^3\* 3^1 -🡪 [3,1,0,0,0,0…]**

**Do you think the algorithms (first and second) will work better? Run them and write the results and compare them.**

**Run fizzbuzz:**

Accuracy : 1.0

Confusion matrix: [[54 0 0 0] [27 0 0 0] [0 0 14 0] [ 6 0 0 0]]



**Run mishbuzz:**

Accuracy : 1.0

Confusion matrix :

[ [27 0 0 0 0 0 0]

[ 0 27 0 0 0 0 0]

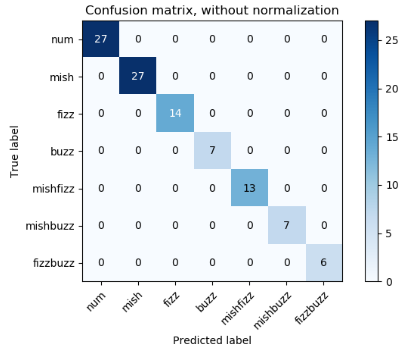
[ 0 0 14 0 0 0 0]

[ 0 0 0 7 0 0 0]

[ 0 0 0 0 13 0 0]

[ 0 1 0 0 0 6 0]

[ 0 0 0 0 0 0 6] ]



**Accuracy comparison table for first run**

|  |  |  |
| --- | --- | --- |
|  | fizzbuzz | Mishbuzz |
| Binary | 0.6336633663366337 | 0.5346534653465347 |
| prime | 1.0 | 1.0 |

Therefore ,the algorithm with prime encode work better. The accuracy is 1.0 even we decrease the number of samples to test.

Submit your code and the report.