TECHNICAL REPORT ON BANK-NOTE AUTHENTICATION DATA

ABSTRACT:

The choice of Activation Functions (AF) has proven to be an important factor that affects the performance of an Artificial Neural Network (ANN). Use a 1-hidden layer neural network model that adapts to the most suitable activation function according to the data-set. The ANN model can learn for itself the best AF to use by exploiting a flexible functional form, $k0 + k1 * x$ with parameters $k0, k1$ being learned from multiple runs.

1. Introduction

Despite a decrease in the use of currency due to the recent growth in the use of electronic transactions, cash transactions remain very important in the global market. Banknotes are used to carry out financial activities. There has been a drastic increase in the rate of fake notes in the market. Fake money is an imitation of the genuine notes and is created illegally for various motives. It is difficult for human-eye to recognize a fake note because they are created with great accuracy to look alike a genuine note. Hence, there is a dire need in banks and ATM machines to implement a system that classifies a note as genuine or fake.

2. Data set

The dataset contains 1,372 rows with 5 numeric variables. It is a classification problem with two classes (binary classification).

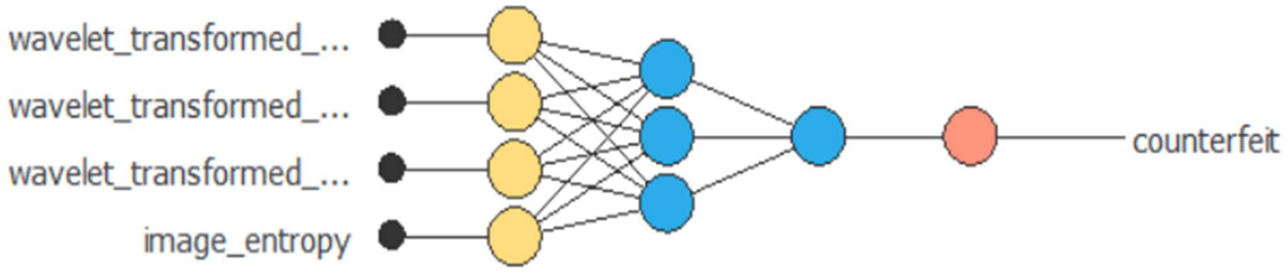
Below provides a list of the five variables in the dataset.

- variance of Wavelet Transformed image (continuous).
- skewness of Wavelet Transformed image (continuous).
- curtosis of Wavelet Transformed image (continuous).
- entropy of image (continuous).
- class (integer).

The goal is to predict whether a bank note is authentic / real / genuine or a forgery/real based on four predictor values.

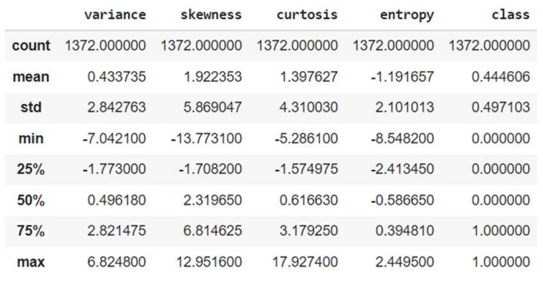
3. Neural Network

The second step is to configure a neural network to represent the classification function. The below picture shows the neural network that defines the model.

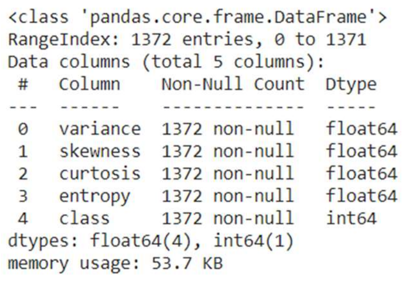


4. Data Analysis

To see the statistical details of the data, the “describe()” function can be used, getting information about the data



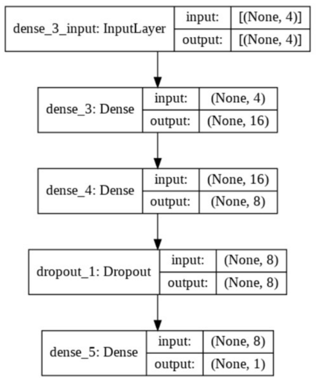
	variance	skewness	curtosis	entropy	class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

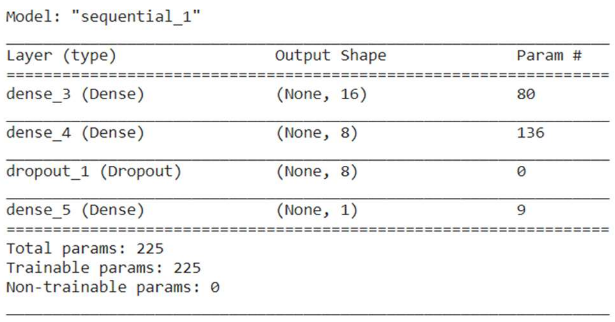


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   variance  1372 non-null   float64
 1   skewness  1372 non-null   float64
 2   curtosis  1372 non-null   float64
 3   entropy   1372 non-null   float64
 4   class     1372 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
```

5. Architecture of ANN:

The flow structure and summary of the model





```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 16)                80
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout_1 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 9
=================================================================
Total params: 225
Trainable params: 225
Non-trainable params: 0
_________________________________________________________________
```

6. Optimizers :

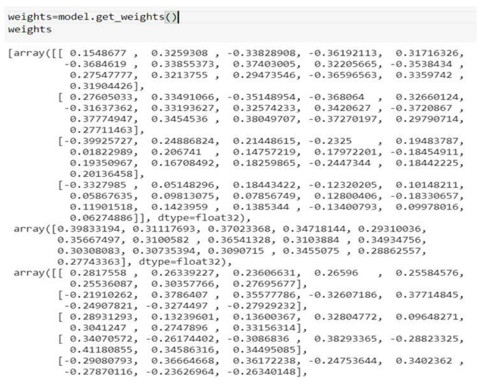
We have used “Adam” Optimizer with learning rate “0.0001” and “Binary Cross Entropy” Loss to compile the model

7. Intial parameter values:

➔Input layer = 4*16= 64+16 bias = 80 ➔Between hidden layer = 16*8=128 +8 bias = 136

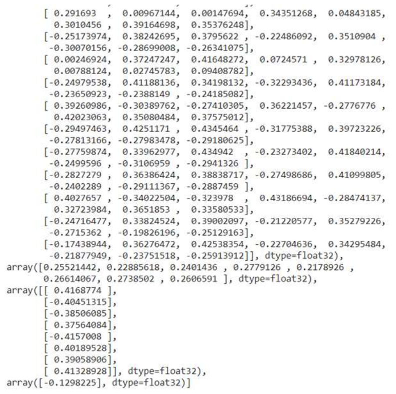
➔Between output layer = 8 * 1 = 8 + 1 bias = 9 ➔Intial k_0, k_1 values are 16, 8 , 1

8. Final parameter value:



```
weights=model.get_weights()
weights
```

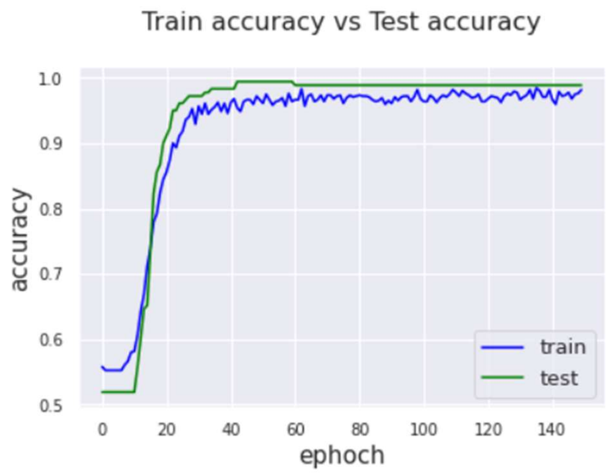
```
[array([[ 0.1548677 ,  0.3259308 , -0.33828908, -0.36192113,  0.31716326,
         -0.3684619 ,  0.33855373,  0.37403005,  0.32205665, -0.3538434 ,
          0.27547777,  0.3213755 ,  0.29473546, -0.36596563,  0.3359742 ,
          0.31904426],
        [ 0.27605033,  0.33491066, -0.35148954, -0.368064  ,  0.32660124,
         -0.31637362,  0.33193627,  0.32574233,  0.3420627 , -0.3720867 ,
          0.37774947,  0.3454536 ,  0.38049707, -0.37270197,  0.29790714,
          0.27711463],
        [-0.39925727,  0.24886824,  0.21448615, -0.2325    ,  0.19483787,
          0.01822989,  0.206741  ,  0.14757219,  0.17972201, -0.18454911,
          0.19350967,  0.16708492,  0.18259865, -0.2447344 ,  0.18442225,
          0.20136458],
        [-0.3327985 ,  0.05148296,  0.18443422, -0.12320205,  0.10148211,
          0.05867635,  0.09813075,  0.07856749,  0.12800406, -0.18330657,
          0.11901518,  0.1423959 ,  0.1385344 , -0.13400793,  0.09978016,
          0.06274886]], dtype=float32),
 array([0.39833194, 0.31117693, 0.37023368, 0.34718144, 0.29310036,
        0.35667497, 0.3100582 , 0.36541328, 0.3103884 , 0.34934756,
        0.30308083, 0.30735394, 0.3090715 , 0.3455075 , 0.28862557,
        0.27743363], dtype=float32),
 array([[ 0.2817558 ,  0.26339227,  0.23606631,  0.26596   ,  0.25584576,
          0.25536087,  0.30357766,  0.27695677],
        [-0.21910262,  0.3786407 ,  0.35577786, -0.32607186,  0.37714845,
         -0.24907821, -0.3274497 , -0.27929232],
        [ 0.28931293,  0.13239601,  0.13600367,  0.32804772,  0.09648271,
          0.3041247 ,  0.2747896 ,  0.33156314],
        [ 0.34070572, -0.26174402, -0.3086836 ,  0.38293365, -0.28823325,
          0.41180855,  0.34586316,  0.34495085],
        [-0.29080793,  0.36664668,  0.36172238, -0.24753644,  0.3402362 ,
         -0.27870116, -0.23626964, -0.26340148],
```

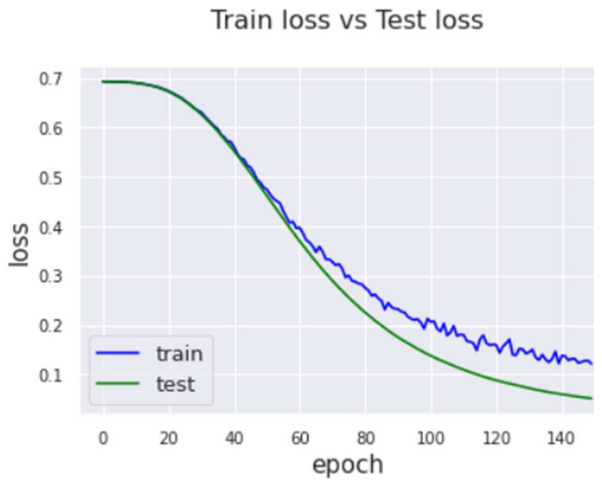


```
        [ 0.291693  ,  0.00967144,  0.00147694,  0.34351268,  0.04843185,
          0.3010456 ,  0.39164698,  0.35376248],
        [-0.25173974,  0.38242695,  0.3795622 , -0.22486092,  0.3510904 ,
         -0.30070156, -0.28699008, -0.26341075],
        [ 0.00246924,  0.37247247,  0.41648272,  0.0724571 ,  0.32978126,
          0.00788124,  0.02745783,  0.09408782],
        [-0.24979538,  0.41188136,  0.34198132, -0.32293436,  0.41173184,
         -0.23650923, -0.2388149 , -0.24185082],
        [ 0.39260986, -0.30389762, -0.27410305,  0.36221457, -0.2776776 ,
          0.42023063,  0.35080484,  0.37575012],
        [-0.29497463,  0.4251171 ,  0.4345464 , -0.31775388,  0.39723226,
         -0.27813166, -0.27983478, -0.29180625],
        [-0.27759874,  0.33962977,  0.434942  , -0.23273402,  0.41840214,
         -0.2499596 , -0.3106959 , -0.2941326 ],
        [-0.2827279 ,  0.36386424,  0.38838717, -0.27498686,  0.41099805,
         -0.2402289 , -0.29111367, -0.2887459 ],
        [ 0.4027657 , -0.34022504, -0.323978  ,  0.43186694, -0.28474137,
          0.32723984,  0.3651853 ,  0.33580533],
        [-0.24716477,  0.33824524,  0.39002097, -0.21220577,  0.35279226,
         -0.2715362 , -0.19826196, -0.25129163],
        [-0.17438944,  0.36276472,  0.42538354, -0.22704636,  0.34295484,
         -0.21877949, -0.23751518, -0.25913912]], dtype=float32),
 array([0.25521442, 0.22885618, 0.2401436 , 0.2779126 , 0.2178926 ,
        0.26614067, 0.2738502 , 0.2606591 ], dtype=float32),
 array([[ 0.4168774 ],
        [-0.40451315],
        [-0.38506085],
        [ 0.37564084],
        [-0.4157008 ],
        [ 0.40189528],
        [ 0.39058906],
        [ 0.41328928]], dtype=float32),
 array([-0.1298225], dtype=float32)]
```

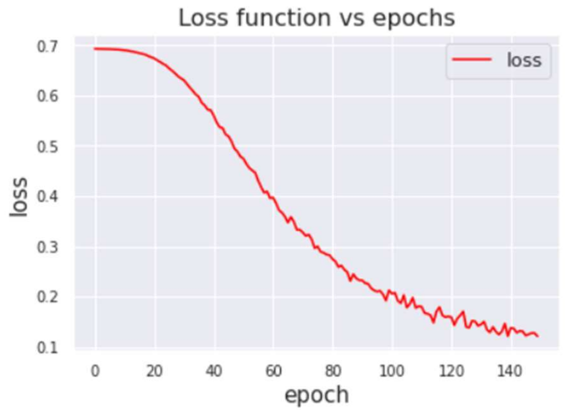
Final parameters values at the end of the training are -0.1298225

9. Train and Test Accuracy , Train and Test Loss:

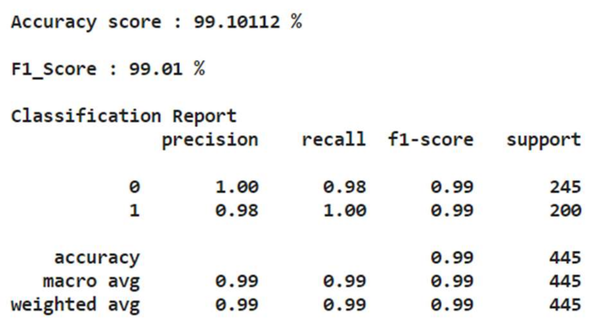




10. Loss function and epochs:



11. F1_Score, Classification Report and Confusion Matrix:



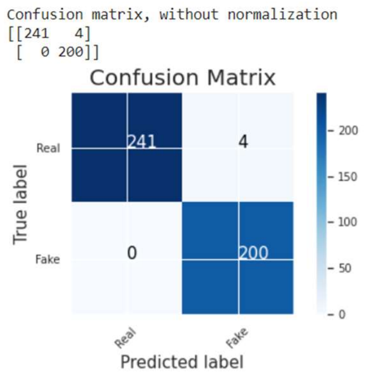
```
Accuracy score : 99.10112 %

F1_Score : 99.01 %

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       245
           1       0.98      1.00      0.99       200

    accuracy                           0.99       445
   macro avg       0.99      0.99      0.99       445
weighted avg       0.99      0.99      0.99       445
```



```
Confusion matrix, without normalization
[[241   4]
 [  0 200]]
```

12. Conclusion:

Banknote authentication is an important task. It is difficult to manually detect fake bank notes. In this report we explained how we solved the problem of banknote authentication using neural network. We conclude that Artificial neural network is the best algorithm for banknote authentication with an accuracy of 99.1%

GitHub: https://github.com/arr9666/Bank_note