Technical Report on Bank Note Authentication

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.

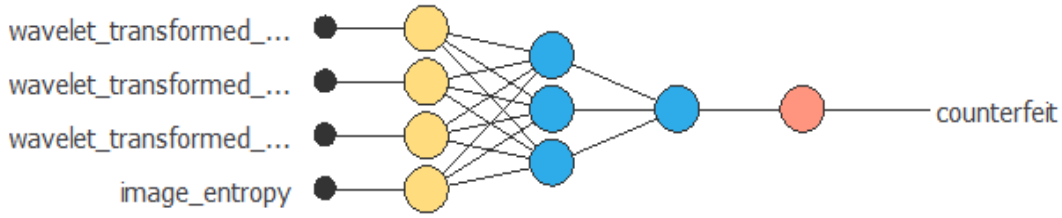
1. **Application type :** This is a classification project, since the variable to be predicted is binary (fraudulent or legal). The goal here is to model the probability that a banknote is fraudulent, as a function of its features.

2. **Data set** : The data file banknote_authentication.csv is the source of information for the classification problem. The number of instances (rows) in the data set is 1372, and the number of variables (columns) is 5.

   In that way, this problem has the following variables:

   - variance, used as input. (Continuous)
   - skewness, used as input. (Continues)
   - curtosis, used as input. (Continuous)
   - entropy, used as input. (Continuous)
   - class, used as the target. It can only have two values: 0 (non-counterfeit) or 1 (counterfeit).

3. **Neural Network**: The second step is to configure a neural network to represent the classification function. The next picture shows the neural network that defines the model



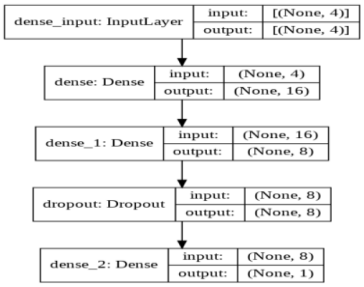
4. **Data Preprocessing :** Statistical Analysis done in data set

	variance	skewness	curtosis	entropy	class
count	1348.000000	1348.000000	1348.000000	1348.000000	1348.000000
mean	0.445785	1.909039	1.413578	-1.168712	0.452522
std	2.862906	5.868600	4.328365	2.085877	0.497925
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.786650	-1.627000	-1.545600	-2.393100	0.000000
50%	0.518735	2.334150	0.605495	-0.578890	0.000000
75%	2.853250	6.796025	3.199800	0.403863	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

5. **Train and Test :** The data will divided into training and testing

```
# split into train and test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
# determine the number of input features
n_features = X.shape[1]
```

6. **Architecture of ANN** : This is architecture of model and summary




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                80
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout (Dropout)            (None, 8)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 225
Trainable params: 225
Non-trainable params: 0
```

7. **Optimizers :**

We have used “Adam” as Optimizer and “Binary Cross Entropy” Loss to compile the model . We have choosed the dam Optimizer because, it proves that it is best among the adaptive optimizers in most of the cases.

8. **Model Training/Evaluation :**

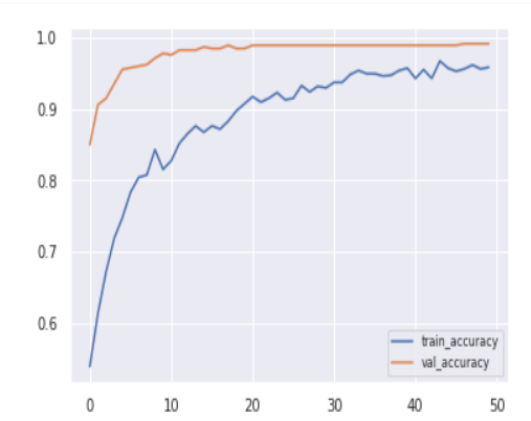
```
Epoch 46/50
29/29 [==============================] - 0s 3ms/step - loss: 0.1161 - accuracy: 0.9553 - val_loss: 0.0163 - val_accuracy: 0.9888
Epoch 47/50
29/29 [==============================] - 0s 2ms/step - loss: 0.1010 - accuracy: 0.9569 - val_loss: 0.0154 - val_accuracy: 0.9910
Epoch 48/50
29/29 [==============================] - 0s 3ms/step - loss: 0.1001 - accuracy: 0.9630 - val_loss: 0.0149 - val_accuracy: 0.9910

14/14 [==============================] - 0s 3ms/step - loss: 0.0142 - accuracy: 0.9910
[0.014227494597434998, 0.9910112619400024]
```

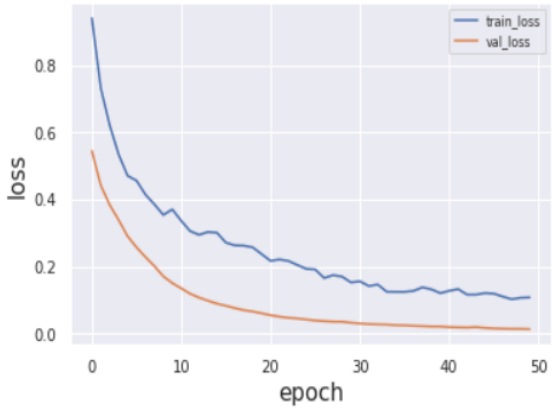
9. **Final Parameter Values :**

```
[array([[-0.38938534,  0.77810967,  0.11944607,  0.5408176 ,  0.02201378,
         0.4085316 ,  0.49519655, -0.5236578 ,  0.24977097, -0.3989526 ,
        -0.0457873 , -0.8060377 , -0.5822191 ,  0.4611979 , -0.2613468 ,
         0.37451366],
        [-0.46149492, -0.46388716,  0.31769505,  0.34448248,  0.49466264,
        -0.21985638, -0.06789729,  0.1326401 ,  0.692466  ,  0.09814979,
        -0.41810793, -0.51917255, -0.00400952, -0.22139588,  0.27198482,
        -0.3311997 ],
        [ 0.32555184,  0.2387872 , -0.1383937 ,  0.02527408,  0.08527688,
         0.12785998,  0.1971719 , -0.21089824,  0.76404047, -0.22387847,
        -0.05375976, -0.29284558, -0.640266  , -0.48563036, -0.4714024 ,
         0.23842964],
        [ 0.17309198, -0.42904493,  0.20556256, -0.12678759,  0.08780822,
        -0.15144292,  0.02326566, -0.1200237 ,  0.08747981, -0.04798467,
         0.24520326,  0.460424  , -0.02077629, -0.07706224,  0.28170255,
        -0.13320638]], dtype=float32),
 array([ 0.18086895,  0.03944197, -0.01002894, -0.0537216 , -0.07428209,
        -0.09548197, -0.10497155,  0.07148936,  0.0101041 ,  0.43991885,
        -0.01306288,  0.57837194,  0.4325271 ,  0.22553398, -0.03524149,
        -0.03094647], dtype=float32),
 array([[-0.20516686,  0.53531396,  0.54977006, -0.06689514, -0.17985831,
         0.11816873, -0.14962606, -0.20154843],
        [ 0.19175991,  0.08038948, -0.55061966, -0.5112084 ,  0.18958738,
         0.19278036,  0.44267863,  0.14903666],
        [-0.02760437, -0.51731116,  0.1957547 ,  0.22828352,  0.05243278,
         0.5776226 , -0.3028046 ,  0.51967055],
        [ 0.40482664, -0.04210953, -0.21939094, -0.30535832, -0.02832645,
         0.2555174 ,  0.32411587,  0.18580122],
        [ 0.144419  , -0.3899573 , -0.09190665,  0.28820428,  0.51198715,
        -0.12506083, -0.3650422 ,  0.371023  ],
        [ 0.49949864, -0.62564677, -0.5495821 ,  0.08735015,  0.19757022,
         0.24577127, -0.28153545,  0.46973276],
        [ 0.3525546 , -0.86303747,  0.14731456,  0.32917184,  0.425752  ,
         0.21944138, -0.15218179,  0.7088508 ],
        [ 0.3525546 , -0.86303747,  0.14731456,  0.32917184,  0.425752  ,
         0.21944138, -0.15218179,  0.7088508 ],
        [ 0.09884027,  0.4855416 ,  0.07984734, -0.07076473, -0.4782806 ,
        -0.19276935,  0.2337762 , -0.45671383],
        [ 0.38659433, -0.1291307 , -0.52680624, -0.33734548,  0.26795965,
         0.40452012,  0.5343974 ,  0.5941377 ],
        [-0.23963872,  0.5138383 ,  0.52518815, -0.07297705,  0.04870244,
        -0.00179575, -0.17579536,  0.17390807],
        [ 0.19013807,  0.3168372 ,  0.4731765 ,  0.2998414 , -0.23579833,
        -0.31901565, -0.04083973,  0.358435  ],
        [-0.21065554,  1.144284  ,  0.6897057 , -0.10052156, -0.07175042,
        -0.7504418 , -0.470353  , -0.3588449 ],
        [-0.27954435,  0.5857546 ,  0.8754167 , -0.17588767, -0.6138409 ,
        -0.36646542, -0.39717296, -0.2810864 ],
        [-0.18170482,  0.776181  ,  0.49451247,  0.12326659, -0.2063457 ,
        -0.530569  ,  0.28581464, -0.03125415],
        [-0.20113994,  0.29705188, -0.1778978 , -0.08751747,  0.38416415,
         0.01275051,  0.24055822, -0.63394386],
        [-0.08286774,  0.06659429,  0.18664102,  0.1694199 ,  0.10247317,
         0.15239893,  0.3333171 , -0.20495692]], dtype=float32),
 array([-0.10869835,  0.47665665,  0.380506  , -0.11564545, -0.09990919,
         0.00891381, -0.0685266 , -0.20250764], dtype=float32),
 array([[-1.0199943 ],
        [ 0.31642383],
        [ 0.46506423],
        [-0.41764772],
        [-1.1059995 ],
        [-1.0622793 ],
        [-0.76917464],
        [-0.51676995]], dtype=float32),
 array([0.78438276], dtype=float32)]
```

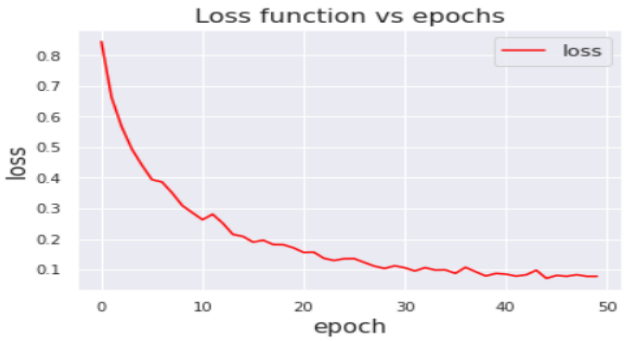
10.Train and Test Accuracy :



11. Train and Test Loss:



12. Loss Function vs Ephocs :



13. Classification Report / F1 Score :

Test Accuracy score : 99.10112 %

F1_Score : 99.034 %

	precision	recall	f1-score	support
0	1.00	0.99	0.99	239
1	0.99	1.00	0.99	206
accuracy			0.99	445
macro avg	0.99	0.99	0.99	445
weighted avg	0.99	0.99	0.99	445

14.Train Classification/F1 score report:

Train Accuracy score : 99.77852 %

F1_Score : 99.752 %

	precision	recall	f1-score	support
0	1.00	1.00	1.00	499
1	1.00	1.00	1.00	404
accuracy			1.00	903
macro avg	1.00	1.00	1.00	903
weighted avg	1.00	1.00	1.00	903

15. Conclusion :

Bank Note Authentication is an important task. It is very difficult to manually detect fake bank notes. Artificial Neural Networks can help in this regard. In this report, we explained how we solve the bank note Authentication using ANN. The ANN is best method with accuracy of 99.1%.

Github Link : https://github.com/ETTIKALABHAVANA/Bank-Note-Authentication-Data