

Bank Note Authentication

“a small Effort to Stop black money”

Vividh Talesara | Big Data Systems Intelligence & Analytics | 02/02/2018

# Abstract

There are Different Techniques is conducting Data Analysis that range from clustering, prediction and Classification. These Techniques are applied using learning Algorithms such as SVM (Support Vector Machine), Neural Networks. While training the model, the choice of algorithm is very important to reach best accuracy rate and correct solution. In this Project I have worked on a Classification problem to identify Forged Bank Notes from Original one. Whole world is facing the problem of Black Money that is one of big reasons for growing corruption and terrorism so rectifying fake notes and stopping them from being readily usable is the motive for this project. I have used Tensorflow and Random Forest to Predict the accuracy of my modeled training set. Fortunately, could get accuracy score which fluctuated between 0.99 and 1.00, i.e. 100% accuracy score.

# Introduction

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images. The dataset was taken from the [UCI Repository](https://archive.ics.uci.edu/ml/datasets/banknote+authentication). It contains 1,372 observations (banknotes) and 4 attributes.

• Dataset Characteristics – Multivariate • Number of Instances - 1372

• Attributes Characteristics – Real • Number of Attributes - 5

• Date Donated - 2013/04/16 • Missing Values - None

• Variance of Wavelet Transformed image – Continuous

• Skewness of Wavelet Transformed image - Continuous

• Curtosis of Wavelet Transformed image – Continuous

• Entropy of image – Continuous

• Class - Integer

**Note:** Data were numeric values extracted from Wavelet Transformed Images (WTIs) of banknote specimens.

| **Column** | **Definition** |
| --- | --- |
| Image.Var | Variance of the WTI |
| Image.Skew | Skewness of the WTI |
| Image.Curt | Curtosis of the WTI |
| Entropy | Entropy of the image |
| Class | Authenticity of the banknote |

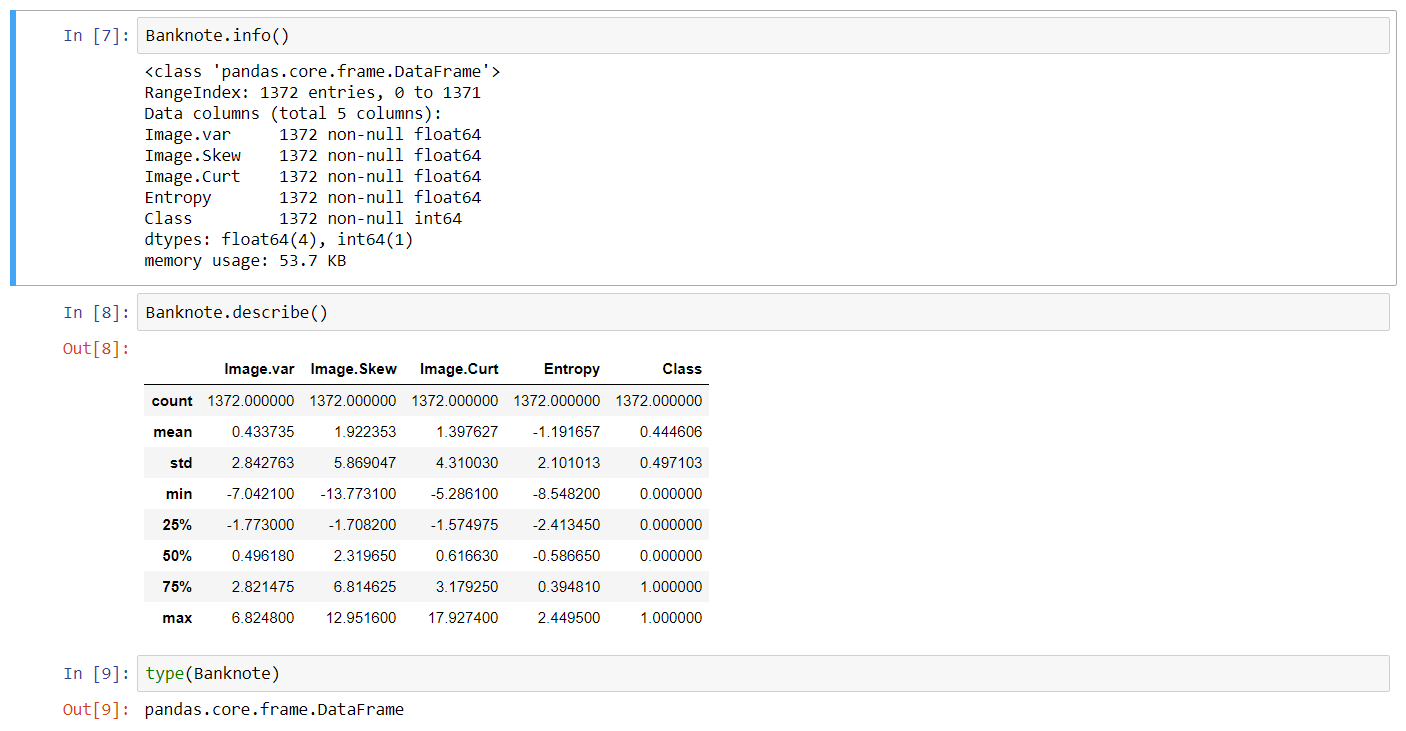
# Keywords

• Numpy • Pandas • Scikit-learn • Tensorflow • Multilayer Perceptron • Standard Scaler • RELU • Cross Entropy • Adam Optimizer • Epoch • Random Forest Classifier • Confusion Matrix • Classification Report

# Code with Documentation

# Importing Dataset, Libraries and using Info() and Describe() to know more about data.



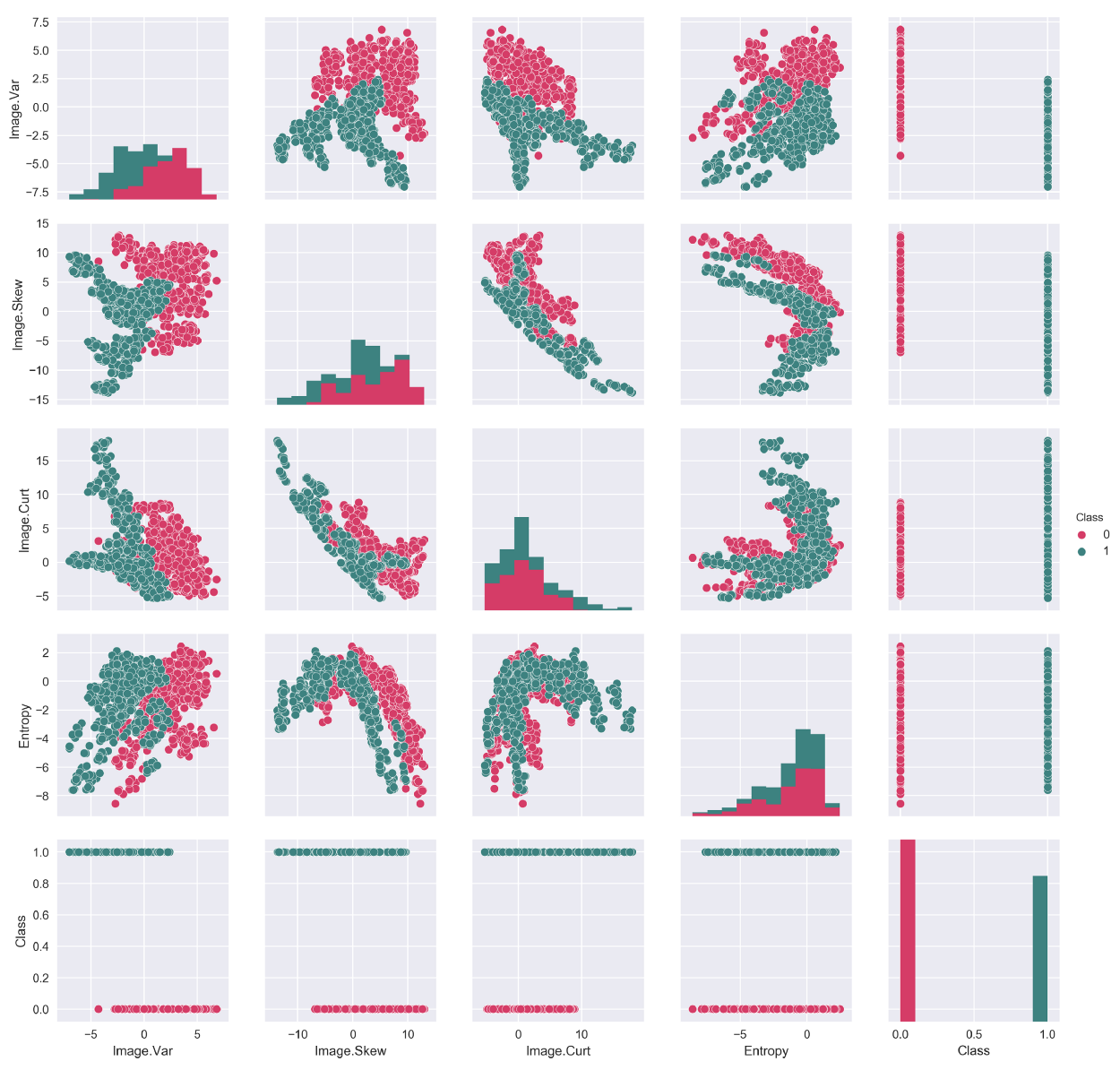


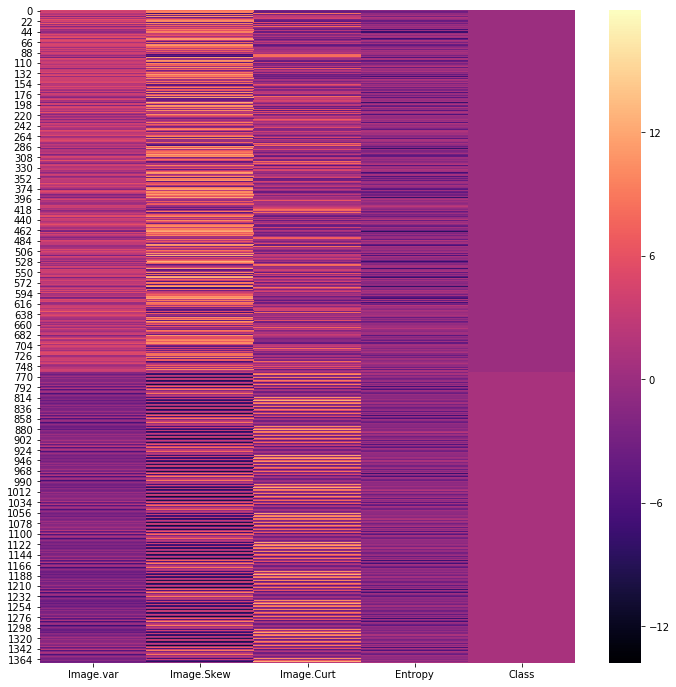
# Exploring Dataset, Plotting and Visualizing Graphs.

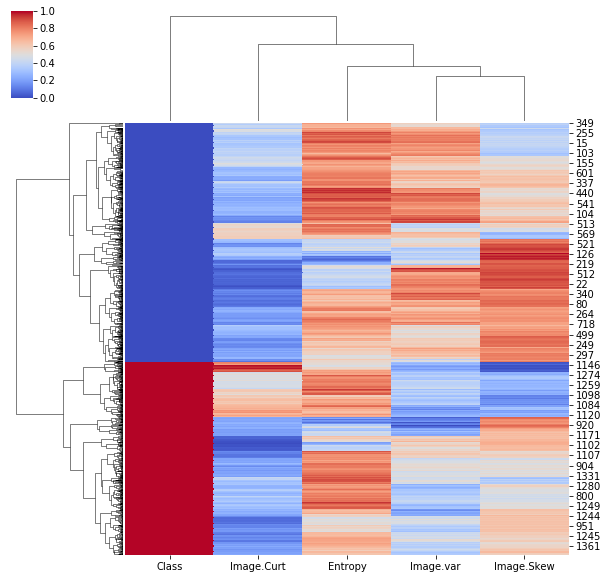
It seems like we have a lot more fake banknote in our dataset. Around 40% of our notes are fake.



Next, let’s try to find relationships between the other attributes in our dataset (in relation to our target class). We can use a [Pairplot](http://seaborn.pydata.org/generated/seaborn.pairplot.html" \t "_blank) from Seaborn, with the hue set to the Class attribute. This way we can easily see how the relationships differ between real and fake banknotes.



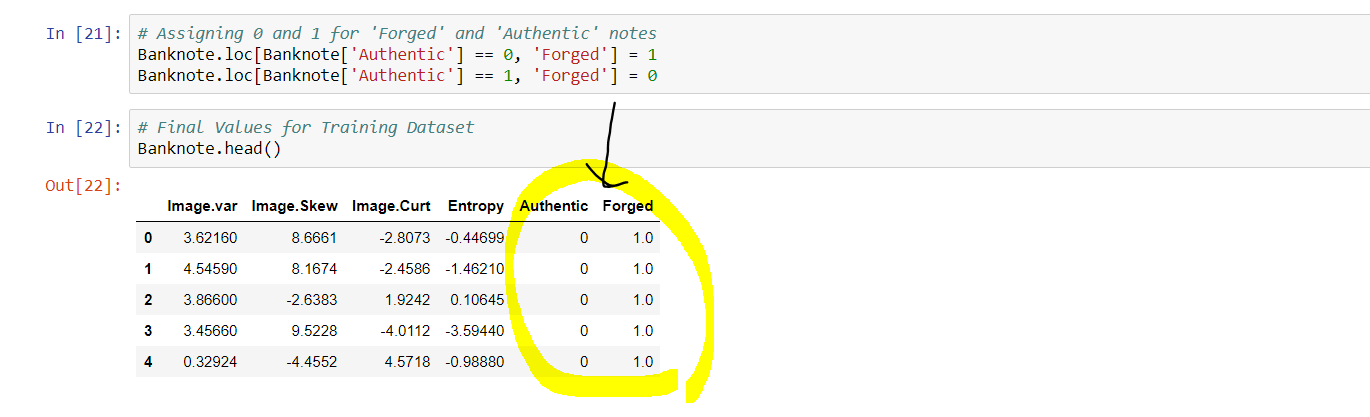


Checking the clustering or Pairing of all attributes/features if they match each other.

# Data Preparation, Testing and Training Data.

We don’t need to standardize the Class attribute, so I created a separate data frame to store the other features.



Next, I tried to fit a [StandardScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) object from the Scikit-learn library on the independent variables and store the transformed data in a new data frame called scaled features.

Also, since we have 2 classes (authentic and forged) for our dependent variable, we can separate these into two different columns. Let’s rename Class to Authentic, and create a new Forged column.

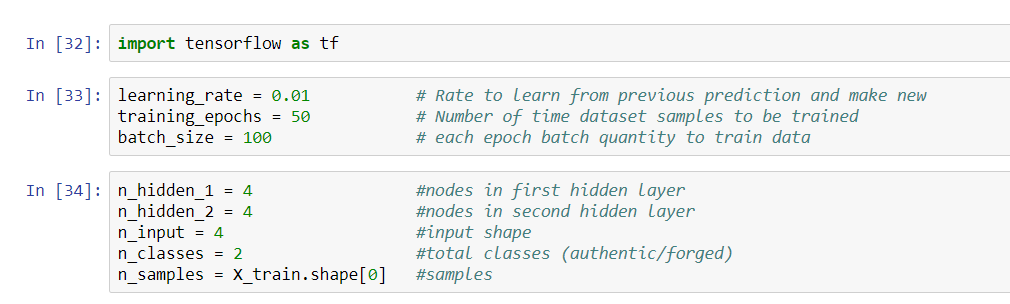


Our X will be the scaled features, and our y will be both the Authentic and Forged attributes. Since [Numpy](http://www.numpy.org/) arrays are compatible with Tensor Flow, we can convert X and y into Numpy arrays using as matrix().

I have used Scikit-learn’s [train test split](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) to split our data into a training and a test set. We will use 20% of the original dataset for testing.

# Parameters like Hidden Layers, Learning Rate, Epoch, Batch Size.

Before setting up our neural network, it is important to define the parameters for our model. We may want to adjust these a bit later on, depending on how our model performs. Let’s first set the learning rate, the number of training epochs, and the batch size.



The **learning rate** of the model is a value between 0 and 1. It can be thought of as a measure of how quickly our model abandons old beliefs for new ones. A high rate means that the network changes its mind more quickly, and a lower rate means that it is reluctant to change. Here we will choose a learning rate of 0.01.

**One epoch means one pass of the training set.** We want our model to go through the training set more than just once, to improve accuracy. However, it is important to note that a very high number of epochs results in the risk of overfitting. Overfitting reduces the performance of our neural net on unseen data. Let’s set the number of training epochs to 100.

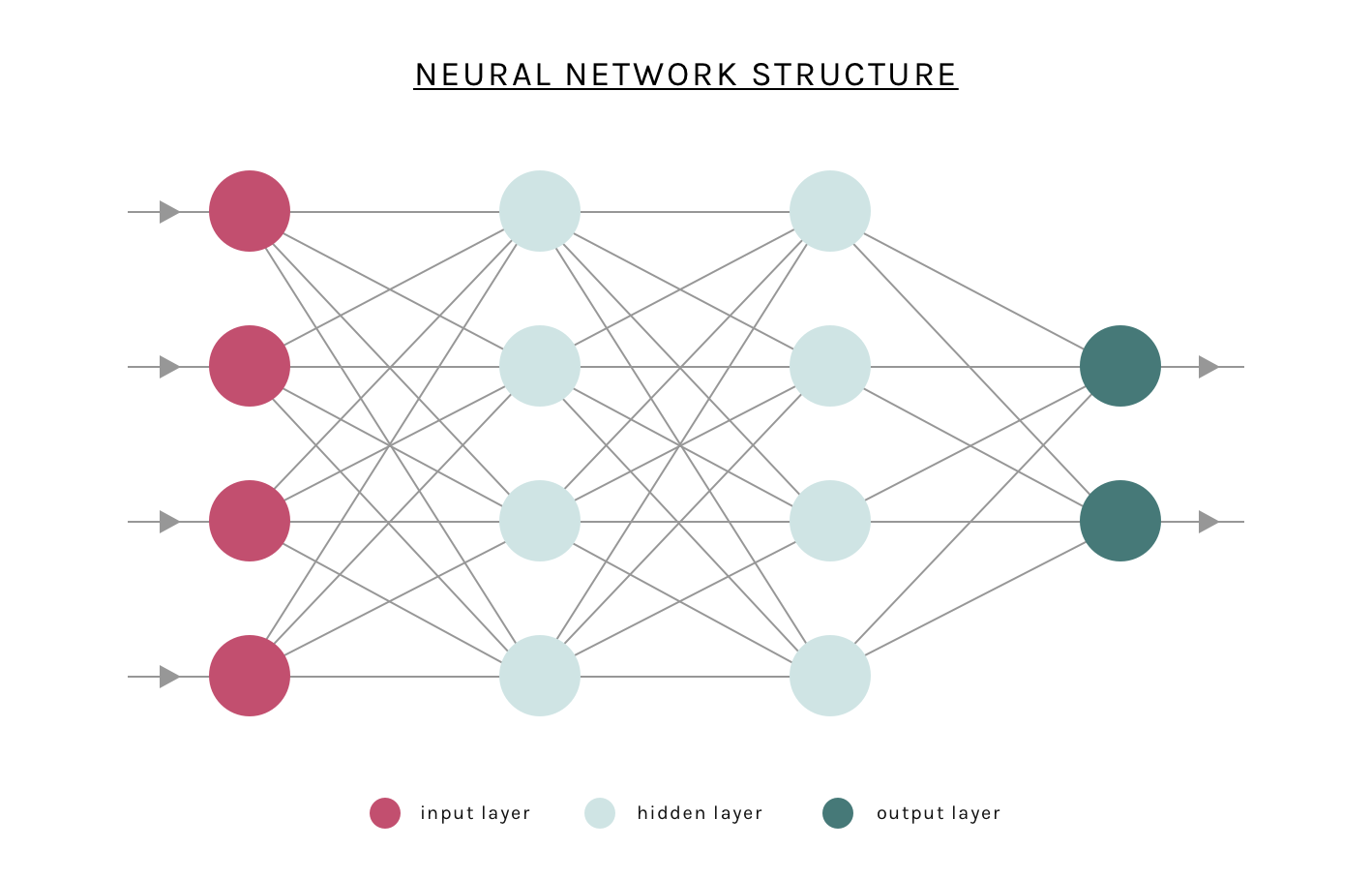
Finally, we can set the **batch size** to 100. We will be using batch learning, and a batch size of 100 means that we will update our weights using back-propagation after every 100 predictions.



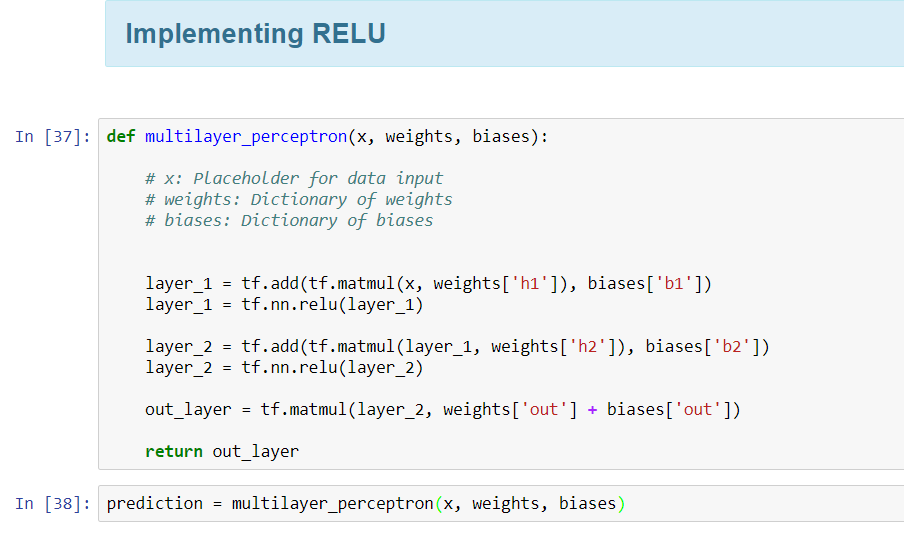
we need to define the weights and bias for each layer in our network. We will create dictionaries of weights and biases using the parameters we’ve already defined.

Our network will have **3 layers** (2 hidden layers and an output layer, excluding the input layer).

# Tensorflow, weights & biases.

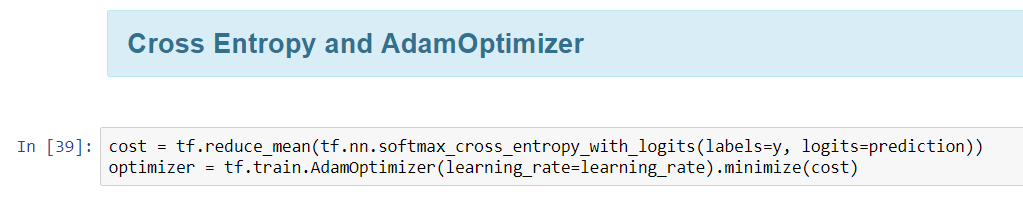


We can set the predictions to be a tensor called prediction, and it will contain the output from our neural network.



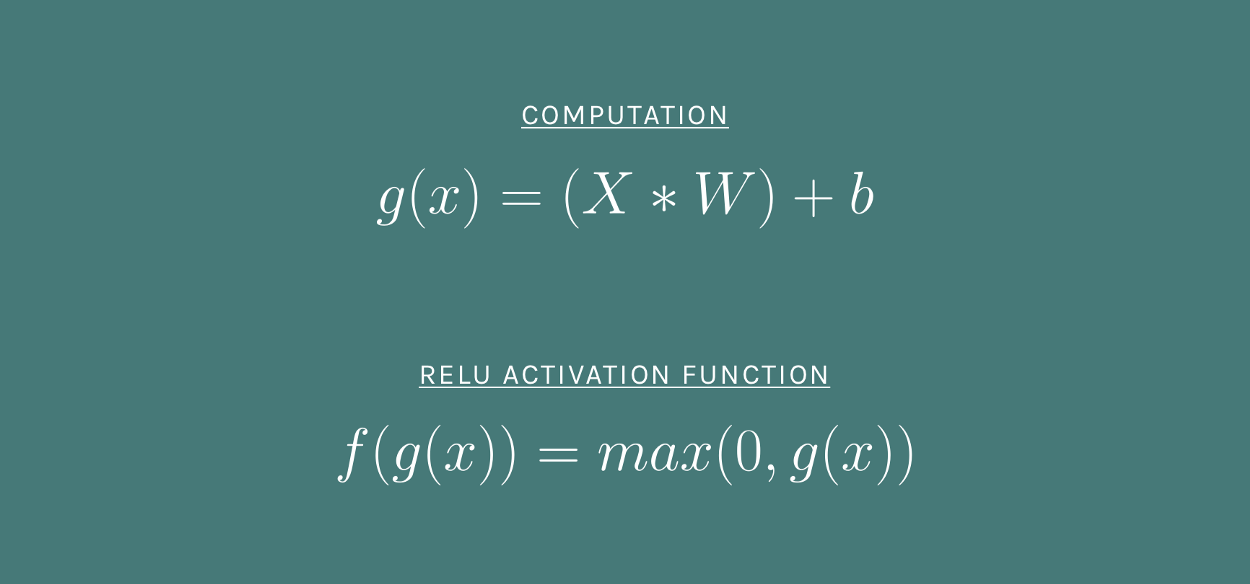
# Cost & Optimization.

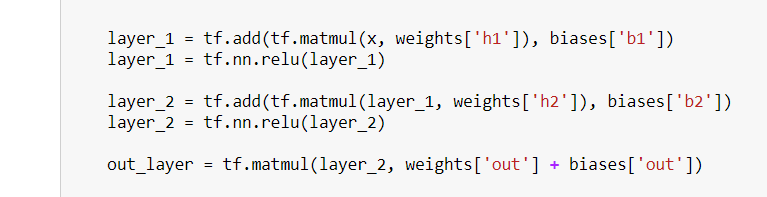
We will be using a [softmax cross-entropy](http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/) function for calculating the loss, and the [Adam Optimiser](https://arxiv.org/abs/1412.6980) to minimise cost.



# Constructing Neural Network.

We will create a function that accepts the input x, a dictionary of weights, and a dictionary of biases. Let’s use the [ReLU](http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf" \t "_blank) activation function for each layer.





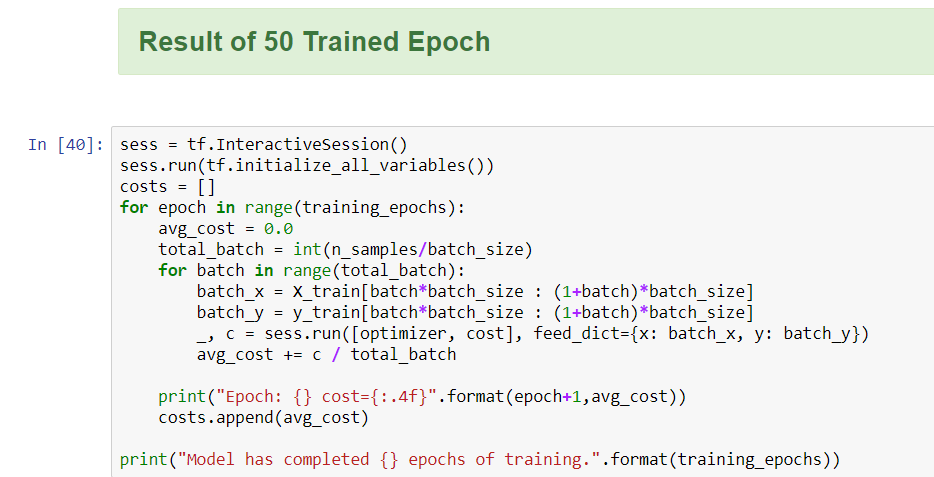
In Tensor Flow, graphs aren’t executed unless a [Session](https://www.tensorflow.org/api_docs/python/tf/Session) is created and run. The session allocates resources for the graph, and holds the actual values of intermediate results and variables.

We will have two loops:

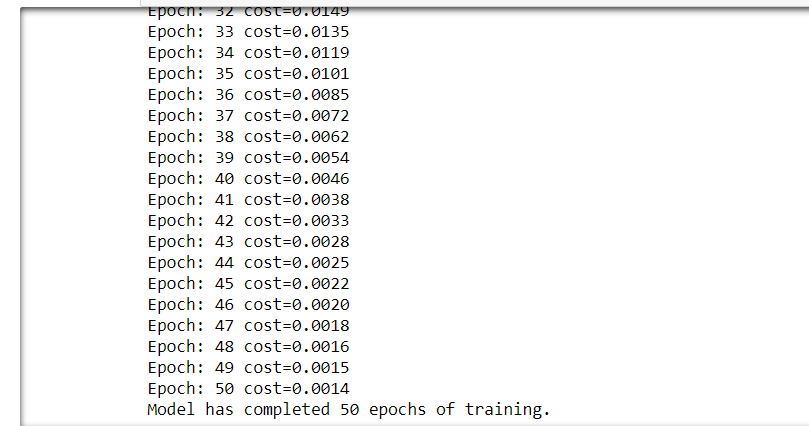
1. The outer loop runs the epochs, and
2. The inner loop runs the batches for each epoch.

After each epoch, we can print out the cost and append it to a list of costs. The way we can plot a line graph after training to visualise how our cost has been minimized.

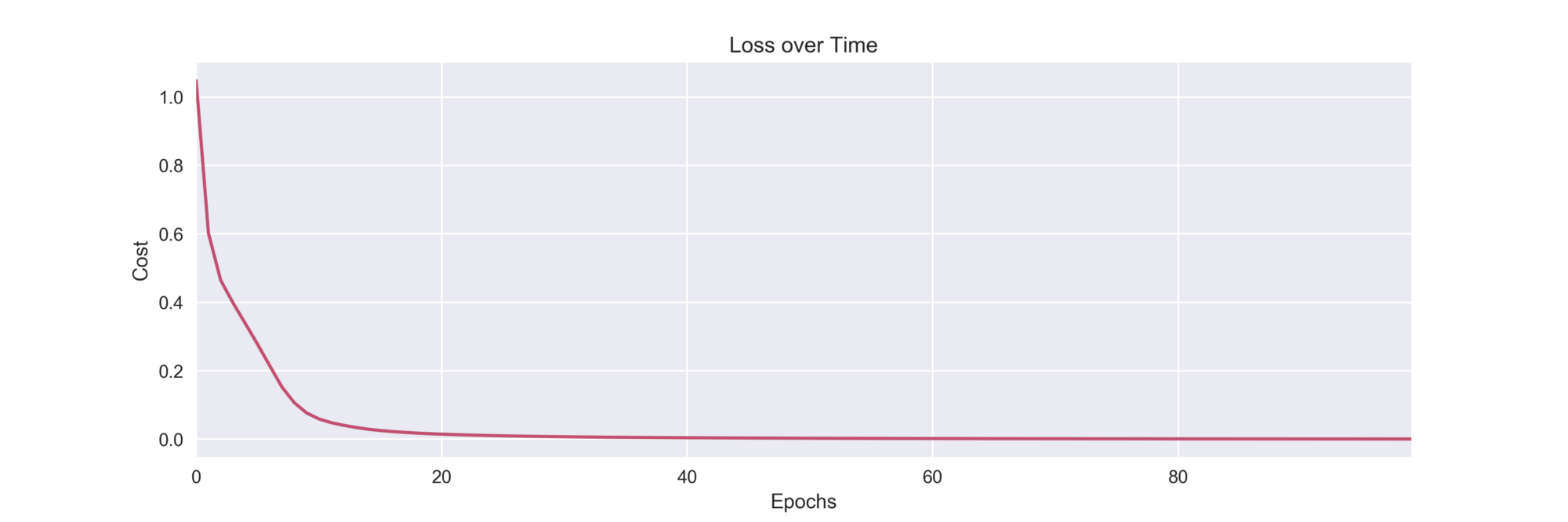
# Training Network (Epoch).



Here’s the output:



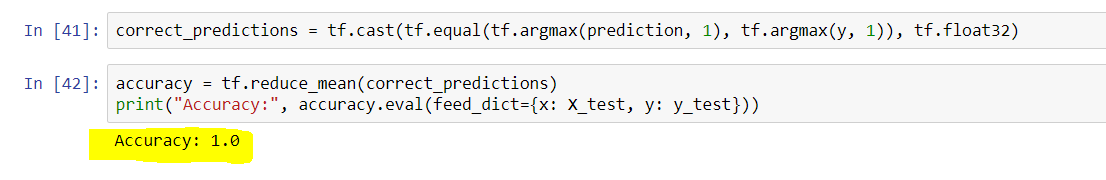
Below is a graph of the cost over time, created using the list of costs.



# Model Evaluation.

Our model has now been trained. To see how well it performs on the test set, I will count the number of correct predictions on the test set. We can then define the accuracy as the mean percentage of correct predictions.

**correct\_predictions = tf.cast(tf.equal(tf.argmax(preds, 1), tf.argmax(y, 1)), tf.float32)**

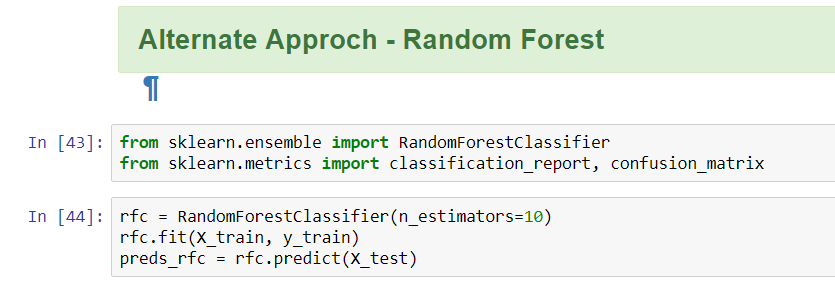


To get the accuracy, we have to use the .eval() method and pass in a dictionary for the placeholder’s x and y.

**our model has achieved a 100% accuracy with the test set.**Probably the dataset was a little easy for our model to classify.

# Comparing Random Forest with Tensorflow

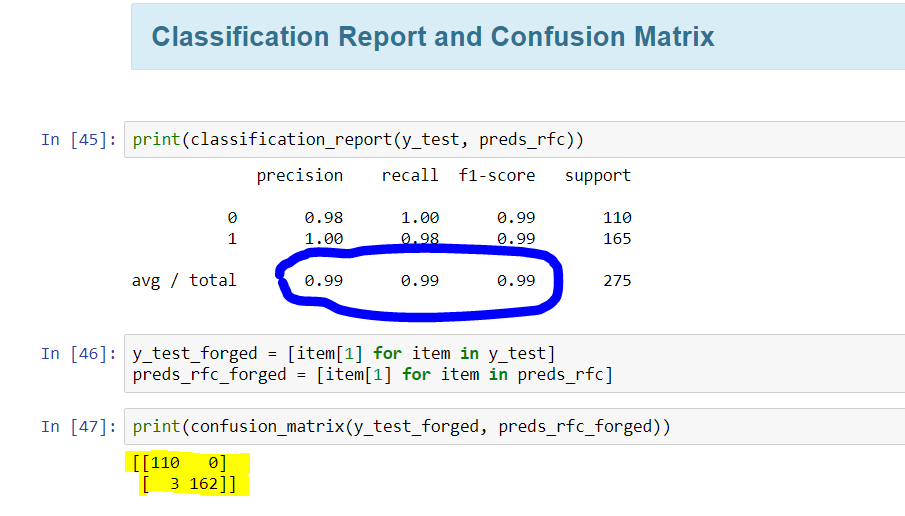
I will use a random forest classifier. Let’s train it on the same dataset, and store the predictions in a separate dataframe called preds\_rfc.



# The random forest classifier was able to achieve a 99% accuracy (only 1% lower than our neural net), so it’s safe to conclude that our dataset was probably just easy to classify.

# Results

I have evaluated our predictions using a classification report and a confusion matrix.



# Conclusion

In this Project, we have assessed the Performance of Classifications and Neural Network Algorithm. The major goal was to get the at most accuracy as possible. Fortunately, the dataset was easy and getting 100% accuracy was not that though task. We have used Random Forest and Tensorflow and found the original notes can be easily separated from fake or forged notes. In future, we can implement this algorithm with some other data and try checking the authenticity of out model.

# References

* 1. <http://www.ijesi.org/papers/Vol(5)2/H052062070.pdf>
  2. <https://github.com/vivianrajkumar/banknote-auth>
  3. <https://medium.com/tensorist/detecting-fake-banknotes-using-tensorflow-be21ffd2c478>
  4. <https://archive.ics.uci.edu/ml/datasets/banknote+authentication>