

Report On : Project - 1.1 Software 1.0 Versus Software 2.0

Neeraj Ajit Abhyankar

UB Person No. : 50290958

UBID : nabyank

Objective :

The project is to compare two problem solving approaches to software development :

- 1) The logic-based approach (Software 1.0)
- 2) The machine learning approach (Software 2.0).

It is also designed to quickly gain familiarity with Python and machine learning frameworks.

Task :

We consider the task of FizzBuzz. In this task an integer divisible by 3 is printed as *Fizz*, and integer divisible by 5 is printed as *Buzz*. An integer divisible by both 3 and 5 is printed as *FizzBuzz*. If an integer is not divisible by 3 or 5 or 15, it simply prints the input as output (for this last case, the input number can be classified as Other in Software 2.0, it should then be handled using Software 1.0, which prints the input as output).

Software 1.0 :

Logic Explanation :

The Software 1.0 part is a Python code and is simply based on the working of modulus function.

a) Say we have a number which is divisible by 3, e.g. $9 / 3 = 3$. Since 9 is completely divisible by 3 modulo returns 0(Fizz) as output else returns the remainder. Say for $10 / 3 = 1$ modulo will return 1 as the output. Similarly for a number divisible by 5 output = 0(Buzz) else the remainder.

b) Also for the numbers divisible by 3 & 5 respectively modulo will return 0(FizzBuzz) else remainder is displayed. The remainder can be also written as 'Other' for simplification.

Software 2.0 :

The Software 2.0 is more of a machine learning approach which is comparatively more complex. In this approach we train the model for learning with the help of a training set and test the learning accuracy of the model on the testing set.

Steps in Software 2.0 :

1) First the output from Software 1.0 is recorded into the Testing.csv file. Also Since the input data has some traits the input data is encoded and processed in the array of range of 10.

2) The testing data is placed in a .csv file in a grid pattern as inputs and labels. Inputs from 0-101 with the appropriate labels as 'Fizz' 'Buzz' 'FizzBuzz' 'Other' are imported in the datafile. Also a training set is defined from inputs 101-1000 and the same labels as testing data in .csv file.

- 3) Next we define the model for the training data set where we define the input size, drop-out, definition of the layers to be involved and the neural network layers, the operations and transformations to be performed on them.
- 4) The next Step involves the creation of the training and testing datafiles for recording the data for learning purpose. Then the training model is created and is run using the parameters such as validation data split, number of epochs, batch size etc for processing and training of the model.
- 5) The model is then decoded from the encoded datafile and graphs are plotted for various factors, which in turn decide the accuracy of the model.
- 6) The end output of the model is then recorded in an output.csv file which is compared with the testing datafile which contains the original python based outputs of the FizzBuzz program for accuracy with number of wrong answers to number of right answers.

Key Concepts :

- 1) pandas :** pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.
- 2) DataFrames :** DataFrames in Python come with the Pandas library, and they are defined as a two-dimensional labeled data structures with columns of potentially different types. The Pandas DataFrame consists of three main components: the data, the index, and the columns.
- 3) Encoding of Data :** Machine learning algorithms cannot work with categorical data directly. Categorical data must be converted to numbers and that's why there is need to encode the data before processing. The encoder is responsible for reading the source document and encoding it to an internal representation. An approach to encoding categorical values is to use a technique called label encoding. Label encoding is simply converting each value in a column to a number.
- 4) Decoding of Data :** In Machine Learning the encoded data has to be decoded first so we use a decoder. The decoder is a language model responsible for generating each word in the output summary using the encoded representation of the source document.
- 5) Training Dataset :** The model is initially fit on a training dataset, that is a set of examples used to fit the parameters (e.g. weights of connections between neurons in artificial neural networks) of the model. In practice, the training dataset often consist of pairs of an input vector and the corresponding answer vector or scalar, which is commonly denoted as the target. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation.
- 6) Testing Dataset :** The testing dataset is used to provide an unbiased evaluation of a final model fit on the training dataset. A test dataset is a dataset that is independent of the training dataset, but that follows the same probability distribution as the training dataset. A test set is therefore a set of examples used only to assess the performance (i.e. generalization) of a fully specified classifier.

7) .csv File : CSV is a simple file format used to store tabular data, such as a spreadsheet or database. Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or OpenOffice Calc. CSV stands for "comma-separated values". Its data fields are most often separated, or delimited, by a comma. For example, let's say you had a spreadsheet containing the following data.

Key Imports :

1) import pandas as pd - Imports panda methods as pd since python built-in methods can overlap panda methods.

2) from keras.utils import np_utils - Import numpy utils from keras.utils to transform data into numerical sequences.

3) from keras.models import Sequential - Import the sequential layered model from keras.

4) from keras.layers import Dense, Activation, Dropout - Import layered models from keras.

5) from keras.callbacks import EarlyStopping, TensorBoard - Import callback functions from keras. Stop training when a monitored quantity has stopped improving is early Stopping & Visualization tool for dynamic graphs is tensorboard.

6) from keras.utils import np_utils - Import numpy utils from keras.utils

7) import numpy as np - Import numpy library as np

Hyper - Parameter Changes :

Parameter Set 1 :

validation_data_split = 0.2 ; num_epochs = 10000 ; model_batch_size = 128 ; tb_batch_size = 32 ; early_patience = 100 ; optimizer = rmsprop ; first layer activation = relu ; second layer activation = softmax ; Errors : 8 ; Correct : 92 ; Testing Accuracy : 92.0

a) The Parameters Used :

```

Fizzling and Buzzing TensorFlow X Fizzling and Buzzing Keras X +
localhost:8889/notebooks/Desktop/Fall 2018/ML/Projects/Project 1.1/Given Code/Fizzling and Buzzing Keras.ipynb 80% ...
jupyter Fizzling and Buzzing Keras Last Checkpoint: 2 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Model

In [97]: validation_data_split = 0.2
num_epochs = 10000
model_batch_size = 128
tb_batch_size = 32
early_patience = 100

tensorboard_cb = TensorBoard(log_dir='logs', batch_size=tb_batch_size, write_graph=True)
earlystopping_cb = EarlyStopping(monitor='val_loss', verbose=1, patience=early_patience, mode='min')

# Read Dataset
dataset = pd.read_csv('training.csv')

# Process Dataset
processedData, processedLabel = processData(dataset)
history = model.fit(processedData
                    , processedLabel
                    , validation_split=validation_data_split
                    , epochs=num_epochs
                    , batch_size=model_batch_size
                    , callbacks=[tensorboard_cb, earlystopping_cb]
                    )

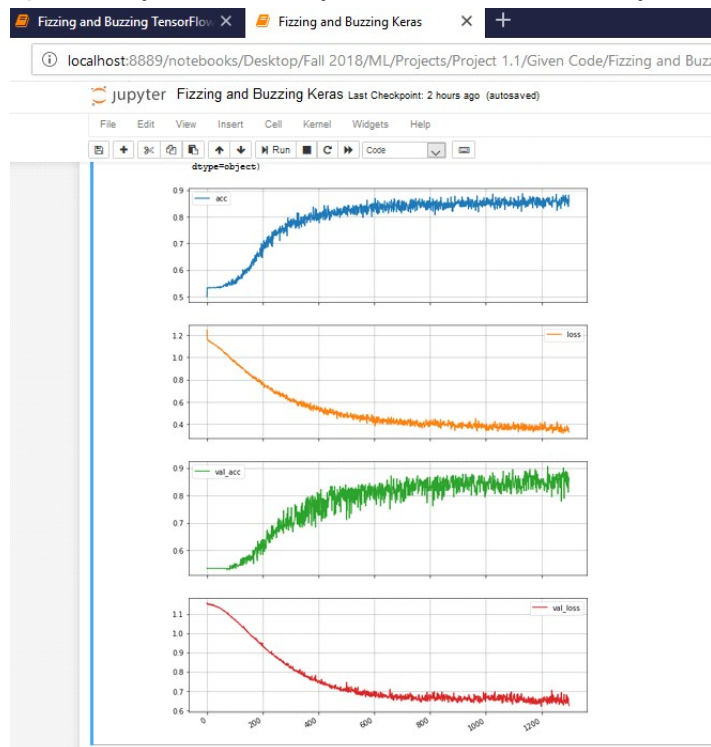
Train on 720 samples, validate on 180 samples
Epoch 1/10000
720/720 [=====] - 0s 375us/step - loss: 1.2448 - acc: 0.4986 - val_loss: 1.1568 - val_
acc: 0.5333
Epoch 2/10000
720/720 [=====] - 0s 49us/step - loss: 1.1657 - acc: 0.5333 - val_loss: 1.1526 - val_
acc: 0.5333
Epoch 3/10000
720/720 [=====] - 0s 28us/step - loss: 1.1574 - acc: 0.5333 - val_loss: 1.1516 - val_
acc: 0.5333
Epoch 4/10000
720/720 [=====] - 0s 28us/step - loss: 1.1521 - acc: 0.5333 - val_loss: 1.1482 - val_
acc: 0.5333
Epoch 5/10000
720/720 [=====] - 0s 28us/step - loss: 1.1517 - acc: 0.5333 - val_loss: 1.1484 - val_
acc: 0.5333

```

Due to early stopping implementation the epochs stop early and not all 10,000 are executed.

Also the accuracy gradually improves as the number of epochs and the trained number of inputs increase.

b) The Graphs of the Output based on the use of optimizer, loss and metrics :



c) The Accuracy of the Algorithm :

```
output["label"] = testDataLabel
output["predicted_label"] = predictedTestLabel

opdf = pd.DataFrame(output)
opdf.to_csv('output.csv')

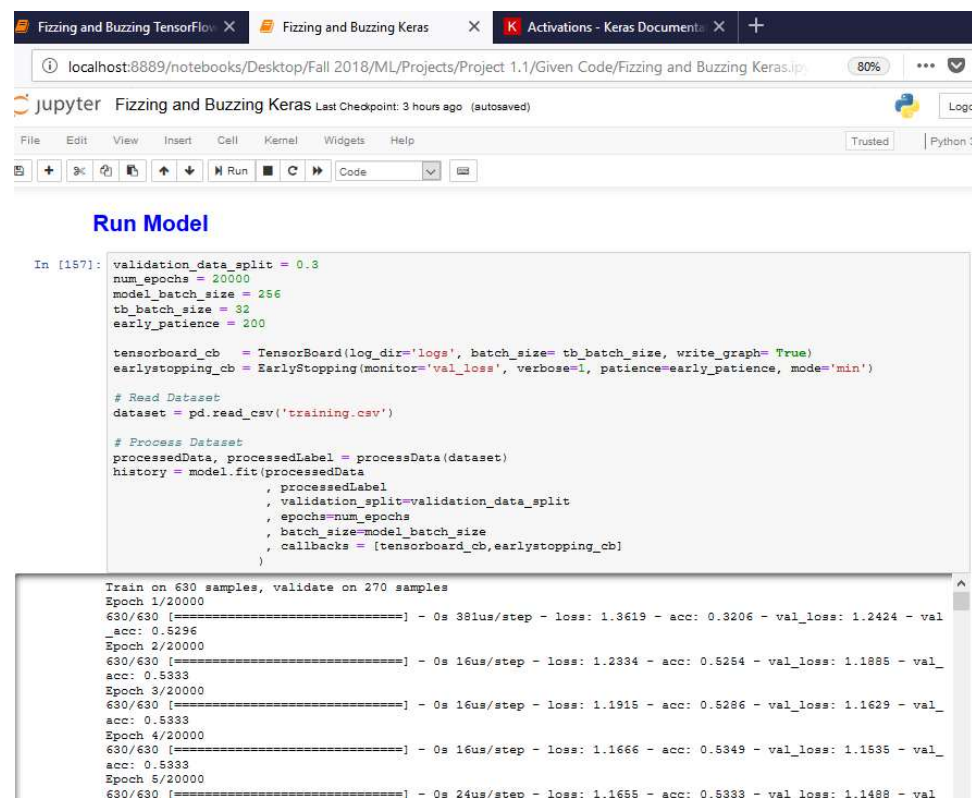
Errors: 8 Correct :92
Testing Accuracy: 92.0
```

In []:

Parameter Set 2 :

validation_data_split = 0.3 ; num_epochs = 20000 ; model_batch_size = 256 tb_batch_size = 32 ;
early_patience = 200 ; drop_out = 0.3 ; optimizer = rmsprop ; first layer activation = relu ; second
layer activation = softmax ; Errors: 19 ; Correct :81 ; Testing Accuracy: 81.0

a) The Parameters Used :



```
validation_data_split = 0.3
num_epochs = 20000
model_batch_size = 256
tb_batch_size = 32
early_patience = 200

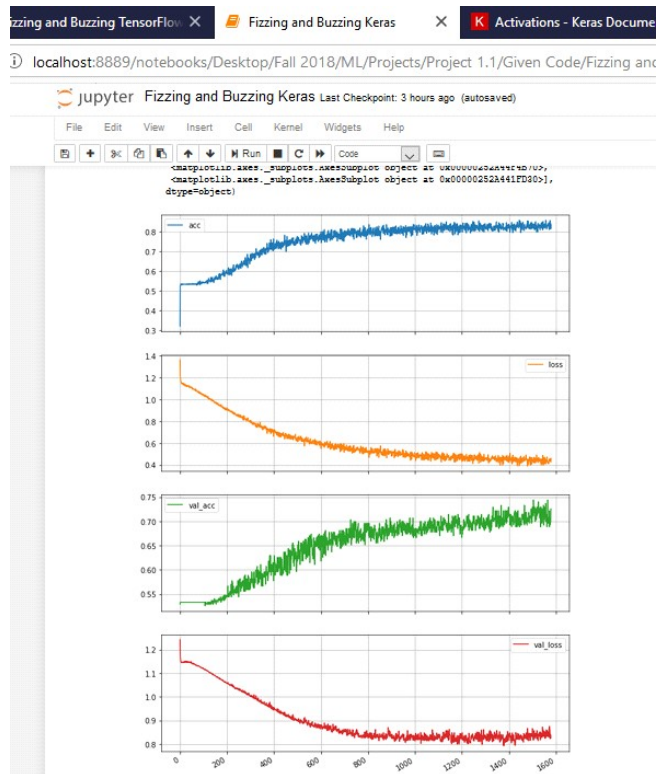
tensorboard_cb = TensorBoard(log_dir='logs', batch_size=tb_batch_size, write_graph=True)
earlystopping_cb = EarlyStopping(monitor='val_loss', verbose=1, patience=early_patience, mode='min')

# Read Dataset
dataset = pd.read_csv('training.csv')

# Process Dataset
processedData, processedLabel = processData(dataset)
history = model.fit(processedData
                    , processedLabel
                    , validation_split=validation_data_split
                    , epochs=num_epochs
                    , batch_size=model_batch_size
                    , callbacks = [tensorboard_cb,earlystopping_cb]
                    )

Train on 630 samples, validate on 270 samples
Epoch 1/20000
630/630 [=====] - 0s 381us/step - loss: 1.3619 - acc: 0.3206 - val_loss: 1.2424 - val_
_acc: 0.5296
Epoch 2/20000
630/630 [=====] - 0s 16us/step - loss: 1.2334 - acc: 0.5254 - val_loss: 1.1885 - val_
acc: 0.5333
Epoch 3/20000
630/630 [=====] - 0s 16us/step - loss: 1.1915 - acc: 0.5286 - val_loss: 1.1629 - val_
acc: 0.5333
Epoch 4/20000
630/630 [=====] - 0s 16us/step - loss: 1.1666 - acc: 0.5349 - val_loss: 1.1535 - val_
acc: 0.5333
Epoch 5/20000
630/630 [=====] - 0s 24us/step - loss: 1.1655 - acc: 0.5333 - val_loss: 1.1488 - val_
```

b) The Graphs of the Output based on the use of optimizer, loss and metrics :



c) The Accuracy of the Algorithm :

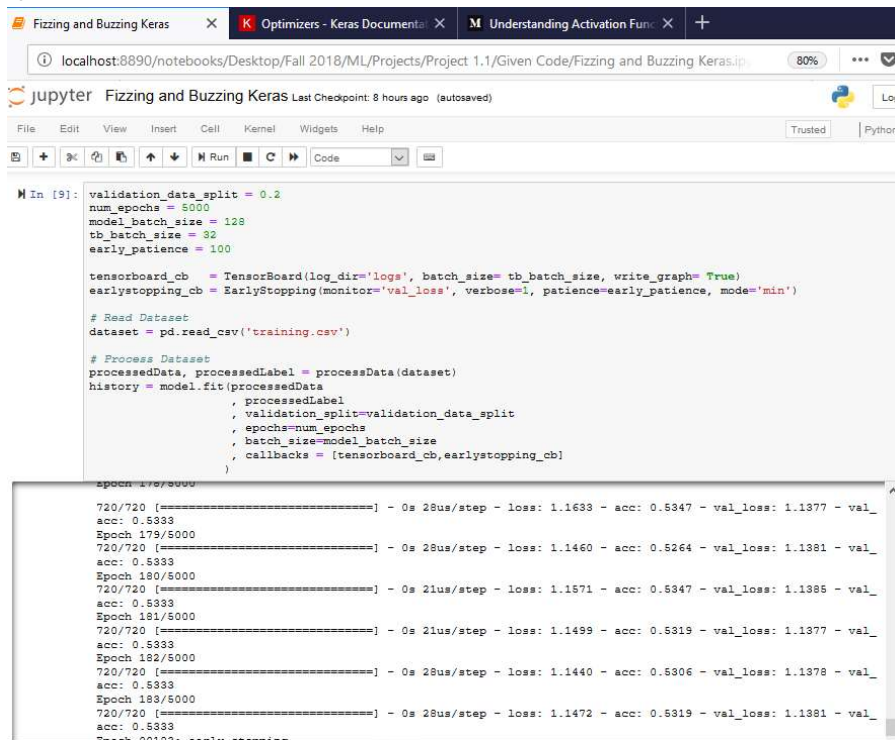
```
output["label"] = testDataLabel
output["predicted_label"] = predictedTestLabel

opdf = pd.DataFrame(output)
opdf.to_csv('output.csv')
```

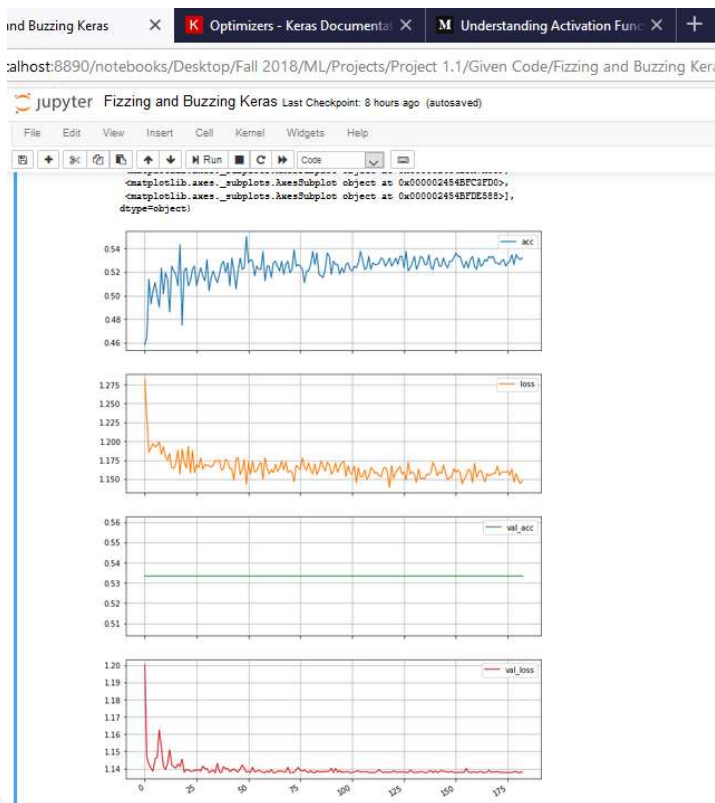
Errors: 19 Correct :81
Testing Accuracy: 81.0

Parameter Set 3 :

validation_data_split = 0.2 ; num_epochs = 10000 ; model_batch_size = 128 ; tb_batch_size = 32 ; early_patience = 100 optimizer = adagrad ; first layer activation = sigmoid ; second layer activation = softmax ; Errors: 47 ; Correct : 53 ; Testing Accuracy : 53.0



b) The Graphs of the Output based on the use of optimizer, loss and metrics :



c) The Accuracy of the Algorithm :

```
output = {}
output["input"] = testDataInput
output["label"] = testDataLabel

output["predicted_label"] = predictedTestLabel

opdf = pd.DataFrame(output) #print the data frame into output file.
opdf.to_csv('output.csv') #convert the output file to .csv file.

Errors: 47 Correct :53
Testing Accuracy: 53.0
```

Graph Analysis :

1) For the Parameter Set 1 the Accuracy of the model is the highest i.e. 92.0, the data split is 0.2 with number of epochs = 10,000. Due to early stopping not all the epochs are executed. The model optimizer used is rmsprop which is a good choice for recurrent neural networks. In general rmsprop Divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. Also this is one of the ways to speed up mini batch learning. The first layer of network includes relu – it turns the negative value to 0, the scope of relu is from 0 to infinity. The second layer activation includes softmax – which gives a non-linear variant of multinomial logistic regression.

2) For the Parameter Set 2 the Accuracy of the model is 81.0, the data split is 0.3 with the epochs = 20,000. The drop out for this is increased to 0.3 with model batch size = 128. Also we can see a significant drop in the accuracy with varying parameters from the first Parameter Set. Selecting the right hyper-parameters contributes largely to the accuracy of the model. Also the behavior of the activation functions of the first and second layer changes with respect to change in parameters, which can result in the change of accuracy or errors the model can face while execution.

3) For the Parameter Set 3 the Accuracy of the model is 53.0, the data split is 0.2 with the epochs = 10,000. The drop out was again set to 0.2, but the optimizer was changed from rmsprop to adagrad. The optimizer adagrad adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. Also the First Layer Activation was changed to sigmoid which maps arbitrary real values back to the range [0, 1]. The larger the value, the closer to 1 you'll get to no error state. Also the second layer was activated with the softmax function, this resulted in higher step loss, lower accuracy and higher val_loss too.

Conclusion :

It is observed that there are changes in the output of the learning model based on changes in hyper-parameters and changes in the accuracy due to the changes in the optimizer and loss function respectively. Also the graphs of the models show that there are significant changes in the output with change in drop-out rate.

The Software 1.0 is a simple logic based Python code which calculates the output using modulus function which is fairly simple and accurate.

The Software 2.0 is the machine learning approach to the same problem which predicts the output based on the training of the model with complex algorithms and processes.

Thus, Project 1.1 compares the Software 1.0 Vs Software 2.0 to get a fair idea of the working principles of both and the logic behind them.

References :

<http://pandas.pydata.org/pandas-docs/stable/>

<https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python>

<http://pbpython.com/categorical-encoding.html>

<https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>

https://en.wikipedia.org/wiki/Training,_test,_and_validation_sets

<https://machinelearningmastery.com/encoder-decoder-models-text-summarization-keras/>

<https://www.computerhope.com/issues/ch001356.htm>

<https://keras.io/getting-started/sequential-model-guide/>

<https://keras.io/layers/core/>

<https://keras.io/callbacks/>

https://faroit.github.io/keras-docs/1.2.1/utils/np_utils/

<http://ruder.io/optimizing-gradient-descent/index.html#adagrad>

<https://nathanbrixius.wordpress.com/2016/06/04/functions-i-have-known-logit-and-sigmoid/>

<https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/>

<http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/>

<https://jovianlin.io/cat-crossentropy-vs-sparse-cat-crossentropy/>

<https://jovianlin.io/cat-crossentropy-vs-sparse-cat-crossentropy/>

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

<https://stackoverflow.com/questions/35697404/why-its-a-convention-to-import-pandas-as-pd>

<https://github.com/keras-team/keras/issues/597>

<https://stats.stackexchange.com/questions/231061/how-to-use-early-stopping-properly-for-training-deep-neural-network>

<https://stackoverflow.com/questions/2672936/multiple-counters-in-a-single-for-loop-python>

*The above references are for the report and the comments to the code file where noting reference links was not possible.