CT475 – Machine Learning & Data Mining

Assignment 3 – Artificial Neural Network

Name: *Cian Duffy*
Course: *Electronic and Computer Engineering (4BP)*
Student ID: *13399326*

Introduction

Artificial Neural Networks (ANNs) are computational models based on the structure and on the structure and function of biological neurons found in mammalian central nervous systems. The network's structure consists of highly-interconnected processing elements (units). The neurons (connections between each unit) can be updated adjusted systematically based on the input and output values, making ANNs very well suited to supervised learning problems.

Artificial Neural Network Structure

The units in an ANN are arranged in various layers. There are three types of layer: input, hidden and output.

The input layer is where the data is fed into the ANN from the outside world. The network will try to learn about this data. Every ANN has exactly one input layer. The input layer has one unit for each feature in the training data.

The output layer is where the prediction of the ANN appears. Its size depends on the application for which the ANN is being used. If the network is performing a regression task, then the output layer will have exactly one unit which will return the numerical value predicted by the network. If the network is being used for a classification task, then there will be either be exactly one output unit or else one output unit for each class label.

The layers between the input and output layers are known as hidden layers. ANNs can have multiple hidden layers, however there is no real consensus on the optimal number of hidden layers. However, there is widespread agreement that there are relatively few situations in which performance is significantly improved with additional hidden layers. For most applications, a single layer is sufficient as additional layers add complexity without significantly improving the accuracy of the network.

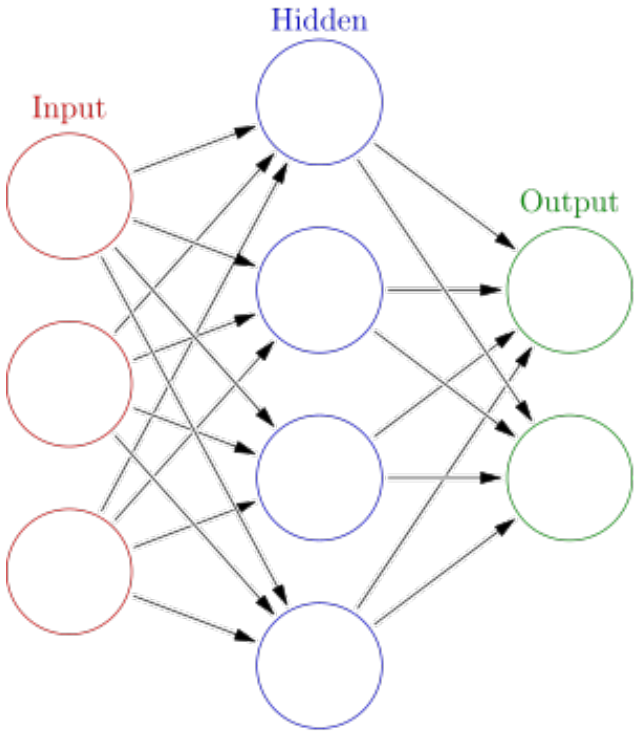

Figure 1: Structure of a Fully Connected Artificial Neural Network (Unknown, 2016)

In general, these layers are fully connected, meaning that each unit is connected to every unit in the adjacent layers. The connections between the individual units are assigned a numerical weight value that can be positive or negative. Positive weights promote the connection between the two units while negative weights suppress the connection. The greater the weight, the more the units influence each other.

Training a Neural Network

During the training of a Neural Network, information passes through the network in two directions. When the network is being trained, and when it is being used to make predictions after it has been trained, data is fed into the network through the input layer. This triggers the units in the network's hidden layer or layers. After passing through the hidden layers, the data will arrive at the output layer.

Each unit receives a signal from the units to its "left" (as visualised in figure 1). Depending on the weights of the connections between the units, some of the outputs will have higher values than others. Each unit add up the inputs from the connected units and applies some activation function to the value to normalise the value it passes forward to the next layer. This stage is known as **forward propagation**.

For the network to learn, some form of feedback must be implemented. To do this, the values on the output units are treated as an estimate. This estimate is compared to the value that the network is trying to predict which is included in the training data. The error of the network is calculated by getting the difference between the expected and predicted results.

This error is then propagated back through the network, in the same way that the data was passed forward through the network. The change amount of the error that each the connections between each pair of layers is responsible for is calculated and the weights on each connection are adjusted accordingly. This process is known as **backpropagation**.

With each iteration of forward propagation followed by backpropagation, the weights are updated in such that the error on the output is minimised, increasing the network's prediction accuracy over time.

Matrix Representation

It is possible and useful to represent the forward and backpropagation operations using matrix algebra. This greatly simplifies the notation used in calculations and can increase the computational performance when using software well suited to matrix-based calculations such as Matlab or the Numpy library in Python.

If the input data consists of *n* training examples with *m* features each, then we can represent the entire input dataset as an *n* x *m* matrix. If we take the owls15.csv dataset provided for this assignment as an example, the dataset contains 150 examples and each training example has four features: body-length, wing-length, body-width and wing-width. If we were to use all the training examples to train the network, then we could express the input data in the form of a 150 x 4 matrix.

In the same way, we can represent the values of the units in each network layer as well weights on the connections between each layer of units, as matrices.

Forward Propagation

The dimensions of the input matrix $\boldsymbol{X}$ are n x m, where n is the number of training examples and m is the number of features in each example, i.e. the input layer size. $\boldsymbol{W^{(1)}}$ is a matrix containing the values of the weights on each connection between the input and hidden layers. Its dimensions are *m* x *a* where *m* is the size of the input layer and *a* is the size of the hidden layer.
The first expression calculated is $\boldsymbol{z^{(2)}}$, the activation of the second layer of the network. Each entry in the matrix $\boldsymbol{z^{(2)}}$ is the sum of the each of the input units, multiplied by the weight on the connection between the units. We can calculate this by multiplying the input matrix $\boldsymbol{X}$ by $\boldsymbol{W^{(1)}}$, giving us an *n* x *a* matrix. This gives us our first expression:

$$z^{(2)} = XW^{(1)}$$

Now we need to apply an activation function to $\boldsymbol{z^{(2)}}$ in order to normalise it. For this network, I decided to use a Sigmoid activation function. This function is given by the equation:

$$g(z) = \frac{1}{1 + e^{-z}}$$

After the activation function is applied to each element in $z^{(2)}$, we are left with the matrix $a^{(2)}$ which has the same dimensions as $z^{(2)}$. This gives us the following expression:

$$a^{(2)} = g(z^{(2)})$$

where ***g(z)*** is the Sigmoid function.

We repeat the same method as before, replacing ***X*** with $a^{(2)}$ and $W^{(1)}$ with $W^{(2)}$ to calculate ***z(3)*** and then using the same sigmoid activation function as before to calculate $a^{(3)}$. We will call $a^{(3)}$ $\hat{y}$ as it is the network's prediction for the value of ***y***, the target from the training set. This gives us the following two expressions:

$$z^{(3)} = a^{(2)}W^{(2)}$$
$$\hat{y} = a^{(3)} = g(z^{(3)})$$

Backpropagation

We use backpropagation to adjust the weights on each of the connections such that the error in $\hat{y}$ is reduced. We do this by using batch gradient descent to minimise a cost function based on the difference between ***y*** and $\hat{y}$. For this network, we will use the following cost function:

$$J = \frac{1}{2}\sum(y - \hat{y})^2$$

If we combine all the expressions we used to calculate y, we get the following expression:

$$J = \frac{1}{2}\sum(y - (g(g(XW^{(1)})W^{(2)}))^2$$

This means that J is dependent on ***X***, $W^{(1)}$ and $W^{(2)}$. We don't have any control over ***X***, so we must change $W^{(1)}$ and $W^{(2)}$ to minimise the value of ***J***. We do this by calculating the partial derivative of J with respect to each weight matrix. First we calculate the rate of change of the cost function with $W^{(2)}$:

$$\frac{\partial J}{\partial W^{(2)}} = (a^{(2)})^T\delta^{(3)}$$

Where $\delta^{(3)}$ is the backpropagation error, given by the following expression:

$$\delta^{(3)} = -(y - \hat{y})g'(z^{(3)})$$

Where g'(z) is the derivative of the Sigmoid function, given by the equation:

$$g'(z) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

Next we calculate the derivative of the cost function with respect to $W^{(1)}$:

$$\frac{\partial J}{\partial W^{(1)}} = X^T\delta^{(2)}$$

Where:

$$\delta^{(2)} = \delta^{(3)}(W^{(2)})^T g'(z^{(2)})$$

Finally, having calculated these gradients, we can reduce the cost function by subtracting the product of the gradients and the learning rate ε from our weights.

Building the Artificial Neural Network

I decided to use the Python programming language primarily because the Numpy library make matrix operations very easy to carry out. Python also has libraries for manipulating CSV files, generating graphs and building simple GUIs.

First, I created the python class NeuralNetworkClassifier.py that can build and train an ANN using the methodology shown above.

Network Parameters

In the ANN built for this assignment, I decided to use a single hidden layer to reduce the complexity of the network. That meant that there were three parameters that needed to be optimised: The number of hidden units, the number of training iterations and the learning rate.

To find a good combination of hidden layer size and training iterations, I wrote a Python program that iterated through a list of iteration lengths and hidden layer sizes and graphed the results. These graphs can be seen in figure 2. The iteration lengths used were: 100, 300, 500, 750, 1000, 2000, 3000, 4000 and 5000. Hidden layer sizes from one to 30 were tested.

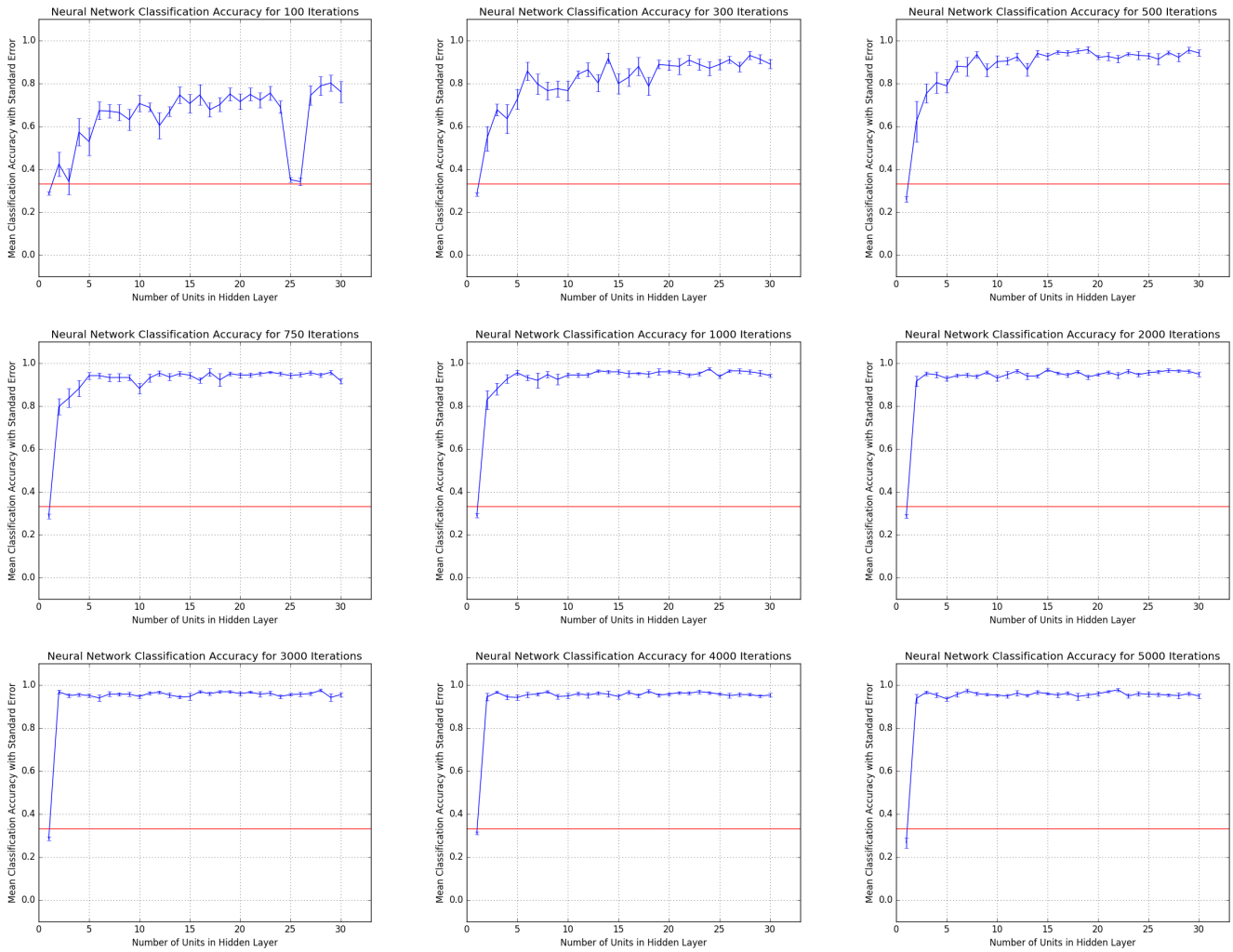

Figure 2: Test results for various hidden layer size and training iteration values

From these graphs, we ca see that the classifier's prediction accuracy doesn't increase appreciably beyond 3 hidden units and 2000 training iterations.

At this point I wrote another script that would build networks with the parameters established above and then vary the learning rate and graph the results. The result of this script is shown in figure 3.

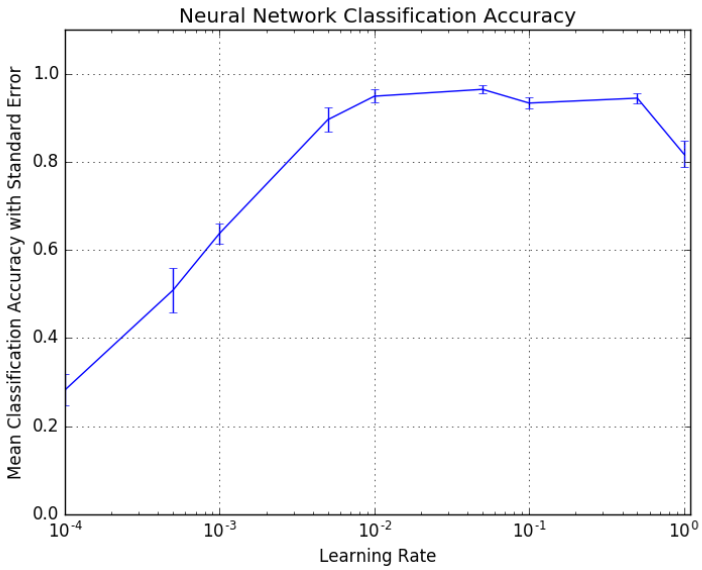

Figure 3: Graph of Accuracy vs Learning Rate

A learning rate of 0.05 provided the greatest prediction accuracy. The optimal parameters for the provided dataset, therefore are three hidden units, 2000 training iterations and a learning rate of 0.05.

Artificial Neural Network Classifier Operation

The ANN classifier is operated using a simple GUI. The GUI operation is shown in figure 4 below.

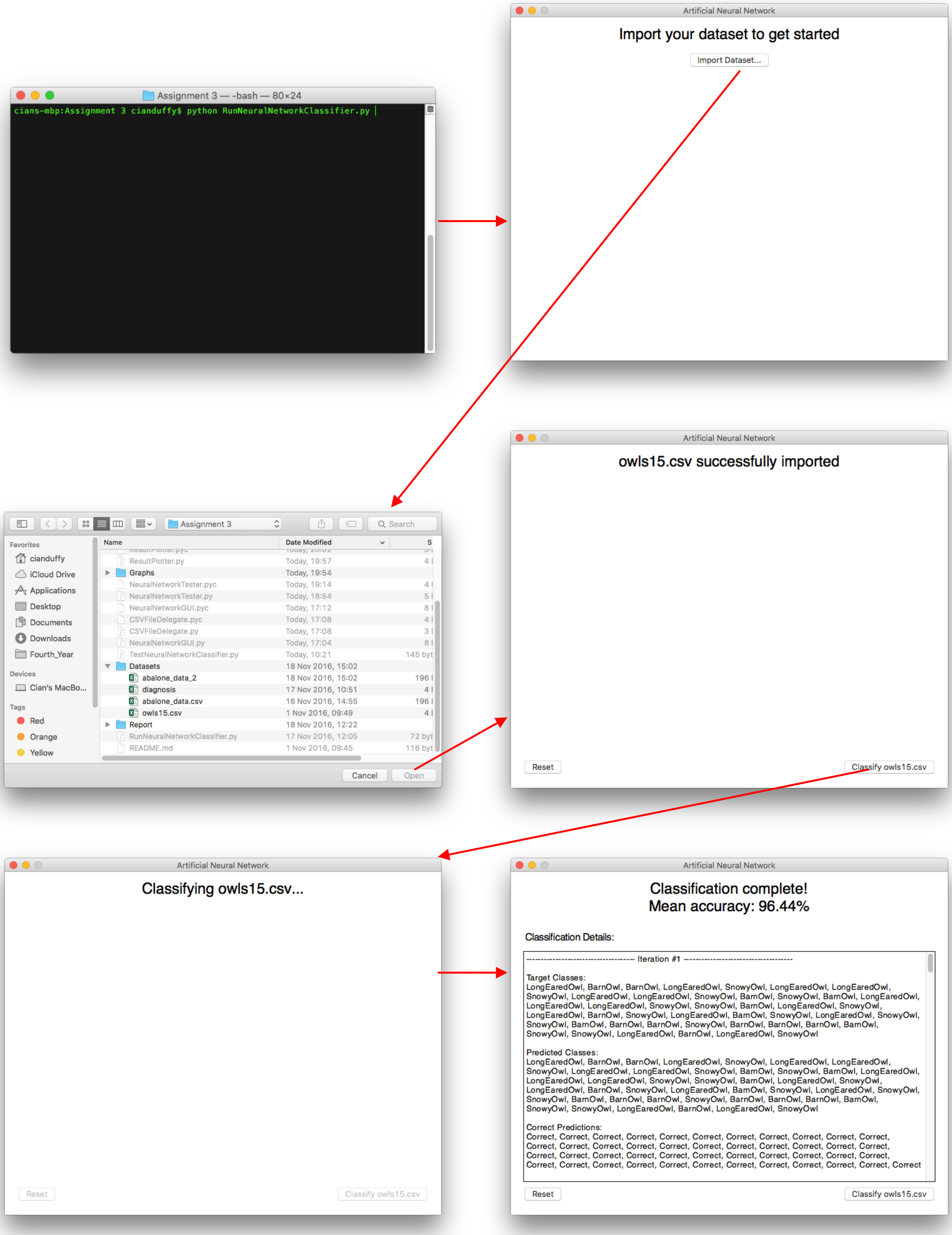

Figure 4: Artificial Neural Network Classifier GUI Operation

The GUI is opened by running the Python script file RunNeuralNetworkClassifier.py. When the “Import Dataset…” button is clicked, the dataset’s filepath is obtained using the file browser dialog. The dataset must be stored in CSV format with no header. The learning examples in the input dataset can contain of both numerical and categorical features but the target must be categorical.

When the classify button is clicked, the GUI creates a new *NeuralNetworkDriver* object and calls its *build_network_and_classify_data()* method. This creates a new *CSVFileDelegate* object that takes the filepath as an argument, imports the data from the CSV file, shuffles it and then separates two thirds of it into training data and the remaining third into testing data. Then it separates the training examples from the target classes for each.

The *NeuralNetworkDriver* then initialises a new *NeuralNetworkClassifier* object. The driver trains the classifier with the training set and then uses the testing set to evaluate its performance. It stores the predicted and target values for the training data as well as the prediction accuracy value in a dictionary which it adds to a list.

This process is repeated ten times. With each iteration, a new *CSVFileDelegate* and *NeuralNetworkClassifier* is created, meaning that each iteration is independent of the others.

After the ten iterations, the results are outputted to a CSV file called “output_<filename>.csv”. An extract from output_owls15.csv can be seen in figure 5. The accuracy of each iteration and the mean accuracy are displayed for the user in the GUI. This can be seen in figure 6.

```
fold_number,input,result
1,LongEaredOwl,LongEaredOwl
1,BarnOwl,BarnOwl
1,BarnOwl,BarnOwl
1,LongEaredOwl,LongEaredOwl
1,SnowyOwl,SnowyOwl
1,LongEaredOwl,LongEaredOwl
1,LongEaredOwl,LongEaredOwl
1,SnowyOwl,SnowyOwl
1,LongEaredOwl,LongEaredOwl
1,LongEaredOwl,LongEaredOwl
1,SnowyOwl,SnowyOwl
1,BarnOwl,BarnOwl
1,SnowyOwl,SnowyOwl
1,BarnOwl,BarnOwl
```

Figure 5: Extract from output_owls15.csv

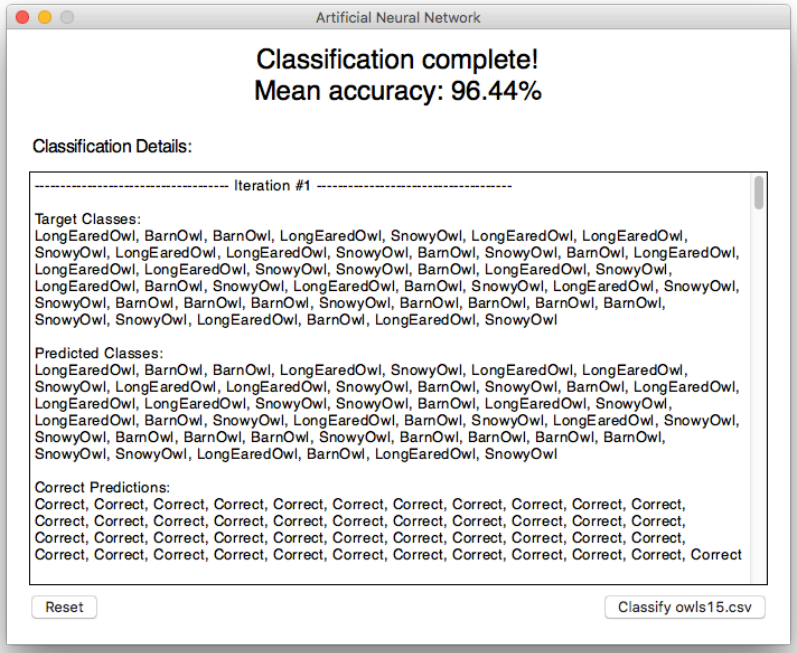

Figure 6: Classification results displayed for user in GUI

In my testing, the ANN had a prediction accuracy rate of 94-98% for the owls15.csv dataset. However, it did not perform as well for two other datasets that I downloaded from the internet.

References

Welch Labs. (2014). *Neural Networks Demystified*. [Online Video Series]. 4 November 2014.
Available at: https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU.
[Accessed: 1 November 2016].

Warren S. Sarl. 2002. *Neural Network FAQ*. [Website]
Available at: ftp://ftp.sas.com/pub/neural/FAQ.html.
[Accessed 5 November 2016].

beagle tk. *(2014). Machine Learning*. [Online Video Series]. 2 June 2014.
Available at: https://www.youtube.com/playlist?list=PLXBnOEEAyvLb1KF3h2FjJM7ZVf1YITUUM.
[Accessed: 5 November 2016].

Haykin S, 2008. *Neural Networks and Learning Machines*. 3rd ed. McMaster University, Canada: Pearson.

Unknown, 2016. *Colored_neural_network.svg* [ONLINE].
Available at:
https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg
[Accessed 1 November 2016].

The full code for this project is available at: https://github.com/CianDuffy/CT475_Assignment_3