5,265,1.00 Machine Learning Methods for Data Analysis

Agne Macijauskaite
Gabriel Leibovich
Andrea Posada Cardenas

2017-12-19

University of St.Gallen

Colour Recognition

Red, Green and Blue colour classification using Neural Networks and Logistic Regression with web-cam colour labelling demonstration

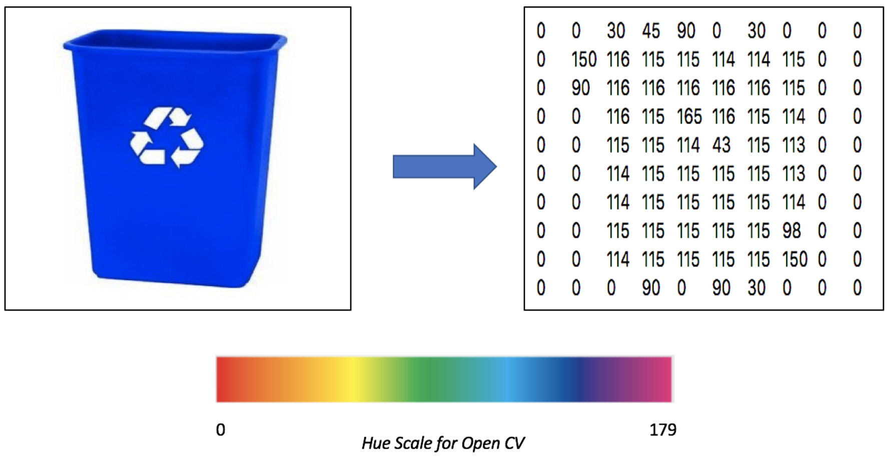
Contents

1 Introduction

Computer vision is an increasingly popular sub-filed of machine learning, which attempts to emulate biological sight in mechanical machines. It does so by reducing light sensor data into various numerical forms, like RGB, HSV or CYMK matrices, allowing this data to be used as an input into algorithms. Our goal is to train a learning algorithm to differentiate between Red, Blue, and Green (RGB) colored objects. Using a dataset of 685 images of RGB colored objects, our team trains nonlinear learning algorithms, and then select the best model based on error metrics from validation data. Linear algorithm is used to benchmark the results. One of the chosen trained algorithm is used by a standard webcam-enabled computer to label the colors of different objects shown to it by a user. The repository with the data and the code can be found in `https://github.com/gabeleibo/color_recognition`.

2 Data Collection and Preparation

Using the Google Search API, our team extracted 685 images Red, Blue, and Green colored objects. As the API limits search results to 100 images, thus we searched for the keywords ‘objects’, ‘things’, and ‘stuff’ for each color and compiled the results into folders for the respective color. We then ran a script to process the JPG image files using Open CV, a python computer vision library, and check for duplicates (215 duplicates or non-JPEG images were cleaned from the data). The processing step resized each image to 10 x 10 pixels and converted the RGB images to HSV (Hue Saturation Value) images. Open CV uses a Hue scale from 0 - 179, as shown in Figure 1 below. Finally we extracted just the Hue matrix from the HSV matrix and saved it to a file with its corresponding color encoding - ‘Blue’: [0,0,1], ‘Red’: [0,1,0], ‘Green’: [1,0,0] - to be used in an Neural Network. Pretreatment of data inputs and outputs, like normalization, were left to the discretion of the algorithm’s trainer. We had in total 240 blue pictures, 227 - red and 218 - green.



0	0	30	45	90	0	30	0	0	0
0	150	116	115	115	114	114	115	0	0
0	90	116	116	116	116	116	115	0	0
0	0	116	115	165	116	115	114	0	0
0	0	115	115	114	43	115	113	0	0
0	0	114	115	115	115	115	113	0	0
0	0	114	115	115	115	115	114	0	0
0	0	115	115	115	115	115	98	0	0
0	0	114	115	115	115	115	150	0	0
0	0	0	90	0	90	30	0	0	0

Figure 1: The Figure depicts an example of image preparation, converting the image on the left into a the Hue Matrix on the right.

3 Methodologies

3.1 Artificial Neural Network

A three layer Neuronal Network with bias was built, following the algorithm described in lectures from 5,265: Machine Learning Methods for Data Analysis at the University of St. Gallen (Ortega, 2017a). Given the goal of classification, the sigmoid function was selected as the activation function. This is also beneficial for later comparison purposes. The ideal weights are found through the minimization of the cost function (Eq. 3) and its gradient. This is done using *nlminb* in *R*.

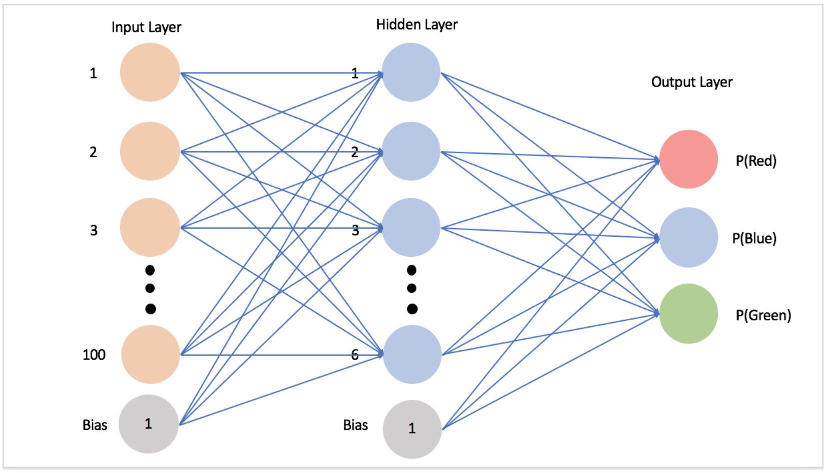


Figure 2: Simplified Neural Network Architecture Schematic

3.1.1 Pretreatment of data: Normalization

There is no need to normalize the output due to its values (vector of 0s and 1s). On the other hand, the normalization of the input is done as follows.

Let $X^i_{max} = max_{k \in 1,...,n} X_{k,i}$ and $X^i_{min} = min_{k \in 1,...,n} X_{k,i}$ where n is the number of samples. Each element $X_{k,i}$ is replaced by

$$\hat{X}_{k,i} = \frac{(X_{k,i} - X^i_{min})}{X^i_{max} - X^i_{min}} \tag{1}$$

3.1.2 Algorithm description

Let $\mathbf{x} := (x_1, x_2, ..., x_p)'$ be the input values (HUE values for pixel per image) with p being the number of parameters. x_0 is an intercept or bias. $\mathbf{a}^{(I)} := (a^{(I)}_1, a^{(I)}_2, ..., a^{(I)}_{s_I})'$ are the activation values of neurons in layer I where s_I the number of neurons in the layer without the bias/intercept, a^I_0 is an intercept.

Being J the number of layers, Θ^I the weights for layer I -rank of Θ is $1 : (J-1)$- and $\sigma(\cdot)$ the activation function,

$$\mathbf{a}^{(I)} = \begin{cases} \mathbf{x} & I = 1 \\ \sigma(\Theta^{(I-1)}\mathbf{a}^{(I-1)}) & 1 < I \leq J \end{cases} \tag{2}$$

$a^J \coloneqq f(\mathbf{x}, \Theta) = f_\Theta(\mathbf{x})$ (following the notation presented in the Lecture 5: Neural Network foil (Ortega, 2017a)). The value of $f_\Theta(\mathbf{x})$ is calculated recursively. This procedure is known as Forward Propagation. The bias required appending a column of ones to each set of weights and a row of ones to each $\mathbf{a}^{(I)}$ excepting when $I = J$.

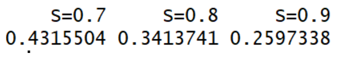
Being K the number of neurons in the last layer (number of classes to classify) and n the sample size, the cost function is defined as

$$\begin{aligned} J(\Theta, x, y) \coloneqq & -logL(\Theta, x, y) = \\ & -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}[y_k^{(i)}log((f_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)})log(1 - (f_\Theta(x^{(i)}))_k)] \\ & + \frac{\lambda}{2n}\sum_{I=1}^{L-1}\sum_{i=1}^{s_I}\sum_{j=1}^{s_{(I+1)}}(\Theta_{j,i}^{(I)})^2 \end{aligned} \tag{3}$$

The reader is invited to refer to the foil, *Lecture 5: Neural Networks* (Ortega, 2017a), for further details on gradient calculation.

3.1.3 Size of training data and number of neurons in the hidden layer

In order to design a Neural Network with good performance, the values for the size of training data, and therefore the size of testing data, as well as the the size of the number of neurons in the hidden layer have to be determined. Using a nested loop, where the external loop makes reference to the size of training data as a proportion of the total sample -rank 0.7 up to 0.9 with step 0.1- and the internal one to the number of neurons in hidden layer with minimum 5 neurons and maximum 50 neurons-, the MSEs over training and testing are calculated. The ideal size is taken as the one that leads to the minimum average MSEs for testing over all number of neurons per size. This ideal size according to the results obtained, shown below, is 0.9.



```
    S=0.7     S=0.8     S=0.9
0.4315504 0.3413741 0.2597338
```

Figure 3: Average MSEs per size S of training data

The number of neurons in the hidden layer is estimated with same logic: the one that leads to the minimum MSE out of the ideal size. 6 neurons with a MSE for testing of 0.2307 is the one performing the best. In the following graph the MSE values of testing (red line) and training (black line) for different number of neurons in the hidden layer are shown. As said before, it can be seen that 6 neurons leads to the minimum MSE of testing(highest depression). MSEs of training have a stable behaviour around 0.06. However, MSEs of testing tell a different story. They present huge fluctuation around 0.25.

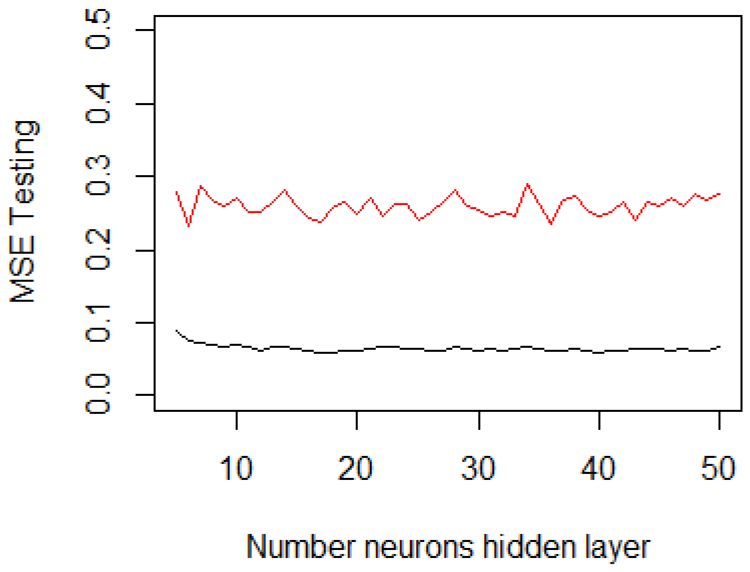


Figure 4: MSE of testing (red line) and training (black line) for different number of neurons in the hidden layer with size 0.9 for training data set

The procedure explained above has a time complexity issue. This runtime could be reduced using parallelization due to independence of the calculations.

3.2 Logistic Regression

In order to benchmark and compare the results of Neural Networks classifier, we run the data using logistic regression. We use Logistic Regression only as a benchmark as it is not well adapted to multi-class classification and has unstable parameters when classes are well separated (Ortega, 2017b). Since we have a non-binary classification problem and those three classes (red, blue and green) are exclusive, we use **One-vs-all** logistic regression model.

One-vs-all logistic regression is done by creating three separate binary classification problems. For example, when training the data and finding β coefficients for blue colour, we equate the dependent variable of blue colour to 1 and everything else (both red and green) to 0. With this new dataset, we run the standard logistic regression (using *glm* function in *R*), where the dependent variable is a probability that y is equal to class i given x is parametrized by β (Ortega, 2017b):

$$h_\beta^{(i)}(X) = P(y = i|x; \beta) = \frac{e^{\beta_0+\beta_1*x_1+\ldots+\beta_{100}*x_{100}}}{1 + e^{\beta_0+\beta_1*x_1+\ldots+\beta_{100}*x_{100}}} \tag{4}$$

where x is HUE values extracted from the picture, β is coefficients for each HUE value in the picture, i is the class (red, green or blue)

After this, we have three classifiers, each of which is trained to recognize one of the three classes. We run all three of our classifiers on the HUE values input and we pick the class (red, blue or green) that maximizes the three.

$$\max_i h_\beta^{(i)}(X) \tag{5}$$

We run the code for training data and testing data using the same cut as in Neural Networks classification problem (90%).

3.2.1 Aggregated Logistic Regression Alternatives

Logistic regression is better suited to identify the objects that depend on the place within the picture (i.e. pixels, HUE values location) because it gives the β parameters to each and every square in the picture (in our case 100 + constant). If the dependent variables are exclusive and each case have the unique shapes (i.e. numbers), one-vs-all logistic regression classifier would work pretty well. However, in our case we try to identify colours without focusing on their shape or location within the picture. The structure of logistic regression model would perform poorly in identifying the colours of the pictures that have the key object in an unusual location (i.e. only in the corner) since the weights given to that location would be very low or the exact same picture with different colour would be hard to classify. Therefore, we run a a logistic regression on the aggregated data (based on mean and the most common HUE value). These models may work well in this situation, but they lack scalability - they eliminate the properties of the object and hinders the possibility to build the new algorithms on top of it (i.e. identifying the object itself) and they also lack the applicability to identify more than one colour in the picture. In addition, the choice of aggregation method might work well with some sample data, but under-perform with another. Finally, since the red colour is both in the beginning and the end of HUE values spectrum, using aggregation might be misleading.

We can see the characteristics of aggregated variables in the Figure 5. The usual value for blue colour is supposed to be around 100, for green ∼50 and for red both ∼1 and ∼179. We can see from the a) graph that blue and green colours have the average HUE value around the expected levels, however, the red colour HUE value means are, as expected, dispersed between the two extremes and, thus, result in being far from any of the correct values. This provides us with insight that the results of the classifiers should perform the worst for the red colour. Also, running the logistic regression based on the averages would not lead to reliable results, since red colour would be under-classified because its aggregated values average to the values of blue colour. Median does not solve this problem, however, Most Common HUE value aggregation reduces this problem and should work correctly in classifying the red colours that are close to the maximum of HUE values. However, as noted before, logistic regressions results on aggregated data becomes sensitive to the choice of measure: testing other samples with mainly low HUE values for the red colour, this trained logistic regression would work poorly.

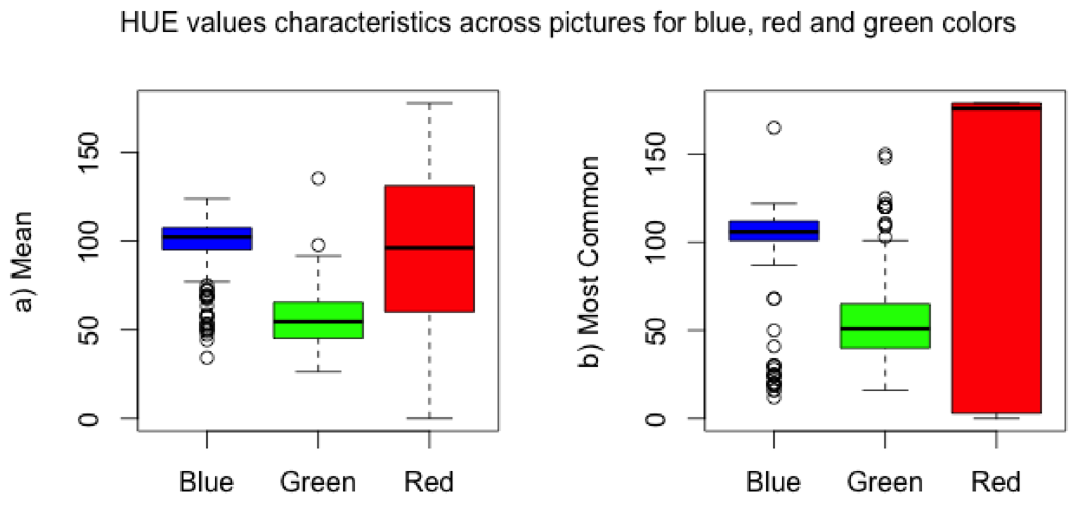


Figure 5: The Figure depicts the characteristics of HUE values across the pictures in the dataset. In Figure a) it shows the characteristics of HUE value means distributions across the pictures and b) depicts the distributions of the most prevalent HUE values across the pictures. The aggregated numbers are calculated excluding the noise: the values of white (0) are excluded

We run the logistic regression classifier using the most common aggregated values as independent variables, reducing the equation to:

$$h_\beta^{(i)}(X) = P(y = i|x; \beta) = \frac{e^{\beta_0+\beta_1*x_1}}{1 + e^{\beta_0+\beta_1*x_1}} \tag{6}$$

where x_1 is aggregated independent variable (the most common HUE value).

4 Results and Discussion

Important terminology

- Accuracy: Percentage correctly classified over all.
- Precision: Fraction of correctly predicted of an actual class.
- Recall: Fraction of correctly predicted of a predicted class.

4.1 Artificial Neural Network results

For a testing data size of 0.1 of total sample and with a 6 neurons in the hidden layer, an accuracy of 82.61% is obtained. The precision is 0.9583, 0.5652 and 0.9545 and the recall is 0.8519, 0.9286 and 0.75 for blue, red, and green, respectively. The weighted precision and recall is 0.9020, 0.7027 and 0.8400, following the correspondence expressed before. The confusion matrix -the matrix that shows correct classification and misclassification- and the matrix with the values represented as percentage are shown below.

Table 1: Neural Network Confusion Matrix - Testing data. **Accuracy 82.61%**

		Predicted		
Actual		*Blue*	*Red*	*Green*
	Blue	23	0	1
	Red	4	13	6
	Green	0	1	21

Table 2: Neural Network results - Testing data. **Accuracy 82.61%**

		Predicted		
Actual		*Blue*	*Red*	*Green*
	Blue	95.83%	0%	4.17%
	Red	17.39%	56.52%	26.09%
	Green	0%	4.54%	95.45%

For a training data size of 0.9 of total sample and with a 6 neurons in the hidden layer, an accuracy of 98.05% is obtained. The precision is 0.9769, 0.9853 and 0.9796 and the recall is 0.9814, 0.9950 and 0.9648 for blue, red, and green, respectively. The weighted precision and recall is 0.9791, 0.9901 and 0.9722 following the correspondence expressed before. The confusion matrix and the matrix for the percentage of actual classified are shown below.

Table 3: Neural Network Confusion Matrix - Training data. **Accuracy 98.05%**

		Predicted		
Actual		*Blue*	*Red*	*Green*
	Blue	211	0	5
	Red	1	201	2
	Green	3	1	192

Table 4: Neuronal Network results - Training data. **Accuracy 98.05%**

		Predicted		
Actual		*Blue*	*Red*	*Green*
	Blue	97.68%	0%	2.31%
	Red	0.49%	98.53%	0.98%
	Green	1.53%	0.51%	97.96%

As a conclusion, it would be important to use parallelization to improve run time and to use more data given the relative small amount of samples (685 images: 240 blues, 227 red and 218 green. In addition, the greater amount of blue samples could explain why blue is the colour with higher precision, contrary to red being the one with worst probability of being correctly classified, although it shows the opposite when the confusion matrix for training data is analyzed.

4.2 Logistic regression results

The benchmarked logistic regression classifiers perform significantly worse than the neural network classifier. From the Table 5 we can see that accuracy dropped from 82.61% to 62.32%. The main drop in the accuracy comes from misclassification of red and green colours. The Neural Network, as expected, has more flexibility in coping with dichotomy of red colour hue values. The red colour classification in testing data is so weak that it classifies the red colour as green more often than as the red. We can also see that the classifier identifies less red colour than there is in the sample, giving the stronger importance of other two colours (both in training and testing data).

Table 5: Confusion matrix of logistic regression - *testing data.* **Accuracy - 62.32%**

		Predicted		
		Blue	*Red*	*Green*
Actual	*Blue*	21	1	2
	Red	7	8	11
	Green	1	4	14

Table 6: Confusion matrix of logistic regression - *training data.* **Accuracy 72.56%**

		Predicted		
		Blue	*Red*	*Green*
Actual	*Blue*	179	29	8
	Red	47	105	49
	Green	17	19	163

The use of aggregated measures in the logistic regression reduced the fall in accuracy from training to testing data. In addition, accuracy increased compared to the original logistic regression model, but not so much (see Table 7). The increase in accuracy is hindered by the difficulty to classify the red color using the aggregated data. In the aggregated data the Green color becomes better classified than blue (which is the case in original logistic regression), because it has less outliers (see Figure 5). It is also interesting to note that the misclassification of blue and green colors to red basically disappears. Therefore, the aggregated model would perform quite well if the colors would have only one range of HUE values, but it will provide the limitations discussed in the methodology part. Also, having one explanatory value reduced the flexibility of the model to separate the colors in the way that blue color in the training sample was classified more often as green compared to original logistic model.

In comparison to neural network classifier, the accuracy dropped, but not so much. The key advantage of neural networks is its ability to classify the red color, which the logistic regression fails to do at expense of accuracy. Thus, it could be concluded that the neural network classifier outperformed the benchmark logistic regression and is able to perform with high accuracy even with a small sample size.

Table 7: Confusion matrix of logistic regression using most common HUE values - *testing data.* **Accuracy - 73.91%**

		Predicted		
		Blue	*Red*	*Green*
Actual	*Blue*	22	0	2
	Red	3	11	12
	Green	1	0	18

Table 8: Confusion matrix of logistic regression using most common HUE values - *training data.* **Accuracy - 77.60%**

		Predicted		
		Blue	*Red*	*Green*
Actual	*Blue*	193	1	22
	Red	8	113	80
	Green	25	2	172

4.3 Final comments and Recommendations

As we expected, the neural network performed significantly better than the logistic regression classifier. The accuracy in training data was almost 100%, which dropped to 83% in testing data, indicating the possibility of over-fitting. The main drop in the accuracy came from the red colour misclassification, which we expected due to distribution of HUE values. The model failed to identify the cases of red, which led to over-classification of the other two colours. The manipulation of data to have red colour around the same limits (i.e. change all of the 1-10 HUE values to 179 or the opposite) or using different than HUE values specification of colours could be used to overcome this problem, however, it might lead to other limitations (i.e. scalability). In addition to that, this data sample was relatively small and a bigger sample could lead to higher precision both in training and testing. Neural network classifier has also advantages of scalability: the code could be used to train more colours without losing much accuracy (as would be the case in logistic regression). In addition, Neural network classifiers do not eliminate the object form, which means that the code could be the initial basis for further build ups, i.e. identification of the object itself. The disadvantage of neural network classifier is its slow execution.

5 Prototype Product Demonstration

The demonstration program was written using Python and Open CV to directly access webcam data from the end-users machine. When the user wants the program to label an object in the frame, he or she simply press ‘c’ on their keyboard to freeze the frame, and then use their mouse to select the desired object. This will trigger the the same data preprocessing steps as for the training data and run the selected frame through the trained Neural Network. The probabilistic, numerical output is reduced to a color class based on the output neuron with the highest probability. Finally, the cropped image is displayed to the user with label of the predicted color.

The real-time color classifier predicts colors reasonably well when the objects are well-lit, vibrant, and uni-colored. When conditions become sub-optimal, the algorithm performed extremely poorly. We attribute these poor results to the uniformity of the training data. Although data features many objects at different angles, all the shots are professionally taken with expensive camera and lighting equipment. The field tests of our algorithm featured a basic laptop webcam and mediocre lighting conditions. Future iterations of this project should include more photos similar to the testing context and ones with variations in lighting and quality to improve performance. These improvements in data quality should also allow for experiments using more color classes to be conducted.

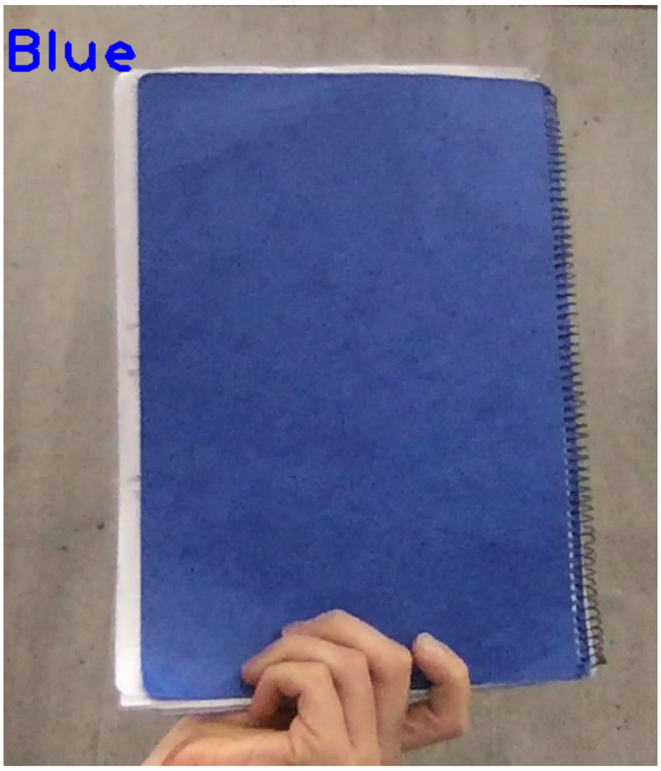


Figure 6: Example output of the real-time classifier on a blue notebook

6 References

Gene Kogan, (2017). *Scraping Full Size Images from Google Images.* `https://gist.github.com/genekogan/ebd77196e4bf0705db51f86431099e57`

Juan-Pablo Ortega, (2017a). *Machine Learning Methods for Data Analysis: Chapter 5: Neural Methods.* Class notes.

Juan-Pablo Ortega, (2017b). *Machine Learning Methods for Data Analysis: Chapter 4: Clustering and classification methods.* Class notes.

Juan-Pablo Ortega, (2017c). *Exercise4˙Handwriting˙recognition* R code.

Juan-Pablo Ortega, (2017d). *Exercise1˙ANN˙Handwriting˙recognition* R code.

Juan-Pablo Ortega, (2017e). *Exercise2˙backpropogation* R code.

7 Appendix

7.1 Code Repository

`https://github.com/gabeleibo/color_recognition`

7.2 Logistic regression results

Table 9: Logistic regression results - testing data. **Accuracy - 62.32%**

		Predicted		
		Blue	*Red*	*Green*
Actual	*Blue*	82.87%	4.17%	8.33%
	Red	26.92%	30.77%	42.31%
	Green	5.26%	21.05%	73.68%

Table 10: Logistic regression results using most common HUE values - *testing data.* **Accuracy - 73.91%**

		Predicted		
		Blue	*Red*	*Green*
Actual	*Blue*	91.67%	00.00%	8.33%
	Red	11.54%	42.31%	46.15%
	Green	5.26%	0.00%	94.74%

7.3 MSE Results from Variable Iteration

Training Error

	Training Data / Total Data		
# of Hidden Neurons	70%	80%	90%
5	7.85%	9.44%	8.75%
6	6.52%	6.82%	7.55%
7	6.20%	7.48%	7.29%
8	6.30%	6.94%	6.89%
9	6.91%	6.76%	6.58%
10	6.70%	6.52%	6.89%
11	6.15%	6.53%	6.67%
12	6.27%	6.57%	6.20%
13	6.01%	6.26%	6.63%
14	6.08%	6.33%	6.66%
15	6.19%	6.28%	6.55%
16	5.99%	6.06%	6.13%
17	6.69%	6.24%	5.86%
18	6.13%	6.16%	5.99%
19	5.62%	6.78%	6.21%
20	5.92%	6.19%	6.26%
21	6.44%	6.34%	6.49%
22	6.66%	6.16%	6.57%
23	6.38%	6.67%	6.81%
24	6.15%	6.49%	6.46%
25	6.53%	6.49%	6.37%
26	5.76%	6.05%	6.14%
27	5.73%	6.01%	6.25%
28	5.98%	6.11%	6.72%
29	6.35%	6.43%	6.42%
30	6.24%	6.46%	6.11%
31	6.27%	6.39%	6.43%
32	5.81%	6.21%	6.03%
33	6.74%	6.33%	6.42%
34	6.54%	6.51%	6.59%
35	5.90%	6.35%	6.41%
36	6.18%	6.27%	6.02%
37	6.61%	6.53%	6.16%
38	5.62%	6.52%	6.51%
39	6.11%	6.17%	6.24%
40	6.00%	6.34%	5.79%
41	6.58%	6.18%	6.26%
42	6.10%	5.83%	6.21%
43	6.52%	6.37%	6.37%
44	6.29%	6.43%	6.29%
45	6.27%	6.02%	6.30%
46	6.45%	6.41%	6.18%
47	5.88%	6.44%	6.41%
48	6.05%	6.15%	6.01%
49	6.81%	6.46%	6.23%
50	5.95%	6.18%	6.71%

Testing Error

	Training Data / Total Data		
# of Hidden Neurons	70%	80%	90%
5	44.08%	35.00%	27.85%
6	41.80%	33.70%	23.07%
7	42.83%	33.20%	28.61%
8	42.13%	34.65%	26.65%
9	44.35%	34.23%	26.06%
10	43.29%	34.52%	27.08%
11	41.96%	33.79%	25.13%
12	44.17%	33.59%	25.00%
13	43.21%	34.53%	26.37%
14	42.12%	33.31%	28.19%
15	42.72%	34.10%	26.04%
16	42.71%	32.05%	24.24%
17	42.95%	34.36%	23.86%
18	42.11%	35.41%	25.63%
19	42.91%	34.88%	26.57%
20	43.55%	34.53%	24.94%
21	43.79%	34.18%	26.93%
22	43.97%	32.89%	24.59%
23	42.58%	33.29%	26.25%
24	42.69%	33.41%	26.15%
25	42.77%	33.69%	24.02%
26	43.03%	32.69%	25.21%
27	42.09%	32.70%	26.53%
28	42.12%	34.43%	28.02%
29	43.19%	34.09%	25.92%
30	43.37%	33.03%	25.49%
31	43.78%	33.94%	24.56%
32	42.24%	34.08%	25.23%
33	43.78%	33.94%	24.61%
34	43.68%	35.23%	29.02%
35	43.09%	32.81%	26.25%
36	42.83%	33.17%	23.48%
37	44.29%	35.10%	26.70%
38	41.77%	32.62%	27.43%
39	43.29%	33.72%	25.34%
40	43.24%	35.43%	24.59%
41	43.85%	35.10%	25.16%
42	42.70%	34.90%	26.46%
43	44.03%	34.94%	24.13%
44	43.49%	37.24%	26.54%
45	43.83%	33.65%	25.84%
46	43.67%	35.26%	27.10%
47	43.85%	35.82%	25.88%
48	43.28%	34.41%	27.60%
49	44.54%	34.20%	26.76%
50	43.42%	34.52%	27.72%