Dresses and Trousers

Nicholas Mulligan

Introduction

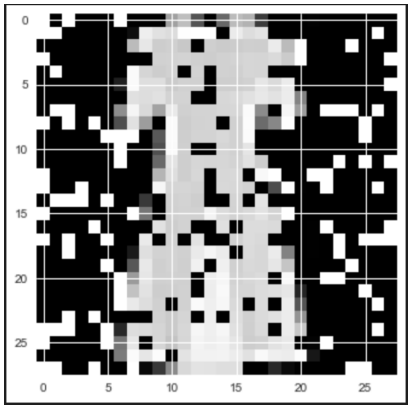
This project was for my Machine Learning class, where we were assigned to create a model to classify images into two categories: dresses or trousers. This document discusses the various parameters and techniques I used to make a model with the highest possible accuracy.

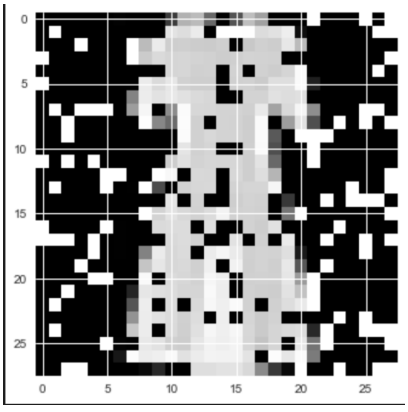
Testing

Throughout the process, I relied on k-fold cross validation (k = 5) testing to assess the training and testing accuracy of the data. For each data set, I used the average accuracy (or error rate) of the cross validation to measure performance.

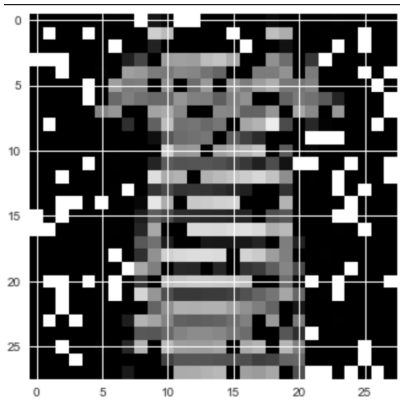
Image Transformations and Manipulation

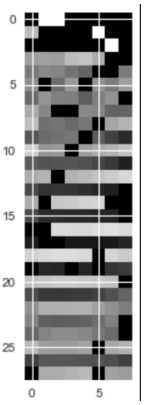
The first aspect of the data that I wanted to modify was to increase its amount. I took the recommendation from the spec to flip all images to double the size of the training set for the model. I created a function that made a copy of each image, rotated the image on its vertical axis, and then added it to the given training set as shown below. The right image is a left-right mirror image of the left image (taken from the raw data set).



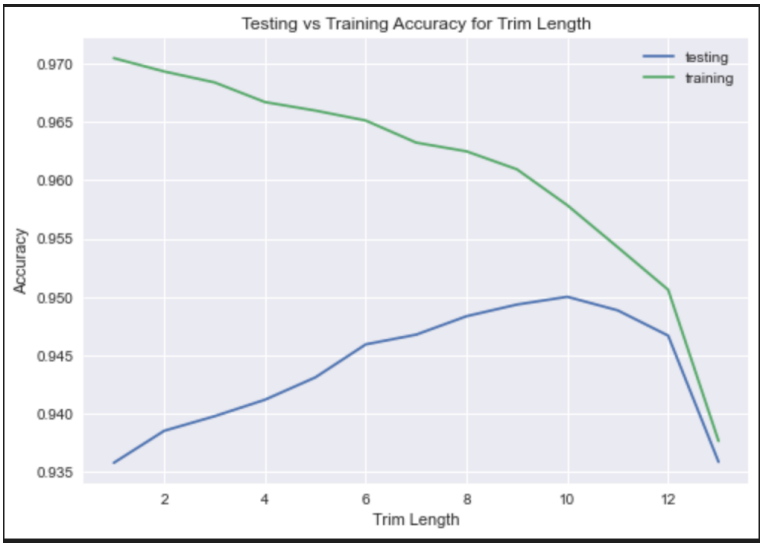


Another idea that I had for manipulating the data was trimming the image itself. Observing these images, most of the pixels on the outer left and right borders appear unneeded, as many of the dress and trouser images are centered, surrounded by noise. To remove the bordering images, I created a function whose arguments include the raw training data, and an integer representing the amount of pixels to trim from the left and right sides of the image. An example is shown below for trim length of 10:



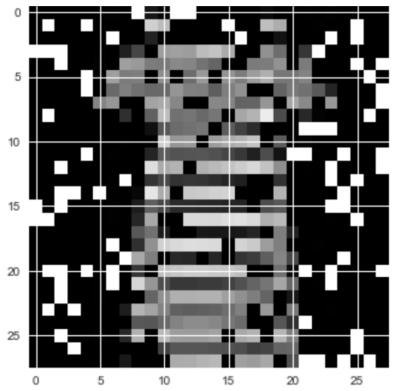


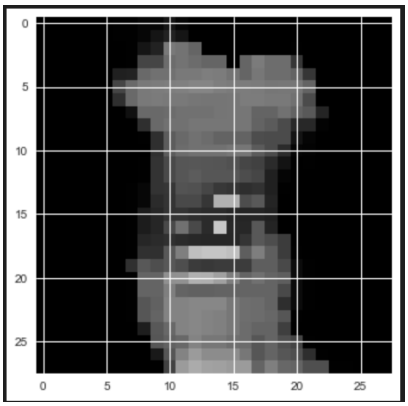
I then tested the function for varying trim values from 0 to 13. As the original image has a width of 28, the maximum usable trim value is 13, as anything higher would result in no image. The accuracy for models that trim the image with values 0 (no trimming) to 13 is shown to the right.



The last idea I had for data manipulation was to reduce the noise in the data. I used the cv2 library, which includes a function called medianBlur. I believed this would be the best option in the cv2 library, as the documentation stated that it was recommended for "reducing salt and pepper noise." medianBlur replaces each pixel with the median value of the surrounding pixels. Observing the difference on

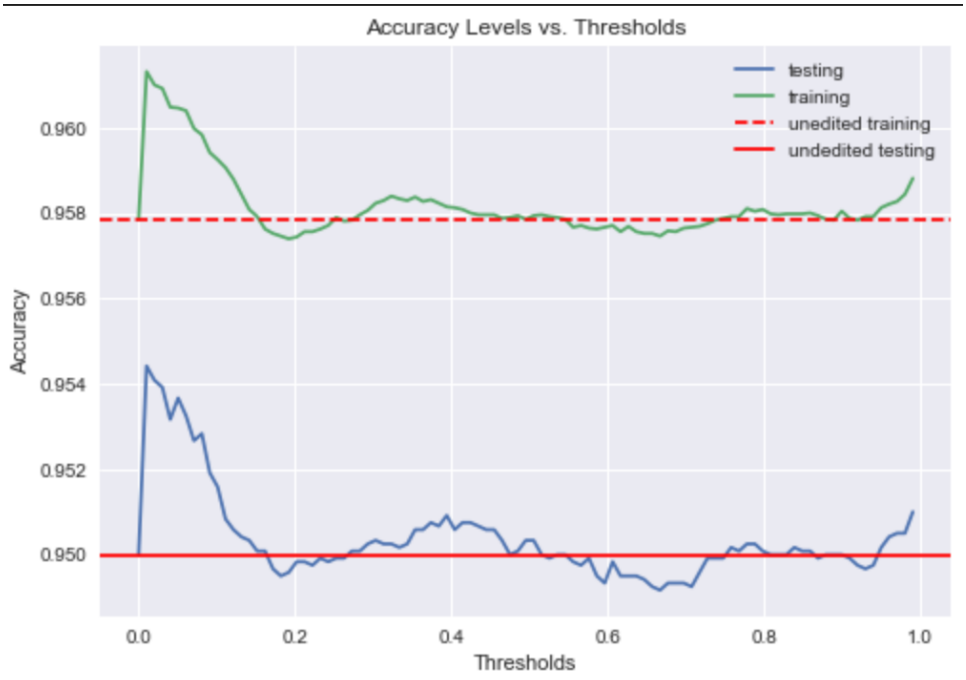
individual images, I predicted a large improvement in the model's accuracy. An example is provided below, where the image on the right is the result of applying medianBlur to the image on the left:





Descriptive Feature Additions

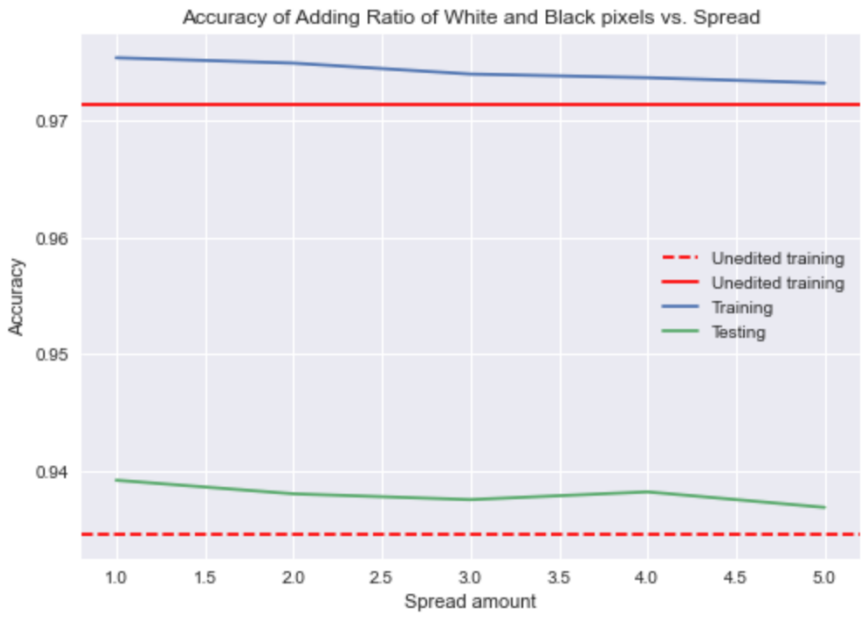
My first feature addition was calculating the ratio of black to white pixels. This seemed like a good place to start, and easy to implement accurately. To do so, I needed to decide on the threshold for the value that qualifies as black. I tested various values between 0 and 1. The results are plotted below:



The graph clearly shows a small increase in accuracy in both training and testing data with the addition of this feature. I decided on a threshold of 0.01, as that gives the highest accuracy.

Similarly, I also added two more features: the total number of white pixels, and the total number of black pixels (both scaled using MinMaxScaler). I wasn't sure what to expect, but I did observe a small boost in performance with the addition of these features.

I also added a feature that calculated the average brightness for the middle *n* columns. The inspiration for this was that I believed the trousers would have a column of black between the legs, while the dresses would not. The figure below shows the testing and training improvement relative to an unedited data set when adding the feature of brightness in the middle *n* columns for the values one through five. The highest testing accuracy occurred when adding the ratio of the middle two columns (spread = 1):



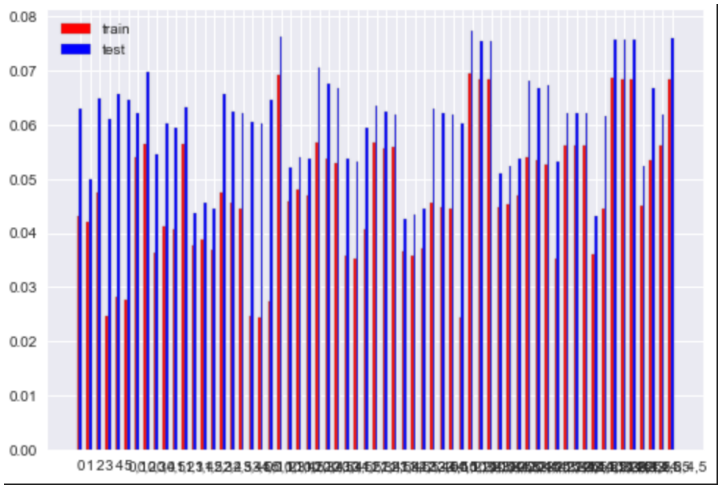
Feature Selection

To find the best subset, I created a function array, with each function adding a feature or editing the data set in some way. I then created a function to calculate each subset of the array, and calculate the error rate of that feature subset. The feature function array key is:

- 0: adds rotated [flipped] images to the data set

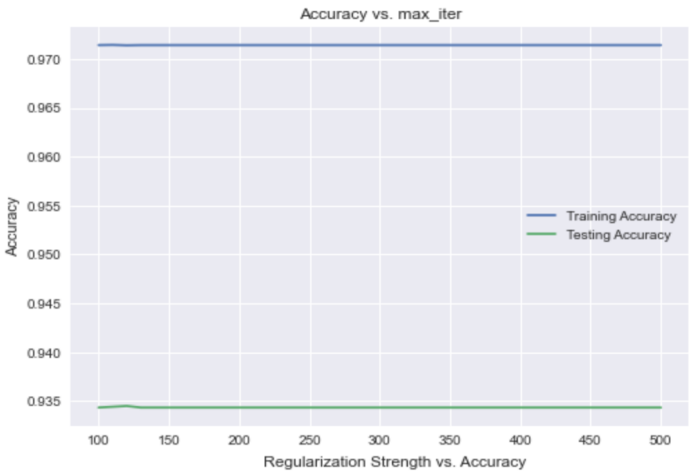
- 1: trims all data by removing the left and right 10 columns
- 2: reduces noise by blurring images
- 3: adds a ratio black and white in the middle two columns
- 4: adds a ratio of total amount of black and white pixels
- 5: adds the raw amount of black and white pixels (then scales with MinMaxScaler)

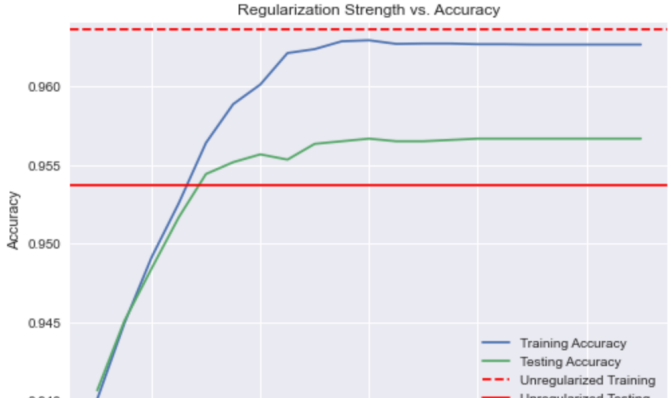
Though difficult to interpret, the graph showing the error rates for each of the 63 subsets is shown below. The y-axis represents the error rate, and the x-axis represents each subset tested. I will later show a more interpretable graph of the best subsets, but I thought that I would include this to demonstrate my testing procedure, along with showing the variance in the different subsets.



Logistic Regression Parameters

Before selecting the best subset of, I conducted tests for the regularization strength and max_iter parameters in the LogisticRegression function. For the max_iter parameter, I tested values from 100 (default) to 500. I saw a miniscule improvement for a value of max_iter = 120, so I decided to use this value for my final model. As for regularization strength, I found that a strength of 1000 performed best. The graphs are located to the right, with the top graph demonstrating

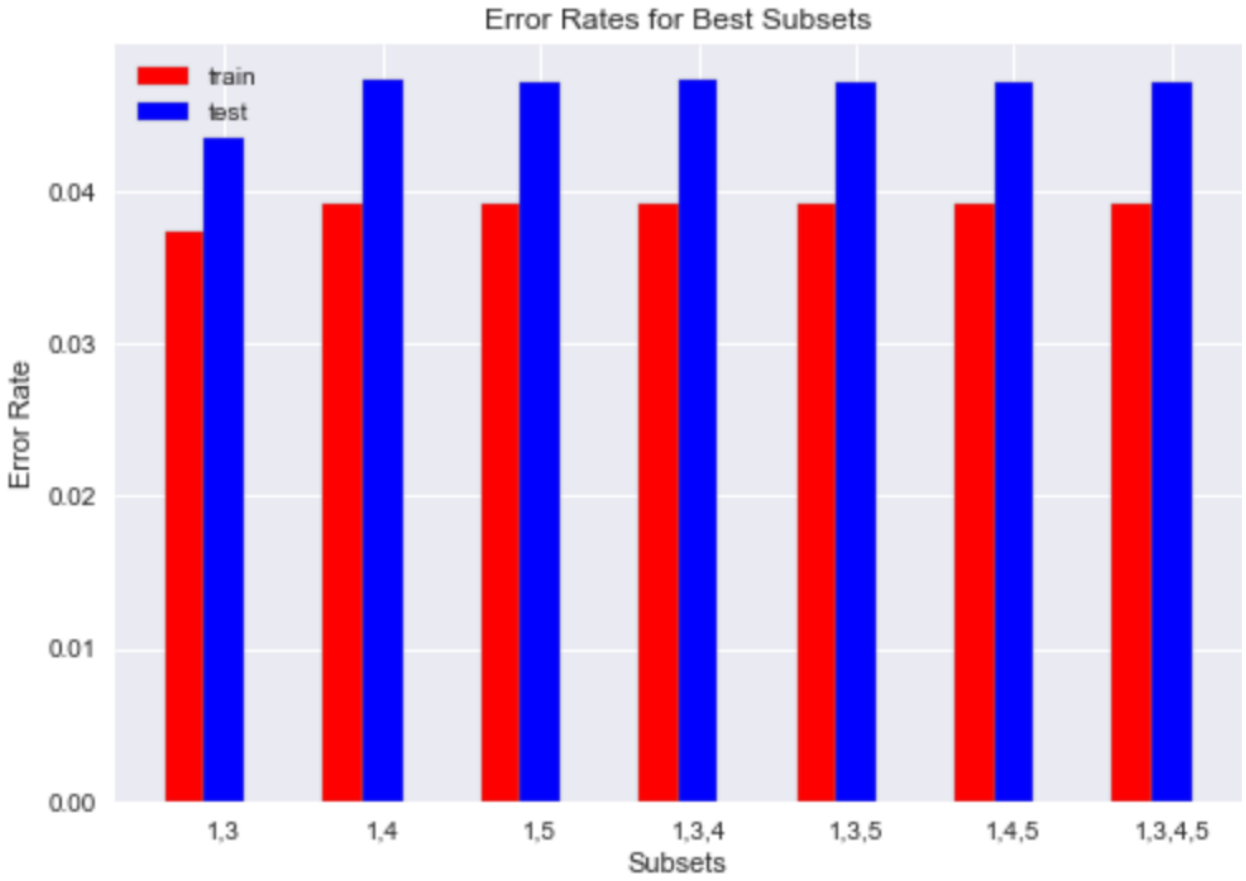




the effect of max_iter variation on accuracy, and the bottom graph showing the effect of regularization strength on accuracy

Best Subset

Now having calculated the ideal parameters for the LogisticRegression function, I found the best subsets of features (only subsets that resulted in an accuracy of over 95%), and logged their training and testing accuracy:



The most accurate model used only two modifications to the data: Trimming the dataset (1), and adding a feature of the ratio of black and white pixels (3). To summarize, the final model trimmed the data by a length of 10, added a feature of calculating the ratio of black and white pixels in the middle two columns with a threshold of 0.01, then created a model with a regularization strength of 1000 with a max_iter of 120. The model had an accuracy of 95.66%.