Classification of Handwritten Symbols Using Support Vector Machines (2021)

Aaryan Kumar

***Abstract* — Recognizing and identifying handwritten characters and symbols has been a topic of interest in the field of machine learning. In the past years, researchers have used different feature extraction and model approaches to improve the task of classifying handwritten symbols. In this experiment, we used a Support Vector Machine model accompanied with HOG feature extraction to classify handwritten symbols. We were able to achieve a maximum of 95% accuracy on our test dataset.**

I. Introduction

Training a computer to recognize and identify handwritten symbols quickly and effectively is a classic problem of machine learning. In this experiment, we develop a model that takes in an image and identifies the symbol from a set of 25 different symbols. These symbols contain greek letters, mathematical operators, and other similar-looking characters.

Since we have been moving away from traditional paper media, researchers have developed lots of ways to digitize handwriting. This exploration process has resulted in the MNIST dataset, which is a set of photos taken of handwritten digits 0-9 that machine learning engineers have used to test their models. One method that has been proposed for identifying different characters is the Support Vector Machine (SVM). Pairing the SVM with the Histogram of Oriented Gradients (HOG) is an effective method of feature extraction and distinguishing different characters. Dalal and Triggs utilized this approach in 2005 when they tried to identify human figures in the MIT database of street photos containing pedestrians [1]. More recently, Chajri, Maarir, and Bouikhalene found that the HOG and SVM combination outperformed the artificial neural network in identifying handwritten mathematical symbols [2]. Our problem is similar to theirs because our training set does contain a lot of mathematical symbols that vary in the size and thickness of the handwriting.

II. Implementation

Using the sklearn libraries, we created a Python implementation of a Support Vector Classifier to distinguish between 25 symbols. We trained this classifier on scanned images of these hand-drawn symbols that were submitted by all the students in our university course. Each student submitted 10 drawings of each symbol to the collective dataset. Each scanned symbol was supposed to be 150x150 pixels, and it needed to be grayscale. This means that no colors were present, and each pixel only contained the value 0-255 of how dark the pixel should be. After some pruning of the data, the training set had 23,502 samples.

TABLE I
List of symbols and their labels

Symbol	#	$	&	%	Ξ	π	Σ	θ	Γ	Θ	O	≤
Class	0	1	2	3	4	5	6	7	8	9	10	11

<	>	≥	«	»	∈	⊂	∉	∈	≅	≈	≠	=
12	13	14	15	16	17	18	19	20	21	22	23	24

To train the classifier, the first step was to determine some features that the classifier would try to recognize in the image samples. Many examples of machine learning solutions to handwritten digit recognition involve taking each image sample, flattening out the image data into a vector of all the pixel values, and then using each pixel itself as a feature. Since the images are 150x150 pixels, each sample would have 22,500 features. If that was sent to the classifier, and get some label predictions, but the classifier would not be accurate due to the variability of the sample quality. Thus instead of using the pixels as features, the HOG library from skimage was used to generate a feature vector for each sample based on the locations of the edges it detects in the pictured object. The length of the HOG feature vector depends on a lot of different factors, and we will discuss our exploration of this issue in the next section, but in our experiment, it was mostly determined by the size of the image sample. Like a neural network, the SVM will project the feature data that it receives into a higher-dimensional space, and then draw a hyperplane boundary between each class it detects, but the problem with an SVM is that it is highly susceptible to the curse of dimensionality [3].

"It becomes too easy to draw a good boundary on the training set, but which has poor generalisation properties" [4]

By resizing the images from 150x150 to 60x60 before sending them into the HOG feature extractor simplified the feature space. This kept the HOG vector as short as possible while still being able to carry the important information for the classifier. This is important for a number of other reasons. A smaller number of dimensions means the training of our classifier is faster, which helps because our training dataset is 23,502 images. Since the model is less complex due to the

smaller number of features, the file size is smaller when the model is exported. The Pickle library was chosen to save our trained classifier, and the output file was compressed with bzip2 so that we could easily transfer it in and out of the Github repository. This saved model is the output of our training python script.

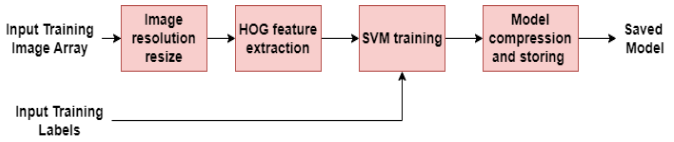

Fig. 1. Program flow of our train_svc.py script.

III. Experiments

The next step was to implement another script that takes the saved model and tests it on a brand new set of images and labels. In order for our model to be able to interpret this new image data, the images must be resized and ran through a HOG feature extractor, exactly the same way we did in the training python script. The testing datasets we used were all much smaller than our training dataset, so this process happens relatively quickly. At the end of our testing script, we print out a percentage of how accurate our model is by comparing the actual labels to our predicted labels and counting up the number of correctly identified images. We created two additional datasets to test our model's performance using this script. The first one had multicolored characters, where they were drawn with black, red, blue, and green colors. This set had 6 images of each character, for a total of 150 images. This data set also contained a few blank images that were totally white or totally black, each labeled with a -1. The second test set focused on the thickness of the characters. There were four different thicknesses tested, and a total of 100 images were in this dataset. This is the mechanism we used to get feedback when we experimented with different model parameters in our training script.

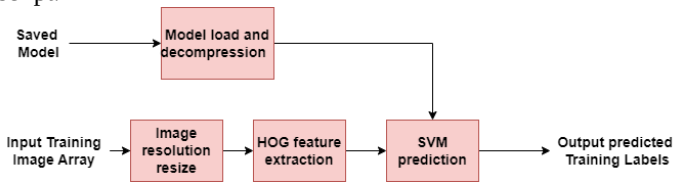

Fig. 2. Program flow of our test.py script.

We used the OpenCV library in python to resize the image samples before extracting their HOG features, but we had to experiment with different image resolutions until we found one that gave us the best performance with the least amount of data. For an image of 150x150 pixels, we obtained a HOG feature vector with length 7569, so our input to the classifier was 23502x7569 – number of samples by number of features. This is better than having all 22,500 pixels in the feature space, but it is still kind of clunky due to the high number of features. As the images provided in the dataset mostly consist of whitespace, a lot of HOG features extracted at full resolution contained noise, or data that was just useless to our model. We found that 60x60 resolution had the best HOG features with no degradation of accuracy. This operation reduced the HOG feature vector length from 7569 to 1089, thus reducing the time taken to extract features and train the model. We tried to slowly increase the resolution by 10 pixels for each run of the model, 50x50, 70x70, 80x80, etc. The performance took a small hit, about 2 or 3 percentage points, each time we raised the resolution by 10 pixels in the range of 60x60 to 100x100. We saw the best performance with no resizing, when we left the images as 150x150, and the saved model was far too large at 100x100, so we did not test beyond that point. When we tried to go lower to 50x50, the performance went down to only 76% correctly identified images, so we decided to keep it 60x60.

Another factor that contributed to the length of our feature vector was the settings that we used to initialize our HOG descriptor. The window size is just inherited from the shape of the image array, so this would be equal to the resolution of each image: (60,60). For the block size, cell size, and stride length, we tried a few different values but it was hard to systematically optimize those parameters because they all depend closely on one another, and we noticed when we made a slight change there it drastically affected the number of features created. The SVM classifier has its own hyperparameters that can be optimized. There are many different kernel functions that we could have used, but we just stuck to the default kernel type which is the radial basis function (RBF) in this sklearn implementation of SVM that we used in our code. All kernel types have the C parameter, but only nonlinear kernels like RBF have the gamma parameter. When C is small, the model does not perform very well because the decision margin between two classes is very large [5]. So we decided to keep it at 100. The gamma parameter is related to how far the points are from one another. When gamma is very large, the points in the feature space need to be very close to one another for the classifier to put them into the same class [5]. For this reason, SVM classifiers with a high gamma value are very susceptible to overfitting. With this in mind, we decided to put the gamma value pretty low at 0.001.

IV. Conclusion

Once we had finalized all our parameter values for the image resizing, HOG descriptor, and SVM classifier, our model correctly identified 90% of images in testing set 1 and 95% in testing set 2. We determined that an image resolution of 60x60 pixels was the smallest amount of pixel data that was required by our model to identify the symbols in the pictures with an accuracy above 90%. If we had more time to work on this project, we would have explored other methods of feature extraction besides HOG. There are lots of variations of HOG that we could try out, but there is a totally different descriptor we encountered in the image processing literature called GIST that outperformed HOG in the study done by Chajri, Maarir, and Bouikhalene. We would also like to spend more time in the future trying out different kernel types for the SVM classifier, because a lot of the time we spent in the experimentation phase was trying out different classification methods like artificial

neural networks, K nearest neighbors, decision tree, stochastic gradient descent, etc to see which one performed the best.

REFERENCES

Examples:

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[2] Y. Chajri, A. Maarir and B. Bouikhalene, "A Comparative Study of Handwritten Mathematical Symbols Recognition," 2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV), 2016, pp. 448-451, doi: 10.1109/CGiV.2016.92.

[3] Drew Wilimitis, "The Kernel Trick in Support Vector Classification," towardsdatascience.com, 2018

[4] Igor F. (https://stats.stackexchange.com/users/169343/igor-f), Are neural networks better than SVMs?, URL (version: 2021-02-18): https://stats.stackexchange.com/q/510100

[5] Soner Yıldırım, "SVM Hyperparameters Explained with Visualizations," towardsdatascience.com, 2020.