International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 01 | January - 2023 Impact Factor: 7.185 ISSN: 2582-3930

Logo Detection using Machine Learning

Kandala Sreeja[1], I.S Harshitha Yadav[2], Gogu. Nagaraju[3]

[1]CSE Department, Sreenidhi Institute of Science and Technology.
[2]CSE Department, Sreenidhi Institute of Science and Technology.
[3]CSE Department, Sreenidhi Institute of Science and Technology.

Abstract - Logo detection is a part of a broader family of object detection. In this paper, we discuss about the methods which help in logo detection using the python libraries like OpenCV, Sklearn, Skimage and machine learning ,computer vision techniques. In the project, we develop a model for logo detection using K nearest Neighbors classifier which detects the logo in the image and is used to predict the logo name. To detect the logo, we transform the images by removing the unwanted data like color of the image by changing it to gray color and detecting the edges of the image, resizing the image and performing feature extraction. Then we use the KNN classifier to train our model to predict the logo names.

Key Words: KNN classifier, OpenCV, canny edge detection, feature extraction, logo detection, machine learning, computer vision.

1. INTRODUCTION

With increasing in brands, companies and organizations as startups, existing ones as market rulers; there is also increment in the scams of fake/cloned variants of products and services from the existing companies. Users are unable to distinguish between the original and a duplicate variant of the services and products provided from one company or brand. So there comes “Logo Detection Using Machine Learning”, with our detection mechanism implementing KNN (K Nearest Neighbors) algorithm, we compare the user provided logo from the original ones and predict the logo name. Our proposed system detects by the original logo based on pixel’s color, design format, service marks and slogan/quotation. . Logo detection is an aspect of image recognition, a computer vision technology that can serve a large number of purposes. Logo detection has many use cases like brand monitoring and social listening, counterfeit detection, product authentication, copyright and trademark compliance, sponsorship monitoring, digital piracy monitoring.

The input data taken is the set of logo images. We preprocess the data, apply edge detection function and feature extraction is done and the resultant image is inputted to the KNN classifier to train the model and it is used to predict the new logos.

2. HARDWARE AND SOFTWARE REQUIREMENTS

Any Windows(or any operating system) PC having:

- 2 GB RAM
- Camera Device
- Internet (Optional, If accessing through online database)
- Enough disk space for the datasets/database
- =/> 1.60 GHz Processor

Software requirements:

- Python
- Windows 7 or later OS

3. METHODS AND IMPLEMENTATION:

The following are the steps to be performed for logo detection:

1. Load the image, convert it to grayscale, and detect edges.
2. Extract HOG(Histogram of oriented gradients) features from our training data to characterize and quantify each logo.
3. Train a machine learning classifier to distinguish between each logo.
4. Apply a classifier to recognize new, unseen logos.

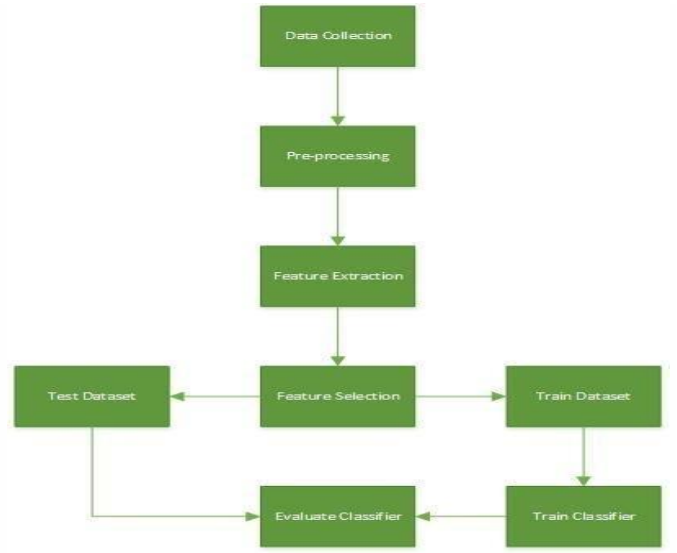

Fig 1. Flowchart

The input data is the set of logo images taken from the internet. We start by importing the required libraries numpy, cv, sklearn, skimage, glob, os.

First we load the images by defining the path of the files by using the python OS module .OS module in Python provides various methods for interacting with the operating system. Os.path.abspath()method returns the pathname to the path passed as a parameter to this function. os.path.dirname() method in Python is used to get the directory name from the specified path. os.path.split() method in Python is used to Split the path name into a pair head and tail. Here, tail is the last path name component and head is everything leading up to that. We define the paths to the training data and testing data.The first path specified is –training, which is the path to where the example logos reside on disk. The second is –test, the path to our directory of testing images we'll use to evaluate our logo classifier.

The glob module is a useful part of the Python standard library. Glob (short for global) is used to return all file paths that match a specific pattern.

We'll also initialize hists and labels, two lists that will hold the HOG features and brand name for each image in our training set, respectively.

Next, we start looping over each of the image paths in the training directory. An example image path looks like this: car_logos/audi/audi_01.pngUsing this image path, we are able to extract the logo brand by splitting the path and extracting the second sub-directory name, or in this case Audi.

cv2.imread() method takes an absolute path/relative path of your image file as an argument and returns its corresponding image matrix. cv2.cvtColor() method is used to convert an image from one color space to another. We perform a bit of pre-processing and prepare the logo to be described using the Histogram of Oriented Gradients descriptor.

```
gray = cv.cvtColor(image,
cv.COLOR_BGR2GRAY)
```

So what we to do is to load the image from disk, convert it to grayscale, and then use our canny function to detect edges in the brand logo.

- ❖ CANNY EDGE DETECTION

```
edged = cv.Canny(gray, low, up)
```

Canny () Function in OpenCV is used to detect the edges in an image.

numpy.median(arr, axis = None) **:** Compute the median of the given data (array elements) along the specified axis.

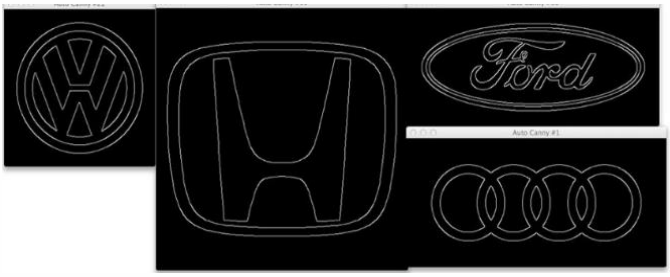

Fig 2. Edge detection of logos.

These contours are then used as a parameter in the Boundingrect function to get the best fitting rectangle of the set of points. This slices the gray array. It basically selects row starting with y till y+h & column starting with x till x+w. So essentially you are selecting height of gray pixels starting up from y and going till y+h (as h denotes height). And selecting width staring from x pixel and going till x+w pixels (w denotes width here).Having various widths and heights for our image can lead to HOG feature vectors of different sizes — in nearly all situations this is not the intended behavior that we want. So, we resize the image**.**

Now that our logo is resized to a known, pre-defined *200 x 100* pixels, we can then apply the HOG descriptor using orientations=9 , pixels_per_cell=(10, 10) , cells_per_block=(2, 2) , and square-root

```
normalization.hist = feature.hog(logo, orientations=9,pixels_per_cell=(10, 10),cells_per_block=(2, 2), transform_sqrt=True, block_norm="L1")
```

A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. Typically, a feature descriptor converts an image of size width x height x 3 (channels) to a feature vector / array of length n. In the case of the HOG feature descriptor, the input image is of size 64 x 128 x 3 and the output feature vector is of length 3780.HOG descriptors are mainly used to describe the structural shape and appearance of an object in an image, making them excellent descriptors for object classification. However, since HOG captures local intensity gradients and edge directions, it also makes for a good texture descriptor. The HOG descriptor returns a real-valued feature vector. The dimensionality of this feature vector is dependent on the parameters chosen for the orientations, pixels_per_cell, and cells_per_block parameters. Implementing this descriptor requires dividing the image into small connected regions called cells, and then for each cell, computing a histogram of oriented gradients for the pixels within each cell. We can then accumulate these histograms across multiple cells to form our feature vector.

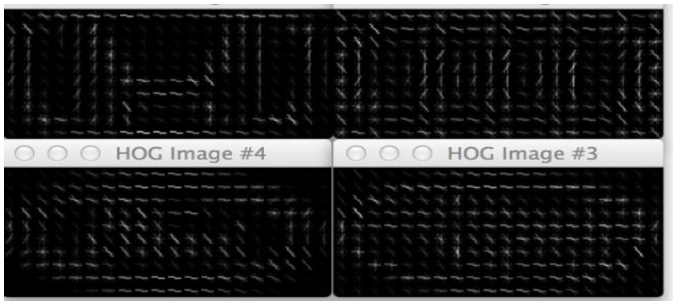

Fig 3. HOG Images

Finally, given the HOG feature vector, we then update our data matrix and labels list with the feature vector and car make, respectively. Given our data and labels we can now train our classifier. To recognize and distinguish the difference between our logo brands, we are going to use scikit-learns KNeighborsClassifier.

```
model = KNeighborsClassifier(n_neighbors=1)
model.fit(hists, labels)
```

KNN algorithm:

The k-nearest neighbor classifier is a type of "lazy learning" algorithm where nothing is actually "learned". Instead, the k-Nearest Neighbor (k-NN) training phase simply accepts a set of feature vectors and labels and stores them — that's it! Then, when it is time to classify a new feature vector, it accepts the feature vector, computes the distance to all stored feature vectors (normally using the Euclidean distance, but any distance metric or similarity metric can be used), sorts them by distance, and returns the top *k* "neighbors" to the input feature vector. From there, each of the *k* neighbors vote as to what they think the label of the classification is. In our case, we are simply passing the HOG feature vectors and labels to our k-NN algorithm and ask it to report back what is the closest logo to our query features using *k=1* neighbors.

Let's see how we can use our k-NN classifier to recognize various logos. We start looping over the images in our testing set. For each of these images, we'll load it from disk; convert it to grayscale; resize it to a known, fixed size; and then extract HOG feature vectors from it in the exact same manner as we did in the training phase. We, then make a call to our k-NN classifier, passing in our HOG feature vector for the current testing image and asking the classifier what it thinks the logo is.

4. RESULTS

Fig 4. Output(Logo detection)

5.CONCLUSION

Logo detection is one of the most useful and popular computer vision application dealing with object localization and classification of a logo. By using the machine learning classifier and computer vision, sklearn techniques and modules, we are able to detect the logo precisely. The latest research on this area has been making great progress in many directions. Future

enhancements can be focused by implementing the project on the system having GPU for faster results and better accuracy. We mainly review the recent advances in deep learning-based logo detection by summarizing classical solutions. In addition, we comprehensively review the commonly used datasets, summarize the related applications of logo detection, and predict future research directions. Although the achievement of logo detection has been effective recently, there is still much room for further development.

ACKNOWLEDGEMENT

We would like to express our special gratitude to our Guide Mr.Prabhakar Vadakattu and Mentor Mr.Devarapu Sreenivasa rao who gave us a golden opportunity to do a wonderful project on this topic. It makes us to do a lot of research and learnt new things. We are really thankful to that.

In addition to that, we would also thank my friends who helped us a lot in finalizing this project within the limited time frame.

REFERENCES

[1] https://learnopencv.com/histogram-of-oriented-gradients/
[2] https://customers.pyimagesearch.com/lesson-sample-histogram-of-oriented-gradients-and-car-logo-recognition/
[3] https://www.geeksforgeeks.org/python-os-path-abspath-method-with-example/
[4] https://www.geeksforgeeks.org/numpy-median-in-python/
[5] https://www.geeksforgeeks.org/k-nearest-neighbours/
[6] https://www.pythonpool.com/cv2-boundingrect/
[7] https://docs.opencv.org/4.x/index.html
[8] https://towardsdatascience.com/fit-vs-predict-vs-fit-predict-in-python-scikit-learn-f15a34a8d39f
[9] https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html
[10] https://www.geeksforgeeks.org/python-opencv-cv2-puttext-method/
[11] https://www.geeksforgeeks.org/python-opencv-cv2-imshow-method/
[12] https://www.geeksforgeeks.org/python-opencv-waitkey-function/
[13] https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature