2021 International Electrical Engineering Congress (iEECON2021)
March 10-12, 2021, Pattaya, THAILAND

Logo Detection with Artificial Intelligent

Karn Pinitjitsamut
Department of Electrical and Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
s5901011620017@email.kmutnb.ac.th

Kanabadee Srisomboon
Department of Electrical and Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
kanabadee.s@eng.kmutnb.ac.th

Wilaiporn Lee
Department of Electrical and Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
wilaiporn.l@eng.kmutnb.ac.th

2021 9th International Electrical Engineering Congress (iEECON) | 978-1-7281-9584-1/20/$31.00 ©2021 IEEE | DOI: 10.1109/iEECON51072.2021.9440236

***Abstract*— A logo detector is a device that detects illegal produced or counterfeit brand products. The focus of this research is to detect the product logo and consider the likeness of the sample product logo. Building the detector's product logo detector, we used the image detection process with a darknet framework and YOLO algorithm. Through this process, the logo of the copyright products is being set as sample product data for having a dataset. Furthermore, OpenCV image classification by DNN module is used in Python language to read our dataset and work in Windows OS platform, to create a Graphical User Interface (GUI) simply including the creating a function to support the various application. The YOLO algorithm will be the main variable in this research. With this precision, we can detect the fake logo with 97% confidence scores and for the authentic logo with 99% confidence scores.**

Keywords: YOLO Algorithm, Logo detection, Darknet, Deep learning, Machine learning, Object detection

I. INTRODUCTION

In the past, commodity logos were performed by people pondering through analyzing and distinguishing the similarities and differences of the product logos. This makes the discrepancy of consideration and might cause arguments in the assessment system, so the logo detector will be a help in establishing standards and reliability in the inspection and analysis. One of the dominant approaches for this technology is deep learning. Deep learning falls under the category of artificial intelligence where it can act or think like a human. Ordinarily, the system itself will be set with thousands of input data to make the 'training' session to be more efficient and faster so this will be discussed in the result of the paper. Machine learning is also a frequent system that has been applied to image classification. Therefore, image classification is going to be occupied with a deep learning system. Classification is easy for humans, but it has proved to be a major problem for machines. It consists of unidentified patterns compared to detecting an object as it should be classified into the proper categories.

In this paper, the YOLO algorithm is used for the training session to works with lots of input data real fast and perform to high accuracy dataset. For the testing part (detection part), a deep neural network, based on OpenCV, is used with Python for image classification. The accuracy of the output data will be studied and compared with the testing image from low to high similarity.

II. YOLO ALGORITHM AND NETWORK STRUCTURES

The first version of the You Only Look Once (YOLO) algorithm [1] is an object detection system targeted for real-time processing started from R-CNN with pre-processing that grid cell is responsible for detecting that object inspired by the GoogLeNet model for image classification [2].

Following with YOLO9000 [3], which improves the accuracy significantly while makes it faster, this system predicts bounding boxes using dimension clusters as anchor boxes. It combines three different scale feature maps, using a high resolution of low-level features and high semantic information of high-level features. By upsampling the features of different layers, objects are detected on three different scale feature layers. As shown in Fig. 1, the input image will be resized to 416x416 till the bottom level downsampling feature map is 13x13, and the two upsampling feature maps are 26x26, 52x52, respectively.

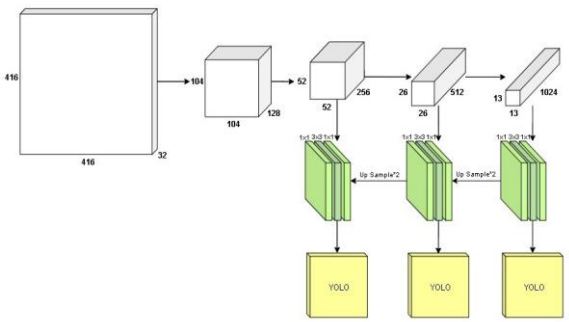

Fig. 1. Yolov3 network architecture.

YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN). In the YOLOv3 paper [4], it proposes a deeper design of a feature extractor known as Darknet-53. As its name suggests, it contains 53 convolutional layers, of which each layer is followed by a "batch normalization" layer. The convolutional layer with a stride of 2 is used to downsamples the feature maps.

Firstly, the cells that the bounding box belongs to is needed. So, the algorithm will divide the input image into a grid of dimensions 416 x 416 (default example), and the batch of the network is 32. As mentioned earlier, the scale of the feature map will be 13 x 13. Then we divide the input image into 13 x 13 cells.

Fig. 2. Center of a ground truth box

Then, the cell of our input image contains the center of the ground truth box to predict the object. As shown in Fig. 2, a red square is a center of a ground truth box which is the yellow box. The following formula describes how the output of the neural network is transformed to obtain bounding box predictions.

$$bx = \sigma(tx) + cx \quad (1)$$
$$by = \sigma(ty) + cy \quad (2)$$
$$bw = exp(tw) * pw \quad (3)$$
$$bh = exp(th) * ph \quad (4)$$

The network predicts 4 coordinates for each bounding box (bx, by, bw, and bh). If the cell is offset from the top left corner of the image by (cx, cy) and the bounding box prior has width and height (pw, ph), then the predictions correspond to (1),(2),(3), and (4).

To enable each grid cell to detect three objects, the concept of anchor boxes is used. The idea of anchor boxes adds one more "dimension" to the output labels by pre-defining several anchor boxes and the dimensions of anchor boxes. Therefore, the dimensions of the anchor boxes are calculated on the training dataset before the training of the network.

III. DATASET OVERVIEWS

In this section, we describe the details of the dataset, including how we collected the image data (Section III.I), how we constructed and collected the bounding box for training, validation, and test splits (Section III.II).

III.I. *Dataset collection*

Choosing the type of data for basic image training, we will select 5 types of original logos including Niki, Adidas, Gucci, Chanel, Puma. Technically, the average training of our data is 1000 images per class, but we will make an experiment example 500 images per type since we must use information that is diverse in each context. Therefore, selected 500 images per type is being used initially from various free image website such as Pinterest website [7]. We select images that are clear in the logo so that the logo is not distorted. The number of images per class is shown in Table 1.

TABLE 1. NUMBER OF TRAINING IMAGES

Classes	Number of images
Adidas	850
Nike	550
Gucci	460
Chanel	390
Puma	400
Total	**2,650**

III.II. *Bounding box creation*

From section II, we would understand the principle of this algorithm very well. Hence, in this section, we will draw the bounding box of the dataset to teach and train where is the logo located. By drawing a bounding box, it can be done in a variety of formats, but in this session, we represent the simple tool call "labelImg" by "Tzutalin" [8]. It is used a Python Qt5 library, which brings Python to create a simple UI.

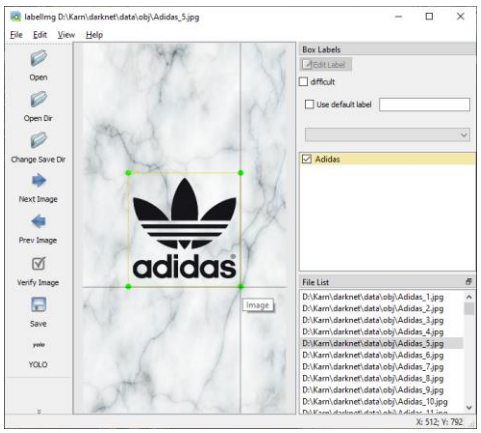

Fig. 3. LabelImg program by Tzutalin.

As the program window in Fig. 3, the file list box shows the input data and the class that we define will appear above the box. For each image we have drawn, the result will be in 5 sets of data: 1) image class 2) center of the bounding box in x coordinate 3) center of the bounding box in y coordinate 4) width of the bounding box 5) high of the bounding box as shown in Fig. 4.

```
0 0.493939 0.499167 0.533333 0.290000
```

Fig. 4. The bounding box coordinate with a class of the logo

IV. EXPERIMENTS

IV.I. *Training images setup*

After the bounding box session, the configuration file (.cfg) for training data will be set. In this case, we use batch = '64' and subdivision = '8' and the pre-train file for the convolution layers is also necessary. After running a command, Yolo will run pass through all training images.

When should we stop training? – at the beginning, we are talking about the confidence of the output image. Likewise, our training image is used to take some time to train dataset output, but we could know where the best time is to stop training by average loss, the lower is better. It either can stop at around below 1% loss (red line) for the big dataset or it reaches the maximum iterations, the simple calculation is classes multiply by 2000 that is 10000 iterations.

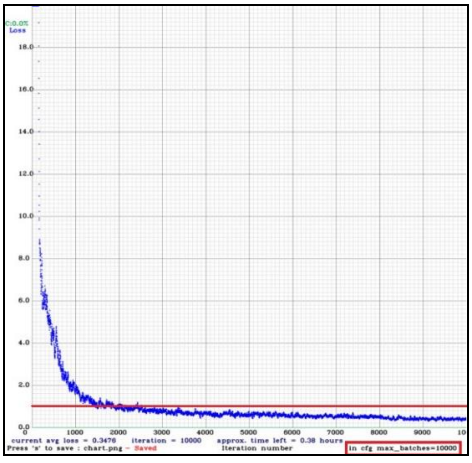

Fig. 5. The mAP loss graph from all iterations in the training process.

Fig. 5 shows, the mAP loss that varies by iterations number. Apparently, the number of losses is initially steady at around 2000 iterations. However, more iterations give the dataset more accuracy. The output result will be in the form of a weight file (.weight) in the directory that has been set before. Lastly, our dataset which comes out as a weight file will be the main core of the research.

IV.II. *Testing with darknet*

After we get the dataset weight file from the previous session that the testing step is as vital as a training process since we consider only 500 images per class. The darknet test is the most basic, it is using the Windows command prompt (cmd) as a command line to run darknet, but there will be a difference in the configuration in the .cfg file. As we mentioned above, in this test, we will use batch and subdivision = '1' run pass the image to find the object (Logo) to be tested.

IV.III. *Testing with OpenCV DNN and Python*

DNN (deep neural network) module is a module of OpenCV. In this section, we will be creating a Python script that can be used to classify input images. Moreover, we can now generate more functions to apply with the dataset such as giving information to the detected image or even UI creation.

V. RESULTS & UI

V.I. *Result*

V.I.I. *Darknet test*

Fig. 6. is a result of the logo detector using a darknet framework with 1.00 accuracy that means the system strongly confirms the logo of Nike in the tester image 100%. Although it is very accurate and precise, the device is quite difficult and inconvenient to use.

Fig. 6. Darknet output test.

Fig.7. OpenCV DNN output test with Python coding.

V.I.II. *OpenCV DNN Test*

This is the output image after pass the DNN module working with Python. As shown in Fig. 7, the percent confidence is more detailed which is 99.87% confidence scores than darknet that cannot config the

output data at all. Therefore, OpenCV DNN is more effective to use as an application.

V.I.III. *Fake logo test*

Toward to our main objective, our logo detector will be used to detect either fake logo or new logo creation, so in this session, the detector will be tested with a fake logo tester. Fig. 8 is a fake logo of the Nike brand so the faker changes the letter of Nike to NKIE and the result is as expected, the detector can detect the fake logo as Nike logo with a 97.7% and 96.67% confidence score, which the authentic logo as 99.45% confidence average scores. Therefore, this detector can detect various counterfeit logos effectively.

Fig. 8. The detection result of a fake logo tester.

V.II. *User Interface (UI)*

Since we use Windows OS to make it easy to display the screen for the user. So, we have created a simple UI for running our program. Likewise, we chose to use a library called "Tkinter", which has the advantage that we can freely design our layout. The output function is designed to be able to display when detecting multiple logo types, displaying the detected image box, and forming a preview of the detected logo, and various detecting details one by one. So, the details of the image will be displayed according to that image. Besides, the program supports manually inserting the dataset by insert a new configuration file and weight file.

VI. CONCLUSIONS AND FUTURE WORK

Our custom dataset can detect the logo in the list correctly and can detect the fake logo in a high prediction score, which we have summarized the results of the experiment in Table 2. So, we will have the confidence scores of each class we have trained and the average score of the authentic logo of 99.45%, and the fake logo ran into 97.09% confidence. So, we could set the threshold at 99% confidence. However, the confidence score is high but the bounding box of some cases of a tester is too large. This resulted case might happen from the test image that has a low resolution than 416x416 or unsuitable size such as the picture that has the main logo is too big compared to the resolution. Therefore, all images in the dataset need to have more quality and, as we mentioned before, each class of images in the list should have more than 1000 images as the developer suggested.

We present the YOLO with darknet framework to train the dataset by the way of YOLOv4 that is announced by AlexeyAB with more Frame Per Second (FPS). Our future work is surely developing this detector to a real-time detector, so that would make our device more updateable than a single picture detection. Moreover, it needs to respond to the user every time of detection that will likely to AI.

TABLE 2. CONFIDENT SCORE BETWEEN AUTHENTIC VS FAKE LOGO.

Classes	**Authentic Logo**		**Fake Logo**	
	Number of images	Average Score	Number of images	Average Score
Adidas	20	99.64%	32	96.68%
Nike	20	99.00%	28	97.53%
Gucci	20	99.86%	6	99.79%
Chanel	20	99.34%	10	93.29%
Puma	20	99.42%	24	98.17%
Total	**100**	**99.45%**	**100**	**97.09%**

ACKNOWLEDGMENT

This research is partly funded by King Mongkut's University of Technology North Bangkok. Contact no. KMUTNB-64-DRIVE-10 and partly support by Electrical and Computer Engineering Department.

REFERENCES

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection
[2] C. Szegedy, W. Liu, Y. Jia. Going deeper with convolutions. CoRR, abs/1409.4842,2014.
[3] Joseph Redmon, Ali Farhadi. YOLO9000: Better, Faster, Stronger. In Computer Vision and Pattern Recognition.
[4] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.
[5] J. Redmon. Darknet: Open-source neural networks inc. http://pjreddie.com/darknet/, 2013–2016.
[6] Aleksey Bochkovskiy. Darknet Framework. https://github.com/AlexeyAB/darknet. [17 Dec 2020]
[7] Pinterest. https://www.pinterest.com.[17 Dec 2020]
[8] Tzuta Lin. labelImg. https://github.com/tzutalin/labelImg [17 Dec 2020]