SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY

CYBER SECURITY – 4TH YEAR

MLCS – ASSIGNMENT

IT17138482

PRAVEENI CHETHANA FERNANDO

08-05-2020

Table of Content

1. Introduction

Machine learning is based on our day to day lifestyle. And it is a combination of computer algorithms that are experienced automatically. Machine learning focuses on the development of computer programs that can access data and use it of their own. So that Machine learning has become a very popular topic these days. Since there is a huge rate of development in Artificial Intelligence and modern network architectures this is high demand. There are two main types of machine learning algorithm types as supervised and unsupervised. But there are semi-supervised machine learning and reinforcement machine learning as well. Machine learning provides to analyze a huge amount of data. When developing a model with machine learning, the results may take time, because it may require additional time to train the gathered data and come up with a more accurate result. But the combination of AI and cognitive technologies can make the processing more effective.

When it comes to machine learning models in these days, people are coming up with new ideas that can be developed through a simple code and an online dataset. But developing a machine learning model for a real-world scenario is not much easy. Under "computer vision" technology, Face detection is so popular. It is the process in which algorithms are developed and trained to properly locate faces or objects (in object detection, a related system), in images. These can be in real-time from a video camera or photographs. An example where this technology is used is in airport security systems. The camera software must first detect, and identify the features before making identification to recognize a face. Likewise, when Facebook makes tagging suggestions to identify people in photos it must first locate the face. On social media apps like Snapchat, face detection is required to augment reality which allows users to virtually wear dog face masks using fancy filters. Another use of face detection is in smartphone face ID security.

Some algorithms detect what is either a face(1) or not a face(0) in an image which is called as classifiers, in Face Detection. Classifiers have been trained to detect faces using thousands to millions of images to get more accuracy. LBP (Local Binary Pattern) and Haar Cascades are two main types of classifiers used in OpenCV. I will be using the latter classifier.

Furthermore, this document will take you to a model that is developed in python, which can be used to detect whether a person is sleeping or not. That can be useful to train a system to take the view taking from a camera that will be placed in vehicles to detect the driver's eyes and keep the driver alarmed of sleeping while driving.

2. Classifiers

A classifier is an algorithm that sorts data into labeled classes or categories of information. A simple practical example is spam filters that scan incoming “raw” emails and classify them as either “spam” or “not-spam.” Classifiers are a concrete implementation of pattern recognition in many forms of machine learning. Classifiers are where high-end machine theory meets application. These algorithms are quite an easy sorting device to arrange or “map” unlabeled data instances into discrete classes. Classifiers are with a set of dynamic rules, which incorporates an interpretation procedure to handle vague or unknown values, all tailored to the sort of inputs being examined. Most classifiers also employ probability estimates that allow end-users to control data classification with utility functions.

In unsupervised learning, classifiers form the backbone of cluster analysis and in supervised or semi-supervised learning, classifiers are how the system characterizes and evaluates unlabeled data.

Common Types of Classification Algorithms in Machine Learning:

Since no single sort of classification is acceptable for all datasets, a huge toolkit of off-the-shelf classifiers is available for developers to experiment with.

- Cascade classifiers
- Linear Classifiers (such as Logistic Regression, Naive Bayes Classifier, Fisher's Linear Discriminant, Perceptron)
- Support Vector Machines
- Decision Trees (including Boosted Trees and Random Forest)
- Neural Networks
- Quadratic classifiers
- Kernel estimation (such as Nearest Neighbor)
- Boosting (meta-algorithm)
- Learning vector quantization

From all the above-mentioned classifier types, cascade classifiers are used in this model.

2.1 Cascade Classifier

The cascade classifier consists of a set of stages, where each stage is an ensemble of weak learners. These are called decision stumps. Each stage is trained to employ a technique called boosting. Boosting provides the power to coach a highly accurate classifier by taking a weighted average of the choices made by the weak learners.

Each stage of the classifier labels the region defined by the present location of the window as either positive or negative. When an object was found, it indicates as Positive and negative indicates when no objects were found. If the label is negative, the classification of this region is complete, and therefore the detector slides the window to the subsequent location. If the label is positive, the classifier passes the region to a subsequent stage. The detector reports an object found at the present window location when the ultimate stage classifies the region as positive. To reject negative samples as fast as possible, the stages are designed. The assumption is that the vast majority of windows do not contain the object of interest. Conversely, true positives are rare, and price taking the time to verify.

- A *true positive* occurs when a positive sample is correctly classified.
- A *false positive* occurs when a negative sample is mistakenly classified as positive.
- A *false negative* occurs when a positive sample is mistakenly classified as negative.

To work well, each stage within the cascade must have a false-negative rate. If a stage incorrectly labels an object as negative, the classification stops, and you can't correct the error. But each stage can have a false-positive at a higher rate. Even if the detector incorrectly labels a non-object as positive, you'll correct the error in subsequent stages. Adding more stages reduces the general false-positive rate, but it also reduces the general true positive rate.

Cascade classifier training requires a group of positive samples and a group of negative images. You must provide a group of positive images with regions of interest specified to be used as positive samples. You can use the Image Labeler to label objects of interest with bounding boxes. The Image Labeler is there to output a table to use in positive samples. You also must provide a group of negative images from which the function generates negative samples automatically. To achieve acceptable detector accuracy, set the number of stages, feature type, and other functional parameters.

2.2 Haar Cascade

This is based on the Haar Wavelet technique to analyze pixels in the image into squares by function. This uses machine learning techniques to get a high degree of accuracy from what is called “training data”. This uses “integral image” concepts to compute the “features” detected.

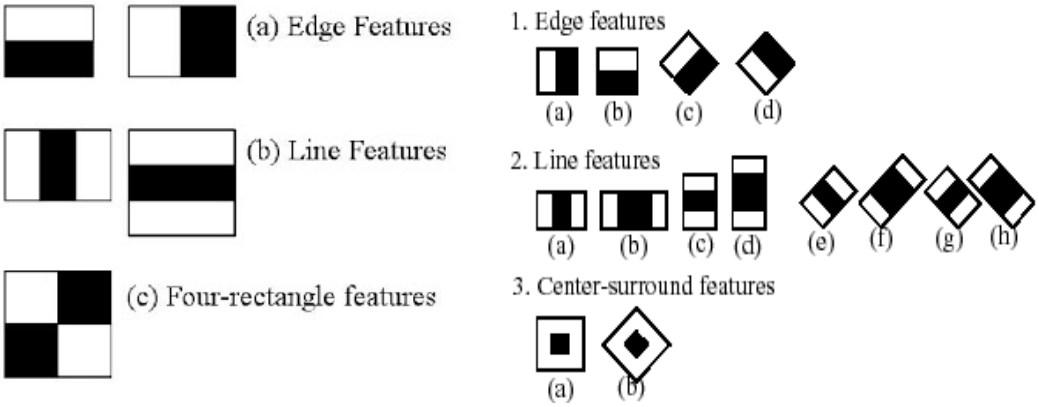
When it comes to object detection in an image or a video, Haar cascade is the most popular algorithm used in machine learning. There was a paper published about rapid object detection using a boosted cascade of simple features by Paul Viola and Michael Jones in the year 2001. That is a huge approach to machine learning where cascade function is trained with the use of positive and negative images. After that, it is used to detect objects in other images.

The algorithm has four stages:

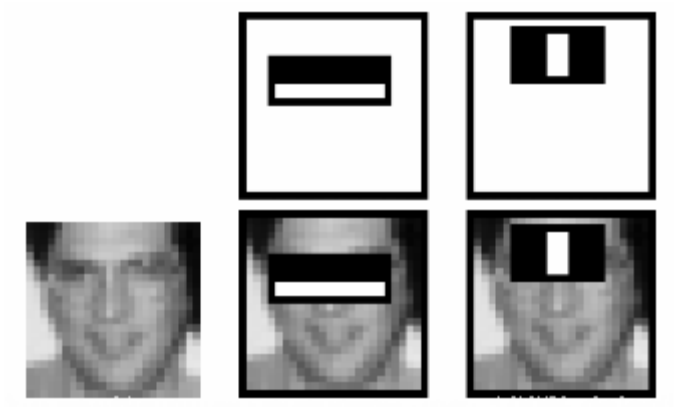
1. Haar Feature Selection
2. Creating Integral Images
3. Adaboost Training
4. Cascading Classifiers

It is well known in detecting faces and body parts in an image but can be trained to identify almost any object.

In face detection kinds of models, initially, the algorithm needed a lot of positive images and negative images without faces to train the classifier. After that, the features can be extracted. As the first step, the model needs to collect Haar Features. Adjacent rectangular regions at a specific location in a detection window, and sums up the pixel intensities in each region and calculates the difference between these sums gives the haar feature as the answer.



To make this super-fast, you can use Integral Images. But most of them are irrelevant among all these features we have calculated. Consider the image below as an example. Two good features are there in the top two rows. The first feature seems to focus on the property that the region of the eyes is often darker than the nose and cheeks. The second feature relies on the property that the eyes are darker than the bridge of the nose. But if the windows are applying on cheeks or any other place is trivial.



So how do we can select the best features out of 160000+ features? This is adept using a concept called Adaboost. This is a method that selects the best features and trains the classifiers that are using them. This constructs a strong classifier as a combination of the weighted weak classifier. The process is as follows. A window of the targeted size is moved over the input image, and for each subsection of the image and Haar features are calculated during the phase of detection. Then it is comparing with a learned threshold that separates non-objects from objects. Since each Haar feature is only a 'Weak classifier' (its detection quality is marginally better than random guessing), a large number of Haar features are required to classify an item with adequate accuracy and therefore grouped into cascade classifiers to form a strong classifier.

3. Development

The training data used in this project are XML files called:
haarcascade_frontalface_default.xml
haarcascade_eye_tree_eyeglasses.xml

Here in this project, the detectMultiscale module from OpenCV is used. What this does is create a rectangle with coordinates (x,y,w,h) around the face detected in the image. This contains code parameters that are the most important to consider.

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
    cv2.putText(img,'FACE',(x,y-10),cv2.FONT_HERSHEY_SIMPLEX,0.5

    face_img=gray[y:y+w,x:x+w] # : ---> sita
    eyes=eye_clsfr.detectMultiScale(face_img)

    detected=0


    for(ex,ey,ew,eh) in eyes:
        cv2.rectangle(img,(x+ex,y+ey),(x+ex+ew,y+ey+eh),(255,0,0
        cv2.putText(img,'Eyes',(x+ex,y+ey-10),cv2.FONT_HERSHEY_S
```

3.1 Methods used

cv2.CascadeClassifier()

.detectMultiScale()

Cascade classifier is a class to detect objects in a video stream. Likewise we can use **.load** or **.detectMultiScale** functions as well. **.load** is to load a .xml classifier file. It can be either Haar or an LBP classifier. And **.detectMultiScale** is to perform detection.

cv2.VideoCapture()

This is the default constructor capture the video. And can be overloaded as preferred. In this project, this method is used with an index. This allows opening a camera to capture a video. If you want to open the default camera, then just pass '0' as the index.

cv2.cvtColor()

This **c**onverts an image from one color space to another.

The function converts an input image from one color space to another. In the case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

The conventional ranges for R, G, and B channel values are:

- 0 to 255 for CV_8U images
- 0 to 65535 for CV_16U images
- 0 to 1 for CV_32F images

In the case of linear transformations, the range does not matter. But in case of a non-linear transformation, an input RGB image should be normalized to the proper value range to get the correct results, for example, for RGB → L*u*v* transformation. For example, if you have a 32-bit floating-point image directly converted from an 8-bit image without any scaling, then it will have the 0..255 value range instead of 0..1 assumed by the function. So, before calling **cvtColor**, you need first to scale the image down.

cv2. COLOR_BGR2GRAY

This is to convert the colored image to grayscale.

cv2.FONT_HERSHEY_SIMPLEX

This denotes the font type.

cv2.rectangle()

This is a method used to draw a rectangle on any image.

cv2.putText()

This method is used to draw a text string on any image.

cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])

cv2.imshow()

The function imshow displays an image in the specified window. If the window was created with the **cv:: WINDOW_AUTOSIZE** flag, the image is shown with its original size, however, it is still limited by the screen resolution. Otherwise, the image is scaled to fit the window. The function may scale the image, depending on its depth.

cv2.waitKey()

Whenever you call this method, the system waits for a pressed key.

The function waitKey waits for a key event infinitely (when delay≤0) or delay milliseconds when it is positive. Since the OS has a minimum time between switching threads, the function will not wait exactly delay ms, it will wait at least delay ms, depending on what else is running on your computer at that time. It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

4. Applicability of the model

There is a higher rate of accidents caused by drowsy drivers. But no preventive method has been found so far. And this is developed as a solution for that. This is a face detection system that can identify whether your eyes are open or closed. Simply this can detect whether you are sleeping or not. In this model, it can identify your fully opened eyes, and display them as not sleeping. And if your eyes are closed or widely opened it displays as sleeping.

This model can be improved to use in-vehicle cameras that can check the drivers' faces all the time and keep the driver awaken while driving.

5. Useful Links

https://deepai.org/machine-learning-glossary-and-terms/classifier

https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623

https://expertsystem.com/machine-learning-definition/

https://en.wikipedia.org/wiki/Machine_learning

https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

https://en.wikipedia.org/wiki/Cascading_classifiers

https://towardsdatascience.com/computer-vision-detecting-objects-using-haar-cascade-classifier-4585472829a9

https://www.geeksforgeeks.org/python-tutorial