Eye and occlusion detection for liveness detection

*Computer Vision Group Project

Parth Kalkar
Dept. of Computer Science
Innopolis University
Innopolis, Russia
p.kalkar@innopolis.university

Bekhruz Nutfilloev
Dept. of Computer Science
Exchange Student, Innopolis University
Innopolis, Russia
b.Nutfilloyev@innopolis.university

***Abstract*—This electronic document is a group project report. It defines the overall components of the project, including introduction, methodology, observations, and analysis.**

Keywords—eye detection, occlusion detection, computer vision, CNN, YOLO, face detection

I. Introduction

Our project focuses on automatically identifying eyes and occlusion for liveness detection. In face detection and face recognition applications, computer vision plays a significant role. Face recognition is the technology that allows computers and machines to match images containing people's faces and their identities. In this section, computer vision algorithms detect facial features in images.

Face detection is used in biometrics, often as a part of (or together with) a facial recognition system. It is also used in video surveillance, human-computer interface, and image database management. Liveness detection is a technique where an algorithm securely detects whether the source of a biometric sample comes from a fake representation or is a live human being.

II. Related Work

In recent decades, researchers considered liveness detection as an intriguing computer vision problem in several instances. In this section dedicated to a brief literature review, we focus on three reputable papers presenting similar projects.

The first example is a chapter from a book written in 2008 on liveness detection for face recognition [1]. Most current face recognition systems are based on intensity images and equipped with a generic camera. An anti-spoofing method without additional devices will be preferable since it could be easily integrated into the existing face recognition approach and system.

This chapter briefly introduces a blinking-based liveness detection approach to prevent photograph spoofing. It requires no extra hardware except for a generic web camera. Eyeblink sequences often have a complex underlying structure. We formulate blink detection as inference in an undirected conditional graphical framework and can learn compact and efficient observation and transition potentials from data. An easily-computed discriminative measure derived from the adaptive boosting algorithm is developed for quick and accurate recognition of the blinking behavior, eye closed, and then smoothly embedded into the conditional model.

The advantages of the eyeblink-based approach lie in the following:
1. It can complete in a non-intrusive manner, generally without user collaboration.
2. No extra hardware is required.
3. Eyeblink behavior is the prominently distinct character of a live face from a facial photo, which would be much more helpful for liveness detection only from the generic camera.

The scholars approached the problem as follows: An eyeblink behavior could be represented as a temporal image sequence after being digitally captured by the camera. One typical method to detect blink is to classify each image in the sequence independently as one state of either closed eye or open eye, for example, using Viola's cascaded Adaboost approach, like face detection (Viola & Jones, 2001). The problem with this method is that it assumes all of the images in the temporal sequence are independent. The neighboring images of blinking are dependent since the blink is a procedure of the eye from opening to closing, then to opening.

The temporal information is ignored for this method, which may be very helpful for recognition.

The authors say this independence assumption can be relaxed by disposing of the state variables in a linear chain. For instance, an HMM (the hidden Markov model) (Rabiner, 1989) models a sequence of observations by assuming that there is an underlying sequence of states drawn from a finite state set. The features of the image could be regarded as observations, and the eye state label is for the underlying states. An HMM makes two independent assumptions to model the joint probability tractably. It assumes that each state depends only on its immediate predecessor and that each observation variable depends only on the current state, depicted in Fig. 1(a). However, on the one hand, the generative-model-based approaches should compute a model of p(x), which is not needed for classification anyway. On the other hand, for our task of eyeblink recognition, the two independence assumptions are too restrictive since there exist dependencies among observations and states, which will benefit blink detection, in particular when the current observation is disturbed by noise such as highlighted in the eye region, variation of

glasses' reflection. We model eyeblink behaviors in an undirected Conditional Random Field framework, incorporated with a discriminative measure of eye states to simplify inference complexity and simultaneously improve performance. One of the advantages of the proposed method is that it allows us to relax the assumption of conditional independence of
the observed data.

Another approach was implemented by Yizhang Xia and Bailing Zhang [2] using a multi-task convolution neural network for face occlusion detection.

Contrary to the previous studies, a recent entry by Saptarshi Chakraborty and Dhrubajyoti Das gave a complete overview of face liveness detection and the use of state-of-the-art Deep Learning models to solve this task [3].

III. METHODOLOGY

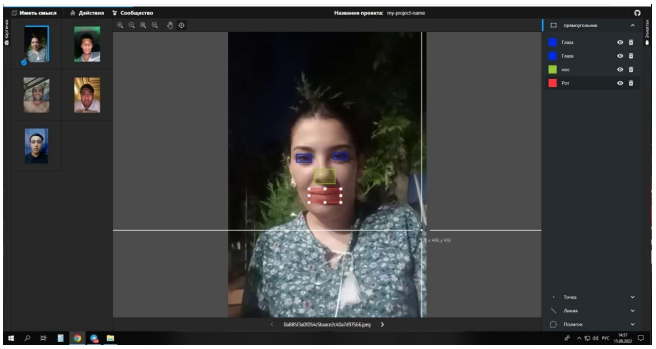
Supposedly, various filters are used in the pre-authentication period. The first of these is blur detection, followed by eye detection and then occlusion detection. Eye detection is used because if a person's eyes are closed, he does not want to undergo identification or is being forced to do so. Therefore, it is advisable to have him get his eyes checked. That is why all smartphones have eye detection technology. Next is occlusion. As mentioned above, the half-view of the face, the nose, and the lips are not visible, which leads to a low identification result.

CNN technology is used for eye detection. To do this, you will first need to use facial recognition technology (which is optional). We want to use the MTCNN face recognition system. Next, it will be necessary to determine the eye. For this, defining a hundred points (optional) is essential. Next, it is required to determine the eye area, and we will mark the eye according to the 0 and 1 classification.

For occlusion, we use only *yolov5*. First of all, pictures with blocked faces and regular faces are collected. In the next step, two eyes, a nose, and a mouth are defined. Then, the symbol in the following sequence is selected. 0 0 1 2. These characters are checked in all pictures. If any sign is not found, this image is defined as not passing the filter in occlusion detection.

A. *Dataset creation.*

First, we need pictures of people with their eyes open and eyes closed. We compiled it ourselves (the GitHub repo hosts all datasets). Furthermore, we divided them into separate folders for two classes. The next step is to cut out the eyes in each picture. First, it is necessary to identify the face from the images and identify the eye in the identified face. For this, we used the Dlib face 68 landmark model. The points between 36:42 correspond to the left eye, and the points between 42:48 correspond to the right eye. These points separated the eyes, and we predicted them through the CNN model.



The image above shows data labeling:

For occlusion, we need random face images and images of certain objects with their eyes and mouth occluded. We also collected these manually, all these datasets are listed in the GitHub repository. We labeled them using makesense.ai and “LabelImg” built-in programs. These are

```
names: ['eye,' 'nose,' 'mouth']
```

by three classes. Furthermore, the train part was executed based on the Yolov5 architecture.

B. *Dataset Explanation*

We collect images with eyes open and closed for eye detection. we have compiled and uploaded the datasets to the project's GitHub. To make the most of our dataset, we work with only one eye, i.e., we can turn the right eye into a left eye by simply flipping it, and we only use one eye during the training process. We could have also used Kaggle datasets for eye detection. Eye Gaze, Eye-dataset, Female and male eyes

For occlusion, we need to label the eyes, nose, and lips. Of course, we can use “LabelImg” or similar GUIs. There is no problem with the codes; we must improve the presentation. Above is how We can do it; now we will try to collect datasets for it; we will take simple pictures from Kaggle and try to take a picture of it with a handphone and any object, slightly blocking certain parts of the face. Naturally, our 0 0 1 2 classification in the following images is incorrect. Therefore these should be set to false. For this part, we will use the yolov5 module medium-level model.

C. *Training*

For the simplicity of this tutorial, we train the small parameter size model YOLOv5s6, though bigger models can be used for improved results. Different training approaches might be considered for different situations, and here we will cover the most commonly used techniques.

When having a large enough dataset, the model will benefit most by training from scratch. The weights are randomly initialized by passing an empty string (‘ ‘) to the weights argument. The following command induces training:

```
$ python train.py --img 640 --batch 32 --epochs 100 --data custom_data.yaml --weights yolov5s.pt
```

D. *CNN*

```
inputs = Input(shape=(26, 34, 1))

net = Conv2D(32, kernel_size=3, strides=1, padding='same', activation='relu')(inputs)
net = MaxPooling2D(pool_size=2)(net)

net = Conv2D(64, kernel_size=3, strides=1, padding='same', activation='relu')(net)
net = MaxPooling2D(pool_size=2)(net)

net = Conv2D(128, kernel_size=3, strides=1, padding='same', activation='relu')(net)
net = MaxPooling2D(pool_size=2)(net)

net = Flatten()(net)

net = Dense(512)(net)
net = Activation('relu')(net)
net = Dense(1)(net)
outputs = Activation('sigmoid')(net)
```

E. *YOLO Configuration files*

The model-configurations file dictates the model architecture. Ultralytics supports several YOLOv5 architectures, named P5 models, which vary mainly by their parameters size: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), YOLOv5x (extra large). These architectures are suitable for training with an image size of 640*640 pixels. An additional series optimized for training with a larger image size of 1280*1280, called P6 (YOLOv5n6, YOLOv5s6, YOLOv5m6, YOLOv5l6, YOLOv5x6). P6 models include an extra output layer for the detection of larger objects. They benefit the most from higher-resolution training and produce better results.

Ultralytics provides build-in, model-configuration files for the above architectures, placed under the ‘models’ directory. If you’re training from scratch, choose the model-configurations YAML file with the desired architecture (‘YOLOv5s6.yaml’ in this tutorial), and just edit the number of classes (NC) parameter to the correct number of classes in your custom data.

IV. GitHub

https://github.com/YoshlikMedia/CV-Final-exam

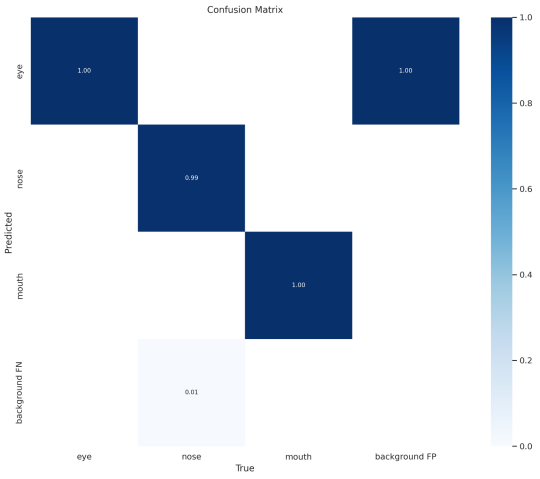
V. EXPERIMENTS & EVALUATION

We tried training and testing our model with the following parameters and got the respective accuracy.

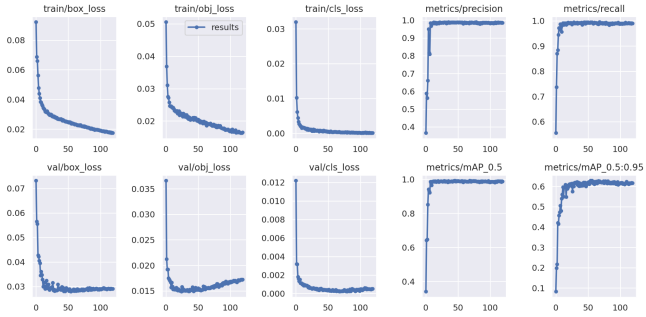
batch_size	Learning rate	Image size	Accuracy
256	0.001	26x34	99.898
128	0.001	256x258	96.989
64	0.0001	28x28	98.992

VI. ANALYSIS & OBSERVATIONS

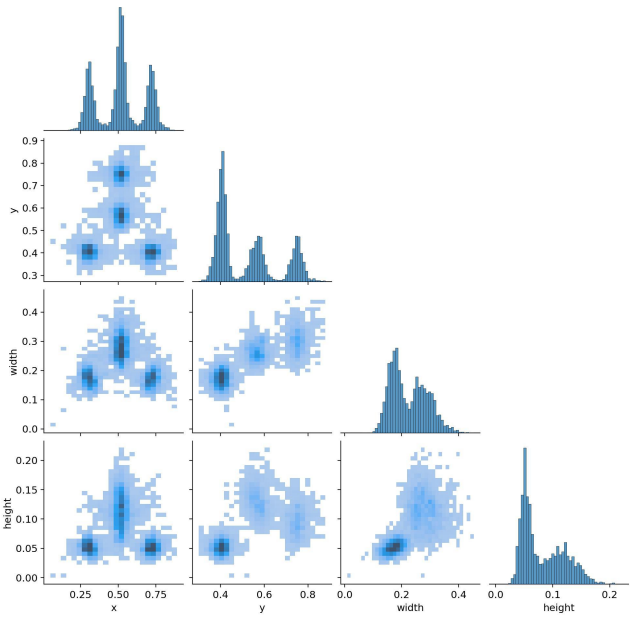
After conducting a series of experiments based on the above-mentioned values, we got the following results:



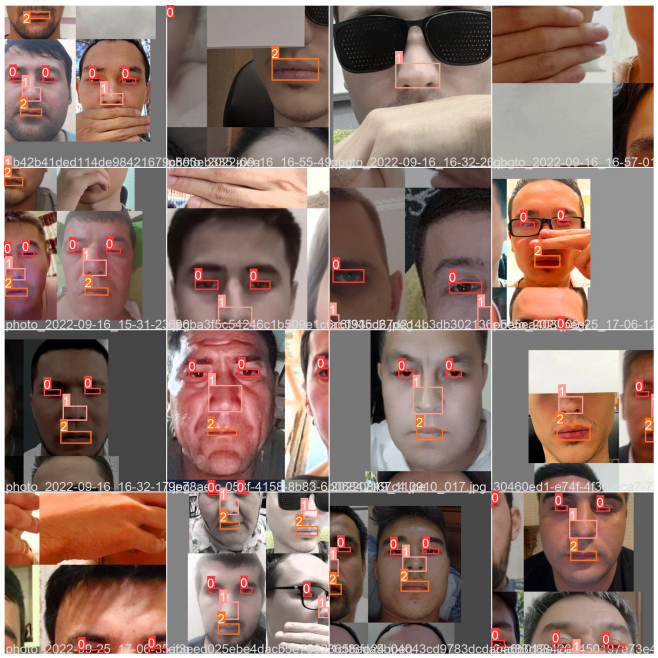
Occlusion metrics



labels_correlogram

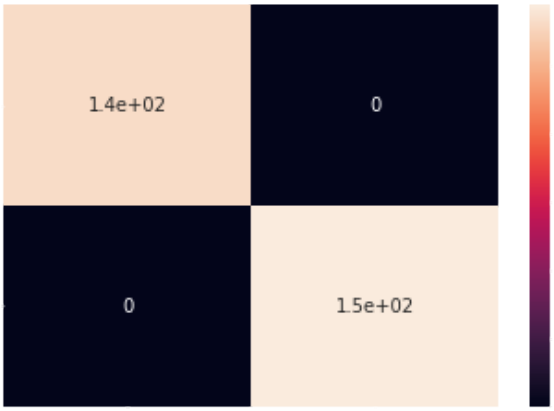


Labels Metrics

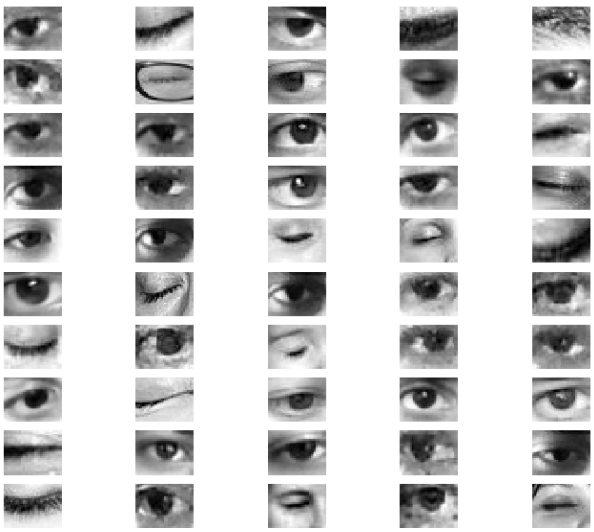


Some results

Eye detection results



confusion matrix



VII. CONCLUSION

Finally, we solved the task we aimed for when we started this project. We learned a lot about Face detection, eye detection, and occlusion detection and their extensive use case for liveness detection.

We created a docker image of this product to be accessed easily. We plan to develop a full-fledged application and host it for further use.

REFERENCES

[1] Pan, Gang, Zhaohui Wu, and Lin Sun. "Liveness detection for face recognition." Recent advances in face recognition (2008): 109-124.

[2] Yizhang Xia, Bailing Zhang and F. Coenen, "Face occlusion detection based on multi-task convolution neural network," 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2015, pp. 375-379, DOI: 10.1109/FSKD.2015.7381971.

[3] Chakraborty, Saptarshi, and Dhrubajyoti Das. "An overview of face liveness detection." arXiv preprint arXiv:1405.2227 (2014)

[4] Arie, L. G., PhD. (2022, April 1). The practical guide for Object Detection with YOLOv5 algorithm. Medium. https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843