Doubango AI

Passive face, fingerprint and iris liveness detection (anti-spoofing) using deep learning

https://github.com/DoubangoTelecom/FaceLivenessDetection-SDK

Table of Contents

This is a short technical guide to help developers and integrators take the best from our Liveness detection SDK (anti-spoofing). You don't need to be a developer or expert in deep learning to understand and follow the recommendations defined in this guide.

1 Intro

Question: Which one of these next 3 pictures is genuine?

Response: **None. They are all spoofs.**

The original images are from wikimedia and can be found here, here and here. We recaptured the original images using a Galaxy S10+ with 4K resolution on an iMac with 4K retina display. Then, we cropped the images. The entire images can be found here, here and here.

These kind of high resolution (quality) Print-Attacks are hard to detect and most liveness detectors would fail the test.

Our passive (frictionless) face liveness detector uses **SOTA** (State Of The Art) deep learning techniques to spot both `Print-Attack` and `Replay-Attack`. You can freely test our implementation with your own images at https://www.doubango.org/webapps/face-liveness/

We're working to package and release the source code of the SDK. This document will be updated to include the API reference and a Getting Started guide.

Insert demo video here

2 Supported attacks

We support both Print and Replay attacks.

2.1 Print-Attack

It's when a user shows a picture to the camera.

YouTube video showing Print-Attack: https://www.youtube.com/watch?v=QVdLJlSaeHM

2.2 Replay-Attack

It's when a user shows a video to the camera.

YouTube video showing Replay-Attack: https://www.youtube.com/watch?v=4Z8VRTS8WrA

3 Accuracy

We have a perfect score on multiple public datasets (**NUAA**, **CASIA FASD**, **MSU**...). These datasets are too small to train a model but large enough to help design a deep learning model.

Our deep learning model was trained on our own very challenging dataset. We have **99.62% accuracy on the validation set and 99.27% on the test set.**

The model was developed using Keras with Tensorflow backend and has **80M parameters.**

You can test the accuracy with your own images at https://www.doubango.org/webapps/face-liveness/.

4 Why liveness detector is required for facial recognition systems?

A facial recognition system without a liveness detector is useless. The liveness detector is required to make sure the face to recognize is from a real (live) person and not from a photo or video.

5 Passive versus Active liveness detection

5.1 Passive liveness

A single selfie-like image is needed to compute a liveness score (within **[0 - 100]%**). The decision is solely based on intrinsic elements like the micro textures on your skin, the light reflection on your eyes, the direction of the edges...

5.2 Active liveness

Active liveness detector will ask the user to perform some actions like: smile, blink your eyes, turn your head left/right/top/bottom... It requires active user cooperation and take time to validate. You'll probably end with low conversion rates as most of your customers will exit the application before the end of process.

5.3 Why passive is better?

This section explain why passive detectors are better.

5.3.1 Accuracy

Passive liveness is more accurate because it's solely based on intrinsic information. Intrinsic information is almost invariant and easy to predict.

Try our online web demo at https://www.doubango.org/webapps/face-liveness/ with your mobile phone using your camera to take selfies.

5.3.2 Faster enrollment

As explained above, a passive liveness detector needs a single selfie-like image while active detector will ask the user to perform multiple actions. Our passive liveness detector can run at **45 frames per second** (server-side) which means it takes only **22 milliseconds per user**.

5.3.3 Frictionless

Unlike the active liveness detectors, passive detectors will just ask for a single selfie-like image. You can use the same image as the one already used by your facial recognition system to authenticate the user.

5.3.4 Harder to break

As explained above, active liveness detector will ask you to perform some actions and this makes it easy to break as you're already giving fraudsters information on how spoofs are detected.

On the other side, passive liveness detector will not ask the user to perform any action. So, no indication on how to break the system. Even better, the fraudsters don't know that it exists. Passive liveness detection is based on micro texture analysis and very hard to break even with high resolution cameras (4K+) and displays (Retina+).

5.3.5 Easier to integrate to an existing application

Passive liveness detector needs a single image to compute a score. You don't need to change your user interface to integrate the SDK. Put the SDK on the server side, the image used by your facial recognition system to authenticate the user is a good candidate for the liveness detector.

5.3.6 Less bandwidth and CPU usage

Active liveness detector requires multiple frames to compute a score. This is CPU intensive if the frames are analyzed on the mobile device itself. Some active liveness detectors run on a remote server which means each frame is sent over the network, in such scenario the issues are multiple: bandwidth usage, scalability, costs...

A passive liveness detector needs a single image.

6 Recommendations

Use passive liveness as first guard, then fallback to active liveness detection when the user is flagged as spoof.

7 Improving the recall score

The precision score is very high (Spoof images correctly flagged as Spoof) but the recall score may be low (Genuine images incorrectly flagged as Spoof) if your images do not match some requirements. This section explains how to improve the recall score by controlling the kind of images the user will provide. More info about precision and recall scores at https://en.wikipedia.org/wiki/Precision_and_recall.

Our web demo at https://www.doubango.org/webapps/face-liveness/ is provided in unconstrained environment which means any kind of images could be used to test our implementation. Trying with random images you'll find it hard to pass the liveness test. Most of your images will be rejected as being spoofs. This is normal, most images on the internet are heavily edited (photoshopped) or recaptured. Also, the face pose, lighting and expression should match ISO/IEC 19794-5 recommendations.

These kind of mistakes can be easily avoid in controlled environment. A controlled environment may be your mobile application, you have the control of the camera.

7.1 The 7 commandments

These are the 7 commandments to pass the liveness test. **The first commandment is a must and the others are optional.**

7.1.1 Face must be large and close

If there is one commandment you have to respect it would be this one. We recommend having the face **very close to the camera lens** and **occupying between 80% and 90% the surface of the image**. The face should be **at least 600px in width and height**.

To determine the liveness of the face we have to analyze the texture of your skin, the light reflection on your eyes, the direction of the edges and other parameters.

YouTube video showing how to take a perfect selfie: https://www.youtube.com/watch?v=fIw0Hk7oHqk

7.1.2 Do not trust online images

Most images on the web are heavily edited (photoshopped) or recaptured. So, don't assume an image is spoof or genuine unless it's yours our created in controlled environment (e.g. from a Liveness dataset).

7.1.3 Selfie-type image (front facing) with neutral expression

The deep learning model was trained on selfie-type images (front facing). If the face is not looking at the camera lens, then we'll most likely flag it as spoof.

The face should have neutral expression.

7.1.4 Use high resolution images

Higher the resolution easier it would be to pass the liveness test. We recommend **at least HD resolution (1280x 720).** Off course you must not try to enlarge (resize) a small image to match the recommended resolution.

The camera should have **3MP resolution or more**.

You can for example check this image or this one to see the kind of quality we expect.

7.1.5 Do not use grayscale images

The model was trained with RGB images, grayscale images will be rejected as spoofs. You may even not reach the liveness module as the face detector was also trained on RGB images.

7.1.6 Use reasonable compression ratio

Make sure your images (PNG, JPEG, BMP...) are not heavily compressed. We use the edges as a metric. Heavily compressed images are smooth and will likely be flagged as spoofs.

7.1.7 Disable "Beauty" filter

On most Android phones this filter is enabled on default. This is specially true with Samsung devices (e.g. Galaxy S10+). This filter smoothen the skin and the selfie may be flagged as spoof if the smoothing threshold is too high.

We recommend disabling all filters (beauty, red eye remover, background blur...).

Disabling the filters is a recommendation not a requirement.

8 Integration with your existing application

Nothing special to change. Run the liveness detector on a server and use a single selfie-like image to compute a liveness score. The image used by your facial recognition system to authenticate the user is a good candidate for the liveness detector.

9 Error and success codes

The success codes are prefixed with `s_` and the error codes with `e_`.

Our web demo at https://www.doubango.org/webapps/face-liveness/ and SDK on Github will return one of these codes:

`s_genuine:` The face was analyzed and is most likely genuine. In this case the liveness score is within **[98 - 100]%**.

`s_spoof:` The face was analyzed and is most likely spoof. In this case the liveness score is within **[0 - 50[%**.

`s_disputed:` The face was analyzed but we are not sure if it's genuine. Ask the user to try again or review it. In this case the liveness score is within **[50, 98[%**.

`e_back:` The image contains multiple faces and this one is considered as being at the background and no liveness check was done on it.

`e_too_small:` The face is too small (width or height is < 64px). No liveness analyze was done on it.

`e_not_selfie:` The face is not a selfie (front facing). For now we don't check if a face is a selfie or not. No liveness analyze was done on it.

10 Known issues

Some known issues to be fixed in the next versions.

10.1 Masks

The face will be flagged as spoof if the person is wearning a mask. You may even not pass the face detector.

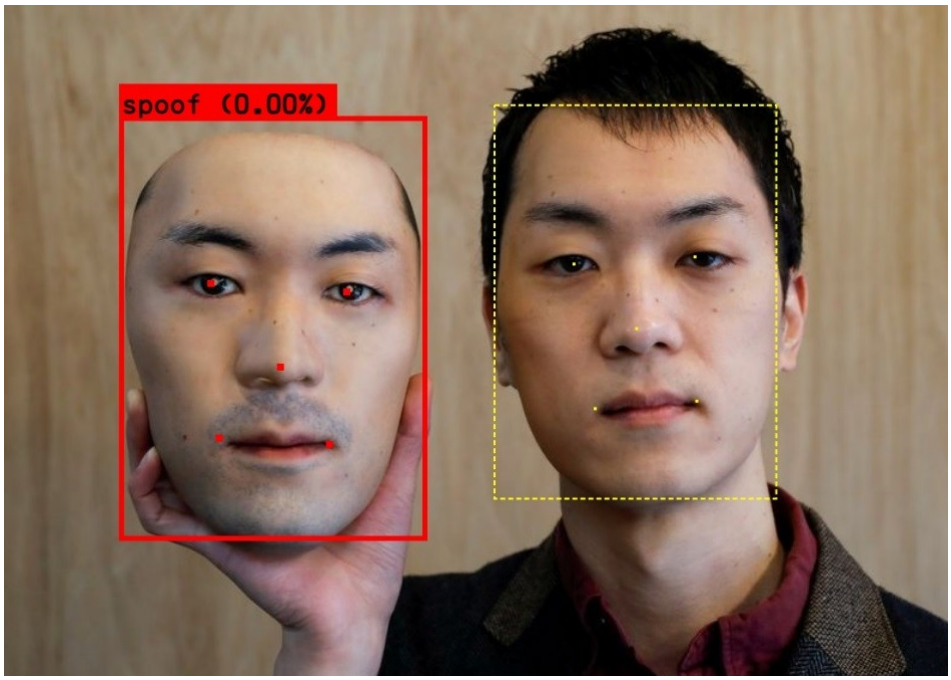
10.2 Very dark skin

The face detector is slightly less accurate on people with very dark skin. The face will be correctly detected if the selfie is taken as explained in the previous sections. The liveness detector isn't impacted with this minor issue.

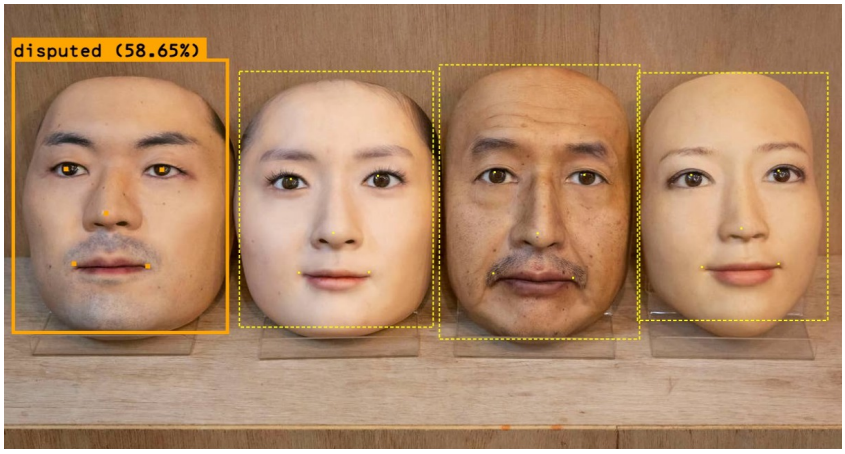
10.3 3D realistic masks

Our liveness detector support both 2D Print-Attacks and Replay-Attacks. Our dataset contains very few 3D realistic masks, the number of samples isn't enough to claim we fully support such attacks.

If you try with the next image (Copyright here) using our online demo web app, the detector will correctly flag it as spoof. This one is also correctly flagged.



But, if you try with the next image (Copyright here) you'll get a 58% liveness score.



This issue will be fixed in the coming versions if we manage to build a large database with 3D masks. Unlike other companies we are very careful and will claim full support for 3D masks only if we can train on a large dataset and report a score > 98%.

Do not panic, a fraudster has to build such mask representing a known user (present in your database) in order to pass the facial recognition wall and crack the liveness system. These masks cost $3,000 or more. **Off course the liveness detector will reject any Print-Attack or Replay-Attack using these 3D masks.**