Doubango AI

3D 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

To our knowledge we're **the only company in the world** that can perform 3D liveness check and identity concealment detection from a single 2D image. We outperform the competition (FaceTEC, BioID, Onfido, Huawei...) in speed and accuracy. **Our implementation is Passive/Frictionless and only takes few milliseconds.**

Identity concealment detects when a user tries to partially hide his/her face (e.g. 3D realistic mask, dark glasses...) or alter the facial features (e.g. heavy makeup, fake nose, fake beard...) to impersonate another user.

A facial recognition system without liveness detector is just useless.

We can detect and block all known spoofing attacks: Paper Print, Screen, Video Replay, 3D (silicone, paper, tissue...) realistic face mask, 2D paper mask, Concealment...

The next video shows a stress test on our implementation using different type of attacks:

https://youtu.be/4irfRCiOx1w

Our passive (frictionless) face liveness detector uses **SOTA** (State Of The Art) deep learning techniques and can be freely tested 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.

2 Supported attacks

We can detect and block all known spoofing attacks: Paper Print, Screen, Video Replay, 3D (silicone, paper, tissue...) realistic face mask, 2D paper mask, identity concealment...

The next video shows a stress test on our implementation using different type of attacks:

https://youtu.be/4irfRCiOx1w

3 Configuration options

The configuration options are provided when the engine is initialized and **they are case-sensitive.**

Name	Type	values	Description
`assets_folder`	`STRING`	`Folder path`	Path to the folder containing the configuration files and deep learning models. Default value is the current folder. The SDK will look for the models in "`$(assets_folder)/models`" folder and configuration files in "`$(assets_folder)`". Default: **.**

`debug_level`	`STRING`	`verbose`
		`info`
		`warn`
		`error`
		`fatal`
			Defines the debug level to output on the console.
			You should use `verbose` for diagnostic, `info` in development stage and `warn` in production.
			Default: `info`

`debug_write_input_image_enabled`	`BOOLEAN`	`true` `false`	Whether to write the transformed input image to the disk. This could be useful for debugging. Default: `false`
`debug_internal_data_path`	`STRING`	`Folder path`	Path to the folder where to write the transformed input image. Used only if `debug_write_input_image_enabled` is `true`. Default: *“”*
`license_token_file`	`STRING`	`File path`	Path to the file containing the license token. First you need to generate a Runtime Key using `requestRuntimeLicenseKey()` function then activate the key to get a token. You should use `license_token_file` or `license_token_data` but not both.
`license_token_data`	`STRING`	`BASE64`	Base64 string representing the license token. First you need to generate a Runtime Key using `requestRuntimeLicenseKey()` function then activate the key to get a token. You should use `license_token_file` or `license_token_data` but not both.
`num_threads`	`INTEGER`	`Any`	Defines the maximum number of threads to use. You should not change this value unless you know what you're doing. Set to -1 to let the SDK choose the right value. The right value

			the SDK will choose will likely be equal to the number of virtual cores. For example, on an octa-core device the maximum number of threads will be #8, on quad-core it will be #4. Default: `-1`
`gpgpu_enabled`	`BOOLEAN`	`true` `false`	Whether to enable GPGPU computing. This will enable or disable GPGPU computing on the computer vision and deep learning libraries. On ARM devices this flag will be ignored when fixed-point (integer) math implementation exist for a well-defined function. For example, this function will be disabled for the bilinear scaling as we have a fixed-point SIMD accelerated implementation: https://github.com/DoubangoTelecom/compv/blob/master/base/image/asm/arm/compv_image_scale_bilinear_arm64_neon.S . Same for many deep learning parts as we're using QINT8 quantized inference. This option could also be ignored when the memory transfer time is high compared to the computation time. Default: `true`
`max_latency`	`INTEGER`	`[0, inf[`	The parallel processing method could introduce delay/latency in the delivery callback on low-end CPUs. This parameter controls the maximum latency you can tolerate. The unit is number of frames. The default value is -1 which means auto. Default: `-1`

`image_interpolation`	`STRING`	`nearest`
		`bilinear`
		`bicubic`
			Defines the interpolation method to use when pixels are scaled, deskewed or deslanted. `bicubic` offers the best quality but is slow as there is no SIMD or GPU acceleration yet. `bilinear` and `nearest` interpolations are multithreaded and SIMD accelerated. For most scenarios `bilinear` interpolation is good enough to provide high accuracy/precision results while the code still runs very fast.
			Default: `bicubic`

`asm_enabled`	`BOOLEAN`	`true` `false`	Whether to enable assembler code to use SIMD acceleration (SSE, AVX, NEON). Default: **`true`**
`intrin_enabled`	`BOOLEAN`	`true` `false`	Whether to enable intrinsic code to use SIMD acceleration (SSE, AVX, NEON). Default: **`true`**

`openvino_enabled`	`BOOLEAN`	`true` `false`	Whether to use OpenVINO instead of Tensorflow as deep learning backend engine. Default: **`true`**

`openvino_device`	`STRING`	`GNA`
		`HETERO`
		`CPU`
		`MULTI`
		`GPU`
		`MYRIAD`
		`HDDL`
		`FPGA`
			OpenVINO device to use for computations. We recommend using `CPU` which is always correct. If you have an Intel GPU, VPU or FPGA, then you can change this value. If you try to use any other value than CPU without having the right device, then OpenVINO will be completely disabled with a fallback on Tensorflow.
			Default: **`CPU`**

`detect_tf_num_threads`	`INTEGER`	`Any`	Defines the maximum number of CPU threads Tensorflow (TF) should use for the face detection pipeline. You should not change this value unless you know what you're doing. Set to −1 to let the SDK choose the right value. The right value the SDK will choose will likely be equal to the number of virtual cores. For example, on an octa-core device the maximum number of threads will be #8, on quad-core it will be #4. Decrease this value if you have OOM (Out Of Memory) errors or need to decrease the memory usage. Default: **`-1`**
`detect_tf_gpu_memory_alloc_max_percent`	`FLOAT`	`]0, 1]`	Defines the maximum GPU memory Tensorflow (TF) should use for the face detection pipeline. This value is expressed in percentage of the total available GPU memory size. Default: **`0.2`**
`detect_minscore`	`FLOAT`	`]0, 1]`	Face detection threshold. Any face detection score (percentage) with a score lower than this threshold will be discarded and and tagged as false-positive. Default: **`0.93`**
`detect_face_minsize`	`INTEGER`	`]0, inf]`	Face size threshold. Any face with a size (in pixels) lower than this threshold will be tagged as “small” and ignored. The value is expressed in pixels. You must not use a value lower than 64px. Default: **`65`**
`detect_roi`	`FLOAT[4]`	`Any`	Defines the Region Of Interest (ROI) for the detector. Any pixels outside the region of

			interest will be ignored by the detector. Defining an WxH region of interest instead of resizing the image at WxH is very important as you'll keep the same quality when you define a ROI while you'll lose in quality when using the later. Format: `[left, right, top, bottom]` Default: `[0.f, 0.f, 0.f, 0.f]`
`liveness_detect_enabled`	`BOOLEAN`	`true` `false`	Whether to enable liveness detection. Default: **`true`**
`liveness_tf_num_threads`	`INTEGER`	`Any`	Defines the maximum number of CPU threads Tensorflow (TF) should use for the liveness detection pipeline. You should not change this value unless you know what you're doing. Set to -1 to let the SDK choose the right value. The right value the SDK will choose will likely be equal to the number of virtual cores. For example, on an octa-core device the maximum number of threads will be #8, on quad-core it will be #4. Decrease this value if you have OOM (Out Of Memory) errors or need to decrease the memory usage. Default: **`-1`**
`liveness_tf_gpu_memory_alloc_max_percent`	`FLOAT`	`]0, 1]`	Defines the maximum GPU memory Tensorflow (TF) should use for the liveness detection pipeline. This value is expressed in percentage of the total available GPU memory size. Default: **`0.2`**
`liveness_face_minsize`	`INTEGER`	`]0, inf]`	Face size threshold. Any face with a size (in pixels) lower than this threshold will not pass through the liveness pipeline. The value is expressed as pixels. You must not use a value lower than 64px. Default: **`64`**
`liveness_genuine_minscore`	`FLOAT`	`]0, 1]`	Threshold for genuine faces. Any face with liveness score higher than or equal to this threshold will be tagged as genuine. Default: **`0.98`**
`liveness_disputed_minscore`	`FLOAT`	`]0, 1]`	Threshold for disputed faces. Any face with liveness score higher than or equal to this threshold but lower than `liveness_genuine_minscore` will be

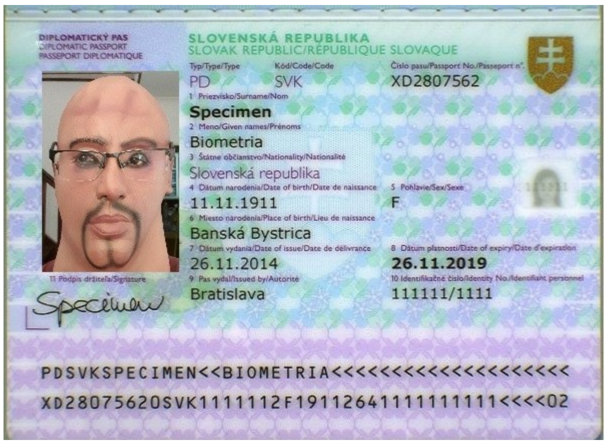
			tagged as disputed. In short, the score is within `[liveness_disputed_minscore, liveness_genuine_minscore[` interval. Default: **`0.50`**
`liveness_toofar_threshold`	`FLOAT`	`]0, 1]`	Threshold for toofar faces. This is the size of the face relative to the entire image in percentage. X percent means the face is x% of the entire image. Faces with size lower than this threshold will be tagged as toofar and logged as warning. Default: **`0.50`**
`deepfake_detect_enabled`	`BOOLEAN`	`true` `false`	Whether to enable deepfake detection. Default: **`true`**
`deepfake_tf_num_threads`	`INTEGER`	`Any`	Defines the maximum number of CPU threads Tensorflow (TF) should use for the deepfake detection pipeline. You should not change this value unless you know what you're doing. Set to `-1` to let the SDK choose the right value. The right value the SDK will choose will likely be equal to the number of virtual cores. For example, on an octa-core device the maximum number of threads will be #8, on quad-core it will be #4. Decrease this value if you have OOM (Out Of Memory) errors or need to decrease the memory usage. Default: **`-1`**
`deepfake_tf_gpu_memory_alloc_max_percent`	`FLOAT`	`]0, 1]`	Defines the maximum GPU memory Tensorflow (TF) should use for the deepfake detection pipeline. This value is expressed in percentage of the total available GPU memory size. Default: **`0.2`**
`deepfake_minscore`	`FLOAT`	`]0, 1]`	Threshold for deepfake faces. Any face with deepfake score higher than or equal to this threshold will be tagged as deepfake. Default: **`0.50`**
`disguise_detect_enabled`	`BOOLEAN`	`true` `false`	Whether to enable disguise detection. Default: **`true`**
`disguise_tf_num_threads`	`INTEGER`	`Any`	Defines the maximum number of CPU threads Tensorflow (TF) should use for the disguise

			detection pipeline. You should not change this value unless you know what you're doing. Set to `-1` to let the SDK choose the right value. The right value the SDK will choose will likely be equal to the number of virtual cores. For example, on an octa-core device the maximum number of threads will be #8, on quad-core it will be #4. Decrease this value if you have OOM (Out Of Memory) errors or need to decrease the memory usage. Default: **`-1`**
`disguise_tf_gpu_memory_alloc_max_percent`	`FLOAT`	`]0, 1]`	Defines the maximum GPU memory Tensorflow (TF) should use for the disguise detection pipeline. This value is expressed in percentage of the total available GPU memory size. Default: **`0.2`**
`disguise_minscore`	`FLOAT`	`]0, 1]`	Threshold for disguise faces. Any face with disguise score higher than or equal to this threshold will be tagged as disguise. Default: **`0.50`**

4 Identity concealment

Identity concealment feature is incredibly important and used to check if a user is trying to hide his/her identity. Let's say a user is about to open a bank account and is trying to enroll. The role of this feature is to make sure that the user exists in real life (not a photo or video or 3D mask) and most importantly, **his/her face is clear and not hidden**.

Let's say a user creates one of these 2 fake passports:

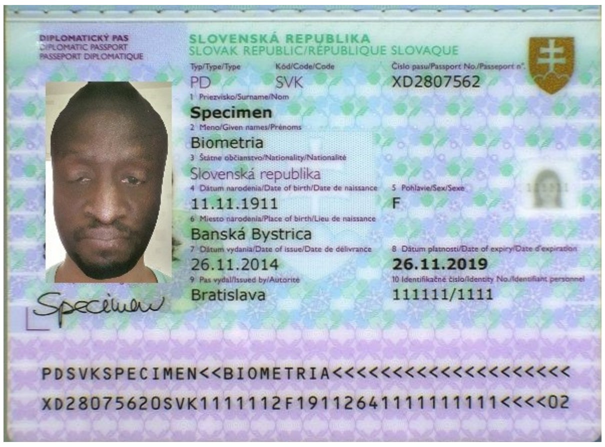


DIPLOMATICKÝ PAS
DIPLOMATIC PASSPORT
PASSEPORT DIPLOMATIQUE

SLOVENSKÁ REPUBLIKA
SLOVAK REPUBLIC/RÉPUBLIQUE SLOVAQUE

Typ/Type/Type: PD
Kód/Code/Code: SVK
Číslo pasu/Passport No./Passeport n°: XD2807562
1 Priezvisko/Surname/Nom: Specimen
2 Meno/Given names/Prénoms: Biometria
3 Štátne občianstvo/Nationality/Nationalité: Slovenská republika
4 Dátum narodenia/Date of birth/Date de naissance: 11.11.1911
5 Pohlavie/Sex/Sexe: F
6 Miesto narodenia/Place of birth/Lieu de naissance: Banská Bystrica
7 Dátum vydania/Date of issue/Date de délivrance: 26.11.2014
8 Dátum platnosti/Date of expiry/Date d'expiration: 26.11.2019
9 Pas vydal/Issued by/Autorité: Bratislava
10 Identifikačné číslo/Identity No./Identifiant personnel: 111111/1111
11 Podpis držiteľa/Signature: Specimen

```
PDSVKSPECIMEN<<BIOMETRIA<<<<<<<<<<<<<<<<<<<<<
XD28075620SVK1111112F1911264111111111<<<<02
```



DIPLOMATICKÝ PAS
DIPLOMATIC PASSPORT
PASSEPORT DIPLOMATIQUE

SLOVENSKÁ REPUBLIKA
SLOVAK REPUBLIC/RÉPUBLIQUE SLOVAQUE

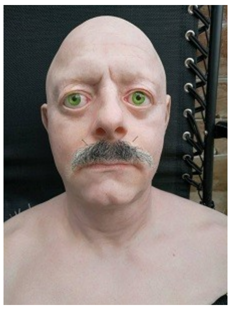
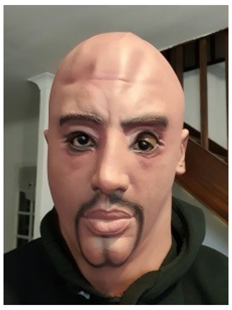
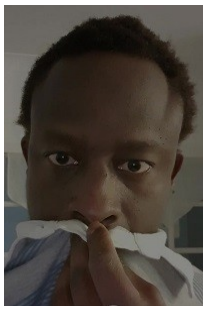
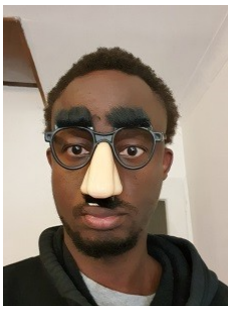
Typ/Type/Type: PD
Kód/Code/Code: SVK
Číslo pasu/Passport No./Passeport n°: XD2807562
1 Priezvisko/Surname/Nom: Specimen
2 Meno/Given names/Prénoms: Biometria
3 Štátne občianstvo/Nationality/Nationalité: Slovenská republika
4 Dátum narodenia/Date of birth/Date de naissance: 11.11.1911
5 Pohlavie/Sex/Sexe: F
6 Miesto narodenia/Place of birth/Lieu de naissance: Banská Bystrica
7 Dátum vydania/Date of issue/Date de délivrance: 26.11.2014
8 Dátum platnosti/Date of expiry/Date d'expiration: 26.11.2019
9 Pas vydal/Issued by/Autorité: Bratislava
10 Identifikačné číslo/Identity No./Identifiant personnel: 111111/1111
11 Podpis držiteľa/Signature: Specimen

```
PDSVKSPECIMEN<<BIOMETRIA<<<<<<<<<<<<<<<<<<<<<
XD28075620SVK1111112F1911264111111111<<<<02
```

On the left he is trying to hide his face behind a 3D mask while on the right he's trying to change his facial features (**identity concealment**). Both attempts will fail on liveness check.

4.1 Type of concealment

We support an infinite number of concealment types. To say it like it is: **The AI understands when something is "fishy"**. Next table shows some "fishy" faces:

5 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/.

5.1 The competition

Our demo web app is provided in unconstrained environment. You can drag&drop any image, use any resolution (4K or more), use your front or back camera, put the camera at any distance...

Other companies would not provide their product in such unconstrained environment for multiple reasons:

1. Some implementations are texture based and use LBP variants or deep learning. This requires a very close face. Some companies clearly state on their website that "it will result in poor/undefined behavior" if you don't follow this rule. This is why they're requiring you to put your face in a box. We also recommend a close face but not doing so will result in predictable behavior.
2. They will not let you choose the resolution. It's very hard to distinguish a spoof and a genuine face when the resolution and image quality are very high. This is why some companies will not let you choose resolutions higher than HD.
3. Some will not let you choose the back camera. The back camera has higher resolution and it's easier to take a picture of a spoof. The front camera has lower resolution and it's very uncomfortable to take a picture of someone else using it without having distortions. Try using the front camera and putting the face of someone else in a tiny box without distortion and you'll feel the pain. Also, using the front camera at close range will have the screen light reflecting on the paper or screen, this make it easier to detect spoofs. Dim-out the screen light and you'll be able to crack some implementations.

6 Testing the competition

This page is written and maintained by Mamadou DIOP. Please contact him if you have any question or clarification about your application.

I'm using Samsung Galaxy S10+ and have installed the latest version (June 26, 2021) for each application.

I have just concatenated the videos with no editing. You can contact me if you need the raw videos (rushes).

Please keep in mind that unlike the competition our implementation is available for testing in unconstrained environment which means you can use images at any resolution and the face can be at any distance from the camera lens.

6.1 FaceTEC

Web site: https://www.facetec.com

I have to give credit to these guys: **They are very good in marketing.**

They have disabled screen shot/screen recording on the liveness screen which makes it impossible to record a video like what I did for other applications.

<1% FRR: They claim on their website that the FRR (False Rejection Rate) is less than 1%. That's clearly not true. Using a Galaxy S10+ (high-end camera) or Wiko View4 (low-end camera) it was a nightmare to enroll a genuine face (myself). The app keeps asking me to **"Move away"**, **"Move closer"**, **"Hold Steady"**... and at the end I get a message asking me to try again. Sometimes I have to try 10 to 15 times in order to enroll a genuine face. The comments on Google Play (back to 2017) show multiple users have the same issue. Tried in different rooms with very good lighting but had the same issues. The experience is different on iPhone12 Pro: It's fast and accurate.

1/12,800,000 FAR: This claim may be true but comes with a cost (low FRR). Again, it's easy to achieve a good FAR if you accept to sacrifice your FRR to the marketing gods. My next code has a perfect FAR (0/1,000,000,000,000):

```
def liveness(image):
    return False # Always say it's spoof
```

Liveness detection in 2 seconds: The network transfer alone to send the video selfie takes way more than 2 seconds (used with very good 4G network). In my experience (Galaxy S10+) it takes at least 15 seconds, sometimes it's several minutes. On iPhone 12 Pro it's way faster.

$100,000 Spoof Bounty Program: That's clearly a marketing message. I have managed to bypass the liveness check using several 3D masks. As already explained, screen recording is disabled on their app which made it impossible to capture.

800% improvements: I've subscribed to their mailing list and few weeks ago they sent a mail saying their accuracy has improved by 800%. You can also find the same information here. No comment.

6.2 BioID

Web site: https://www.bioid.com/

Despite being ISO/IEC 30107 (level 1 and 2) certified the implementation can be easily fooled. I have managed to bypass (enroll and verify) the liveness detector using Replay-Attacks (video) and 3D face masks. I have also managed to enroll and verify myself while hiding almost 80% of my face with dark glasses and a COVID mask (check the video).

The next video shows the results:

https://youtu.be/j24lDxll2qc

6.3 Huawei

Web site: https://developer.huawei.com/consumer/en/doc/development/HMSCore-Guides-V5/liveness-detection-0000001051386243-V5

I have developed and released an Open Source project on Github in order to test this implementation: https://github.com/DoubangoTelecom/HuaweiFaceLiveness

Months ago I tested this application against Print-Attacks, you can retrieve the video at https://youtu.be/52W2K5yJIl4. This time I wanted to test 3D attacks. The application fails to detect these attacks but I think it's normal as they're not branding it as a 3D liveness detector.

Check the next video (3D attacks):

https://youtu.be/7P0Jyw6rmxE

6.4 Wiko

I have used a View 4 phone to register my face. **Surprisingly, it was very easy to unlock the phone using a high definition picture of myself.**

https://youtu.be/wbHSrYUqmkk

On the video I'm wearing dark glasses and a face mask to make sure I'm not unlocking the phone by inadvertence.

6.5 Samsung

Like Wiko I registered my face (Galaxy S10+) and tried to unlock the phone using a picture of myself but … I failed. I can't say if their implementation is really strong or not because they disable the face recognition when you fail 3 (or 5) times. Disabling face recognition for a period of time and forcing you to enter your password make it painful to try multiple times to find the right angle and lighting.

6.6 Sybrin

Web site: https://corporate.sybrin.com/

The application keeps crashing after OCR'ing the passport. So, haven't managed to test their liveness detector.

https://youtu.be/V2ZlNGzBeaU

6.7 Onfido

Web site: https://onfido.com

They have a very light and basic active liveness detector. I don't know if the one in their application on Google Play is the same as what they provide to their customers but it looks weak and easy to break. This implementation fails on 3D and replay attacks.

https://youtu.be/Iqt9uxCD8ok

6.8 Jumio

Web site: https://www.jumio.com

You have to successfully provide an identity document in order to test the liveness detector. I haven't managed to pass the OCR. Tried with a real French passport and identity card by they fail to recognize the MRZ lines. Next video shows a try:

https://youtu.be/k-_8BJRO3R0

By the way, we (Doubango AI) have an MRZ reader and you can see it at work here:

https://youtu.be/AO5XdbLK9Do

6.9 Minivision-ai (turingtech.vip)

Web site: https://github.com/minivision-ai/Silent-Face-Anti-Spoofing

This is an open source implementation with a friendly commercial license (Apache). They do not claim to be a 3D liveness detector but the deep learning model suggests it is (Sofmax dense 3).

I managed to build the project and did some tests but it was clearly not good. On their home page they claim that they have a better version but it's private. While browsing their Github issue tracker I found a comment about a better implementation. This implementation is from turingtech (https://turingtech.vip) and can be found at https://github.com/turingtech-vip/Face-Anti-Spoofing-Android-iOS. I unziped the APK and scanned the binaries and surprise surprise... it's based on Minivision-ai. The implementation from turingtech is clearly more accurate which suggest they re-trained the model with more data.

This implementation is very sensible to lighting and easily fail to all attacks: Screen display, Paper photo, video replay, 3D masks...

https://youtu.be/vTrKZ5v_MOQ

7 ISO ISO/IEC 30107

All ISO certified applications we have tested can be easily fooled. I guess this certification is for marketing purposes.

8 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.

9 Passive versus Active liveness detection

9.1 *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...

9.2 *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.

9.3 *Why passive is better?*

This section explain why passive detectors are better.

9.3.1 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.

9.3.2 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**.

9.3.3 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.

9.3.4 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+).

9.3.5 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.

9.3.6 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.

10 Recommendations

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

11 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.

11.1 The 7 commandments

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

11.1.1 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

11.1.2 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).

11.1.3 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.

11.1.4 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.

11.1.5 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.

11.1.6 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.

11.1.7 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.

12 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.

13 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[%**.

`s_disguise:` The face was analyzed and is most likely a disguise. It could be a 3D mask or you have your face partially hidden.

`s_pending:` The operation is pending. This code is returned when parallel processing is enabled. The final response will be sent asynchronously using the registered callback function.

`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.

`e_nolicense:` The operation you're trying to perform is not permited.

14 Known issues

Some known issues to be fixed in the next versions.

14.1 Masks

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

14.2 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.

14.3 3D realistic masks

Fixed in v0.2-beta with support for identity concealment**.**