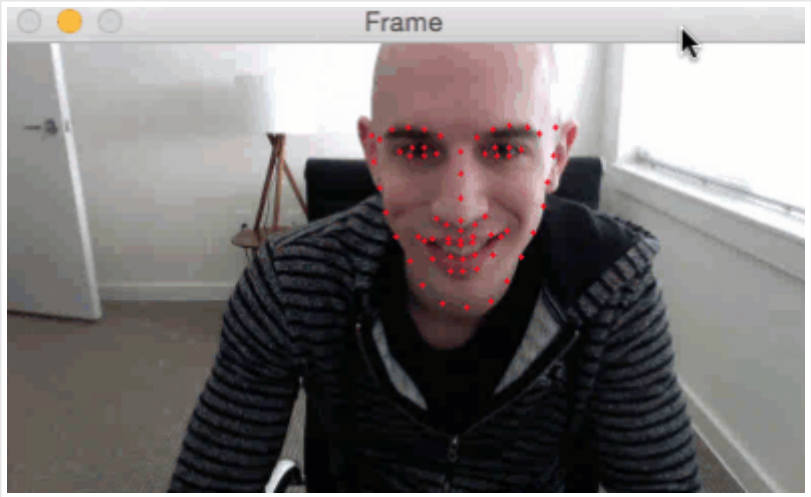
pyimagesearch
be awesome at building image search engines

Real-time facial landmark detection with OpenCV, Python, and dlib

by **Adrian Rosebrock** on April 17, 2017 in **dlib**, **Facial Landmarks**, **Tutorials**



Over the past few weeks we have been discussing *facial landmarks* and the role they play in computer vision and image processing.

We've started off by learning how to detect facial landmarks in an image.

We then discovered how to label and annotate *each of the facial regions*, such as eyes, eyebrows, nose, mouth, and jawline.

Today we are going to expand our implementation of facial landmarks to work in *real-time video streams*, paving the way for more real-world applications, including next week's tutorial on *blink detection.*

To learn how to detect facial landmarks in video streams in real-time, just keep reading.

Looking for the source code to this post?
Jump right to the downloads section.

Real-time facial landmark detection with OpenCV, Python, and dlib

The first part of this blog post will provide an implementation of real-time facial landmark detection for usage in video streams utilizing Python, OpenCV, and dlib.

We'll then test our implementation and use it to detect facial landmarks in videos.

Facial landmarks in video streams

Let's go ahead and get this facial landmark example started.

Open up a new file, name it `video_facial_landmarks`

```
Real-time facial landmark detection with OpenCV,
1 # import the necessary packages
2 from imutils.video import VideoStream
3 from imutils import face_utils
4 import datetime
5 import argparse
6 import imutils
7 import time
8 import dlib
9 import cv2
```

Lines 2-9 import our required Python packages.

We'll be using the `face_utils` sub-module of imutils version, take a second and do so now:

```
Real-time facial landmark detection with OpenCV,
1 $ pip install --upgrade imutils
```

***Note:** If you are using Python virtual environments, take care to ensure you are installing/upgrading `imutils` in your proper environment.*

We'll also be using the `VideoStream` implementation inside of `imutils` , allowing you to access your webcam/USB camera/Raspberry Pi camera module in a *more efficient*, *faster, treaded manner.* You can read more about the `VideoStream` class and how it accomplishes a higher frame throughout in this blog post.

If you would like to instead work with *video files* rather than *video streams*, be sure to reference this blog post on efficient frame polling from a pre-recorded video file, replacing `VideoStream` with `FileVideoStream` .

For our facial landmark implementation we'll be using the dlib library. You can learn how to install dlib on your system in this tutorial (if you haven't done so already).

Next, let's parse our command line arguments:

```
Real-time facial landmark detection with OpenCV,
11 # construct the argument parse and parse the
```

```
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--shape-predictor", required=True,
    help="path to facial landmark predictor")
ap.add_argument("-r", "--picamera", type=int, default=-1,
    help="whether or not the Raspberry Pi camera should be used")
args = vars(ap.parse_args())
```

Our script requires one command line argument, followed by a second optional one, each detailed below:

- `--shape-predictor` : The path to dlib's pre-trained facial landmark detector. Use the ***"Downloads"*** section of this blog post to download an archive of the code + facial landmark predictor file.
- `--picamera` : An optional command line argument, this switch indicates whether the Raspberry Pi camera module should be used instead of the default webcam/USB camera. Supply a value *> 0* to use your Raspberry Pi camera.

Now that our command line arguments have been pa
SVM-based face detector and then load the facial lan

```
Real-time facial landmark detection with OpenCV,
# initialize dlib's face detector (HOG-based)
# the facial landmark predictor
print("[INFO] loading facial landmark predict
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(args["shape_
```

The next code block simply handles initializing our `V`
warm up:

```
Real-time facial landmark detection with OpenCV,
# initialize the video stream and allow the
print("[INFO] camera sensor warming up...")
vs = VideoStream(usePiCamera=args["picamera"]
time.sleep(2.0)
```

The heart of our video processing pipeline can be fou

```
Real-time facial landmark detection with OpenCV, Python, and dlib                Python
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream, resize it to
    # have a maximum width of 400 pixels, and convert it to
    # grayscale
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # detect faces in the grayscale frame
    rects = detector(gray, 0)
```

On **Line 31** we start an infinite loop that we can only break out of if we decide to exit the script by pressing the `q` key on our keyboard.

Line 35 grabs the next frame from our video stream.

We then preprocess this frame by resizing it to have a width of 400 pixels and convert it to grayscale (**Lines 36 an 37**).

Before we can detect facial landmarks in our frame, we first need to localize the face — this is accomplished on **Line 40** via the `detector` which returns the bounding box *(x, y)*-coordinates for each face in the image.

Now that we have detected the faces in the video stream, the next step is to apply the facial landmark predictor to each face ROI:

Real-time facial landmark detection with OpenCV, Python, and dlib — Python

```
42     # loop over the face detections
43     for rect in rects:
44         # determine the facial landmarks for the face region, then
45         # convert the facial landmark (x, y)-coordinates to a NumPy
46         # array
47         shape = predictor(gray, rect)
48         shape = face_utils.shape_to_np(shape)
49
50         # loop over the (x, y)-coordinates fo
51         # and draw them on the image
52         for (x, y) in shape:
53             cv2.circle(frame, (x, y), 1, (0,
54
55     # show the frame
56     cv2.imshow("Frame", frame)
57     key = cv2.waitKey(1) & 0xFF
58
59     # if the `q` key was pressed, break from
60     if key == ord("q"):
61         break
```

On **Line 43** we loop over each of the detected faces.

Line 47 applies the facial landmark detector to the fa convert to a NumPy array (**Line 48**).

Lines 52 and 53 then draw a series of circles on the landmarks. To understand what facial region (i.e., nos to, please refer to this blog post.

Lines 56 and 57 display the output `frame` to our screen. If the `q` key is pressed, we break from the loop and stop the script (**Lines 60 and 61**).

Finally, **Lines 64 and 65** do a bit of cleanup:

Real-time facial landmark detection with OpenCV, Python, and dlib — Python

```
63 # do a bit of cleanup
64 cv2.destroyAllWindows()
65 vs.stop()
```

As you can see, there are very little differences between detecting facial landmarks in *images* versus detecting facial landmarks in *video streams* — the main differences in the code simply involve setting up our video stream pointers and then polling the stream for frames.

The actual process of detecting facial landmarks is the *same,* only instead of detecting facial landmarks in a *single image* we are now detecting facial landmarks in a *series of frames*.

Real-time facial landmark results

To test our real-time facial landmark detector using OpenCV, Python, and dlib, make sure you use the ***"Downloads"*** section of this blog post to download an archive of the code, project structure, and facial landmark predictor model.

If you are using a standard webcam/USB camera, you can execute the following command to start the video facial landmark predictor:

```
Real-time facial landmark detection with OpenCV, Python, and dlib                Shell
1 $ python video_facial_landmarks.py \
2     --shape-predictor shape_predictor_68_face_landmarks.dat
```

Otherwise, if you are on your Raspberry Pi, make sure you append the `--picamera 1` switch to the command:

```
Real-time facial landmark detection with OpenCV,
1 $ python video_facial_landmarks.py \
2     --shape-predictor shape_predictor_68_face_
3     --picamera 1
```

Here is a short GIF of the output where you can see t
detected on my face in real-time:

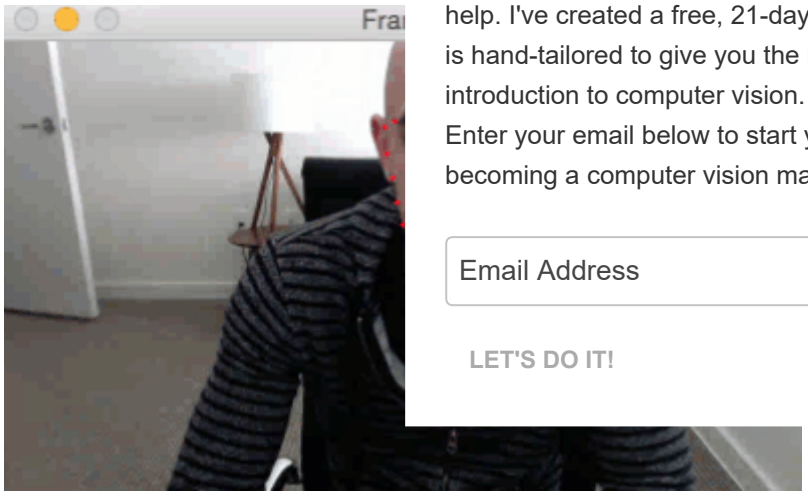


Figure 1: A short demo of real-time facial landmark detection with OpenCV, Python, an dlib.

I have included a full video output below as well:

Real-time facial landmark detection with OpenCV, Python,...

Summary

In today's blog post we extended our previous tutoria k of real-time detection.

As our results demonstrated, we are fully capable of l- time using a system with a modest CPU.

Now that we understand how to access a video strea move on to next week's real-world computer vision a

To be notified when the blink detection tutorial gc the form below — *this is a tutorial you won't wan*

Downloads:

If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 11-page Resource Guide** on Computer Vision and Image Search Engines, including **exclusive techniques** that I don't post on this blog! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).

Free 21-day crash course on computer vision & image search engines

Enter your email address below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own!

Your email address

DOWNLOAD THE GUIDE!

dlib, **face detection**, **face parts**, **face regions**, **facial**

‹ Detect eyes, nose, lips, and jaw with dlib, OpenCV , and Pyth

b ›

Free 21-day crash course on computer vision & image search engines

Interested in computer vision and image search engines, but don't know where to start? Let me help. I've created a free, 21-day crash course that is hand-tailored to give you the best possible introduction to computer vision. Sound good? Enter your email below to start your journey to becoming a computer vision master.

Email Address

LET'S DO IT!

43 Responses to *Real-time facial landm and dlib*

tony April 17, 2017 at 12:03 pm #

Thanks for this tutorial . how the face landma points are shaky

Mansoor Nasir April 17, 2017 at 1:00 pm #

REPLY

Great work Adrian, my only question is will it work with multiple faces? And will it affect the performance or accuracy?

Adrian Rosebrock April 19, 2017 at 12:58 pm #

REPLY

I would suggest going back and reading my previous posts facial landmarks. This method will work with multiple faces provided that each face in the image/video stream can be detected.

Shravan Kumar Parunandula April 17, 2017 at 1:05 pm #

REPLY

Awaiting for this, thank you so much.

Adrian Rosebrock April 19, 2017 at 12:57 pm #

REPLY

Thank you Shravan! 🙂

Linus April 17, 2017 at 1:40 pm #

REPLY

This one is freaking awesome! Will definitively try it out and install dlib. Thanks Adrian for this row of posts! 🙂

Adrian Rosebrock April 19, 2017 at 12:55 p

Thanks Linus — it only gets better from h

Muhammad April 17, 2017 at 2:34 pm #

Beautiful! Thanks a lot!

Levi Blaney April 17, 2017 at 9:02 pm #

Hey this is really great stuff. I can't wait to try
is standing in front of it. I'm sure I could Google a algo
some what reliably detect the same person over and

Adrian Rosebrock April 19, 2017 at 12:53 pm #

REPLY

I wouldn't recommend using facial landmarks for facial recognition. Algorithms such as LBPs for face recognition, Eigenfaces, and Fisherfaces would work well for a magic mirror application. I cover LBPs for face recognition and Eigenfaces inside the PyImageSearch Gurus course.

tbanda April 27, 2017 at 10:33 am #

REPLY

Hello Adrian…
what if i use dlib only for face identification because haarcascades do not identify flipped faces and from there i use LBP for face recognition?

Adrian Rosebrock April 28, 2017

As I mentioned, facial landmarks are not used for face identification. You can use either Haar cascades or the dlib built-in factor detector to detect horizontally flipped faces. To detect vertically flipped faces, simply flip your image prior to passing them into the detector.

Joe April 17, 2017 at 9:53 pm #

REPLY

This is awesome! Thanks for another great blog Adrian….keep it up!

Adrian Rosebrock April 19, 2017 at 12:51 p

Thanks Joe! 🙂

kunal April 18, 2017 at 3:59 am #

Really impressed by the way you have done detection for usage in video streams utilizing Python.

Linus April 18, 2017 at 1:59 pm #

And I can't understand why you import dateti

And the whole thing worked out just fine BTW 😀

Adrian Rosebrock April 19, 2017 at 12:48 pm #

REPLY

The `import datetime` cam be safely removed. I had it imported for a different application I was working on.

David J Axelrod April 19, 2017 at 11:07 am #

REPLY

Woah, super cool Adrian! Another awesome article

Adrian Rosebrock April 19, 2017 at 12:42 pm #

REPLY

Thank you David!

Matt Sandy April 19, 2017 at 11:30 pm #

REPLY

I really want to make a game controlled by facial expressions. I think it would be hilarious to get people to play it in public.

Adrian Rosebrock April 21, 2017 at 10:58 am #

REPLY

That certainly sounds like a neat game, Matt! I'm actually covering how to recognize facial expressions and emotions inside Deep Learning for Computer Vision with Python. Be sure to take a look!

Sidharth Patnaik April 20, 2017 at 9:49 pm #

Just another awesome Tutorial, thanks for sh
was waiting for this, the whole time.
can you upload a tutorial based on OpenFace, please

Adrian Rosebrock April 21, 2017 at 10:51 a

Sure, I will certainly consider this for a fu

tony April 21, 2017 at 4:18 am #

Thanks for this tutorial , I have asked you this question and I haven't got reply.

how the face landmarks can be more stable , I tried the tutorial and the points are shaky

Adrian Rosebrock April 21, 2017 at 10:45 am #

REPLY

Hi Tony — I'm not sure what you mean by "shaky". The facial landmark predictor included by dlib is pre-trained. You could try to train your own predictor on your own data to see if that improves your result.

tony April 22, 2017 at 12:52 am #

REPLY

Thanks , I mean the landmark points are not stable (shaking) . How can I train new predictor for more than 68 landmarks ?

Adrian Rosebrock April 24, 2017 at 9:48 am #

REPLY

You would need to use the dlib library. This example demonstrates how to train a custom shape predictor.

carlos julio pardo April 28, 2017 at 5:44 pm #

REPLY

Hi …how can i set up Dlib on visualstudio 2012 or other version?

Adrian Rosebrock May 1, 2017 at 1:48 pm

Hi Carlos — I only cover how to install dl
Setting up dlib on Windows or other Microsoft pro
looking at the official dlib website.

Thimira Amaratunga May 5, 2017 at 3:45 am #

Hi Adrian,

Thanks for another awesome tutorial.
I noticed that you convert the image frame to graysca
assuming it was for speeding up the face detection pr

If we pass a color image to the Dlib face detector, wou
detection accuracy will be high because more feature data will be available), or have the Dlib face detector designed to work better with grayscale images?

Thanks,

Adrian Rosebrock May 8, 2017 at 12:38 pm #

REPLY

It really depends on the underlying HOG + Linear SVM implementation. Dalal and Triggs (the authors of the original HOG paper) found that computing the gradient over multiple channels and taking the maximum response can increase detection accuracy. The problem is that you end up computing the gradient representation for each channel which makes the detection pipeline slower. In this case I like to explicitly convert to grayscale for speed.

Ivan May 7, 2017 at 3:50 pm #

REPLY

Great stuff! How hard would it be to extract pose data (head pitch, yaw, & roll) from the features here? Thanks!

Jyotsna May 18, 2017 at 1:31 am #

REPLY

I m getting error mentioned below when executing above program of real-time facial landmark detection:
File “video_facial_landmarks.py”, line 22, in

predictor = dlib.shape_predictor(args[“shape_predictor”])

RuntimeError: Unable to open shape_predictor_68_fa

Please help.
Thanks in advance

Adrian Rosebrock May 18, 2017 at 11:48 a

Make sure you use the “Downloads” sec facial landmark predictor file. The issue is that yo

Alex May 22, 2017 at 2:29 am #

Hi Adrian,

Thanks a lot for your amazing work !

I wondered if you have any idea regarding what would be the best way to go in order to create a legs detection algorithm ?

Thanks for your help,
Alex

Adrian Rosebrock May 25, 2017 at 4:41 am #

REPLY

Hi Alex — I would suggest training your own custom HOG + Linear SVM object detector. I demonstrate how to code and train the detector inside the PyImageSearch Gurus course.

Vinod Ramamoorthy May 25, 2017 at 9:49 am #

REPLY

Hi Adrian – thanks for your great work – kudos 😶

I’m stuck and I thought I’ll ask for your help.

Installed opencv and dlib successfully.
site-packages are symliked to my virtualenv as well.

when I run (i'm working within my virtualenv at this point and the files are located within /Documents/cv)

python video_facial_landmarks.py \
–shape-predictor shape_predictor_68_face_landmarks.dat \
–picamera 1

throws the following error

File "/home/pi/.virtualenvs/facecam/lib/python3.4/site-packages/imutils/video/pivideostream.py", line 2, in
from picamera.array import PiRGBArray
ImportError: No module named 'picamera'

Vinod Ramamoorthy May 25, 2017 at 9:57

Update – figured out the problem here –

forgot this~
pip install "picamera[array]"

but have another problem at hand~

(Frame:1158): Gtk-warning **:cannot open display

Vinod Ramamoorthy May 25, 2017 a

update – was running the script from

Once I rebooted the system with pixel GUI en

Adrian Rosebrock May 28, 2017 at 1:23 am #

REPLY

Congrats on resolving the issue Vinod 🙂

Wenliangh Wang May 30, 2017 at 2:38 pm #

REPLY

Hi Adrian,

I referred your above article, it works perfectly fine on my face image as well.

I wanted to use this facial landmark detected image and match with it my other image which I have placed in train folder (Face Recognition)

How can I effectively match these landmark features I
recognition code (Local Binary Patterns with Python &

http://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

Do you have any opencv code which can effectively recognize images based on facial landmarks?

I would be very grateful if you could help me in this regard and if I could successfully complete this assignment, I will surely enroll myself for pyimagesearch gurus.

Regards
Wenliangh Wang

Adrian Rosebrock May 31, 2017 at 1:10 pm #

REPLY

Facial landmarks are not used for face re
as Eigenfaces and LBPs for face recognition. The
the PyImageSearch Gurus course.

Trackbacks/Pingbacks

Eye blink detection with OpenCV, Python, and dlib - PyIm
[…] last week's blog post, I demonstrated how to perform

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Resource Guide (it's totally free).

Click the button below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own.

Free 21-day crash course on computer vision & image search engines

Download for Free!

Deep Learning for Computer Vision with Python Book

You're interested in deep learning and computer vision, *but y*
book will teach you all you need to know about deep lear

CLICK HERE TO PRE-ORDER MY NEW BOOK

Free 21-day crash course on computer vision & image search engines

Interested in computer vision and image search engines, but don't know where to start? Let me help. I've created a free, 21-day crash course that is hand-tailored to give you the best possible introduction to computer vision. Sound good? Enter your email below to start your journey to becoming a computer vision master.

Email Address

LET'S DO IT!

You can detect faces in images & video.

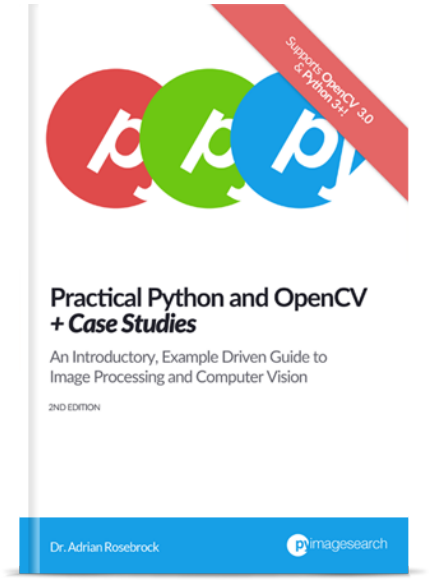
Are you interested in **detecting faces in images & video?** B
me help! I guarantee that my new book will turn you into a **fa**
give it a shot yourself.

Free 21-day crash course on computer vision & image search engines

Hello! I'm Adrian Rosebrock.

I'm an entrepreneur and Ph.D who has launched two successful image search engines, ID My Pill and Chic Engine. I'm here to share my tips, tricks, and hacks I've learned along the way.

Learn computer vision in a single weekend.



Want to learn computer vision & OpenCV? I can teach you in
My new book is your **guaranteed, quick-start guide** to beco
to become a computer vision ninja.

CLICK HERE TO BECOME AN OPENCV NINJA

Subscribe via RSS

Never miss a post! Subscribe to the PyImageSe
engine tutorials, tips, and tricks

POPULAR

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

How to install OpenCV 3 on Raspbian Jessie
OCTOBER 26, 2015

Basic motion detection and tracking with Python and OpenCV
MAY 25, 2015

Accessing the Raspberry Pi Camera with OpenCV and Python
MARCH 30, 2015

Install OpenCV 3.0 and Python 2.7+ on Ubuntu
JUNE 22, 2015

Search

Free 21-day crash course on computer vision & image search engines

Interested in computer vision and image search engines, but don't know where to start? Let me help. I've created a free, 21-day crash course that is hand-tailored to give you the best possible introduction to computer vision. Sound good? Enter your email below to start your journey to becoming a computer vision master.

Email Address

LET'S DO IT!

Free 21-day crash course on computer vision & image search engines

Search...

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.

Free 21-day crash course on computer vision & image search engines