

Instituto Federal Goiano - Campus Ceres
Bacharelado em Sistemas de Informação
Prof. Me. Ronneesley Moura Teles

Adallberto Lucena Moura
Andrey Silva Ribeiro
Anny Karoliny Moraes Ribeiro
Brenner Gomes de Jesus
Davi Ildeu de Faria
Eduardo de Oliveira Silva
Gleyson Israel Alves
Gusttavo Nunes Gomes
Ianka Talita Bastos de Assis
Ígor Justino Rodrigues

Algoritmo Viola-Jones

Sumário

1	O artigo	2
2	Funcionamento do algoritmo	2
3	Vantagem	2
4	Desvantagem	2
5	Implementação Viola-Jones em Python	3
5.1	Porquê a implementação em Python?	3
5.2	IntegralImage.py	3
5.3	HaarLikeFeature.py	4
5.4	AdaBoost.py	6
5.5	Utils.py	9
5.6	example.py	11
6	Referências Bibliográfica	13

Algoritmo Viola-Jones

1 O artigo

Em 2001, *Paul Viola* e *Michael Jones*, dois pesquisadores de Cambridge publicaram um artigo intitulado: “*Rapid Object Detection using a Boosted Cascade of Simple Features*” que demonstrava um novo método de detecção de faces. O artigo se diferencia e se paltava em 3 pontos importantes.

A primeira foi uma nova maneira de se representar uma imagem, a “imagem integral” (*Integral Image*, em inglês), que permitiu os detectores usados por eles, computarem a imagem de maneira mais rápida.

A segunda foi o algoritmo de aprendizado baseado no *AdaBoost*, que selecionava um número pequeno de características visuais críticas de um conjunto maior e com seus classificadores, extremamente eficientes.

O terceiro aspecto importante, foi o método de combinar e incrementar classificadores em “cascata”, o que permitia regiões do fundo da foto de serem rapidamente descartadas, disponibilizando maior processamento computacional em posições com maior possibilidade de ser o objeto no qual se está procurando, como um rosto.

Os autores aprofundaram seus métodos de como construíram esse algoritmo, de como funcionavam as equações e apresentaram os resultados encontrados e compararam com algoritmos similares da época, e como os algoritmos “*Rowley-Baluja-Kanade*”, “*Schneiderman-Kanade*” e “*Roth-Yang-Ahuja*”.

Devido sua implementação seguir uma abordagem diferente para construção de um sistema de detecção de face, aproximadamente 15 vezes mais rápida que os métodos anteriores, o algoritmo *Viola-Jones* revolucionou esse campo da computação se tornando uma referência.

2 Funcionamento do algoritmo

3 Vantagem

- 15 vezes mais rápido que o algoritmo “*Rowley-Baluja-Kanade*” no processamento da imagem.
- 600 vezes se comparado ao “*Schneiderman-Kanade*”.

4 Desvantagem

- A detecção de faces, só é possível se o rosto estiver na posição frontal.
- A base de dados usada, precisa de faces em diferentes condições incluindo: iluminação, brilho, escala, pose e variações de câmera.
- Nível de detecção na literatura - 80% (FAUX,2012)
- É um algoritmo de detecção de face e não de reconhecimento facial.

5 Implementação Viola-Jones em Python

5.1 Porquê a implementação em Python?

Existem inúmeras implementações do algoritmo na internet, nas mais diversas linguagens, sendo mais comuns em *MatLab*, *Java* e *Python*.

A implementação em MatLab não era muito interessante devido a ferramenta em si, é uma boa ferramenta para desenvolver procedimentos tais como esse de detecção facial. entretanto é uma ferramenta paga, e em um projeto nessa escala, no qual muitas pessoas iriam interagir, não ter a licença de uso, dificultaria muito o processo, e por isso foi descartada.

Java era muito abundante no *GitHub*, entretanto a maioria estava incompleta, não possuindo os códigos para treinar a rede, o que é fundamental para o sucesso do algoritmo, ter uma base de dados ampla gerando uma maior eficiência na detecção de imagens. Os códigos que estavam completos, a documentação era quase inexistente, o que dificultaria a utilização, sendo talvez mais vantajoso desenvolver desde o início uma aplicação do que gastar tempo em código alheio.

Entretanto escolhemos a implementação de *Simon Hohberg* do algoritmo em *Python* disponibilizado no seu *GitHub* pessoal: <https://github.com/Simon-Hohberg/Viola-Jones>

Python é uma linguagem mais simples, mas nem por isso menos robusta, é amplamente usada em processos de *machine learning* e a implementação encontrada, está com todas as partes do algoritmo, desde a parte de detecção até a de treinamento, e a documentação do código está bem feita, o que possibilitaria o uso.

A seguir segue os principais códigos do algoritmo implementado em *Python* por *Simon Hohberg*: <https://github.com/Simon-Hohberg/Viola-Jones>

5.2 IntegralImage.py

```
1 import numpy as np
2
3
4 """
5 In an integral image each pixel is the sum of all pixels in the
6 original image
7 that are 'left and above' the pixel.
8
9 Original      Integral
10 +-----+    +-----+
11 | 1 2 3 . |    | 0 0 0 0 . |
12 | 4 5 6 . |    | 0 1 3 6 . |
13 | . . . . |    | 0 5 12 21 . |
14 |         |    | . . . . . |
15
16 """
17
18 def to_integral_image(img_arr):
19     """
20     Calculates the integral image based on this instance's original
21     image data.
22     :param img_arr: Image source data
23     :type img_arr: numpy.ndarray
24     :return Integral image for given image
25     :rtype: numpy.ndarray
```

```

25     """
26     # an index of -1 refers to the last row/column
27     # since row_sum is calculated starting from (0,0),
28     # rowSum(x, -1) == 0 holds for all x
29     row_sum = np.zeros(img_arr.shape)
30     # we need an additional column and row
31     integral_image_arr = np.zeros((img_arr.shape[0] + 1, img_arr.shape
32     [1] + 1))
33     for x in range(img_arr.shape[1]):
34         for y in range(img_arr.shape[0]):
35             row_sum[y, x] = row_sum[y-1, x] + img_arr[y, x]
36             integral_image_arr[y+1, x+1] = integral_image_arr[y+1, x
37             -1+1] + row_sum[y, x]
38     return integral_image_arr
39
40 def sum_region(integral_img_arr, top_left, bottom_right):
41     """
42     Calculates the sum in the rectangle specified by the given tuples.
43     :param integral_img_arr:
44     :type integral_img_arr: numpy.ndarray
45     :param top_left: (x, y) of the rectangle's top left corner
46     :type top_left: (int, int)
47     :param bottom_right: (x, y) of the rectangle's bottom right corner
48     :type bottom_right: (int, int)
49     :return The sum of all pixels in the given rectangle
50     :rtype int
51     """
52     # swap tuples
53     top_left = (top_left[1], top_left[0])
54     bottom_right = (bottom_right[1], bottom_right[0])
55     if top_left == bottom_right:
56         return integral_img_arr[top_left]
57     top_right = (bottom_right[0], top_left[1])
58     bottom_left = (top_left[0], bottom_right[1])
59     return integral_img_arr[bottom_right] - integral_img_arr[top_right]
60     - integral_img_arr[bottom_left] + integral_img_arr[top_left]

```

recursos/codigo.python/Viola-Jones-master/violajones/IntegralImage.py

5.3 HaarLikeFeature.py

```

1 import violajones.IntegralImage as ii
2
3
4 def enum(**enums):
5     return type('Enum', (), enums)
6
7 FeatureType = enum(TWO_VERTICAL=(1, 2), TWO_HORIZONTAL=(2, 1),
8     THREE_HORIZONTAL=(3, 1), THREE_VERTICAL=(1, 3), FOUR=(2, 2))
9 FeatureTypes = [FeatureType.TWO_VERTICAL, FeatureType.TWO_HORIZONTAL,
10     FeatureType.THREE_VERTICAL, FeatureType.THREE_HORIZONTAL,
11     FeatureType.FOUR]
12
13 class HaarLikeFeature(object):
14     """
15     Class representing a haar-like feature.
16     """

```

```

15
16 def __init__(self, feature_type, position, width, height, threshold
17 , polarity):
18     """
19     Creates a new haar-like feature.
20     :param feature_type: Type of new feature, see FeatureType enum
21     :type feature_type: violajonse.HaarLikeFeature.FeatureTypes
22     :param position: Top left corner where the feature begins (x, y
23 )
24     :type position: (int, int)
25     :param width: Width of the feature
26     :type width: int
27     :param height: Height of the feature
28     :type height: int
29     :param threshold: Feature threshold
30     :type threshold: float
31     :param polarity: polarity of the feature -1 or 1
32     :type polarity: int
33     """
34     self.type = feature_type
35     self.top_left = position
36     self.bottom_right = (position[0] + width, position[1] + height)
37     self.width = width
38     self.height = height
39     self.threshold = threshold
40     self.polarity = polarity
41     self.weight = 1
42
43 def get_score(self, int_img):
44     """
45     Get score for given integral image array.
46     :param int_img: Integral image array
47     :type int_img: numpy.ndarray
48     :return: Score for given feature
49     :rtype: float
50     """
51     score = 0
52     if self.type == FeatureType.TWO_VERTICAL:
53         first = ii.sum_region(int_img, self.top_left, (self.
54 top_left[0] + self.width, int(self.top_left[1] + self.height / 2)))
55         second = ii.sum_region(int_img, (self.top_left[0], int(self
56 .top_left[1] + self.height / 2)), self.bottom_right)
57         score = first - second
58     elif self.type == FeatureType.TWO_HORIZONTAL:
59         first = ii.sum_region(int_img, self.top_left, (int(self.
60 top_left[0] + self.width / 2), self.top_left[1] + self.height))
61         second = ii.sum_region(int_img, (int(self.top_left[0] +
62 self.width / 2), self.top_left[1]), self.bottom_right)
63         score = first - second
64     elif self.type == FeatureType.THREE_HORIZONTAL:
65         first = ii.sum_region(int_img, self.top_left, (int(self.
66 top_left[0] + self.width / 3), self.top_left[1] + self.height))
67         second = ii.sum_region(int_img, (int(self.top_left[0] +
68 self.width / 3), self.top_left[1]), (int(self.top_left[0] + 2 *
69 self.width / 3), self.top_left[1] + self.height))
70         third = ii.sum_region(int_img, (int(self.top_left[0] + 2 *
71 self.width / 3), self.top_left[1]), self.bottom_right)
72         score = first - second + third
73     elif self.type == FeatureType.THREE_VERTICAL:

```

```

64         first = ii.sum_region(int_img, self.top_left, (self.
bottom_right[0], int(self.top_left[1] + self.height / 3)))
65         second = ii.sum_region(int_img, (self.top_left[0], int(self
.top_left[1] + self.height / 3)), (self.bottom_right[0], int(self.
top_left[1] + 2 * self.height / 3)))
66         third = ii.sum_region(int_img, (self.top_left[0], int(self.
top_left[1] + 2 * self.height / 3)), self.bottom_right)
67         score = first - second + third
68         elif self.type == FeatureType.FOUR:
69             # top left area
70             first = ii.sum_region(int_img, self.top_left, (int(self.
top_left[0] + self.width / 2), int(self.top_left[1] + self.height /
2)))
71             # top right area
72             second = ii.sum_region(int_img, (int(self.top_left[0] +
self.width / 2), self.top_left[1]), (self.bottom_right[0], int(self
.top_left[1] + self.height / 2)))
73             # bottom left area
74             third = ii.sum_region(int_img, (self.top_left[0], int(self.
top_left[1] + self.height / 2)), (int(self.top_left[0] + self.width
/ 2), self.bottom_right[1]))
75             # bottom right area
76             fourth = ii.sum_region(int_img, (int(self.top_left[0] +
self.width / 2), int(self.top_left[1] + self.height / 2)), self.
bottom_right)
77             score = first - second - third + fourth
78             return score
79
80     def get_vote(self, int_img):
81         """
82         Get vote of this feature for given integral image.
83         :param int_img: Integral image array
84         :type int_img: numpy.ndarray
85         :return: 1 iff this feature votes positively, otherwise -1
86         :rtype: int
87         """
88         score = self.get_score(int_img)
89         return self.weight * (1 if score < self.polarity * self.
threshold else -1)

```

recursos/codigo_python/Viola-Jones-master/violajones/HaarLikeFeature.py

5.4 AdaBoost.py

```

1 from functools import partial
2 import numpy as np
3 from violajones.HaarLikeFeature import HaarLikeFeature
4 from violajones.HaarLikeFeature import FeatureTypes
5 import progressbar
6 from multiprocessing import Pool
7
8 LOADING_BAR_LENGTH = 50
9
10
11 # TODO: select optimal threshold for each feature
12 # TODO: attentional cascading
13
14 def learn(positive_iis, negative_iis, num_classifiers=-1,
min_feature_width=1, max_feature_width=-1, min_feature_height=1,

```

```

max_feature_height=-1):
"""
15
16     Selects a set of classifiers. Iteratively takes the best
classifiers based
17     on a weighted error.
18     :param positive_iis: List of positive integral image examples
19     :type positive_iis: list[numpy.ndarray]
20     :param negative_iis: List of negative integral image examples
21     :type negative_iis: list[numpy.ndarray]
22     :param num_classifiers: Number of classifiers to select, -1 will
use all
23     classifiers
24     :type num_classifiers: int
25
26     :return: List of selected features
27     :rtype: list[violajones.HaarLikeFeature.HaarLikeFeature]
28     """
29     num_pos = len(positive_iis)
30     num_neg = len(negative_iis)
31     num_imgs = num_pos + num_neg
32     img_height, img_width = positive_iis[0].shape
33
34     # Maximum feature width and height default to image width and
height
35     max_feature_height = img_height if max_feature_height == -1 else
max_feature_height
36     max_feature_width = img_width if max_feature_width == -1 else
max_feature_width
37
38     # Create initial weights and labels
39     pos_weights = np.ones(num_pos) * 1. / (2 * num_pos)
40     neg_weights = np.ones(num_neg) * 1. / (2 * num_neg)
41     weights = np.hstack((pos_weights, neg_weights))
42     labels = np.hstack((np.ones(num_pos), np.ones(num_neg) * -1))
43
44     images = positive_iis + negative_iis
45
46     # Create features for all sizes and locations
47     features = _create_features(img_height, img_width,
min_feature_width, max_feature_width, min_feature_height,
max_feature_height)
48     num_features = len(features)
49     feature_indexes = list(range(num_features))
50
51     num_classifiers = num_features if num_classifiers == -1 else
num_classifiers
52
53     print('Calculating scores for images..')
54
55     votes = np.zeros((num_imgs, num_features))
56     bar = progressbar.ProgressBar()
57     # Use as many workers as there are CPUs
58     pool = Pool(processes=None)
59     for i in bar(range(num_imgs)):
60         votes[i, :] = np.array(list(pool.map(partial(_get_feature_vote,
image=images[i]), features)))
61
62     # select classifiers
63
64     classifiers = []

```



```

65
66 print('Selecting classifiers..')
67 bar = progressbar.ProgressBar()
68 for _ in bar(range(num_classifiers)):
69
70     classification_errors = np.zeros(len(feature_indexes))
71
72     # normalize weights
73     weights *= 1. / np.sum(weights)
74
75     # select best classifier based on the weighted error
76     for f in range(len(feature_indexes)):
77         f_idx = feature_indexes[f]
78         # classifier error is the sum of image weights where the
classifier
79         # is right
80         error = sum(map(lambda img_idx: weights[img_idx] if labels[
img_idx] != votes[img_idx, f_idx] else 0, range(num_imgs)))
81         classification_errors[f] = error
82
83     # get best feature, i.e. with smallest error
84     min_error_idx = np.argmin(classification_errors)
85     best_error = classification_errors[min_error_idx]
86     best_feature_idx = feature_indexes[min_error_idx]
87
88     # set feature weight
89     best_feature = features[best_feature_idx]
90     feature_weight = 0.5 * np.log((1 - best_error) / best_error)
91     best_feature.weight = feature_weight
92
93     classifiers.append(best_feature)
94
95     # update image weights
96     weights = np.array(list(map(lambda img_idx: weights[img_idx] *
np.sqrt((1 - best_error) / best_error) if labels[img_idx] != votes[
img_idx, best_feature_idx] else weights[img_idx] * np.sqrt(
best_error / (1 - best_error)), range(num_imgs))))
97
98     # remove feature (a feature can't be selected twice)
99     feature_indexes.remove(best_feature_idx)
100
101 return classifiers
102
103
104 def _get_feature_vote(feature, image):
105     return feature.get_vote(image)
106
107
108 def _create_features(img_height, img_width, min_feature_width,
max_feature_width, min_feature_height, max_feature_height):
109     print('Creating haar-like features..')
110     features = []
111     for feature in FeatureTypes:
112         # FeatureTypes are just tuples
113         feature_start_width = max(min_feature_width, feature[0])
114         for feature_width in range(feature_start_width,
max_feature_width, feature[0]):
115             feature_start_height = max(min_feature_height, feature[1])
116             for feature_height in range(feature_start_height,
max_feature_height, feature[1]):

```

```

117         for x in range(img_width - feature_width):
118             for y in range(img_height - feature_height):
119                 features.append(HaarLikeFeature(feature, (x, y)
, feature_width, feature_height, 0, 1))
120                 features.append(HaarLikeFeature(feature, (x, y)
, feature_width, feature_height, 0, -1))
121         print('..done. ' + str(len(features)) + ' features created.\n')
122         return features

```

recursos/codigo_python/Viola-Jones-master/viola_jones/AdaBoost.py

5.5 Utils.py

```

1 import numpy as np
2 from PIL import Image
3 from violajones.HaarLikeFeature import FeatureType
4 from functools import partial
5 import os
6
7
8 def ensemble_vote(int_img, classifiers):
9     """
10     Classifies given integral image (numpy array) using given
11     classifiers, i.e.
12     if the sum of all classifier votes is greater 0, image is
13     classified
14     positively (1) else negatively (0). The threshold is 0, because
15     votes can be
16     +1 or -1.
17     :param int_img: Integral image to be classified
18     :type int_img: numpy.ndarray
19     :param classifiers: List of classifiers
20     :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
21     :return: 1 iff sum of classifier votes is greater 0, else 0
22     :rtype: int
23     """
24     return 1 if sum([c.get_vote(int_img) for c in classifiers]) >= 0
25     else 0
26
27
28 def ensemble_vote_all(int_imgs, classifiers):
29     """
30     Classifies given list of integral images (numpy arrays) using
31     classifiers,
32     i.e. if the sum of all classifier votes is greater 0, an image is
33     classified
34     positively (1) else negatively (0). The threshold is 0, because
35     votes can be
36     +1 or -1.
37     :param int_imgs: List of integral images to be classified
38     :type int_imgs: list[numpy.ndarray]
39     :param classifiers: List of classifiers
40     :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
41     :return: List of assigned labels, 1 if image was classified
42     positively, else
43     0
44     :rtype: list[int]
45     """
46     vote_partial = partial(ensemble_vote, classifiers=classifiers)

```

```

39     return list(map(vote_partial, int_imgs))
40
41
42 def reconstruct(classifiers, img_size):
43     """
44     Creates an image by putting all given classifiers on top of each
45     other
46     producing an archetype of the learned class of object.
47     :param classifiers: List of classifiers
48     :type classifiers: list[violaJones.HaarLikeFeature.HaarLikeFeature]
49     :param img_size: Tuple of width and height
50     :type img_size: (int, int)
51     :return: Reconstructed image
52     :rtype: PIL.Image
53     """
54     image = np.zeros(img_size)
55     for c in classifiers:
56         # map polarity: -1 -> 0, 1 -> 1
57         polarity = pow(1 + c.polarity, 2)/4
58         if c.type == FeatureType.TWO.VERTICAL:
59             for x in range(c.width):
60                 sign = polarity
61                 for y in range(c.height):
62                     if y >= c.height/2:
63                         sign = (sign + 1) % 2
64                     image[c.top_left[1] + y, c.top_left[0] + x] += 1 *
65                     sign * c.weight
66             elif c.type == FeatureType.TWO.HORIZONTAL:
67                 sign = polarity
68                 for x in range(c.width):
69                     if x >= c.width/2:
70                         sign = (sign + 1) % 2
71                     for y in range(c.height):
72                         image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
73                         sign * c.weight
74             elif c.type == FeatureType.THREE.HORIZONTAL:
75                 sign = polarity
76                 for x in range(c.width):
77                     if x % c.width/3 == 0:
78                         sign = (sign + 1) % 2
79                     for y in range(c.height):
80                         image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
81                         sign * c.weight
82             elif c.type == FeatureType.THREE.VERTICAL:
83                 sign = polarity
84                 for y in range(c.height):
85                     if x % c.height/3 == 0:
86                         sign = (sign + 1) % 2
87                     image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
88                     sign * c.weight
89             elif c.type == FeatureType.FOUR:
90                 sign = polarity
91                 for x in range(c.width):
92                     if x % c.width/2 == 0:
93                         sign = (sign + 1) % 2
94                     for y in range(c.height):
95                         if x % c.height/2 == 0:
96                             sign = (sign + 1) % 2

```

```

93         image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
    sign * c.weight
94     image -= image.min()
95     image /= image.max()
96     image *= 255
97     result = Image.fromarray(image.astype(np.uint8))
98     return result
99
100
101 def load_images(path):
102     images = []
103     for _file in os.listdir(path):
104         if _file.endswith('.png'):
105             img_arr = np.array(Image.open((os.path.join(path, _file))),
106                               dtype=np.float64)
107             img_arr /= img_arr.max()
108             images.append(img_arr)
109     return images

```

recursos/codigo.python/Viola-Jones-master/violajones/Utils.py

5.6 example.py

```

1 import violajones.IntegralImage as ii
2 import violajones.AdaBoost as ab
3 import violajones.Utils as utils
4
5 if __name__ == "__main__":
6     pos_training_path = 'trainingdata/faces'
7     neg_training_path = 'trainingdata/nonfaces'
8     pos_testing_path = 'trainingdata/faces/test'
9     neg_testing_path = 'trainingdata/nonfaces/test'
10
11     num_classifiers = 2
12     # For performance reasons restricting feature size
13     min_feature_height = 8
14     max_feature_height = 10
15     min_feature_width = 8
16     max_feature_width = 10
17
18     print('Loading faces..')
19     faces_training = utils.load_images(pos_training_path)
20     faces_ii_training = list(map(ii.to_integral_image, faces_training))
21     print('..done. ' + str(len(faces_training)) + ' faces loaded.\n\
22 nLoading non faces..')
23     non_faces_training = utils.load_images(neg_training_path)
24     non_faces_ii_training = list(map(ii.to_integral_image,
25                                     non_faces_training))
26     print('..done. ' + str(len(non_faces_training)) + ' non faces
27 loaded.\n')
28
29     # classifiers are haar like features
30     classifiers = ab.learn(faces_ii_training, non_faces_ii_training,
31                           num_classifiers, min_feature_height, max_feature_height,
32                           min_feature_width, max_feature_width)
33
34     print('Loading test faces..')
35     faces_testing = utils.load_images(pos_testing_path)
36     faces_ii_testing = list(map(ii.to_integral_image, faces_testing))

```

```

32     print('..done. ' + str(len(faces_testing)) + ' faces loaded.\n\
nLoading test non faces..')
33     non_faces_testing = utils.load_images(neg_testing_path)
34     non_faces_ii_testing = list(map(ii.to_integral_image ,
non_faces_testing))
35     print('..done. ' + str(len(non_faces_testing)) + ' non faces loaded
.\n')
36
37     print('Testing selected classifiers..')
38     correct_faces = 0
39     correct_non_faces = 0
40     correct_faces = sum(utils.ensemble_vote_all(faces_ii_testing ,
classifiers))
41     correct_non_faces = len(non_faces_testing) - sum(utils.
ensemble_vote_all(non_faces_ii_testing , classifiers))
42
43     print('..done.\n\nResult:\n      Faces: ' + str(correct_faces) + '/'
+ str(len(faces_testing))
44           + ' (' + str((float(correct_faces) / len(faces_testing)) *
100) + '%)\n    non-Faces: '
45           + str(correct_non_faces) + '/' + str(len(non_faces_testing))
+ ' ('
46           + str((float(correct_non_faces) / len(non_faces_testing)) *
100) + '%)')
47
48     # Just for fun: putting all haar-like features over each other
generates a face-like image
49     recon = utils.reconstruct(classifiers , faces_testing[0].shape)
50     recon.save('reconstruction.png')

```

recursos/codigo_python/Viola-Jones-master/example.py

6 Referências Bibliográfica

VIOLA, Paul; JONES, Michael. **Rapid object detection using a boosted cascade of simple features**. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, v. 1, p. I-511-I-518, 2001. Disponível em: <<http://ieeexplore.ieee.org/document/990517/>>.

VIOLA, Paul; JONES, Michael. **Robust Real-Time Face Detection** International Journal of Computer Vision 57(2), 137–154, 2004.

CHAVES, Bruno Butilhão. **Estudo do algoritmo AdaBoost de aprendizagem de máquina aplicado a sensores e sistemas embarcados**. 2011. Dissertação (Mestrado em Engenharia de Controle e Automação Mecânica) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2011. doi:10.11606/D.3.2011.tde-12062012-163740. Acesso em: 2017-10-11..

IRGENS, Peter et al. **An efficient and cost effective FPGA based implementation of the Viola-Jones face detection algorithm**. HardwareX, v. 1, p. 68–75, 2017. Disponível em: <<http://linkinghub.elsevier.com/retrieve/pii/S2468067216300116>>.

SANTOS, Ligneul. **Detecção de faces através do algoritmo de Viola-Jones**. Coppe/Ufrj, 2011.

FAUX, Francis e LUTHON, Franck. **Theory of evidence for face detection and tracking**. International Journal of Approximate Reasoning, v. 53, n. 5, p. 728–746, 2012. Disponível em: <<http://dx.doi.org/10.1016/j.ijar.2012.02.002>>.

BODHI, S. R. e NAVEEN, S. **Face detection, registration and feature localization experiments with RGB-D face database**. Procedia Computer Science, v. 46, n. Ict 2014, p. 1778–1785, 2015. Disponível em: <<http://dx.doi.org/10.1016/j.procs.2015.02.132>>.