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
Brener 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

Outubro
2017

Sumário

Algoritmo Viola-Jones

1 O artigo

O método *Viola-Jones* proposto por *Paul Viola* e *Michael Jones* em 2001, é um algoritmo utilizado em diversas áreas da tecnologia, uma delas é a detecção de faces. Paul e Michael eram pesquisadores de Cambrigde onde optaram por explorar o lado radical da programação, sendo assim, publicaram um artigo intitulado: "*Rapid Object Detection using a Boosted Cascade of Simple Features*" que demonstrou uma nova forma de detectar faces. O artigo bem como o conceito Viola-Jones faz uso de 3 abordagens diferenciadas, pautando pontos considerados de extrema importância, sendo eles: as características de *Haar*, o algoritmo de aprendizado *AdaBoost* e os classificadores em Cascata.

As características de Haar proposta pelo matemático húngaro Alfred Haar em 1909, foi a primeira maneira considerada nova de representar uma "*imagem integral*" (*Integral Image*, em inglês), que permitiu os detectores utilizados por eles, computarem as imagens de forma rápida e eficaz. *Haar* é uma característica intitulada como transformada onde varia da matemática discreta que faz uso em diversos processos de análises de sinais, em meios de compressão de dados e amplas outras aplicações no ramo de engenharia e ciência da computação.

A segunda abordagem essencial foi o algoritmo de aprendizado baseado em *AdaBoost*. O algoritmo AdaBoost é caracterizado por algoritmo de *Machine Learning* (aprendizado de máquina, em português) inventado por Yoav Freund e Robert Schapire, o algoritmo meta-heurístico é usado para aumentar significativamente a performance dos algoritmos de aprendizagem. *AdaBoost* é derivado do nome *Adaptive Boosting* (impulso adaptativo, significado em português) algoritmo colocado como ajustável a diversas circunstâncias e adaptável para classificações subsequentes. O algoritmo seleciona um número pequeno de características visuais críticas de um conjunto maior e com seus classificadores torna-se extremamente eficiente.

Tendo em vista tais ferramentas úteis para elaboração do algoritmo, Paul e Michael implementaram os classificadores em Cascata. Os classificadores são responsáveis por selecionar criteriosamente um determinado objeto, levando em consideração a concomitância de suas principais características. As características necessárias para utilização são encontradas a partir de amplos algoritmos de aprendizado como *Support Vector Machine*, redes neurais, entre outros. Os classificadores em cascata possuem como principal função combinar e incrementar métodos que tendenciam melhorar a perspectiva do objeto, ou seja, permitiam com que diversas regiões do fundo da foto fossem rapidamente destacadas disponibilizando maior processamento computacional.

Por conseguinte, rejeitavam um largo número de regiões que demostravam ter o aspecto escolhido tornando-o então, cada vez mais prudente para que não ocorresse casos de falsos objetos nas análises dos mesmos. Os autores aprofundaram seus métodos de acordo com a construção do algoritmo, deste modo, realizaram comparações em relação a diferentes algoritmos similares da época, bem como, *Rowley-Baluja-Kanade*, *Schneiderman-Kanade* e *Roth-Yang-Ahuja* para que pudessem encontrar meios de solucionar a forma como funcionavam as equações e obter assim, resultados satisfatórios.

O algoritmo Viola-Jones é uma variação do *AdaBoost*, algoritmo de aprendizado. Porém,por ser uma alternância, o algoritmo Viola-Jones é bastante utilizado. Devido

sua implementação seguir uma abordagem diferente para construir um novo sistema de detecção, ele consegue alcançar aproximadamente 15 vezes mais rápido que os métodos anteriores. Viola-Jones é perfeitamente apto a detectar faces com maior precisão usando *Haar* como uma das implementações, é notório a perspectiva que há em aplicar o algoritmo em diversas situações. O método Viola-Jones revolucionou o campo da computação tornando-se referência para os demais.

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 Por quê 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 mais vantajoso desenvolver desde o início uma amplicação 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

```
import numpy as np


"""
In an integral image each pixel is the sum of all pixels in the original image
that are 'left and above' the pixel.

Original     Integral
+---------   +-------------
| 1 2 3 .    | 0  0  0  0 .
| 4 5 6 .    | 0  1  3  6 .
| . . . .    | 0  5 12 21 .
             | . . . . . . .

"""


def to_integral_image(img_arr):
    """
    Calculates the integral image based on this instance's original image data.
    :param img_arr: Image source data
    :type img_arr: numpy.ndarray
    :return Integral image for given image
    :rtype: numpy.ndarray
    """
    # an index of -1 refers to the last row/column
    # since row_sum is calculated starting from (0,0),
    # rowSum(x, -1) == 0 holds for all x
    row_sum = np.zeros(img_arr.shape)
    # we need an additional column and row
    integral_image_arr = np.zeros((img_arr.shape[0] + 1, img_arr.shape[1] + 1))
    for x in range(img_arr.shape[1]):
        for y in range(img_arr.shape[0]):
            row_sum[y, x] = row_sum[y-1, x] + img_arr[y, x]
            integral_image_arr[y+1, x+1] = integral_image_arr[y+1, x-1+1] + row_sum[y, x]
    return integral_image_arr


def sum_region(integral_img_arr, top_left, bottom_right):
    """
    Calculates the sum in the rectangle specified by the given tuples.
    :param integral_img_arr:
    :type integral_img_arr: numpy.ndarray
    :param top_left: (x, y) of the rectangle's top left corner
    :type top_left: (int, int)
    :param bottom_right: (x, y) of the rectangle's bottom right corner
    :type bottom_right: (int, int)
    :return The sum of all pixels in the given rectangle
```

```
    :rtype int
    """
    # swap tuples
    top_left = (top_left[1], top_left[0])
    bottom_right = (bottom_right[1], bottom_right[0])
    if top_left == bottom_right:
        return integral_img_arr[top_left]
    top_right = (bottom_right[0], top_left[1])
    bottom_left = (top_left[0], bottom_right[1])
    return integral_img_arr[bottom_right] - integral_img_arr[top_right]
    - integral_img_arr[bottom_left] + integral_img_arr[top_left]
```

recursos/codigo_python/Viola–Jones–master/violajones/IntegralImage.py

5.3 HaarLikeFeature.py

```
import violajones.IntegralImage as ii


def enum(**enums):
    return type('Enum', (), enums)

FeatureType = enum(TWO_VERTICAL=(1, 2), TWO_HORIZONTAL=(2, 1),
    THREE_HORIZONTAL=(3, 1), THREE_VERTICAL=(1, 3), FOUR=(2, 2))
FeatureTypes = [FeatureType.TWO_VERTICAL, FeatureType.TWO_HORIZONTAL,
    FeatureType.THREE_VERTICAL, FeatureType.THREE_HORIZONTAL,
    FeatureType.FOUR]


class HaarLikeFeature(object):
    """
    Class representing a haar-like feature.
    """

    def __init__(self, feature_type, position, width, height, threshold
    , polarity):
        """
        Creates a new haar-like feature.
        :param feature_type: Type of new feature, see FeatureType enum
        :type feature_type: violajonse.HaarLikeFeature.FeatureTypes
        :param position: Top left corner where the feature begins (x, y
    )
        :type position: (int, int)
        :param width: Width of the feature
        :type width: int
        :param height: Height of the feature
        :type height: int
        :param threshold: Feature threshold
        :type threshold: float
        :param polarity: polarity of the feature -1 or 1
        :type polarity: int
        """
        self.type = feature_type
        self.top_left = position
        self.bottom_right = (position[0] + width, position[1] + height)
        self.width = width
        self.height = height
        self.threshold = threshold
        self.polarity = polarity
```

```
39          self.weight = 1
40
41      def get_score(self, int_img):
42          """
43          Get score for given integral image array.
44          :param int_img: Integral image array
45          :type int_img: numpy.ndarray
46          :return: Score for given feature
47          :rtype: float
48          """
49          score = 0
50          if self.type == FeatureType.TWO_VERTICAL:
51              first = ii.sum_region(int_img, self.top_left, (self.
    top_left[0] + self.width, int(self.top_left[1] + self.height / 2)))
52              second = ii.sum_region(int_img, (self.top_left[0], int(self
    .top_left[1] + self.height / 2)), self.bottom_right)
53              score = first - second
54          elif self.type == FeatureType.TWO_HORIZONTAL:
55              first = ii.sum_region(int_img, self.top_left, (int(self.
    top_left[0] + self.width / 2), self.top_left[1] + self.height))
56              second = ii.sum_region(int_img, (int(self.top_left[0] +
    self.width / 2), self.top_left[1]), self.bottom_right)
57              score = first - second
58          elif self.type == FeatureType.THREE_HORIZONTAL:
59              first = ii.sum_region(int_img, self.top_left, (int(self.
    top_left[0] + self.width / 3), self.top_left[1] + self.height))
60              second = ii.sum_region(int_img, (int(self.top_left[0] +
    self.width / 3), self.top_left[1]), (int(self.top_left[0] + 2 *
    self.width / 3), self.top_left[1] + self.height))
61              third = ii.sum_region(int_img, (int(self.top_left[0] + 2 *
    self.width / 3), self.top_left[1]), self.bottom_right)
62              score = first - second + third
63          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/violajones/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
    classifiers, i.e.
```

```
    if the sum of all classifier votes is greater 0, image is classified
    positively (1) else negatively (0). The threshold is 0, because votes can be
    +1 or -1.
    :param int_img: Integral image to be classified
    :type int_img: numpy.ndarray
    :param classifiers: List of classifiers
    :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
    :return: 1 iff sum of classifier votes is greater 0, else 0
    :rtype: int
    """
    return 1 if sum([c.get_vote(int_img) for c in classifiers]) >= 0 else 0


def ensemble_vote_all(int_imgs, classifiers):
    """
    Classifies given list of integral images (numpy arrays) using classifiers,
    i.e. if the sum of all classifier votes is greater 0, an image is classified
    positively (1) else negatively (0). The threshold is 0, because votes can be
    +1 or -1.
    :param int_imgs: List of integral images to be classified
    :type int_imgs: list[numpy.ndarray]
    :param classifiers: List of classifiers
    :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
    :return: List of assigned labels, 1 if image was classified positively, else
    0
    :rtype: list[int]
    """
    vote_partial = partial(ensemble_vote, classifiers=classifiers)
    return list(map(vote_partial, int_imgs))


def reconstruct(classifiers, img_size):
    """
    Creates an image by putting all given classifiers on top of each other
    producing an archetype of the learned class of object.
    :param classifiers: List of classifiers
    :type classifiers: list[violajones.HaarLikeFeature.HaarLikeFeature]
    :param img_size: Tuple of width and height
    :type img_size: (int, int)
    :return: Reconstructed image
    :rtype: PIL.Image
    """
    image = np.zeros(img_size)
    for c in classifiers:
        # map polarity: -1 -> 0, 1 -> 1
        polarity = pow(1 + c.polarity, 2)/4
        if c.type == FeatureType.TWO_VERTICAL:
            for x in range(c.width):
                sign = polarity
                for y in range(c.height):
                    if y >= c.height/2:
                        sign = (sign + 1) % 2
```

```
                    image[c.top_left[1] + y, c.top_left[0] + x] += 1 *
    sign * c.weight
            elif c.type == FeatureType.TWO_HORIZONTAL:
                sign = polarity
                for x in range(c.width):
                    if x >= c.width/2:
                        sign = (sign + 1) % 2
                    for y in range(c.height):
                        image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
    sign * c.weight
            elif c.type == FeatureType.THREE_HORIZONTAL:
                sign = polarity
                for x in range(c.width):
                    if x % c.width/3 == 0:
                        sign = (sign + 1) % 2
                    for y in range(c.height):
                        image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
    sign * c.weight
            elif c.type == FeatureType.THREE_VERTICAL:
                for x in range(c.width):
                    sign = polarity
                    for y in range(c.height):
                        if x % c.height/3 == 0:
                            sign = (sign + 1) % 2
                        image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
    sign * c.weight
            elif c.type == FeatureType.FOUR:
                sign = polarity
                for x in range(c.width):
                    if x % c.width/2 == 0:
                        sign = (sign + 1) % 2
                    for y in range(c.height):
                        if x % c.height/2 == 0:
                            sign = (sign + 1) % 2
                        image[c.top_left[0] + x, c.top_left[1] + y] += 1 *
    sign * c.weight
        image -= image.min()
        image /= image.max()
        image *= 255
        result = Image.fromarray(image.astype(np.uint8))
        return result


def load_images(path):
        images = []
        for _file in os.listdir(path):
            if _file.endswith('.png'):
                img_arr = np.array(Image.open((os.path.join(path, _file))),
        dtype=np.float64)
                img_arr /= img_arr.max()
                images.append(img_arr)
        return images
```

recursos/codigo_python/Viola–Jones–master/violajones/Utils.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. Icict 2014, p. 1778–1785, 2015. Disponível em: <http://dx.doi.org/10.1016/j.procs.2015.02.132>.