Inteligência Artificial

Reconhecimento de Movimentos de Mãos

Objetivo

Através de exemplos de vídeos (explodidos em frames) de 9 tipos diferentes de movimentos de mãos, extrair características usando a biblioteca OpenCV, e usar a base de dados gerada para treinar algoritmos de machine learning com auxílio do Weka.

Seleção de Frames

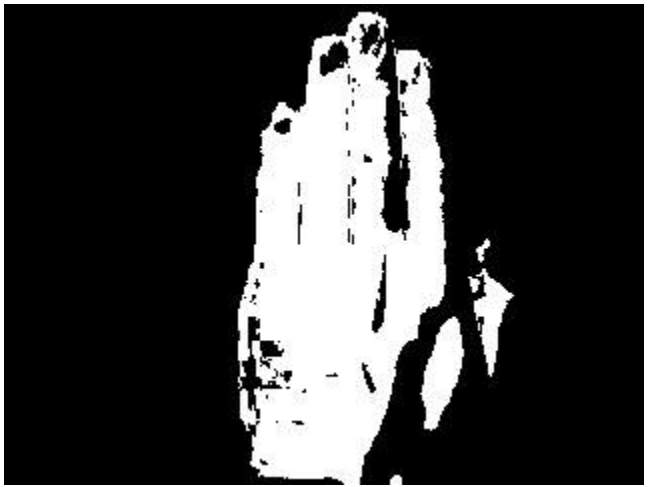
O programa extrator recebe como input o número de frames a serem extraídos de cada exemplo

Por exemplo, N=5:

0%, 25%, 50%, 75%, 100%

Segmentação das Imagens

Usando HSV com a faixa de cor da pele



[0,30,60] to [50,150,255]

Extração de Características

1. Achar o contour da imagem segmentada com maior área
2. Derivar primitivas: retângulo de área, mínima, elipse englobante, convex hull
3. Derivar características:

area, perimeter, convex_hull_area, solidity, rect_center_x, rect_center_y, rect_width, rect_height, rect_angle, rect_aspect_ratio, ellipse_center_x, ellipse_center_y, ellipse_major_axis, ellipse_minor_axis, ellipse_angle, farthest_convex_defect, hu_moment_1, hu_moment_2, hu_moment_3, hu_moment_4, hu_moment_5, hu_moment_6, hu_moment_7

Arquivo .arff

Script que faz a extração das características e escreve num arquivo.

Neste exemplo, usa 5 frames por exemplo e usa todos os sets (1 a 5), salvando num arquivo:

```
$ python extractor/arff_generator.py 5 1 2 3 4 5 > All.arff
```

Testando as Funções

Os seguintes testes usam somente 2 frames por exemplo.

Os parâmetros foram variados agressivamente.

Árvore de Decisão

binarySplits had no influence on the results.

confidenceFactor did not impact much on classification performance, but higher levels slowed down the classification quite a bit. We maintained it at 0.25.

minNumObj was optimal at 10.

reducedErrorPruning deemed worse results, so we turned it off.

```
Time taken to build model: 0.33 seconds
Correctly Classified Instances         634               70.4444 %
Kappa statistic                          0.6675
```

K-Nearest Neighbors

KNN was optimal at 1.

distanceWeighting had no influence on the results.

meanSquares had no influence on the results.

nearestNeighbourSearchAlgorithm we tried different ones, but all yielded the same result.

```
Time taken to build model: 0 seconds
Correctly Classified Instances         712               79.1111 %
Kappa statistic                          0.765
```

SVM

SVMType nu-CSV was better than C-CSV.

coef0 1.0 (small influence).

cost 1.0 (no influence).

degrees was optimal at 3

nu was optimal at 0.1

Time taken to build model: 1.75 seconds

Correctly Classified Instances	771	85.6667 %
Kappa statistic	0.8388	

Neural Network

hiddenLayers t (attribs+classes)

learningRate 0.3

momentum 0.5

trainingTime 500 (diminishing returns).

Time taken to build model: 22.3 seconds

Correctly Classified Instances	780	86.6667 %
Kappa statistic	0.85	

Testando as Características

Usando os parâmetros ótimos encontrados anteriormente, variamos as características para descobrir se há alguma que influencia negativamente os resultados.

Para isso, foi usando o Set1 com 2 frames.

Testando as Características

A retirada de características se mostrou prejudicial ao desempenho dos algoritmos de classificação.

Exceto por “`farthest_convex_defect`”, que indica o maior defeito de convexidade.

Farthest Convex Defect

Rede Neural

```
Com: 96.1111 %
Sem: 96.6667 %
```

Outros classificadores apresentaram resultados similares

Maximizando

Somente com o Set 1, com 10 frames/exemplo, usando os parâmetros relatados anteriormente.

Mostramos também as matrizes de confusão.

Árvore de Decisão

```
Time taken to build model: 0.15 seconds
Correctly Classified Instances         141               78.3333 %
Kappa statistic                          0.7563
  a  b  c  d  e  f  g  h  i   <-- classified as
 15  1  1  0  0  0  3  0  0 |  a = 0000
  0 16  0  0  1  0  0  2  1 |  b = 0001
  1  0 16  0  0  0  2  0  1 |  c = 0002
  1  0  0 18  0  0  1  0  0 |  d = 0003
  0  0  1  0 18  0  0  0  1 |  e = 0004
  0  1  0  0  3 13  1  0  2 |  f = 0005
  1  0  0  3  0  1 15  0  0 |  g = 0006
  0  0  1  0  2  0  0 15  2 |  h = 0007
  0  0  0  0  1  1  2  1 15 |  i = 0008
```

K-Nearest Neighbors

```
Time taken to build model: 0 seconds
Correctly Classified Instances         164               91.1111 %
Kappa statistic                          0.9
  a  b  c  d  e  f  g  h  i   <-- classified as
 17  0  0  2  0  0  1  0  0 |  a = 0000
  0 19  0  0  1  0  0  0  0 |  b = 0001
  0  0 19  0  0  0  0  0  1 |  c = 0002
  0  0  0 17  0  0  3  0  0 |  d = 0003
  0  1  0  0 18  0  0  1  0 |  e = 0004
  0  1  0  0  0 18  0  0  1 |  f = 0005
  0  0  0  0  0  0 20  0  0 |  g = 0006
  0  0  0  0  0  0  0 18  2 |  h = 0007
  0  0  0  0  0  0  1  1 18 |  i = 0008
```

SVM

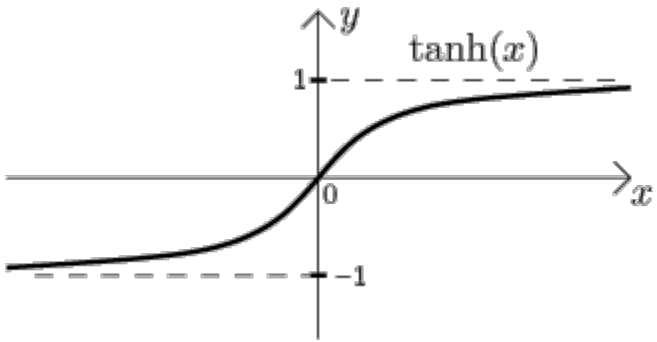
```
Time taken to build model: 1.13 seconds
Correctly Classified Instances         172               95.5556 %
Kappa statistic                          0.950
  a  b  c  d  e  f  g  h  i   <-- classified as
 20  0  0  0  0  0  0  0  0 |  a = 0000
  0 19  0  0  0  1  0  0  0 |  b = 0001
  0  0 19  0  0  0  0  0  1 |  c = 0002
  0  0  0 19  0  0  1  0  0 |  d = 0003
  0  0  0  0 19  0  0  1  0 |  e = 0004
  0  0  0  0  0 20  0  0  0 |  f = 0005
  0  0  0  0  0  0 20  0  0 |  g = 0006
  0  0  0  0  0  0  0 20  0 |  h = 0007
  0  0  0  0  0  0  0  0 20 |  i = 0008
```

Neural Network

```
Time taken to build model: 89.95 seconds
Correctly Classified Instances         174               96.6667 %
Kappa statistic                          0.9625
  a  b  c  d  e  f  g  h  i   <-- classified as
 20  0  0  0  0  0  0  0  0 |  a = 0000
  0 19  0  0  0  1  0  0  0 |  b = 0001
  0  0 20  0  0  0  0  0  0 |  c = 0002
  0  0  0 19  0  0  1  0  0 |  d = 0003
  0  0  0  0 19  0  0  1  0 |  e = 0004
  0  0  0  0  0 20  0  0  0 |  f = 0005
  0  0  0  0  0  0 20  0  0 |  g = 0006
  0  0  0  0  0  0  0 20  0 |  h = 0007
  0  0  1  0  0  0  2  0 17 |  i = 0008
```

Implementação

Implementação em Ruby de um algoritmo de Rede Neural com backpropagation



Implementação - XOR

2 input, 1 output, 2 hidden cells in 1 hidden layer

Uso:

```
xor = [
  [[0,0], [0]],
  [[0,1], [1]],
  [[1,0], [1]],
  [[1,1], [0]]]
NeuralNetwork
.new(2, 2, 1)
.train(xor)
.test(xor)
.internals
```

Resultado:

```
Got 4/4 -> 100.0%

Input Activation:
1.0
1.0
1.0

Hidden Layer Activation:
-0.9999648742143428
-0.9678124516804766

Output Activation:
-0.19664136237293744

Input Weights:
[[-1.9706870887241765, -3.794446904016083],
 [-1.9823678135520832, -3.811420648109506],
 [1.9021664183757332,   2.1829678415844036]]

Output Weights:
[[2.6446273179756994],
 [-2.810229779367159]]
```

Inteligência Artificial - INF1771 - 2014.1

PUC-RIO - Professor Edirlei

Guilherme Berger - 1210518