Layered BDM as a Texture and Weighted Network Descriptor to Estimate Kolmogorov Complexity

September 3rd, 2016

Author: Antonio Rueda-Toicen

antonio.rueda.toicen@algorithmicnaturelab.org
antonio.rueda.toicen@gmail.com

A layered version of the Block Decomposition Method[1], serves as a descriptor of both weighted networks and grayscale or color images. This descriptor provides an estimate of Kolmogorov Complexity that's sensitive to morphological perturbative [2]. To estimate the complexity of a grayscale texture, we quantize it and aggregate the estimated Kolmogorov complexity values of binary 4 x 4 squares, estimated through the Coding Theorem Method [3, 4].

In[62]:=
```
Clear["Global`*"]
SetDirectory[NotebookDirectory[]]
```

Out[63]= C:\Users\antonio\Dropbox\LayeredBDM

In[64]:=
```
data = Import["fourByFourCTMs.csv", "CSV", "Numeric" → False]
```

Out[64]=
```
{{0000000000000000, 22.006706292292176}, {0000000000000001, 23.347935957593144},
 {0000000000000010, 24.325701071360243}, {0000000000000011, 24.60462140484821},
 ··· 65 529 ···, {1111111111111101, 24.325701071360243},
 {1111111111111110, 23.347935957593144}, {1111111111111111, 22.006706292292176}}
```

salida grande | Mostrar menos | Mostrar más | Mostrar salida completa | Establecer límite de tamaño

In[65]:=
```
fourByFourCTMs = Transpose@{data[[All, 1]], ToExpression /@ data[[All, 2]] };
```
transposición · todo · convierte en expresión · todo

In[66]:=
```
Table[
  (CTM[fourByFourCTMs[[i, 1]]] = fourByFourCTMs[[i, 2]]), {i, 1, 65536, 1 }];
```
tabla

In[67]:=
```
CTM["0000000000000000"]
```

Out[67]= 22.0067

In[68]:=
```
Clear[data, fourByFourCTMs]
```
borra

"Layered BDM" works through the binary quantization of a texture's digital levels. The exam-

ples below quantize textures using 256 digital levels (byte resolution); 2^{16}, 2^{32}, etc. quantizations are obviously also possible.

In[69]:=
```
layerDecomposition[image_] :=
 Module[{getLayers, getBlocks, blockCount, stringifiedBlocks},
 módulo
  getLayers[imag_] := ParallelTable[Unitize[
                      tabla en paralelo  uno excepto en zero
     ImageData[ColorConvert[imag, "Grayscale"], "Byte"], i], {i, 1, 255, 1}];
               convierte colores                      byte
  getBlocks[layers_] := Nest[Flatten[#, 1] &,
                        anida aplana
    Partition[#, {4, 4}, 1] & /@ layers, 2];
    particiona
  blockCount = Tally[getBlocks[getLayers[image]]];
               recuenta
  stringifiedBlocks =
   StringJoin /@ Map[ToString, (Flatten /@ blockCount[[All, 1]]), {2}];
                 apl··· convierte a ···  aplana         todo
  Total[CTM /@ stringifiedBlocks] + Total[Log2[blockCount[[All, 2]]]]
                                    total logaritmo en base 2  todo
 ]
```

Layered BDM as a Texture Descriptor

In[70]:=
```
woodTextures =
  (Image[ColorConvert[#, "Grayscale"], "Byte"] & /@ ImageResize[#, {128, 128}] & /@
   imagen convierte colores              byte           reescalado de imagen
    (ExampleData[#] & /@
     datos de ejemplos
      {{"Texture", "Wood"}, {"Texture", "Wood2"}, {"Texture", "Wood3"}}));
        textura               textura                textura
```

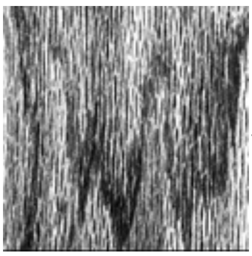
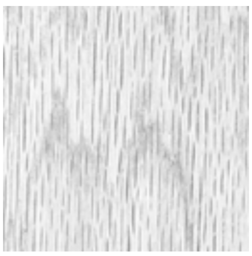
In[71]:=
```
smallWT = ImageResize[woodTextures[[1]] , {15, 15}]
          reescalado de imagen
```

Out[71]= 

In[72]:=
```
layerDecomposition[smallWT]
```

Out[72]= 28 364.8

In[73]:=
```
woodTextures
```

Out[73]= {  ,  ,  }

BDM seems to align with the intuitive notion of a texture's complexity.

In[74]:=
```
layerDecomposition /@ woodTextures
```

Out[74]= {457 899., 569 941., 398 956.}

Test on MR Image with/without an Artifact

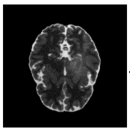
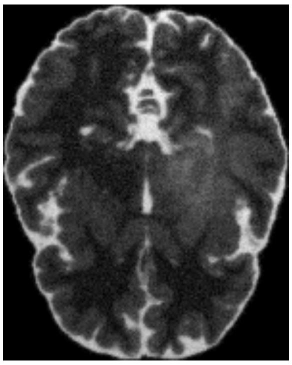
In[75]:=
```
t2BrainSlice = Image[ColorConvert[ImageCrop@
```

```
, "Grayscale"], "Byte"]
```

imagen convierte colores recorta imagen byte

Out[75]=


In[76]:=
```
t2BrainSlice // ImageData // Dimensions
```

datos de imagen dimensiones

Out[76]= {185, 149}

In[77]:=
```
aImage = ColorNegate@
  ColorConvert[Image[Rasterize[Style["a", FontSize → 50]], "Byte"], "Grayscale"]
```

niega color

convierte colores imagen convierte e⋯ estilo tamaño de tipo de letra byte

Out[77]=


In[78]:=
```
brainWLetter =
 Image[ColorConvert[ImageAdd[t2BrainSlice, aImage], "Grayscale"], "Byte"]
```

imagen convierte colores añade a imagen byte

Out[78]=


In[79]:=
```
brainWLetter // ImageData // Dimensions
```

datos de imagen dimensiones

Out[79]= {185, 149}

The image with a simple texture artifact has lower BDM and lower texture complexity.

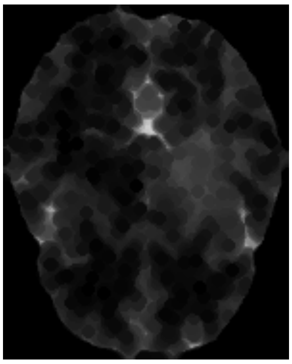
In[80]:=
```
layerDecomposition /@ {t2BrainSlice, brainWLetter}
```

Out[80]= $\{1.39659 \times 10^6, 1.3826 \times 10^6\}$

Sensitivity to Morphological Perturbation

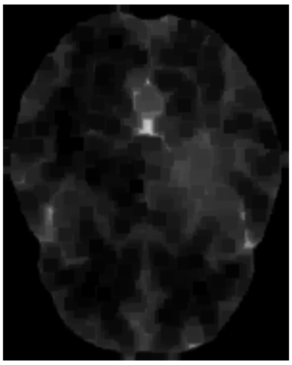
In[93]:=
```
t2BrainSliceWDiskErosion = Erosion[t2BrainSlice, DiskMatrix[3]]
```
erosión matriz disco

Out[93]=



In[92]:=
```
t2BrainSliceWBoxedErosion = Erosion[t2BrainSlice, BoxMatrix[3]]
```
erosión matriz caja

Out[92]=



LayeredBDM is highly sensitive to morphological perturbations of the data.

In[94]:=
```
layerDecomposition /@ {t2BrainSliceWBoxedErosion, t2BrainSliceWDiskErosion}
```

Out[94]= {59 706.4, 66 779.8}

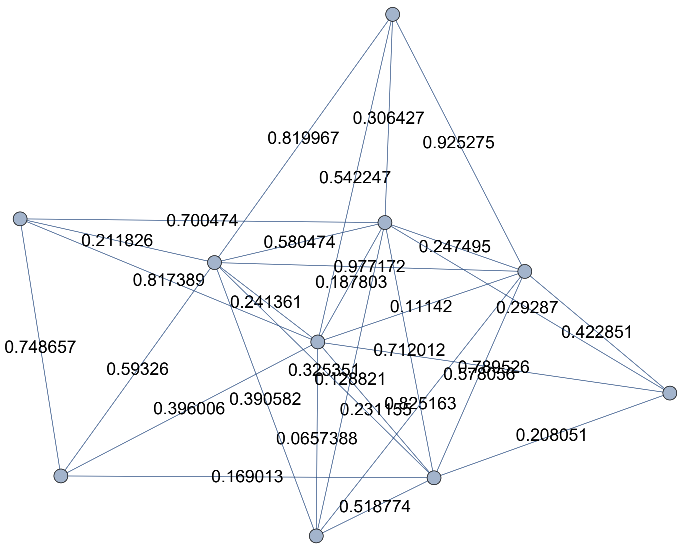
Layered BDM as a Weighted Network Descriptor

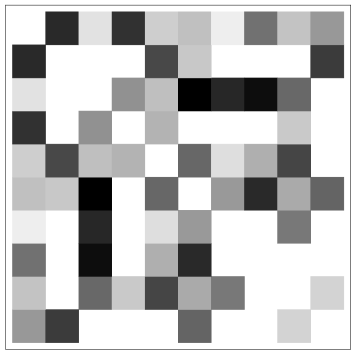
BDM can be used to evaluate networks trained through backpropagation.

In[81]:=
```
SeedRandom[1]; rg = RandomGraph[{10, 30}];
```
semilla aleatoria grafo aleatorio

In[82]:=
```
SeedRandom[1];
semilla aleatoria
{wrg = Graph[EdgeList[rg], EdgeWeight → RandomReal[1, Length[EdgeList[rg]]],
      grafo   lista de aristas    peso de arista  real aleatorio  longitud  lista de aristas
   EdgeLabels → "EdgeWeight", ImageSize → Medium],
   etiquetas de arista  peso de arista  tamaño de i···  tamaño medio
 ArrayPlot@WeightedAdjacencyMatrix[wrg]}
 representac···  matriz de adyacencia ponderada
```

Out[82]= {



,  }

In[83]:=
```
floatToByte[float_] := Floor[If[float === 1.0, 255, float * 256.0]]
                       entero··  si
```

In[84]:=
```
floatToByte /@ {0.001, 0.999}
```

Out[84]= {0, 255}

In[85]:=
```
byteWeightedMatrix = Map[floatToByte, Normal[WeightedAdjacencyMatrix[wrg]], {2}];
                     aplica a todos   normal  matriz de adyacencia ponderada
MatrixForm[byteWeightedMatrix]
forma de matriz
```

Out[85]//MatrixForm=

$$\begin{pmatrix} 0 & 209 & 28 & 202 & 48 & 61 & 16 & 138 & 59 & 101 \\ 209 & 0 & 0 & 0 & 179 & 54 & 0 & 0 & 0 & 191 \\ 28 & 0 & 0 & 108 & 63 & 250 & 211 & 236 & 147 & 0 \\ 202 & 0 & 108 & 0 & 74 & 0 & 0 & 0 & 53 & 0 \\ 48 & 179 & 63 & 74 & 0 & 148 & 32 & 78 & 182 & 0 \\ 61 & 54 & 250 & 0 & 148 & 0 & 99 & 209 & 83 & 151 \\ 16 & 0 & 211 & 0 & 32 & 99 & 0 & 0 & 132 & 0 \\ 138 & 0 & 236 & 0 & 78 & 209 & 0 & 0 & 0 & 0 \\ 59 & 0 & 147 & 53 & 182 & 83 & 132 & 0 & 0 & 43 \\ 101 & 191 & 0 & 0 & 0 & 151 & 0 & 0 & 43 & 0 \end{pmatrix}$$

In[86]:=
```
layerDecompositionForWeightedGraphs[weightedGraph_] :=
 Module[{floatToByte, getLayers, getBlocks,
 módulo
   blockCount, stringifiedBlocks, weightedAdjMatrix},
  floatToByte[float_] := Floor[If[float === 1.0, 255, float * 256.0]];
                         entero· si
  getLayers[w_] := ParallelTable[
                   tabla en paralelo
    Unitize[ Map[floatToByte, weightedAdjMatrix , {2}], i], {i, 1, 255, 1}];
             aplica a todos
  getBlocks[layers_] := Nest[Flatten[#, 1] &,
                        anida aplana
    Partition[#, {4, 4}, 1] & /@ layers, 2];
    particiona
  weightedAdjMatrix = Normal[WeightedAdjacencyMatrix[weightedGraph]];
                      normal matriz de adyacencia ponderada
  blockCount = Tally[getBlocks[getLayers[weightedAdjMatrix]]];
               recuenta
  stringifiedBlocks =
   StringJoin /@ Map[ToString, (Flatten /@ blockCount[[All, 1]]), {2}];
                 apl·· convierte a ··· aplana    todo
  Total[CTM /@ stringifiedBlocks] + Total[Log2[blockCount[[All, 2]]]]
                                    total logaritmo en base 2  todo
 ]
```

In[87]:=
```
layerDecompositionForWeightedGraphs[wrg]
```

Out[87]= 12 323.2

References

[1] Hector Zenil, Santiago Hernández - Orozco, Narsis A.Kiani, Fernando Soler - Toscano, Antonio Rueda - Toicen, and Jesper Tegner "A Decomposition Method for Global Evaluation of Shannon Entropy and Local Estimations of Algorithmic Complexity", https://arxiv.org/abs/1609.00110

[2] Antonio Rueda-Toicen, Narsis A. Kiani, and Hector Zenil, "Morphological Image Analysis through Estimations of Kolmogorov Complexity" (in preparation)

[3] Fernando Soler - Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit (2014) "Calculating Kolmogorov Complexity from the Output Frequency Distributions of Small Turing Machines." PLoS ONE 9 (5) : e96223.

[4] Hector Zenil, Fernando Soler - Toscano, K. Dingle.and Aard Louis (2014) "Correlation of Automorphism Group Size and Topological Properties with Program-size Complexity Evaluations of Graphs and Complex Networks", Physica A : Statistical Mechanics and its Applications, vol.404, pp.341–358.