Artefact

November 2, 2021

[1]:
```
import numpy as np
import geopandas as gpd
import matplotlib.pyplot as plt
from shapely.geometry import Point, Polygon
import folium
from sklearn.metrics import cohen_kappa_score
from google.colab import drive
```

[ ]:
```
drive.mount('/content/drive/')
```

[2]:
```
m = folium.Map(location=[-25.72,28.11], tiles='CartoDB positron', zoom_start=15)
```

[3]:
```
melusi_area = gpd.read_file('DataFiles/GTI_Data_Delivery20210902/SHP/Melusi_Area.
 ↪shp')
```

[4]:
```
for _, r in melusi_area.iterrows():
    sim_geo = gpd.GeoSeries(r['geometry']).simplify(tolerance=0.001)
    geo_j = sim_geo.to_json()
    geo_j = folium.GeoJson(data=geo_j,
                           style_function=lambda x: {'fillColor': 'blue'})
    geo_j.add_to(m)
```

[7]:
```
m
```

[7]:
```
<folium.folium.Map at 0x23654803a90>
```

[6]:
```
m.save('melusi_area.html')
```

[ ]:
```
melusi_area_2010 = gpd.read_file('DataFiles/GTI_Data_Delivery20210902/SHP/
 ↪Melusi_Building_Based_Land_Use_Points_2010.shp')
melusi_area_2020 = gpd.read_file('DataFiles/GTI_Data_Delivery20210902/SHP/
 ↪Melusi_Building_Based_Land_Use_Points_2020.shp')
roads = gpd.read_file('DataFiles/roads_lines_shp/hotosm_zaf_roads_lines.shp')
population = rioxarray.open_rasterio("/content/drive/MyDrive/HonsProj-Data/
 ↪population_zaf_2019-07-01_geotiff/population_zaf_2019-07-01.tif")
population = population.rio.clip_box(minx=xmin, miny=ymin, maxx=xmax, maxy=ymax)
```

[ ]:
```
melusi_area.crs
```

[ ]:
```
xmin, ymin, xmax, ymax = melusi_area.total_bounds
```

[ ]:
```
roads = roads.cx[xmin:xmax, ymin:ymax]
population = population.cx[xmin:xmax, ymin:ymax]
```

[ ]:
```
melusi_area.plot(color="black")
plt.savefig("melusi-area.jpg",dpi=1200)
```

[ ]:
```
ax = melusi_area.plot(color="black")
ax.set_axis_off()
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
plt.savefig("melusi-area-nox.jpg",dpi=1200)
```

[ ]:
```
melusi_area2010 = gpd.read_file('/content/drive/MyDrive/HonsProj-Data/
 ↪GTI_Data_Delivery20210902/SHP/Melusi_Building_Based_Land_Use_Points_2010.shp')
melusi_area2010 = melusi_area2010.cx[xmin:xmax, ymin:ymax]
ax = melusi_area2010.plot(color='black', figsize=(7,5))
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
plt.savefig("melusi-area2010.jpg",dpi=1200)
```

[ ]:
```
ax = melusi_area2010.plot(color="black", figsize=(7,5))
ax.set_axis_off()
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
ax.autoscale(enable=False)
plt.savefig("melusi-area2010-nox.jpg",dpi=1200)
```

[ ]:
```
melusi_area2020 = gpd.read_file('/content/drive/MyDrive/HonsProj-Data/
 ↪GTI_Data_Delivery20210902/SHP/Melusi_Building_Based_Land_Use_Points_2020.shp')
melusi_area2020 = melusi_area2020.cx[xmin:xmax, ymin:ymax]
ax = melusi_area2020.plot(color="black",figsize=(7,5))
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
ax.autoscale(enable=False)
plt.savefig("melusi-area2020.jpg",dpi=1200)
```

[ ]:
```
ax = melusi_area2020.plot(color="black",figsize=(7,5))
ax.set_axis_off()
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
ax.autoscale(enable=False)
plt.savefig("melusi-area2020-nox.jpg",dpi=1200)
```

[ ]:
```
roads = gpd.read_file('/content/drive/MyDrive/HonsProj-Data/roads_lines_shp/
 ↪hotosm_zaf_roads_lines.shp')
roads = roads.cx[xmin:xmax, ymin:ymax]
ax = roads.plot(color='black', figsize=(7,5))
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
plt.savefig("roads.jpg",dpi=1200)
```

[ ]:
```
ax = roads.plot(color="black", figsize=(7,5))
ax.set_axis_off()
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
ax.autoscale_view(scalex=False, scaley=False)
ax.autoscale(enable=False)
ax.autoscale(enable=False)
plt.savefig("roads-nox.jpg",dpi=1200)
```

[ ]:
```
import rioxarray
pop = rioxarray.open_rasterio("/content/drive/MyDrive/HonsProj-Data/
 ↪population_zaf_2019-07-01_geotiff/population_zaf_2019-07-01.tif")
pop = pop.rio.clip_box(minx=xmin, miny=ymin, maxx=xmax, maxy=ymax)
```

[ ]:
```
print(pop.rio.crs)
print(pop.rio.nodata)
print(pop.rio.bounds())
print(pop.rio.width)
print(pop.rio.height)
```

[ ]:
```
pop
```

[ ]:
```
pop.plot(figsize=(7,5))
```

[ ]:
```
ax = pop.plot(figsize=(7,5))
#ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
#ax.autoscale_view(scalex=False, scaley=False)
#ax.autoscale(enable=False)
plt.savefig("population.jpg",dpi=1200)
```

[ ]:
```
ax = pop.plot()
#ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))
#ax.autoscale_view(scalex=False, scaley=False)
#ax.autoscale(enable=False)
#ax.get_legend().remove()
plt.axis('off')
plt.savefig("population-nox.jpg",dpi=1200)
```

[ ]:
```
!pip install opencv-contrib-python
```

[ ]:
```
import cv2
```

[ ]:
```
image = cv2.imread('2019roads.png')
roads_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#cv2.imshow('Original image',image)
#cv2.imwrite('2019roads_gray.png', roads_gray)
```

[ ]:
```
scale_percent = 33.333333333 # percent of original size
width = int(roads_gray.shape[1] * scale_percent / 100)
height = int(roads_gray.shape[0] * scale_percent / 100)
dim = (width, height)
roads_gray_resized = cv2.resize(roads_gray, dim, interpolation = cv2.INTER_AREA)
```

[ ]:
```
def replaceColor(x):
  if x > 230:
    return 255
  elif x < 230 and x > 0:
    return 0
  else:
    return 0
```

[ ]:
```
vectorizeFunction = np.vectorize(replaceColor)
roads_gray_resized = vectorizeFunction(roads_gray_resized)
```

[ ]:
```
roads_gray_resized.shape
```

[ ]:
```
np.unique(roads_gray_resized, return_counts=True)
```

[ ]:
```
def replacePopulationColor(x):
  if x > 20 and x < 40:
    return 30
  elif x < 110 and x > 90:
    return 99
  elif x > 135 and x < 155:
    return 146
  elif x > 190 and x < 205:
    return 199
  else:
    return 255
```

[ ]:
```
#Unique values
#0.5483527804562259
```

```
#1.9322995070209252
#2.494171667535703
#2.5802248275941975
#2.928383276590716
#3.843662634917949
#3.9742969374931736
#4.0719211311237515
#5.236094447939968
#5.558666899272438
#nan
```

[ ]:

```
vectorizePopulationFunction = np.vectorize(replacePopulationColor)
```

[ ]:

```
population_image = cv2.imread('2019population.png')
population_gray = cv2.cvtColor(population_image, cv2.COLOR_BGR2GRAY)
population_gray_resized = cv2.resize(population_gray, dim, interpolation = cv2.
 ↪INTER_AREA)
#cv2.imwrite('population_gray_resized_0s1s.png', population_gray_resized)
population_gray_resized = vectorizePopulationFunction(population_gray_resized)
cv2.imwrite('population_gray_resized_0s1s.png', population_gray_resized)
```

[ ]:

```
def replaceGrayscaleWithPopulationDensity(x):
  if x == 30:
    return 0.5483527804562259
  elif x == 99:
    return 2.5802248275941975
  elif x == 146:
    return 4.0719211311237515
  elif x == 199:
    return 5.236094447939968
  else:
    return 0.1
```

[ ]:

```
vectorizePopulationDensityFunction = np.
 ↪vectorize(replaceGrayscaleWithPopulationDensity)
population = vectorizePopulationDensityFunction(population)
```

[ ]:

```
def replaceColorsInPopulation(x):
  if x < 220 and x > 195:
    return 215 # Yellows in original picture
  elif x > 25 and x < 35:
    return 30 # Purples in original picture
  elif x > 80 and x < 115:
    return 99 # Blues in original picture
  elif x > 130 and x < 150:
    return 146 # Greens in original picture
  else:
    return 255 # To replace background colors
```

[ ]:

```
vectorizePopulationColorFunction = np.vectorize(replaceColorsInPopulation)
population = vectorizePopulationColorFunction(population)
```

[ ]:

```
def replaceGrayscaleWithPopulationDensity(x):
  if x == 30:
    return 0.5483527804562259
  elif x == 99:
    return 2.5802248275941975
  elif x == 146:
    return 4.0719211311237515
  elif x == 215:
    return 5.236094447939968
  else:
    return 0.1
```

[ ]:

```
vectorizePopulationDensityFunction = np.
 ↪vectorize(replaceGrayscaleWithPopulationDensity)
population = vectorizePopulationDensityFunction(population)
```

[ ]:

```
def replaceGrayscaleRoadsForCalc(x):
  if x == 255:
    return 0
  else:
    return 1
```

[ ]:

```
vectorizeRoadsFunction = np.vectorize(replaceGrayscaleRoadsForCalc)
roads = vectorizeRoadsFunction(roads)
```

[ ]:

```
def replaceWhitesForCalculations(x):
  if x == 255:
    return 0
  else:
    return 1
```

[ ]:

```
vectorizeWhitesFunction = np.vectorize(replaceWhitesForCalculations)
```

[ ]:

```
melusi2010 = cv2.imread('/content/drive/MyDrive/HonsArtifact/Pictures/
 ↪melusi2010gray01s_new.png')
melusi2020 = cv2.imread('/content/drive/MyDrive/HonsArtifact/Pictures/
 ↪melusi2020gray01s_new.png')
#population = cv2.imread('/content/drive/MyDrive/HonsArtifact/Pictures/
 ↪population_gray_01s_new.png') # only for sorting color out
population = cv2.imread('/content/drive/MyDrive/HonsArtifact/Pictures/
 ↪population2019gray01s_new.png')
```

```
roads = cv2.imread('/content/drive/MyDrive/HonsArtifact/Pictures/
 ↪roads_gray_01s_new.png')
```

[ ]:
```
melusi2010 = cv2.cvtColor(melusi2010, cv2.COLOR_BGR2GRAY)
melusi2020 = cv2.cvtColor(melusi2020, cv2.COLOR_BGR2GRAY)
population = cv2.cvtColor(population, cv2.COLOR_BGR2GRAY)
roads = cv2.cvtColor(roads, cv2.COLOR_BGR2GRAY)
```

[ ]:
```
def replaceColorsInPopulation(x):
  if x < 220 and x > 195:
    return 215 # Yellows in original picture
  elif x > 25 and x < 35:
    return 30 # Purples in original picture
  elif x > 80 and x < 115:
    return 99 # Blues in original picture
  elif x > 130 and x < 150:
    return 146 # Greens in original picture
  else:
    return 255 # To replace background colors
```

[ ]:
```
vectorizePopulationColorFunction = np.vectorize(replaceColorsInPopulation)
population = vectorizePopulationColorFunction(population)
```

[ ]:
```
def replaceGrayscaleRoadsForCalc(x):
  if x == 255:
    return 0
  else:
    return 1
```

[ ]:
```
vectorizeRoadsFunction = np.vectorize(replaceGrayscaleRoadsForCalc)
roads = vectorizeRoadsFunction(roads)
```

[ ]:
```
melusi2010_copy = melusi2010.copy()
melusi2020_copy = melusi2020.copy()
population_copy = population.copy()
roads_copy = roads.copy()
```

[ ]:
```
def replaceWhitesForCalculations(x):
  if x == 255:
    return 0
  else:
    return 1
```

[ ]:
```
vectorizeWhitesFunction = np.vectorize(replaceWhitesForCalculations)
```

[ ]:
```
row = 0
column = 0
```

```
step = 10
doubleStep = 20
maxHeight = 830
maxWidth = 2160

firstRow = 0
secondRow = 10
secondLastRow = 810
lastRow = 820


firstColumn = 0
secondColumn = 10
secondLastColumn = 2140
lastColumn = 2150

cumsumPopulationList = []
cumsumRoadList = []

for row in range(0,len(melusi2010_copy),10):
    for column in range(0,len(melusi2010_copy[row]),10):

      if(row < maxHeight and column < maxWidth):
        populationBlock = population_copy[row:row+step,column:column+step]
        roadsBlock = roads_copy[row:row+step,column:column+step]
        roadsCumulativeCalc = np.cumsum(roadsBlock)[-1]/100
        populationCumulativeCalc = np.cumsum(populationBlock)[-1]/100
        cumsumPopulationList.append(populationCumulativeCalc)
        cumsumRoadList.append(roadsCumulativeCalc)
```

[ ]:

```
npArrayPopulation = np.array(cumsumPopulationList)
binsPopulation = np.unique(npArrayPopulation)
graphPopulation = np.histogram(npArrayPopulation, bins=binsPopulation)
xPopulation = list(graphPopulation[1][:-1])
yPopulation = list(graphPopulation[0])
```

[ ]:

```
populationFigure = plt.figure(figsize=(6,6),dpi=120)
plt.plot(xPopulation,yPopulation)
populationFigure.suptitle("Cumulative sum of Population Density for 10x10␣
 ↪pixels")
plt.xlabel('Cumulative Sum')
plt.ylabel('Frequency')
populationFigure.savefig('populationFigure.jpg')
```

[ ]:

```
npArrayRoads = np.array(cumsumRoadList)
binsRoads = np.unique(npArrayRoads)
graphRoads = np.histogram(npArrayRoads, bins=binsRoads)
```

```
xRoads = list(graphRoads[1][:-1])
yRoads = list(graphRoads[0])
```

[ ]:

```
roadsFigure = plt.figure(figsize=(6,6),dpi=120)
plt.plot(xRoads,yRoads)
roadsFigure.suptitle("Cumulative sum of Road pressence for 10x10 pixels")
plt.xlabel('Cumulative Sum')
plt.ylabel('Frequency')
roadsFigure.savefig('roadsFigure.jpg')
```

[ ]:

```
row = 0
column = 0
step = 10
doubleStep = 20
maxHeight = 830
maxWidth = 2160

firstRow = 0
secondRow = 10
secondLastRow = 810
lastRow = 820


firstColumn = 0
secondColumn = 10
secondLastColumn = 2140
lastColumn = 2150

roadsConditional = 0
populationConditional = 0
topBottomConditional = 0
diagonalConditional = 0

tempMelusi = melusi2010.copy()

for generations in range(0,51):
  for row in range(0,len(melusi2010_copy),10):
    for column in range(0,len(melusi2010_copy[row]),10):
      #Blocks for cellular automata
      TopMiddle = None
      BottomMiddle = None
      MiddleRight = None
      MiddleLeft = None
      TopLeft = None
      TopRight = None
      BottomLeft = None
      BottomRight = None
    # Cumulative Sums for Blocks created above
    sumTopMiddle = 0
    sumBottomMiddle = 0
    sumMiddleRight = 0
    sumMiddleLeft = 0
    sumTopLeft = 0
    sumTopRight = 0
    sumBottomLeft = 0
    sumBottomRight = 0
    averageOfSumsTopsBottoms = 0
    averageOfSumsDiagonals = 0

    if(row < maxHeight and column < maxWidth):
      populationBlock = population_copy[row:row+step,column:column+step]
      roadsBlock = roads_copy[row:row+step,column:column+step]
      roadsCumulativeCalc = np.cumsum(roadsBlock)[-1]/100
      populationCumulativeCalc = np.cumsum(populationBlock)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= firstColumn and column <= lastColumn):
        TopMiddle = melusi2010_copy[row-step:row, column:column+step]
        TopMiddle = vectorizeWhitesFunction(TopMiddle)
        sumTopMiddle = np.cumsum(TopMiddle)[-1]/100

      if (row >= firstRow and row <= secondLastRow and column >= firstColumn and column <= lastColumn):
        BottomMiddle = melusi2010_copy[row:row+step, column:column+step]
        BottomMiddle = vectorizeWhitesFunction(BottomMiddle)
        sumBottomMiddle = np.cumsum(BottomMiddle)[-1]/100

      if (row >= firstRow and row <= lastRow and column >= firstColumn and column <= secondLastColumn):
        MiddleRight = melusi2010_copy[row:row+step, column+step:column+doubleStep]
        MiddleRight = vectorizeWhitesFunction(MiddleRight)
        sumMiddleRight = np.cumsum(MiddleRight)[-1]/100

      if (row >= firstRow and row <= lastRow and column >= secondColumn and column <= lastColumn):
        MiddleLeft = melusi2010_copy[row:row+step, column-step:column]
        MiddleLeft = vectorizeWhitesFunction(MiddleLeft)
        sumMiddleLeft = np.cumsum(MiddleLeft)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= secondColumn and column <= lastColumn):
        TopLeft = melusi2010_copy[row-step:row, column-step:column]
        TopLeft = vectorizeWhitesFunction(TopLeft)
        sumTopLeft = np.cumsum(TopLeft)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= firstColumn and␣
↪column <= secondLastColumn):
        TopRight = melusi2010_copy[row-step:row, column+step:column+doubleStep]
        TopRight = vectorizeWhitesFunction(TopRight)
        sumTopRight = np.cumsum(TopRight)[-1]/100

      if (row >= firstRow and row < secondLastRow and column >= secondColumn␣
↪and column <= lastColumn):
        BottomLeft = melusi2010_copy[row+step:row+doubleStep, column-step:
↪column]
        BottomLeft = vectorizeWhitesFunction(BottomLeft)
        sumBottomLeft = np.cumsum(BottomLeft)[-1]/100

      if (row >= firstRow and row <= secondLastRow and column >= firstColumn␣
↪and column <= secondLastColumn):
        BottomRight = melusi2010_copy[row+step:row+doubleStep, column+step:
↪column+doubleStep]
        BottomRight = vectorizeWhitesFunction(BottomRight)
        sumBottomRight = np.cumsum(BottomRight)[-1]/100

      countTopsBottoms = [TopMiddle, BottomMiddle, MiddleRight, MiddleLeft]
      countDiags = [TopLeft, TopRight, BottomLeft, BottomRight]

      noneCountsTops = len([x for x in countTopsBottoms if x is not None])
      noneCountsDiag = len([x for x in countDiags if x is not None])

      averageOfSumsTopsBottoms = (sumTopMiddle + sumBottomMiddle +␣
↪sumMiddleRight + sumMiddleLeft) / noneCountsTops
      averageOfSumsDiagonals = (sumTopLeft + sumTopRight + sumBottomLeft +␣
↪sumBottomRight) / noneCountsDiag

      if (generations == 0):
          roadsConditional = 0.01
          populationConditional = 0.08
          diagonalsConddidtional = 0.1
          topBottomConditional = 0.1
          if ((roadsCumulativeCalc >= roadsConditional or␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsConddidtional or averageOfSumsTopsBottoms > topBottomConditional)):
            for m in range(row,row+step):
              for n in range(column,column+step):
                tempMelusi[m,n] = 0
```

```
    elif (generations == 1):
      roadsConditional = 0.01
      populationConditional = 0.08
      diagonalsCondidtional = 0.1
      topBottomConditional = 0.1
      if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
        for m in range(row,row+step):
          for n in range(column,column+step):
            tempMelusi[m,n] = 0
    elif (generations < 31):
      roadsConditional = 0.1
      populationConditional = 0.1
      diagonalsCondidtional = 0.1
      topBottomConditional = 0.1
      if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
        for m in range(row,row+step):
          for n in range(column,column+step):
            tempMelusi[m,n] = 0
    else:
      roadsConditional = 0.05
      populationConditional = 0.05
      diagonalsCondidtional = 0.2
      topBottomConditional = 0.1
      if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
        for m in range(row,row+step):
          for n in range(column,column+step):
            tempMelusi[m,n] = 0

 melusi2010_copy = tempMelusi
 filename = f"pics/generation-{str(generations)}-melusi.png"
 cv2.imwrite(filename,melusi2010_copy)
```

[ ]:

```
row = 0
column = 0
step = 10
doubleStep = 20
maxHeight = 830
maxWidth = 2160

firstRow = 0
secondRow = 10
```

```
secondLastRow = 810
lastRow = 820


firstColumn = 0
secondColumn = 10
secondLastColumn = 2140
lastColumn = 2150

roadsConditional = 0
populationConditional = 0
topBottomConditional = 0
diagonalConditional = 0

tempMelusi = melusi2010.copy()

accuracyBenchmark = melusi2020_copy.flatten()
generationsList = []
accuracyList = []

for generations in range(1,51):
  for row in range(0,len(melusi2010_copy),10):
    for column in range(0,len(melusi2010_copy[row]),10):
      #Blocks for cellular automata
      TopMiddle = None
      BottomMiddle = None
      MiddleRight = None
      MiddleLeft = None
      TopLeft = None
      TopRight = None
      BottomLeft = None
      BottomRight = None

      # Cumulative Sums for Blocks created above
      sumTopMiddle = 0
      sumBottomMiddle = 0
      sumMiddleRight = 0
      sumMiddleLeft = 0
      sumTopLeft = 0
      sumTopRight = 0
      sumBottomLeft = 0
      sumBottomRight = 0
      averageOfSumsTopsBottoms = 0
      averageOfSumsDiagonals = 0

      if(row < maxHeight and column < maxWidth):
        populationBlock = population_copy[row:row+step,column:column+step]
      roadsBlock = roads_copy[row:row+step,column:column+step]
      roadsCumulativeCalc = np.cumsum(roadsBlock)[-1]/100
      populationCumulativeCalc = np.cumsum(populationBlock)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= firstColumn and␣
↪column <= lastColumn):
        TopMiddle = melusi2010_copy[row-step:row, column:column+step]
        TopMiddle = vectorizeWhitesFunction(TopMiddle)
        sumTopMiddle = np.cumsum(TopMiddle)[-1]/100

      if (row >= firstRow and row <= secondLastRow and column >= firstColumn␣
↪and column <= lastColumn):
        BottomMiddle = melusi2010_copy[row:row+step, column:column+step]
        BottomMiddle = vectorizeWhitesFunction(BottomMiddle)
        sumBottomMiddle = np.cumsum(BottomMiddle)[-1]/100

      if (row >= firstRow and row <= lastRow and column >= firstColumn and␣
↪column <= secondLastColumn):
        MiddleRight = melusi2010_copy[row:row+step, column+step:
↪column+doubleStep]
        MiddleRight = vectorizeWhitesFunction(MiddleRight)
        sumMiddleRight = np.cumsum(MiddleRight)[-1]/100

      if (row >= firstRow and row <= lastRow and column >= secondColumn and␣
↪column <= lastColumn):
        MiddleLeft = melusi2010_copy[row:row+step, column-step:column]
        MiddleLeft = vectorizeWhitesFunction(MiddleLeft)
        sumMiddleLeft = np.cumsum(MiddleLeft)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= secondColumn and␣
↪column <= lastColumn):
        TopLeft = melusi2010_copy[row-step:row, column-step:column]
        TopLeft = vectorizeWhitesFunction(TopLeft)
        sumTopLeft = np.cumsum(TopLeft)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= firstColumn and␣
↪column <= secondLastColumn):
        TopRight = melusi2010_copy[row-step:row, column+step:column+doubleStep]
        TopRight = vectorizeWhitesFunction(TopRight)
        sumTopRight = np.cumsum(TopRight)[-1]/100

      if (row >= firstRow and row < secondLastRow and column >= secondColumn␣
↪and column <= lastColumn):
        BottomLeft = melusi2010_copy[row+step:row+doubleStep, column-step:
↪column]
        BottomLeft = vectorizeWhitesFunction(BottomLeft)
        sumBottomLeft = np.cumsum(BottomLeft)[-1]/100

      if (row >= firstRow and row <= secondLastRow and column >= firstColumn␣
↪and column <= secondLastColumn):
        BottomRight = melusi2010_copy[row+step:row+doubleStep, column+step:
↪column+doubleStep]
        BottomRight = vectorizeWhitesFunction(BottomRight)
        sumBottomRight = np.cumsum(BottomRight)[-1]/100

      countTopsBottoms = [TopMiddle, BottomMiddle, MiddleRight, MiddleLeft]
      countDiags = [TopLeft, TopRight, BottomLeft, BottomRight]

      noneCountsTops = len([x for x in countTopsBottoms if x is not None])
      noneCountsDiag = len([x for x in countDiags if x is not None])

      averageOfSumsTopsBottoms = (sumTopMiddle + sumBottomMiddle +␣
↪sumMiddleRight + sumMiddleLeft) / noneCountsTops
      averageOfSumsDiagonals = (sumTopLeft + sumTopRight + sumBottomLeft +␣
↪sumBottomRight) / noneCountsDiag

      if (generations == 0):
          roadsConditional = 0.01
          populationConditional = 0.08
          diagonalsCondidtional = 0.1
          topBottomConditional = 0.1
          if ((roadsCumulativeCalc >= roadsConditional or␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
            for m in range(row,row+step):
              for n in range(column,column+step):
                tempMelusi[m,n] = 0
      elif (generations == 1):
          roadsConditional = 0.01
          populationConditional = 0.08
          diagonalsCondidtional = 0.1
          topBottomConditional = 0.1
          if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
            for m in range(row,row+step):
              for n in range(column,column+step):
                tempMelusi[m,n] = 0
      elif (generations < 31):
          roadsConditional = 0.1
          populationConditional = 0.1
          diagonalsCondidtional = 0.1
```

```
            topBottomConditional = 0.1
            if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
              for m in range(row,row+step):
                for n in range(column,column+step):
                  tempMelusi[m,n] = 0
        else:
            roadsConditional = 0.05
            populationConditional = 0.05
            diagonalsCondidtional = 0.2
            topBottomConditional = 0.1
            if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
              for m in range(row,row+step):
                for n in range(column,column+step):
                  tempMelusi[m,n] = 0

  melusi2010_copy = tempMelusi
  accuracyTemp = melusi2010_copy.copy().flatten()
  score = cohen_kappa_score(accuracyBenchmark,accuracyTemp)
  accuracyList.append(score)
  generationsList.append(generations)
  #filename = f"pics/generation-{str(generations)}-melusi.png"
  #cv2.imwrite(filename,melusi2010_copy)
```

[ ]:

```
scoresFigure = plt.figure(figsize=(5,5),dpi=120)
plt.plot(generationsList,accuracyList)
scoresFigure.suptitle("Cohen's kappa coefficient after each Generation")
plt.xlabel('Generation')
plt.ylabel('Kappa Score')
scoresFigure.savefig('scoresFigure.jpg')
```

[ ]:

```
# Last Test

row = 0
column = 0
step = 10
doubleStep = 20
maxHeight = 830
maxWidth = 2160

firstRow = 0
secondRow = 10
secondLastRow = 810
lastRow = 820
```

```
firstColumn = 0
secondColumn = 10
secondLastColumn = 2140
lastColumn = 2150

roadsConditional = 0
populationConditional = 0
topBottomConditional = 0
diagonalConditional = 0

tempMelusi = melusi2010.copy()

accuracyBenchmark = melusi2020_copy.flatten()
generationsList = []
accuracyList = []

for generations in range(0,51):
  for row in range(0,len(melusi2010_copy),10):
    for column in range(0,len(melusi2010_copy[row]),10):
      #Blocks for cellular automata
      TopMiddle = None
      BottomMiddle = None
      MiddleRight = None
      MiddleLeft = None
      TopLeft = None
      TopRight = None
      BottomLeft = None
      BottomRight = None

      # Cumulative Sums for Blocks created above
      sumTopMiddle = 0
      sumBottomMiddle = 0
      sumMiddleRight = 0
      sumMiddleLeft = 0
      sumTopLeft = 0
      sumTopRight = 0
      sumBottomLeft = 0
      sumBottomRight = 0
      averageOfSumsTopsBottoms = 0
      averageOfSumsDiagonals = 0

      if(row < maxHeight and column < maxWidth):
        populationBlock = population_copy[row:row+step,column:column+step]
        roadsBlock = roads_copy[row:row+step,column:column+step]
        roadsCumulativeCalc = np.cumsum(roadsBlock)[-1]/100
      populationCumulativeCalc = np.cumsum(populationBlock)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= firstColumn and␣
↪column <= lastColumn):
        TopMiddle = melusi2010_copy[row-step:row, column:column+step]
        TopMiddle = vectorizeWhitesFunction(TopMiddle)
        sumTopMiddle = np.cumsum(TopMiddle)[-1]/100

      if (row >= firstRow and row <= secondLastRow and column >= firstColumn␣
↪and column <= lastColumn):
        BottomMiddle = melusi2010_copy[row:row+step, column:column+step]
        BottomMiddle = vectorizeWhitesFunction(BottomMiddle)
        sumBottomMiddle = np.cumsum(BottomMiddle)[-1]/100

      if (row >= firstRow and row <= lastRow and column >= firstColumn and␣
↪column <= secondLastColumn):
        MiddleRight = melusi2010_copy[row:row+step, column+step:
↪column+doubleStep]
        MiddleRight = vectorizeWhitesFunction(MiddleRight)
        sumMiddleRight = np.cumsum(MiddleRight)[-1]/100

      if (row >= firstRow and row <= lastRow and column >= secondColumn and␣
↪column <= lastColumn):
        MiddleLeft = melusi2010_copy[row:row+step, column-step:column]
        MiddleLeft = vectorizeWhitesFunction(MiddleLeft)
        sumMiddleLeft = np.cumsum(MiddleLeft)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= secondColumn and␣
↪column <= lastColumn):
        TopLeft = melusi2010_copy[row-step:row, column-step:column]
        TopLeft = vectorizeWhitesFunction(TopLeft)
        sumTopLeft = np.cumsum(TopLeft)[-1]/100

      if (row >= secondRow and row <= lastRow and column >= firstColumn and␣
↪column <= secondLastColumn):
        TopRight = melusi2010_copy[row-step:row, column+step:column+doubleStep]
        TopRight = vectorizeWhitesFunction(TopRight)
        sumTopRight = np.cumsum(TopRight)[-1]/100

      if (row >= firstRow and row < secondLastRow and column >= secondColumn␣
↪and column <= lastColumn):
        BottomLeft = melusi2010_copy[row+step:row+doubleStep, column-step:
↪column]
        BottomLeft = vectorizeWhitesFunction(BottomLeft)
        sumBottomLeft = np.cumsum(BottomLeft)[-1]/100
        if (row >= firstRow and row <= secondLastRow and column >= firstColumn␣
↪and column <= secondLastColumn):
          BottomRight = melusi2010_copy[row+step:row+doubleStep, column+step:
↪column+doubleStep]
          BottomRight = vectorizeWhitesFunction(BottomRight)
          sumBottomRight = np.cumsum(BottomRight)[-1]/100

        countTopsBottoms = [TopMiddle, BottomMiddle, MiddleRight, MiddleLeft]
        countDiags = [TopLeft, TopRight, BottomLeft, BottomRight]

        noneCountsTops = len([x for x in countTopsBottoms if x is not None])
        noneCountsDiag = len([x for x in countDiags if x is not None])

        averageOfSumsTopsBottoms = (sumTopMiddle + sumBottomMiddle +␣
↪sumMiddleRight + sumMiddleLeft) / noneCountsTops
        averageOfSumsDiagonals = (sumTopLeft + sumTopRight + sumBottomLeft +␣
↪sumBottomRight) / noneCountsDiag

        if (generations == 0):
            roadsConditional = 0.01
            populationConditional = 0.08
            diagonalsCondidtional = 0.1
            topBottomConditional = 0.1
            if ((roadsCumulativeCalc >= roadsConditional or␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
              for m in range(row,row+step):
                for n in range(column,column+step):
                  tempMelusi[m,n] = 0
        elif (generations == 1):
            roadsConditional = 0.01
            populationConditional = 0.08
            diagonalsCondidtional = 0.1
            topBottomConditional = 0.1
            if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
              for m in range(row,row+step):
                for n in range(column,column+step):
                  tempMelusi[m,n] = 0
        elif (generations < 40):
            roadsConditional = 0.1
            populationConditional = 0.1
            diagonalsCondidtional = 0.1
            topBottomConditional = 0.1
          if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
            for m in range(row,row+step):
              for n in range(column,column+step):
                tempMelusi[m,n] = 0
      else:
          roadsConditional = 0.09
          populationConditional = 0.05
          diagonalsCondidtional = 0.15
          topBottomConditional = 0.1
          if ((roadsCumulativeCalc >= roadsConditional and␣
↪populationCumulativeCalc >= populationConditional) and (averageOfSumsDiagonals␣
↪> diagonalsCondidtional or averageOfSumsTopsBottoms > topBottomConditional)):
            for m in range(row,row+step):
              for n in range(column,column+step):
                tempMelusi[m,n] = 0

melusi2010_copy = tempMelusi
accuracyTemp = melusi2010_copy.copy().flatten()
score = cohen_kappa_score(accuracyBenchmark,accuracyTemp)
accuracyList.append(score)
generationsList.append(generations)
```

```
[ ]: scoresFigure = plt.figure(figsize=(5,5),dpi=120)
     plt.plot(generationsList,accuracyList)
     scoresFigure.suptitle("Cohen's kappa coefficient after each Generation")
     plt.xlabel('Generation')
     plt.ylabel('Kappa Score')
     scoresFigure.savefig('scoresFigure.jpg')
```

```
[ ]: max(accuracyList)
```