Segmentación basada en grafos

Universidad de Chile
Departamento de Ciencias de la Computación
CC5508 - Procesamiento y Análisis de Imágenes
Gabriel De La Parra

Contenidos

In [1]:
```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.js')
```

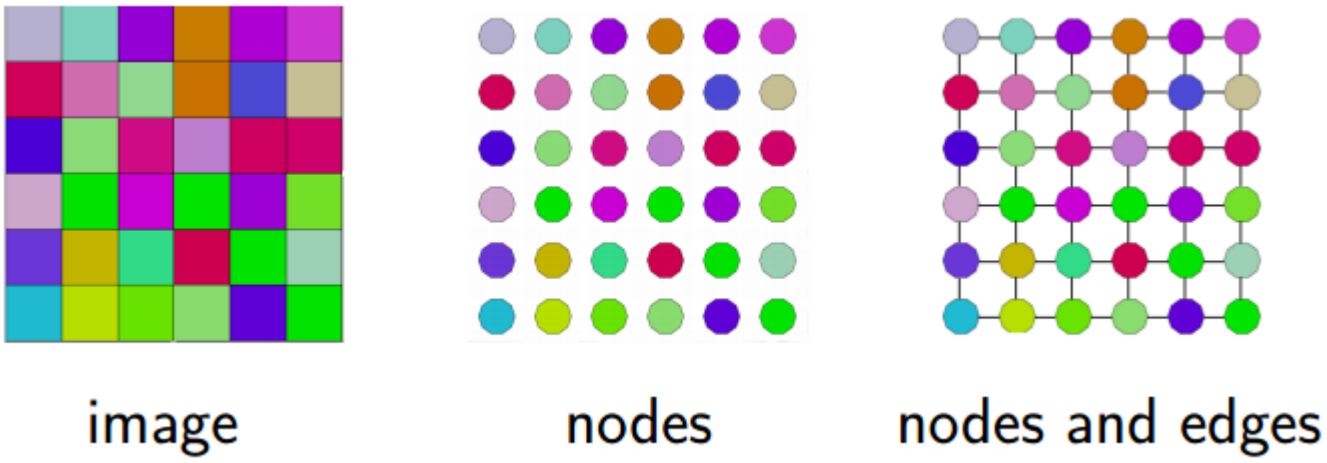
Introducción

Esta tarea se basa en la implementación del paper *Efficient Graph-Based Image Segmentation (https://cs.brown.edu/~pff/papers/seg-ijcv.pdf)* de *Pedro F. Felzenzswalb (https://cs.brown.edu/~pff/)*.

Imágen como grafo

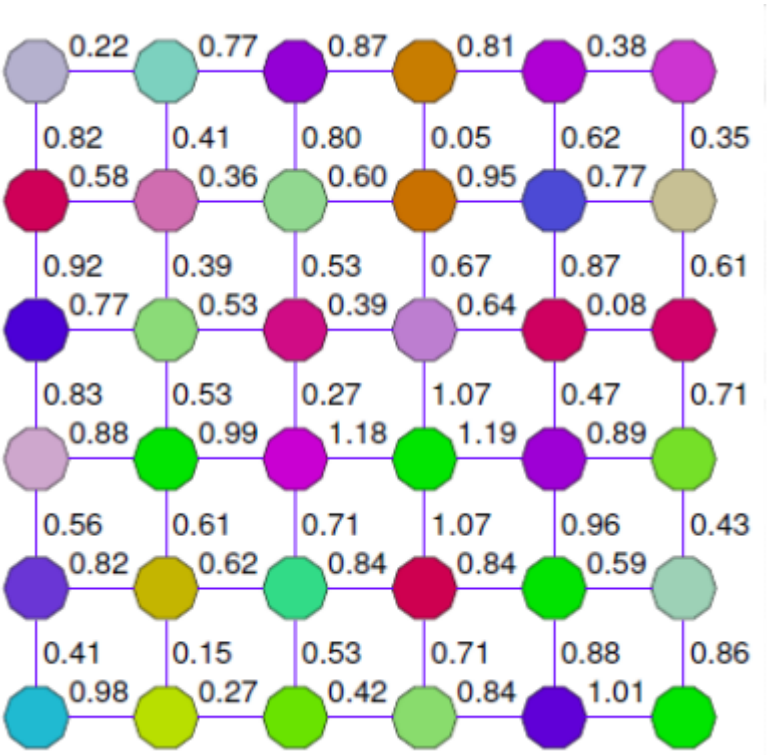
En la investigación se expone una técnica de *segmentación de imagenes (https://en.wikipedia.org/wiki/Image_segmentation)* utilizando un *grafo no dirigido (https://es.wikipedia.org/wiki/Grafo#Grafo_no_dirigido)* para representar la imagen de entrada:

- Los nodos del grafo representan los pixeles de la imagen.
- Los arcos del grafo representan la relación entre pixeles vecinos.
- A cada arco se le asocia un peso. De lo anterior, cada peso determina la relación entre pixeles vecinos.



(ImageGraph1.PNG)

Representación de los pixeles como nodos y arcos. [9]



(ImageGraph2.PNG)

Representación de los pesos de los arcos. [9]

Segmentación

La segmentación consiste en generar conjuntos de nodos (o componentes conexas).

Para lo anterior se definen dos métricas:

- *Diferencia interna de una componente conexa*.
- *Diferencia entre dos componentes conexas*.

Lo anterior pretende agrupar pixeles en regiones que sean similares entre sí y diferentes a otras regiones.

Proceso de la segmentación

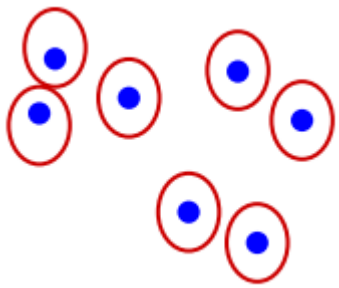
Para la segmentación se procede de la siguiente manera:

- Con las métricas anteriores, se itera sobre las fronteras de cada componente.
- Sobre dicha frontera se busca evaluar si (los pesos de) la *diferencia entre las componentes* es suficientemente mayor a (los pesos de) la *diferencia interna* (de cada componente).
- A partir de esta evaluación se puede generar una única región o mantener las dos regiones.

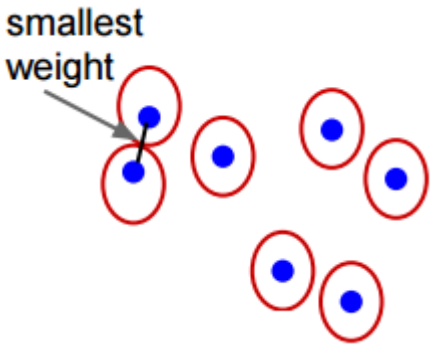
Para mejorar este proceso se define un umbral sobre la evaluación de las diferencias.

Lo anterior se realiza para todos los arcos del grafo.

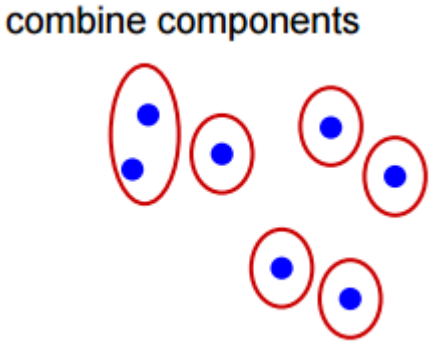
A modo ilustrativo [3]:



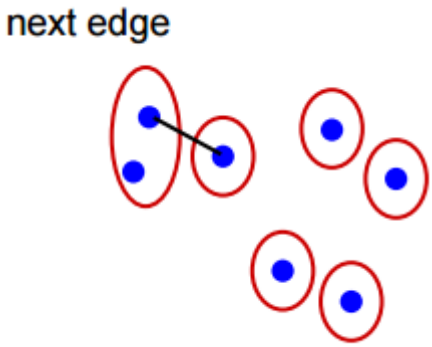
(segment0.PNG) Con los arcos ordenados, crear componentes por si solas.



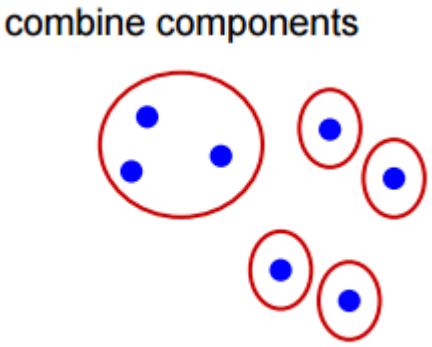
(segment1.PNG) Para cada peso, procesar el peso entre los nodos y el peso de la componente.



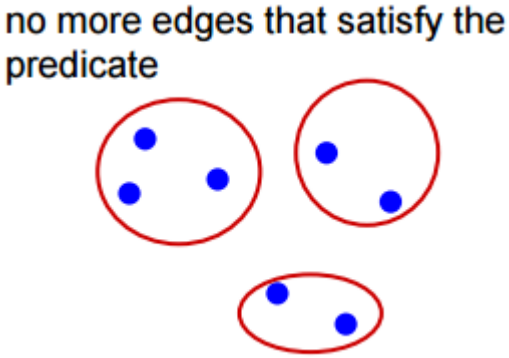
(segment2.PNG) Si el peso del arco es menor que el interno de la componente, unir la componente.



(segment3.PNG) Iterar para el siguiente arco.



(segment4.PNG) Y combinar las componentes.



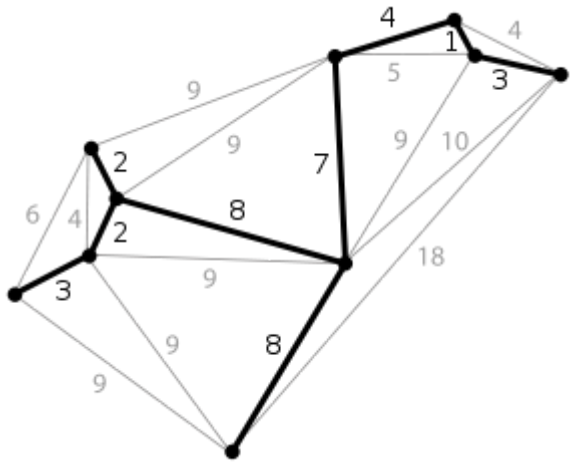
(segment5.PNG) Seguir hasta procesar todos los arcos.

Métricas para la segmentación

La *diferencia entre dos componentes* se define como el peso mínimo a lo largo de (los arcos de) la frontera entre los dos grupos de nodos.

Por su parte, *la diferencia interna*, se define como el peso máximo en el MST de la componente.

Un MST (*minumum spanning tree (https://en.wikipedia.org/wiki/Minimum_spanning_tree)* o *árbol cobertor mínimo (https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo)*) es una estructura de datos que permite ordenar todos los nodos de un grafo.[6]



(https://upload.wikimedia.org/wikipedia/commons/thumb/d/d2/Minimum_spanning_tree.svg/300px-Minimum_spanning_tree.svg.png)

> Un ejemplo de árbol recubridor mínimo. Cada punto representa un vértice, cada arista está etiquetada con su peso, que en este caso equivale a su longitud. - Wikipedia

Una forma de construir un MST (Hint entregado en el enunciado) es por el método de Kruskal (https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal). Para esto se requiere ocupar un algoritmo eficiente para determinar:

1. Los componentes que conectan una arista (Find).
2. Mezclar o unir dos componentes (Union).

Union-Find

Union-Find (https://en.wikipedia.org/wiki/Disjoint-set_data_structure) es un algoritmo para estructuras de datos disjuntas. *Una estructura de datos para conjuntos disjuntos, es una estructura de datos que mantiene un conjunto de elementos particionados en un número de conjuntos disjuntos (no se solapan los conjuntos).*

Union: Une dos subconjuntos en uno solo.

Find: Determina a cual subconjunto pertenece un elemento. Esta operación puede usarse para verificar si dos elementos están en el mismo conjunto.

La complejidad del algoritmo (https://en.wikipedia.org/wiki/Disjoint-set_data_structure#Time_Complexity) se diferencia por 4 formas de implementación:

- *Union* con *Union-by-rank*
- *Find* con *path compression*
- Ambas
- Ninguna

Union-by-rank, consiste en añadir el árbol más pequeño a la raíz del árbol más grande. Como la profundidad del árbol afecta el tiempo de ejecución del algoritmo, el árbol con menor profundidad es añadido a la raíz del árbol con mayor profundidad, el cual aumenta su profundidad solo si sus profundidades son iguales.

Find con *path compression*, es una forma recursiva de aplanar la estructura del árbol. Cada nodo visitado en el camino hacia la raíz se añade directamente a la raíz ya que todos estos nodos la comparten. El árbol resultante es más aplanado, acelerando operaciones futuras no solo en estos elementos sino también en aquellos que referencian a estos.

Indicaciones de la tarea

Para esta tarea se solicita implementar el algoritmo de segmentación de Felzenswalb para imagenes a colores (3 canales). Una restricción dada, es que no se permite convertir la imagen a escala de grises. Adicionalmente se solicita implementar esto para distintos espacios de color: RGB y CIE L*a*b. Finalmente se solicita ejecutar esto para 10 imágenes distintas y comparar los resultados.

Dado lo anterior, en el Paper se indica como trabajar en este tipo de casos:

- Construir el grafo con conectividad 8.
- Utilizar una función de pesos basada en la intensidad absoluta entre pixeles:

$$w(v_i, v_j) = |I(p_i) - I(p_j)|$$

donde:

$w(v_i, v_j)$ es la función de pesos sobre los nodos v_i y v_j

$I(p_i)$ y $I(p_j)$ es la intensidad de los pixeles p_i y p_j

Adicionalmente:

- Se aplica un filtro Gaussiano, con σ recomendado de 0.8, que no produce cambios perceptibles en la imagen.
- Para imágenes en color, se ejecuta el algoritmo 3 veces, una para cada componente del espacio de color y luego se intersectan los 3 conjuntos. Para esto se colocan 2 pixeles vecinos en la misma componente si aparecen en las 3 componentes de color. Alternativamente, se propone ejecutar el algoritmo una sola vez, tomando los pesos como la distancia entre los pixeles en el espacio de color.

La propuesta alternativa del autor es mucho más sencilla para trabajar en distintos espacios de color. Se trabajará con este enfoque.

Desarrollo

A continuación se entrega el código utilizado en el desarrollo de esta tarea.

Librerías de Python

In [2]:
```
%matplotlib inline

import os
from scipy import ndimage as ndi
from skimage import io, filters, exposure, color, img_as_float
import matplotlib.pylab as plt
import matplotlib
import numpy as np
from ipywidgets import interact, widgets, HBox
from IPython import display
import math
from random import randint

#Ajustar el tamaño de las imágenes:
matplotlib.rcParams['figure.figsize'] = (14,12)
```

Python Widgets

In [3]:
```
imageOptions = [f for f in os.listdir(os.getcwd()) if f.endswith('_05.jpg')]

selectImageWidget = widgets.ToggleButtons(
    options = imageOptions,
    description = 'Image:',
    button_style = 'info',
)

colorSpaceOptions = ['RGB', 'CIE LAB', 'CIE LUV', 'HSV', 'RGB CIE', 'XYZ']

colorSpaceWidget = widgets.ToggleButtons(
    options = colorSpaceOptions,
    description = 'Color Space:',
    button_style = 'success',
)

sigmaSliderWidget = widgets.FloatSlider(
    value=0.8,
    min=0,
    max=3,
    step=0.1,
    description='Sigma:',
    continuous_update=False,
    readout=True,
    readout_format='.1f',
)

kSliderWidget = widgets.FloatSlider(
    value=0.8,
    min=0,
    max=1,
    step=0.1,
    description='k:',
    continuous_update=False,
    readout=True,
    readout_format='.1f',
)

minSizeSliderWidget = widgets.IntSlider(
    value=350,
    min=1,
    max=500,
    step=1,
    description='Min Size:',
    continuous_update=False,
    readout=True,
)
```

Pixel Node

In [4]:
```
class PixelNode:
    def __init__(self, image, col, row):
        self.col = col
        self.row = row
        self.r = float(image[row][col][0])
        self.g = float(image[row][col][1])
        self.b = float(image[row][col][2])
        self.cols = image.shape[1]
        self.index = col + (row * self.cols)

    def __str__(self):
        return "index: " + str(self.col) + "+(" + str(self.row)+"x" + str(self.cols)
+ "):" + str(self.index)
```

Pixel Edge

In [5]:
```
class PixelEdge:
    def __init__(self, pixelNode1, pixelNode2):
        self.node1 = pixelNode1
        self.node2 = pixelNode2
        self.weight = self.pixelDistance(pixelNode1, pixelNode2)

    @classmethod
    def pixelDistance(self, pixelNode1, pixelNode2):
        return math.sqrt(math.pow(pixelNode1.r - pixelNode2.r, 2)
                         + math.pow(pixelNode1.g - pixelNode2.g, 2)
                         + math.pow(pixelNode1.b - pixelNode2.b, 2))

    def __str__(self):
        return "Node1: " + str(self.node1) + " Node2: " + str(self.node2) + " Weight:
 " + str(self.weight)
```

Pixel Graph

In [6]:
```
class PixelGraph:
    def __init__(self, image):
        self.nodes = []
        self.edges = []
        self.nodes, self.edges = self.getGraph(image)

    @classmethod
    def getGraph(self, image):
        nodes = []
        edges = []
        rows, cols, _ = image.shape

        for row in range(rows):
            for col in range(cols):
                node = PixelNode(image, col, row)
                nodes.append(node)
                node2 = None

                if col + 1 < cols:
                    # Vecino derecha:
                    node2 = PixelNode(image, col + 1, row)
                    edges.append(PixelEdge(node, node2))

                if row + 1 < rows:
                    # Vecino abajo:
                    node2 = PixelNode(image, col, row + 1)
                    edges.append(PixelEdge(node, node2))

                if col + 1 < cols and row + 1 < rows:
                    # Vecino Diagonal abajo derecha:
                    node2 = PixelNode(image, col + 1, row + 1)
                    edges.append(PixelEdge(node, node2))

                if col - 1 >= 0 and row + 1 < rows:
                    # Vecino Diagonal abajo izquierda
                    node2 = PixelNode(image, col - 1, row + 1)
                    edges.append(PixelEdge(node, node2))

        return nodes, edges
```

Disjoint Set Component

In [7]:
```
class SetComponent:
    def __init__(self, index, k):
        self.parent = index
        self.rank = 0
        self.size = 1
        self.edges = []
        self.weight = k / self.size
```

Disjoint Set

With Threshold, Minimum Size and Paint Methods

In [8]:
```
class DisjointSet:
    def __init__(self, count, imageShape, thresholdConstant, minSize):
        self.components = [SetComponent(i, thresholdConstant) for i in range(count)]
        self.thresholdConstant = thresholdConstant
        self.imageShape = imageShape
        self.minSize = minSize
```

```
    def Find(self, node):
        if(self.components[node].parent != node):
            self.components[node].parent = self.Find(self.components[node].parent)
        return self.components[node].parent

    def WeightUnion(self, edge):
        nodeRoot1 = self.Find(edge.node1.index)
        nodeRoot2 = self.Find(edge.node2.index)
        edgeWeight = edge.weight

        if edgeWeight <= self.components[nodeRoot1].weight and edgeWeight <= self.com
ponents[nodeRoot2].weight:
            self.Union(nodeRoot1, nodeRoot2)
            nodeRoot1 = self.Find(nodeRoot1)
            self.components[nodeRoot1].weight = edgeWeight + (self.thresholdConstant
/ self.components[nodeRoot1].size)

    def MinSizeUnion(self, edge):
        nodeRoot1 = self.Find(edge.node1.index)
        nodeRoot2 = self.Find(edge.node2.index)
        minSize = self.minSize

        if self.components[nodeRoot1].size < minSize or self.components[nodeRoot2].si
ze < minSize:
            self.Union(nodeRoot1, nodeRoot2)

    def Union(self, nodeRoot1, nodeRoot2):

        if(nodeRoot1 == nodeRoot2):
            return

        if self.components[nodeRoot1].rank > self.components[nodeRoot2].rank:
            self.components[nodeRoot2].parent = nodeRoot1
            self.components[nodeRoot1].size += self.components[nodeRoot2].size

        elif self.components[nodeRoot1].rank < self.components[nodeRoot2].rank:
            self.components[nodeRoot1].parent = nodeRoot2
            self.components[nodeRoot2].size += self.components[nodeRoot1].size

        else:
            self.components[nodeRoot1].parent == nodeRoot2
            self.components[nodeRoot2].size += self.components[nodeRoot1].size
            self.components[nodeRoot2].rank += 1

    def Paint(self):
        rows = self.imageShape[0]
        cols = self.imageShape[1]
        chns = self.imageShape[2]

        paintedSet = np.zeros((rows, cols, 3))
        randomPalette = np.array([[randint(0, 255) for c in range(rows*cols)] for p i
n range(chns)])

        for row in range(rows):
            for col in range(cols):
                component = self.Find(col + (row * cols))
                for channel in range(chns):
                    paintedSet[row][col][channel] = randomPalette[channel][component]

        return paintedSet

    def printThis(self):
        for i in range(len(self.components)):
```

```
print("i:",i,"Parent:",self.components[i].parent,"Rank:",self.components[i].rank,"Siz
e:",self.components[i].size,"Threshold:",self.components[i].threshold)
```

Main

In [9]:
```
def segmentImage(imageFile, colorSpace, sigma, k, minimumSize):

    image = exposure.rescale_intensity(img_as_float(io.imread(imageFile)))

    if colorSpace=="CIE LAB":
        image = exposure.rescale_intensity(color.rgb2lab(image))
    elif colorSpace=="CIE LUV":
        image = exposure.rescale_intensity(color.rgb2luv(image))
    elif colorSpace=="HSV":
        image = exposure.rescale_intensity(color.rgb2hsv(image))
    elif colorSpace=="RGB CIE":
        image = exposure.rescale_intensity(color.rgb2rgbcie(image))
    elif colorSpace=="XYZ":
        image = exposure.rescale_intensity(color.rgb2xyz(image))

    image = filters.gaussian(image, sigma, multichannel=True)

    graph = PixelGraph(image)

    disjointSet = DisjointSet(len(graph.nodes), image.shape, k, minimumSize)

    edges = sorted(graph.edges, key=lambda e: e.weight)

    for i in range(len(edges)):
        disjointSet.WeightUnion(edges[i])

    for i in range(len(edges)):
        disjointSet.MinSizeUnion(edges[i])

    fig = plt.figure()
    f1 = fig.add_subplot(121)
    f2 = fig.add_subplot(122)

    f1.imshow(image, cmap="gray")
    f2.imshow(disjointSet.Paint(), cmap="gray")
    plt.show()

interact(segmentImage, imageFile = selectImageWidget, colorSpace = colorSpaceWidget,
sigma = sigmaSliderWidget, k= kSliderWidget, minimumSize=minSizeSliderWidget)

print("Interactive Graph Segmentation")
```

```
Interactive Graph Segmentation
```

Resultados

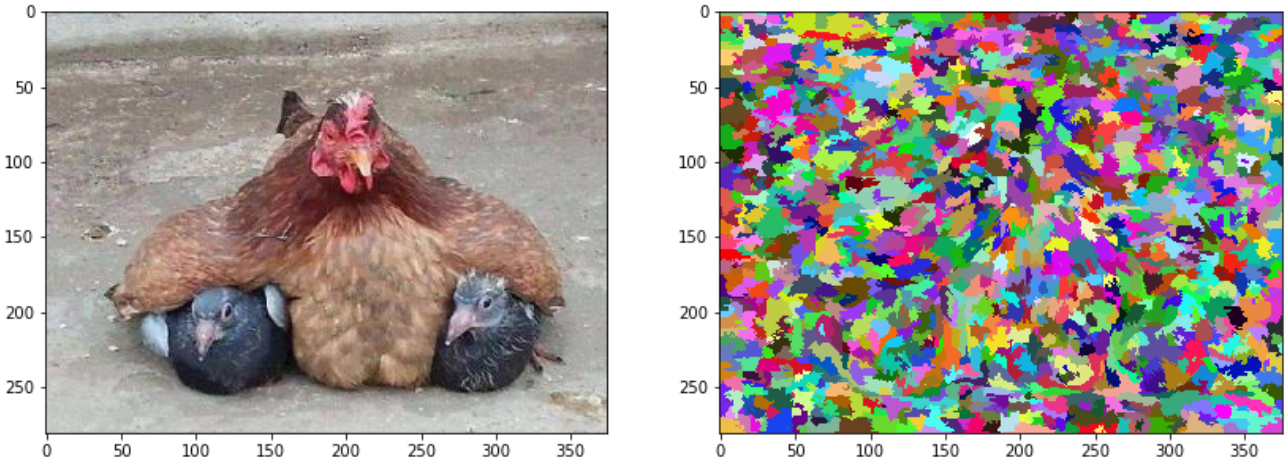
A continuación se presentan diversos resultados generados con distintas configuraciones.

Con una primera imagen, se iteró sobre varias configuraciones de Sigma, K y MinimumSize hasta obtener un resultado aceptable. Con dicho resultado se iteró sobre las distintas imágenes.

Variaciones de k

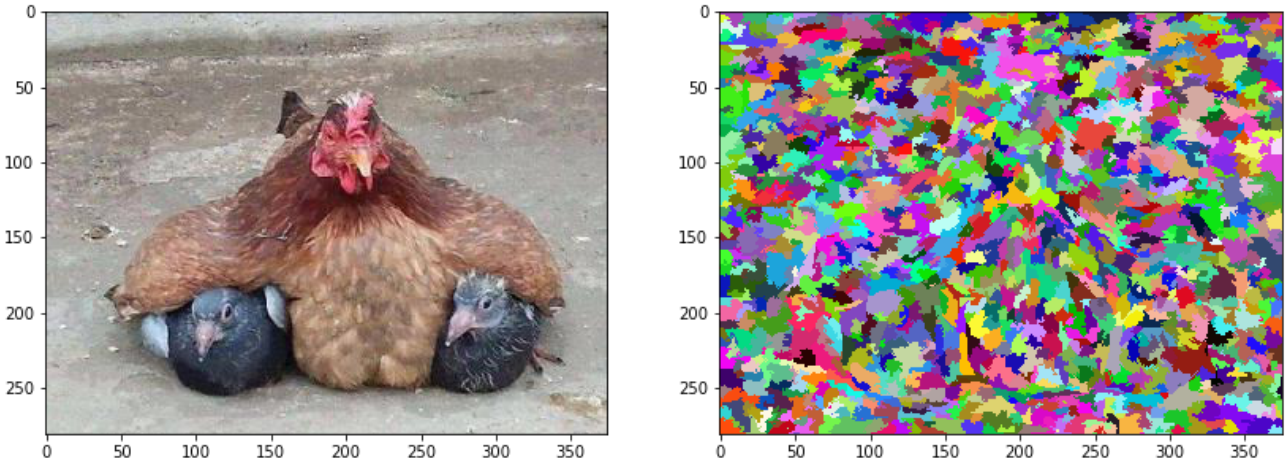
Se probará con Sigma=0.1 (muy bajo) y MinSize = 100 (Muy bajo)

In [10]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



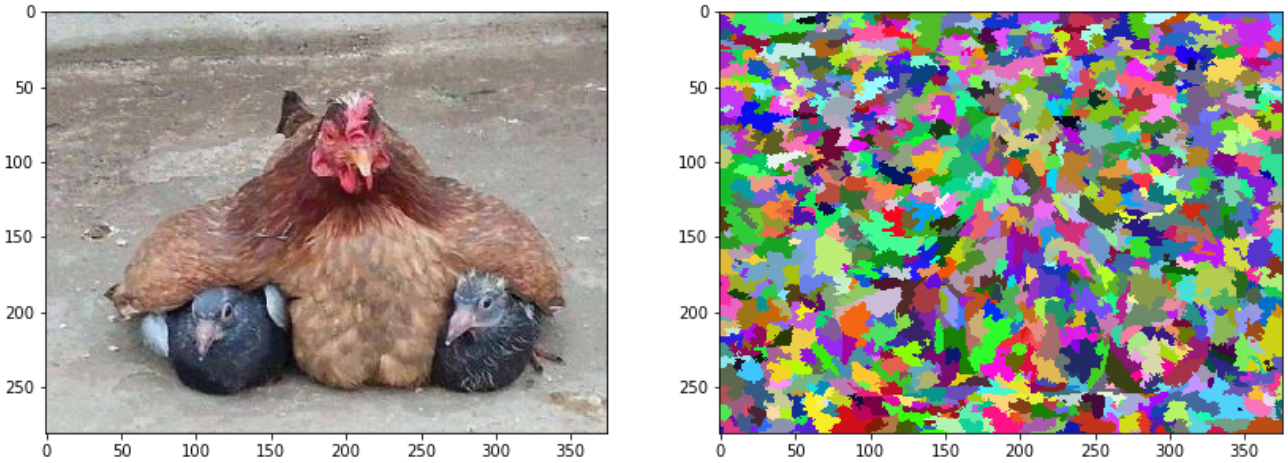
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 1
00
```

In [11]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.3;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



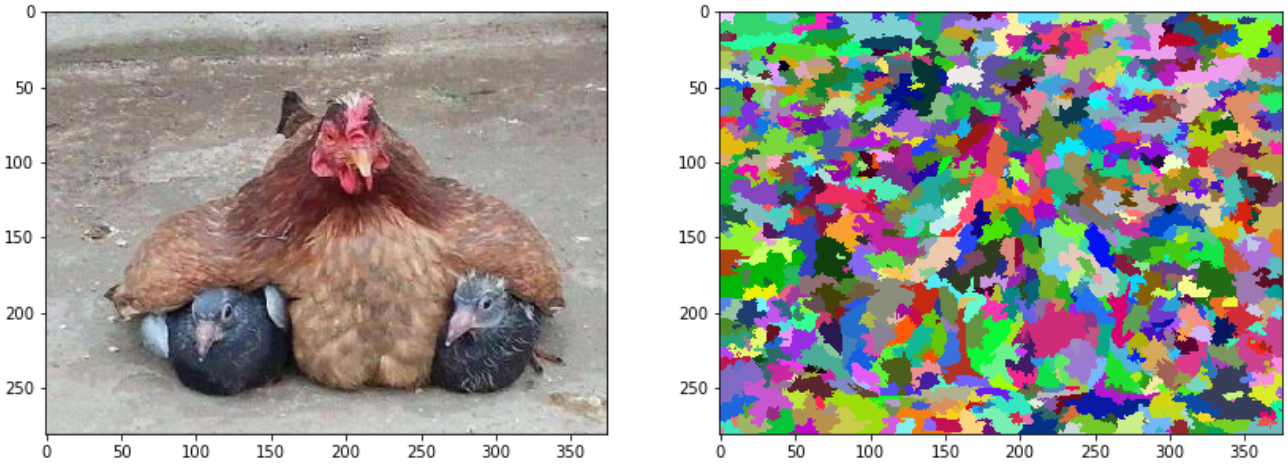
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.3 ; Minimum Size: 1
00
```

In [12]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.5;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



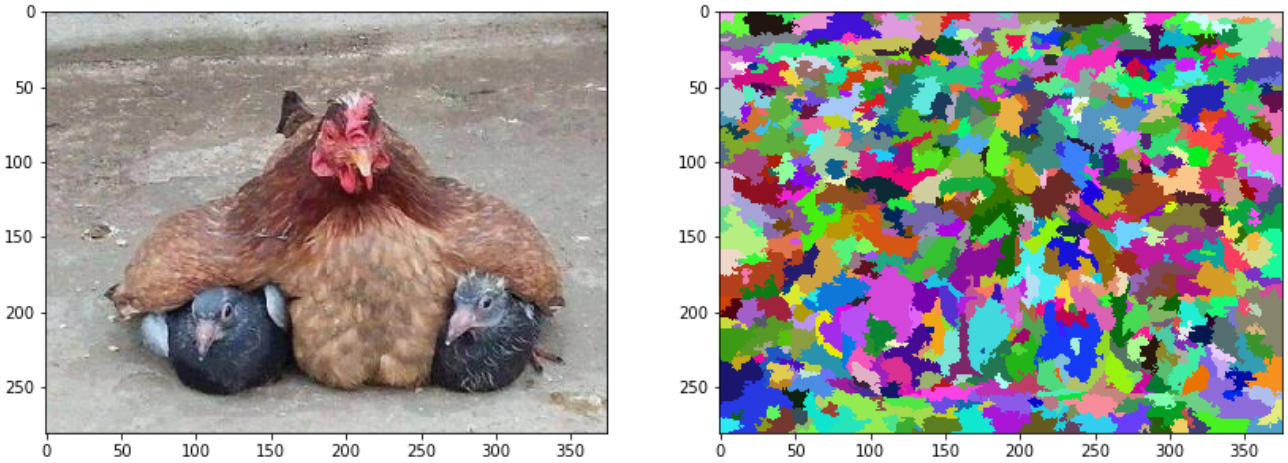
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.5 ; Minimum Size: 1
00
```

In [13]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.7;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.7 ; Minimum Size: 1
00
```

In [14]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.9;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

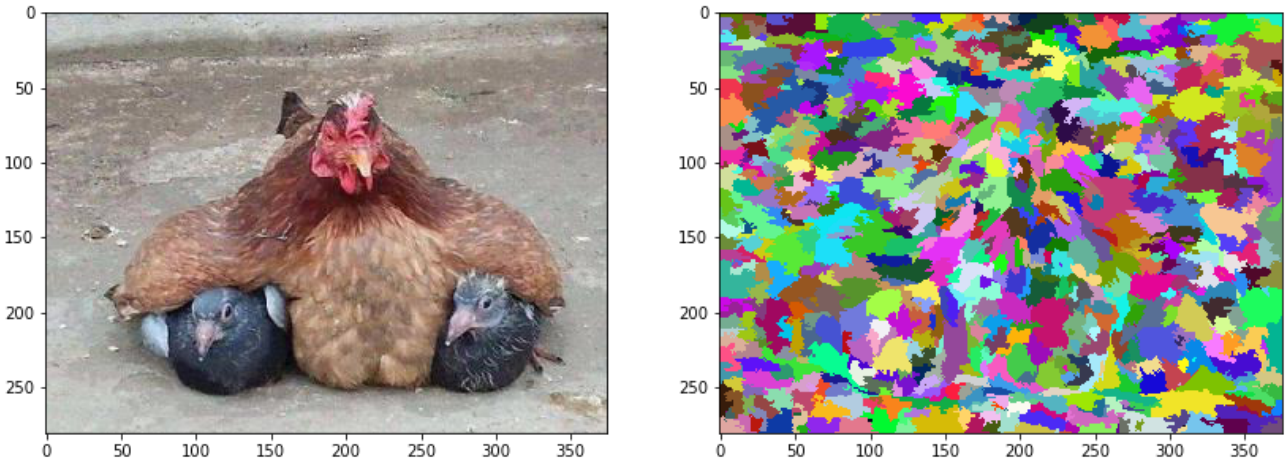


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.9 ; Minimum Size: 1
00
```

Variaciones de MinSize

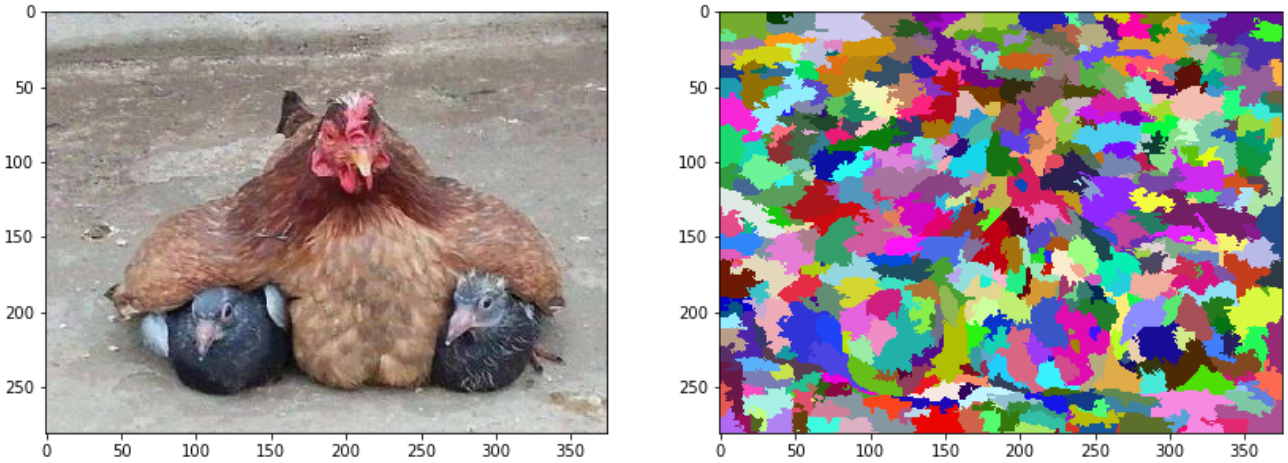
Se probará con Sigma=0.1 (muy bajo) y k = 0.1 (Muy bajo)

In [15]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 200
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



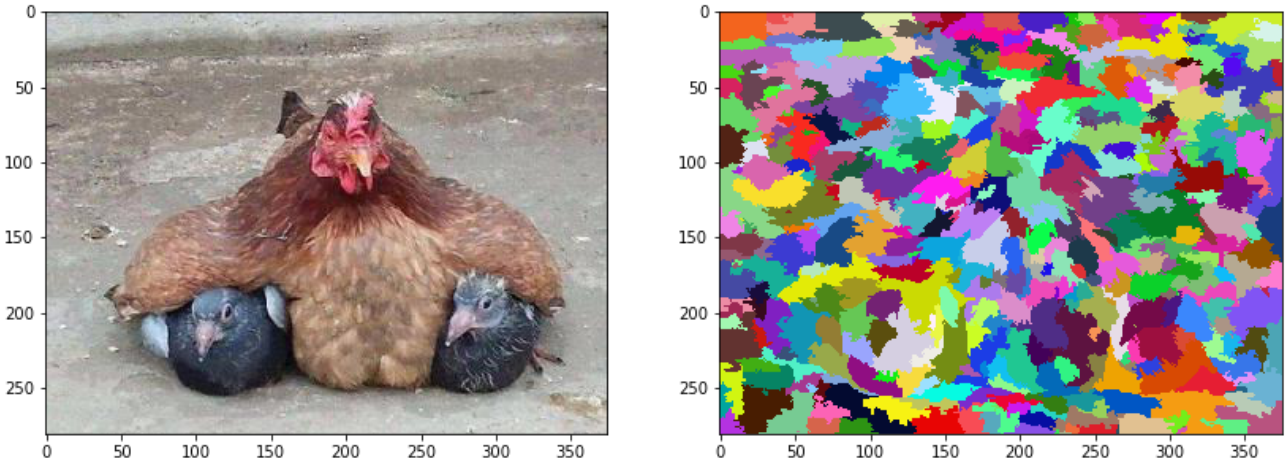
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 2
00
```

In [16]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 300
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



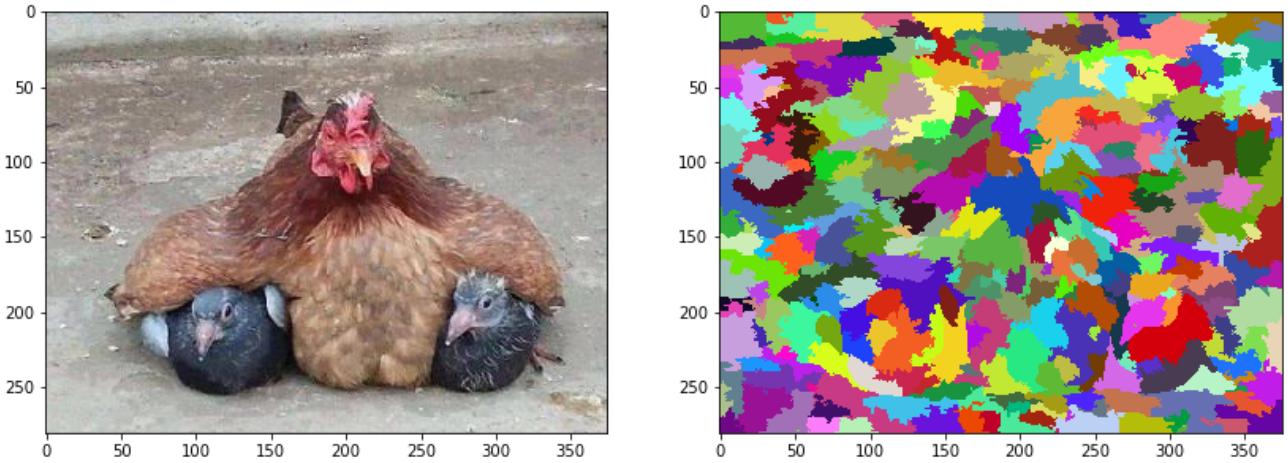
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 3
00
```

In [17]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 4
00
```

In [18]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 500
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

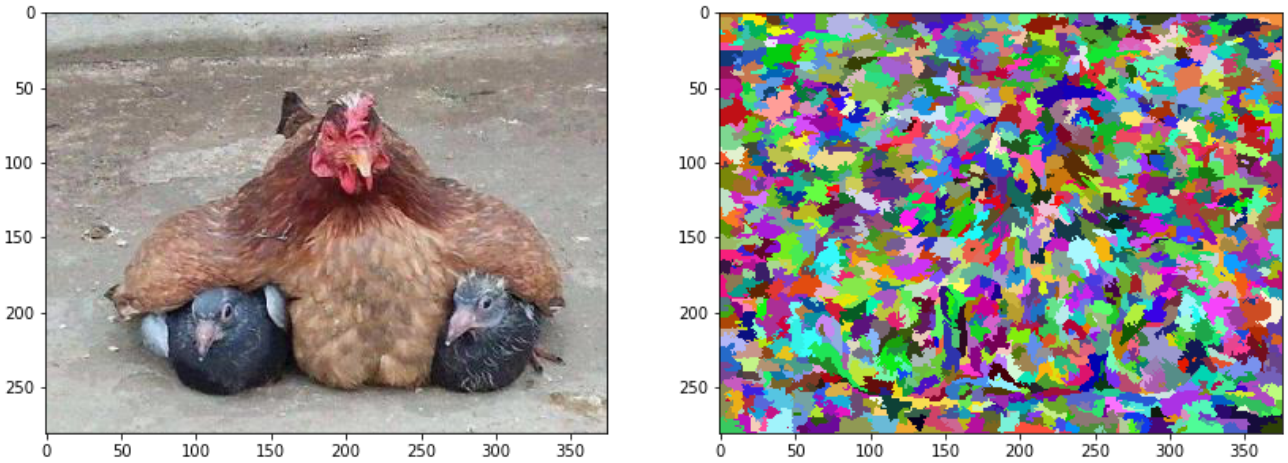


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 5
00
```

Variaciones de Sigma

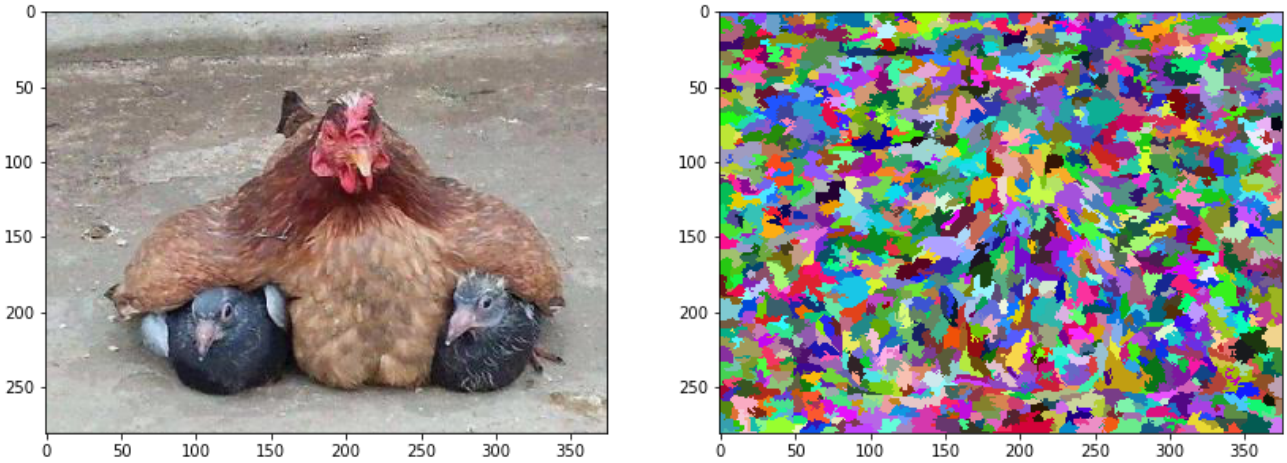
Se probará con k = 0.1 (Muy bajo) y MinSize = 100 (Muy bajo)

In [19]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



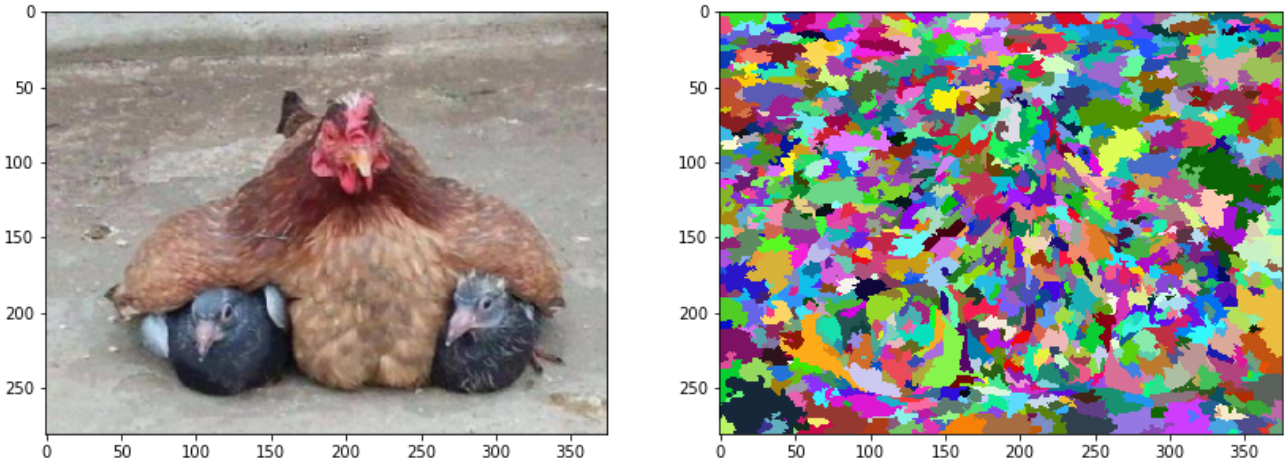
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 1
00
```

In [20]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.3; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



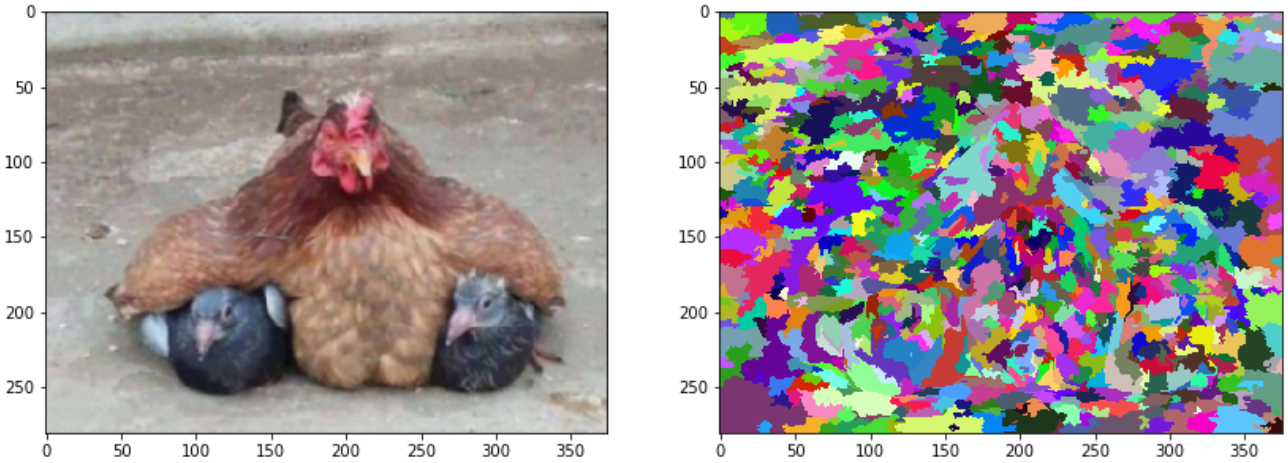
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.3 ; k: 0.1 ; Minimum Size: 1
00
```

In [21]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.6; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



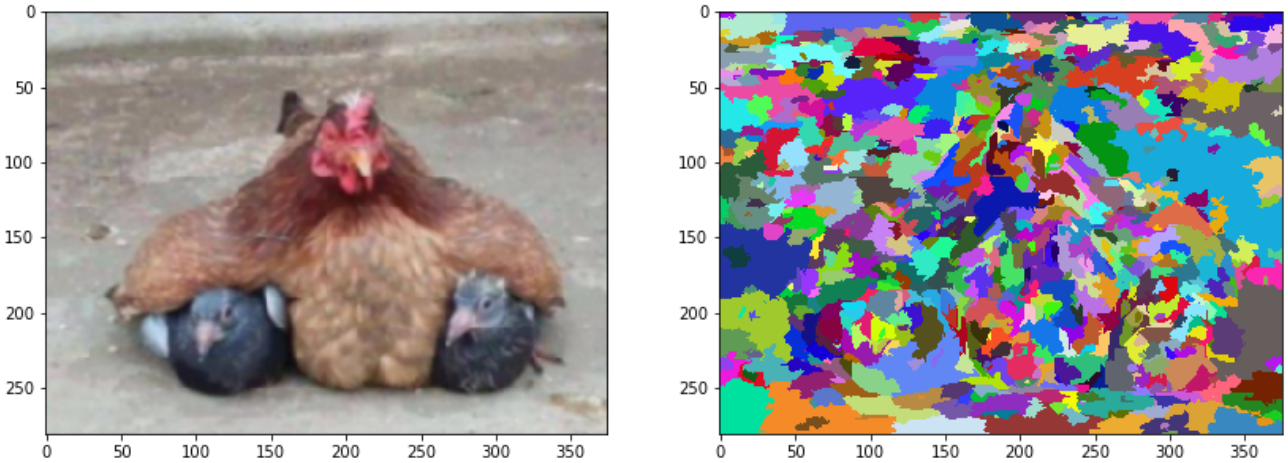
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.6 ; k: 0.1 ; Minimum Size: 1
00
```

In [22]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.9; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



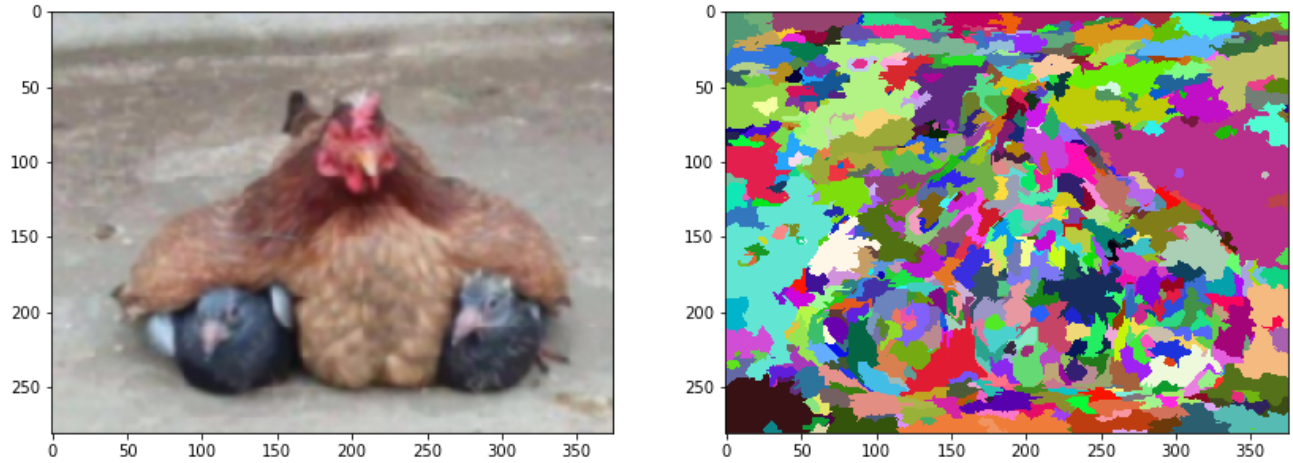
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.9 ; k: 0.1 ; Minimum Size: 1
00
```

In [23]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1.2; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



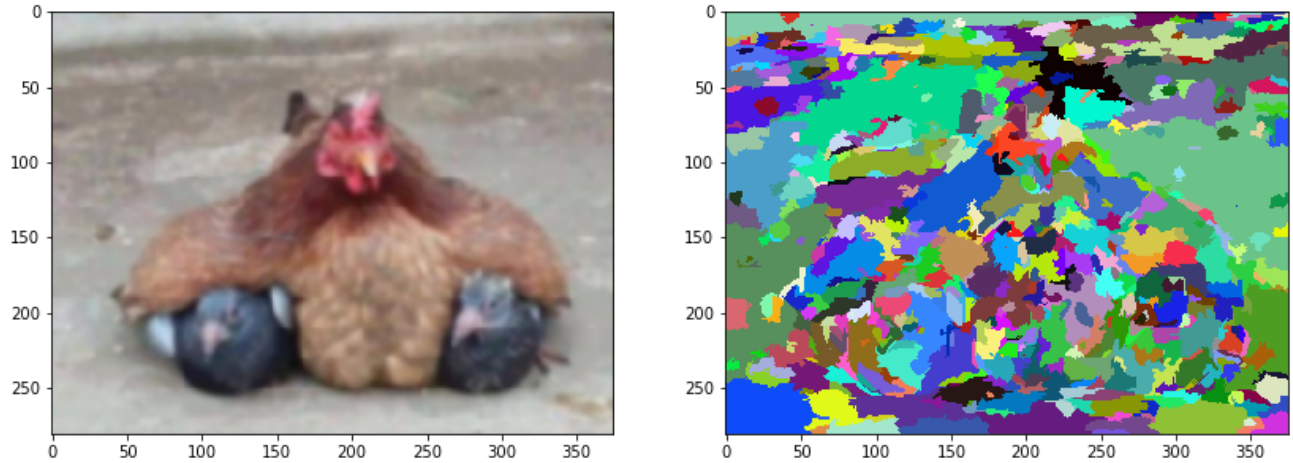
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1.2 ; k: 0.1 ; Minimum Size: 1
00
```

In [24]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1.5; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1.5 ; k: 0.1 ; Minimum Size: 1
00
```

In [25]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1.8; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

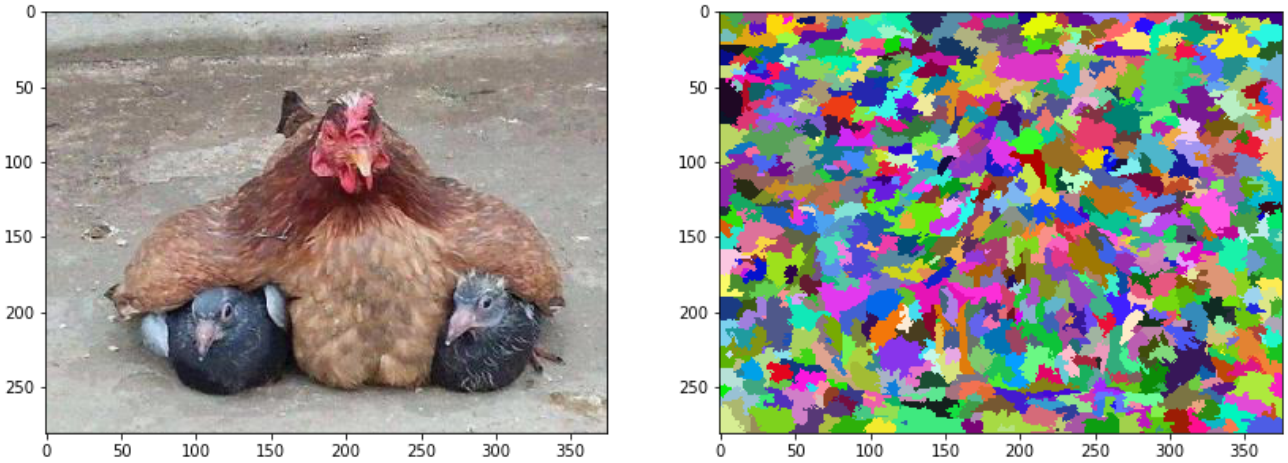


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1.8 ; k: 0.1 ; Minimum Size: 1
00
```

Variaciones conjuntas de k y MinSize

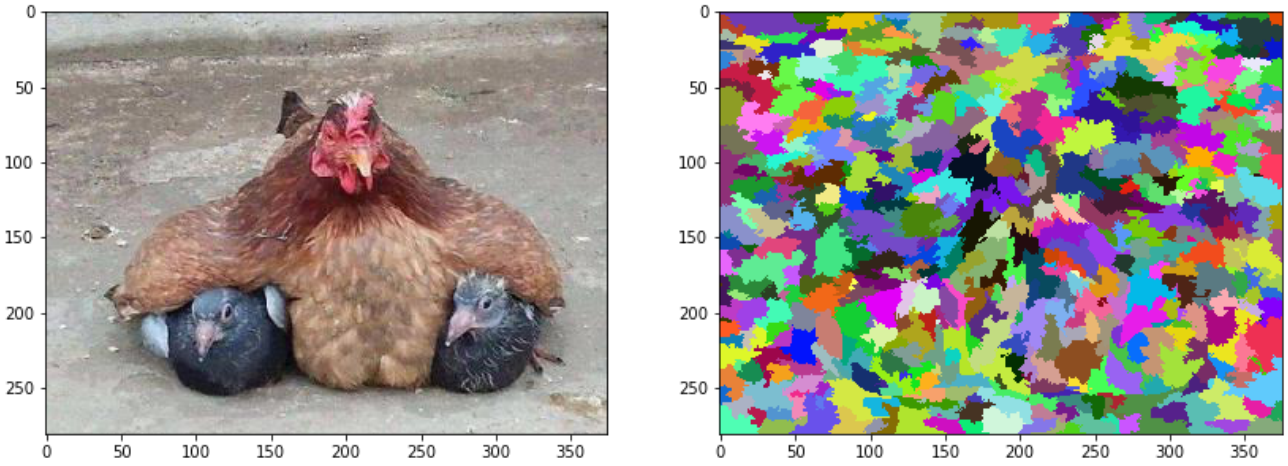
Se probará con Sigma=0.1 (muy bajo)

In [26]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.3;
 minimumSize = 200
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



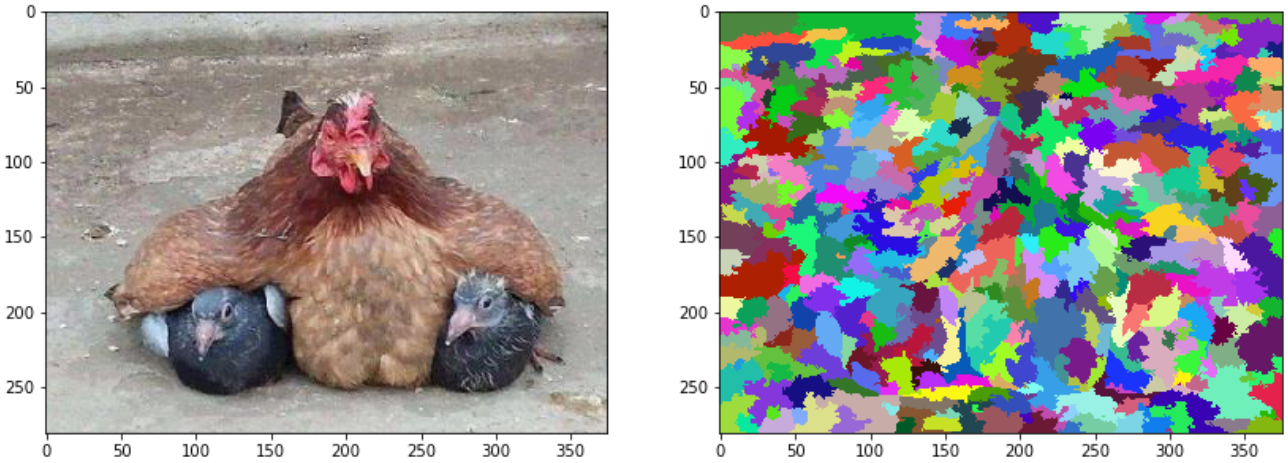
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.3 ; Minimum Size: 2
00
```

In [27]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.5;
 minimumSize = 300
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



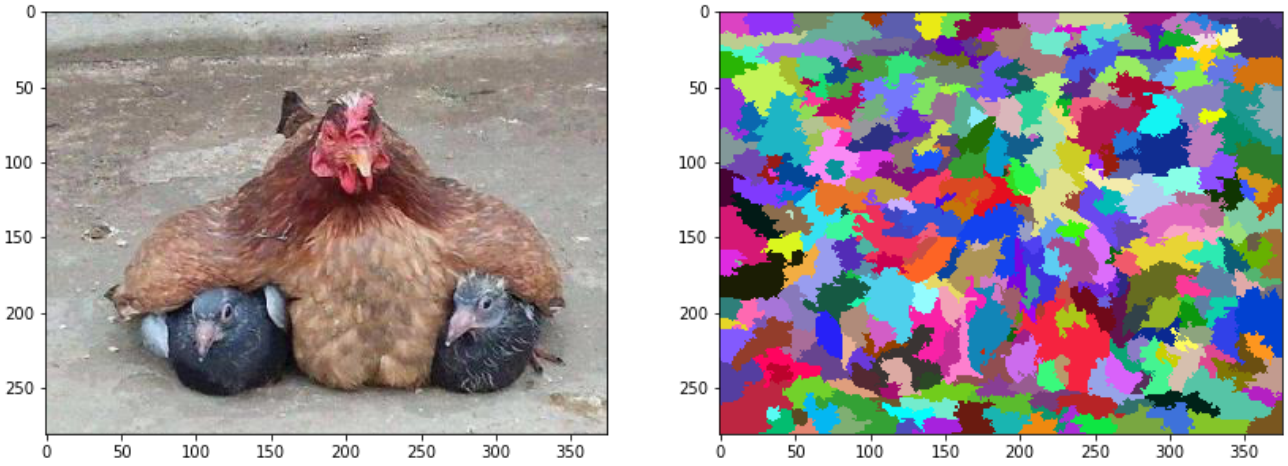
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.5 ; Minimum Size: 3
00
```

In [28]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.7;
 minimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.7 ; Minimum Size: 4
00
```

In [29]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.9;
 minimumSize = 500
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

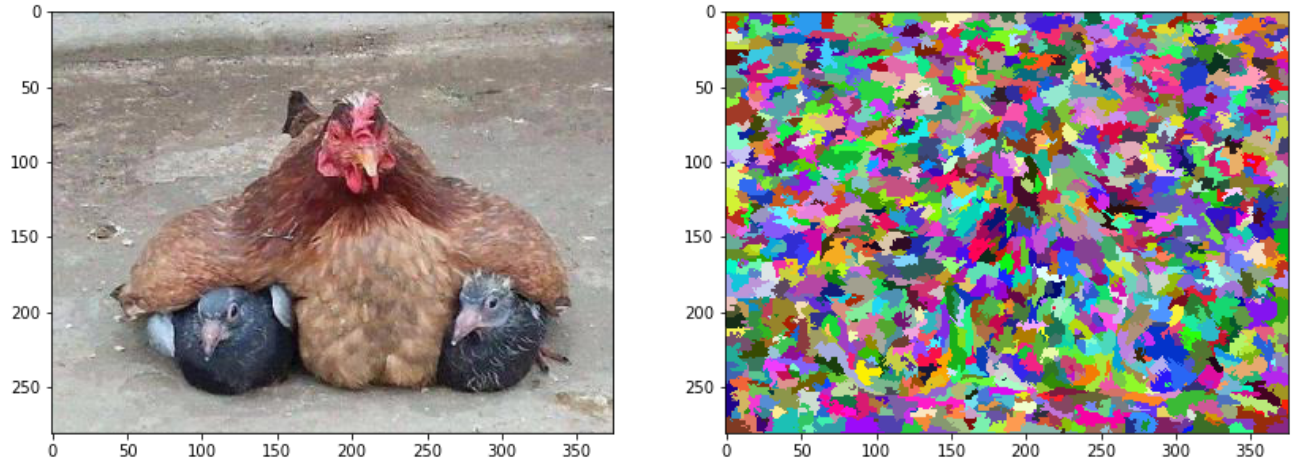


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.9 ; Minimum Size: 5
00
```

Variaciones conjuntas de Sigma y k

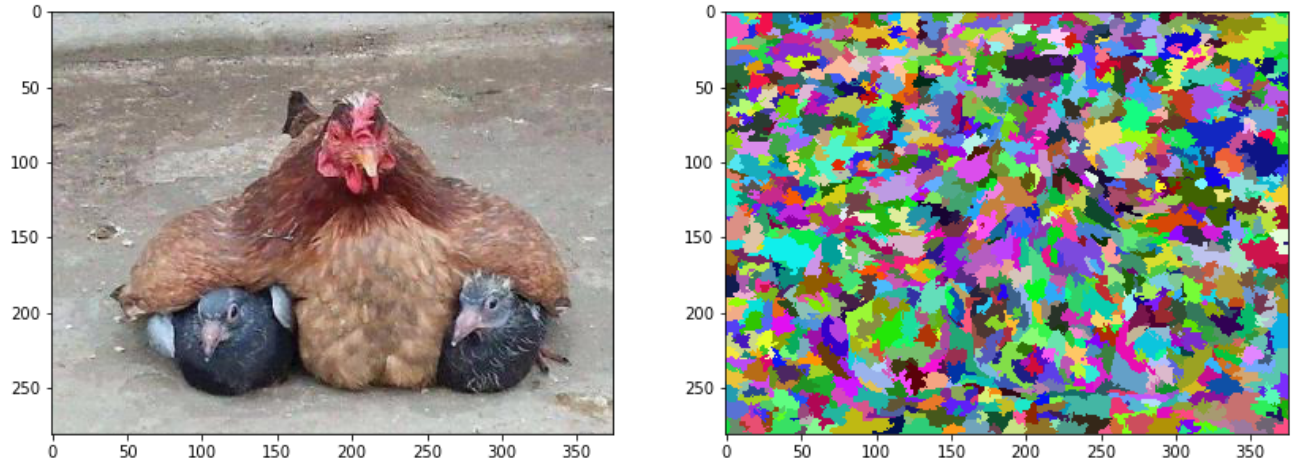
Se probará con MinSize=100 (muy bajo)

In [30]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



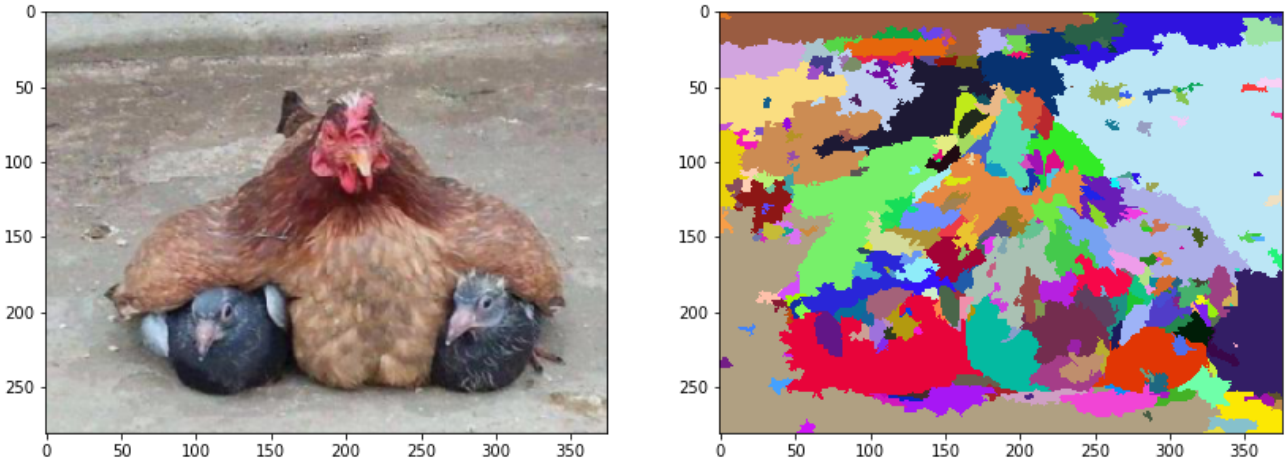
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 1
00
```

In [31]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.3; k = 0.3;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



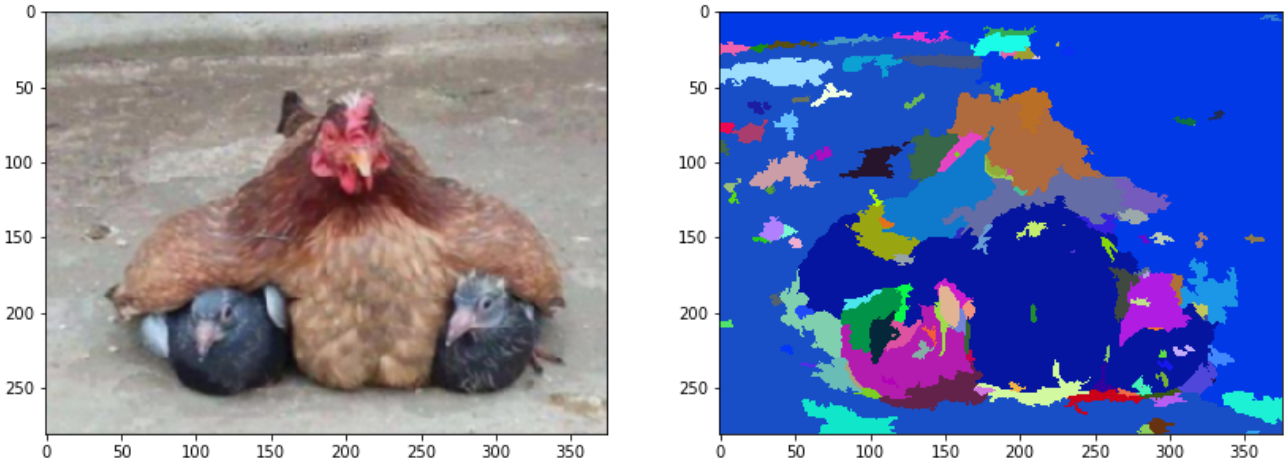
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.3 ; k: 0.3 ; Minimum Size: 1
00
```

In [32]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.6; k = 0.5;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



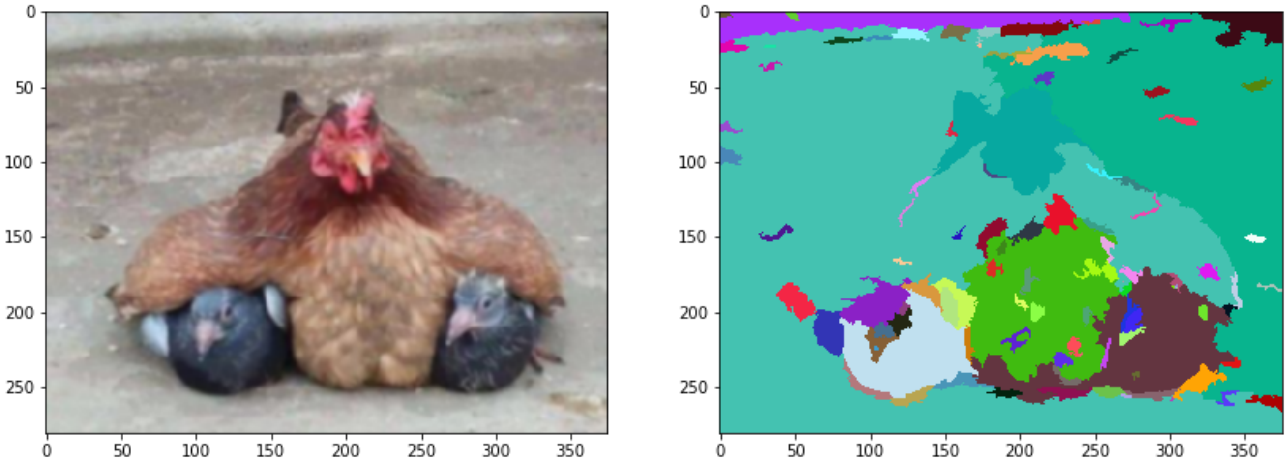
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.6 ; k: 0.5 ; Minimum Size: 1
00
```

In [33]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.9; k = 0.7;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.9 ; k: 0.7 ; Minimum Size: 1
00
```

In [34]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1.2; k = 0.9;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

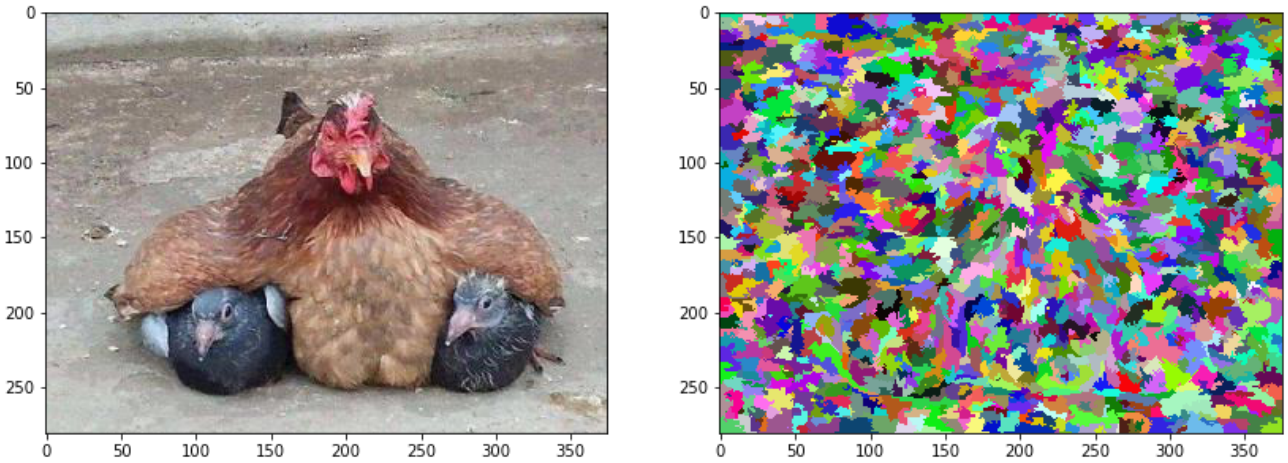


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1.2 ; k: 0.9 ; Minimum Size: 1
00
```

Variaciones conjuntas de Sigma y MinSize

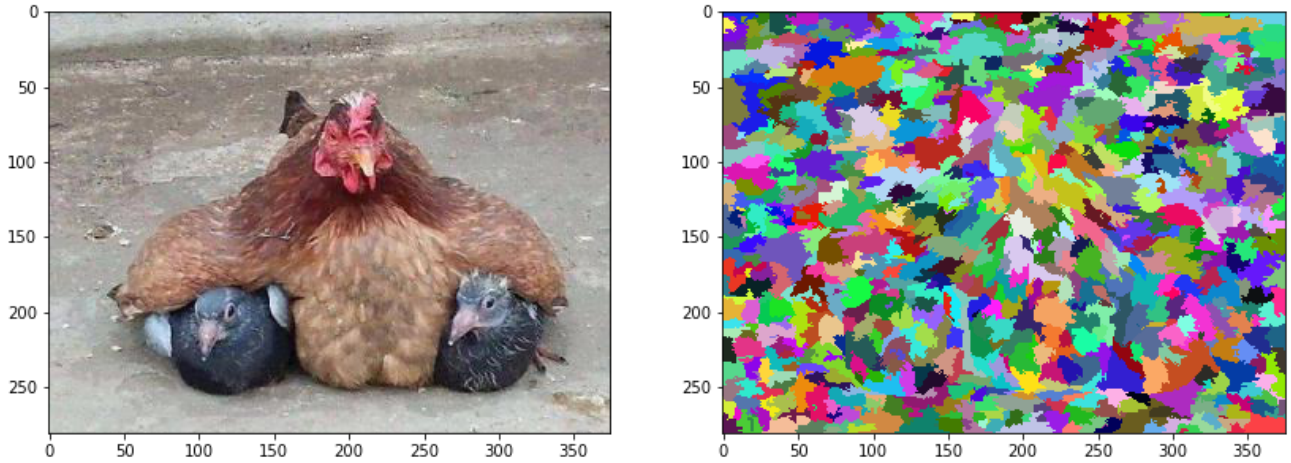
Se probará con k=0.1 (muy bajo)

In [35]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



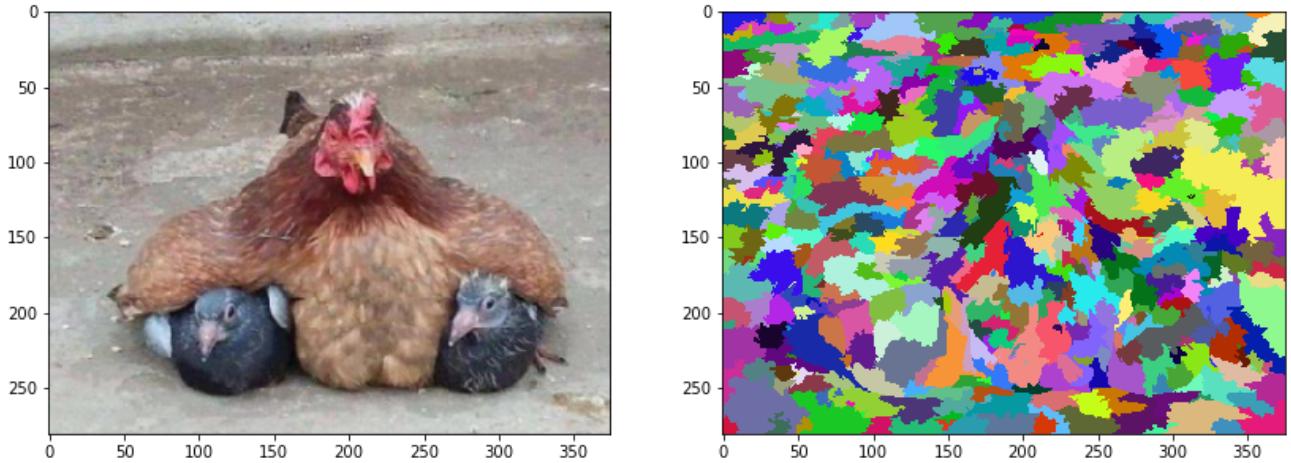
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 1
00
```

In [36]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.3; k = 0.1;
 minimumSize = 200
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



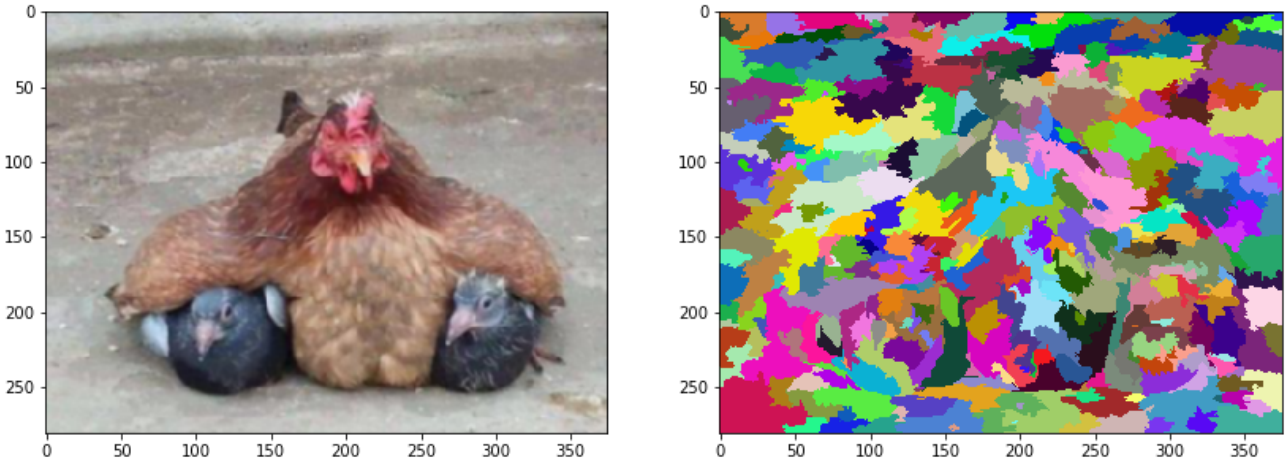
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.3 ; k: 0.1 ; Minimum Size: 2
00
```

In [37]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.6; k = 0.1;
 minimumSize = 300
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



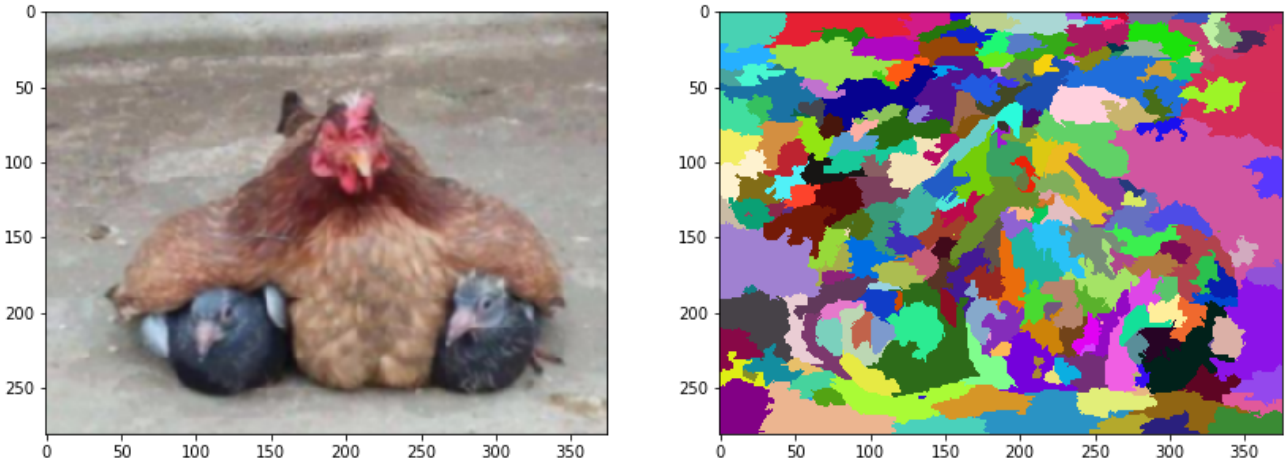
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.6 ; k: 0.1 ; Minimum Size: 3
00
```

In [38]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.9; k = 0.1;
 minimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.9 ; k: 0.1 ; Minimum Size: 4
00
```

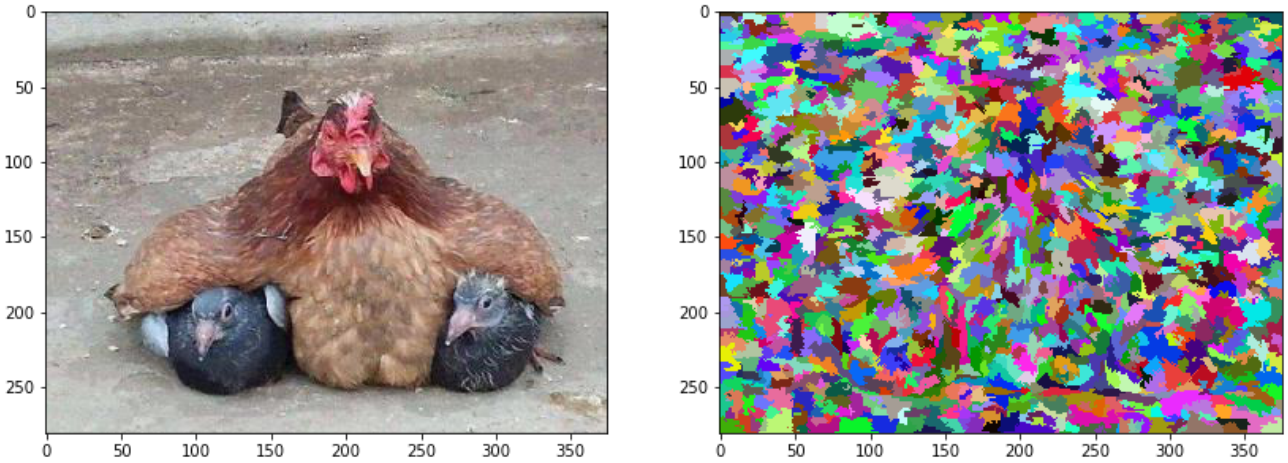
In [39]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1.2; k = 0.1;
 minimumSize = 500
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1.2 ; k: 0.1 ; Minimum Size: 5
00
```

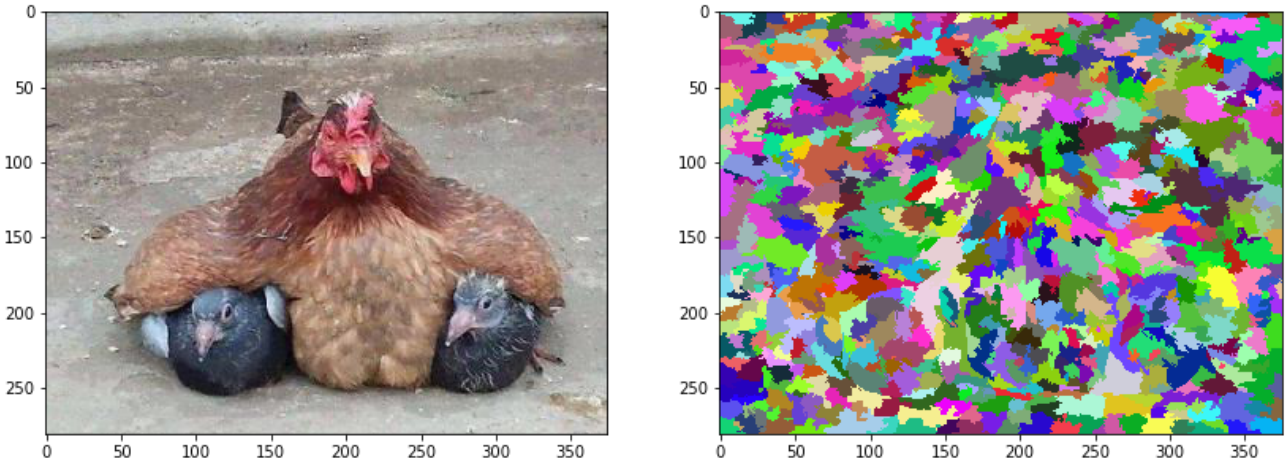
Variaciones conjuntas de Sigma, k y MinSize

In [40]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.1; k = 0.1;
 minimumSize = 100
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



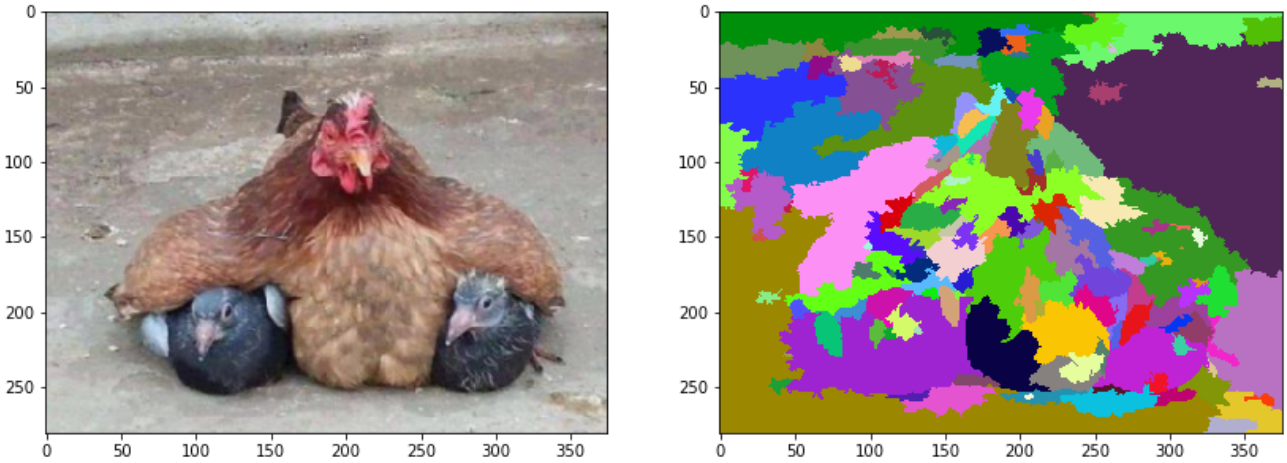
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.1 ; k: 0.1 ; Minimum Size: 1
00
```

In [41]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.3; k = 0.3;
 minimumSize = 200
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



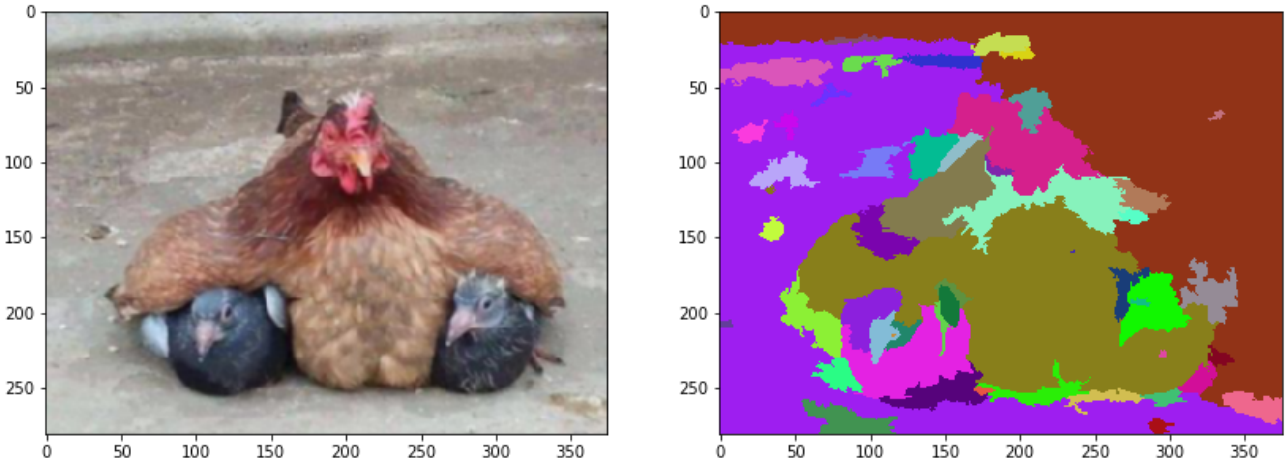
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.3 ; k: 0.3 ; Minimum Size: 2
00
```

In [42]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.6; k = 0.5;
 minimumSize = 300
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



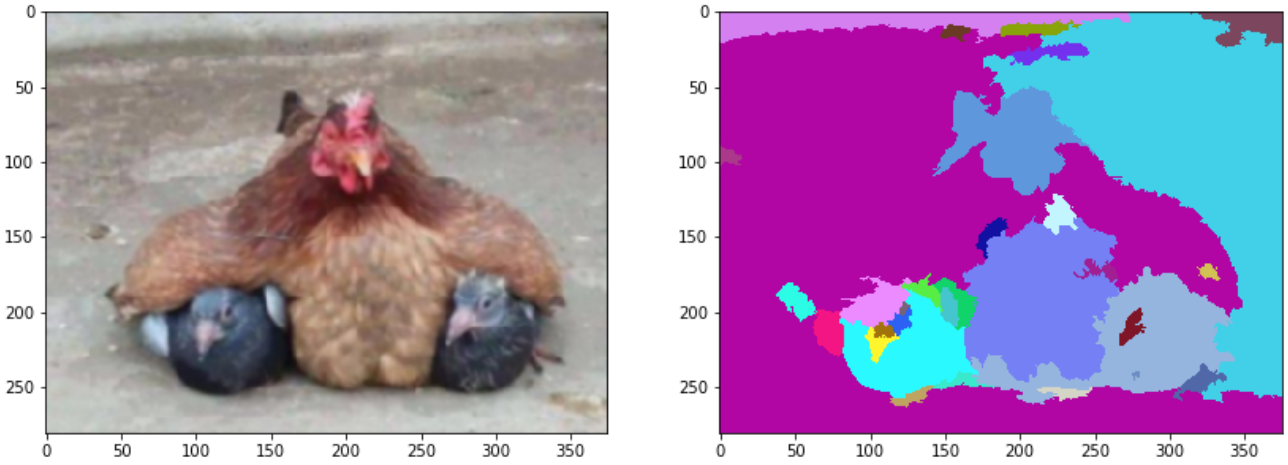
```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.6 ; k: 0.5 ; Minimum Size: 3
00
```

In [43]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 0.9; k = 0.7;
 minimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 0.9 ; k: 0.7 ; Minimum Size: 4
00
```

In [44]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1.2; k = 0.9;
 minimumSize = 500
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

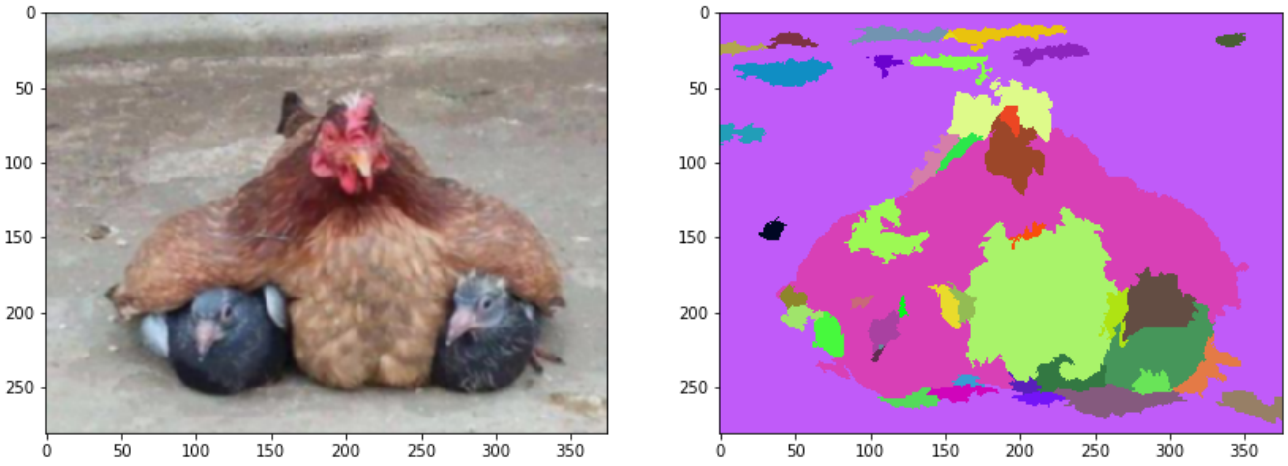


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1.2 ; k: 0.9 ; Minimum Size: 5
00
```

Mejor resultado

Sigma = 1; K = 0.8; MinSize = 400

In [48]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

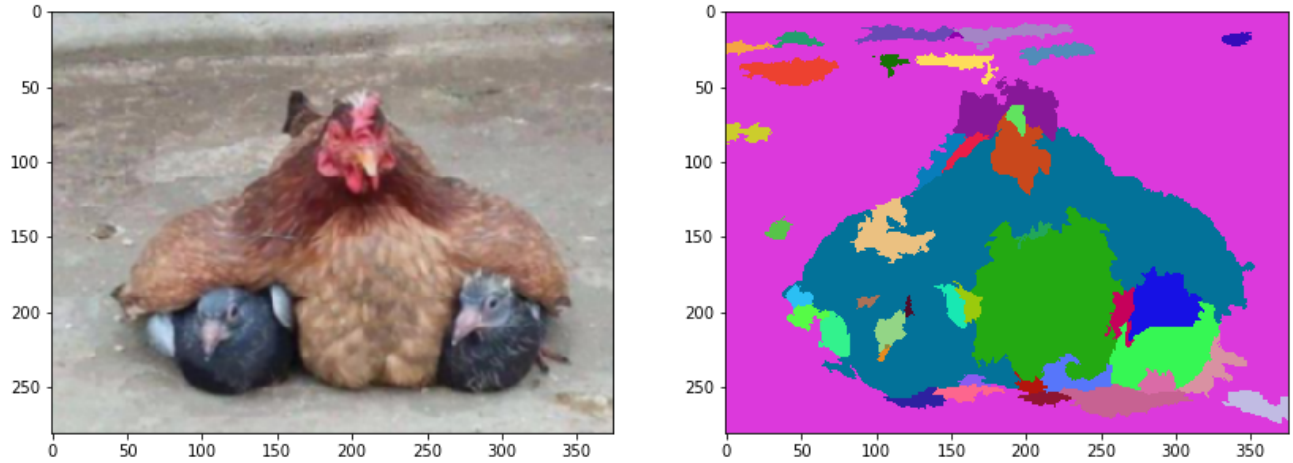


```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400
```

Variaciones en espacios de color

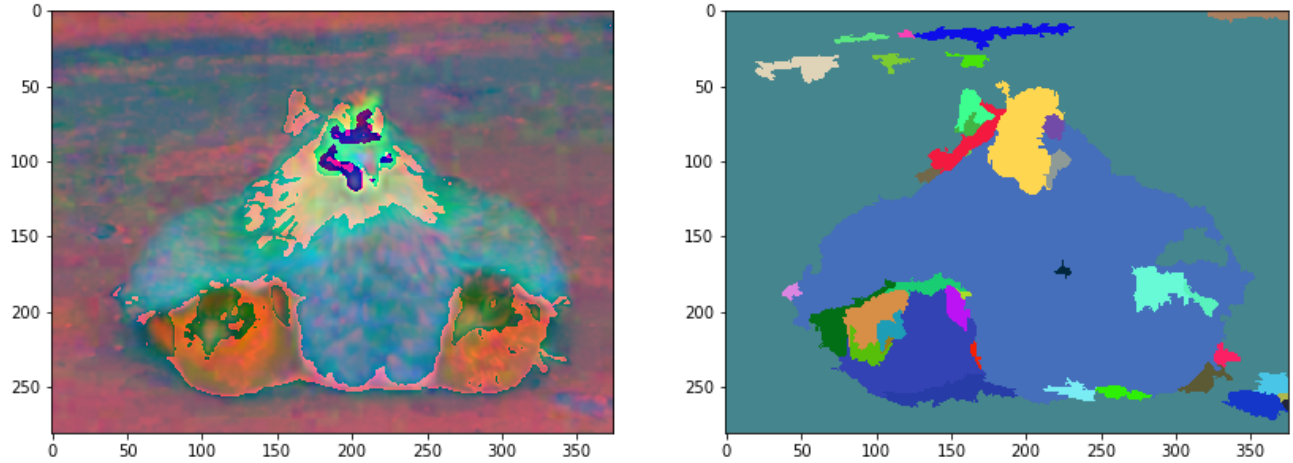
Se considera el mejor caso anterior: Sigma = 1, k = 0.8 y MinSize = 400

In [49]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[0]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400
```

In [50]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[1]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size:
400
```

In [51]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[2]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

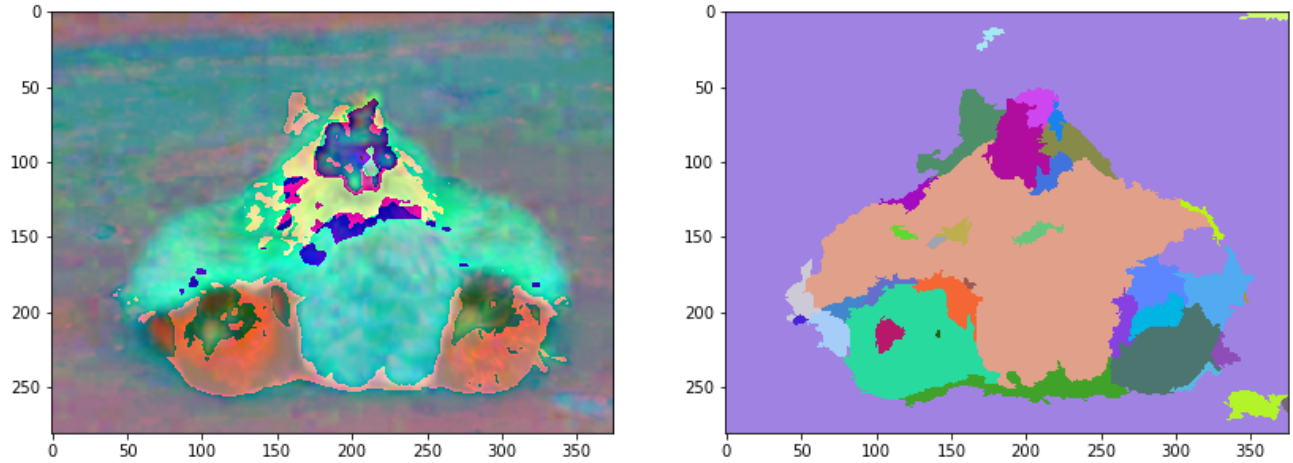


Image: photos-1-17_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

In [52]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[3]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```

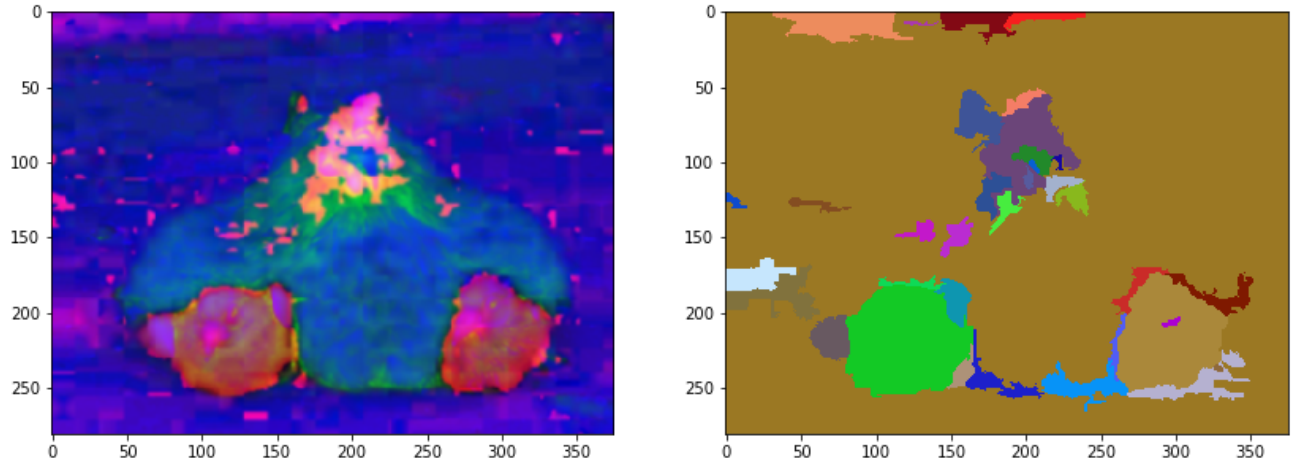
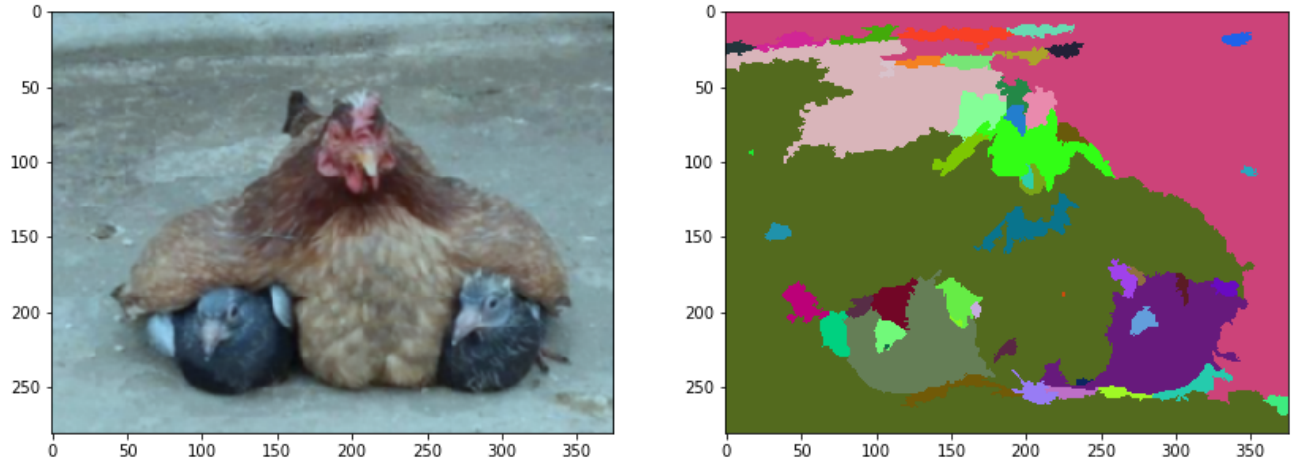


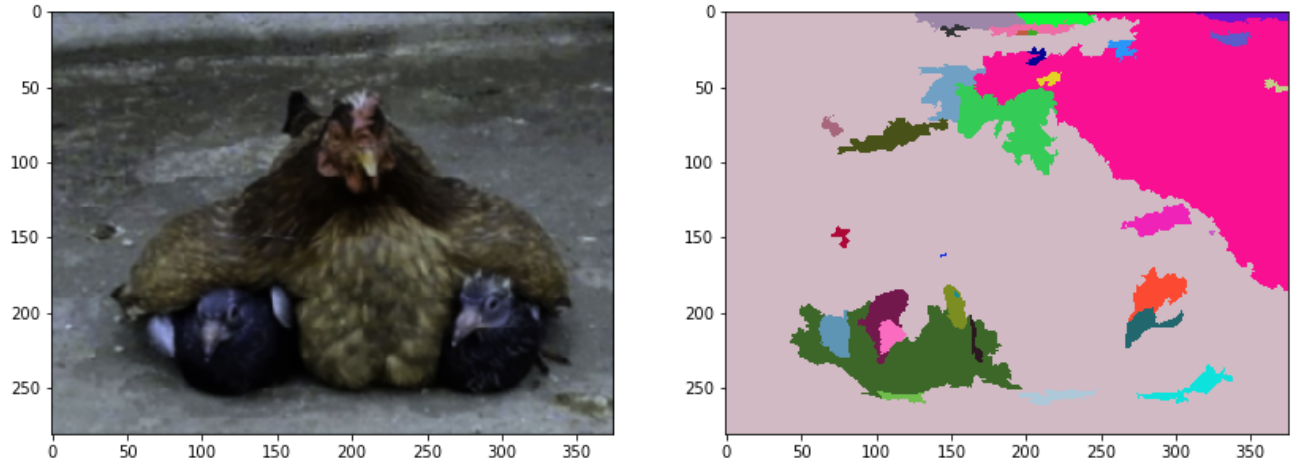
Image: photos-1-17_05.jpg ; Color Space: HSV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

In [53]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[4]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: RGB CIE ; Sigma: 1 ; k: 0.8 ; Minimum Size:
400
```

In [54]:
```
imageFile = imageOptions[0]; colorSpace = colorSpaceOptions[5]; sigma = 1; k = 0.8; m
inimumSize = 400
segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k = k, mi
nimumSize = minimumSize)
print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, "; k:",
k, "; Minimum Size:", minimumSize)
```



```
Image: photos-1-17_05.jpg ; Color Space: XYZ ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400
```

Pruebas con otras imágenes

Si bien se solicitan pruebas con RGB, CIE LAB, se entregan también resultados para CIE LUV, que entregó muy buenos resultados en la primera imagen.

In [56]:
```
for i in range(1, 10):
    imageFile = imageOptions[i]; colorSpace = colorSpaceOptions[0]; sigma = 1; k = 0.
8; minimumSize = 400
    segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k =
k, minimumSize = minimumSize)
    print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, ";
 k:", k, "; Minimum Size:", minimumSize)

    imageFile = imageOptions[i]; colorSpace = colorSpaceOptions[1]; sigma = 1; k = 0.
8; minimumSize = 400
    segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k =
k, minimumSize = minimumSize)
    print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, ";
 k:", k, "; Minimum Size:", minimumSize)

    imageFile = imageOptions[i]; colorSpace = colorSpaceOptions[2]; sigma = 1; k = 0.
8; minimumSize = 400
    segmentImage(imageFile = imageFile, colorSpace = colorSpace, sigma = sigma, k =
k, minimumSize = minimumSize)
    print("Image:", imageFile, "; Color Space:", colorSpace, "; Sigma:", sigma, ";
 k:", k, "; Minimum Size:", minimumSize)
```

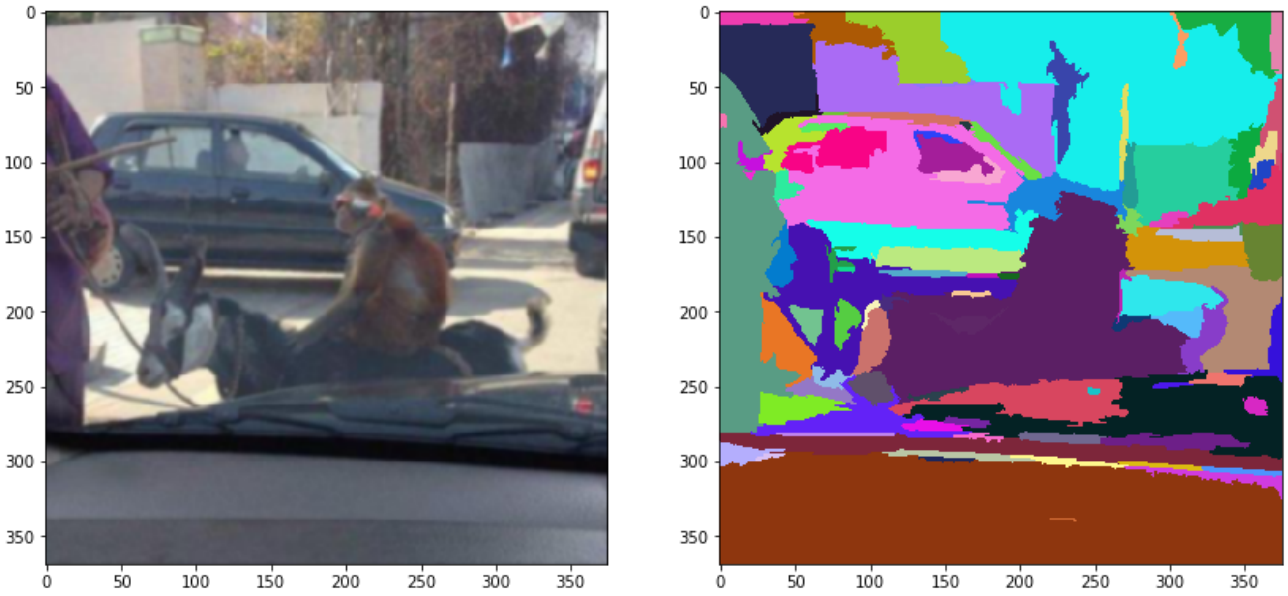


Image: photos-16-15_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

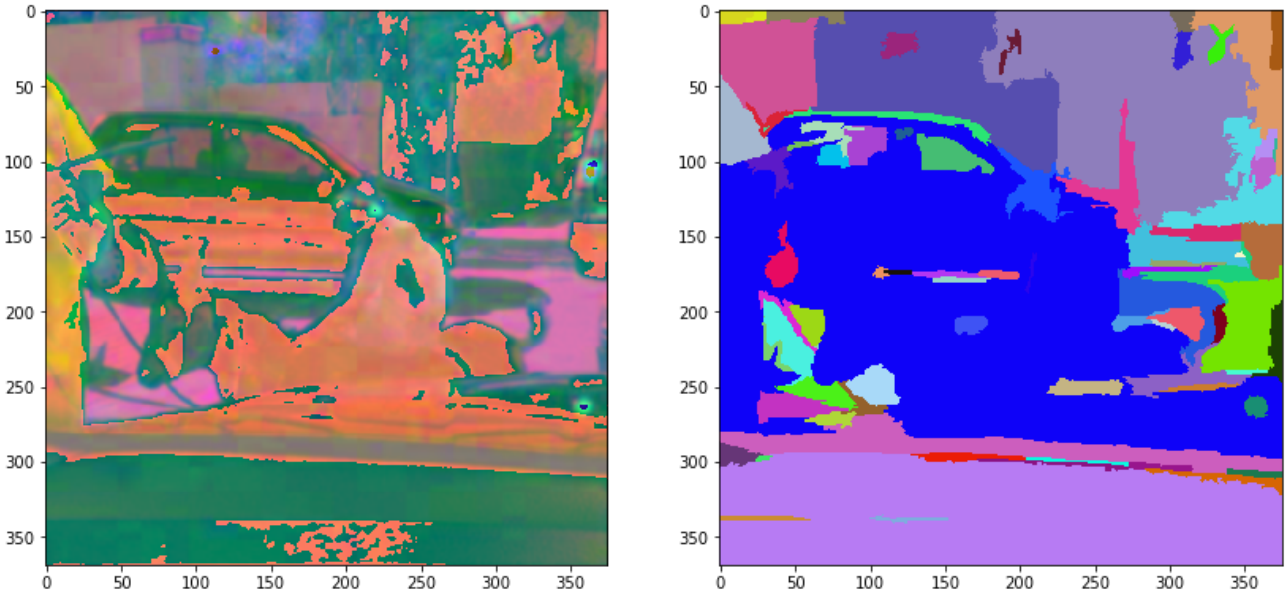


Image: photos-16-15_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

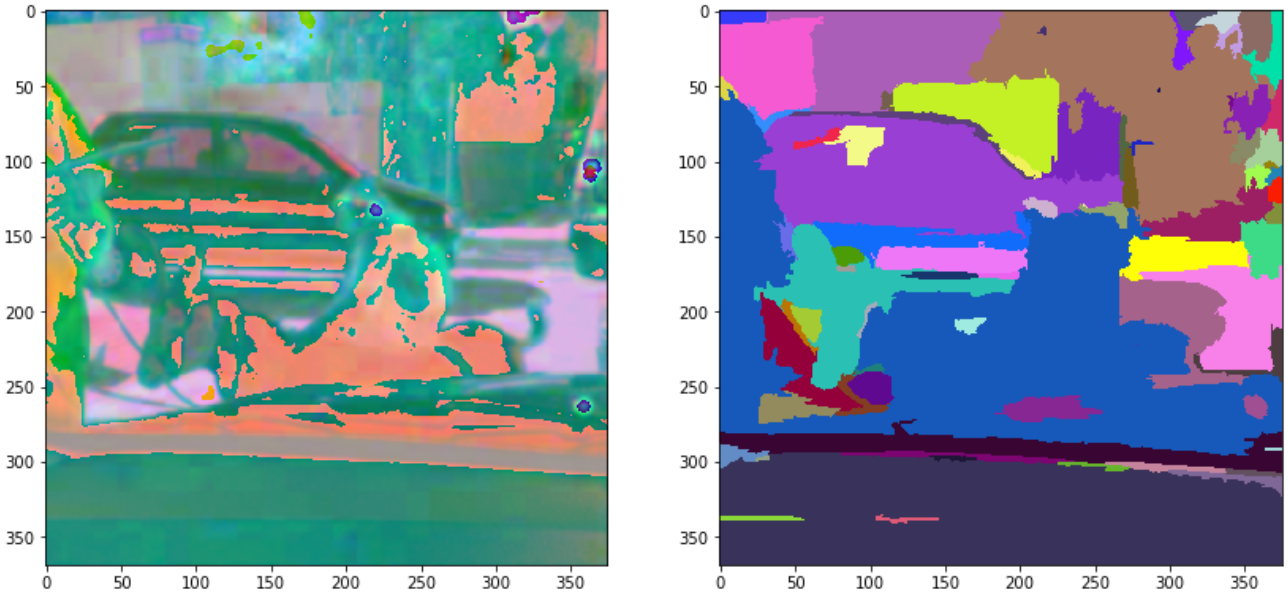


Image: photos-16-15_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

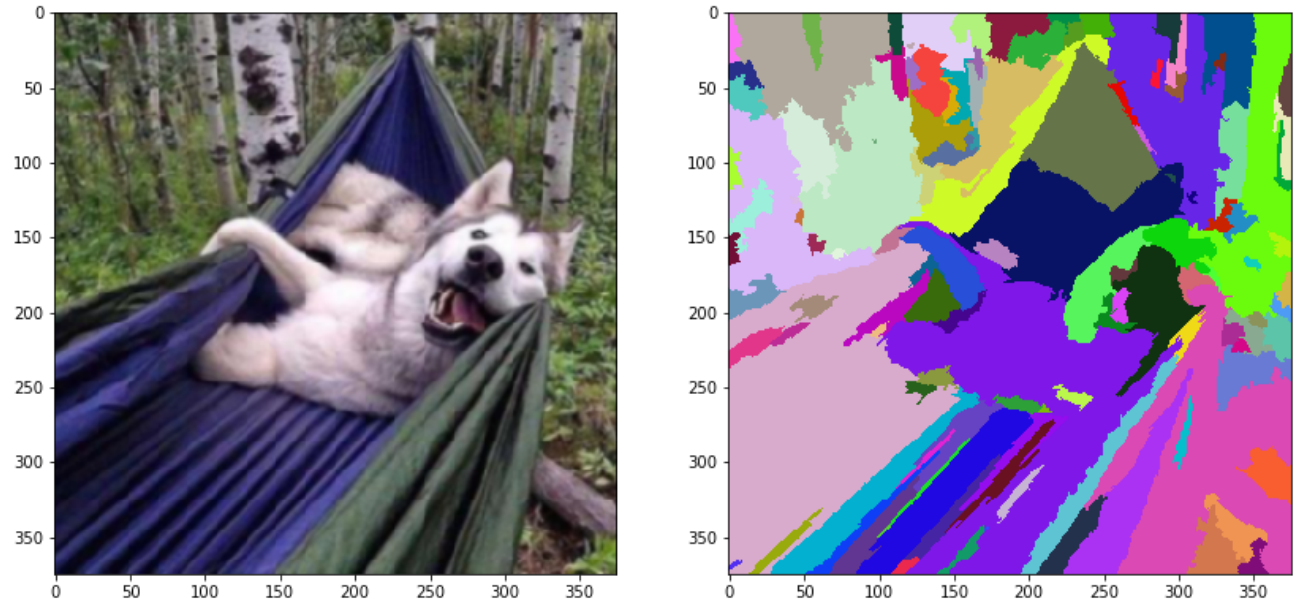


Image: photos-18-15_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

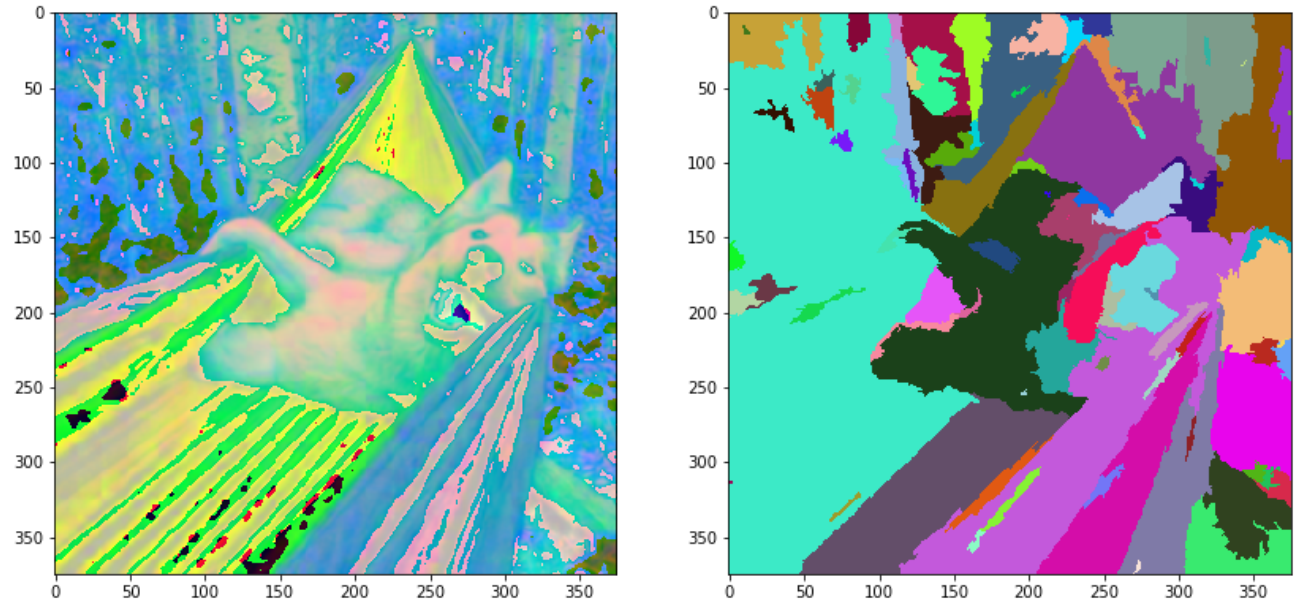


Image: photos-18-15_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

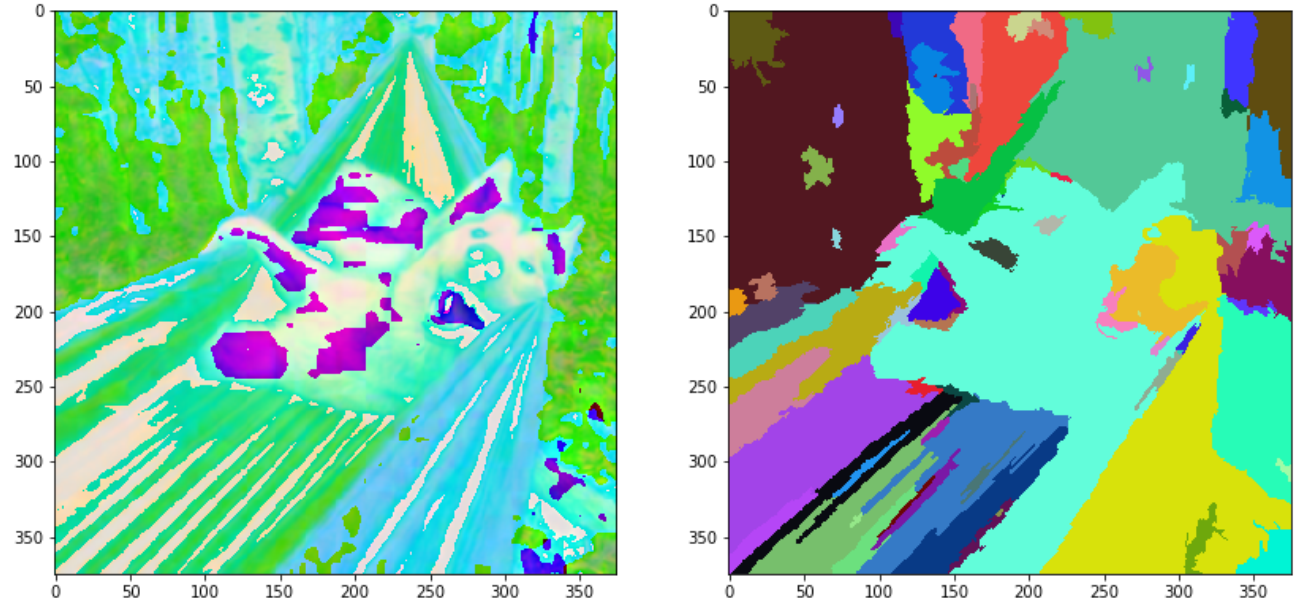


Image: photos-18-15_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

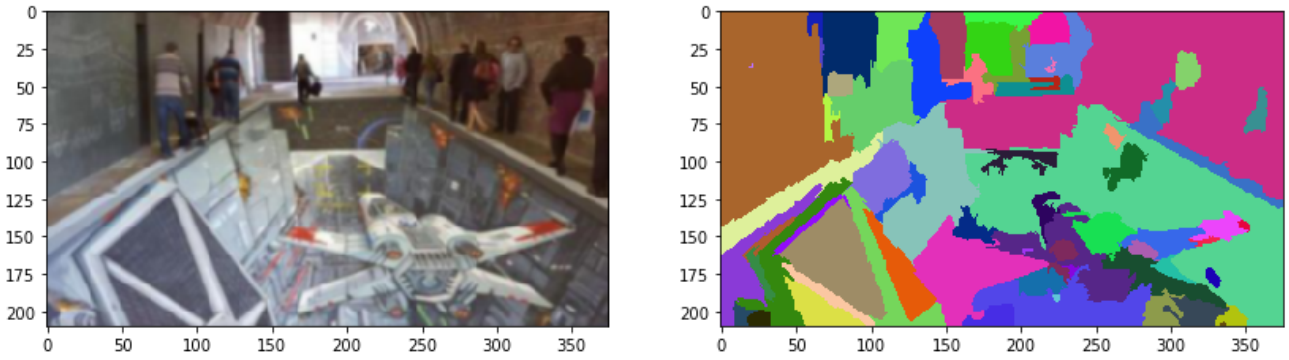


Image: photos-20-18_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

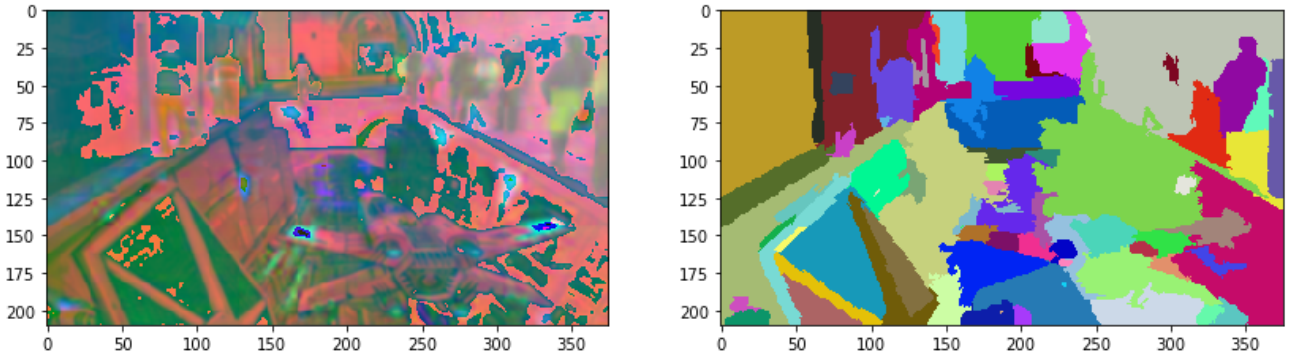


Image: photos-20-18_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

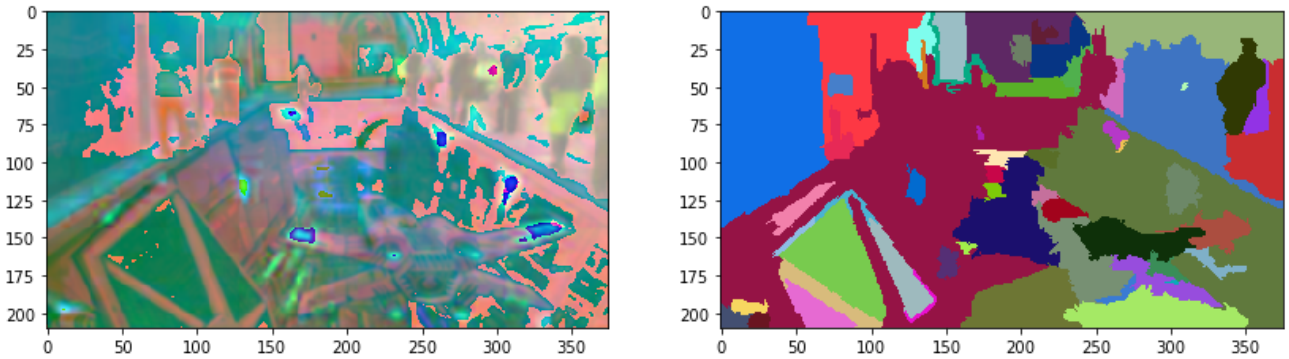


Image: photos-20-18_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

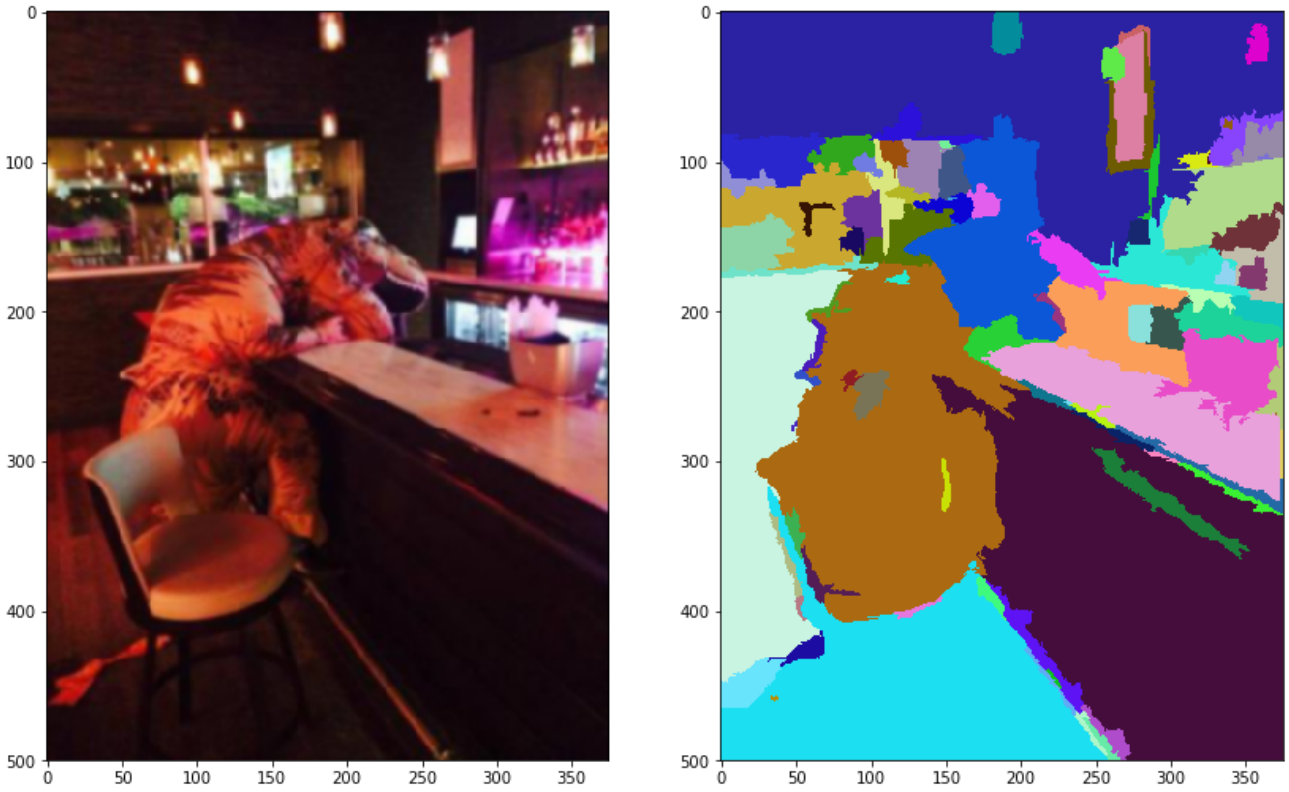


Image: photos-21-15_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

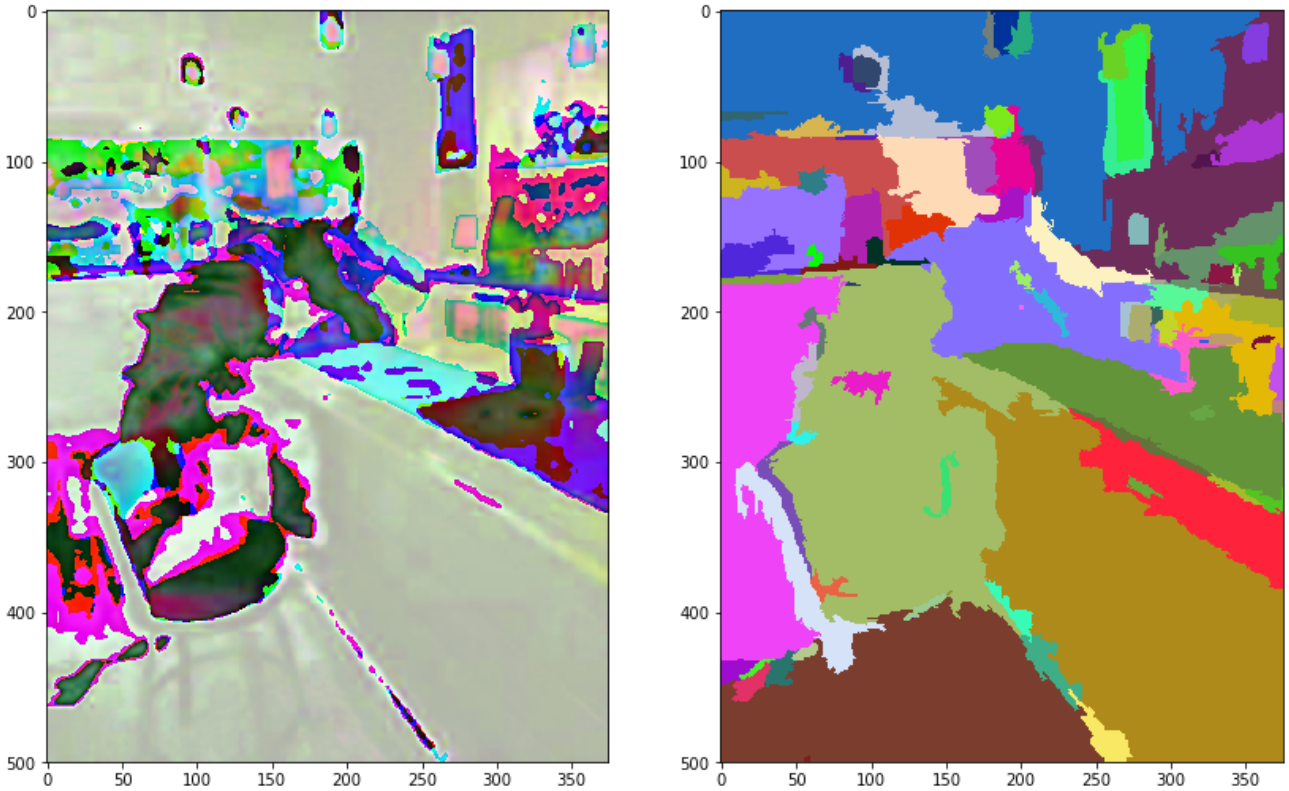


Image: photos-21-15_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

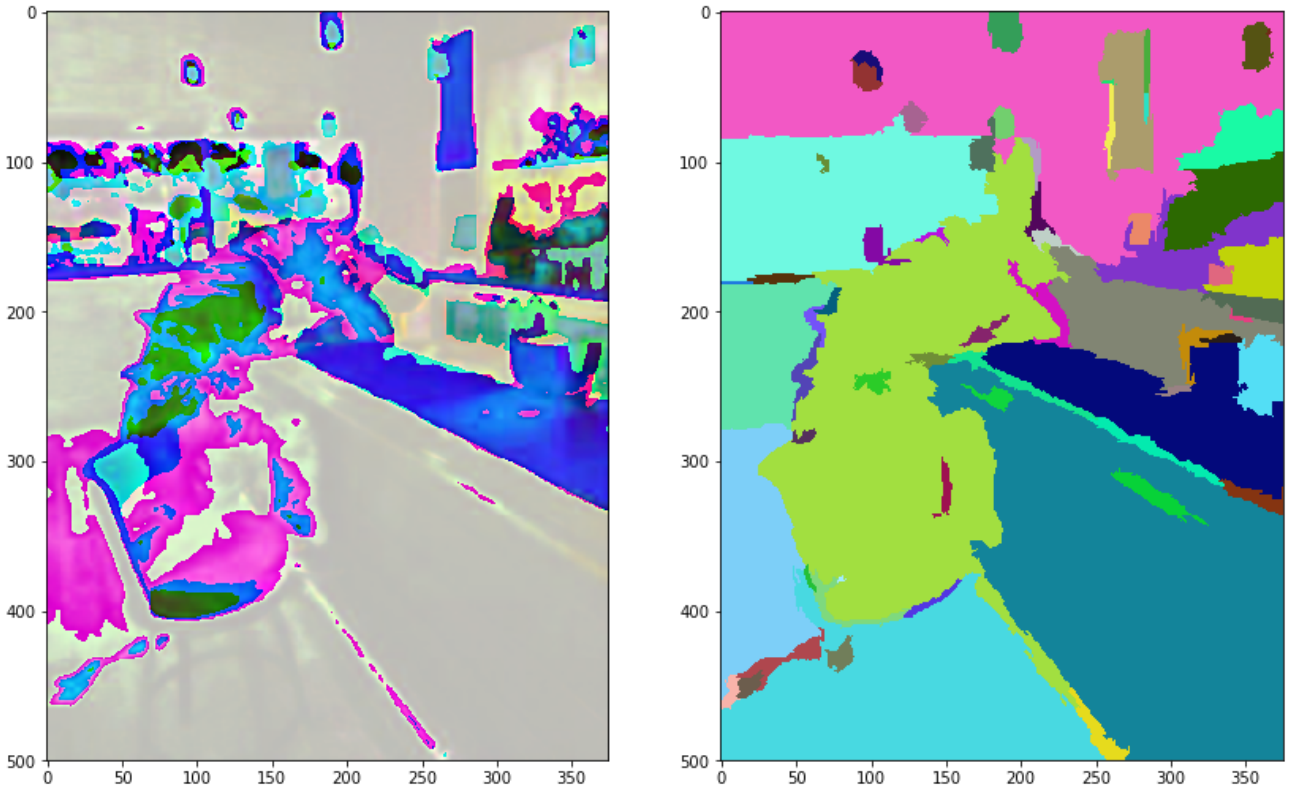


Image: photos-21-15_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

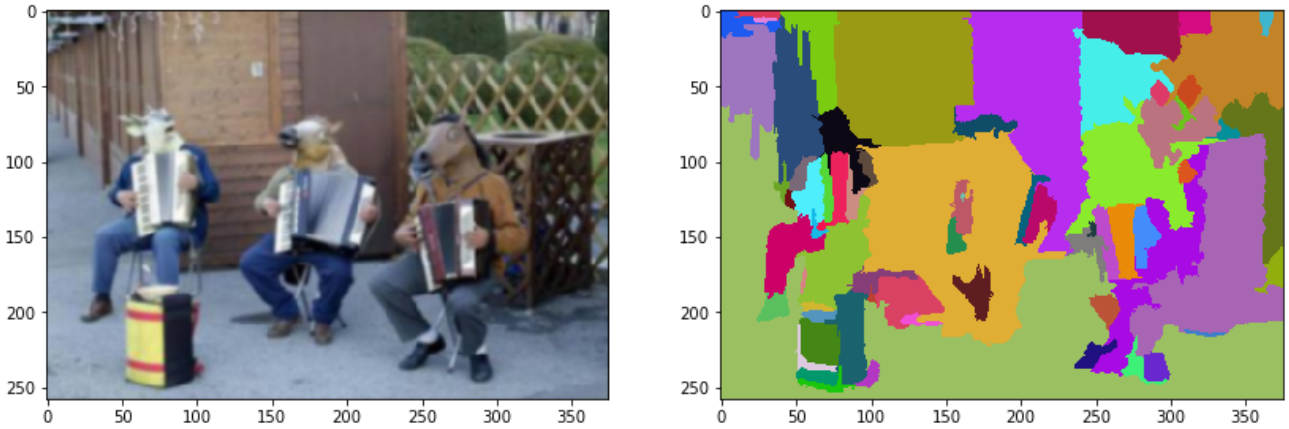


Image: photos-21-18_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

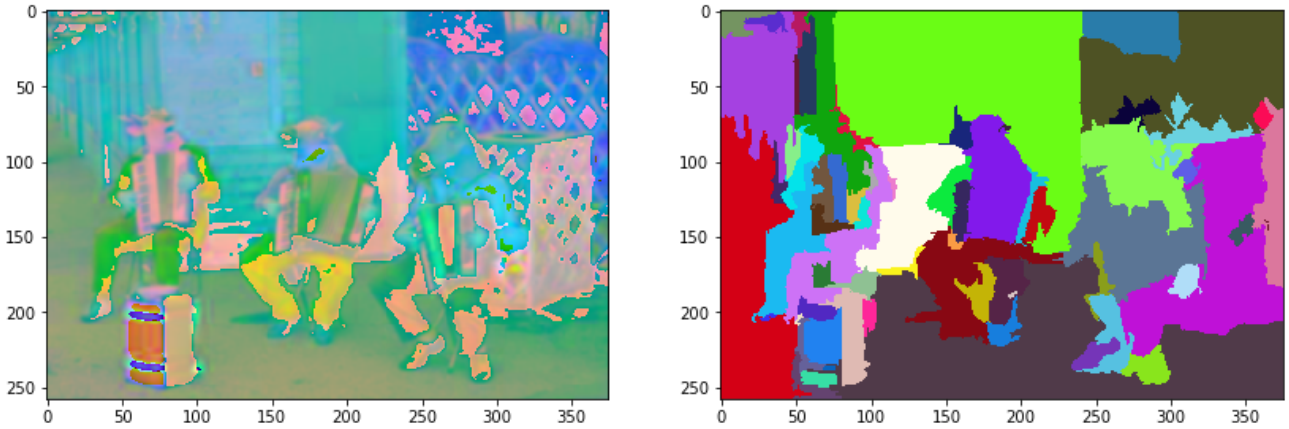


Image: photos-21-18_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

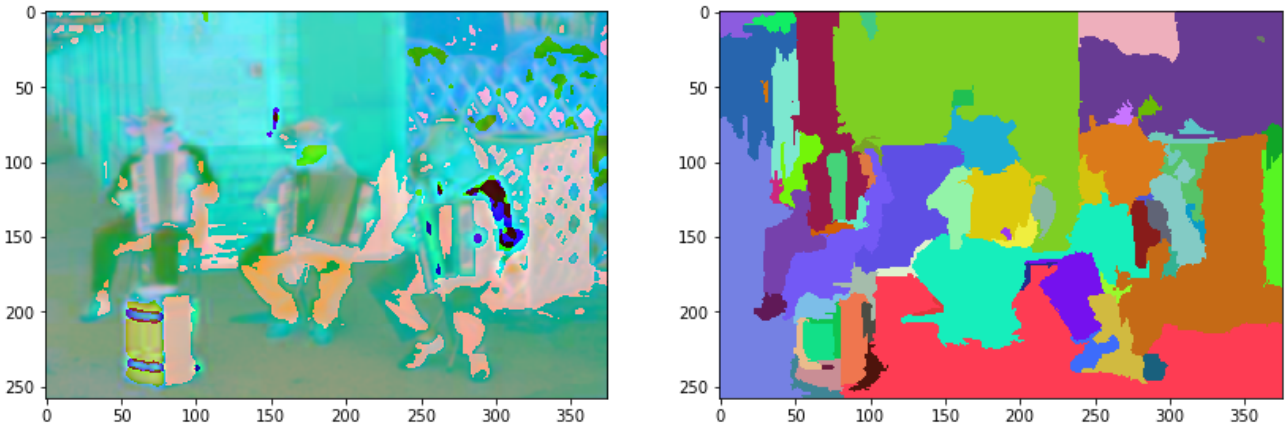


Image: photos-21-18_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

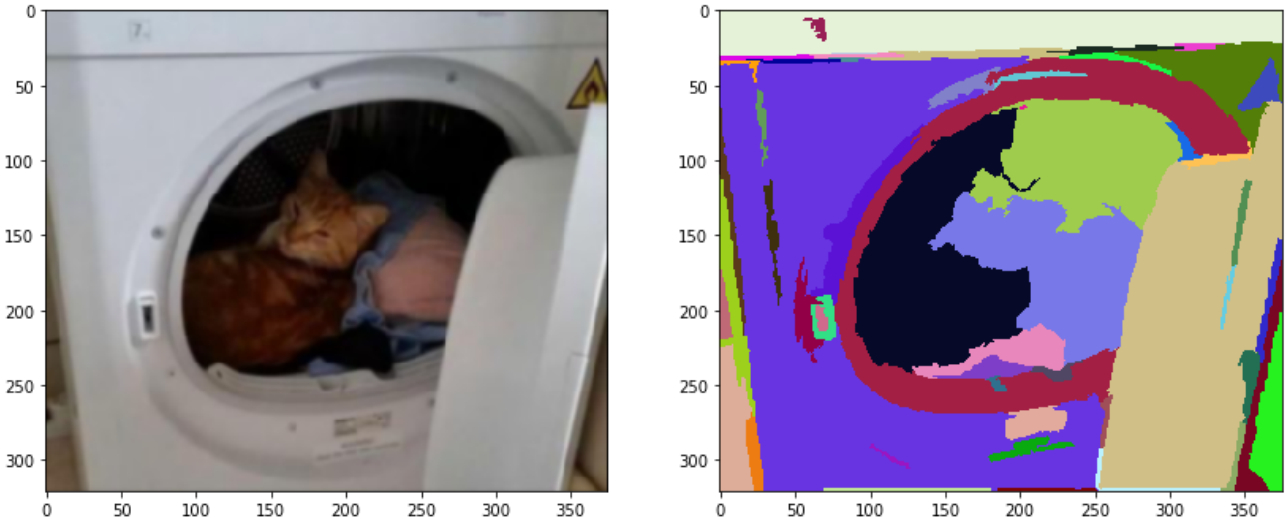


Image: photos-26-18_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

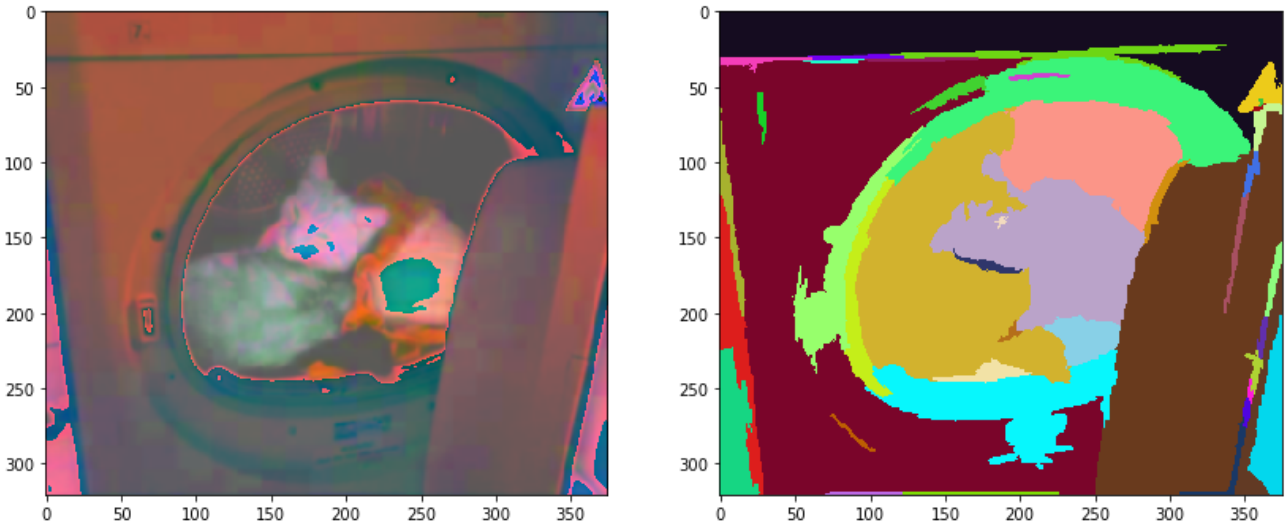


Image: photos-26-18_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

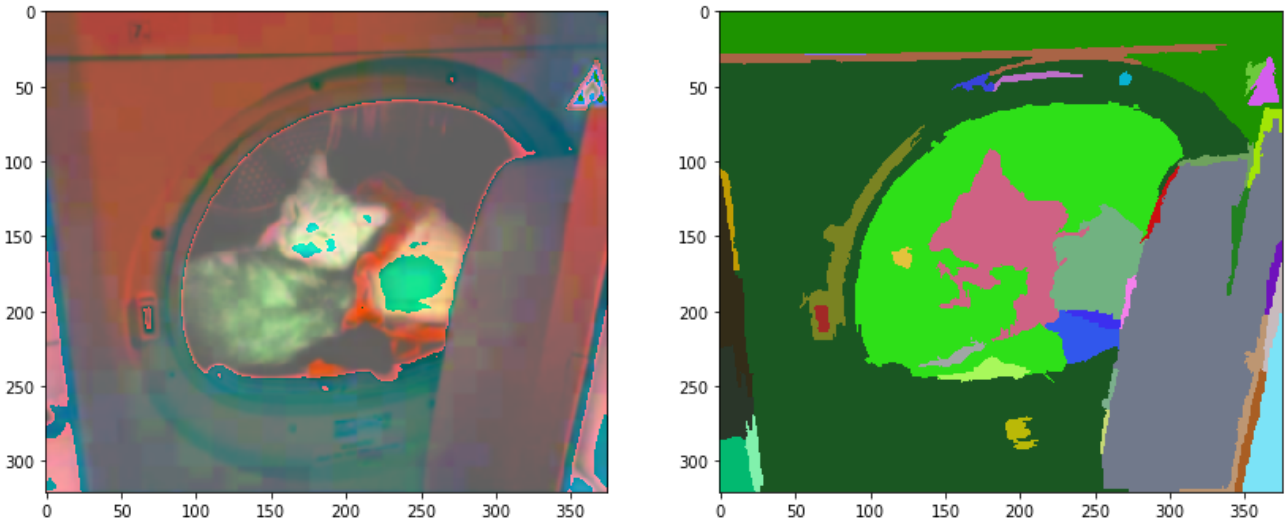


Image: photos-26-18_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

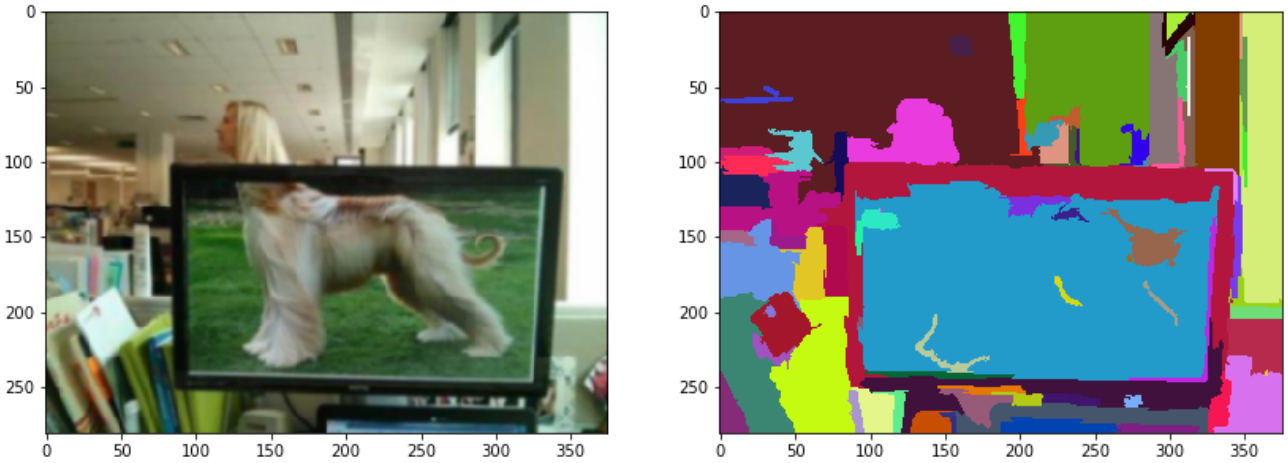


Image: photos-29-18_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

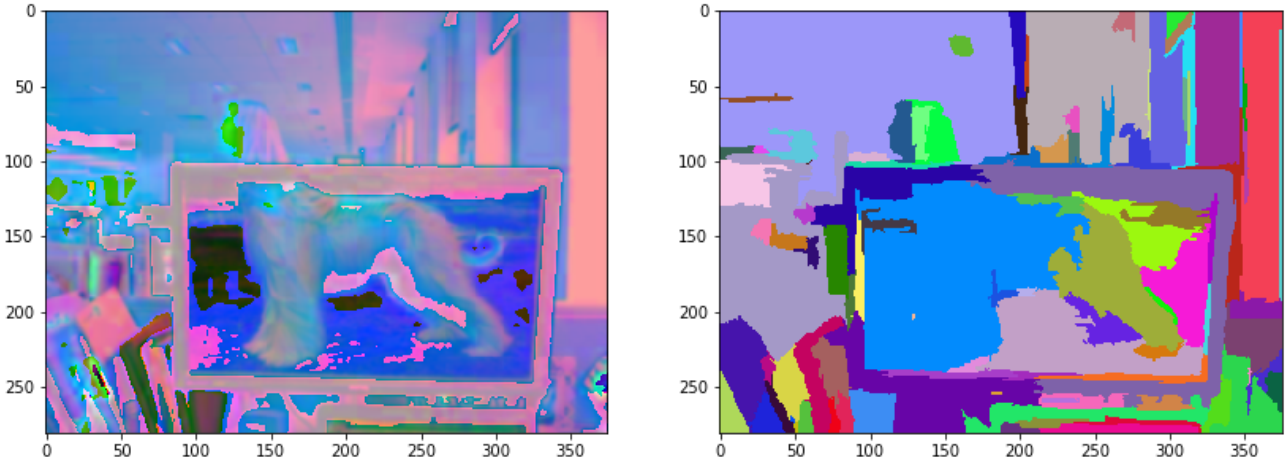


Image: photos-29-18_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

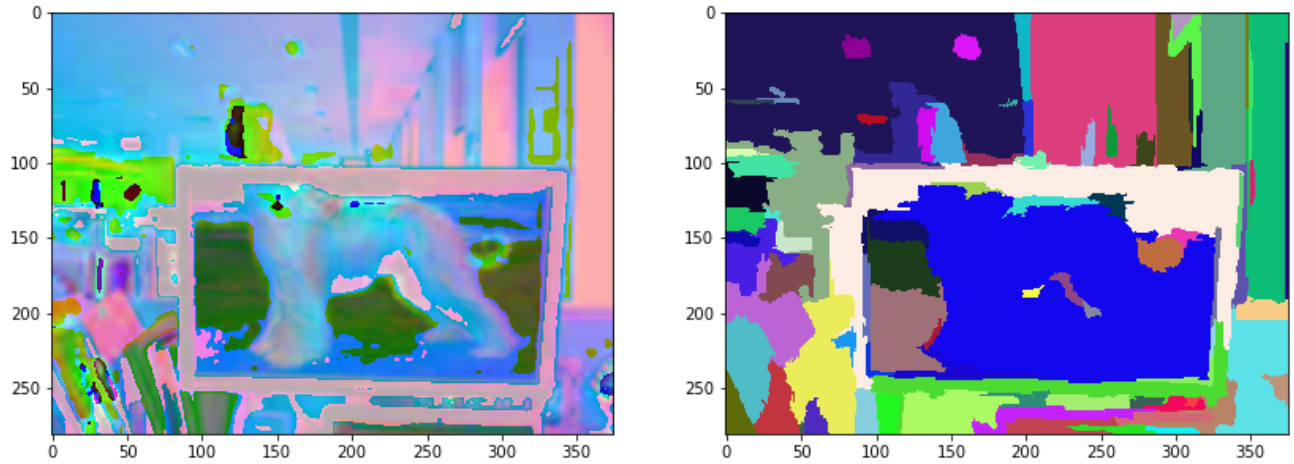


Image: photos-29-18_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

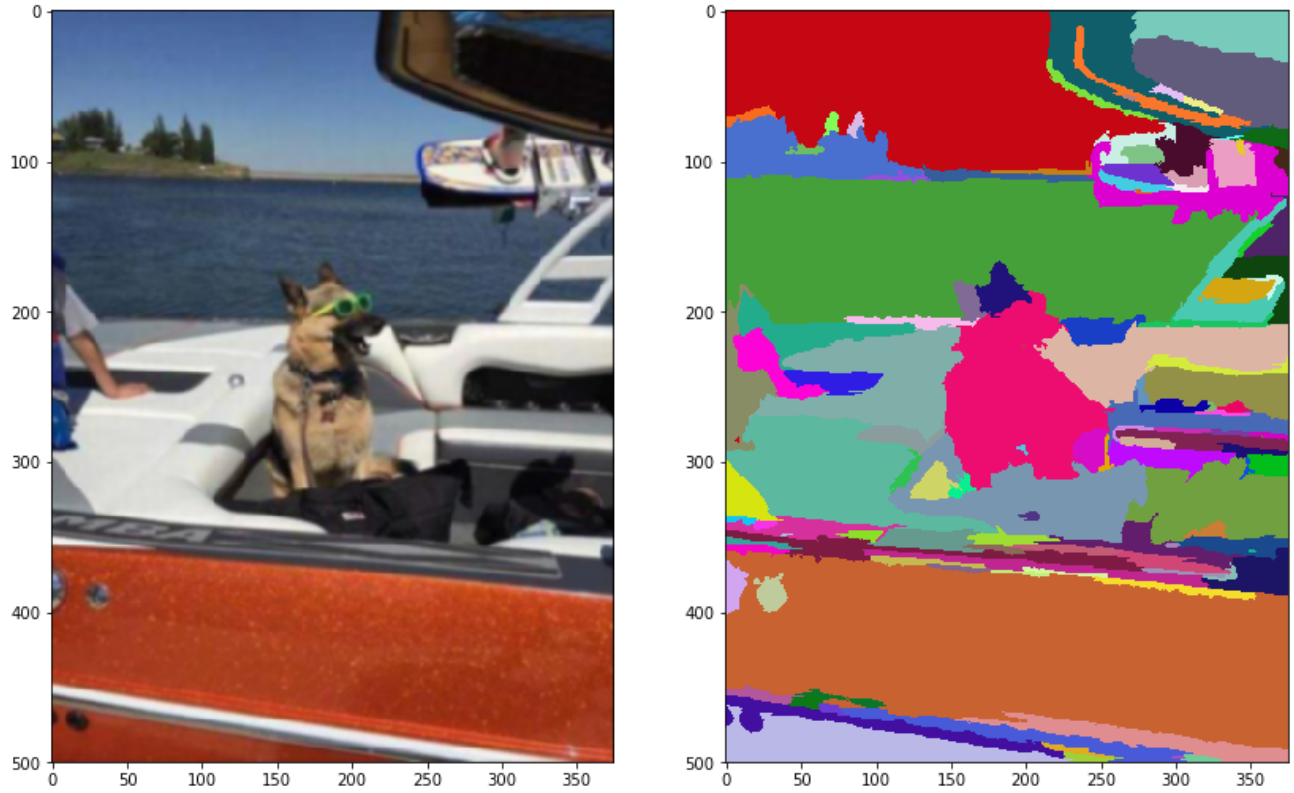


Image: photos-33-17_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

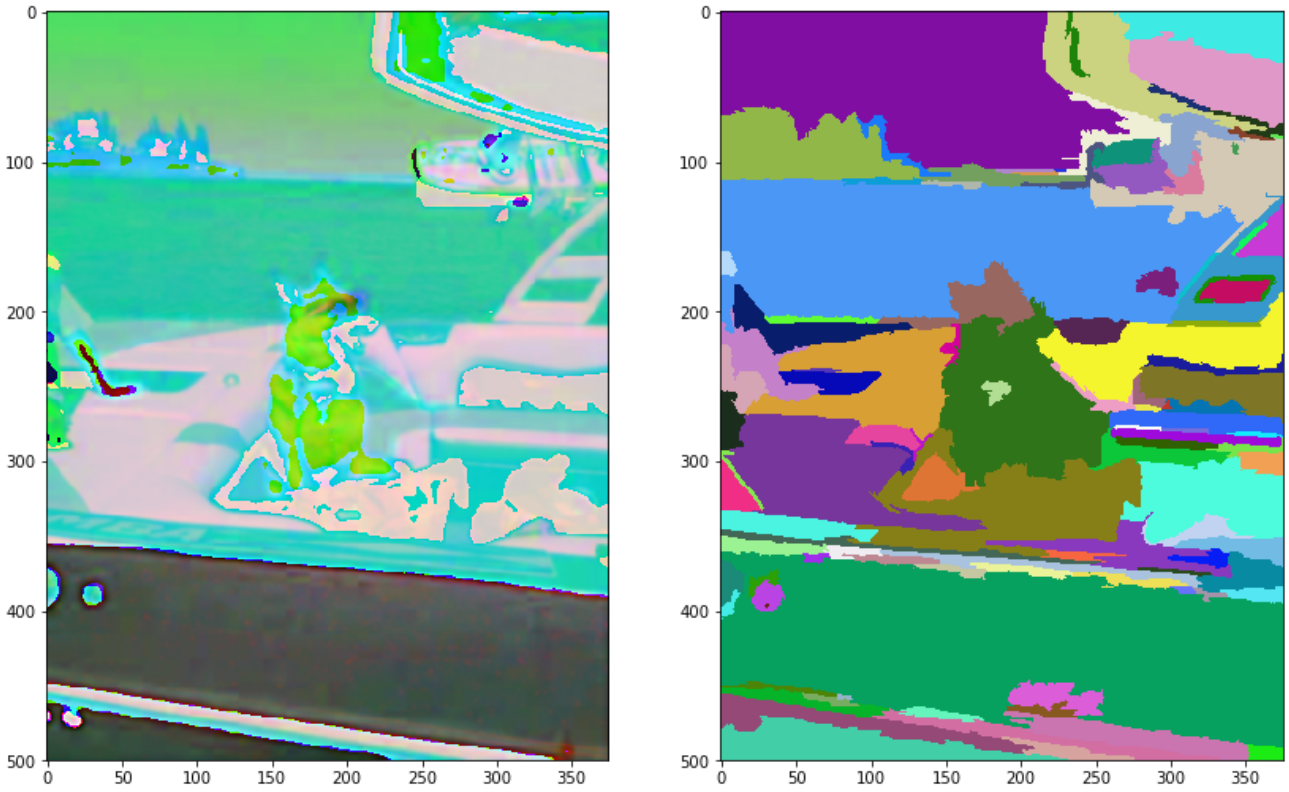


Image: photos-33-17_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

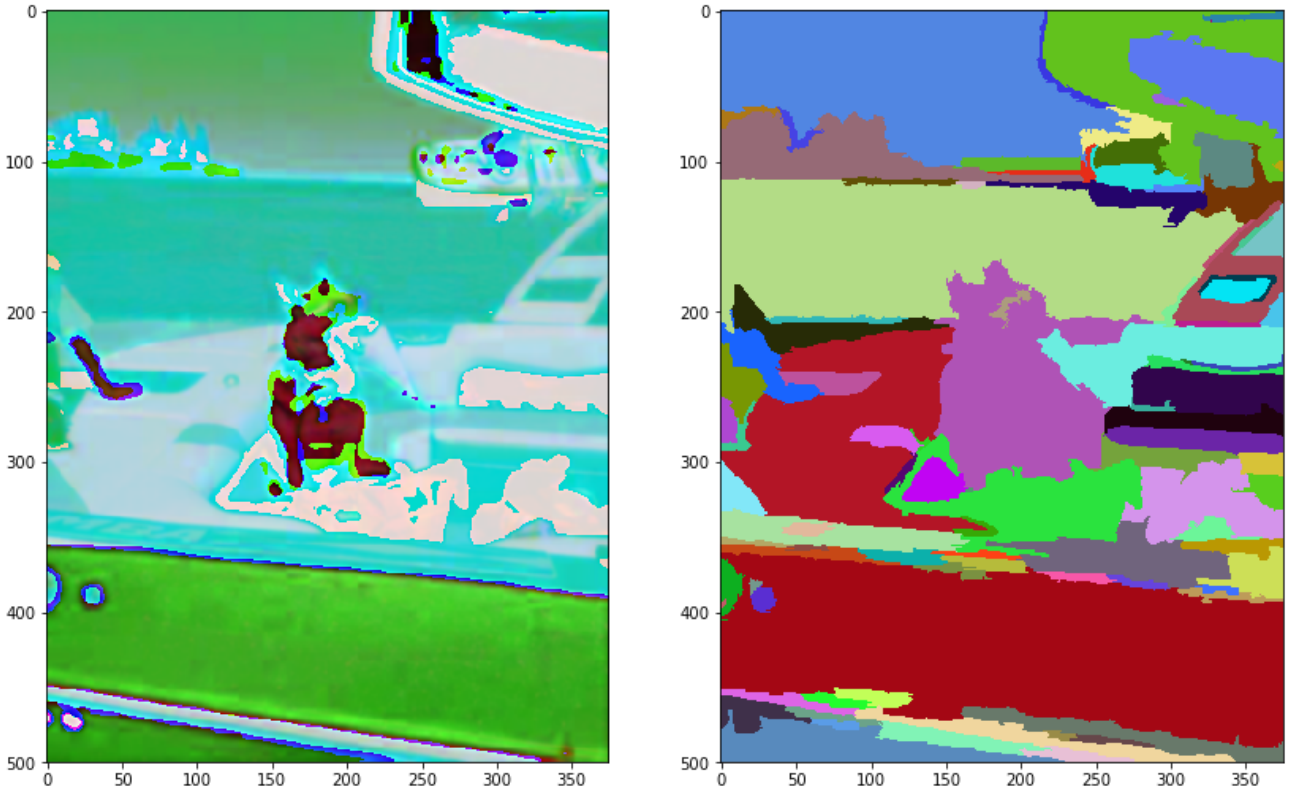


Image: photos-33-17_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

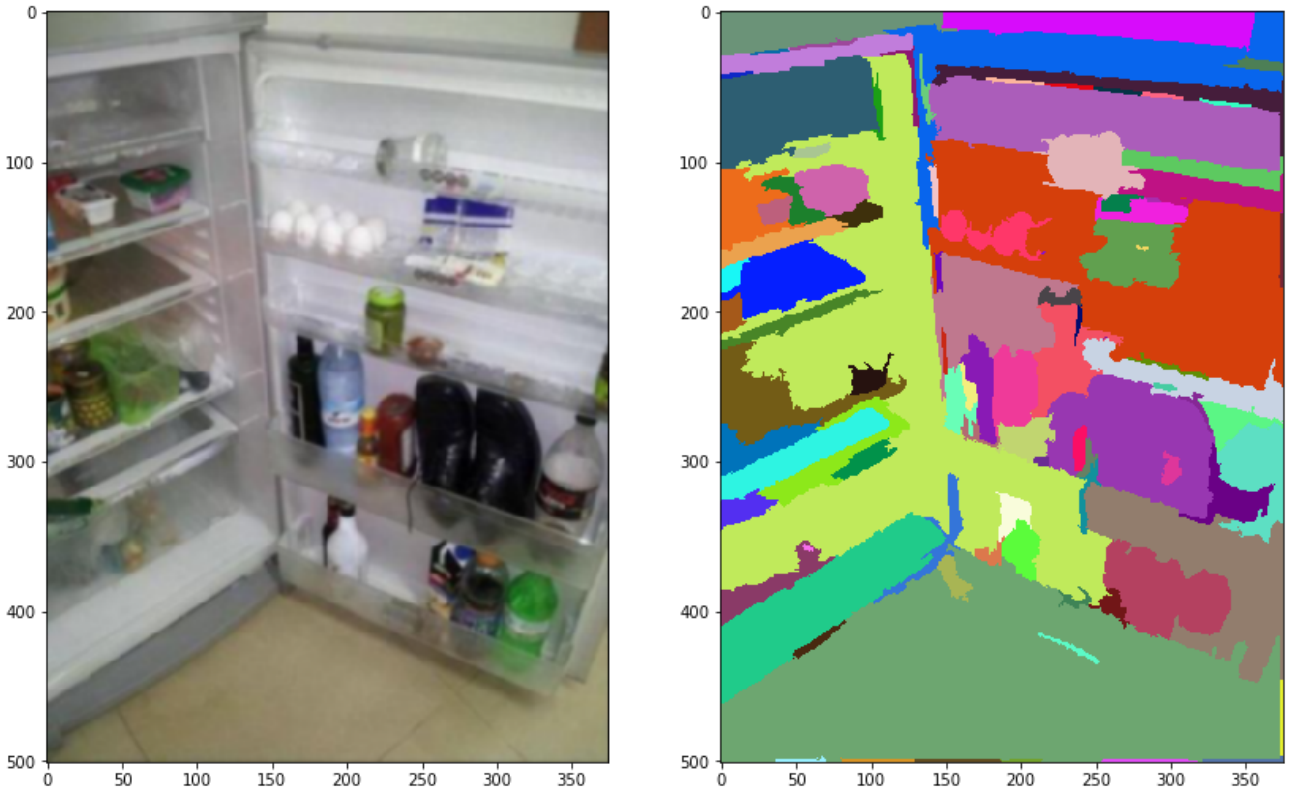


Image: photos-35-14_05.jpg ; Color Space: RGB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

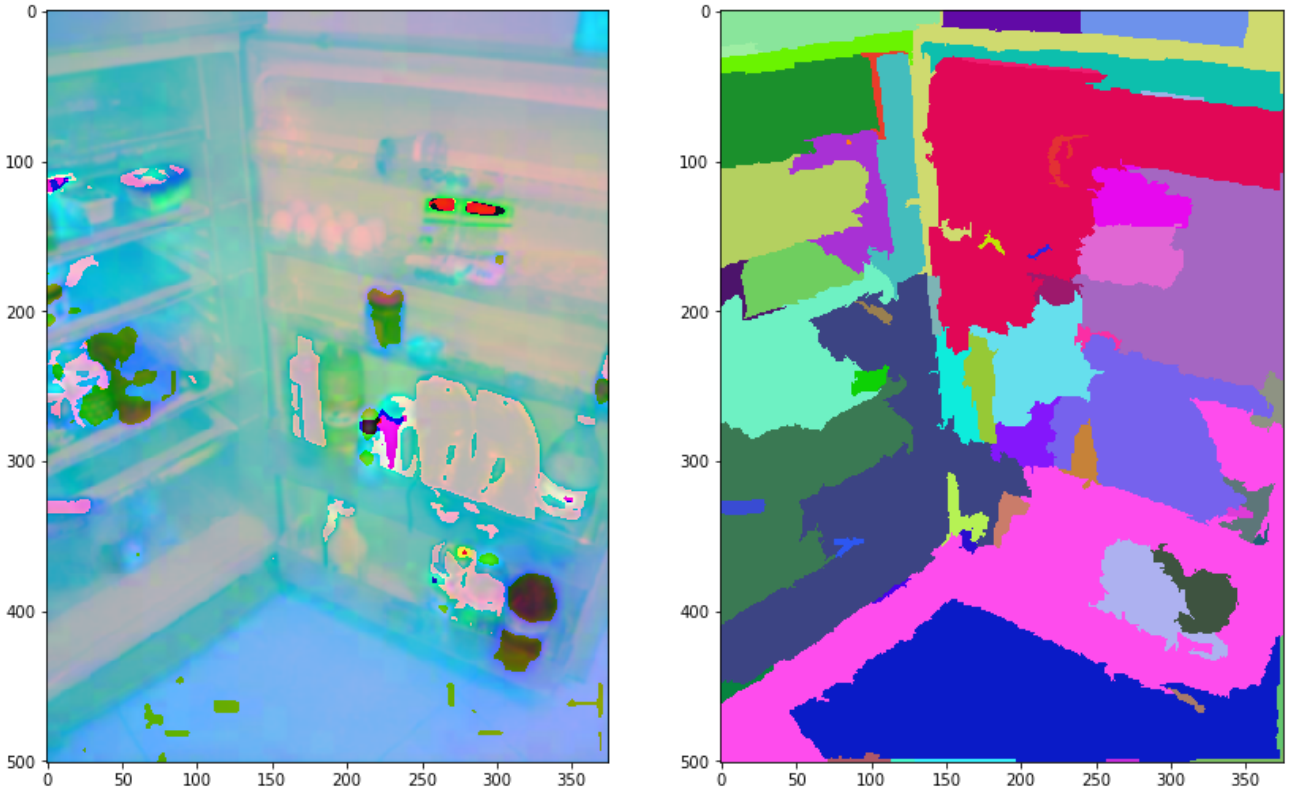


Image: photos-35-14_05.jpg ; Color Space: CIE LAB ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

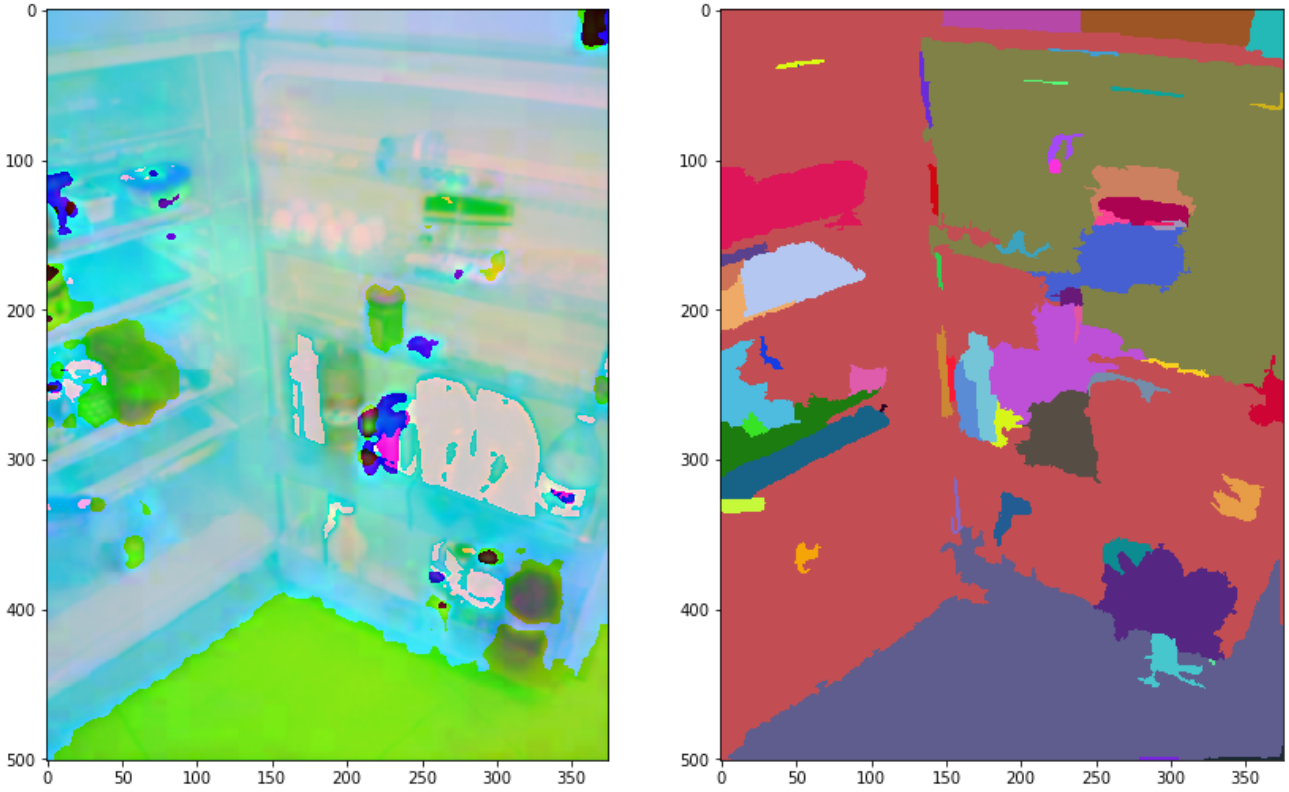


Image: photos-35-14_05.jpg ; Color Space: CIE LUV ; Sigma: 1 ; k: 0.8 ; Minimum Size: 400

Conclusiones

Apreciaciones del Paper

En primera instancia se analizarán algunas conclusiones y extractos sobre el paper que se consideran de interés.

La investigación parte con la necesidad de tener algoritmos eficientes para segmentación de imágenes. Algunas de las aplicaciones que se mencionan son los problemas de dificultad media como reconocimiento estéreo, en donde son necesarias definir *region of support* para las operaciones de correspondencia. Dichas regiones pueden ser encontradas con algoritmos de segmentación. Los problemas de alto nivel (ej.: reconocimiento en contextos) también pueden beneficiarse con algoritmos que sirvan para diferenciar el fondo de los objectos y el reconocimiento de las distintas partes.

Según el autor, un algoritmo de segmentación debe ser capaz de:

1. *Capturar regiones o grupos de importancia perceptual que reflejan aspectos globales de la imagen*. Sobre esto, dos problemas principales son poder caracterizar aquello que es importante de la imagen y especificar que es lo que hace una determinada técnica de segmentación.
2. *Ser eficiente, funcionar en tiempo linear sobre el número de pixeles de una imágen*. Los métodos de segmentación deben funcionar a la par con los métodos de detección de borde u otros algoritmos de bajo nivel.

En la época en que se escribió esta investigación, se analizaron otras técnicas existentes. El documento explica como los procesos basados en vectores propios son muy lentos para aplicaciones prácticas. Adicionalmente se menciona que otros métodos, si eficientes, fallan en capturar las propiedades globales (no-locales) de la imágen que son perceptiblemente importantes.

Observaciones sobre la experiencia

El método estudiado en esta experiencia entrega algunos resultados destacables sobre la segmentación, sin embargo, los resultados se aprecian no como una segmentación *perfecta* como aquellas vistas en clase de Redes Neuronales Convolucionales, cuyos resultados son mucho mejores.

Los resultados entregados permiten visualizar la variación con respecto a la modificación individual de cada parámetro. Se puede observar que considerar valores bajos en cualquiera de los parámetros entrega resultados no deseados.

Sobre los parámetros con los que se aplicó la técnica, se puede apreciar una clara mejora al aplicar filtro Gaussiano, se utilizó para los resultados globales un sigma de 1, un k de 0.7 y un minimumSize de 400. Estos son similares a los entregados por el autor. La principal diferencia, del minimumSize se encuentra en que las imágenes utilizadas en esta experiencia son mayores (~400px) que las del paper (~320 y ~160).

Las imágenes utilizadas fueron extraidas desde el sitio Dump-A-Day - Random Pictures (www.dumpaday.com/category/random-pictures/), por lo que se considera que la experiencia utilizó una muestra de datos reales y no un set preparado específicamente para esta experiencia. Lo anterior implica que los contrastes, calidad, diversidad, aislamiento de objetos u otros factores no fueron considerados.

Una gran variación perceptible en los resultados se da por la aplicación de distintos espacios de color. Si bien se solicitaron resultados en dos espacios de color (RGB y CIE LAB), se trabajó con varios. Esto permite tener una percepción mayor sobre los cambios que proporcionan los espacios de color y las ventajas sobre la aplicación del algoritmo.

Otra conclusión importante, es que al igual que en la experiencia anterior, en esta se trabajó con iPyWidgets, lo que permite estudiar y analizar las imágenes en tiempo real, lo que permite iterar mucho más rápido sobre los resultados y obtener mejores conclusiones en menos tiempo. La demostración con Widgets contiene 14 imágenes y 6 espacios de color: RGB, CIE LAB, CIE LUV, HSV, RGB CIE y RGB XYZ.

Bibliografía

1. Efficient Graph-Based Image Segmentation; P. Felzenszwalb, D. Huttenlocher; International Journal of Computer Vision, Vol. 59, No. 2, September 2004; http://cs.brown.edu/~pff/papers/seg-ijcv.pdf (http://cs.brown.edu/~pff/papers/seg-ijcv.pdf)
2. Efficient Graph-Based Image Segmentation C++ Implementation; P. Felzenszwalb, D. Huttenlocher; http://cs.brown.edu/~pff/segment/segment.zip (http://cs.brown.edu/~pff/segment/segment.zip)
3. Efficient Graph-Based Image Segmentation; Ranjay; Stanford; http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ranjay_pres.pdf (http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ranjay_pres.pdf)
4. An implementation of Efficient Graph-Based Image Segmentation; Sahil Narang, Kishore Rathinavel, University of North Carolina, Chapel Hill; http://www.cs.unc.edu/~sahil/data/Segmentation-Report.pdf (http://www.cs.unc.edu/~sahil/data/Segmentation-Report.pdf)
5. UNION FIND DATA STRUCTURE (PYTHON RECIPE); Josiah Carlson; http://code.activestate.com/recipes/215912/ (http://code.activestate.com/recipes/215912/)
6. Algoritmo de Kruskal; Wikipedia; https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal (https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal)
7. Union Find Algorithm; D. Eppstein; https://www.ics.uci.edu/~eppstein/PADS/UnionFind.py (https://www.ics.uci.edu/~eppstein/PADS/UnionFind.py)
8. Minimum Spanning Tree; D. Eppstein; https://www.ics.uci.edu/~eppstein/PADS/MinimumSpanningTree.py (https://www.ics.uci.edu/~eppstein/PADS/MinimumSpanningTree.py)
9. Graphs, graph algorithms (for image segmentation); Václav Hlaváč; Czech Technical University in Prague; http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/15ImageAnalysis/32-08aGraphsForImageSegmentation.pdf (http://people.ciirc.cvut.cz/~hlavac/TeachPresEn/15ImageAnalysis/32-08aGraphsForImageSegmentation.pdf)
10. Image Processing using Graphs; Filip Malmberg; http://www.cb.uu.se/~filip/ImageProcessingUsingGraphs/LectureNotes/Lecture1.pdf (http://www.cb.uu.se/~filip/ImageProcessingUsingGraphs/LectureNotes/Lecture1.pdf)