Computer Vision Assignment 3

March 18, 2020

Name : Meher Shashwat Nigam

Roll Number : 20171062

Drive Link: https://drive.google.com/open?id=1IJSrAB07Up2cPwIhUfjgz1M9K6sCl59U

[1]:
```
import glob
import sys
import numpy as np
import cv2 as cv
import igraph as ig
from sklearn.cluster import KMeans
```

Setting up basic flags and global variables

[2]:
```
BLUE = [255, 0, 0]
DRAW_BG = 0
DRAW_FG = 1
DRAW_PR_BG = 2
DRAW_PR_FG = 3

rect = (0, 0, 1, 1)
rectangle = False
rect_over = False
```

Defining the GMM function

[3]:
```
class GaussianMixture:
    def __init__(self, X, n_components=5):
        self.n_components = n_components
        self.n_features = X.shape[1]
        self.n_samples = np.zeros(self.n_components)

        self.coefs = np.zeros(self.n_components)
        self.means = np.zeros((self.n_components, self.n_features))
        # Full covariance
        self.covariances = np.zeros(
            (self.n_components, self.n_features, self.n_features))
        self.init_with_kmeans(X)
```

```
    def init_with_kmeans(self, X):
        label = KMeans(n_clusters=self.n_components, n_init=1).fit(X).labels_
        self.fit(X, label)

    def calc_score(self, X, ci):
        score = np.zeros(X.shape[0])
        if self.coefs[ci] > 0:
            diff = X - self.means[ci]
            mult = np.einsum(
                'ij,ij->i', diff, np.dot(np.linalg.inv(self.covariances[ci]),␣
↪diff.T).T)
            score = np.exp(-.5 * mult) / np.sqrt(2 * np.pi) / np.sqrt(np.linalg.
↪det(self.covariances[ci]))
        return score

    def calc_prob(self, X):
        prob = [self.calc_score(X, ci) for ci in range(self.n_components)]
        return np.dot(self.coefs, prob)

    def which_component(self, X):
        prob = np.array([self.calc_score(X, ci) for ci in range(self.
↪n_components)]).T
        return np.argmax(prob, axis=1)

    def fit(self, X, labels):
        assert self.n_features == X.shape[1]
        self.n_samples[:] = 0
        self.coefs[:] = 0
        uni_labels, count = np.unique(labels, return_counts=True)
        self.n_samples[uni_labels] = count
        variance = 0.01
        for ci in uni_labels:
            n = self.n_samples[ci]
            self.coefs[ci] = n / np.sum(self.n_samples)
            self.means[ci] = np.mean(X[ci == labels], axis=0)
            self.covariances[ci] = 0 if self.n_samples[ci] <= 1 else np.cov(
                X[ci == labels].T)
            det = np.linalg.det(self.covariances[ci])
            if det <= 0:
                # Adds the white noise to avoid singular covariance matrix.
                self.covariances[ci] += np.eye(self.n_features) * variance
                det = np.linalg.det(self.covariances[ci])
```

1 Implementing the grabcut algorithm (Section 3.1,3.2)

Allows options for changing : - Number of iterations of running the algorithm - gamma value - Number of gmm components - Considering 8-connectivity or 4-connectivity in the image

[4]:
```
class GrabCut:
    def __init__(self, img, mask, rect=None, gmm_components=5,n_iters =␣
↪1,gamma=50,conn8=False):
        self.img = np.asarray(img, dtype=np.float64)
        self.rows, self.cols, _ = img.shape
        self.niters = n_iters
        self.mask = mask
        self.conn8 = conn8

        if rect is not None:
            self.mask[rect[1]:rect[1] + rect[3],rect[0]:rect[0] + rect[2]] =␣
↪DRAW_PR_FG
        self.classify_pixels()

        # Best number of GMM components K suggested in paper
        self.gmm_components = gmm_components
        self.gamma = gamma  # Best gamma suggested in paper formula = 50(5)
        self.beta = 0

        self.left_V = np.empty((self.rows, self.cols - 1))
        if self.conn8:
            self.upleft_V = np.empty((self.rows - 1, self.cols - 1))
            self.upright_V = np.empty((self.rows - 1, self.cols - 1))
        self.up_V = np.empty((self.rows - 1, self.cols))

        self.bgd_gmm = None
        self.fgd_gmm = None
        self.comp_idxs = np.empty((self.rows, self.cols), dtype=np.uint32)

        self.gc_graph = None
        self.gc_graph_capacity = None           # Edge capacities
        self.gc_source = self.cols * self.rows  # "object" terminal S
        self.gc_sink = self.gc_source + 1       # "background" terminal T

        self.calc_beta_smoothness()
        self.init_GMMs()
        self.run(self.niters)

    def calc_beta_smoothness(self):
        _left_diff = self.img[:, 1:] - self.img[:, :-1]
        _up_diff = self.img[1:, :] - self.img[:-1, :]
```

```
        if self.conn8:
            _upleft_diff = self.img[1:, 1:] - self.img[:-1, :-1]
            _upright_diff = self.img[1:, :-1] - self.img[:-1, 1:]
            self.beta = np.sum(np.square(_left_diff)) + np.sum(np.
↪square(_upleft_diff)) + np.sum(np.square(_up_diff))+np.sum(np.
↪square(_upright_diff))
            self.beta = 1 / (2 * self.beta / (
                  4 * self.cols * self.rows  # Each pixel has 4 neighbors␣
↪(left, upleft, up, upright)
                - 3 * self.cols # The 1st column doesn't have left, upleft and␣
↪the last column doesn't have upright
                - 3 * self.rows  # The first row doesn't have upleft, up and␣
↪upright
                + 2))  # The first and last pixels in the 1st row are removed␣
↪twice
            print('Beta:', self.beta)
        else:
            self.beta = np.sum(np.square(_left_diff)) +np.sum(np.
↪square(_up_diff))
            self.beta = 1 / (2 * self.beta / (
                  2 * self.cols * self.rows # Each pixel has 2 neighbors (left,␣
↪up)
                - 1 * self.cols  # The first column doesn't have left
                - 1 * self.rows))  # The first row doesn't have up
            print('Beta:', self.beta)

        # Smoothness term V described in formula (11)
        # axis=2 for summing up RGB pixels at each cell
        self.left_V = self.gamma * np.exp(-self.beta * np.sum(np.
↪square(_left_diff), axis=2))
        self.up_V = self.gamma * np.exp(-self.beta * np.sum(np.
↪square(_up_diff), axis=2))
        if self.conn8:
            self.upleft_V = self.gamma / np.sqrt(2) * np.exp(-self.beta * np.
↪sum(np.square(_upleft_diff), axis=2))
            self.upright_V = self.gamma / np.sqrt(2) * np.exp(-self.beta * np.
↪sum(np.square(_upright_diff), axis=2))

    def classify_pixels(self):
        self.bgd_indexes = np.where(np.logical_or(self.mask == DRAW_BG, self.
↪mask == DRAW_PR_BG))
        self.fgd_indexes = np.where(np.logical_or(self.mask == DRAW_FG, self.
↪mask == DRAW_PR_FG))

        print('(pr_)bgd count: %d, (pr_)fgd count: %d' % (self.bgd_indexes[0].
↪size, self.fgd_indexes[0].size))
```

```
    def init_GMMs(self):
        self.bgd_gmm = GaussianMixture(self.img[self.bgd_indexes], self.
↪gmm_components)
        self.fgd_gmm = GaussianMixture(self.img[self.fgd_indexes], self.
↪gmm_components)

    def assign_GMMs_components(self):
        self.comp_idxs[self.bgd_indexes] = self.bgd_gmm.which_component(self.
↪img[self.bgd_indexes])
        self.comp_idxs[self.fgd_indexes] = self.fgd_gmm.which_component(self.
↪img[self.fgd_indexes])

    def learn_GMMs(self):
        self.bgd_gmm.fit(self.img[self.bgd_indexes],self.comp_idxs[self.
↪bgd_indexes])
        self.fgd_gmm.fit(self.img[self.fgd_indexes],self.comp_idxs[self.
↪fgd_indexes])

    def construct_gc_graph(self):
        bgd_indexes = np.where(self.mask.reshape(-1) == DRAW_BG)
        fgd_indexes = np.where(self.mask.reshape(-1) == DRAW_FG)
        pr_indexes = np.where(np.logical_or(self.mask.reshape(-1) ==␣
↪DRAW_PR_BG, self.mask.reshape(-1) == DRAW_PR_FG))

        print('bgd count: %d, fgd count: %d, uncertain count: %d' %␣
↪(len(bgd_indexes[0]), len(fgd_indexes[0]), len(pr_indexes[0])))

        edges = []
        self.gc_graph_capacity = []

        # t-links
        edges.extend(list(zip([self.gc_source] * pr_indexes[0].size,␣
↪pr_indexes[0])))
        _D = -np.log(self.bgd_gmm.calc_prob(self.img.reshape(-1,␣
↪3)[pr_indexes]))
        self.gc_graph_capacity.extend(_D.tolist())
        assert len(edges) == len(self.gc_graph_capacity)

        edges.extend(list(zip([self.gc_sink] * pr_indexes[0].size,␣
↪pr_indexes[0])))
        _D = -np.log(self.fgd_gmm.calc_prob(self.img.reshape(-1,␣
↪3)[pr_indexes]))
        self.gc_graph_capacity.extend(_D.tolist())
        assert len(edges) == len(self.gc_graph_capacity)
```

```
        edges.extend(list(zip([self.gc_source] * bgd_indexes[0].size,␣
↪bgd_indexes[0])))
        self.gc_graph_capacity.extend([0] * bgd_indexes[0].size)
        assert len(edges) == len(self.gc_graph_capacity)

        edges.extend(list(zip([self.gc_sink] * bgd_indexes[0].size,␣
↪bgd_indexes[0])))
        self.gc_graph_capacity.extend([9 * self.gamma] * bgd_indexes[0].size)
        assert len(edges) == len(self.gc_graph_capacity)

        edges.extend(list(zip([self.gc_source] * fgd_indexes[0].size,␣
↪fgd_indexes[0])))
        self.gc_graph_capacity.extend([9 * self.gamma] * fgd_indexes[0].size)
        assert len(edges) == len(self.gc_graph_capacity)

        edges.extend(list(zip([self.gc_sink] * fgd_indexes[0].size,␣
↪fgd_indexes[0])))
        self.gc_graph_capacity.extend([0] * fgd_indexes[0].size)
        assert len(edges) == len(self.gc_graph_capacity)


        # n-links
        img_indexes = np.arange(self.rows * self.cols,dtype=np.uint32).
↪reshape(self.rows, self.cols)

        mask1 = img_indexes[:, 1:].reshape(-1)
        mask2 = img_indexes[:, :-1].reshape(-1)
        edges.extend(list(zip(mask1, mask2)))
        self.gc_graph_capacity.extend(self.left_V.reshape(-1).tolist())
        assert len(edges) == len(self.gc_graph_capacity)

        mask1 = img_indexes[1:, :].reshape(-1)
        mask2 = img_indexes[:-1, :].reshape(-1)
        edges.extend(list(zip(mask1, mask2)))
        self.gc_graph_capacity.extend(self.up_V.reshape(-1).tolist())
        assert len(edges) == len(self.gc_graph_capacity)

        if self.conn8:
            mask1 = img_indexes[1:, 1:].reshape(-1)
            mask2 = img_indexes[:-1, :-1].reshape(-1)
            edges.extend(list(zip(mask1, mask2)))
            self.gc_graph_capacity.extend(
                self.upleft_V.reshape(-1).tolist())
            assert len(edges) == len(self.gc_graph_capacity)

            mask1 = img_indexes[1:, :-1].reshape(-1)
            mask2 = img_indexes[:-1, 1:].reshape(-1)
```

```
            edges.extend(list(zip(mask1, mask2)))
            self.gc_graph_capacity.extend(self.upright_V.reshape(-1).tolist())
            assert len(edges) == len(self.gc_graph_capacity)

            assert len(edges) == 4 * self.cols * self.rows - 3 * (self.cols +␣
↪self.rows) + 2 + 2 * self.cols * self.rows

        else:
            assert len(edges) == 2 * self.cols * self.rows - 1 * (self.cols +␣
↪self.rows)  +  2 * self.cols * self.rows

        self.gc_graph = ig.Graph(self.cols * self.rows + 2)
        self.gc_graph.add_edges(edges)

    def estimate_segmentation(self):
        mincut = self.gc_graph.st_mincut(self.gc_source, self.gc_sink, self.
↪gc_graph_capacity)
        print('Foreground pixels: %d, Background pixels: %d' % (len(mincut.
↪partition[0]), len(mincut.partition[1])))
        pr_indexes = np.where(np.logical_or(self.mask == DRAW_PR_BG, self.mask␣
↪== DRAW_PR_FG))
        img_indexes = np.arange(self.rows * self.cols,dtype=np.uint32).
↪reshape(self.rows, self.cols)
        self.mask[pr_indexes] = np.where(np.isin(img_indexes[pr_indexes],␣
↪mincut.partition[0]), DRAW_PR_FG, DRAW_PR_BG)
        self.classify_pixels()

    def run(self, num_iters=1):
        print("Number of GMM components:",self.gmm_components)
        print("Number of iterations:", self.niters)
        print("Gamma Value:", self.gamma)
        for _ in range(num_iters):
            self.assign_GMMs_components()
            self.learn_GMMs()
            self.construct_gc_graph()
            self.estimate_segmentation()
```

2 GUI component (Section 3.3)

Handles user input: Making a rectangle using the mouse marking the probable foreground region in the image.

[5]:
```
def onmouse(event, x, y, flags, param):
    global img, img2, mask, rectangle, rect, rect_or_mask, ix, iy, rect_over
```

```
    if event == cv.EVENT_RBUTTONDOWN:
        rectangle = True
        ix, iy = x, y

    elif event == cv.EVENT_MOUSEMOVE:
        if rectangle == True:
            img = img2.copy()
            cv.rectangle(img, (ix, iy), (x, y), BLUE, 2)
            rect = (min(ix, x), min(iy, y), abs(ix-x), abs(iy-y))

    elif event == cv.EVENT_RBUTTONUP:
        rectangle = False
        rect_over = True
        cv.rectangle(img, (ix, iy), (x, y), BLUE, 2)
        rect = (min(ix, x), min(iy, y), abs(ix-x), abs(iy-y))
        print(" Now press the key 'n' a few times until no further change \n")
```

3 Code for running on sample images provided

Runs all the given test images and corresponding bounding boxes in loop according to given parameters and stores them in a specified output folder

[6]:
```
# Fixed bounding boxes

img_folder = './data/images'
bb_folder = './data/bboxes'
op_folder = './output_k10/'
gmm_comps = 5
num_iters = 1
gamma = 50
conn8 = True
files = glob.glob(img_folder+"/*.jpg")

for file in files:
    print()
    bbox = bb_folder+str("/")+file[14:-4]+str(".txt")
    file1 = open(bbox, "r")
    rect = tuple([int(x) for x in file1.read().strip().split(" ")])
    file1.close()

    rect2 = tuple([rect[0], rect[1], abs(rect[0]-rect[2]),␣
 ↪abs(rect[1]-rect[3])])

    img = cv.imread(file)
#     input_img = cv.cvtColor(img.copy(), cv.COLOR_BGR2YCrCb)
```

```
    input_img = img.copy()
    img_rect = img.copy()
    original = img.copy()                                    # a copy of original␣
↪image
    mask = np.zeros(img.shape[:2], dtype=np.uint8)  # mask initialized to BG
    output = np.zeros(img.shape, np.uint8)           # output image to be shown

    gc = GrabCut(input_img, mask, rect2, gmm_comps, num_iters, gamma, conn8)
    mask2 = np.where((mask == 1) + (mask == 3), 255, 0).astype('uint8')
    output = cv.bitwise_and(input_img, input_img, mask=mask2)
#     output = cv.cvtColor(cv.bitwise_and(input_img, input_img, mask=mask2),cv.
↪COLOR_YCrCb2BGR)
    bar = np.zeros((img.shape[0], 5, 3), np.uint8)

    img_rect = cv.rectangle(
        img_rect, tuple([rect[0], rect[1]]), tuple([rect[2], rect[3]]), BLUE, 3)

    res = np.hstack((original, bar, img_rect, bar, output))

    output_file = op_folder+file[14:-4]+"_k" + \
        str(gmm_comps)+"_iters"+str(num_iters) + \
        "_gamma"+str(gamma)+"_conn8"+str(conn8*1)+".png"
    cv.imwrite(output_file, res)
    print(" Result saved as image: "+output_file)
```

4 Interactive Image Segmentation using GrabCut algorithm.

At first, in input window, draw a rectangle around the object using mouse right button. Then press 'n' to segment the object (once or a few times)

Key 'n' - To update the segmentation

Key 'r' - To reset the setup

Key 's' - To save the results

[7]:
```
## GUI based grabcut

## Change input filename here:
file = './data/images/bush.jpg'

img = cv.imread(file)
img2 = img.copy()
original = img.copy()                                    # a copy of original image
mask = np.zeros(img.shape[:2], dtype=np.uint8)  # mask initialized to PR_BG
output = np.zeros(img.shape, np.uint8)           # output image to be shown
```

```
## Set your input parameters here:
gmm_comps = 5
num_iters = 1
gamma = 50
conn8 = False

cv.namedWindow('output')
cv.namedWindow('input')
cv.setMouseCallback('input', onmouse)
cv.moveWindow('input', img.shape[1]+10, 90)

print(" Instructions: \n")
print(" Draw a rectangle around the object using right mouse button \n")

while(1):
    cv.imshow('output', output)
    cv.imshow('input', img)
    k = cv.waitKey(1)
    if k == 27:         # esc to exit
        break
    elif k == ord('s'):  # save image
        bar = np.zeros((img.shape[0], 5, 3), np.uint8)
        res = np.hstack((img2, bar, img, bar, output))
        cv.imwrite('grabcut_output.png', res)
        print(" Result saved as image \n")
    elif k == ord('n'):  # segment the image
        print(rect)
        gc = GrabCut(img2, mask, rect, gmm_comps, num_iters, gamma, conn8)

    mask2 = np.where((mask == 1) + (mask == 3), 255, 0).astype('uint8')
    output = cv.bitwise_and(img2, img2, mask=mask2)
cv.destroyAllWindows()
```

5 Report

Study various parameters and how they affect segmentation

5.1 1. The number of iterations of GMM updating and energy minimization.

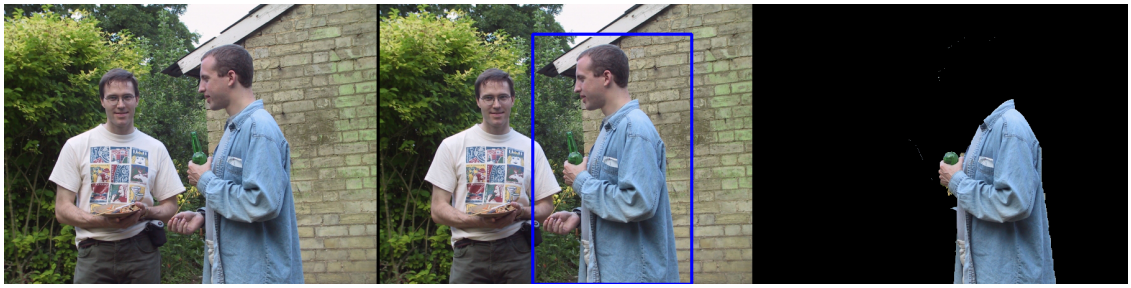
As the number of iterations increase, the foreground extracted becomes better and borders get neater. This is because GMM parameters get fine-tuned in successive iterations, and energy gets minimized further.‘

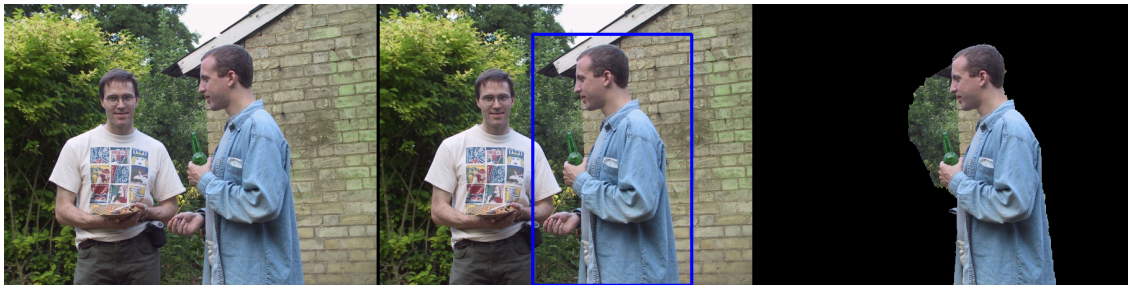
Using 1 iteration 

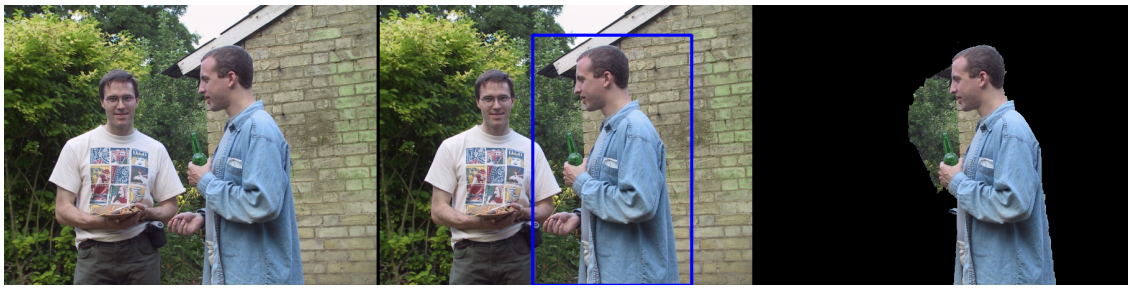
Using 5 iterations 

5.2 2. The number of mixture components used for GMM.

Using the appropriate number of gmm components is crucial to the working of the algorithm. The paper suggests 5 GMM components to be used. Changing the number of components shows bad results for fewer components(tested for 3) and similar results for 5,10 GMM components. The number of components used must be able to capture the information in the foreground/background adequately

K=3 

K=5 

K=10 

5.2.1 3. 4-connectivity or 8-connectivity

Results by using 8 connectivity are far better than using 4 conectivity as the former includes more information from its neighbours and hence gives smoother, better results

8 connectivity 

4 connectivity 

5.2.2 4. Gamma Value

According to the paper the gamma value determines the weight to be given to the smoothness term V while calculating the total energy. Lower values of gamma relate to more smoothness and hence a lot of background region being wrongly identified as foreground, but increasing it a lot can lead to some parts of foreground being left out. The paper suggests an optimal value of 50 found to work the best from experiments.

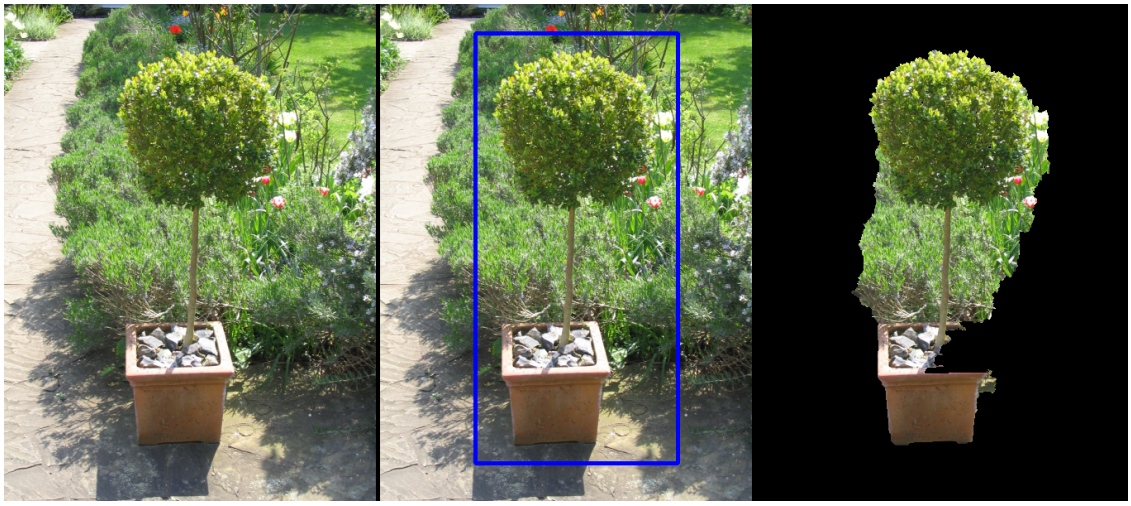
Gamma = 10 

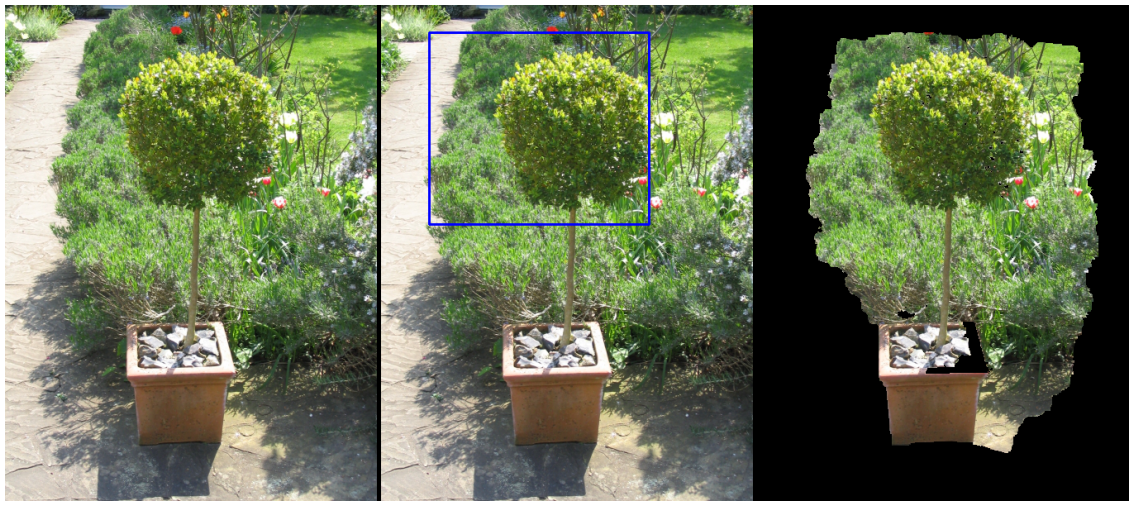
Gamma = 50

Gamma = 100

5.3 5. Tight or loose initial bounding box

A good initial bounding box helps learn good GMM parameters for background and foreground. A loose bounding box will put a lot of background information in the foreground GMM and leave out information in the background GMM.

Tight bounding box 

Loose bounding box 

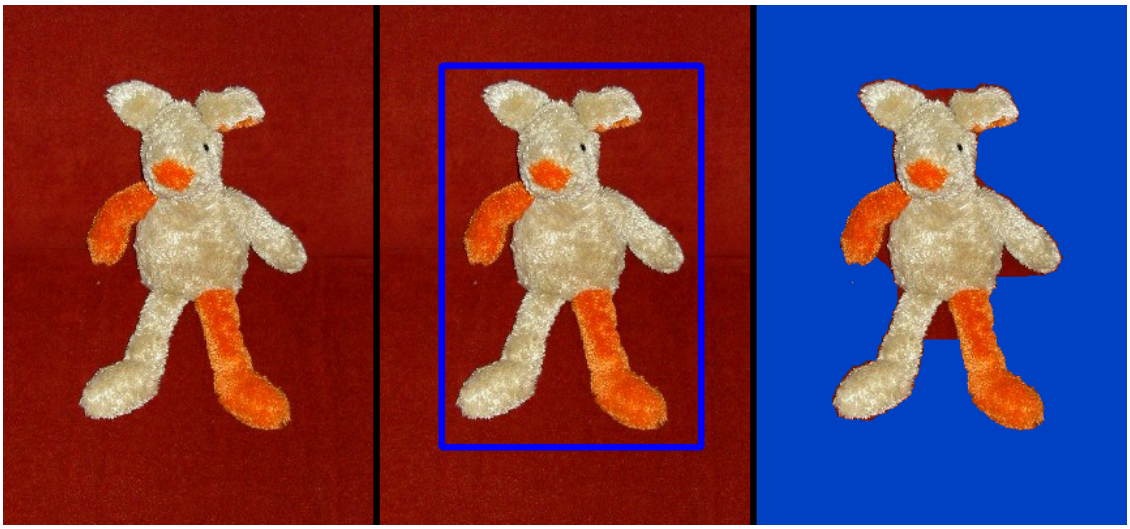
5.4 6. Different colour spaces

Colour spaces used: - RGB - LAB - HSV - YCrCb

There was no particular advantage observed while using a different colour space.

RGB

LAB



HSV

YCrCb