0. Import Dependencies

In [1]:

```
from PIL import Image
import matplotlib.pyplot as plt
import torch
from torchvision import models
import numpy as np
import pandas as pd
import torchvision.transforms as T
from PIL import Image
import cv2
import sys
```

3. Semantic segmentation

Read paper Fully Convolutional Networks for Semantic Segmentation (https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf). From the class drive download script segmentation.py, containing example how to use pretrained models for semantic segmentation.

main references:

```
 https://github.com/shelhamer/fcn.berkeleyvision.org

https://github.com/mozhuqing/fcn/blob/f6213010926c561b3e4f35eb5ee59b7ccc6390ec/data_augmenta
.ipynb

https://github.com/jedichien/ssd_keras/blob/b3008773d833501da1b7a21e8c37e6640e229d5f/T3_Assi
oxes.ipynb
 https://github.com/zplab-
dev/Nicolette/blob/398be5b8a306972dfc4d530687b1a8ae290bcae5/image_analysis/analyze_images.py
```

a) Briefly explain:

```
 i) How FCN upsamples predictions to match the original image size.
 ii) How had the authors improved the coarse predictions produced by the deepest layer.
```

Answer:

```
 i) Different from classic CNNs that use fully connected layer after convolution to obtain
fixed-length feature vectors for classification, FCN can accept input images of any size. U
psampling the feature map of the last convolutional layer using a deconvolution layer to
revert to original size, so that prediction can be generated by pixels and retain the
spatial information of input image. Finally, pixel-by-pixel classification is performed on
the upsampled feature map.
 However, upsampling cannot guarantee that the final pixelized prediction will be exactly t
he same as the original one. So, to ensure that the highest level of prediction, there will
be a crop layer after the upsampling layer that crops the output results.

 ii) The author discards the final classifier layer and converts all fully connected layers
into convolutions and appends a 1×1 convolution to channel dimension 21 to roughly predict
the rough result of each class. And he uses a deconvolution layer to perform coarse
prediction and bilinear upsampling. Since the finer the prediction requires fewer layers, s
o he combined the semantic information from deeper coarse layers with the fine appearance i
nformation from shallower layers to produce accurate and detailed segmentation.
```

In [2]:

```python
def decode_segmap(image, nc=21):
    label_colors = np.array([(0, 0, 0),  # 0=background
                # 1=aeroplane, 2=bicycle, 3=bird, 4=boat, 5=bottle
                (128, 0, 0), (0, 128, 0), (128, 128, 0), (0, 0, 128), (128, 0, 128),
                # 6=bus, 7=car, 8=cat, 9=chair, 10=cow
                (0, 128, 128), (128, 128, 128), (64, 0, 0), (192, 0, 0), (64, 128, 0),
                # 11=dining table, 12=dog, 13=horse, 14=motorbike, 15=person
                (192, 128, 0), (64, 0, 128), (192, 0, 128), (64, 128, 128), (192, 128, 128),
                # 16=potted plant, 17=sheep, 18=sofa, 19=train, 20=tv/monitor
                (0, 64, 0), (128, 64, 0), (0, 192, 0), (128, 192, 0), (0, 64, 128)])
    r = np.zeros_like(image).astype(np.uint8)
    g = np.zeros_like(image).astype(np.uint8)
    b = np.zeros_like(image).astype(np.uint8)

    for l in range(0, nc):
        idx = image == l
        r[idx] = label_colors[l, 0]
        g[idx] = label_colors[l, 1]
        b[idx] = label_colors[l, 2]

    rgb = np.stack([r, g, b], axis=2)
    return rgb
```

In [3]:

```python
def apply_mask(im, im_pred):
    """
    Overlays the predicted class labels onto an image using the alpha channel.
    This function assumes that the background label is the black color.
    This function is provided as an inspiration for the masking function you should write.
    """
    r_channel, g_channel, b_channel = cv2.split(im_pred)
    alpha_channel = 127 * np.ones(b_channel.shape, dtype=b_channel.dtype)
    # Make background pixels fully transparent
    alpha_channel -= 127 * np.all(im_pred == np.array([0, 0, 0]), axis=2).astype(b_channel.dtype)
    im_pred = cv2.merge((r_channel, g_channel, b_channel, alpha_channel))
    mask = Image.fromarray(im_pred, mode='RGBA')
    masked_img = Image.fromarray(im)
    masked_img.paste(mask, box=None, mask=mask)
    return np.array(masked_img)
```

In [4]:

```python
# define the model
fcn = models.segmentation.fcn_resnet101(pretrained=True).eval()
```

b) Download YourEmail.png and take a class index assigned to you from classes.csv. Modify segmentation.py so that you predict segmentation mask of this class (by FCN model) on the image given to you and highlight prediction via red mask blended with the original image. Submit this image as YourEmail_predicted.png.

In [5]:

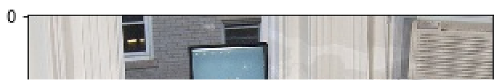
```python
csv=pd.read_csv('classes.csv',header=18)
csv.head(0)
#9=chair
```

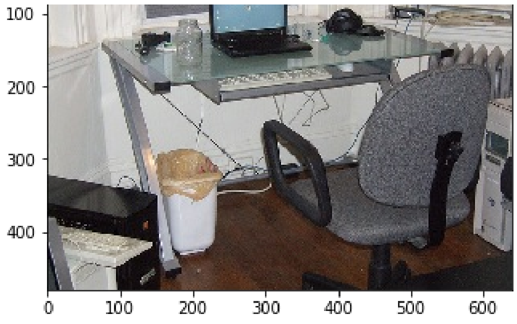
Out[5]:

XZHANG6@TCD.IE	9

In [6]:

```python
# load an image
img = Image.open('./XZHANG6@TCD.IE.png')
plt.imshow(img)
plt.show()
```





In [7]:

```
# transform the image
trf = T.Compose([T.ToTensor(),
                 T.Normalize(mean = [0.485, 0.456, 0.406],
                             std = [0.229, 0.224, 0.225])])
inp = trf(img).unsqueeze(0)
```

In [8]:

```
# pass the input through the net
out = fcn(inp)['out']
print (out.shape)
```

```
torch.Size([1, 21, 480, 640])
```

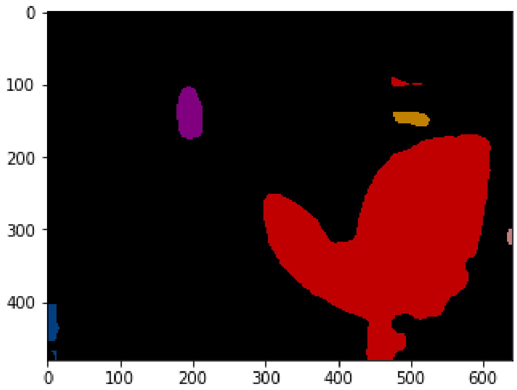
In [9]:

```
# calculate labels
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()
print (np.unique(om))
```

```
[ 0  5  9 11 15 20]
```

In [10]:

```
# show segmentation output
rgb = decode_segmap(om)
plt.imshow(rgb)
plt.show()
```



In [11]:

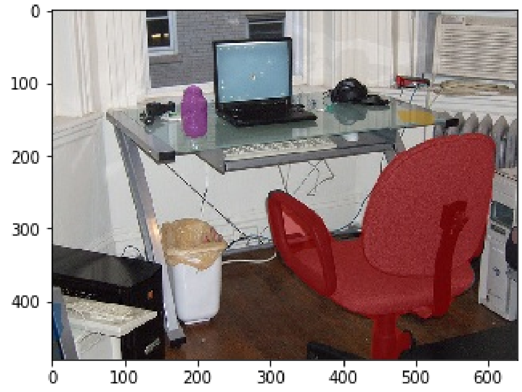
```
# show red mask blended with the original image output
img = cv2.imread ('XZHANG6@TCD.IE.png', cv2.IMREAD_COLOR)
rgb1 = rgb[:,:,::-1]
mask = apply_mask(img, rgb1)
cv2.imwrite("XZHANG6@TCD.IE_predicted.png",mask)
```

Out[11]:

```
True
```

In [12]:

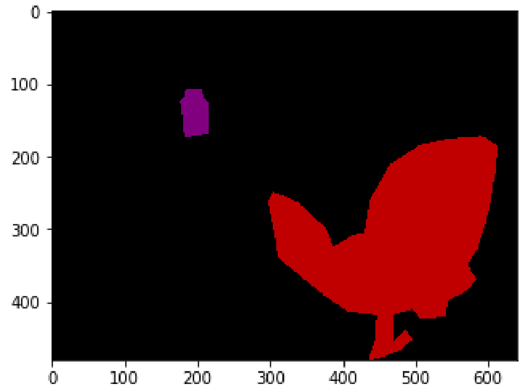
```python
merge = cv2.imread ('XZHANG6@TCD.IE_predicted.png', cv2.IMREAD_COLOR)
merge = merge[:,:,::-1]
plt.imshow(merge)
plt.show()
```



c) Decode the groundtruth image YourEmail_mask.png and calculate intersection over union (IOU) with the prediction for the class assigned to you. Report IOU in file YourEmail_iou.csv.

In [13]:

```python
# load an mask
ground = Image.open('./XZHANG6@TCD.IE_mask.png')
plt.imshow(ground);
plt.show()
```



In [14]:

```python
def calculate_iou(prediction, ground_truth, plot_val=False, save_val=False, save_dir=None, pyr=False):
    """Find intersection over union for 2 images
    Prediction and ground_truth are boolean arrays
    """
    #intersect and union
    intersect = (prediction & ground_truth)
    union = (prediction|ground_truth)
    IoU=intersect.sum()/union.sum()

    return intersect.sum()/union.sum()
```

In [15]:

```python
calculate_iou(rgb, ground)
```

Out[15]:

```
0.8864342045382537
```

```
0.8864342045382537
```

In [16]:

```
final=pd.DataFrame(data={
        'IoU':calculate_iou(rgb, ground)},index=[0])
final.to_csv('XZHANG6@TCD.IE_iou.csv',header=None,index=None)
```