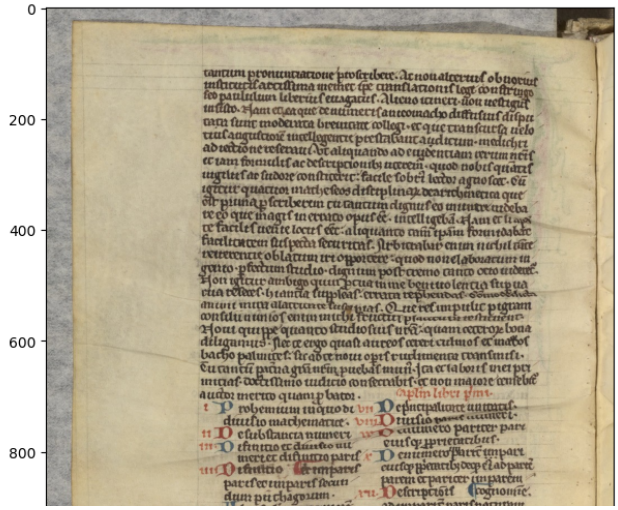
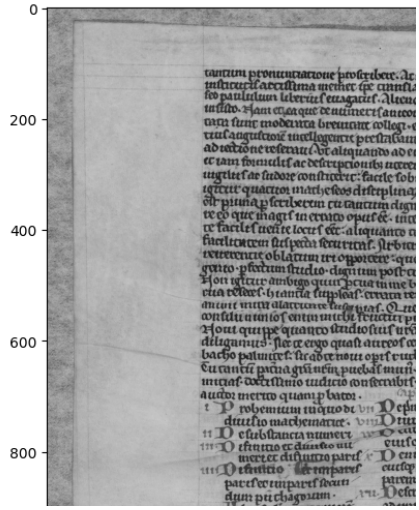
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
#from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error


# Read in an image
image = cv2.imread("../ImagesCodicologie/Corpus de jeu/Lebec/Arithmetica-f7v.jpeg")


# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
# normalize the image
norm_image = gray/255.0
#f_image = ft_image(norm_image)
# Display the images
f, (ax1,ax2) = plt.subplots(1, 2, figsize=(20,10))
ax1.imshow(image)
ax2.imshow(norm_image, cmap="gray")
#ax2.imshow(f_image, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x2afba9da490>
```

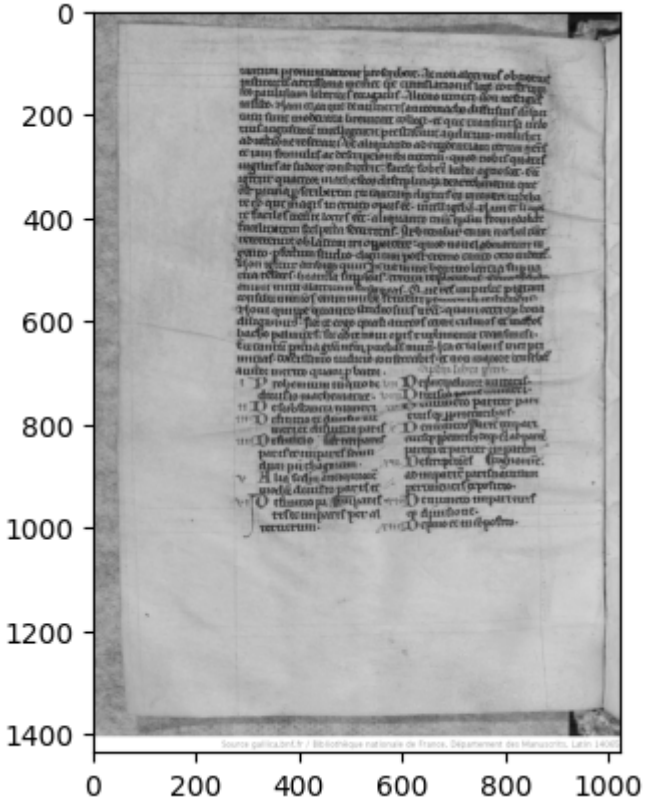




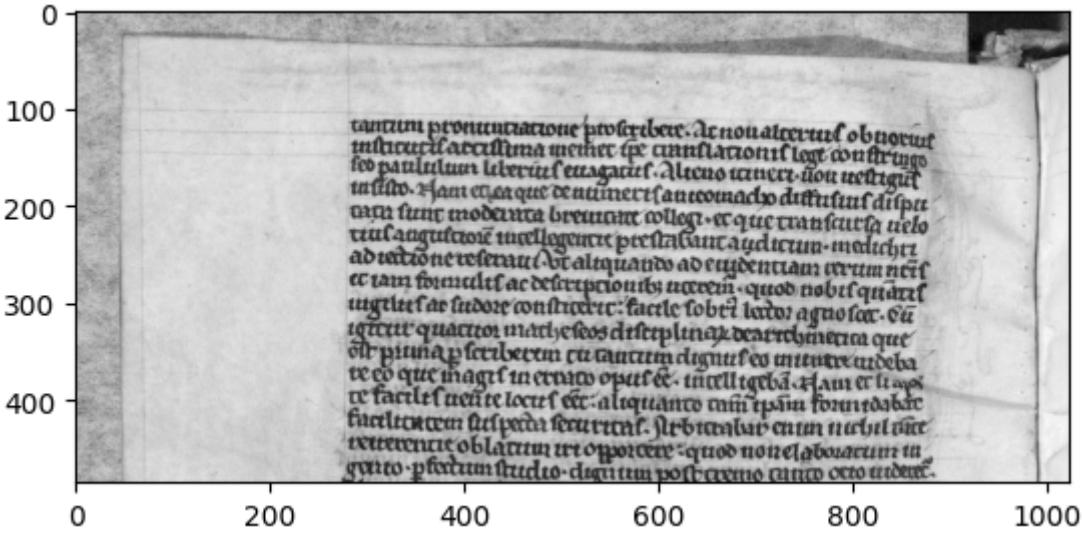
```
def ponderated_criterion(image,criterion,split,axis):
    """ - axis can only be 'x' or 'y'
        - image can only be in gray
        Return ponderated sum of a certain criterion on splitted image
    """
    if axis == 'x':
        left  = image[:,:split]
        right = image[:,split:]
        pond  = criterion(left,np.ones_like(left)+np.mean(left)-1) * left.size + criterion
        pond /= image.size
    else:
        left  = image[:split]
        right = image[split:]
        pond = criterion(left,np.ones_like(left)+np.mean(left)-1) * left.size + criterion(
        pond /= image.size
    return pond

def best_split(image,criterion,to='min',eps=1e-4):
    """ - normalized image only
        Return best split for an image under a certain criterion, considering both directi
    """
    all_splits = []

    for dir in [('x',len(image[0])),('y',len(image))]:
        axis,t_pos = dir
        if t_pos < 2: # if image is a line, splitting impossible
            all_splits.append((axis,None,1))
            continue
        t_hat = np.random.randint(1, t_pos)
        ponsum = ponderated_criterion(image,criterion,t_hat,axis)
        new_ponsum = ponderated_criterion(image,criterion,t_hat+1,axis) # consider going r
        i = 1
        #print(ponsum,new_ponsum)
        if new_ponsum > ponsum: # change direction if worse
            i *= -1
            new_ponsum = ponderated_criterion(image,criterion,t_hat-1,axis)
        #print(ponsum,new_ponsum)
        t_hat += i
        while new_ponsum < ponsum and t_hat < t_pos and t_hat > 1:
```

```
            #print('-->',t_hat, ponsum, 'new', new_ponsum)
            ponsum = new_ponsum
            t_hat += i
            new_ponsum = ponderated_criterion(image,criterion,t_hat,axis)
        all_splits.append((axis,t_hat-i,ponsum)) # negative to compare to 0

    all_splits = np.array(all_splits)
    print(all_splits)
    return all_splits[np.argmin(all_splits[:,2])]

def cart_regression(image,criterion,x_start=0,y_start=0,max_depth=0):
    """ Recursively splitting image
    """
    plt.imshow(image,cmap='gray')
    plt.show()
    splits = []

    #print('size',image.size, len(image), len(image[0]))
    print(len(image),len(image[0]))
    if image.size == 0 or max_depth > 10:
        return splits

    # get best splitting position and direction
    #print(best_split(image, criterion))
    best = best_split(image, criterion)
    #print(best)
    axis,t,emp_risk_split = best

    # see if splitting is neccessary
    emp_risk = criterion(image,np.ones_like(image)+np.mean(image)-1)
    t = int(t)
    emp_risk_split = float(emp_risk_split)
    #print(emp_risk_split,emp_risk)
    if emp_risk_split < emp_risk: # if yes
        if axis == 'x':
            children  = [image[:,:t], image[:,t:]]
            risks = [criterion(im, np.ones_like(im)+np.mean(im)-1) for im in children] # s
            # discard homogeneous zone, keep splitting hetero zone
            hm = np.argmin(risks)
            splits.append((axis,x_start+t,hm,risks[hm])) # hm = 0 for left/up, 1 for right
            x_start = x_start if hm else x_start+t
            splits += cart_regression(children[1-hm],criterion,x_start,y_start,max_depth+1
        else:
            children  = [image[:t], image[t:]]
            risks = [criterion(im, np.ones_like(im)+np.mean(im)-1) for im in children] # s
            # discard homogeneous zone, keep splitting hetero zone
            hm = np.argmin(risks)
            splits.append((axis,y_start+t,hm,risks[hm])) # hm = 0 for left/up, 1 for right
            y_start = y_start if hm else y_start+t
            splits += cart_regression(children[1-hm],criterion,x_start,y_start,max_depth+1
    else:
        splits.append(('o',0,0,criterion(image, np.ones_like(image)+np.mean(image)-1)))

    return splits
```

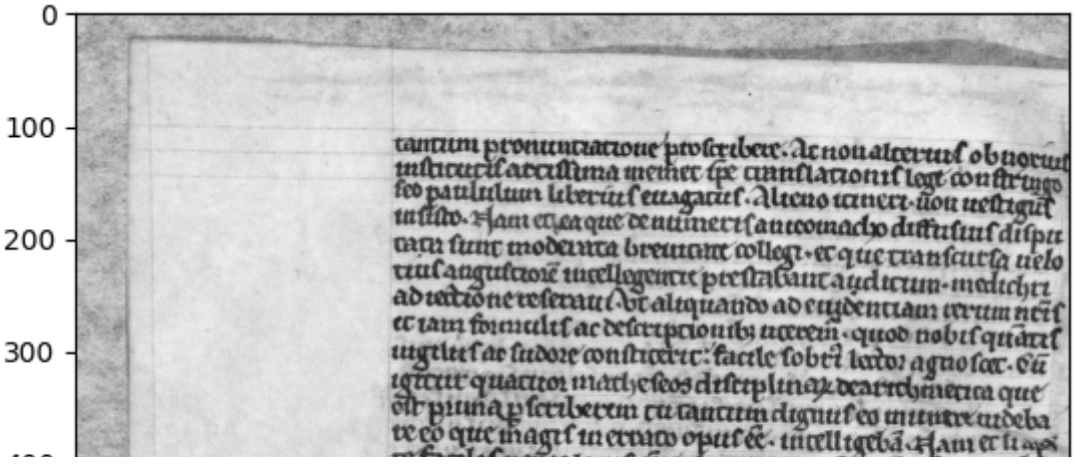
```
def show_splits(image,splits):
    plt.figure(figsize = (20,10))
    plt.imshow(image)
    #a = 0.9
    for (axis,t,d,r) in splits:
        #a *= a
        if axis == 'x':
            plt.axvline(t,linewidth=1,color='red')#,alpha=a)
        elif axis == 'y':
            plt.axhline(t,linewidth=1,color='red')#,alpha=a)



res = cart_regression(norm_image,mean_squared_error,max_depth=5)
print(res)
show_splits(image,res)
```

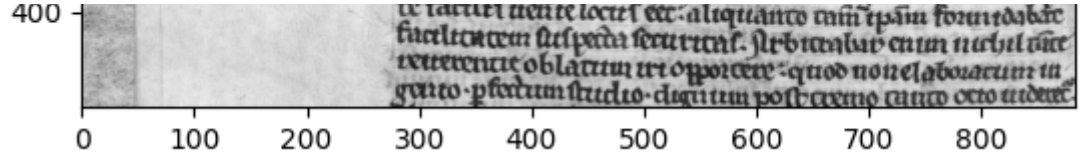


```
1436 1024
[['x' '875' '0.026651963506155427']
 ['y' '485' '0.02630307819064439']]
```

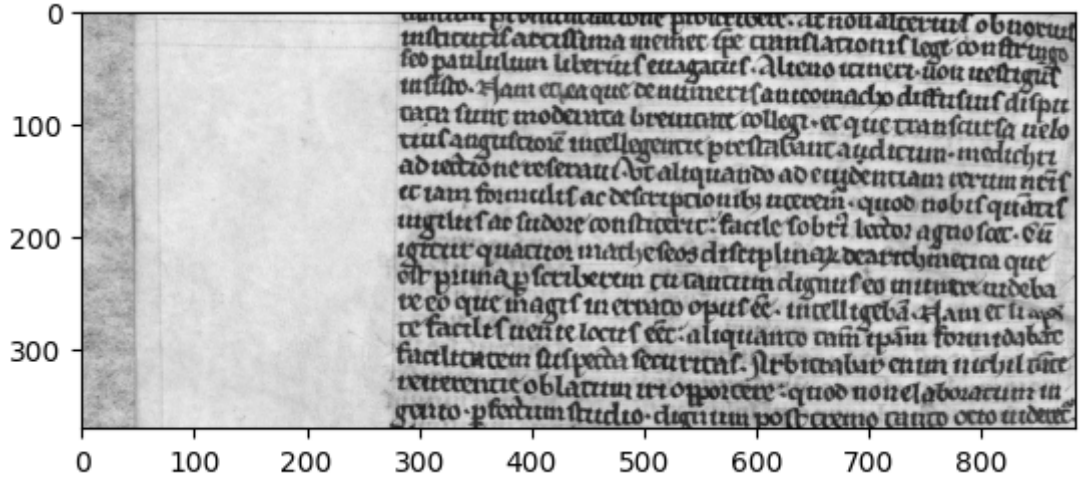


```
485 1024
[['x' '884' '0.028445507498596912']
 ['y' '299' '0.029397131650691962']]
```

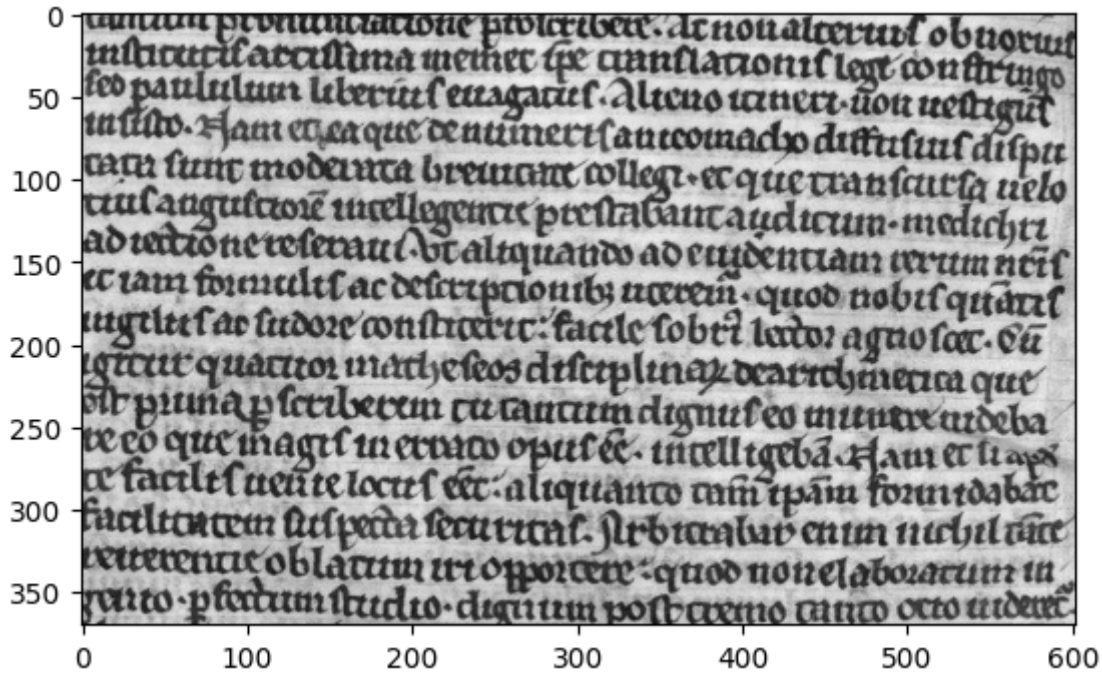




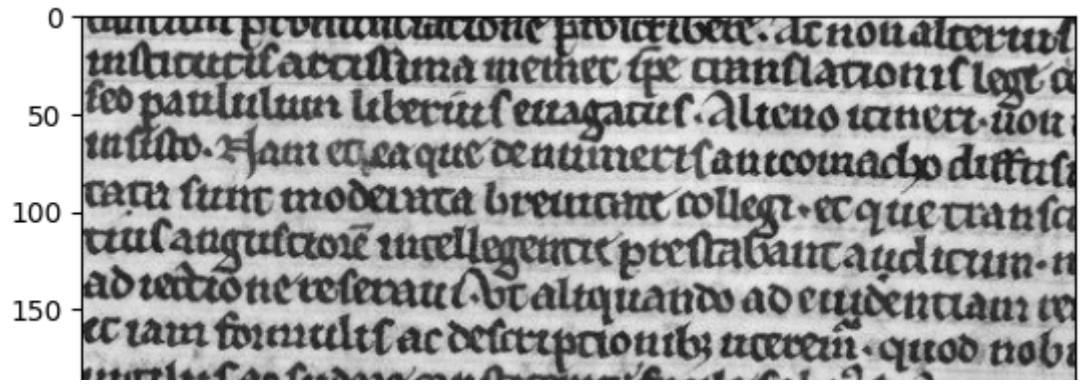

```
485 884
[['x' '877' '0.02969561131022891']
 ['y' '115' '0.027944251533332923']]
```

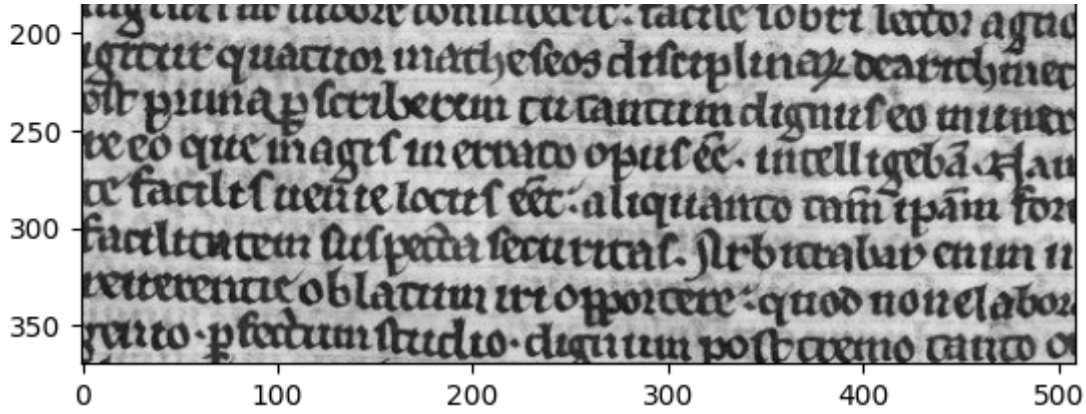



```
370 884
[['x' '282' '0.026086074835546594']
 ['y' '332' '0.034419463547350225']]
```

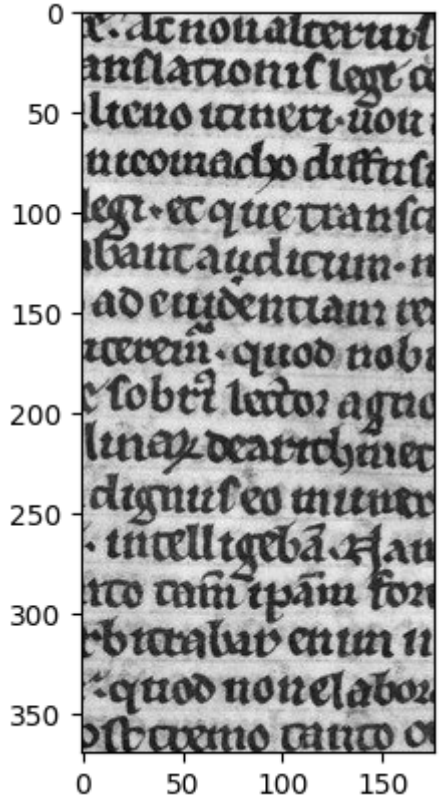



```
370 602
[['x' '509' '0.03664705302985208']
 ['y' '141' '0.03670299322594464']]
```

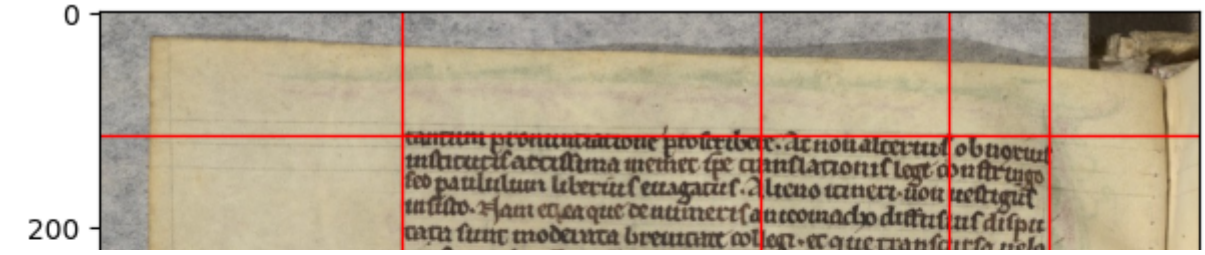





```
370 509
[['x' '333' '0.037043983230729396']
 ['y' '92' '0.037053065469697166']]
```



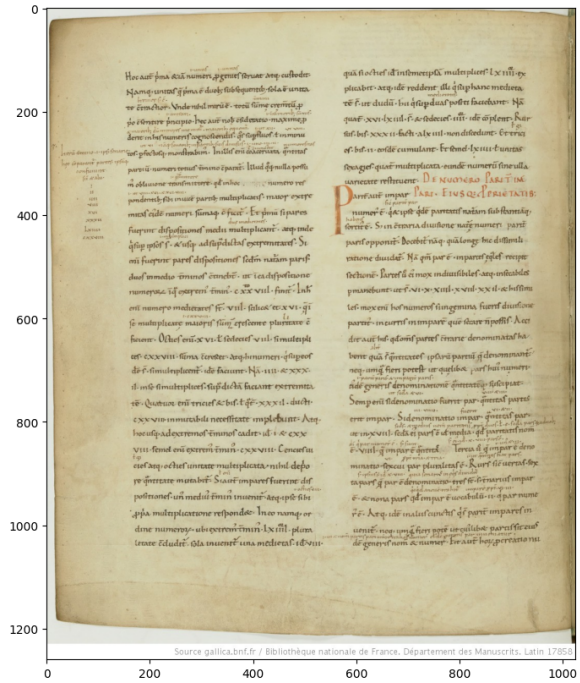

```
370 176
[('y', 485, 1, 0.024613884337580327), ('x', 884, 1, 0.020490143313368497), ('y',
```

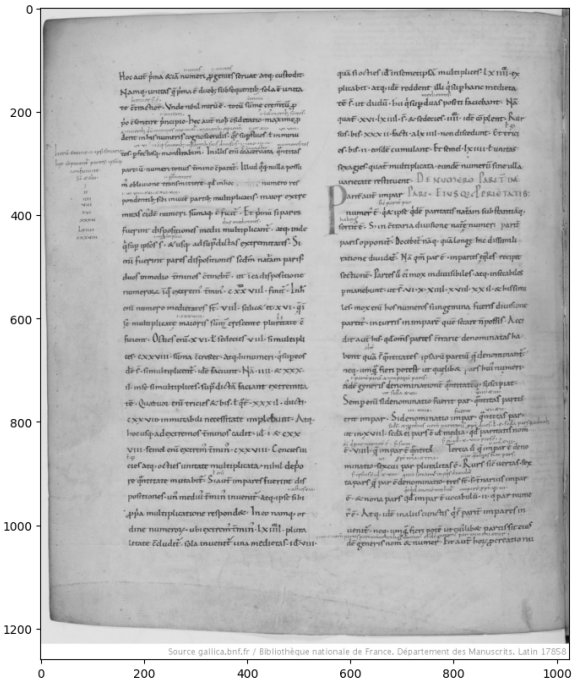



```
image = cv2.imread("../ImagesCodicologie/Corpus de jeu/Lebec/Boethius_De_institutione_arit

# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
# normalize the image
norm_image = gray/255.0
#f_image = ft_image(norm_image)
```

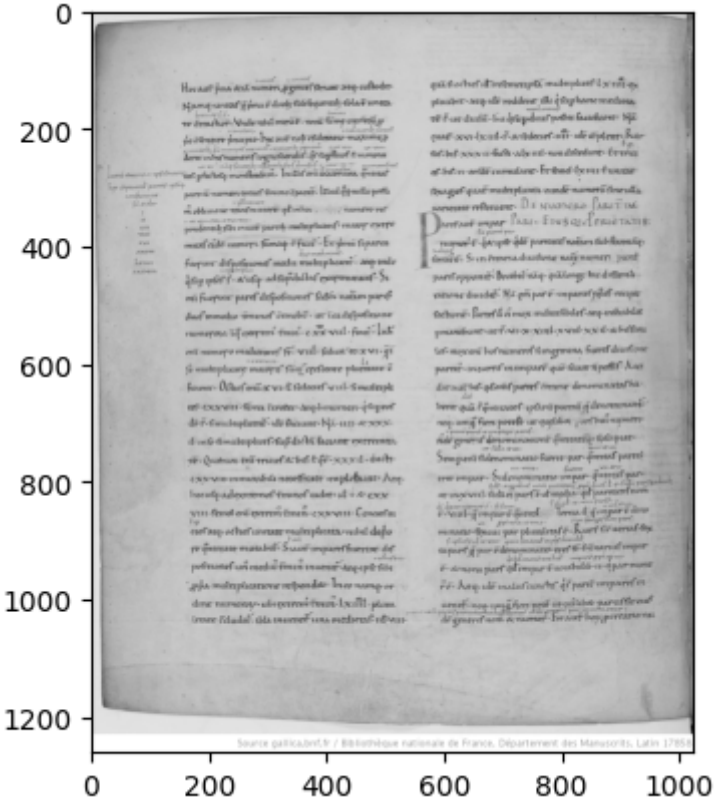
```
# Display the images
f, (ax1,ax2) = plt.subplots(1, 2, figsize=(20,10))
ax1.imshow(image)
ax2.imshow(norm_image, cmap="gray")
#ax2.imshow(f_image, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x2afbfca8cd0>
```

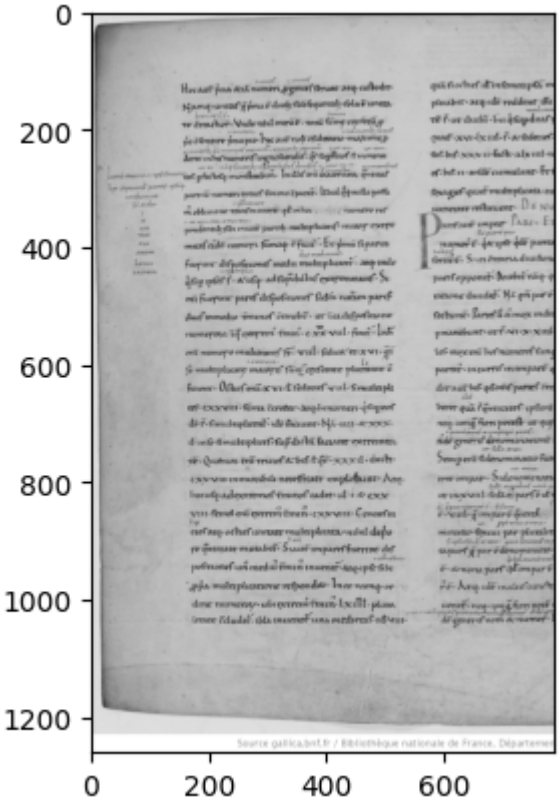




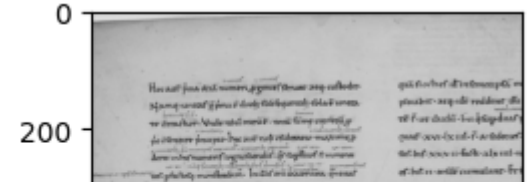

```
res = cart_regression(norm_image,mean_squared_error,max_depth=6)
print(res)
show_splits(image,res)
```

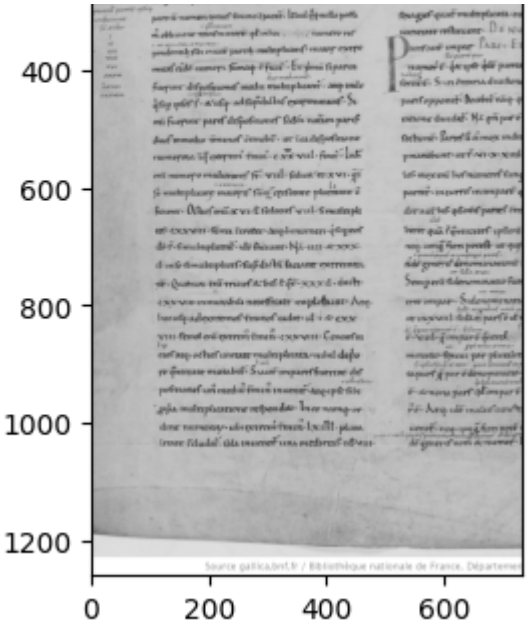



```
1260 1024
[['x' '788' '0.012837788028065109']
 ['y' '119' '0.013197590355213408']]
```

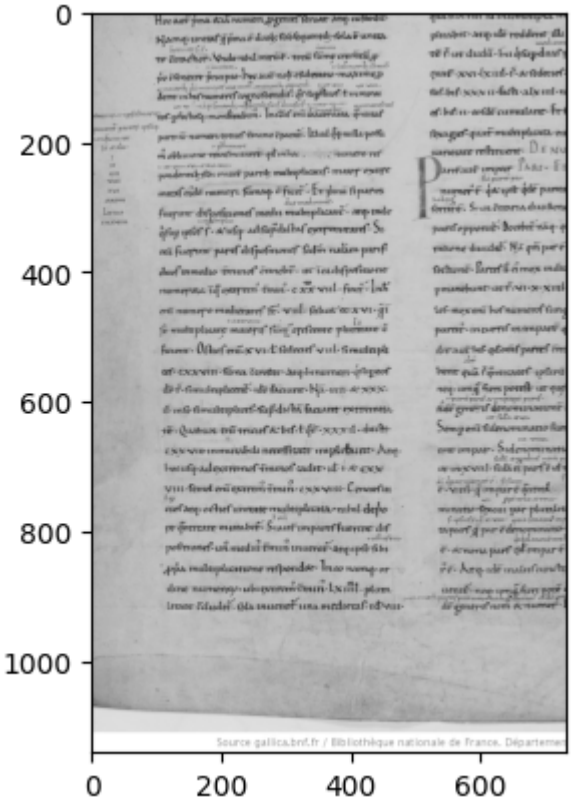



```
1260 788
[['x' '55' '0.01285050334872118']
 ['y' '449' '0.012854555827052439']]
```

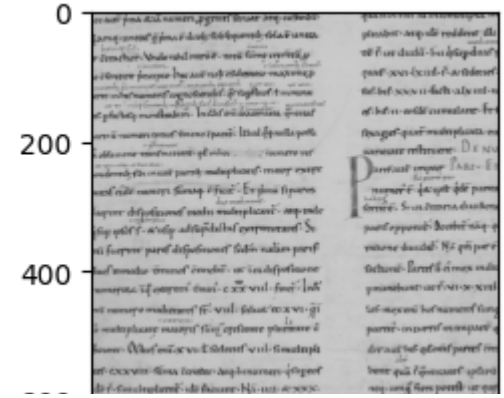


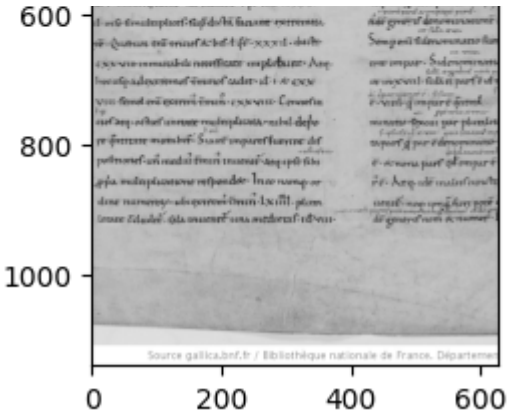



```
1260 733
[['x' '106' '0.012833278098347944']
 ['y' '119' '0.012735062731656453']]
```

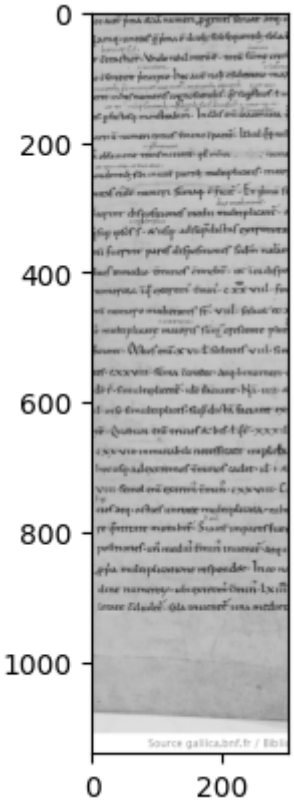



```
1141 733
[['x' '106' '0.013808989495227672']
 ['y' '496' '0.013851119961912774']]
```

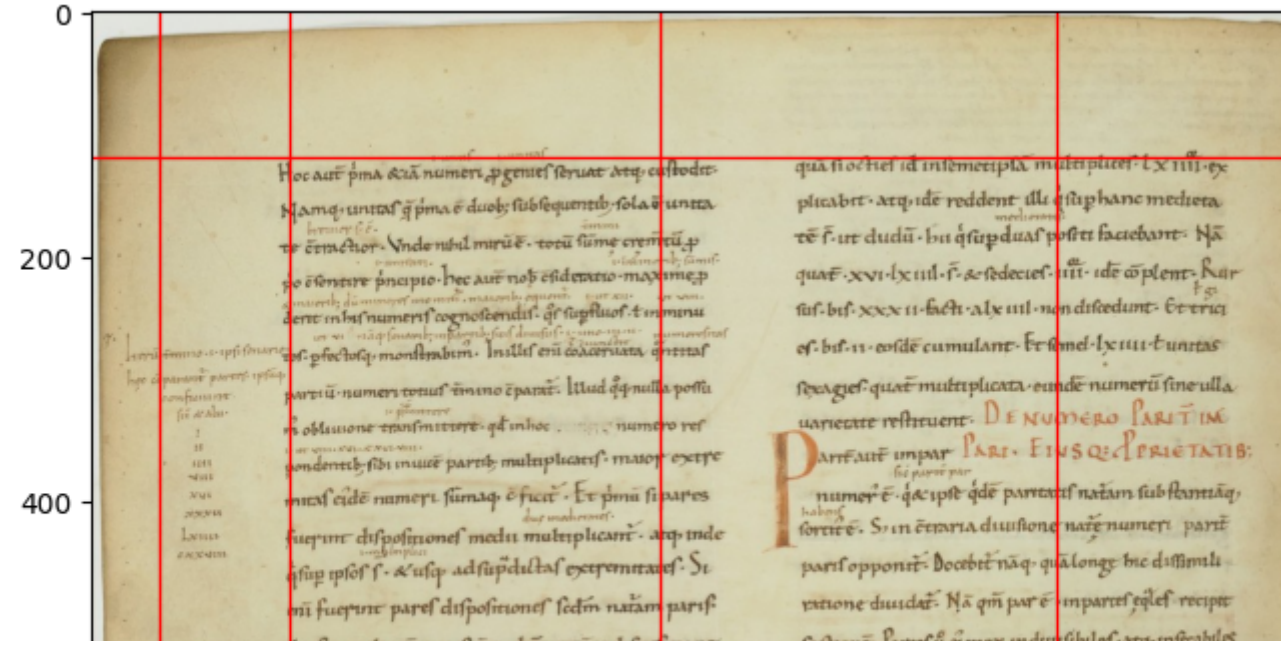


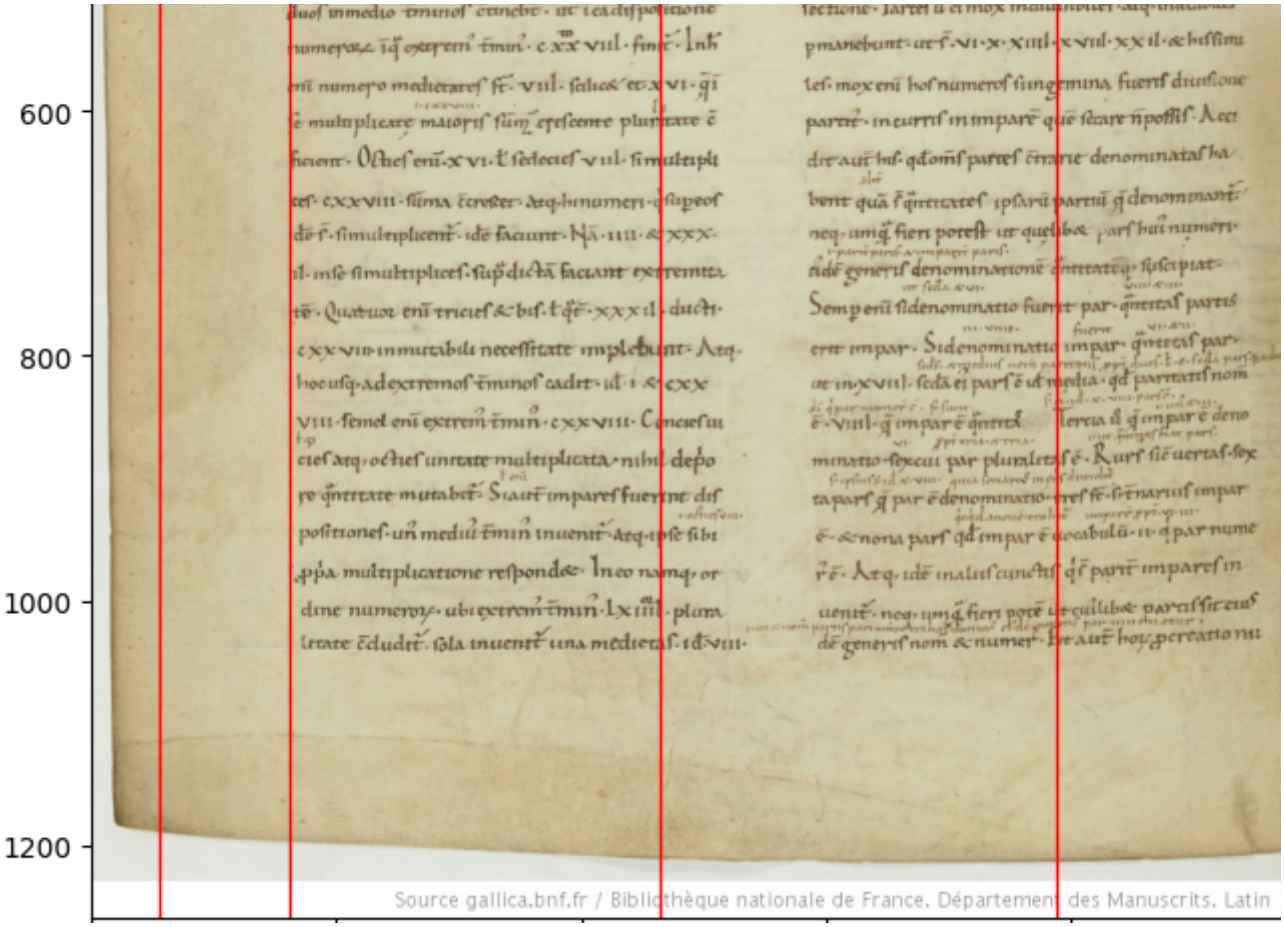



```
1141 627
[['x' '304' '0.015496092165298363']
 ['y' '164' '0.0155051775737909']]
```




```
1141 304
[('x', 788, 1, 0.012778103387465121), ('x', 55, 0, 0.012572728463747273), ('y', 119,
```

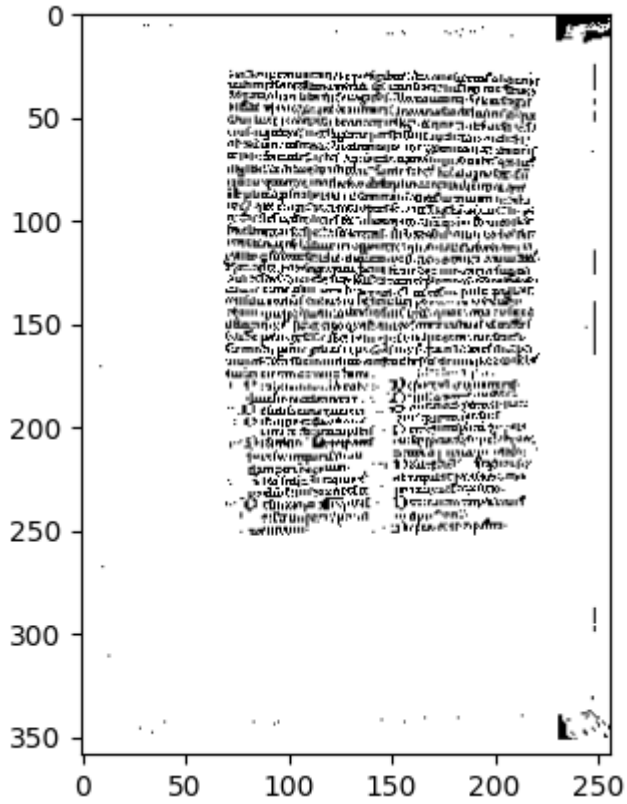





```
# perform a fast fourier transform and create a scaled, frequency transform image
def ft_image(norm_image):
 f = np.fft.fft2(norm_image)
 fshift = np.fft.fftshift(f)
 frequency_tx = 20*np.log(np.abs(fshift))

 return frequency_tx

def cart_regression_2(image,criterion):
    x = []
    y = []
    axis,t,emp_risk_split = best_split(image,criterion)
    emp_risk = criterion(image,np.ones_like(image)+np.mean(image)-1)
    t = int(t)
    emp_risk_split = int(emp_risk_split)
    if emp_risk < emp_risk_split:
        return [],[]
    return
```

```
sub_norm_image = norm_image[::4,::4]
print(len(sub_norm_image[0]))
print(sub_norm_image)
bw_sub_norm_image = np.where(sub_norm_image < 100/255,0,1)
plt.imshow(bw_sub_norm_image,cmap='gray')
```

```
256
[[0.6627451  0.6745098  0.6627451  ... 0.21960784 0.33333333 0.36862745]
 [0.62745098 0.61960784 0.69803922 ... 0.21176471 0.26666667 0.41568627]
 [0.68235294 0.6        0.61568627 ... 0.21568627 0.31372549 0.35294118]
 ...
 [1.         1.         1.         ... 1.         0.91372549 1.        ]
 [1.         1.         1.         ... 1.         0.96078431 1.        ]
 [1.         1.         1.         ... 1.         1.         1.        ]]
<matplotlib.image.AxesImage at 0x227de4004d0>
```




```
a = np.array([[2,3,5],[9,0,2]])
print(a[:,:2])
```

```
[[2 3]
 [9 0]]
```

```
print(a.size)
```

```
6
```

```
def best_split(image):
    gini = []
    for i in range(1,len(image[0])-1):
        left = image[:,:i]
        right = image[:,i:]
```

```
            gini1 = 1-np.mean(left)*(1-np.mean(left))
            gini2 = 1-np.mean(right)*(1-np.mean(right))
            gini.append(gini1*left.size + gini2*right.size)
        print(np.argmax(gini))

best_split(bw_sub_norm_image)
```

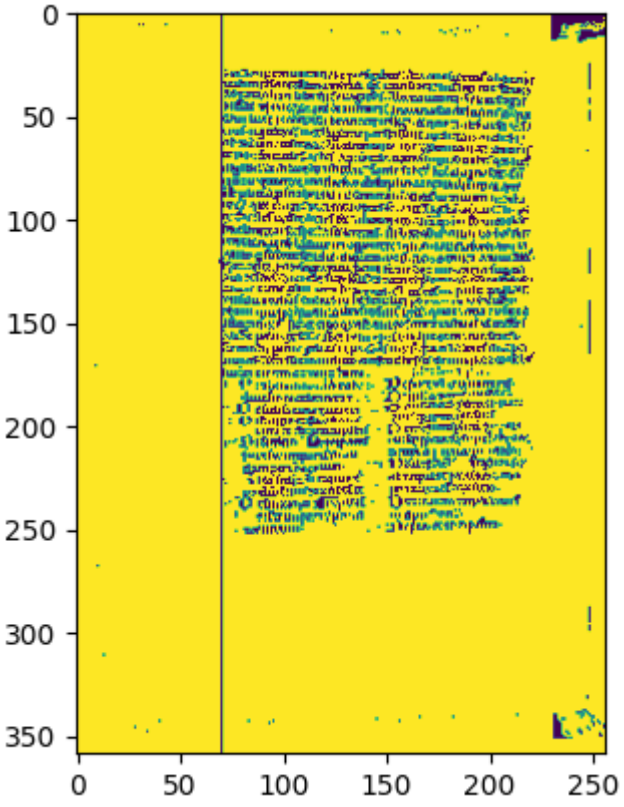
```
69
```

```
best_split(norm_image)
```

```
279
```

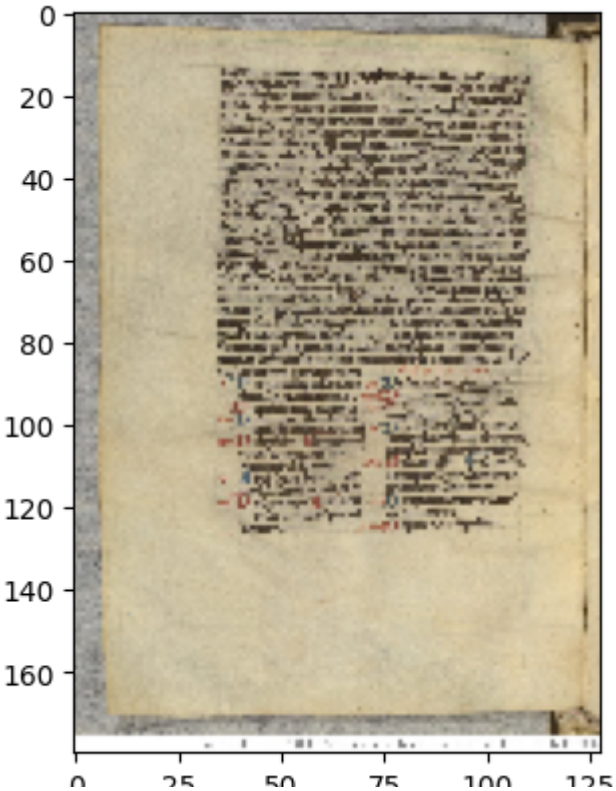
```
bw_sub_norm_image[:,70]=0
plt.imshow(bw_sub_norm_image)
```

```
<matplotlib.image.AxesImage at 0x227e0ae4b50>
```




```
from PIL import Image

imimage = np.array(Image.open("ImagesCodicologie/Corpus de jeu/Lebec/Arithmetica-f7v.jpeg"

def subSample2(image,rate):
    """ Array -> Array """
    return image[::rate,::rate]

subimage = subSample2(imimage,8)
plt.imshow(subimage)
print(len(subimage),len(subimage[0]))
```

```
180 128
```




```
imimage = np.array(Image.open("ImagesCodicologie/Corpus de jeu/Lebec/Boethius_De_instituti

def subSample2(image,rate):
    """ Array -> Array """
    return image[::rate,::rate]

subimage = subSample2(imimage,8)
plt.imshow(subimage)
print(len(subimage),len(subimage[0]))
```

```
158 128
```