```python
# Python packages used for this test. Feel free to use other packages, if needed.
import cv2
import numpy as np
from tensorflow import keras
from keras.models import Model, load_model
from keras.layers.core import Lambda
from keras.layers import Dropout, Input, concatenate
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.optimizers import Adam
from keras import backend as K
############################### 1 ###################################
# Load image (W0001_0001.png) in color.
# Load bin_image (W0002_0001.png) in color.
# Load bounding box file (XYCoordinates.txt)
# Format is (x1,x2,y1,y2)

#define image directory
load_dir = 'Load These Files/'

#load data into numpy arrays, in color by default
image = cv2.imread(load_dir + 'W0001_0001.png')
bin_image = cv2.imread(load_dir + 'W0002_0001.png')
file = np.loadtxt(load_dir + 'XYCoordinates.txt', dtype=float).astype(int)

############################### 2 ###################################
# Save full weld image with bounding boxes (append boxes to image)
# See: bound_test.png image for a smaller example.

#create copy of bound image
bound_image = image.copy()

#iterate through rows of file to place red bounding boxes at given coordinates
for f in file:
  bound_image = cv2.rectangle(bound_image,(f[0],f[2]),(f[1],f[3]),(0,0,255),1)

#save bound images to working directory
cv2.imwrite('bound_image.png',bound_image)

############################### 3 ###################################
# Save cropped images from bounding box coordinates
# (Example initially withheld) See: cropped_test_{integer}.png images
# file.seek(0,0)

# iterate through file rows and crop images by given coordinates
for i,f in enumerate(file):
  cropped_image = image[f[2]:f[3],f[0]:f[1],:]
  cropped_image = image[r[2]:r[3],r[0]:r[1],:]
  # save cropped and numbered images to working directory
  cv2.imwrite('cropped_image_{}.png'.format(i),cropped_image)

################################ 4 ###################################
# Load 4 cropped images one by one and put images in grayscale
# Then resize images using Bilinear Interpolation and save images.
# (Example initially withheld) See: resized_test_{integer}.png images

# iterate through the 4 previously saved cropped images to resize
# resized to (256,256) to allow for input to U-Net model below
for i in range(4):
  cropped_image = cv2.imread('cropped_image_{}.png'.format(i), 0) # 0 for grayscale
  resized_image = cv2.resize(cropped_image, (256, 256), interpolation=cv2.INTER_LINEAR)
  #save resized images to working directory
  cv2.imwrite('resized_image_{}.png'.format(i), resized_image)

################################ 5 ###################################
# Load 4 resized_test_{integer}.png images
# one by one and rotate 90 degrees clockwise and save images.
# (Example initially withheld)See: rotated_test_{integer}.png images

# iterate through the 4 previously saved resized images to rotate
for i in range(4):
  resized_image = cv2.imread('resized_image_{}.png'.format(i), 0)
  rotated_image = cv2.rotate(resized_image, cv2.ROTATE_90_CLOCKWISE)
  # save rotated images to working directory
  cv2.imwrite('rotated_image_{}.png'.format(i), rotated_image)

################################ 6 ###################################
# Turn temp_bin_image values from black/white to black/red.
# Ie, [[[0,0,0],[255,255,255],...]] to [[[0,0,0],[0,0,255],...]] when BGR.
# Get max of “image” and “temp_bin_image” together to show defect overlay and save image.
# See: red_overlay.png image

# create copy of bin_image for manipulation
temp_bin_image = bin_image.copy()

#replace all white pixels with red
temp_bin_image[temp_bin_image[:,:,2]==255] = [0,0,255]

# superimpose and output maximum of image and temp_bin
red_overlay = np.maximum(temp_bin_image,image)

# save red_overlay image to working directory
cv2.imwrite('red_overlay.png',red_overlay)
```

```
################################# / #####################################
# Show an example of: (AB)T = (BT)(AT)
A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
B = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])

# transpose matrix multiplcation product of A and B
left_side = np.dot(A,B).T

# matrix multiplication product of transpose(B) and transpose(A)\\
# by definition of tranpose multiplication identity
right_side = np.dot(B.T,A.T)
print(np.equal(left_side, right_side))

################################# 8 #####################################

# convolution layers: CXX, max pooling layers: PXX, upscaling layers: UXX
def build_model(input_layer, start_filters, start_kernal_size):
    C1 = Conv2D(start_filters,start_kernal_size,padding='same',activation='relu')(input_layer)
    C2 = Conv2D(start_filters,start_kernal_size,padding='same',activation='relu')(C1)
    P1 = MaxPooling2D(pool_size=(2,2))(C2)
    C3 = Conv2D(2*start_filters,start_kernal_size,padding='same',activation='relu')(P1)
    C4 = Conv2D(2*start_filters, start_kernal_size,padding='same', activation='relu')(C3)
    P2 = MaxPooling2D(pool_size=(2,2))(C4)
    C5 = Conv2D(4*start_filters,start_kernal_size,padding='same',activation='relu')(P2)
    C6 = Conv2D(4*start_filters, start_kernal_size,padding='same', activation='relu')(C5)
    P3 = MaxPooling2D(pool_size=(2,2))(C6)
    C7 = Conv2D(4*start_filters, start_kernal_size,padding='same', activation='relu')(P3)
    C8 = Conv2D(4*start_filters, start_kernal_size,padding='same', activation='relu')(C7)
    U1 = Conv2DTranspose(4*start_filters,start_kernal_size,padding='same',strides=(2,2))(C8)
    U1 = Dropout(0.1)(concatenate([U1,C6]))
    C9 = Conv2D(2*start_filters,start_kernal_size,padding='same',activation='relu')(U1)
    C10 = Conv2D(2*start_filters,start_kernal_size,padding='same',activation='relu')(C9)
    U2 = Conv2DTranspose(2*start_filters, start_kernal_size,padding='same',
```

```python
strides=(2, 2))(C10)
    U2 = Dropout(0.1)(concatenate([U2, C4]))
    C11 = Conv2D(start_filters, start_kernal_size,padding='same', activation='relu')(U2)
    C12 = Conv2D(start_filters, start_kernal_size,padding='same', activation='relu')(C11)
    U3 = Conv2DTranspose(start_filters, start_kernal_size,padding='same', strides=(2, 2))(C12)
    U3 = Dropout(0.1)(concatenate([U3, C2]))
    C13 = Conv2D(start_filters, start_kernal_size,padding='same', activation='relu')(U3)
    C14 = Conv2D(start_filters, start_kernal_size,padding='same', activation='relu')(C13)
    output_layer = Conv2D(1, start_kernal_size,padding='same', activation='sigmoid')(C14)

    model = Model(input_layer, output_layer)
    model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy')
    return model

if __name__ == '__main__':
    input_img = np.expand_dims(cv2.imread('resized_image_1.png',0),axis=2)
    input = Input((np.shape(input_img)))
    input_layer = Lambda(lambda input_img: input_img / 255)(input) #normalized input data
    model = build_model(input_layer=input_layer, start_filters=112, start_kernal_size=3)
    model.save('U-Net')
    print(model.summary())
```