Neural Networks Predictions altogteher

Author: Alexander Goudemond, Student Number: 219030365

Please ensure the following folders containg images exist:

drive > MyDrive > COMP700_Images > COMP700_Processed_Training_GT

drive > MyDrive > COMP700_Images > COMP700_Processed_Training_ST

drive > MyDrive > COMP700_Images > COMP700_Raw_Training_GT

drive > MyDrive > COMP700_Images > COMP700_Raw_Training_ST

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_128_GT

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_256_GT

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_512_GT

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_128_ST

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_256_ST

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_512_ST

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_Processed_Images_GT

drive > MyDrive > COMP700_Images > COMP700_Patchify_Images_Processed_Images_ST

Then, please ensure a seperate folder with the notebooks and text files exists:

drive > MyDrive > COMP700_Neural_Network_Code

The first 4 image folders were generated offline by the other notebooks and then uploaded to Google Drive, whereas the next 6 were generated by the notebook 011. The final 2 were generated by 015

Overall Neural Network Reflection

After comparing the results from notebook 013 and 017, the best models appear to be the following:

Dataset	Model Name	Patch Size	Observations
GT - Raw	UNET_Model_Dimension_256_3	256	Decent
ST - Raw	UNET_Model_Dimension_256_3	256	Excellent
GT - Processed	UNET_Model_Dimension_256_3	256	Fair, crisp boundaries - but partially fragmented
ST - Processed	UNET_Model_Dimension_256_3	256	Excellent

In this notebook, those 4 models will be loaded and used to compare the results for each dataset. These comparisons will initially be done altogether, and then done indivually, for each dataset.

The hope is that by comparing all the work done so far - the results will be clear and then the project goal is complete

Mount Drive

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

In [ ]:
```
from os import getcwd
google_drive_path = "/drive/MyDrive"
training_data_directory = getcwd() + google_drive_path + "/COMP700_Images/"
```

Installs

In [ ]:
```
!pip install patchify
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
ic/simple/
Requirement already satisfied: patchify in /usr/local/lib/python3.7/dist-packages (0.2.
3)
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.7/dist-packages (fr
om patchify) (1.21.6)
```

Imports

In [ ]:
```
from os import getcwd, walk, mkdir, stat, remove
from os import sep # used later on, in a function, to print directory contents
from os.path import exists, basename, join

from shutil import copyfile

from PIL.Image import fromarray
import cv2

import matplotlib.pyplot as plt
import numpy as np

from patchify import patchify, unpatchify

import tensorflow as tf
from tensorflow import keras
from keras.layers import Conv2D, Dropout, MaxPooling2D, Conv2DTranspose, concatenate, In
from keras.metrics import MeanIoU

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from IPython.display import clear_output
```

Useful functions

In [ ]:
```
def getImagePaths(path):
    image_paths = []
    for root, dirs, files in walk(path):
        if (len(files) != 0):
          # print(len(files))
          image_paths.append( root )

    image_paths.sort()

    return image_paths
###
```

In [ ]:
```
# returns tuple
def extractDirectoryPaths(path):
    x_directory_locations, y_directory_locations = [], []
    temp = ""

    with open(path) as f:
        lines = f.readlines()
        for item in lines:
            temp = item[ : -1] # remove newline char at end
            if ("X" in temp):
                x_directory_locations.append( temp )
            else:
                y_directory_locations.append( temp )

    return (x_directory_locations, y_directory_locations)
###

def loadImagePathsFromArray(array):
    image_paths = []

    for path in array:
        for root, dirs, files in walk(path):
            if (len(files) != 0):
                for item in files:
                    image_paths.append(root + "/" + item)

    image_paths.sort()

    return image_paths
###
```

In [ ]:
```
# assume cropped already
def patchifyIndividualImage(image, patch_size, isColourImage=True):
    scaler = MinMaxScaler()
    image_dataset = []

    print("Patchify process starting!")

    #Extract patches from each image
    if (isColourImage):
        patches_img = patchify(image, (patch_size, patch_size, 3), step=patch_size)
    else:
        patches_img = patchify(image, (patch_size, patch_size), step=patch_size) # grays

    for i in range(patches_img.shape[0]):
        for j in range(patches_img.shape[1]):
            single_patch_img = patches_img[i,j,:,:]

            #Use minmaxscaler instead of just dividing by 255.
```

```
            single_patch_img = scaler.fit_transform(
                single_patch_img.reshape(-1, single_patch_img.shape[-1])
            ).reshape( single_patch_img.shape )

            #single_patch_img = (single_patch_img.astype('float32')) / 255.
            #Drop the extra unecessary dimension that patchify adds.
            single_patch_img = single_patch_img[0]

            # scale up values - don't do this for the NN!
            # img_new = (single_patch_img * 255).astype(int)
            img_new = single_patch_img

            image_dataset.append(img_new)

    print("Patchify process complete!")

    return image_dataset
###
```

In [ ]:
```
from tensorflow import keras
from keras.utils import array_to_img

def display(display_list, title=[], figsize=(15, 15)):
    plt.figure(figsize=figsize)

    # update if title not provided
    if (len(title) == 0):
      title = ['Input Image', 'True Mask', 'Predicted Mask']

    for i in range(len(display_list)):
        plt.subplot(1, len(display_list), i+1)
        plt.title(title[i], wrap=True)

        # handle 2D and 3D images
        if (len(display_list[i].shape) == 3 ):
            plt.imshow( array_to_img(display_list[i]), cmap="gray")
        else:
            plt.imshow( display_list[i], cmap="gray")

        plt.axis('off')
    plt.tight_layout() # prevents overlap
    plt.show()
###
```

In [ ]:
```
# create directories for work we create
def tryMakeDirectories(current_directory, myList):
    path = ""
    for item in myList:
        # initialize
        if (path == ""):
            path = item
        else:
            path = path + "/" + item

        try:
            # join comes from os.path
            mkdir( join(current_directory, path) )
        except FileExistsError:
            # print("Folder already exists!")
            pass
        except:
            print("Unknown Error Encountered...")
###
```

In [ ]:
```
def extractRhsString(string, symbol):
    index = string.rfind(symbol)
    return string[ index + len(symbol) : ]
###

def extractLhsString(string, symbol):
    index = string.find(symbol)
    return string[ : index ]
###
```

In [ ]:
```
def cropImage(image, patch_size):
    #Nearest size divisible by our patch size
    SIZE_X = (image.shape[1] // patch_size) * patch_size
    SIZE_Y = (image.shape[0] // patch_size) * patch_size

    image = fromarray(image)
    # Crop entire image into desirable shape
    image = image.crop((0 ,0, SIZE_X, SIZE_Y))
    image = np.array(image)

    return image
###

# save and read all to ensure dtype=uint8
def readImage(img_path):
    image = plt.imread(img_path)
    plt.imsave("temp.png", image, cmap='gray')
    image = cv2.imread("temp.png")

    # try ensure 8 bit images used
    # x = x.astype(int)
    # y = np.array(x, dtype="uint8")

    return image
###
###
```

In [ ]:
```
def redesignImagePaths(array, directory_list):
    temp_2d_array = []
    temp_array = []
    count = 0
    next_directory = ""

    for path in array:
        # initialize
        if (count == 0):
            next_directory = directory_list[count]
            count += 1

        if (next_directory in path and count <= len(directory_list)):
            # print(path)
            # print(count)

            if (count < len(directory_list)):
                next_directory = directory_list[count]

            # place all contents into 2d array
            temp_2d_array.append(temp_array)

            temp_array = [] # reset
            count += 1

        else:
            temp_array.append(path)
```

```
    # at end, append final list
    temp_2d_array.append(temp_array)

    return temp_2d_array[ 1 : ] # remove dummy element
```

In [ ]:
```
from keras.utils import array_to_img

def convertFloatRGBtoBinary(img):
    plt.imsave("temp.png", img)
    gray = cv2.imread("temp.png", cv2.IMREAD_GRAYSCALE)

    if (exists("temp.png")):
        remove("temp.png")

    gray[gray < 255] = 0
    gray[gray == 255] = 1

    return gray.astype(float)
###
```

In [ ]:
```
def reshapePatchifiedImages(og_image, patches, height, width, patch_size, invert_scaling
    # use original patchify to get integer slices of image
    collection = patchify(og_image, (patch_size, patch_size, 3), step=patch_size)
    counter = 0
    row_of_images, organised_patches = [], []

    # i * j == (patchifyIndividualImage(...).shape)[0]
    for i in range(height // patch_size):
        for j in range(width // patch_size):
            single_image = patches[counter]                    # 3D object

            # map to integer again?
            if (invert_scaling):
                # use og_image patches to find min and max ints
                min = np.min(collection[i,j,:,:][0])
                max = np.max(collection[i,j,:,:][0])
                single_image = (single_image * (max - min)) + min

            new_img = [single_image]                           # 4D object

            row_of_images.append( new_img )

            counter += 1

        # print(len(row_of_images))
        organised_patches.append( row_of_images )              # stick rows ontop of one
        row_of_images = []

    return organised_patches
###
```

In [ ]:
```
# crop
def bulkCrop(x_paths, y_paths, patch_size):
    cropped_x, cropped_y = [], []

    for i in range(len(x_paths)):
        cropped_x.append( cropImage( readImage(x_paths[i]), patch_size) )
        cropped_y.append( cropImage( readImage(y_paths[i]) , patch_size) )

    return cropped_x, cropped_y
###
```

In [ ]:
```
from keras.models import load_model
```

```
def getModel(path, model_name):
    return load_model(path + model_name)
###
```

In [ ]:
```
def patchifyAndPredict(image, model, patch_size, threshold=0.5):
    patchified_images = patchifyIndividualImage(image, patch_size=patch_size)

    patchified_images = np.array(patchified_images)
    patchified_predictions = model.predict(patchified_images)

    # adjust threshold here
    threshold = threshold
    patchified_predictions_thresh = patchified_predictions
    patchified_predictions_thresh[patchified_predictions_thresh >= threshold] = 1
    patchified_predictions_thresh[patchified_predictions_thresh < threshold] = 0

    # fix colours - dont do this, for unpatchify process!
    # answer = []
    # for image in patchified_predictions_thresh:
    #     answer.append( convertFloatRGBtoBinary(image) )

    return patchified_predictions_thresh
###
```

In [ ]:
```
def rebuildImageFromPatches(og_image, image_patches, patch_size, invert_scaling=True):
    height, width, _ = np.array(og_image).shape


    redesigned_images = reshapePatchifiedImages(og_image,
                                                image_patches,
                                                height,
                                                width,
                                                patch_size,
                                                invert_scaling=invert_scaling)
    # print(np.array(og_image).shape)
    # print(np.array(redesigned_images).shape)

    redesigned_images = np.array(redesigned_images)
    rebuilt_image = unpatchify(redesigned_images, np.array(og_image).shape)

    return rebuilt_image
###
```

In [ ]:
```
# expects images of patch_size
def conductPrediction(image, patch_size, path, model, threshold = 0.5, invert_scaling=Tr

    predicted_patches = patchifyAndPredict(image, model, patch_size, threshold=threshold

    rebuilt_image = rebuildImageFromPatches(image,
                                            predicted_patches,
                                            patch_size,
                                            invert_scaling=invert_scaling
                                            )

    # fix colours and return
    return convertFloatRGBtoBinary(rebuilt_image)
###


def getBinaryClasses(mask_array):
    answer = []
    for image in mask_array:
        temp = image
        temp[temp > 0] = 1
```

```
        answer.append(temp)

    return answer
###
```

In [ ]:
```
from tensorflow import keras
from keras.utils import array_to_img

def display2DPlot(display_list, title, figsize=(15, 15)):
    plt.figure(figsize=figsize)
    counter = 0

    for i in range(len(display_list)):
        for j in range(len(display_list[0])):
            counter += 1
            plt.subplot(len(display_list), len(display_list[0]), counter)
            plt.title(title[i][j], wrap=True)

            # handle 2D and 3D images
            if (len(display_list[i][j].shape) == 3 ):
                plt.imshow( array_to_img(display_list[i][j]), cmap="gray")
            else:
                plt.imshow( display_list[i][j], cmap="gray")

            plt.axis('off')


        # print("i", i)

    plt.tight_layout() # prevents overlap
    plt.show()
###
```

In [ ]:
```
def conductBulkPrediction(cropped_x, cropped_y, patch_size, path, model_name, threshold=
    random_cropped_x, random_cropped_y =  bulkCrop(cropped_x, cropped_y, patch_size)
    random_cropped_y = getBinaryClasses(random_cropped_y) # doesnt impact preditions, us

    model = getModel(path, model_name)     # load model
    array, labels = [], []

    for i in range(len(random_cropped_x)):
        prediction = conductPrediction(random_cropped_x[i],
                                       patch_size = patch_size,
                                       path = path,
                                       model = model,
                                       threshold = threshold,
                                       invert_scaling=False
                                       )

        array.append( [random_cropped_x[i], random_cropped_y[i], prediction] )
        labels.append( ["X_" + str(i+1), "Y_" + str(i+1), "Prediction_" + str(i+1)] )

    return array, labels
###

def save2DPlot(display_list, title, location, filename, figsize=(15, 15)):
    plt.figure(figsize=figsize)
    counter = 0

    for i in range(len(display_list)):
        for j in range(len(display_list[0])):
            counter += 1
            plt.subplot(len(display_list), len(display_list[0]), counter)
            plt.title(title[i][j], wrap=True)
```

```
            # handle 2D and 3D images
            if (len(display_list[i][j].shape) == 3 ):
                plt.imshow( array_to_img(display_list[i][j]), cmap="gray")
            else:
                plt.imshow( display_list[i][j], cmap="gray")

            plt.axis('off')


        # print("i", i)

    plt.tight_layout() # prevents overlap
    plt.savefig(location + filename)
###
```

Loading all Training Data

Here, we can read in the contents of our desired text files and prepare them to be shuffled

In [ ]:
```
google_drive_path = "/drive/MyDrive"
text_file_location = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
```

Processed ST

In [ ]:
```
filename = "processed_training_paths_gt.txt"
gt_processed_x_image_paths, gt_processed_y_image_paths = extractDirectoryPaths(text_file

gt_processed_x_images = loadImagePathsFromArray(gt_processed_x_image_paths)
gt_processed_y_images = loadImagePathsFromArray(gt_processed_y_image_paths)

if (len(gt_processed_x_images) == len(gt_processed_y_images)):
    print("Same quantity of images and masks!")
else:
    print("Not all pictures match...")
```

```
Same quantity of images and masks!
```

Processed GT

In [ ]:
```
filename = "processed_training_paths_st.txt"
st_processed_x_image_paths, st_processed_y_image_paths = extractDirectoryPaths(text_file

st_processed_x_images = loadImagePathsFromArray(st_processed_x_image_paths)
st_processed_y_images = loadImagePathsFromArray(st_processed_y_image_paths)

if (len(st_processed_x_images) == len(st_processed_y_images)):
    print("Same quantity of images and masks!")
else:
    print("Not all pictures match...")
```

```
Same quantity of images and masks!
```

In [ ]:
```
print("There are", len(st_processed_x_images), "ST images and", len(gt_processed_x_image
```

```
There are 7742 ST images and 578 GT images
```

Raw GT

In [ ]:
```
filename = "raw_training_paths_gt.txt"
gt_raw_x_image_paths, gt_raw_y_image_paths = extractDirectoryPaths(text_file_location +

gt_raw_x_images = loadImagePathsFromArray(gt_raw_x_image_paths)
gt_raw_y_images = loadImagePathsFromArray(gt_raw_y_image_paths)

if (len(gt_raw_x_images) == len(gt_raw_y_images)):
    print("Same quantity of images and masks!")
else:
    print("Not all pictures match...")
```

```
Same quantity of images and masks!
```

Raw ST

In [ ]:
```
filename = "raw_training_paths_st.txt"
st_raw_x_image_paths, st_raw_y_image_paths = extractDirectoryPaths(text_file_location +

st_raw_x_images = loadImagePathsFromArray(st_raw_x_image_paths)
st_raw_y_images = loadImagePathsFromArray(st_raw_y_image_paths)

if (len(st_raw_x_images) == len(st_raw_y_images)):
    print("Same quantity of images and masks!")
else:
    print("Not all pictures match...")
```

```
Same quantity of images and masks!
```

In [ ]:
```
print("There are", len(st_raw_x_images), "ST images and", len(gt_raw_x_images), "GT imag
```

```
There are 7742 ST images and 578 GT images
```

Prepare directory variables

Next, we need to partition the data, as we desire to select 1 image from each folder:

In [ ]:
```
gt_directory_list = []
temp = ""
count = 0

for path in gt_raw_x_image_paths:
    if (count % 2 == 0):
        temp = extractRhsString(path, "GT/")
        temp = extractLhsString(temp, "/")
        # print(temp)
        gt_directory_list.append(temp)

    count += 1

gt_directory_list
```

Out[ ]:
```
['BF-C2DL-HSC',
 'BF-C2DL-MuSC',
 'DIC-C2DH-HeLa',
 'Fluo-C2DL-Huh7',
 'Fluo-C2DL-MSC',
 'Fluo-N2DH-GOWT1',
 'Fluo-N2DH-SIM+',
 'Fluo-N2DL-HeLa',
 'PhC-C2DH-U373',
 'PhC-C2DL-PSC']
```

In [ ]:
```
st_directory_list = []
temp = ""
count = 0

for path in st_raw_x_image_paths:
    if (count % 2 == 0):
        temp = extractRhsString(path, "ST/")
        temp = extractLhsString(temp, "/")
        # print(temp)
        st_directory_list.append(temp)

    count += 1

st_directory_list
```

Out[ ]:
```
['BF-C2DL-HSC',
 'BF-C2DL-MuSC',
 'DIC-C2DH-HeLa',
 'Fluo-C2DL-MSC',
 'Fluo-N2DH-GOWT1',
 'Fluo-N2DL-HeLa',
 'PhC-C2DH-U373',
 'PhC-C2DL-PSC']
```

Reorganize GT Data

In [ ]:
```
gt_raw_x_images = redesignImagePaths(gt_raw_x_images, gt_directory_list)

len(gt_raw_x_images)
```

Out[ ]: 10

In [ ]:
```
gt_raw_y_images = redesignImagePaths(gt_raw_y_images, gt_directory_list)

len(gt_raw_y_images)
```

Out[ ]: 10

In [ ]:
```
gt_processed_x_images = redesignImagePaths(gt_processed_x_images, gt_directory_list)

len(gt_processed_x_images)
```

Out[ ]: 10

In [ ]:
```
gt_processed_y_images = redesignImagePaths(gt_processed_y_images, gt_directory_list)

len(gt_processed_y_images)
```

Out[ ]: 10

Reorganize ST Data

In [ ]:
```
st_raw_x_images = redesignImagePaths(st_raw_x_images, st_directory_list)

len(st_raw_x_images)
```

Out[ ]: 8

In [ ]:
```
st_raw_y_images = redesignImagePaths(st_raw_y_images, st_directory_list)
```

```
len(st_raw_y_images)
```

Out[ ]:
```
8
```

In [ ]:
```
st_processed_x_images = redesignImagePaths(st_processed_x_images, st_directory_list)

len(st_processed_x_images)
```

Out[ ]:
```
8
```

In [ ]:
```
st_processed_y_images = redesignImagePaths(st_processed_y_images, st_directory_list)

len(st_processed_y_images)
```

Out[ ]:
```
8
```

Verify data corresponds

In [ ]:
```
count = 0

for i in range(len(gt_raw_x_images)):
    for j in range(len(gt_raw_x_images[i])):
        if (extractRhsString(gt_raw_x_images[i][j], "/t")[:-4]
            !=
            extractRhsString(gt_processed_x_images[i][j], "_t")[:-4]
            ):
            count += 1

print(count, "image(s) do not match")
```

```
0 image(s) do not match
```

In [ ]:
```
count = 0

for i in range(len(st_raw_x_images)):
    for j in range(len(st_raw_x_images[i])):
        if (extractRhsString(st_raw_x_images[i][j], "/t")[:-4]
            !=
            extractRhsString(st_processed_x_images[i][j], "_t")[:-4]
            ):
            count += 1
            print("i, j:", i, j)

print(count, "image(s) do not match")
```

```
0 image(s) do not match
```

Select random images

In [ ]:
```
from random import randint

random_gt_raw_x_paths, random_gt_raw_y_paths = [], []
random_gt_processed_x_paths, random_gt_processed_y_paths = [], []

index = 0
count = 0

for paths in gt_raw_x_images:
    # print(paths)
    index = randint(0, len(paths)-1)
```

```
    random_gt_raw_x_paths.append(gt_raw_x_images[count][index])
    random_gt_raw_y_paths.append(gt_raw_y_images[count][index])

    random_gt_processed_x_paths.append(gt_processed_x_images[count][index])
    random_gt_processed_y_paths.append(gt_processed_y_images[count][index])

    count += 1

count = 0
for i in range(len(random_gt_raw_x_paths)):
    if (extractRhsString(random_gt_raw_x_paths[i], "/t")[ : -4]
        !=
        extractRhsString(random_gt_raw_y_paths[i], "/man_seg")[ : -4]
        ):
        count += 1

print(count, "image(s) do not match for RAW")

count = 0
for i in range(len(random_gt_processed_x_paths)):
    if (extractRhsString(random_gt_processed_x_paths[i], "_t")[ : -4]
        !=
        extractRhsString(random_gt_processed_y_paths[i], "/man_seg")[ : -4]
        ):
        count += 1

print(count, "image(s) do not match for Processed")
```

```
0 image(s) do not match for RAW
0 image(s) do not match for Processed
```

In [ ]:

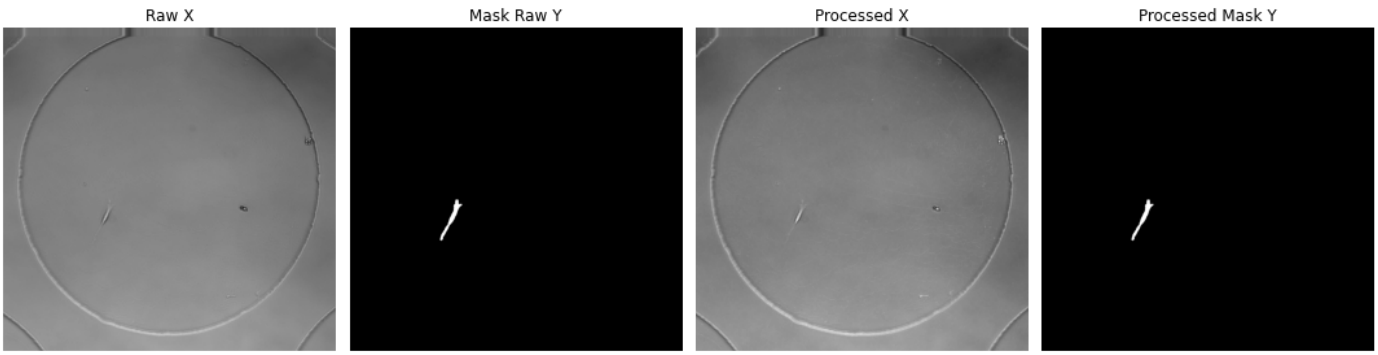
```
from random import randint

random_st_raw_x_paths, random_st_raw_y_paths = [], []
random_st_processed_x_paths, random_st_processed_y_paths = [], []

index = 0
count = 0

for paths in st_raw_x_images:
    # print(paths)
    index = randint(0, len(paths)-1)

    random_st_raw_x_paths.append(st_raw_x_images[count][index])
    random_st_raw_y_paths.append(st_raw_y_images[count][index])

    random_st_processed_x_paths.append(st_processed_x_images[count][index])
    random_st_processed_y_paths.append(st_processed_y_images[count][index])

    count += 1

count = 0
for i in range(len(random_st_raw_x_paths)):
    if (extractRhsString(random_st_raw_x_paths[i], "/t")[ : -4]
        !=
        extractRhsString(random_st_raw_y_paths[i], "/man_seg")[ : -4]
        ):
        count += 1

print(count, "image(s) do not match for RAW")

count = 0
for i in range(len(random_st_processed_x_paths)):
    if (extractRhsString(random_st_processed_x_paths[i], "_t")[ : -4]
        !=
        extractRhsString(random_st_processed_y_paths[i], "/man_seg")[ : -4]
```

```
        ):
        count += 1

print(count, "image(s) do not match for Processed")
```

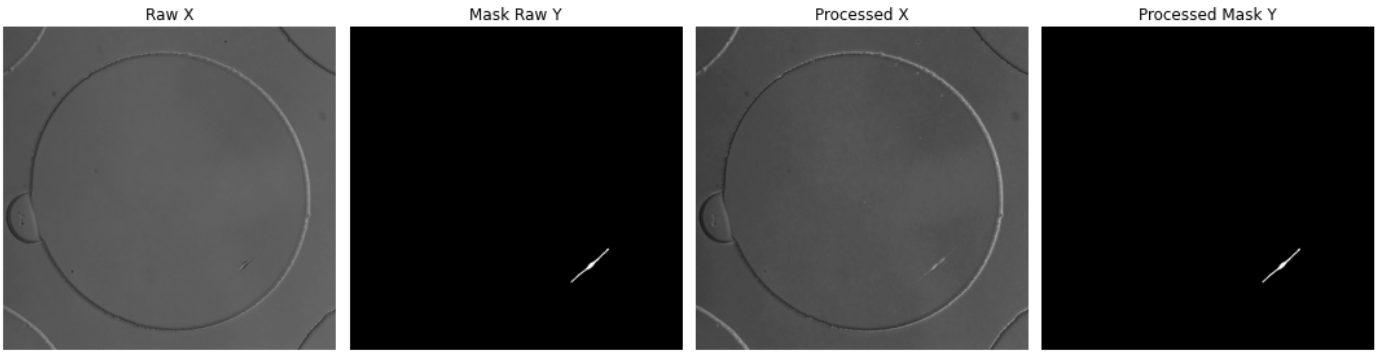
```
0 image(s) do not match for RAW
0 image(s) do not match for Processed
```

I should now be able to display the GT data together, and the ST data together. Let's verify that now:

In [ ]:
```
index = 1
array = [plt.imread(random_gt_raw_x_paths[index]), plt.imread(random_gt_raw_y_paths[inde
         plt.imread(random_gt_processed_x_paths[index]), plt.imread(random_gt_processed_
labels = ["Raw X", "Mask Raw Y", "Processed X", "Processed Mask Y"]

display(array, labels)
```



In [ ]:
```
index = 1
array = [plt.imread(random_st_raw_x_paths[index]), plt.imread(random_st_raw_y_paths[inde
         plt.imread(random_st_processed_x_paths[index]), plt.imread(random_st_processed_
labels = ["Raw X", "Mask Raw Y", "Processed X", "Processed Mask Y"]

display(array, labels)
```



Do Side By Side Predictions

Now that we have loaded our Raw and Processed images alongside one another, we can use the appropriate model to compare the results and see firsthand how we are doing.

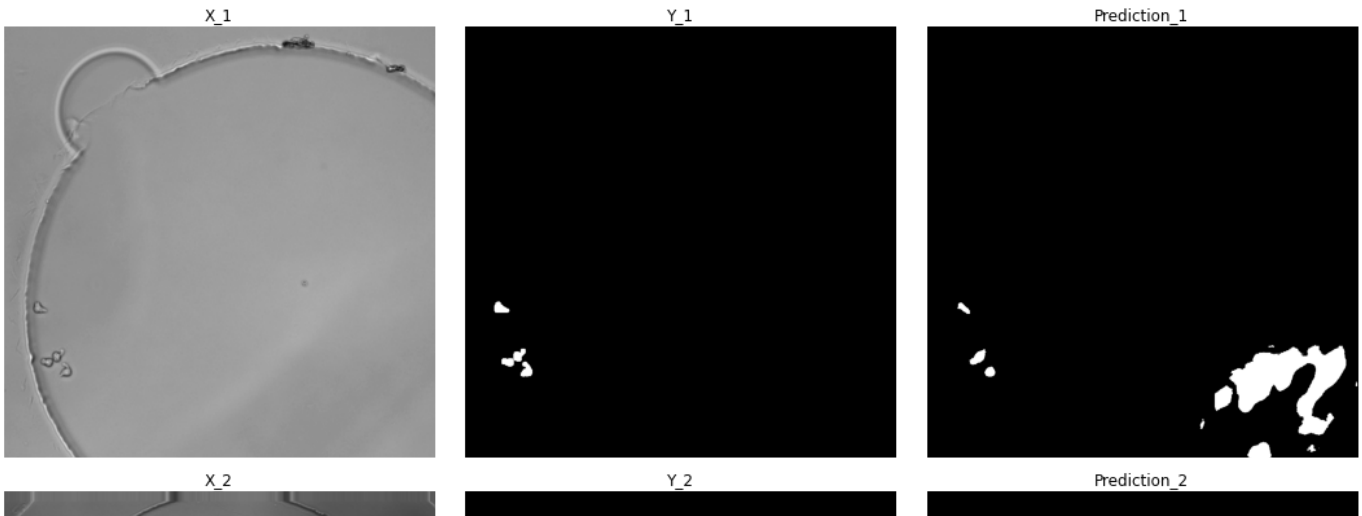
If we are feeling adventurous - we can also attempt to mix models and datasets and see what happens as well!

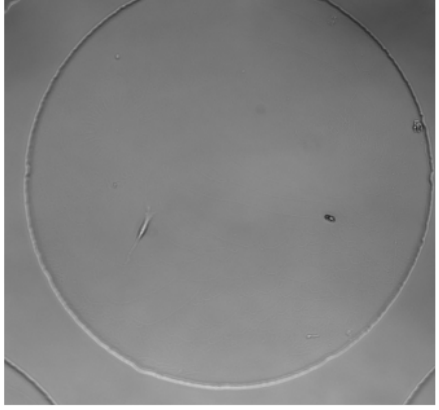
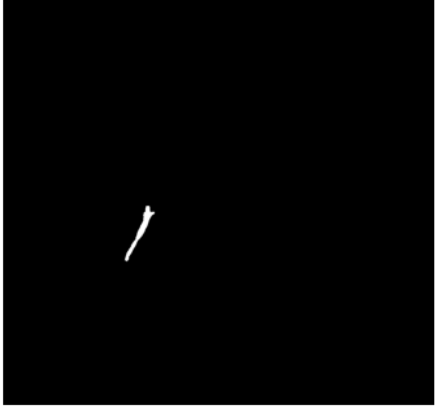
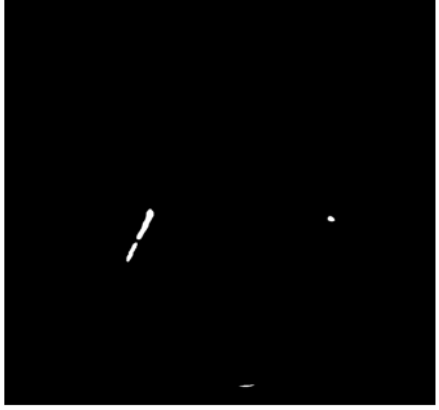
In [ ]:
```
# make directories for images
image_path = getcwd() + google_drive_path + "/COMP700_Images/"

tryMakeDirectories(image_path, ["Matplotlib_Figures", "UNET_Model_Predictions_Overall"])
```

GT Raw Predictions

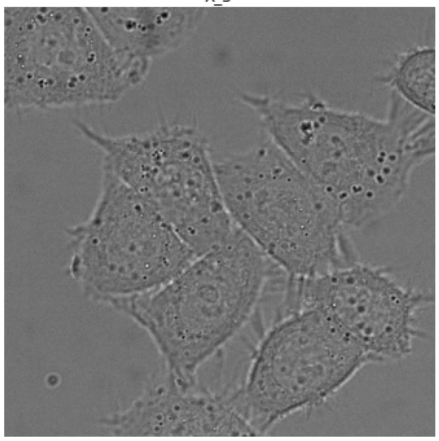
In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/GT_256/"
model_name = "gt_256_model_3"
patch_size = 256

gt_raw_array, gt_raw_labels = conductBulkPrediction(random_gt_raw_x_paths,
                                                    random_gt_raw_y_paths,
                                                    patch_size = patch_size,
                                                    path = path,
                                                    model_name = model_name,
                                                    threshold = 0.5
                                                    )

# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(gt_raw_array, gt_raw_labels, figsize=(15, 50))
```

```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 3s 3s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 5s 5s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 683ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 4s 4s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 3s 3s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 718ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 678ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 687ms/step
```



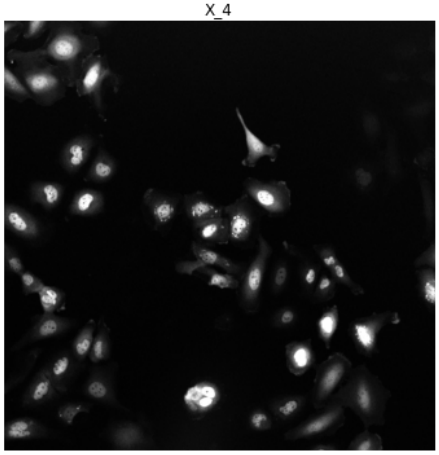
X_3



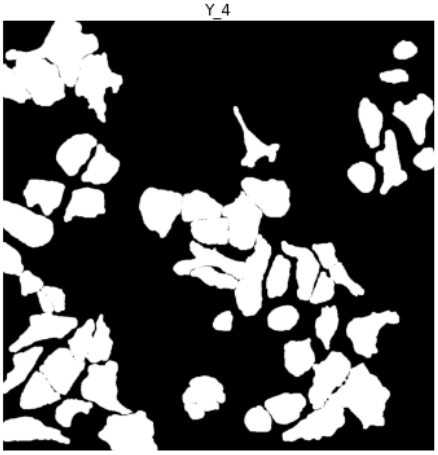
Y_3

Prediction_3

X_4

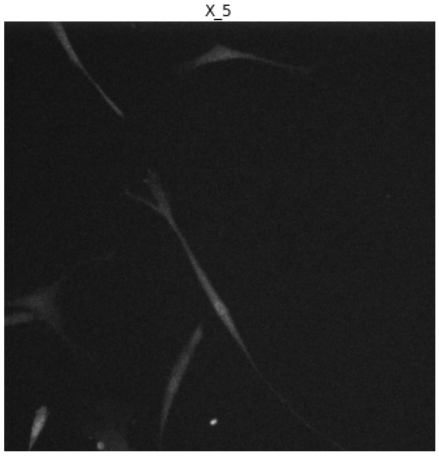


Y_4



Prediction_4

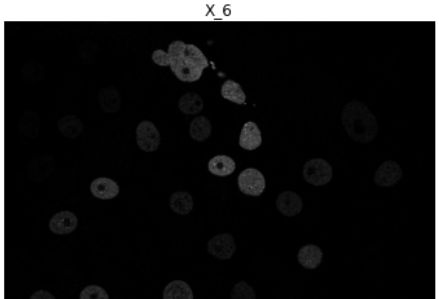
X_5



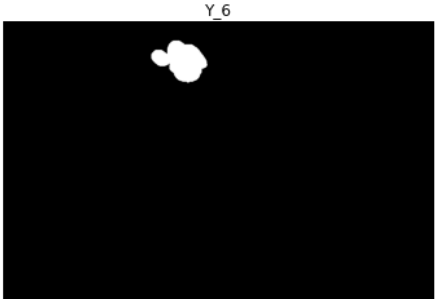
Y_5

Prediction_5

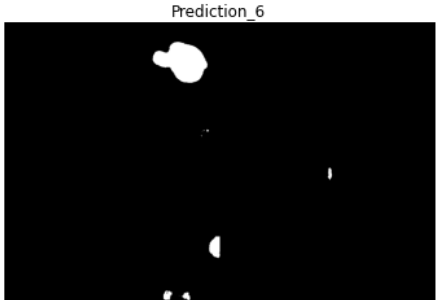
X_6



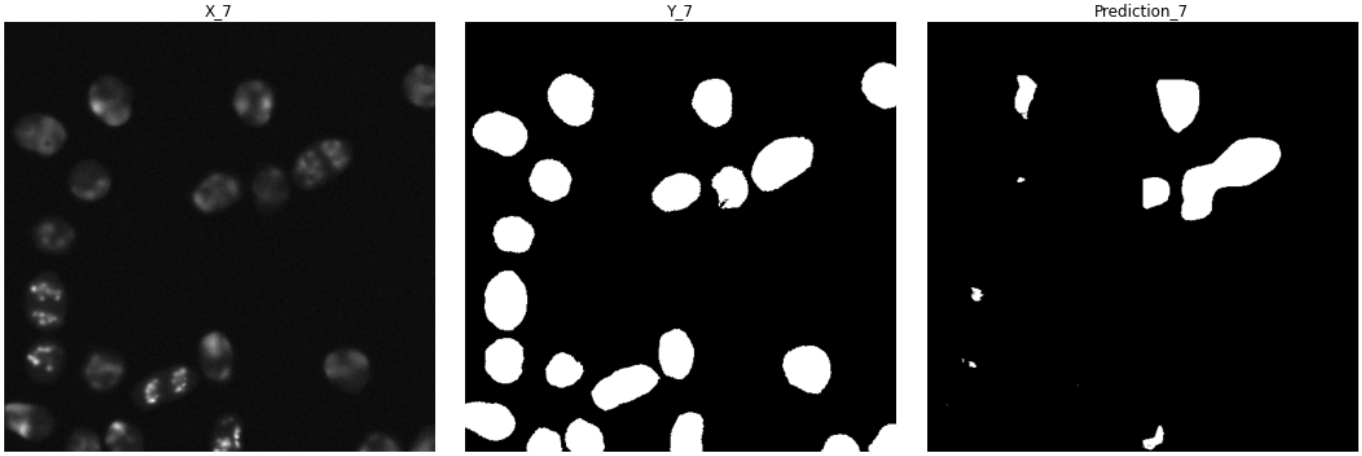
Y_6



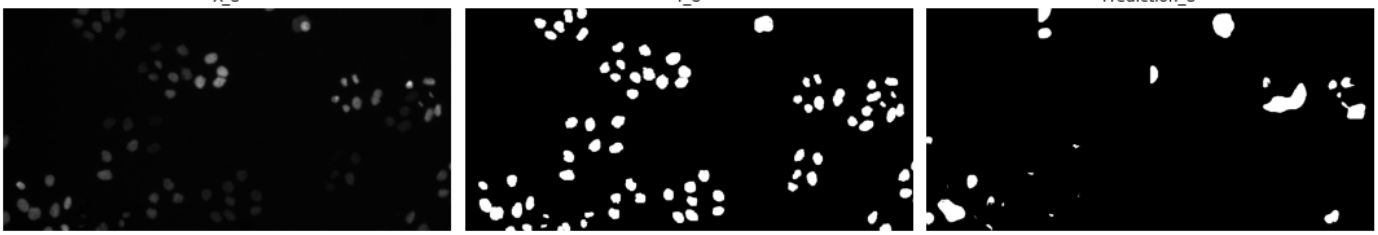
Prediction_6



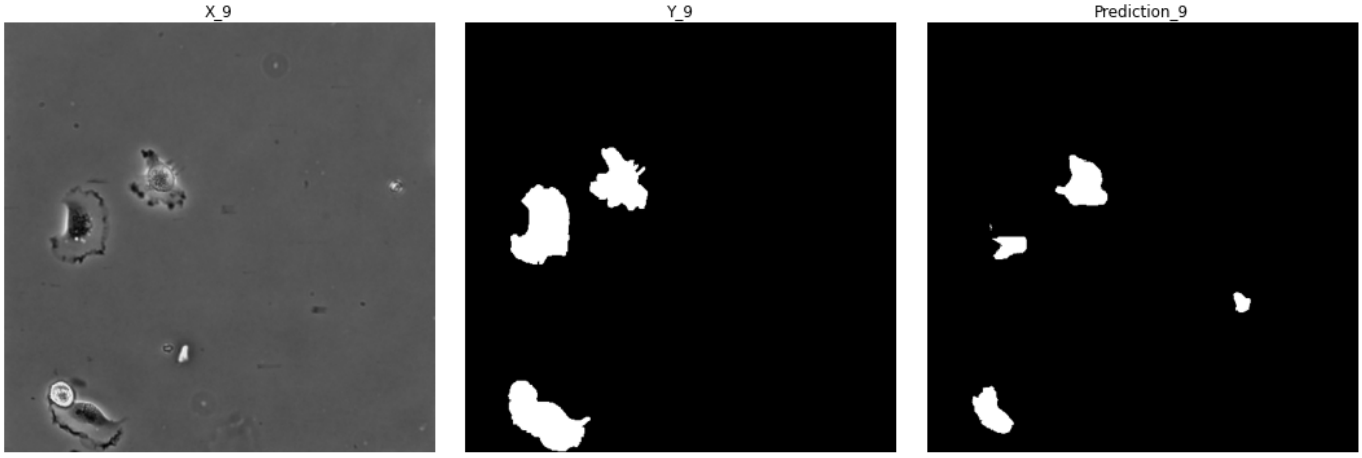
X_7 Y_7 Prediction_7



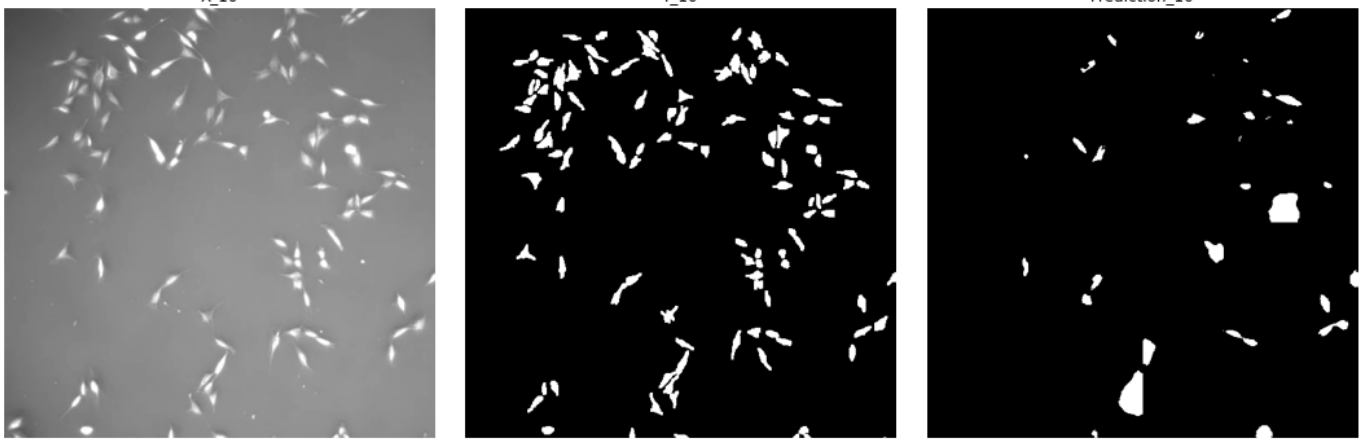
X_8 Y_8 Prediction_8



X_9 Y_9 Prediction_9



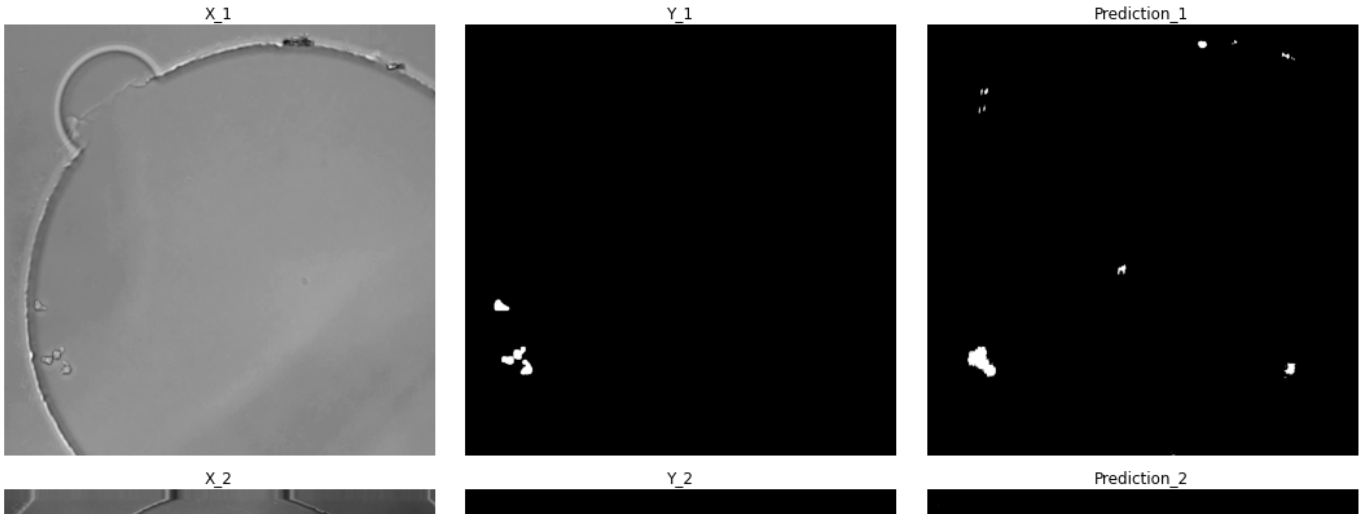
X_10 Y_10 Prediction_10

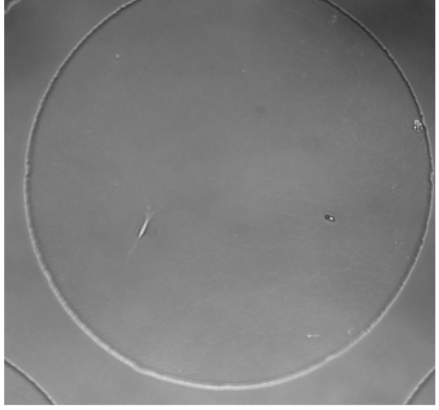
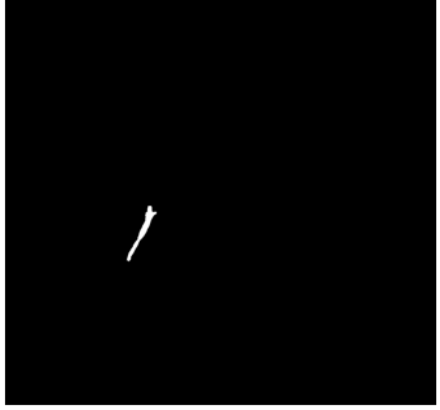


GT Processed Predictions

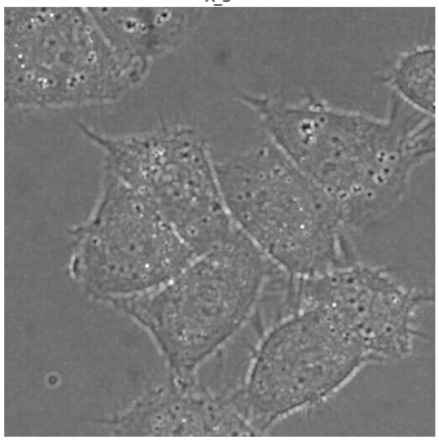
In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/Processed_GT_256/"
model_name = "processed_gt_images_256_model_3"
patch_size = 256

gt_processed_array, gt_processed_labels = conductBulkPrediction(random_gt_processed_x_pa
                                                                random_gt_processed_y_pa
                                                                patch_size = patch_size,
                                                                path = path,
                                                                model_name = model_name,
                                                                threshold = 0.5
                                                                )


# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(gt_processed_array, gt_processed_labels, figsize=(15, 50))
```

```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 3s 3s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 693ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 4s 4s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 699ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 706ms/step
```



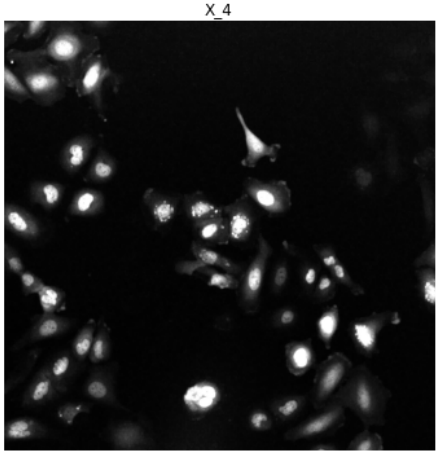

X_3



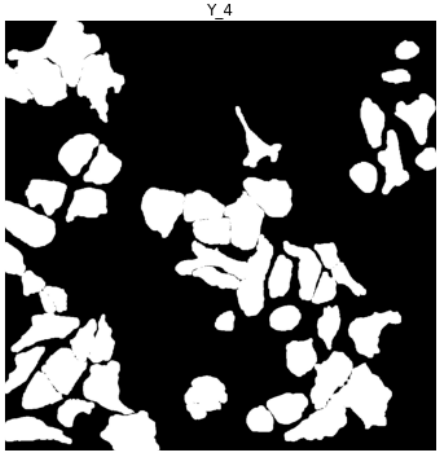
Y_3

Prediction_3

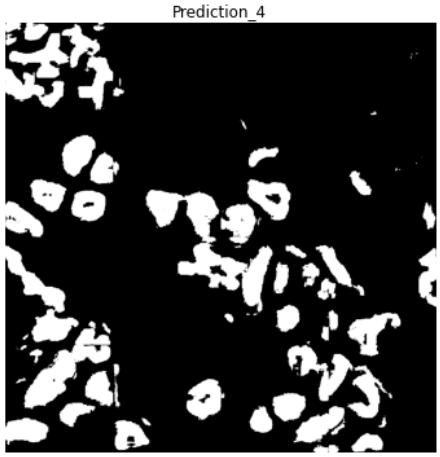
X_4



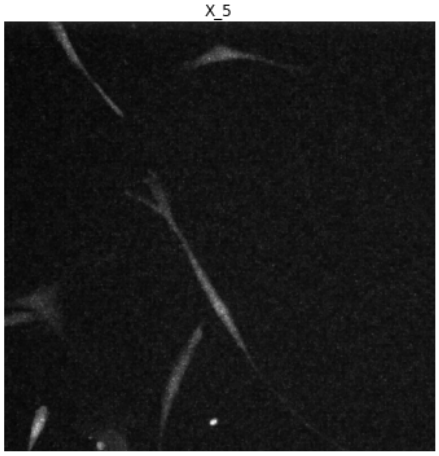
Y_4



Prediction_4



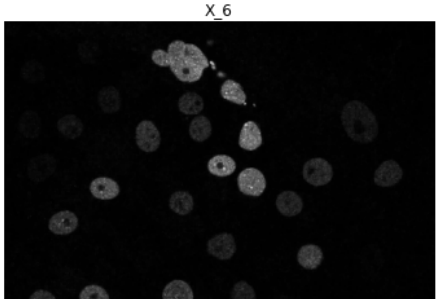
X_5



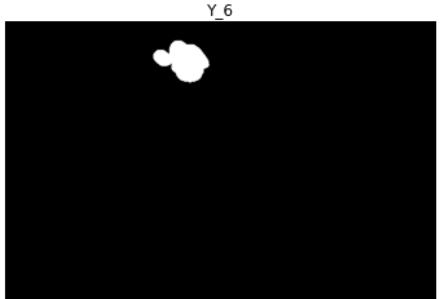
Y_5

Prediction_5

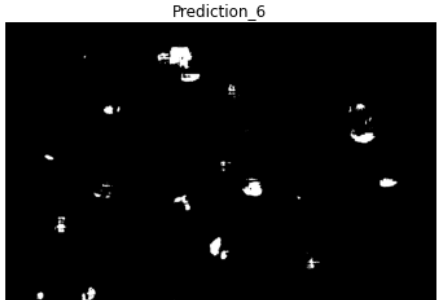
X_6



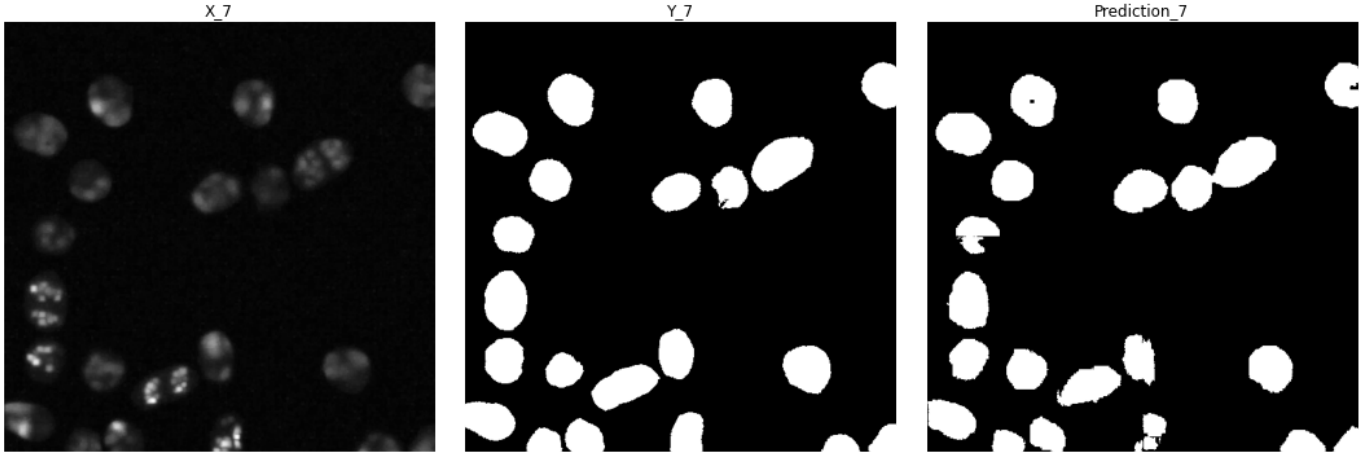
Y_6



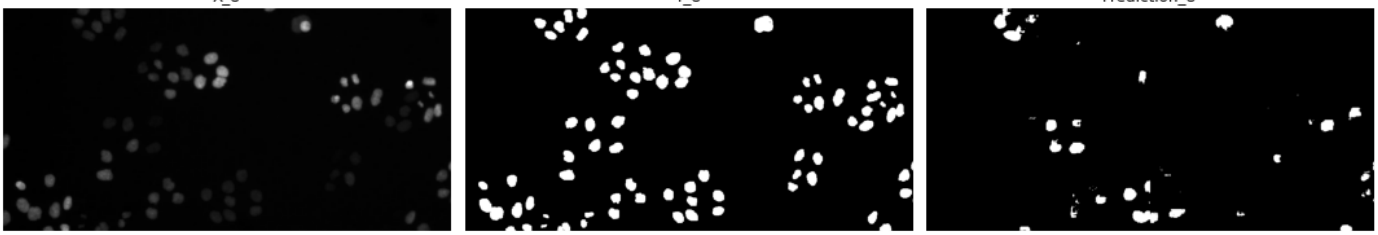
Prediction_6



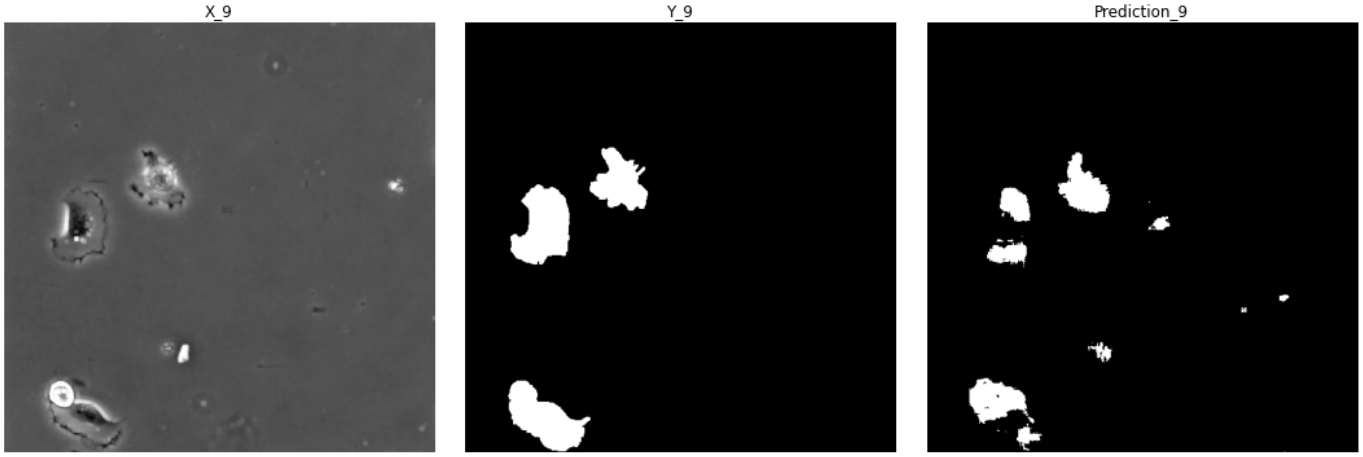
X_7 Y_7 Prediction_7



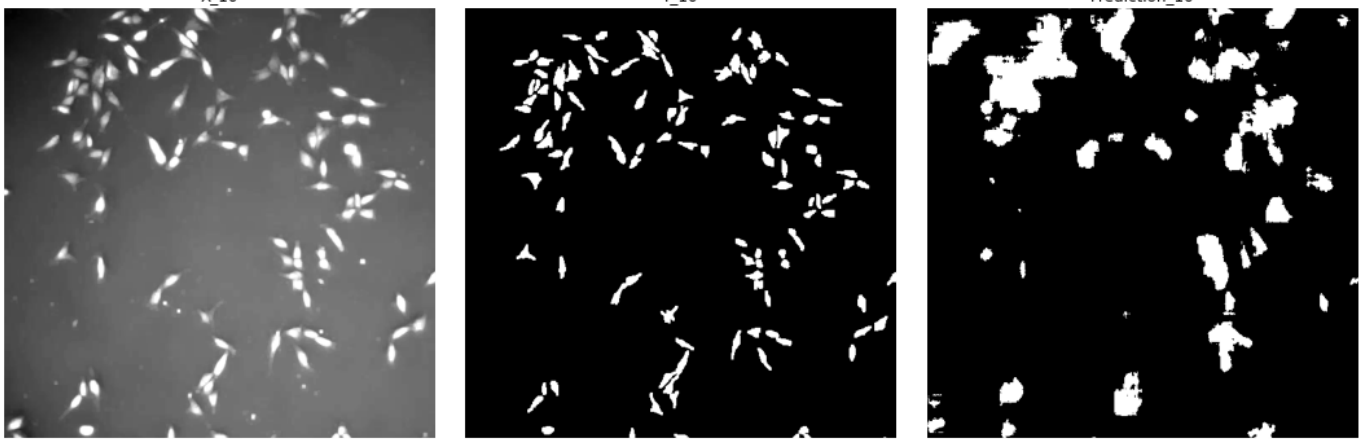
X_8 Y_8 Prediction_8



X_9 Y_9 Prediction_9



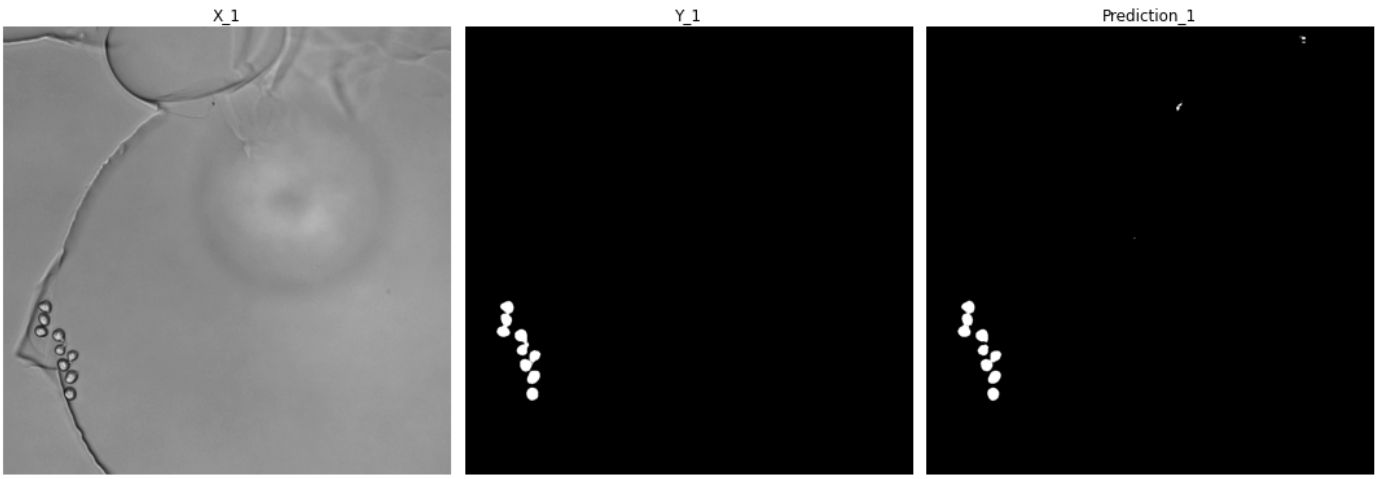
X_10 Y_10 Prediction_10

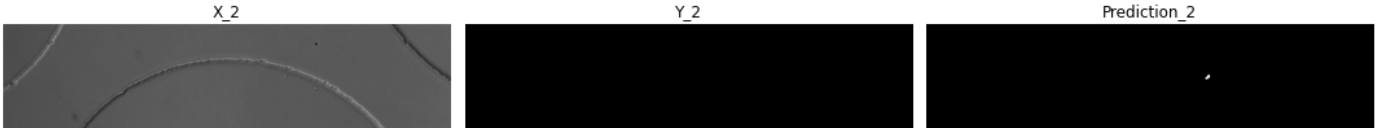


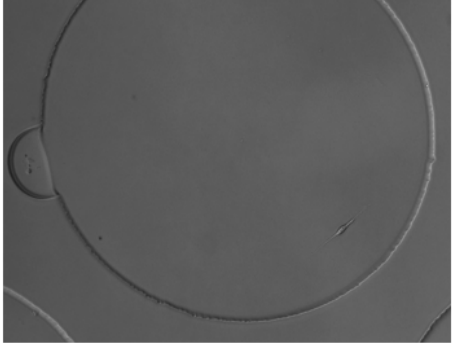
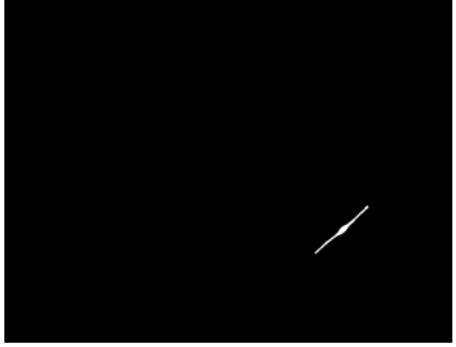
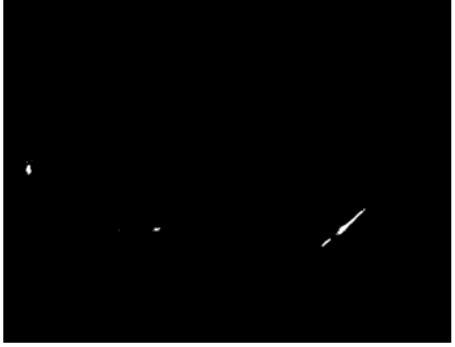
ST Raw Predictions

In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/ST_256/"
model_name = "st_256_model_3"
patch_size = 256

st_raw_array, st_raw_labels = conductBulkPrediction(random_st_raw_x_paths,
                                                    random_st_raw_y_paths,
                                                    patch_size = patch_size,
                                                    path = path,
                                                    model_name = model_name,
                                                    threshold = 0.5
                                                    )

# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(st_raw_array, st_raw_labels, figsize=(15, 50))
```

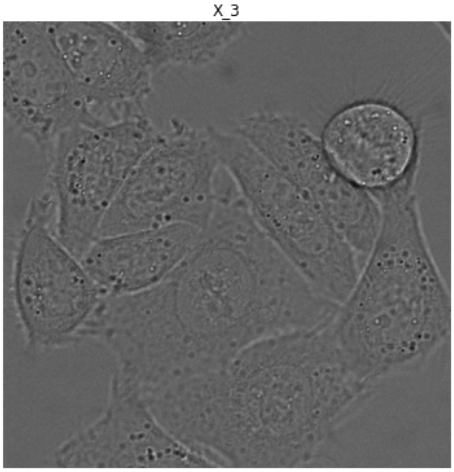
```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 663ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 640ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 669ms/step
```





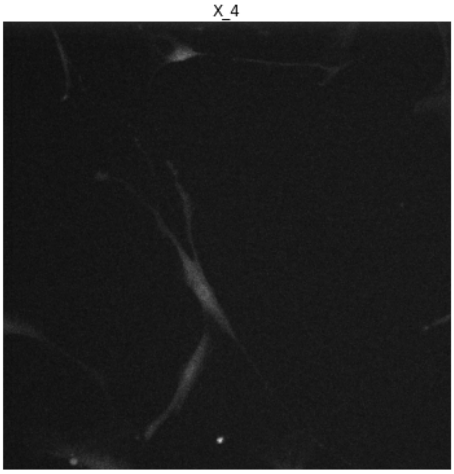
X_3



Y_3

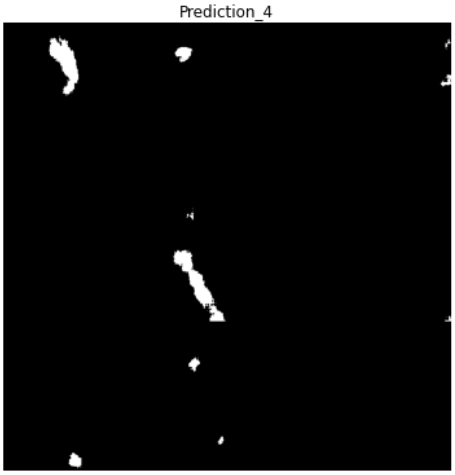
Prediction_3

X_4

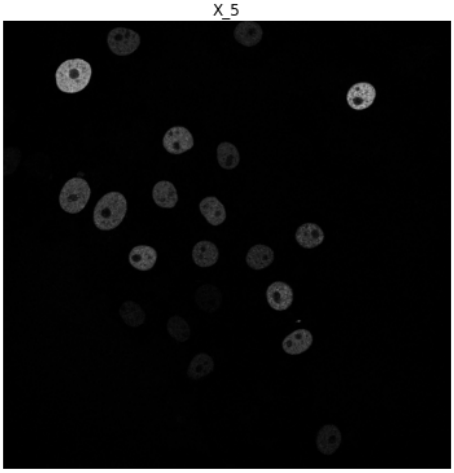


Y_4

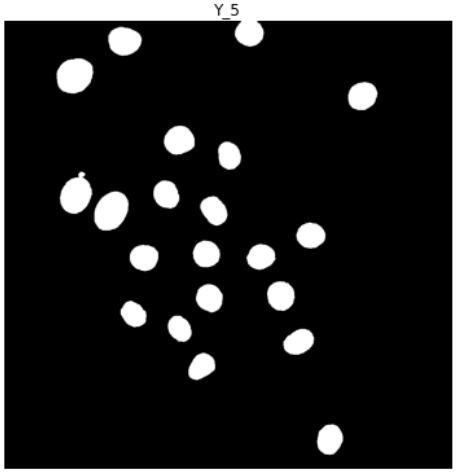
Prediction_4



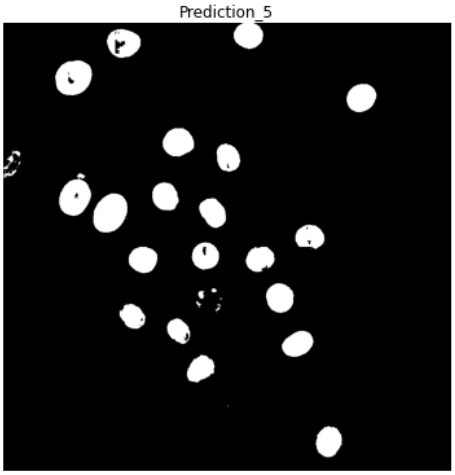
X_5

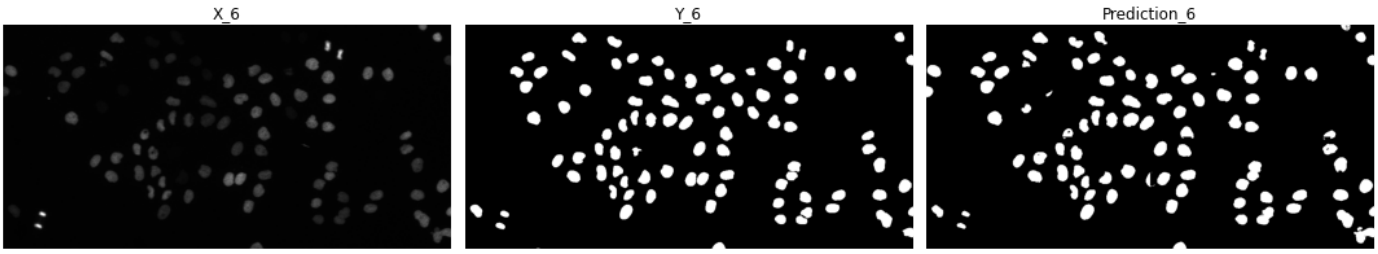


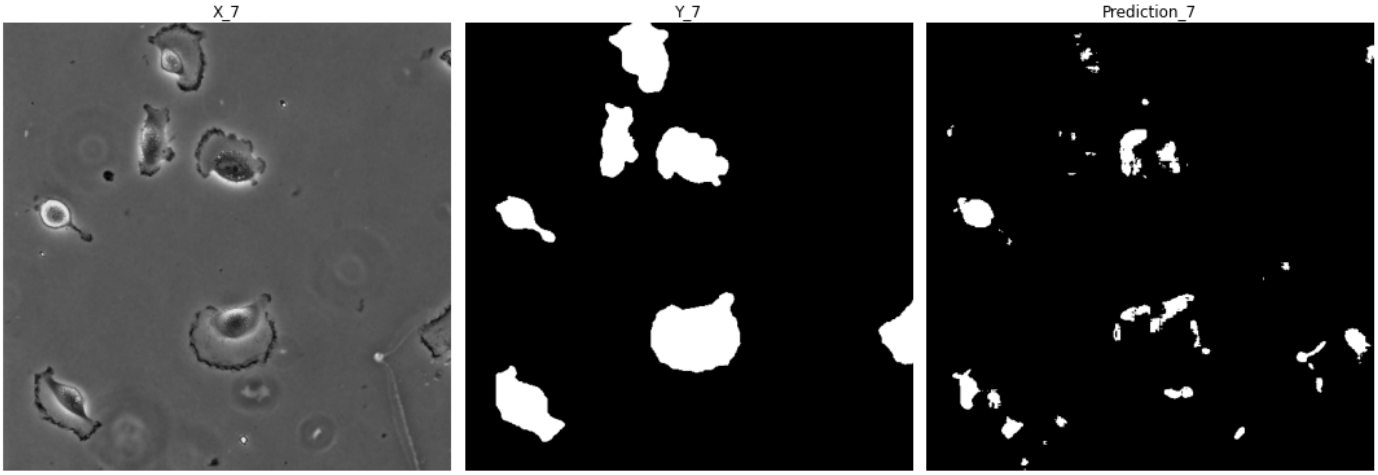
Y_5

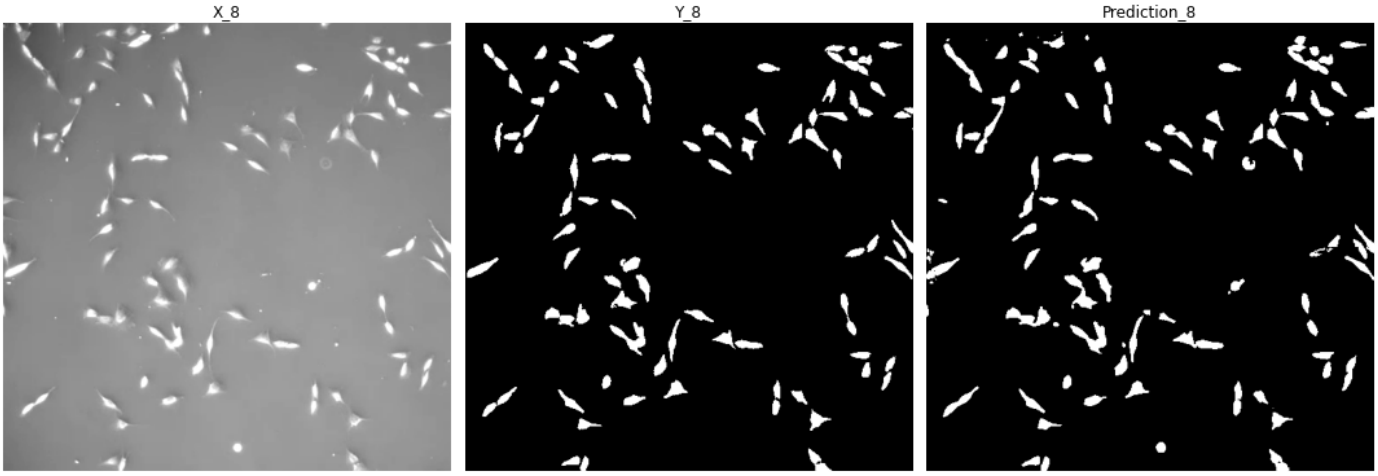


Prediction_5







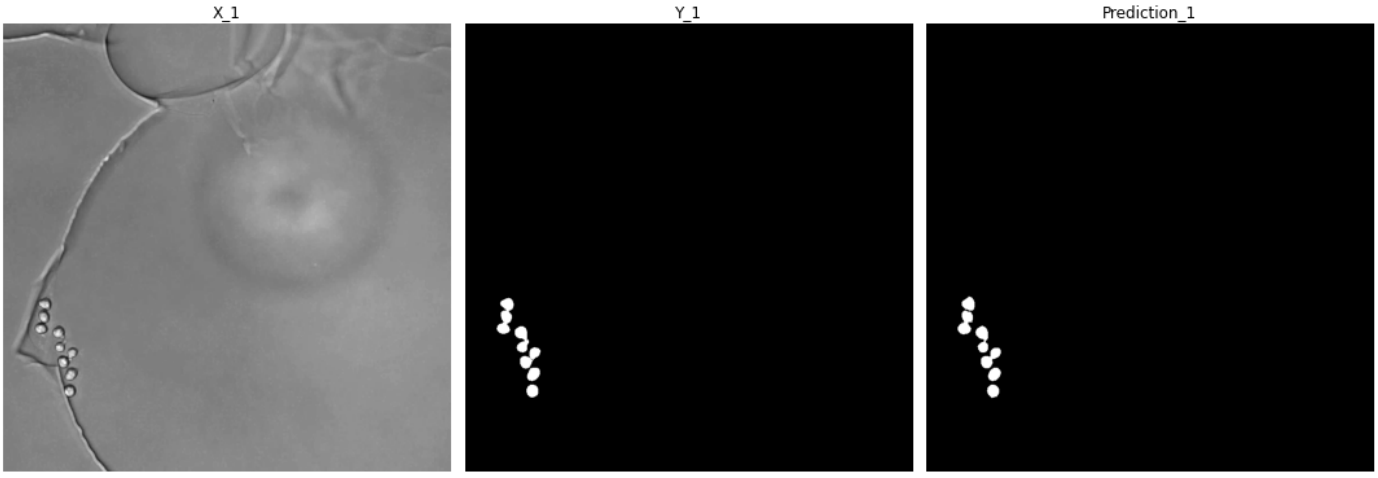


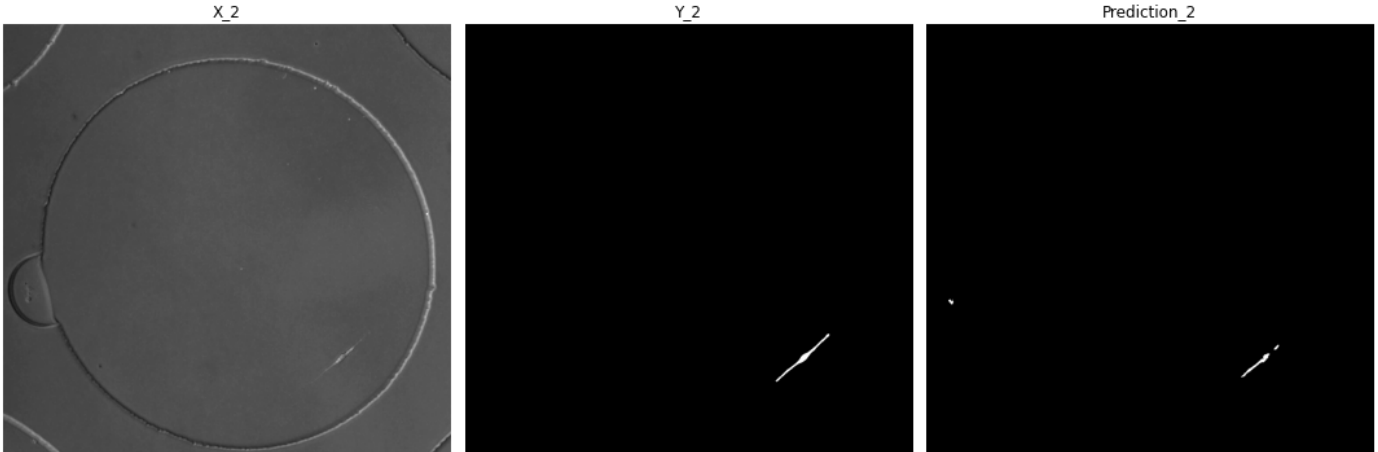
ST Processed Predictions

In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/Processed_ST_256/"
model_name = "processed_st_images_256_model_3"
patch_size = 256

st_processed_array, st_processed_labels = conductBulkPrediction(random_st_processed_x_pa
                                                                random_st_processed_y_pa
```

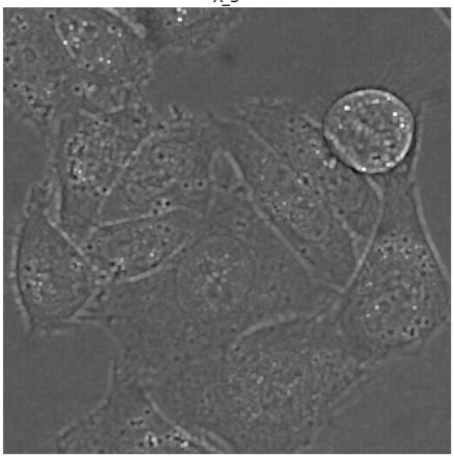
```
                                                patch_size = patch_size,
                                                path = path,
                                                model_name = model_name,
                                                threshold = 0.5
                                                )

# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(st_processed_array, st_processed_labels, figsize=(15, 50))
```

```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 670ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 675ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 674ms/step
```



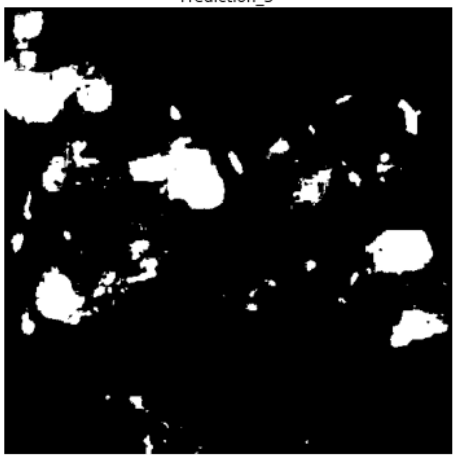


X_3

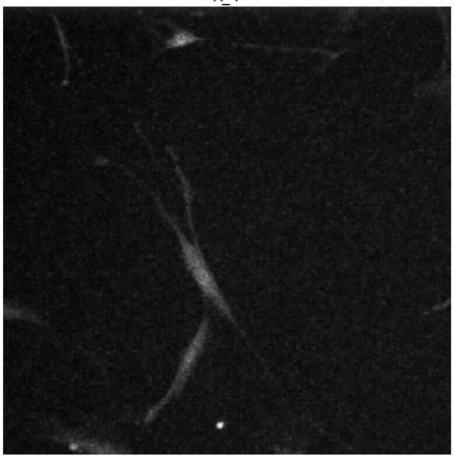


Y_3

Prediction_3



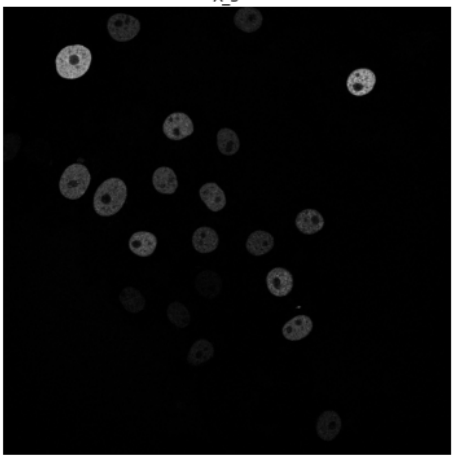
X_4



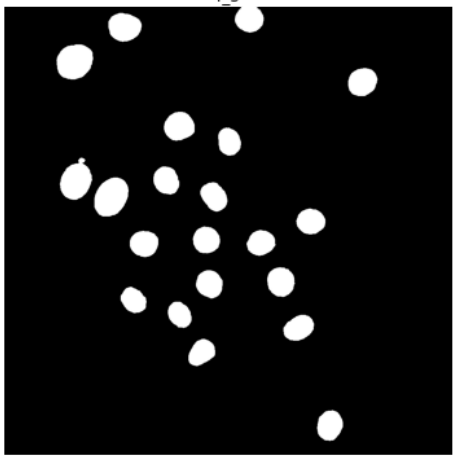
Y_4

Prediction_4

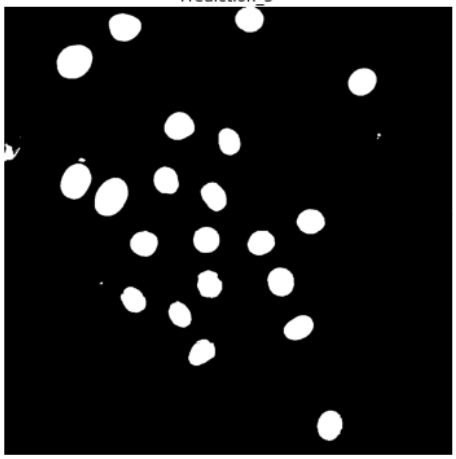
X_5



Y_5

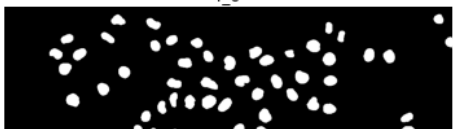


Prediction_5

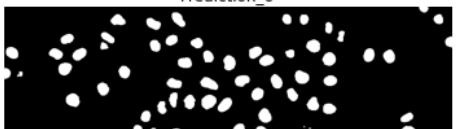


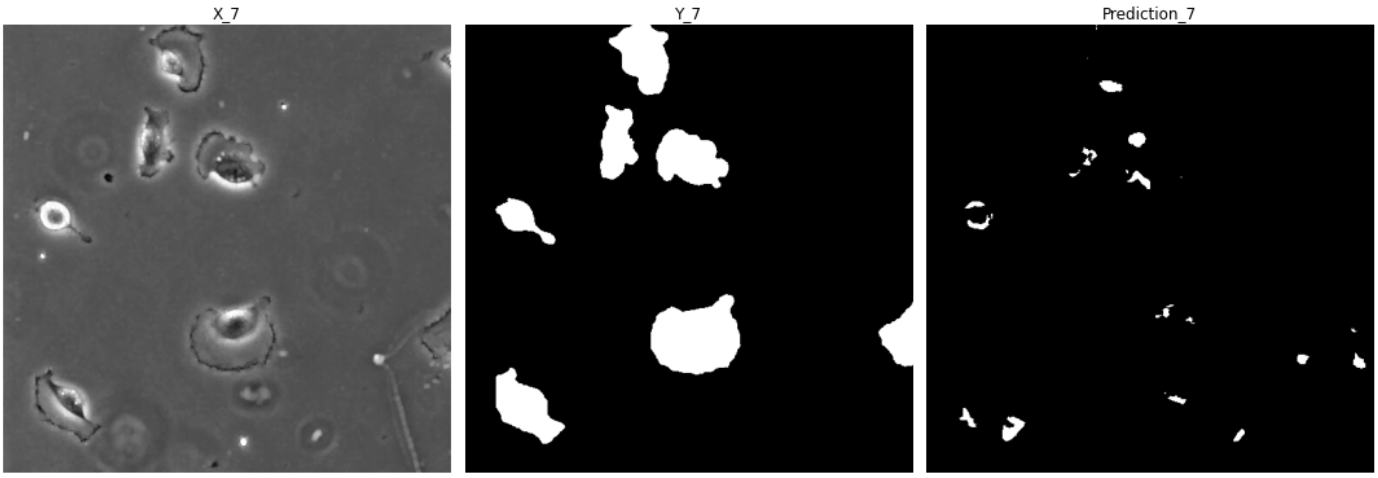
X_6

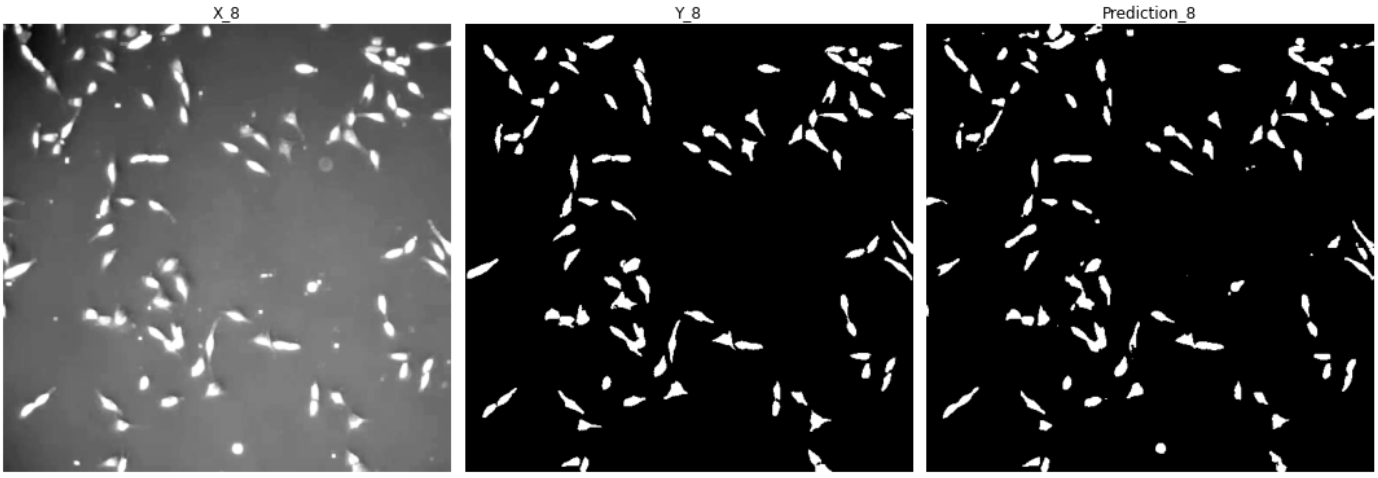
Y_6



Prediction_6







Great! Let us now compare all the GT data together and the ST data together:

GT Altogether:

In [ ]:
```
# gt_processed_array, gt_processed_labels
# gt_raw_array, gt_raw_labels

gt_array = []
gt_labels = []

# runs 10 times
for i in range(len(gt_raw_array)):
    gt_array.append(
        [
            gt_raw_array[i][0], gt_raw_array[i][1], gt_raw_array[i][2],
            gt_processed_array[i][0], gt_processed_array[i][1], gt_processed_array[i][2]
        ]
    )
    gt_labels.append(
```

```
        [
            # gt_raw_labels[i][0], gt_raw_labels[i][1], gt_raw_labels[i][2],
            # gt_processed_labels[i][0], gt_processed_labels[i][1], gt_processed_labels[
            "Raw X", "Raw Y", "Predicted Raw Y",
            "Processed X", "Processed Y", "Predicted Processed Y"
        ]

    )

# display2DPlot(array, labels, figsize=(30, 50))
```

In [ ]:
```
location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
save2DPlot(gt_array, gt_labels, location, "GT_all_predictions.png", figsize=(30,50))
```

Output hidden; open in https://colab.research.google.com to view.

ST Altogether:

In [ ]:
```
# st_processed_array, st_processed_labels
# st_raw_array, st_raw_labels
st_array = []
st_labels = []

# runs 10 times
for i in range(len(st_raw_array)):
    st_array.append(
        [
            st_raw_array[i][0], st_raw_array[i][1], st_raw_array[i][2],
            st_processed_array[i][0], st_processed_array[i][1], st_processed_array[i][2]
        ]
    )
    st_labels.append(
        [
            # st_raw_labels[i][0], st_raw_labels[i][1], st_raw_labels[i][2],
            # st_processed_labels[i][0], st_processed_labels[i][1], st_processed_labels[
            "Raw X", "Raw Y", "Predicted Raw Y",
            "Processed X", "Processed Y", "Predicted Processed Y"
        ]

    )

# display2DPlot(array, labels, figsize=(30, 50))
```

In [ ]:
```
location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
save2DPlot(st_array, st_labels, location, "ST_all_predictions.png", figsize=(30,50))
```

Output hidden; open in https://colab.research.google.com to view.

Fantastic! Surprisingly, the RAW data yields a better prediction than my processed images!

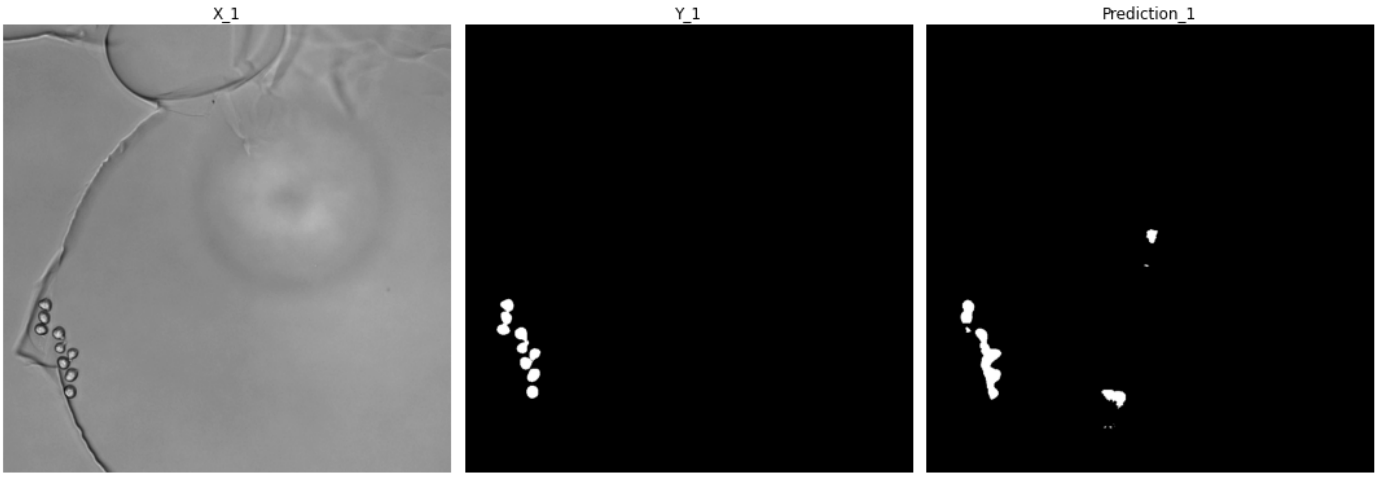
Results of GT model on ST dataset:

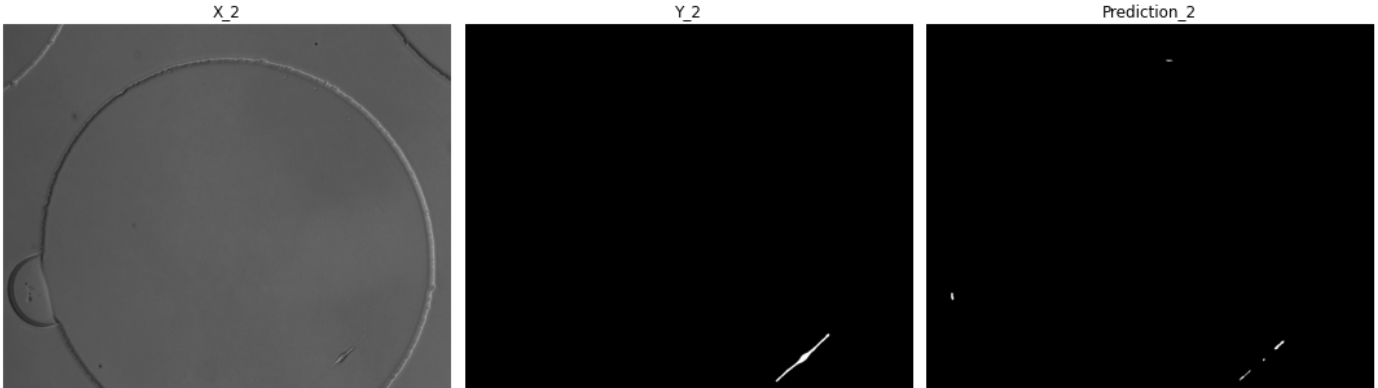
ST Raw Predictions

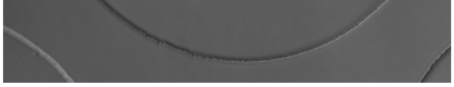
In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/GT_256/"
model_name = "gt_256_model_3"
patch_size = 256
```

```
st_raw_array, st_raw_labels = conductBulkPrediction(random_st_raw_x_paths,
                                                    random_st_raw_y_paths,
                                                    patch_size = patch_size,
                                                    path = path,
                                                    model_name = model_name,
                                                    threshold = 0.5
                                                    )

# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(st_raw_array, st_raw_labels, figsize=(15, 50))
```

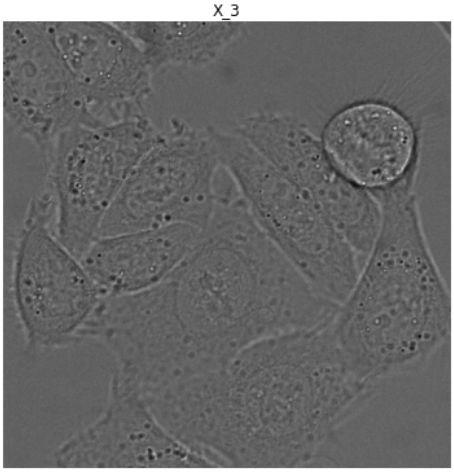
```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 4s 4s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 641ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 663ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 680ms/step
```







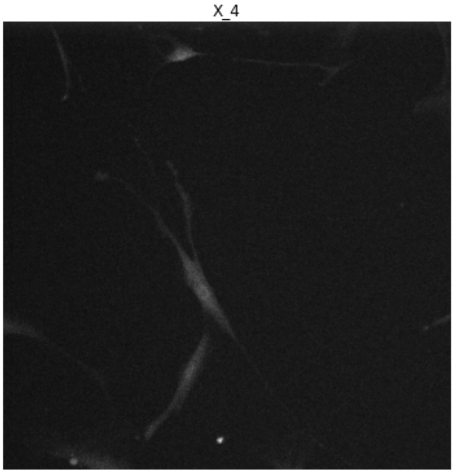
X_3



Y_3

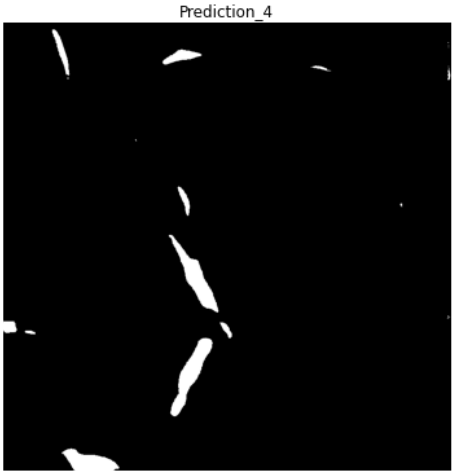
Prediction_3

X_4

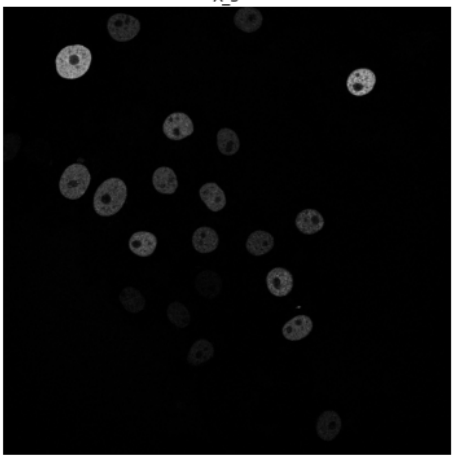


Y_4

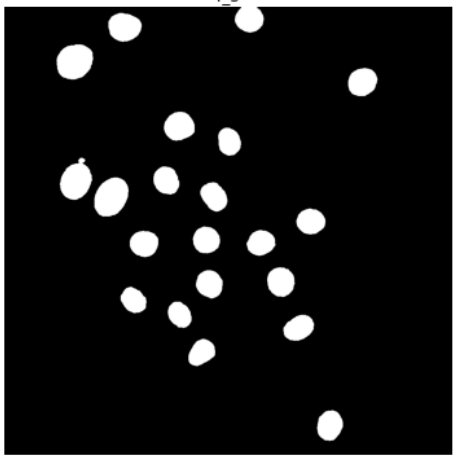
Prediction_4



X_5



Y_5

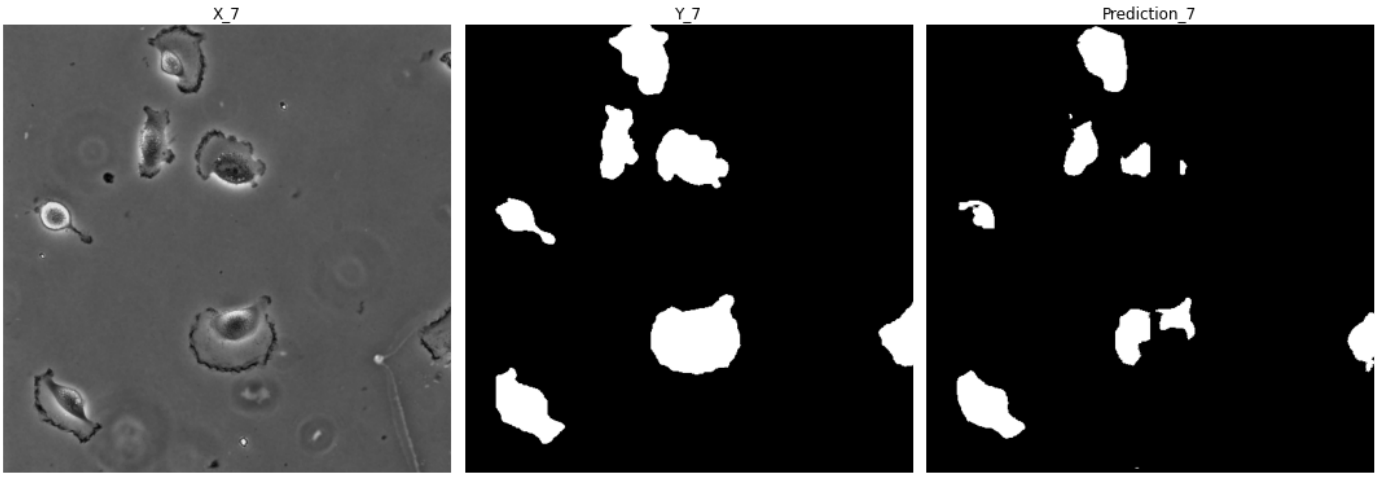


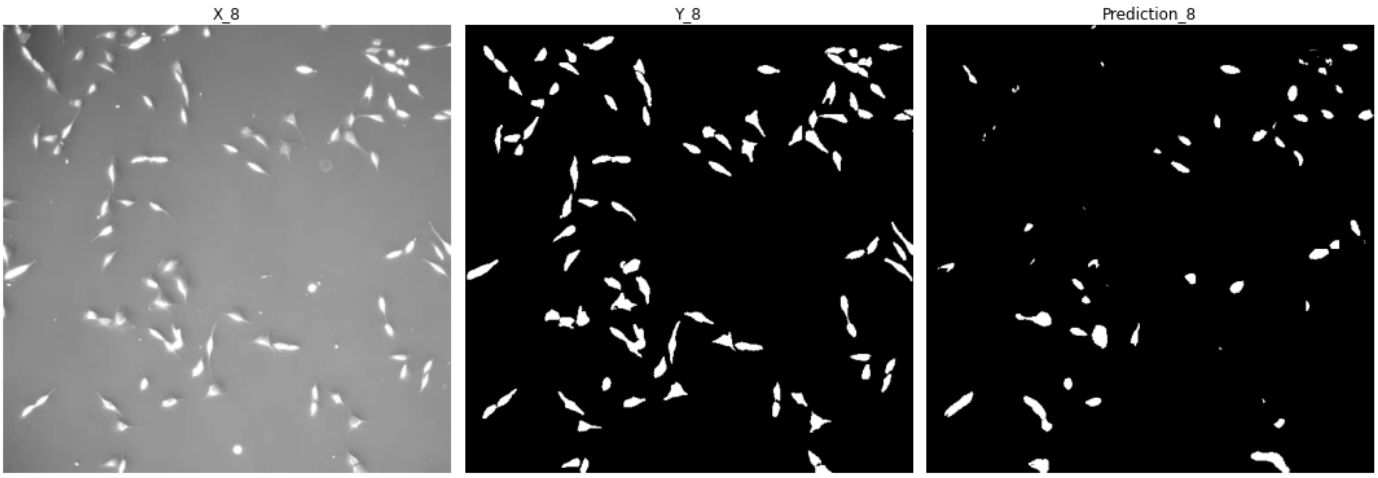
Prediction_5

X_6

Y_6

Prediction_6





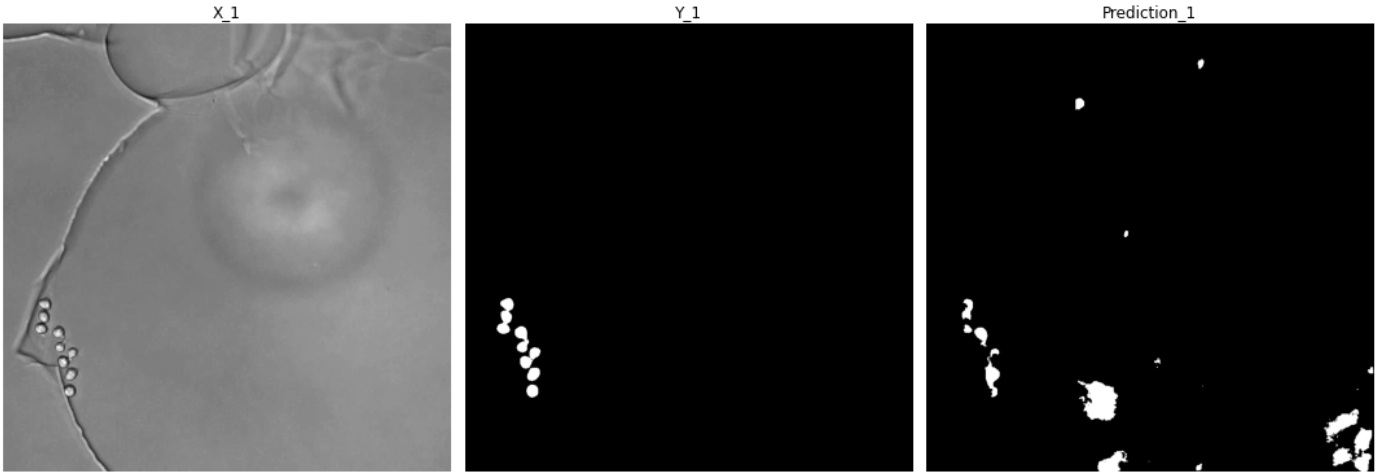
That is not a good prediction! Compared to ST Model on ST RAW - this is poorer quality

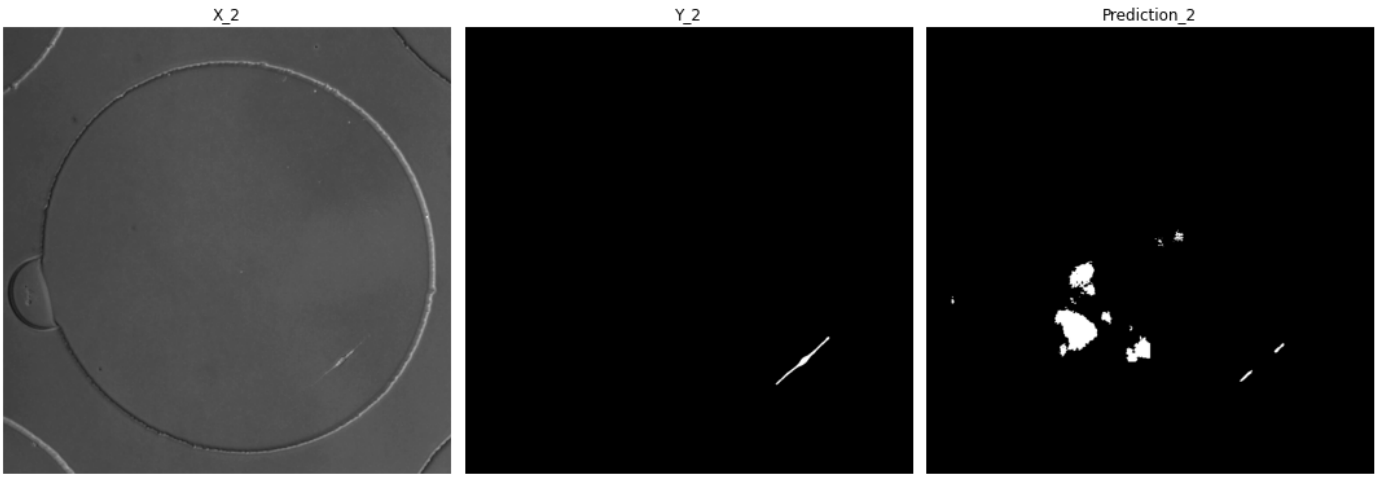
ST Processed Predictions

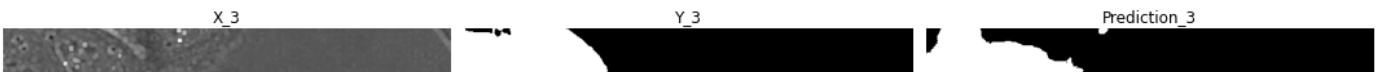
In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/GT_256/"
model_name = "gt_256_model_3"
patch_size = 256

st_processed_array, st_processed_labels = conductBulkPrediction(random_st_processed_x_pa
                                                                 random_st_processed_y_pa
                                                                 patch_size = patch_size,
                                                                 path = path,
                                                                 model_name = model_name,
                                                                 threshold = 0.5
                                                                 )
```

```
# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(st_processed_array, st_processed_labels, figsize=(15, 50))
```

```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 670ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 683ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 675ms/step
```

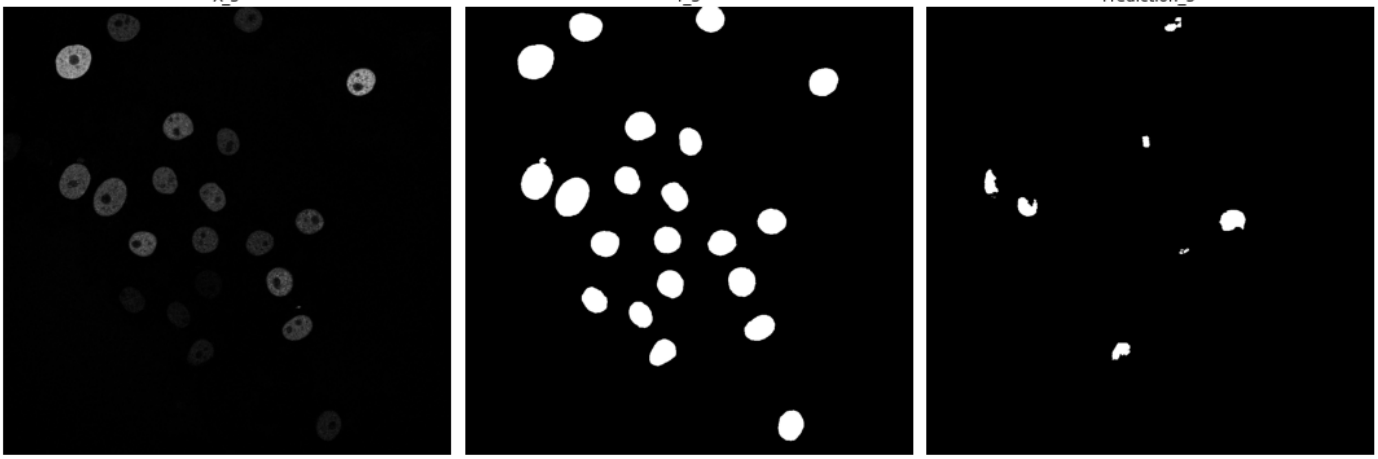




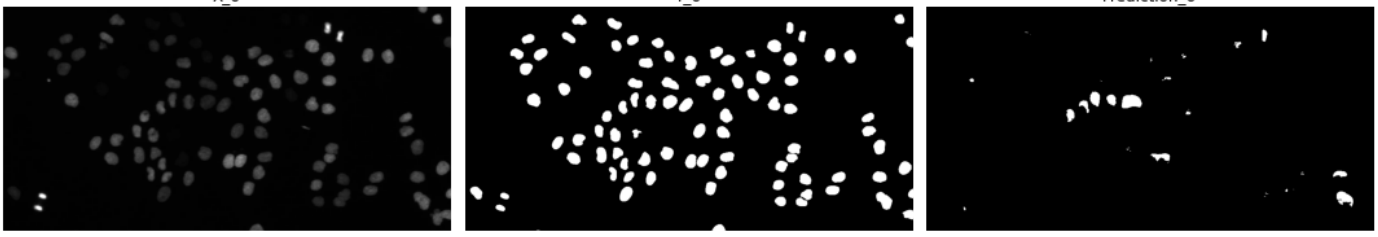


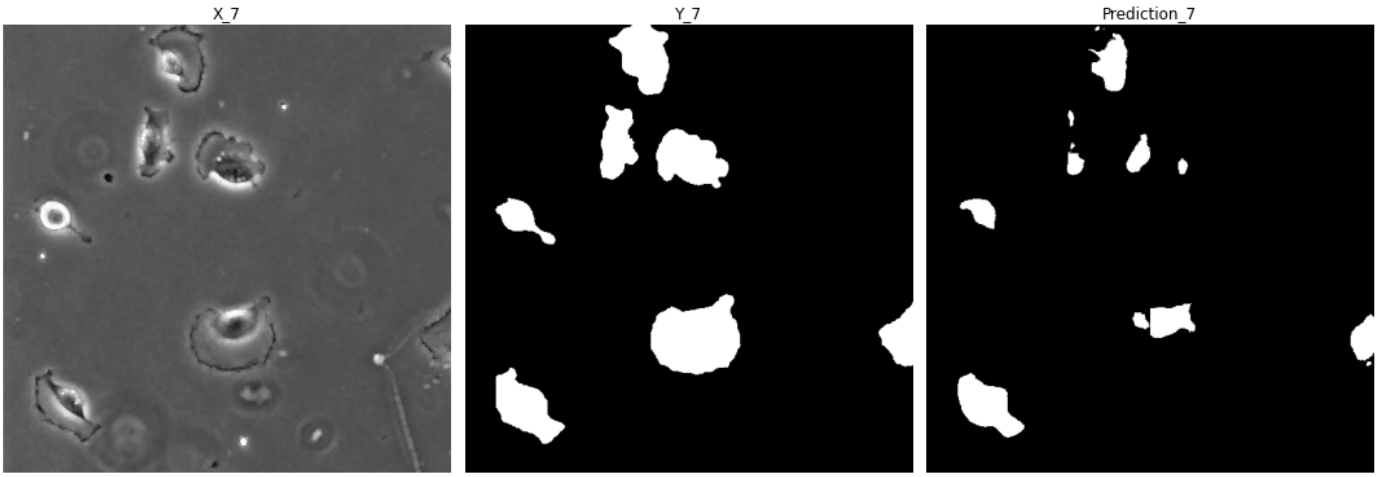
X_4 Y_4 Prediction_4

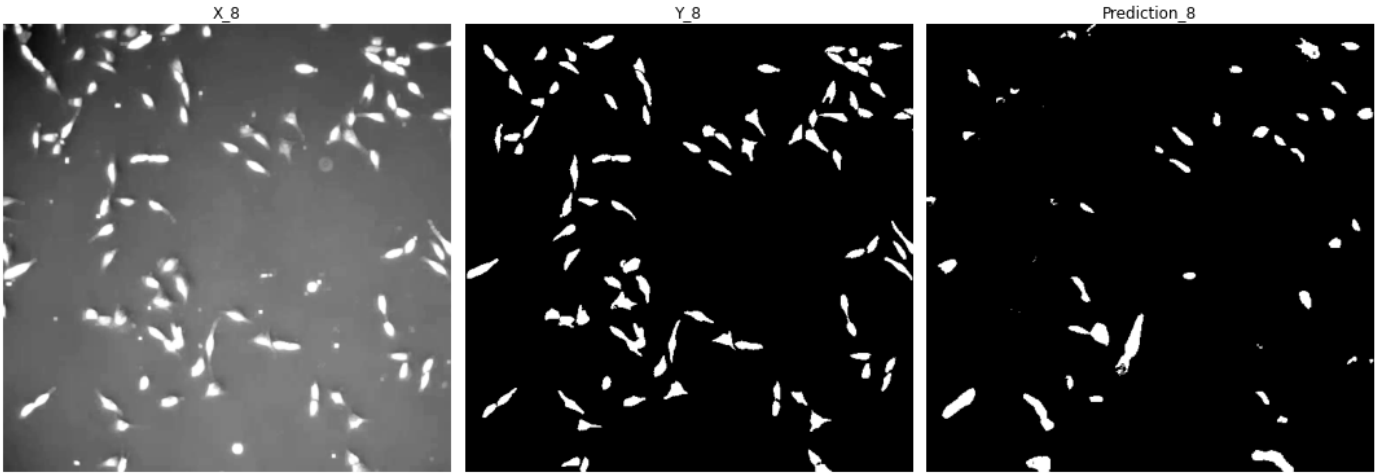
X_5 Y_5 Prediction_5



X_6 Y_6 Prediction_6







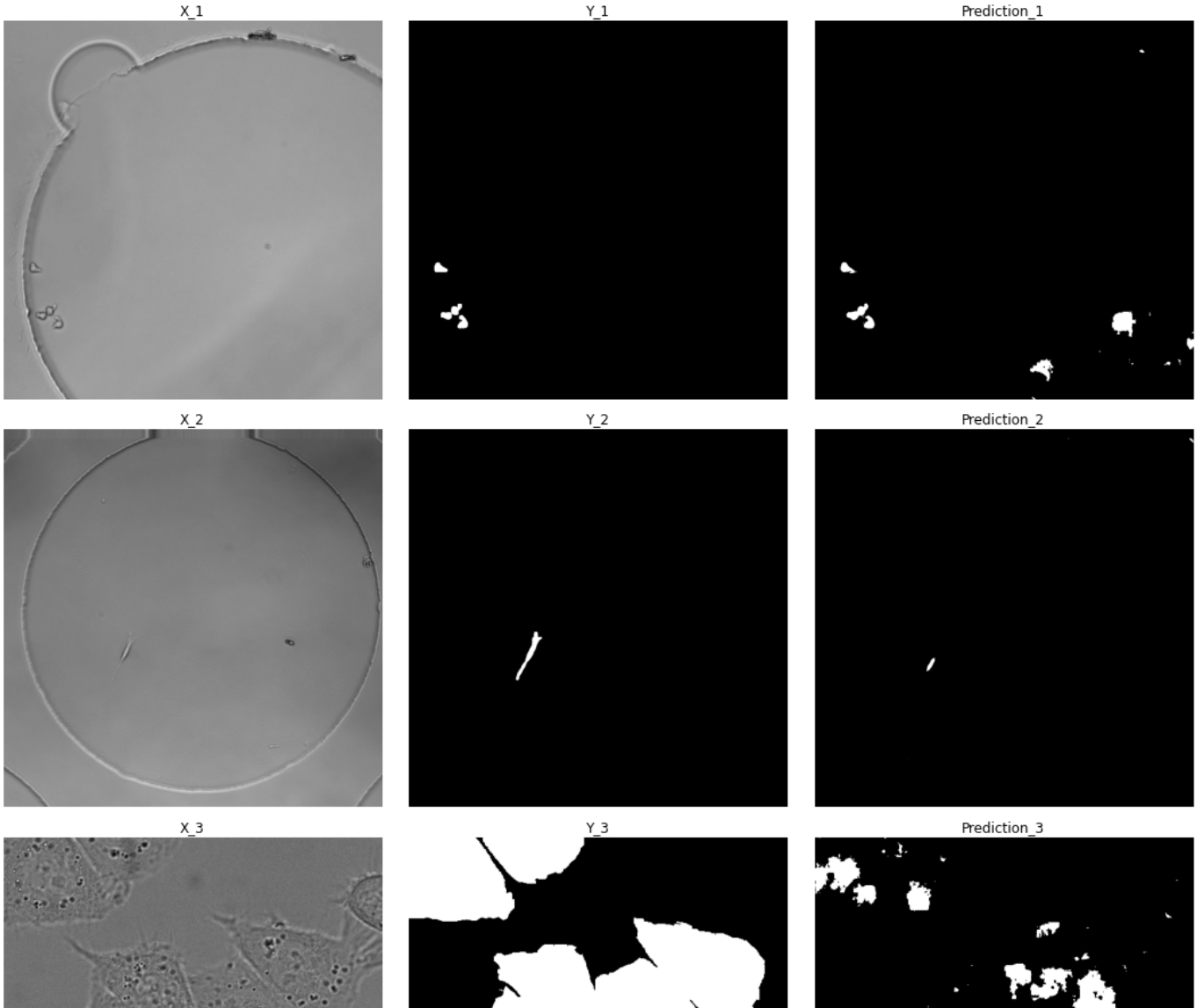
That is not a good prediction! Compared to ST Model on ST Processed - this is poorer quality

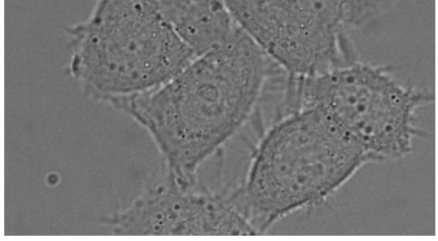
Results of ST model on GT dataset

GT Raw Predictions

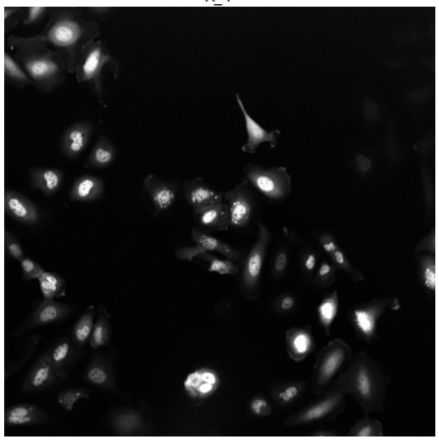
In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/Processed_ST_256/"
model_name = "processed_st_images_256_model_3"
patch_size = 256

gt_raw_array, gt_raw_labels = conductBulkPrediction(random_gt_raw_x_paths,
                                                    random_gt_raw_y_paths,
                                                    patch_size = patch_size,
                                                    path = path,
                                                    model_name = model_name,
                                                    threshold = 0.5
                                                    )

# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(gt_raw_array, gt_raw_labels, figsize=(15, 50))
```

```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 686ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 688ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 687ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 722ms/step
```

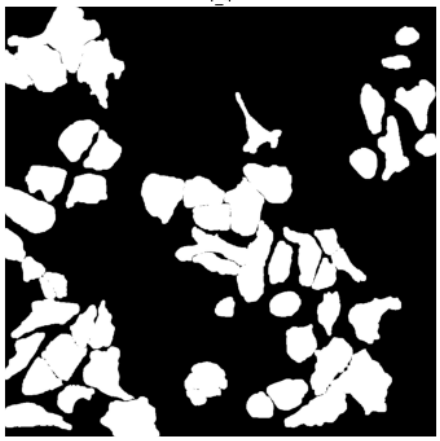




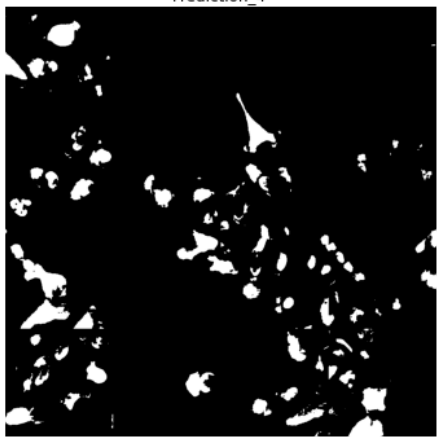
X_4



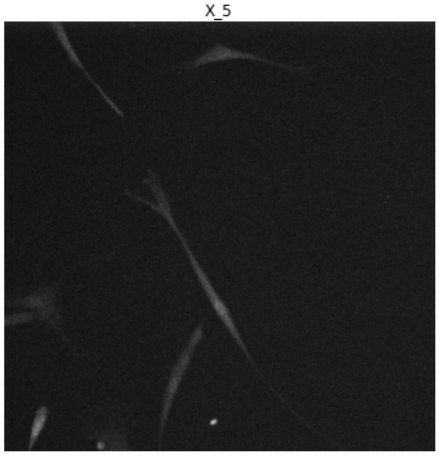
Y_4



Prediction_4



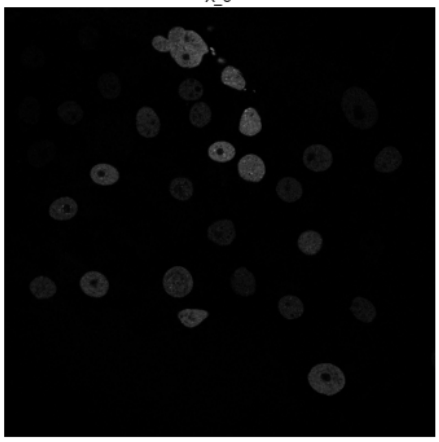
X_5



Y_5

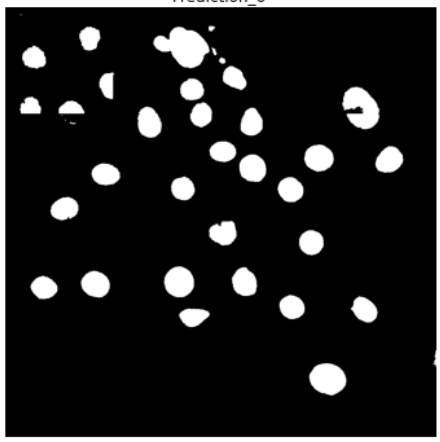
Prediction_5

X_6

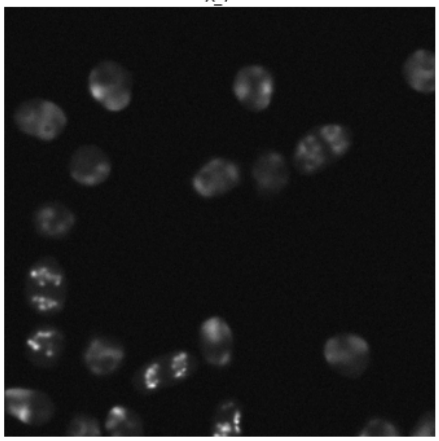


Y_6

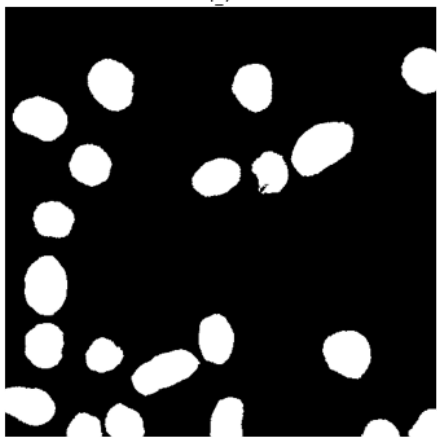
Prediction_6



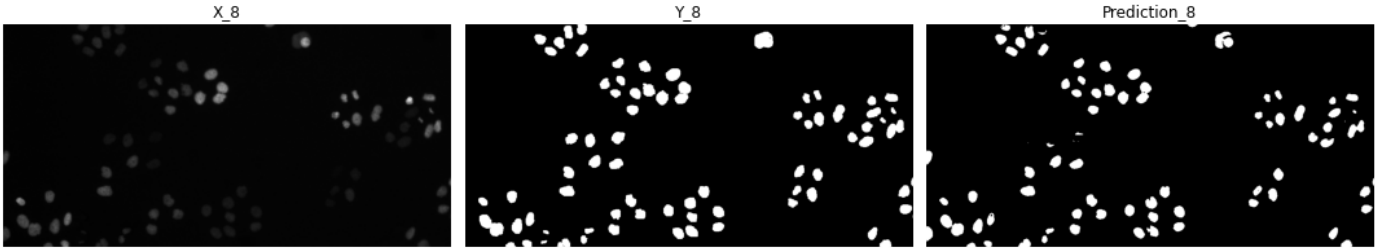
X_7

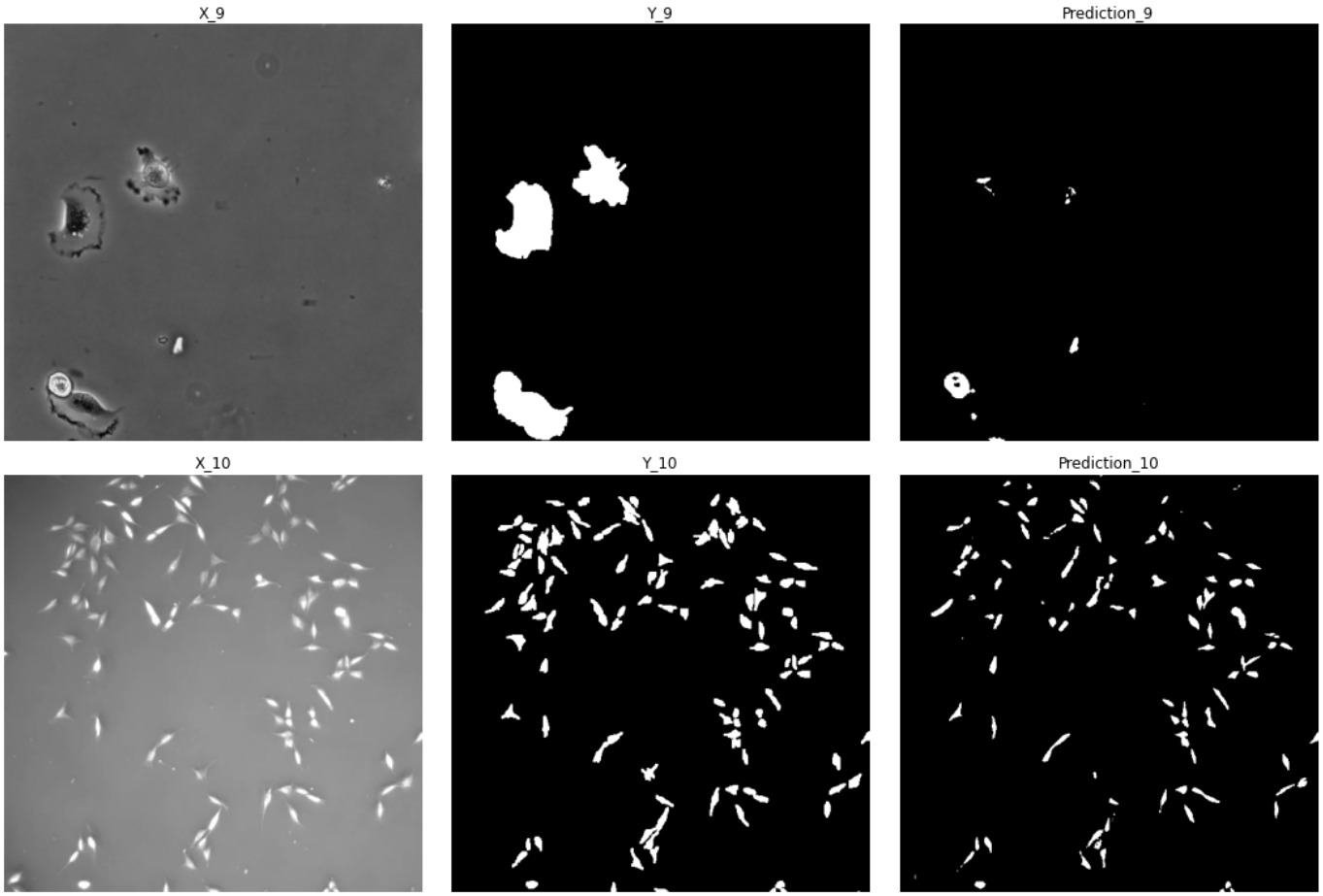


Y_7



Prediction_7





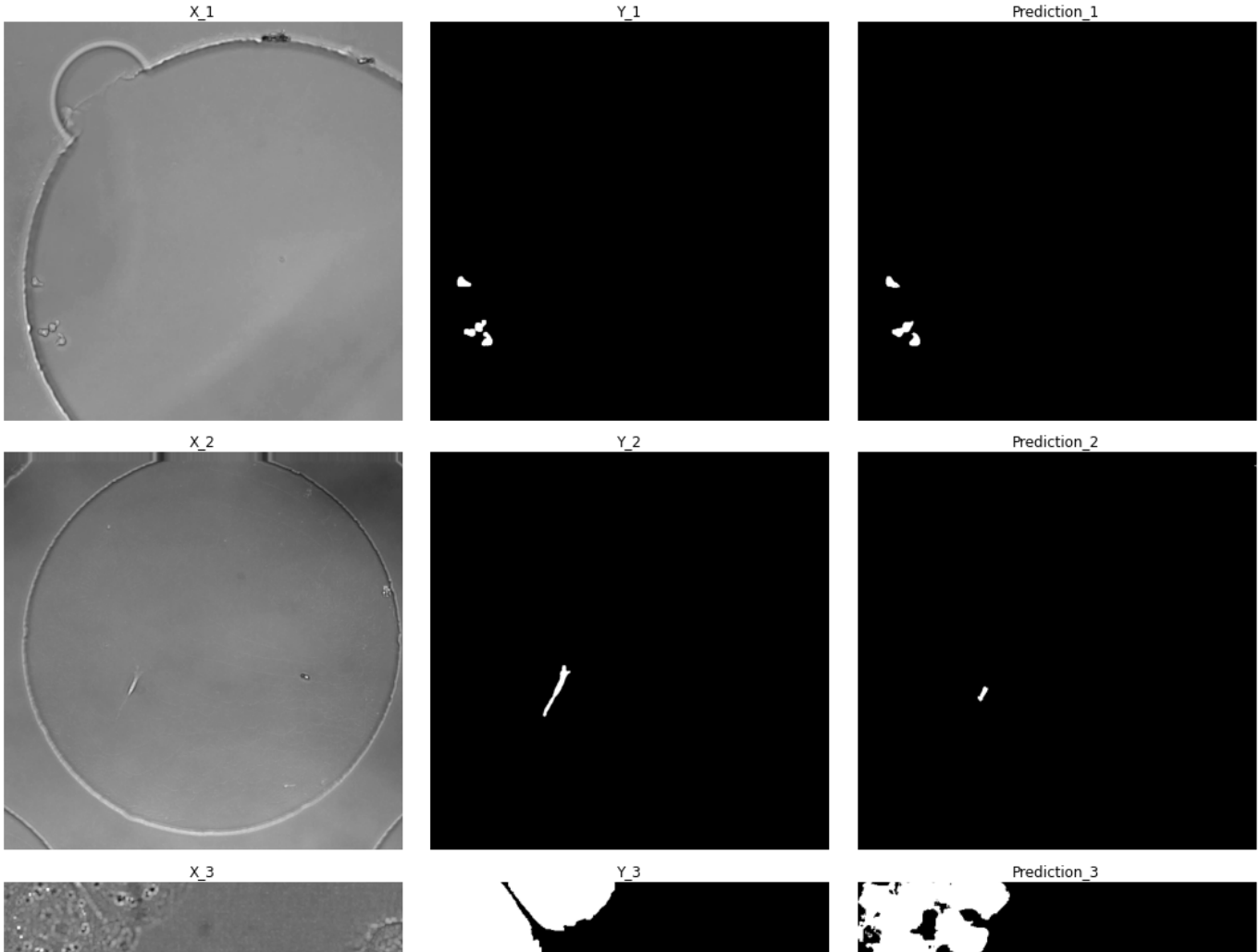
Surprisingly, this is a better prediction that the GT model! Suggesting that the ST model is better at predicting all cells

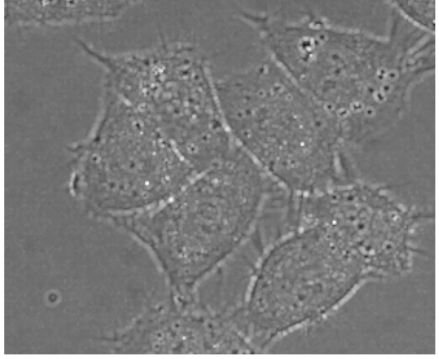
GT Processed Predictions

In [ ]:
```
training_data_code = getcwd() + google_drive_path + "/COMP700_Neural_Network_Code/"
path = training_data_code + "COMP700_UNet_Models/Processed_ST_256/"
model_name = "processed_st_images_256_model_3"
patch_size = 256

gt_processed_array, gt_processed_labels = conductBulkPrediction(random_gt_processed_x_pa
                                                                random_gt_processed_y_pa
                                                                patch_size = patch_size,
                                                                path = path,
                                                                model_name = model_name,
                                                                threshold = 0.5
                                                                )
```

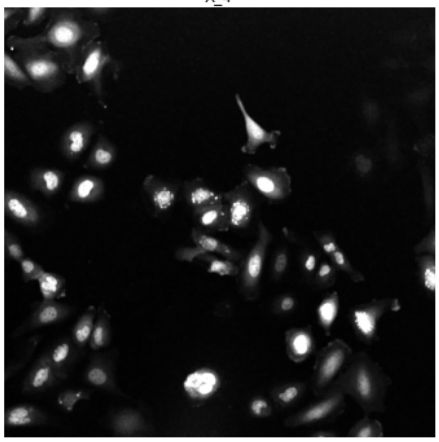
```
# location = image_path + "Matplotlib_Figures/UNET_Model_Predictions_Overall/"
# save2DPlot(array, labels, location, model_name + "_predictions.png", figsize=(15,50))
display2DPlot(gt_processed_array, gt_processed_labels, figsize=(15, 50))
```

```
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 675ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 2s 2s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 668ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 1s/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 696ms/step
Patchify process starting!
Patchify process complete!
1/1 [==============================] - 1s 722ms/step
```

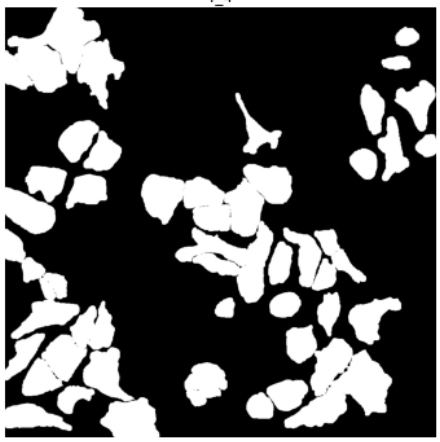




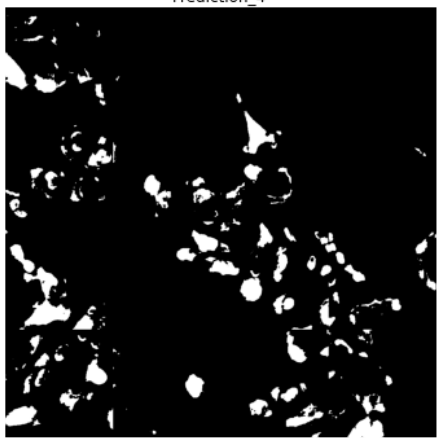
X_4



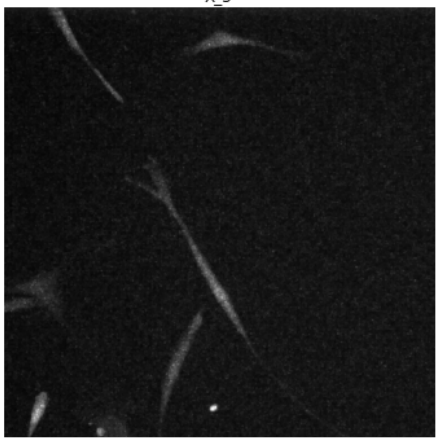
Y_4



Prediction_4



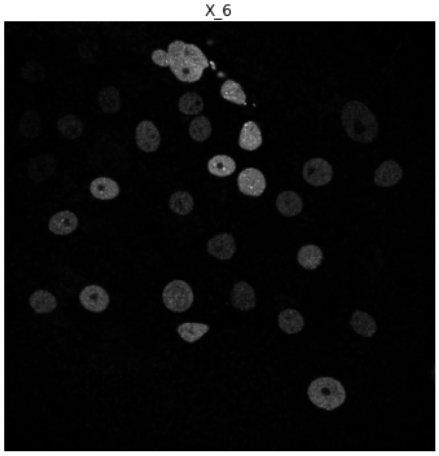
X_5



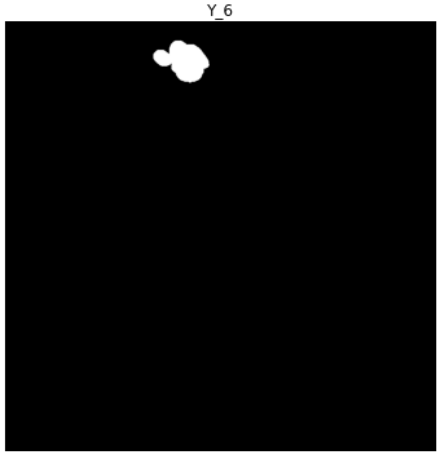
Y_5

Prediction_5

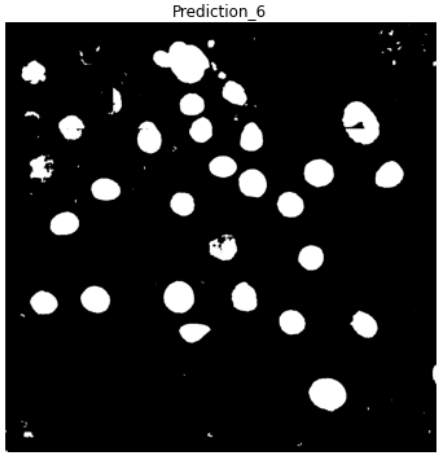
X_6



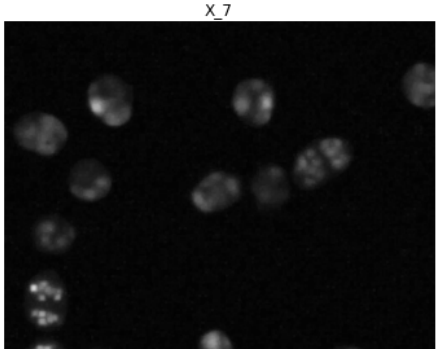
Y_6



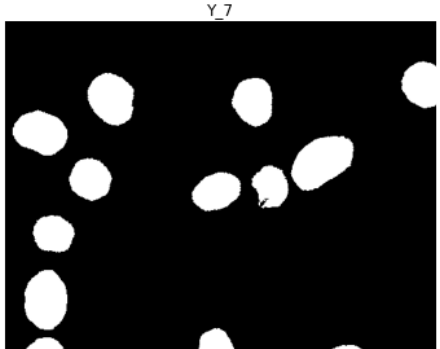
Prediction_6



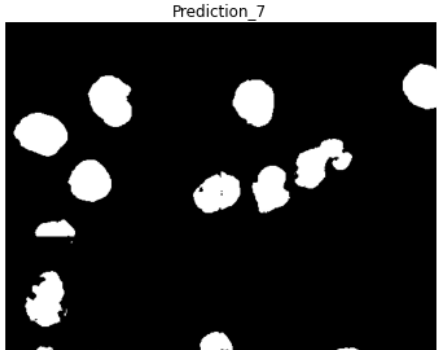
X_7

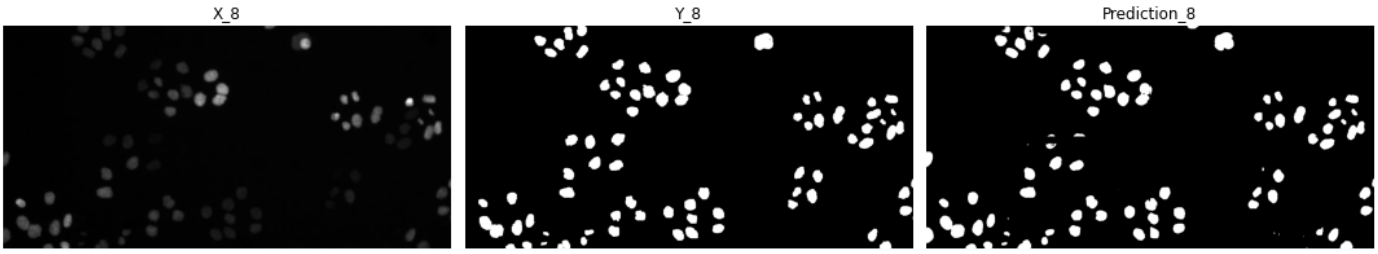


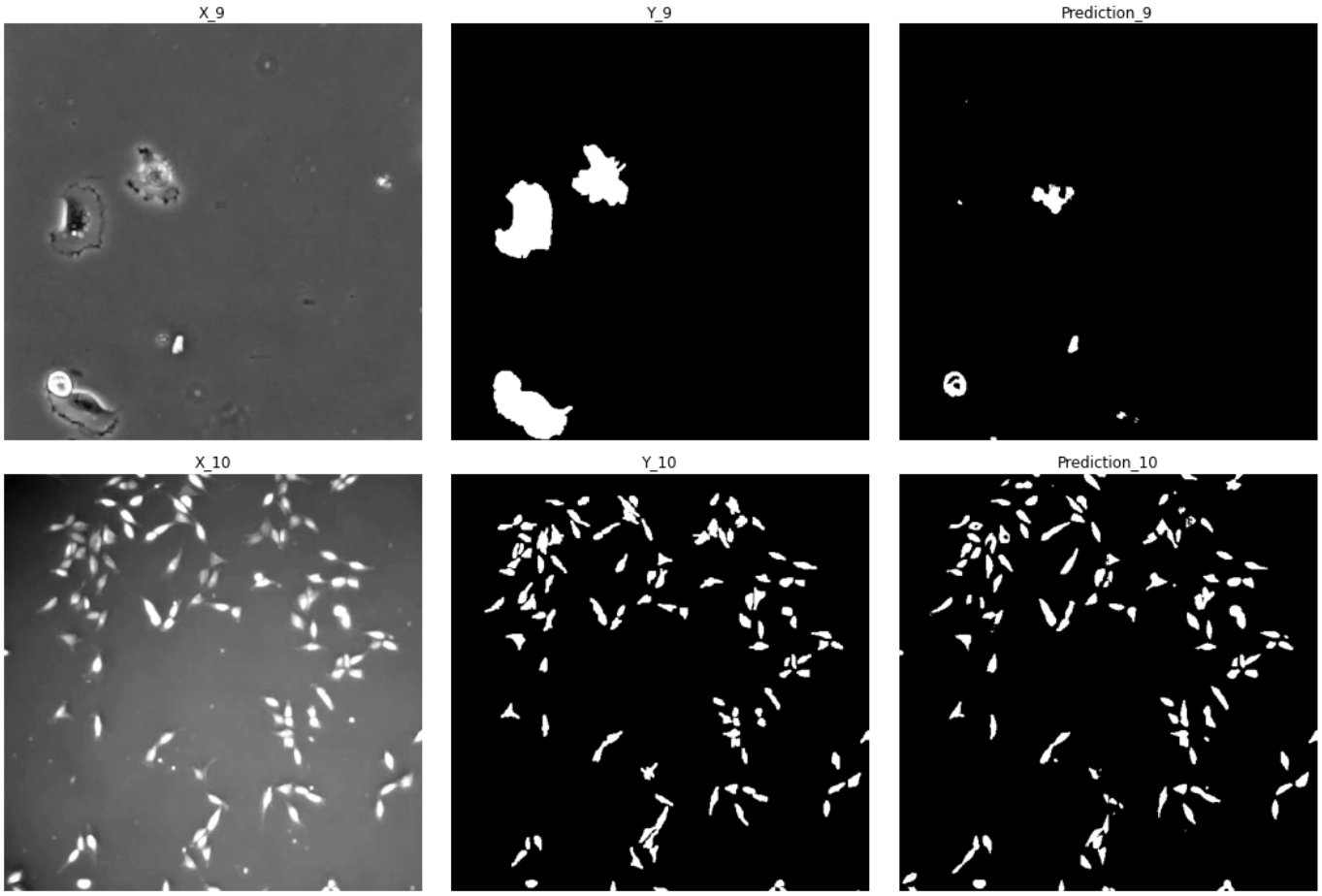
Y_7



Prediction_7







Surprisingly, this is a better prediction that the GT model! Suggesting that the ST model is better at predicting all cells

Conclusion

In the next notebook, videos of the predictions will be generated to encapsulate the work done so far

In [ ]:
```
# import moviepy
```

In [ ]:
```
# def generateVideos(image_array, dimension_array, directories, filename_suffix, use_col
#     i = -1
#     output_video = cv2.VideoWriter()
#     frames_per_second = 2
```

```
#     # Generates Colour Videos
#     for i in range(len(image_array)):
#         for j in range(len(image_array[i])):
#             # initialize
#             if (j == 0):
#                 size = (dimension_array[i][1], dimension_array[i][0] ) # notice order
#                 fileName = directories[i] + filename_suffix + ".mp4"

#                 output_video = cv2.VideoWriter(
#                     fileName,
#                     cv2.VideoWriter_fourcc(*'DIVX'),
#                     frames_per_second,
#                     size,
#                     isColor=use_colour # either True or False
#                 )

#             img = plt.imread( image_array[i][j] )
#             plt.imsave("temp.png", img, cmap='gray')
#             img = cv2.imread( "temp.png", cv2.IMREAD_GRAYSCALE)

#             output_video.write(img)

#         cv2.destroyAllWindows()
#         output_video.release()
#         print("Video finished for ", fileName, sep="")

#     # remove at end
#     if (exists("temp.png")):
#         remove("temp.png")
# ###
```

In [ ]:

```
# generateVideos(gt_raw_x_images,
#                gt_dimensions,
#                gt_directory_list,
#                filename_suffix="_raw_gt",
#                use_colour=False)
```