Part 1

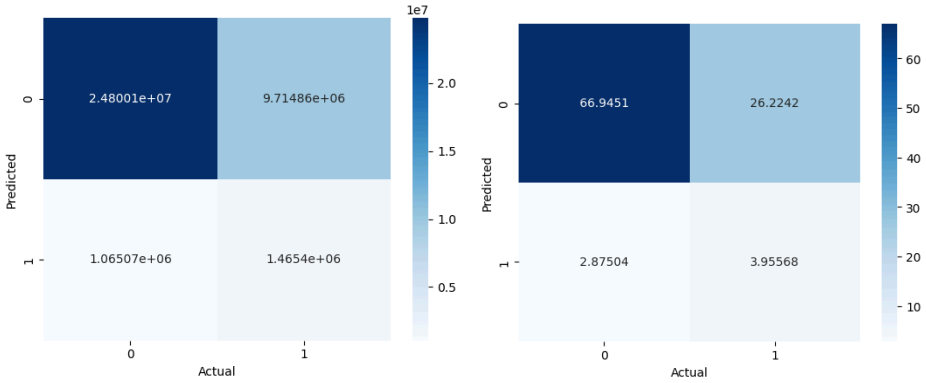
1. To binarize labels, a binarize_labels transform class was created with overloads the __call__ method. Binarize_labels instance is then passed on as a transform function to transform the target into binary labels, 0 for the background and 1 for the remaining labels.

```
class binzarize_label(object):
    def __call__(self, image):
        return torch.where(image > 0, 1, 0)
```

2. To make the image sizes compatible with our 3 layer downsizing by a factor of 2 per layers. We need to have an image which is divisible by 8. Therefore, we take the modulus of the image dimensions with respect to 8. Then we subtract the remainder from the image dimensions to get dimensions which will be compatible with our network architecture.

```
def crop_nearest_2_power(power_2, image):
    H, W = image.shape[-2:]
    new_H = H  - (H % 2 ** power_2)
    new_W = W  - (W % 2 ** power_2)
    return torchvision.transforms.CenterCrop((new_H, new_W))(image)
```

Part 2

1. 

2. The number of background pixels, label 0 is significantly higher than the number of foreground pixels. This makes sense because our images are always going to be surrounded by the background and the number of background pixels in most cases will exceed the number of foreground pixels.
3. I am not satisfied with the network because, while an approximate 70% accuracy for the given network architecture and number of epochs seems alright. Upon further investigation, the precision and recall for the foreground label is pretty poor, sitting at 59% and 13%. Recall is especially poor because for a lot of pixels we are classifying the background to be foreground, this is also justified by the fact that the recall for background class is very high as well, at 95%. Our accuracy is also propped up by the fact that the number of background labels is so high that the model can simply classify all the pixels as negative and the accuracy will still look good.
4. I am not surprised by the fact that our model learned to model the background much better than the foreground simply because there are more pixels for the background. Data augmentation can improve the accuracy of foreground class further. Also we should apply some techniques to penalize the high influence our background class has in the loss function.