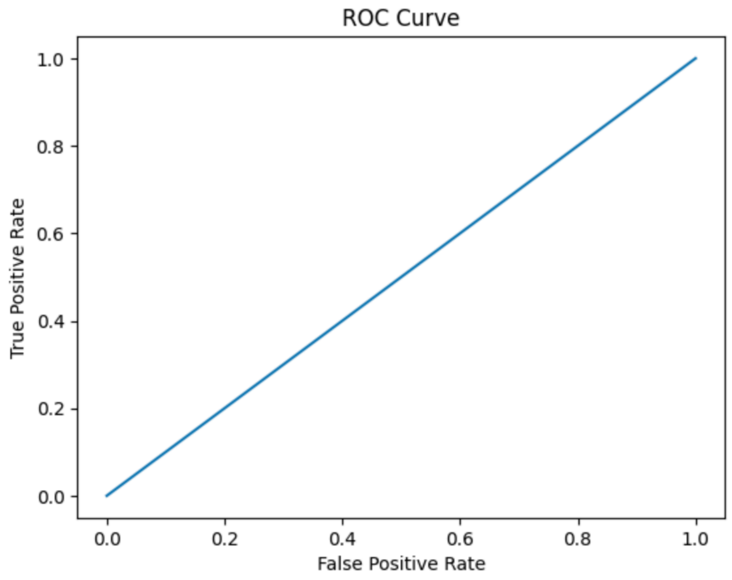
GAN Lab report

1. How does the output look like? Is there any difference with the output of step number 4?

- the output of the first training by the original data:

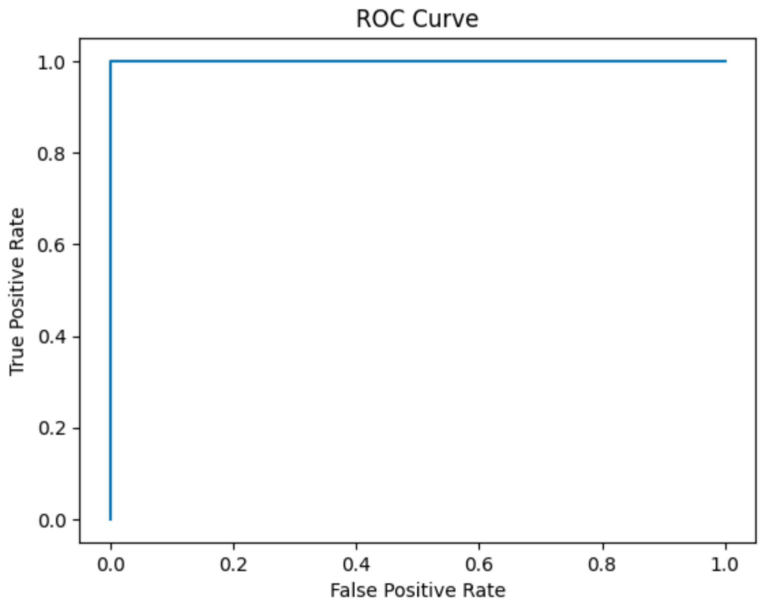
```
confusion matrix:
 [[  0  10]
 [  0 100]]
```



```
Area Under the Curve(AUC) from the imbalanced data: 0.5
```

- the output of the edited data applying GAN

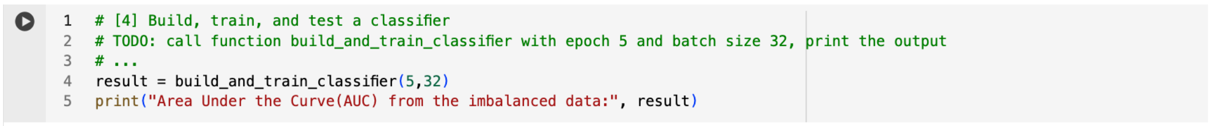
```
confusion matrix:
 [[100   0]
 [  0 100]]
```



```
Area Under the Curve 1.0
```

2. Please create a report to briefly explain the results. Attach the code that you wrote and the screenshots of the outputs.

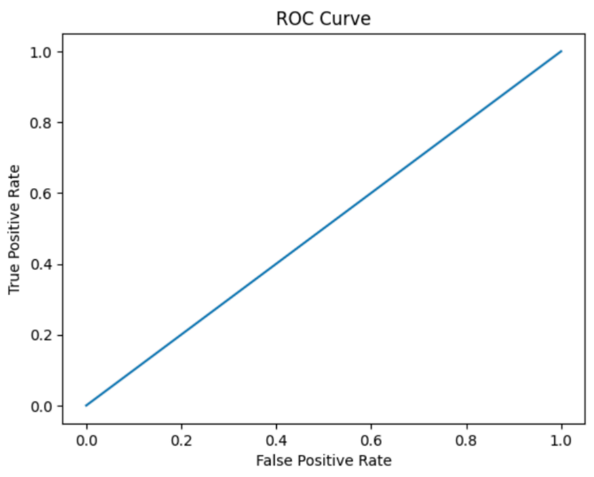
Before proceeding, we set up a pre-defined function and parameters for building the classifier. First, we import the Mnist datasets, comprising 100 samples of '0' and 1000 samples of '1'. Moving forward to step[4], we apply the "build_and_train_classifier" function to construct our classifier, specifying the number of epochs and batch size. Both the training process and testing results will be facilitated by this function as well.



```
# [4] Build, train, and test a classifier
# TODO: call function build_and_train_classifier with epoch 5 and batch size 32, print the output
# ...
result = build_and_train_classifier(5,32)
print("Area Under the Curve(AUC) from the imbalanced data:", result)
```

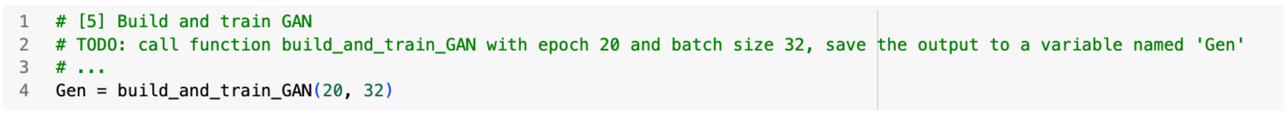
```
Epoch 1/5
28/28 [==============================] - 2s 23ms/step - loss: 0.6864 - accuracy: 0.5499 - val_loss: 0.6450 - val_accuracy: 0.9596
Epoch 2/5
28/28 [==============================] - 0s 9ms/step - loss: 0.6105 - accuracy: 0.8934 - val_loss: 0.5565 - val_accuracy: 1.0000
Epoch 3/5
28/28 [==============================] - 0s 6ms/step - loss: 0.5413 - accuracy: 0.8990 - val_loss: 0.4673 - val_accuracy: 1.0000
Epoch 4/5
28/28 [==============================] - 0s 6ms/step - loss: 0.4713 - accuracy: 0.8990 - val_loss: 0.3740 - val_accuracy: 1.0000
Epoch 5/5
28/28 [==============================] - 0s 6ms/step - loss: 0.3987 - accuracy: 0.8990 - val_loss: 0.2786 - val_accuracy: 1.0000
4/4 [==============================] - 0s 3ms/step

confusion matrix:
 [[  0  10]
 [  0 100]]
```



```
Area Under the Curve(AUC) from the imbalanced data: 0.5
```

Before constructing the balanced datasets for '0' and '1', we need to create a GAN using the predefined function "build_and_train_GAN " which is in step[5].



```
# [5] Build and train GAN
# TODO: call function build_and_train_GAN with epoch 20 and batch size 32, save the output to a variable named 'Gen'
# ...
Gen = build_and_train_GAN(20, 32)
```

After building and training GAN, our next task involves generating fake data for class '0' utilizing a pre-defined function which is in step[6]. We then got 900 more fake pictures, so now we have a total of 1000 pictures for class '0'. We need to rearrange the dataset and check how many pictures we have in total now.

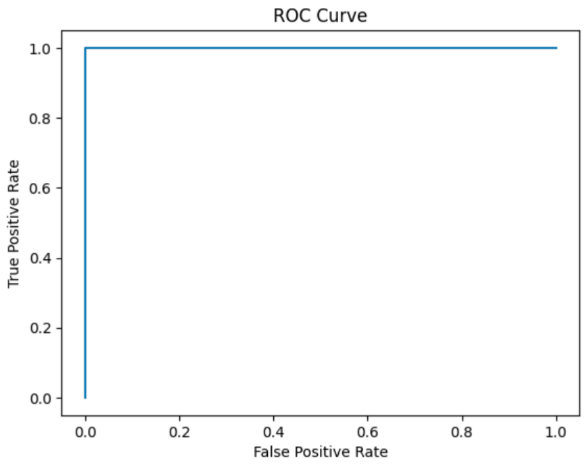
```python
# [6] Create fake data for class '0'
number_of_fake_images = 900
# TODO: call function create_fake_images with given number of fake images, save the output to a variable named 'fake_images'
# ...
fake_images = create_fake_images(number_of_fake_images)
fake_images = fake_images.reshape((-1, image_size, image_size, 1))
print(fake_images.shape)
```

In the final step, we utilize the function "build_and_train_classifier" just like we did for the imbalanced dataset classification.

```python
# [8] Build, train, and test a classifier
# TODO: call function build_and_train_classifier with epoch 5 and batch size 32, print the output
# ...
final_output = build_and_train_classifier(5,32)
print("Area Under the Curve", final_output)
```

```
Epoch 1/5
51/51 [==============================] - 3s 17ms/step - loss: 0.6310 - accuracy: 0.5556 - val_loss: 0.7213 - val_accuracy: 0.0000e+00
Epoch 2/5
51/51 [==============================] - 1s 10ms/step - loss: 0.5689 - accuracy: 0.6488 - val_loss: 0.6713 - val_accuracy: 1.0000
Epoch 3/5
51/51 [==============================] - 0s 10ms/step - loss: 0.4878 - accuracy: 0.9444 - val_loss: 0.5746 - val_accuracy: 1.0000
Epoch 4/5
51/51 [==============================] - 0s 10ms/step - loss: 0.3889 - accuracy: 0.9451 - val_loss: 0.4372 - val_accuracy: 1.0000
Epoch 5/5
51/51 [==============================] - 0s 9ms/step - loss: 0.2784 - accuracy: 0.9654 - val_loss: 0.2663 - val_accuracy: 1.0000
7/7 [==============================] - 0s 6ms/step

confusion matrix:
 [[100   0]
 [  0 100]]
```



```
Area Under the Curve 1.0
```

Conclusion:
According to both output of the line graphs, it's obvious that the output from the balanced dataset (generated using GAN) confuses the computer more effectively. This confirms that GAN and fake data perform as expected, enhancing the model's capability.