**Name: Marzuk Rashid, Snehil Chopra**
**NetID: marzukr2, snehilc2**
**Team name on Kaggle leaderboard: synchro**

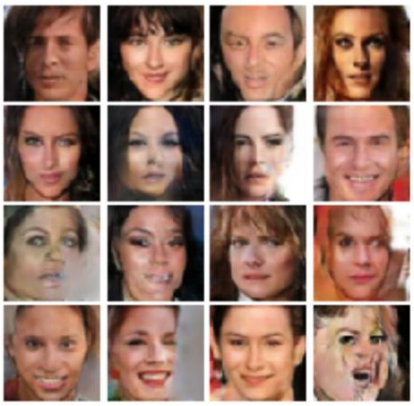
## Part 1:

*Answer the following questions briefly (no more than a few sentences), and provide output images where requested.*

Show final results from training both your GAN and LSGAN (give the final 4x4 grid of images for both):

`Iter: 19650, D: 1.269, G:1.731`

`Iter: 19600, D: 0.2438, G:0.2014`



Left: GAN, Right: LSGAN

Discuss any differences you observed in quality of output or behavior during training of the two GAN models.

I noticed that at the beginning of training, GAN appears to converge to images that look like faces in around 500-600 iterations, whereas LSGAN takes around 1000-1200. I also noticed that after fully training, LSGAN appears to have more background artifacts like the gray patch in this image:

Do you notice any instances of mode collapse in your GAN training (especially early in training)? Show some instances of mode collapse (if any) from your training output.

Iter: 2400, D: 1.29, G:5.246

After 2400 iterations (the middle of the 3rd epoch), some evidence of mode collapse emerges. In particular, these images look very similar:

Discuss briefly how/whether spectral normalization helps generate higher quality images in your implementation. Ideally, you should show samples from models with and without normalization.

Iter: 5850, D: 1.104, G:0.1802 Iter: 5850, D: 0.7667, G:3.06

Left: GAN without spectral normalization, Right: GAN with spectral normalization

As shown in these examples, spectral normalization helps the discriminator perform much better with a loss of 0.77 instead of 1.10 after 5850 iterations. While the images with/without spectral normalization after 5850 iterations don't appear to be significantly different in quality, a better performing discriminator will also allow the generator to perform better over more iterations.

## Part 2 (generation):

Give the hyperparameters for your best network on classification task below. Note any other changes you made to the base network in addition to the hyperparameters listed in the table below.

| Hyperparameter | Value |
|---|---|
| RNN type: | GRU |
| Number of layers: | 4 |
| Hidden layer size: | 150 |
| Learning rate: | 0.001 |

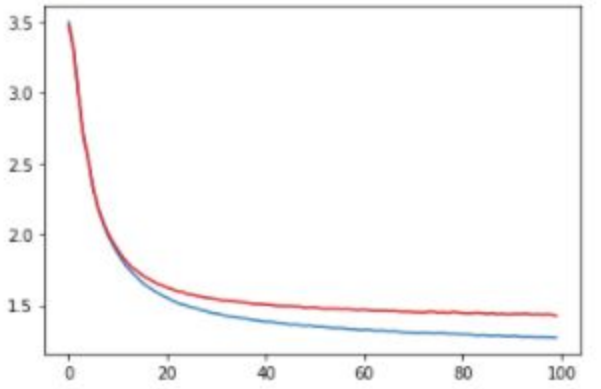
Give an example 1000 character output from your network: (NEXT PAGE)

```
[ ]  1 print(evaluate(rnn, prime_str='Th', predict_len=1000))

Then call his party.
Oft any man I am fairest for arpent, and bear
Are pass. Now, call me to the detenty.

AEGEON:
I will not put on.

LEONATO:
Madam, combat, yet he had never can so strike him
As cheekl to a face to be forth two want.

DUKE VINCENTIO:
What wrong soon,
From the sight, that you should fly to Ford under a
You choose point, and in the sea to liminius,
And so he is none and his babe Cains:
Of heavens and made back and heart to fear
The heart of what he has spleave here as it
Which oft she presently to thine eyes.

LPAVIA:
I'll thy error to early eye on her father.

BENVOLIO:
No, the wrongs of grace that we perchance with a
she is. What says Looked poyer, I will be so, that were
the self, and Timon! may I find up your grace?

BAPTISTA:
Ay, therein by safety, and let them come
Or holy death wide-royal looks and Alic,
Farewell to the Thracious untimely virtue,
What we have, and go to the pound for with his grave
And Naence men well, and I am in Rome.

KING HENRY VIII:
Then he w
```

Insert the training & test loss plot from your RNN generation notebook below:

```
[ ]  1 import matplotlib.pyplot as plt
     2 import matplotlib.ticker as ticker
     3
     4 plt.figure()
     5 plt.plot(all_losses)
     6 plt.plot(test_losses, color='r')
```

```
[<matplotlib.lines.Line2D at 0x7fd543421630>]
```



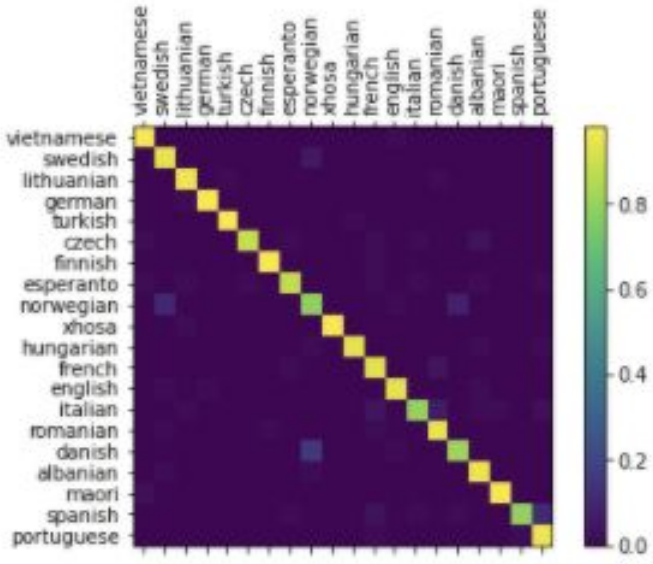

I utilized the GRU model in the first attempt and hit a reasonable test loss, around 1.6, on the first try. I then tuned the parameters which further decreased the test loss to around 1.43. Specifically, I changed the number of layers from 1 to 4, hidden layer size from 100 to 150, the learning rate from 0.01 to 0.001.

## Part 2 (classification):

Give the hyperparameters for your best network on classification task below. Note any other changes you made to the base network in addition to the hyperparameters listed in the table below.

| Hyperparameter | Value |
|---|---|
| RNN type: | GRU |
| Number of layers: | 4 |
| Hidden layer size: | 150 |
| Learning rate: | 0.001 |

You should reach the Kaggle accuracy benchmark with your Kaggle submission. Your notebook evaluation results should be similar to your performance on Kaggle. Insert the confusion matrix image outputted from your best model, and report the corresponding accuracy:




After trying out the original parameters for training, I hit a mere 78% test accuracy. I then increased the n_iters to 2000, which brought me to around 83% test accuracy. Finally, I decreased the chunk_len from 50 to 20, learning rate from 0.01 to 0.001, hidden layer size from 100 to 150, number of layers from 1 to 4, and batch size from 100 to 200. These changes yielded a 91.9% accuracy, which is more than the Kaggle baseline.