Lab Assignment 4: Generative Adversarial Network

Due Date: Sunday November 17, at 11:59 PM

Purpose:

The purpose of this Lab assignment is:

1. To get hands-on experience of applying Deep Neural Networks, specifically deep convolutional generative adversarial network

General Instructions:

Be sure to read the following general instructions carefully:

1. This assignment must be completed individually by all students.
2. Only provide the requested screenshots and make sure to have a complete screenshot, partial screenshots will not earn any marks.
3. You will have to provide a **demonstration video for your solution** and upload the video together with the solution on **eCenntenial** through the assignment link. See the **video recording instructions** at the end of this document.
4. In your 5-minute demonstration video you should explain your solution clearly, going over the main code blocks and the purpose of each module/class/method also demoing the execution of exercises #1. YouTube links and links to google drive or any other media are not acceptable, the actual recording must be submitted.
5. Any submission without an accompanying video will lose 70% of the grade.
6. In your analysis report make sure you provide an introduction and clearly state the facts and findings. Any submission missing Analysis report will lose lost 70%.

Submission:

There are three elements to be submitted for this assignment in one zipped folder (All subject to grading as per rubric for this assignment):

1. For each exercise that require code, please create a project folder and include all project python scripts/modules and screenshot of output, as needed. Name all python scripts your firstname_lab4.py. Name the folder "Exercise#X_firstname", where X is the exercise number and firstname is your first name. (In total 1 folders for this assignment).
2. For all questions that require written or graphic response create one "Word document" and indicate the exercise number and then state your response. Name the document

"Written_response_firstname", where firstname is your firstname. (In total one word or pdf document).

3. All submissions need to be accompanied with a recorded demonstration video not to exceed 5 minutes in length, focus on showing the key code functionalities and run the code.

Create one zipped folder containing all of the above, name it lab4assignment_firstname where firstname is your firstname.

Assignment – exercises:

1. **Exercise #1:** GAN (100 marks)

Requirements:

a. Get the data:
    1. Import and load the 'fashion_mnist' dataset from TensorFlow. Using 2 dictionaries store the fashion_mnist datasets into *ds1_firstname* and *ds2_firstname*, where firstname is your firstname. The first 60,000 data samples will be stored in *ds1_firstname* directory with keys '*images*' and '*labels*', which will contain the images and labels of the dataset. The next 10,000 data samples will be stored in *ds2_firstname* directory with keys '*images*' and '*labels*', which will contain the images and labels of the dataset
    For more info checkout:
    https://keras.io/api/datasets/fashion_mnist/#load_data-function

b. Dataset Pre-preprocessing
    1. Normalize the pixal values in the dataset to a range between -1 to 1. Store result back into *ds1_firstname['images']* and *ds2_firstname['images']*
    2. Display (print) the shape of the *ds1_firstname['images'], ds2_firstname['images'].*
    3. Using np.concatenate, create a new dataset named *dataset_firstname.* The dataset will contain pants images (class label 1) from *ds1_firstname* and *ds2_firstname.* For more info checkout https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html)
    4. Display (print) the shape of the *dataset_firstname.* Note: The dataset should have a total of 7000 images.

5. Display (plot) the first 12 images from the dataset using matplotlip. Remove xticks and yticks when plotting the image. Plot the images using a figure size of 8x8 and a subplot dimension of 4x3
6. Using Tensorflow's Dataset from_tensor_slices(), shuffle(), and batch create training dataset called *train_dataset_firstname* from the *dataset_firstname.* The training dataset will shuffle all 7000 images and have a batch size of 256.

c. Build the Generator Model of the GAN
   1. Use TensorFlow's Sequential() to build a CNN mode (name the model generator_model_firstname) with the following architecture:
      i. Input = Vector with dimension size 100
      ii. 1st Layer = Fully connected Layer with 7*7*256 neurons and no bias term
      iii. 2nd Layer = Batch Normalization
      iv. 3rd Layer = Leaky ReLU activation
      v. 4th Layer = Transposed Convolution Layer with 128 kernels with window size 5x5, no bias, 'same' padding, stride of 1x1. Note: Input to the Transposed Layer should first be reshaped to (7,7,256). For more info, reference: https://keras.io/api/layers/convolution_layers/convolution2d_transpose/
      vi. 5th Layer = Batch Normalization
      vii. 6th Layer = Leaky ReLU
      viii. 7th Layer = Transposed Convolution Layer with 64 kernels with window size 5x5, no bias, 'same' padding, stride of 2x2.
      ix. 8th Layer = Batch Normalization
      x. 9th Layer = Leaky ReLU
      xi. 7th Layer = Transposed Convolution Layer with 1 kernels with window size 5x5, no bias, 'same' padding, stride of 2x2, and tanh activation
   2. Display (print) a summary of the model using summary(). Draw a diagram illustrating the structure of the neural network model, making note of the size of each layer (# of neurons) and number of weights in each layer. Note: The generator model should output an image the same dimension as the dataset

d. Sample untrained generator
   1. Using Tensorflow's random.normal(), create a sample vector with dimension size 100.
   2. Generate an image from generator_model_firstname. Ensure training is disabled.
   3. Display (plot) the generated image using matplot lib.

e. Build the Generator Model of the GAN
    1. Use TensorFlow's Sequential() to build a CNN mode (name the model generator_model_firstname) with the following architecture:
        i. Input = Image
        ii. 1st Layer = Convolution with 64 filter kernels with window size 5x5, stride of 2x2, and 'same' padding
        iii. 2nd Layer = Leaky ReLU activation
        iv. 3rd Layer = Dropout with rate of 0.3
        v. 4th Layer = Convolution with 128 filter kernels with window size 5x5, stride of 2x2, and 'same' padding
        vi. 5th Layer = Leaky ReLU activation
        vii. 6th Layer = Dropout with rate of 0.3
        viii. 7th Layer = Transposed Convolution Layer with 64 kernels with window size 5x5, no bias, 'same' padding, stride of 2x2.
        ix. 8th Layer = Batch Normalization
        x. 9th Layer = Leaky ReLU
        xi. Output = 1 (Note: Input to the output should be flatten first)
    2. Display (print) a summary of the model using summary(). Draw a diagram illustrating the structure of the neural network model, making note of the size of each layer (# of neurons) and number of weights in each layer.

f. Implement Training
    1. Create a loss function using Tensorflow's BinaryCrossentropy() and call it *cross_entropy_firstname.* Make sure to set from_logits=True. This loss function will be used to calculate the loss for the generator and discriminator. For more info checkout: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy
    2. Using Tensorflow's optimizers, create a generator and discriminator optimizer. Both optimizers will use Adam optimizers and should have the name *generator_optimizer_firstname* and *discriminator_optimizer_firstname* respectively.
    3. Create a tensorflow function using tf.function and call it training_step. The function takes a batch of images as input and updates the discriminator and generator using the optimizer and calculating the gradients from the calculated the losses. For more info checkout: https://www.tensorflow.org/api_docs/python/tf/function. The function should be similar to the following code snippet below (Examine the code and make the necessary adjustment):

```
def train_step(images):
    noise = tf.random.normal([256, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator_model_firstname(noise, training=True)

      real_output = discriminator_model_firstname(images, training=True)
      fake_output = discriminator_model_firstname(generated_images, training=True)

      gen_loss = cross_entropy_firstname(tf.ones_like(fake_output), fake_output)
      real_loss = cross_entropy_firstname(tf.ones_like(real_output), real_output)
      fake_loss = cross_entropy_firstname(tf.zeros_like(fake_output), fake_output)
      disc_loss = real_loss + fake_loss

    gradients_of_generator = gen_tape.gradient(gen_loss,
generator_model_firstname.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator_model_firstname.trainable_variables)

    generator_optimizer_firstname.apply_gradients(zip(gradients_of_generator,
generator_model_name.trainable_variables))
    discriminator_optimizer_firstname.apply_gradients(zip(gradients_of_discriminator,
discriminator_model_name.trainable_variables))
```

g. Using the *train_dataset_firstname* from Step b.6 and the training function defined in Step f.3, train the models in batches with 10 epochs. Use Python's time module to calculate and display (print) how long each epoch takes. Note: GAN's are trained typically on tens of thousands to hundreds of thousands samples with large number of epochs. In your report, calculate and explain how long it would take to train the same model using 70,000 training samples on 100 epochs using your current hardware.

h. Visualized Trained Generator
    1. Using Tensorflow's random.normal(), create 16 sample vectors, each with the dimension size of 100.
    2. Generate an image from generator_model_firstname. Ensure training is disabled.
    3. Normalize the pixels in the generated images by multiplying each pixel by 127.5 and adding 127.5 to each pixel.

4. Display (plot) the generated image using matplot lib. Plot the images using a figure size of 8x8 and a subplot dimension of 4x4. Compare and discuss the generated images after training and the initial sample prior to training.

------ End of Exercises ------

Rubric

Evaluation criteria	**Not acceptable**	**Below Average**	**Average**	**Competent**	**Excellent**
	0% - 24%	**25%-49%**	**50-69%**	**70%-83%**	**84%-100%**
Functionality 30%	Missing all functionalities required	Some requirements are implemented.	Majority of requirements are implemented but some are malfunctioning.	Majority of requirements implemented.	All requirements are implemented Correctly.
Classes 30%	Classes have been created incorrectly or completely missing.	Classes have been defined but have errors. Instances are incorrectly used.	Classes have been defined correctly but instances are used incorrectly or not created at all.	Classes have been defined correctly but some instances are used incorrectly.	Classes are correctly defined and makes use of its own functions which are called somewhere else in the code. Instances have been created and used correctly.
Documentation 5%	No comments explaining code changes.	Minor comments are implemented.	Some code changes are correctly commented.	Majority of code changes are correctly commented.	All code changes are correctly commented.
Design 15%	No adherence to object design principles.	Minor adherence to object design principles.	Some object oriented and modulus design principles are adhered to.	Majority of Object oriented and modulus design principles are adhered to.	Object oriented and modulus design principles are adhered to.
Testing & Evaluation 10%	No evidence of testing and evaluation of the requirements.	Minor evaluation and testing efforts.	Some of the requirements have been tested & evaluated.	Majority of requirements are tested & evaluated.	Realistic evaluation and testing, comparing the solution to the requirements.

Demonstration Video 10%	Very weak no mention of the code changes. Execution of code not demonstrated.	Some parts of the code changes presented. Execution of code partially demonstrated.	All code changes presented but without explanation why. Code demonstrated.	All code changes presented with explanation, exceeding time limit. Code demonstrated.	A comprehensive view of all code changes presented with explanation, within time limit. Code demonstrated.

Demonstration Video Recording

Please record a short video (max 4-5 minutes) to explain/demonstrate your assignment solution. You may use the Windows 10 Game bar to do the recording:

1. Press the Windows key + G at the same time to open the Game Bar dialog.

2. Check the "Yes, this is a game" checkbox to load the Game Bar.

3. Click on the Start Recording button (or Win + Alt + R) to begin capturing the video.

4. Stop the recording by clicking on the red recording bar that will be on the top right of the program window.

(If it disappears on you, press Win + G again to bring the Game Bar back.)

You'll find your recorded video (MP4 file), under the Videos folder in a subfolder called Captures.

Submit the video together with your solution and written response.