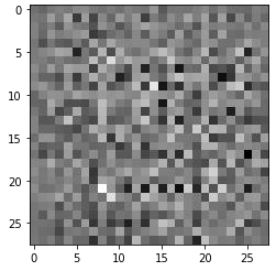
Assignment 6- GAN model

Part 1- Number generator

For the first part of this lab, we want to use GAN to create handwritten numbers from 0 to 9. For this purpose, we will use TensorFlow to create the model from scratch. Nowadays, there are many already developed models which can create these digits, but our goal is learning. So, for this section, we will create one ourselves.

a) From Keras, import the mnist dataset. We need to reshape the images and normalize them.

b) We can now create our generator model. We use Conv2DTranspose (upsampling) layers to produce the image from seed (Generator part). Call this generator model without training just to see what happens when we import a noise to a model.



The result does not look nice, but it is fine. The generator is not trained yet.

c) Now, create a discriminator model using Keras. Like what we did above, apply this discriminative model on top of the above, generate an image, and see the result. Maybe looking at the output does not reveal enough information. Instead, we can apply a cross entropy function to compare the output with the real one.

d) We need to create two loss functions.
- Discriminator loss: This method quantifies how well the discriminator can distinguish real images from fakes. It compares the discriminator's predictions on real images to an array of 1s, and the discriminator's predictions on fake (generated) images to an array of 0s.
- Generator loss: The generator's loss quantifies how well it was able to trick the discriminator. Intuitively. If the generator is performing well, the discriminator will classify the fake images as real (or 1). Here, we will compare the discriminator's decisions on the generated images to an array of 1s.

e) Now, it is time for training the model. It is recommended to take a checkpoint on the model. The training loop begins with the generator receiving a random seed as input. That seed is used to produce an image. The discriminator is then used to classify real images (drawn from the training set) and fake images (produced by the generator). The loss is calculated for each of these models, and the gradients are used to update the generator and discriminator. There is a function provided for you to train the model and save the image.

You can call the train function with two elements: train_dataset and number of epochs. The model takes a while to train but saving checkpoints allows us to get back to model whenever it is needed. While the model is loading, pay attention to the changes in epochs and results. After training, take a checkpoint and save the model. I recommend downloading the model on your Google drive or a local machine. This helps not to re-train the model next time that you need to retrieve the model.

I also provided a code to save the steps of changes as a gif file.

Part 2- GAN and Pokémon!

The next example is inspired from: Intro to Generative Adversarial Networks (GANs) | Towards Data Science and d2l.ai. The goal of this model is to create the characters of Pokémon using a GAN model. The dataset that we use is from d2l package (d2l · PyPI).

a) Read the Pokémon dataset. It contains 40597 images from 71 classes. Write a function to normalize the images by dividing them to 255 and use gaussian normalization with mean of 0.5 and standard deviation of 0.5.

b) As we discussed earlier, it is always good to visualize our data to gain better insights. Run the provided code to see the images.

c) We start with a generator model in which the noise is converted to an RGB image. This is a special case that uses a convolutional layer to create this model. You can read more about this type of model here. Import a signal in the form of zeros into the model. This input has a stride of 2 in a 16x16 image and is for three channels which means `x = tf.zeros((2, 16, 16, 3))`

Now, call the G_block package and use 20 as the number of layers (input of the class). To see the dimension of the output, you can use:

```
g_blk(x).shape
```

d) Now is the time to create a discriminator. The block is already provided for you. Import a signal in the form of zeros into the model. This input has stride of 2 in a 16x16 image and is for three channels which means `x = tf.zeros((2, 16, 16, 3))`

Now, call the D_block package and use 20 as the number of layers (input of the class). To see the dimension of the output, you can use:

```
d_blk(x).shape
```

e) The training function is already provided. Explain your understanding about the training block. Call the train function with appropriate values. What is your understanding from the graph?