Tips

1 APIs

1.1 Google colab:

When working with a rather large dataset, my recomendation is to upload it first to google drive and then mount the drive in your workspace.

To mount your drive into the workspace, just type the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

An authentication code will be asked, just follow the link provided and authenticate yourself with your google drive account.
Now, imagine I had my dataset on my drive in the directory "/My Drive/Trains Dataset", i can simply copy the content of that directory to the local workspace (eg. "Trains Dataset"). This example can be done with a simple unix command as shown bellow.

```
!cp -r "/content/drive/My Drive/Trains Dataset" "Trains_Dataset"
```

1.2 Kaggle:

Follow this easy steps to use a kaggle dataset:

```
!pip install kaggle
!mkdir -p ~/.kaggle
```

If you don't have a kaggle account, create one on their site: `https://www.kaggle.com` Go to 'My account' page, then scroll down a bit and you will see information about Kaggle api. There will be a button named 'Create New API Token' and if you click it a kaggle.json file will be downloaded. Now you upload it to Google Colab and follow the following steps.

```
!cp kaggle.json ~/.kaggle/
!chmod 600  ~/.kaggle/kaggle.json
```

Now to dowload a dataset, you just have choose one by going to their website and providing the ¡dataset user¿/¡dataset name¿. For example, in the following example I chose to download a dataset from the url — `https://www.kaggle.com/jhoward/lsun_bedroom`. When using the api like bellow you, however, just have to provide the user — jhoward and dataset name — lsun_bedroom as I said previously.

```
!kaggle datasets download -d jhoward/lsun_bedroom
!unzip lsun_bedroom.zip
```

Once it downloads, you can unzip the file and use it.

2 General tips

2.1 BatchNorm & Dropout

Never forget to let the model know whether it is training or not (Training vs Inference). It is reasonable to assume that there is no point in continuing learning weights, once you are quite happy with the model you trained. Not only it this time consuming and computationally exhausting, but also not intended. Apart from that, certain layers, such as Batch Normalisation and Dropout, function differently when in training mode and inference mode.

2.2 Sub-classing - @tf.function

Every method associated with the process of learning in your model should be accompanied with the @tf.function decorator. The @tf.function decorator converts your block of code into a graph so your code runs faster, however you should not expect your code to run in a pythonic way. Meaning, python built in features and methods may not behave as expected (eg. Use tf.print() instead of print()). Python default print will only run once, when the graph is built, whereas if you use tf.print(), whatever you put there will be printed the times you would be expecting.

Also, it is worth mentioning that within @tf.function block of code, one can only use tensors, therefore the attempt to convert a tensor to a numpy array will raise an error.

Link for further reading: `https://www.tensorflow.org/api_docs/python/tf/function`

2.3 Sorting directories

When loading files from a folder to code data structures make sure you sort, in some way, the content of your folder so you always get the same results. For example, use alphabetically order. Once, I was loading images from a certain directory and labels from another. Visually in the folders you could see that the first image corresponded to the first label on the other directory, but because I was not sorting them, I was getting weird results in my model. It took me a while to spot it and its not a great experience because you start doubting the architecture you have.

2.4 Dataset size - building model

When working with images in generative models, there is no point in loading loads and loads of images when you are just figuring out if your model is com-

piling or just working decently. It can be a waste of time and resources. Start with something like 5k images or less.

2.5 Pre-process data

Make sure you feed normalised [0,1] or standardised [-1,1] data/images to your models, because they we will be expecting an input in that order of magnitude. The same idea goes backwards if you want to store images. Depending on the activation functions you use you will have to map the values to image values. When using a library like matplotlib the images are store in the interval [0,1] so if you use a sigmoid function as an activation function, there is no need for any further operation. Tanh activation function outputs values in the range of [-1,1], so be careful with that.