**This document contains all the general FAQ. The Siamese experiment using IMFDB database is a bit complicated owing to the dataloader/network logic; hence a lot of these points deals with that. However, there are VERY CRITICAL POINTS ( and frequently encountered errors) related to other aspects (such as saving/loading data; dumping the model to CPU etc) as well here.**

**NOTE1:** “source venv/bin/activate” is extremely critical when working on the server!!

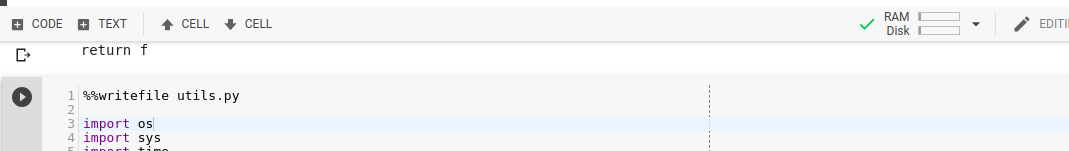
**NOTE2:** All the datasets you want to use for the hacakthon are provided in the starter code of Hackathon. However pasting the wget commands below:

wget <https://cdn.talentsprint.com/aiml/FaceRecogHackathon/Datasets/IMFDB_face_recog/IMFDB_final-20190607T024441Z-001.zip>

wget <https://cdn.talentsprint.com/aiml/Experiment_related_data/Expression_data.zip>

wget <https://cdn.talentsprint.com/aiml/FaceRecogHackathon/Datasets/ATandT/data-20190607T005435Z-001.zip>

1. Editing files directly on colab: (Note you must be aware of other methods to manually upload/download files).
   1. Any file on colab can be edited by first seeing the contents of the file by using **%cat <fileName>** in a new cell. Filename with the right path of course.
   2. Then paste the output content in a ‘new cell’. Then on top use the command **%%writefile <fileName>** to see the file saved.Filename with the right path of course.See screenshot below:

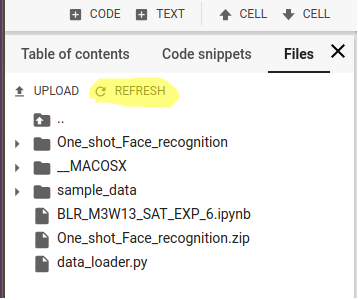


* 1. Do remember to run the file to ensure that the changes that you made reflects in your colab’s ipython environment by using %run magic function as follows **%run <yourFileName>** and the**n ‘Restart runtime’ (NOT ‘reset all runtimes’)**. .After this you will have to re-run the cells (use colab ‘Run above’ and ‘run below’ to your advantage on this) In absence of this, though you make changes to the file it will not reflect on your colab environment. (i.e. say you added a new function, you will not be able to call it within colab).

2. Ensure you’re in the right folder structure when running your Colab cells. Several parts of the code sometimes depend on the right base directory.

3. Add **%%capture** as a top line on the cell to suppress log messages on Colab output. This would be necessary sometimes because log messages might hang the colab session. And this could act as a convenient switch.

4. Ensure you press “Refresh” each time you wish to see the files under the current Colab file structure:



4. The folder structure need not be preserved when loading the images into the **IMFDB\_train\_sorted.txt** and **IMFDB\_test\_sorted.txt files.** For example, though the celeb Aamair Khan’s photo’s are shared in the following format format: **AamairKhan/3Idiots/images/Amir\_103.jpg** , a simple **AamairKhan/Amir\_103.jpg** is also sufficient from the perspective of how the code handles the structure. The folder name “**AamairKhan”** is however critical as that determines the name of the label.

5. You can use the following script (run it on colab by substituting <your folder name> with the name of the folder in colab where you have your images; i.e. mostly your own name ) to generate the file list that you have to add to IMFDB\_train\_sorted.txt and IMFDB\_test\_sorted.txt files:

**for residingFolder,\_,allFiles in os.walk("./data/IMFDB\_final/<your folder name>"):**

**for files in allFiles:**

**print(residingFolder.split('/')[-1]+"/"+files)**

6. One of the keys of cross deployment is that you can train your models on GPU (i.e in Colab) but deploy on a CPU device (such as the server aiml-sandbox.talentsprint.com which is listening to the mobile app). Therefore when you load your state\_dictionary into the model using torch.load , you have to ensure that you map the model to the CPU device. Here is the link to understand this issue: <https://discuss.pytorch.org/t/on-a-cpu-device-how-to-load-checkpoint-saved-on-gpu-device/349>

7. When you pass a single input image to a model to get the feature representation, it is important to ensure that your input image dimension matches the dimension of the first layer of the model. Say **face1** is your input image tensor, you can consider using **torch.unsqueeze(face1,0)** or **face1.unsqueeze\_(0)** i.e. adding an empty dimension.

8. The transforms used Siamese experiment based on IMFDB data (i.e. the experiment you worked on in Lab) is the default which is provided in the face\_recognition\_model.py; (and expression\_recognition\_model.py). If you are using the AT&T Dataset (and the related experiment), you will need to modify the transforms accordingly.

9. **A note about Siamese dataloader**:

The dataloader's job is to give random pairs of images so that Siamese network can be trained. It has been found in the past that participants of the hacakthon get stuck at the point of uploading their images (as they get index out of bound exception) to the Siamese to the existing images and successfully training their images. Imagine trying to work out a machine learning hackathon and being stuck during the dataloading phase. Providing a brief overview of the logic of this program and how you can avoid some common mistakes: (**it is recommended you attempt creating a new class with your name and pick up any existing celeb images and put them there and try data loading. Finally during hackathon you can replace these images with those that you collect from mobile app). This will make understanding what is written below easy. YOU MUST aim at finishing the data loading for your new class before the hacakthon itself or at least on the first day.**

Logic:

a. The input is a python 'list' of filenames (train\_list\_file parameter in the code contains the fileName/location)

b. The idea is to get two indexes i.e. 'index' , 'index2' such that they pull different file names from the list. Step 3 describes how they are selected.

c. To randomize the selection of these two indexes, it first picks a random selection of 'index' ..say the value is 500.

i) now a range of 300 (i.e. range=300) is created, by selecting min and max as i1 and i2; such that i1=500-range ; i2= 500+range.

ii) now a random index2 is picked from this range i..e between 500-300 and 500+300 i..e 200 and 800.

iii) Few details that are taken care in the code w.r.t logic above: to ensure that index is not equal to index2; Also i1 will always be less than i2 due to min/max logic.

iv) you might wonder how negatives are handled?Well, negative number means backward indexing in Python. So there is no problem.

d. Once we have index and index2, we merely check if the first string in the filename exists within self.classes. I.e for example: if your fileName is Aamir/someFolder/image123.jpg, then the first string is 'Aamir' is considered as your class name (note how it doesn't matter what you second folder name is); And that should **exactly (with case sensitivity)** be present in **self.classes** . Which is why it is important that after you add your images to the folder, make an entry into the image-list file, you **have to** make **exact** entry of the class name (i.e. the folder name under which you place your images) into siamese\_data\_loader.py (example given below).

*elif classes == 'IMFDB':*

*self.classes = ['AamairKhan', '****MyCaseSensitiveNameExactlyMatchingFolderName****', 'Rimisen', 'Kajol', 'KareenaKapoor','RishiKapoor', 'AmrishPuri', 'AnilKapoor', 'AnupamKher', 'BomanIrani', 'HrithikRoshan', 'KajalAgarwal', 'KatrinaKaif', 'Madhavan', 'MadhuriDixit', 'Umashri', 'Trisha']*

e. So finally if both the index and index2 point to first string from **same folder** (and hence same class name), then label =0 ; if index and index point to a first string from **different folder**( and hence different class name), the label=1

f. NOTE: whenever you make changes to siamese\_data\_loader.py or adding images to the list file etc, do re-run the trainloader and testloader cell, so that you changes are reflected in your code. Else, the data is not loaded with your new logic (or you new data is not loaded with the existing logic; whichever is applicable).