Homework 3: **Read Me if YOU Can**

1. The task

Unicredit asked to you to provide them a software that given a scanned picture with a character (i.e, a,A,**B**,***B*,...**) is able to recognize:

1. The font
2. The character
3. If it is in **bold**
4. If it is in *italics*

For Example:

A ⇒ (Amatic, A, 1, 1)

Thus, your goal is to provide Unicredit a PoC (Proof of Concept) that your model works properly. In particular, these are the required deliverables:

1. The model (that is going to be secretly tested)
2. A document (*max length 2 pages*) that describes your model and the choices you took during its development.

2. Cluster Preliminaries

Read this section only when you have to train the model on the cluster.
We recall that the development of the model is on your laptop.

1) In general, before being able of doing anything on the cluster:
- Connect to the VPN
    - usr: surname.studentID
    - psw: 0000+ google authenticator password

2) The line to execute the first time you start to work with the cluster is:

```
ssh corso@172.16.34.129 ./init.sh STUDENTID
psw: dmt2018
```

If you have any problem or, at any time, you want to restart everything from zero, you can run this command. It will remove your folder and create it again.

This will create your own folder of the following structure:

```
student_id
     | ------ data
             | ------ fonts

     | ------ data_out
             | ------ test
             | ------ model
             | ------ img
     | ------ src
```

Please, do not change the structure of your own folder because, the evaluation of your model will totally depend on it.

3) You will develop your code on your laptop and then mount it on the cluster using this command

```
scp LOCAL_FILE corso@172.16.34.129:./students/STUDENTID/src/
```

where:

- LOCAL_FILE: is the name (including the path) of the file to copy

Scripts will have to be copied to the folder src. From where the scripts will be runned. Indeed, src is the working dir of your code. Refer to the structure above to properly load and save data.

4) Once you define the code of your model and you are sure to not have bugs, mount it on the cluster as we described above!
You can create classes of modules, as you prefer. But remember that we will run only the *main.py* to create and train the model and *test.py* to test the model.
To run the main.py you will have at most 15 minutes, if you need more time, it means that the model/data are too big… it's not necessary. As soon as you request to run your code, if one of the two gpu is available then it will run, otherwise you will wait the first available slot.

To **run a script** (not the main), run the following command:

```
     ssh corso@172.16.34.129 ./run_script.sh STUDENTID
FILE_NAME.py
```

to **run the main.py**, that is saved in the src folder

```
ssh corso@172.16.34.129 ./run_main.sh STUDENTID
```

To run the **test** on the trained model, run the following command:

```
ssh corso@172.16.34.129 ./test.sh STUDENTID OUTPUT_FILE_NAME
```

5) Once you will find the best model, copy back the folder to your laptop:

```
scp -r corso@172.16.34.129:./students/STUDENTID local_folder
```

3. The Pipeline

Since it is your first serious NN, we will try to sketch a pipeline than can help you throughout the development.
Our pipeline is composed by 3 steps:

1. Dataset creation
2. Build and train the model
3. Run the model and test

3.1 Create the dataset

3.1.1 The idea

Unfortunately, Unicredit has not provided any data to train our model. Therefore, you need to create them by yourself. Practically, you generate a .npy file, containing the numpy array with the b/w luminescence representation of the images, *i.e.* a numpy array (n_samples x 64 x 64 x 1), example. And create another file with the ground truth on the mages, *i.e.* a y (array, dict, Dataframe) with shape (n_samples,4) to be recalled later to train the model.

Be careful, sometimes the characters you pass to the scanner have not perfect definition. Thus, to have a better model you should let it know about this possible scenario. Often, when you deal with images, the *data augmentation* is a necessary step to do in order to tackle the described scenario. You are not required to use tensorflow to compute all the operations, but any package/function you want.

3.1.2 How to

To directly generate the characters as pictures, we suggest you to install the package Pillow and explore it. Each image should have size 64x64 px.

Recalling that a model might change according to the data you feed in, we suggest you to be careful in the evaluation phase. In particular, avoid to compare the performances of different models on different datasets. Indeed, you can change the data but eventually you should compare the models trained on the same set of images.

As you see we are not giving any specific information about the number of images and the data augmentation procedures to adopt. There is not a unique and perfect way to do it. Thus, feel free to do what you consider more appropriate, but of course, consider also that the quality of your choices is going to be evaluated.

3.1.3 Files

The code you use to generate the images is gathered in:

- **`data_generator.py`:**
  create all the possible combinations of font/character/bold/italic and save the matrix representation of the images into a .npy file in the data_folder.
  Shape of the numpy array = (n_samples,64,64,1)
- **`data_augmentation.py`**
  load the file created in the previous step, augment the data and save the matrix representation into a .npy file in the data folder

The images coming from **`data_generator.py`** are the well defined characters. Instead, those generated from **`data_augmentation.py`** are those coming from the data augmentation step.

The files **`data_generator.py`** and **`data_augmentation.py`** have to be stored in the folder scr/

The list of all the possible fonts is provided in the folder `data/fonts/` and for local implementation you can download here. - **be careful:** not all the fonts have bold and italics.

The list of all the possible characters:

```
import string
string.printable[:-6]
```

3.2 Build and train your model

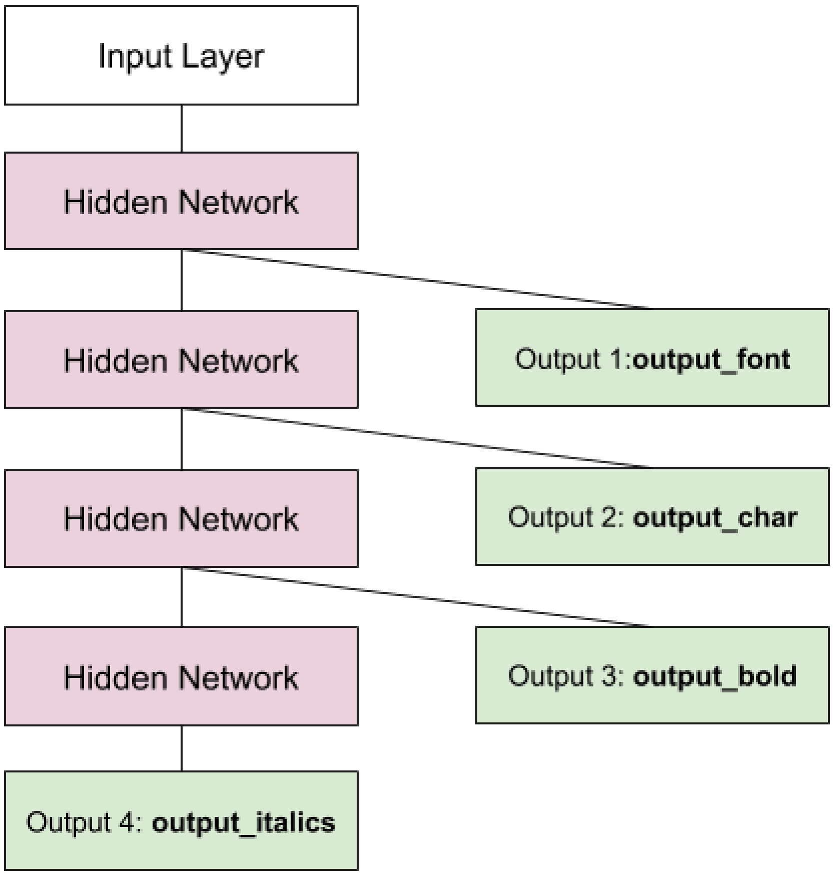
3.2.1 Model

The Convolutional Neural Network will be a multi-tasking CNN, with two task being multi class classifier. Indeed, we will be able to recognize:

- The font, the character, if it is bold, if it is italics

Remember to save the OneHotEncoder in order to load it when running the test.py and reconstruct the class from the predicted vector. Or to save the mapping (string-int) for the font and char features.
Given as input a list of matrices (image) n_samplesx64x64x1 the model should be able to tell us:

- The font as *string*
- The character as *string*
- If it is bold as *boolean*
- If it is italics as *boolean*



Very important. When you define the output layer be sure to set the attribute "name" and give them the name you find in the previous picture (i.e. *output_char*). Take a look here.

Due to the different kind of outputs, the loss function of Output 1 and Output 2 is the *Multiclass Cross Entropy.* Instead, for Output 3 and Output 4 is the *Binary Cross Entropy.*

Save model summary:

```
from contextlib import redirect_stdout

with open('../data_out/model/modelsummary.txt', 'w') as f:
      with redirect_stdout(f):
          model.summary()
```

3.2.2 How to

To develop the CNN you should use Keras Functional API. It is **extremely important** (it will have an impact on the evaluation) you know what your code exactly does. Thus, we ask you to comment the code and to be ready (during the oral session) to answer questions related to the implementation of the network.

3.2.3 Files

You are free to create as many .py files as you want and need. They must be afterwards called as libraries in the:

- `main.py`

This file contains the code to train the model. Any of the created `.py` files you mount on the cluster will have to be placed in `src/`.

The trained model should be saved as shown here (line 28-33 of the first snippet). In particular, you need to store `model.json` and `model.h5` into the folder `data_out/model.`

3.3 Test the model!

In order to test the model, run the ssh command described above. You should be sure to have you folder composed like this:

```
student_id
          | ------ data
                  | ------ fonts
          | ------ data_out
                  | ------ test
                  | ------ model
                         | ------ model.h5
                         | ------ model.json
                         | ------ modelsummary.txt
                  | ------ img
          | ------ src
                  | ------ data_generator.py
                  | ------ data_augmentation.py
                  | ------ main.py
                  | ------ test.py
                  | ------ NAME_LIB.py
                  | ------ NAME_LIB.py
                  | ------ ...
```

The **test.py** has to be a script that takes as input a path/to/file/X_data.npy and a ouput_file_name. It runs the prediction on the given dataset and saves it as a csv file with this path `data_out/test/ouput_file_name.csv`

Ex: `python test.py path/to/file/X_data.npy ouput_file_name`

So the steps are:

- load the .npy file, *i.e.* `np.load(path/to/file/X_data.npy)`
- load the keras model
- make the predictions
- save the predictions as a csv file into ../data_out/test/OUTPUT_FILE_NAME.csv, one per each row (no header):
    - `char,font,bold,italics`
    - `i.e. : ~,Pacifico,1.0,1.0`

Save the test.py script into the src folder with the other scripts (through the scp command).
To run the test, run the ssh command,

- `ssh corso@172.16.34.129 ./test.sh STUDENTID OUTPUT_FILE_NAME`

it will print the results of your model on the 33% of the hidden test set.

4. Extra points!

As extra points we ask you to do the following tasks:

1. Create new_model with intermediate outputs and the same weights of the trained model to see how the hidden layers behave. The intermediate outputs (.png format) should be saved into `data_out/img/`
2. Perform a 90% of accuracy on the entire hidden test set.
   Accuracy is a weighted mean of the four accuracies: font and char accuracy are weighted 0.3 each; italics and bold 0.2 each. (The sum is 1)
3. Surprise us! Do something special and creative! (Describe it in the report)

5. Deliverables

1. You should deliver us a folder that must have the following format:

```
| ------student_id
          | ------ data
                  | ------ fonts
          | ------ data_out
                  | ------ test
                         Partial_result.csv
                  | ------ model
```

```
                    | ------ model.h5
                    | ------ model.json
                | ------ img  (in the case you do Extra Point 1)
                    | ------ img_intermediate_1.png
                    | ------ img_intermediate_2.png
                    | ------ ...(max 10 png)

        | ------ src
                | ------ data_generator.py
                | ------ data_augmentation.py
                | ------ main.py
                | ------ NAME_LIB.py
                | ------ NAME_LIB.py
                | ------ …
        | ------ Report.pdf
```

Where:

5. `model.h5 and model.json` contains the model you want to submit
6. `img_intermediate_X.png` is the intermediate image obtained from the submitted model

All the files have to be the same that are in the cluster.

- A report that contains the description of:
  - Detailed description of the dataset (number of train samples, number of samples for each class, etc)
  - The data augmentation you did (how did you decide to augment the number of samples?)
  - Description of the final model you submit.
  - Performances of the model on your own validation set and on the test run on the cluster.
  - Comparison with models that do not perform well as the one you eventually pick (report of the achievements of previous models that you have been changing)
  - To those who do Extra Point 1 we would like to see the image you get in the intermediate step and an interpretation of what goes on

IMPORTANT. The length of the report can be maximum 2 pages for those that do not do any Extra Point. Otherwise the max is three pages.

The name of the zip file must have this format:
`DTMT4BaS_2018__HW_3__STUDENTID_NAME_SURNAME__STUDENTID_NAME_SURNAME.zip`
Finally you must send the ".zip" file to gentili@diag.uniroma1.it with the following email subject:
`DMT4BaS_2018__HW_3__StudentID_StudentName_StudentSurname_StudentID_StudentName_StudentSurname.`

Important notes:

- Deadline: 21/06/2018 @08:00 a.m.
- Don't create a file png per each character, but only a .npy file with the stored images as a numpy array.
- I won't answer emails, only piazza questions, so that before asking a questions … have a look on previous questions.
- Use only one STUDENTID per group. Write the other STUDENTID only on the email object and in the zip file name.
- Useful trick: reproduce a equal replica of the tree structure of the folder in your laptop, run the code always from the src folder

The homework in 3 lines:

- Develop your code in your PC, follow the structure, respect the input parameters for the test.py script (the others have no inputs) and then send and run it into the cluster
- Essentially are 4 steps: create data, augment them, (have a coffee break), create train and save a model, test the model. Remember
- Consider at least 1 day per each session + 1 extra day for the model + 1 day for the cluster + 1 day for the report … tot 7 days.

Best of luck!!!