Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning

Instructor: Purushottam Kar
Date: 26 June, 2023
Total: 100 marks

ASSIGNMENT
2

1 What should I submit, where should I submit and by when?

Your submission for this assignment will be one PDF (.pdf) file and one ZIP (.zip) file. Instructions on how to prepare and submit these files is given below.

Assignment Package:
`https://www.cse.iitk.ac.in/users/purushot/courses/ml/2022-23-s/material/assn/assn2.zip`
Deadline for all submissions: 11 July, 2023, 9:59PM IST
Code Validation Script: `https://colab.research.google.com/drive/1B8ZkTIM21_Ug1Ls5HST5-xMD0MreMhjR?usp=sharing`
Code Submission: `https://forms.gle/fCKzyk3sbdriFr2K9`
Report Submission: on Gradescope
There is **absolutely no provision** for "late submission" for this assignment

1.1 How to submit the PDF report file

1. The PDF file must be submitted using Gradescope in the *group submission mode*. Note that this means that auditors may not make submissions to this assignment.

2. Make only one submission per assignment group on Gradescope, not one submission per student. Gradescope allows you to submit in groups - please use this feature to make a group submission.

3. Ensure that you validate your submission files on Google Colab before making your submission (validation details below). Submissions that fail to work with our automatic judge since they were not validated will incur penalties.

4. Link all group members in your group submission. If you miss out on a group member while submitting, that member may end up getting a zero since Gradescope will think that person never submitted anything.

5. You may overwrite your group's submission (submitting again on Gradescope simply overwrites the old submission) as many times as you want before the deadline.

6. Do not submit Microsoft Word or text files. Prepare your report in PDF using the style file we have provided (instructions on formatting given later).

1.2 How to submit the code ZIP file

1. Your ZIP file should contain atleast one Python (.py) file called `predict.py`, your learnt model (in raw form or pickled form) and any supporting files e.g. non-standard libraries, additional code etc.

2. The main Python file sitting inside the ZIP file must be named "predict.py" and it must implement a method called `decaptcha` that takes in a list of strings (that are meant to be interpreted as filenames) and outputs a list of strings (see below as well as the `dummy_submit` directory present in the assignment package).

3. Do not submit Jupyter notebooks or files in other languages such as C/Matlab/Java. We will use an automated judge to evaluate your code which will not run code in other formats or other languages (submissions in other languages may simply get a zero score).

4. Password protect your ZIP file using a password with 8-10 characters. Use only alphanumeric characters (a-z A-Z 0-9) in your password. Do not use special characters, punctuation marks, whitespaces etc in your password. Specify the file name properly in the Google form.

5. Remember, your file is not under attack from hackers with access to supercomputers. This is just an added security measure so that even if someone guesses your submission URL, they cannot see your code immediately. A length 10 alphanumeric password (that does not use dictionary phrases and is generated randomly e.g. 2x4kPh02V9) provides you with more than 55 bits of security. It would take more than 1 million years to go through all $> 2^{55}$ combinations at 1K combinations per second.

6. Make sure that the ZIP file does indeed unzip when used with that password (try
   `unzip -P your-password file.zip`
   on Linux platforms).

7. Upload the password protected ZIP file to your IITK (CC or CSE) website (for CC, log on to `webhome.cc.iitk.ac.in`, for CSE, log on to `turing.cse.iitk.ac.in`).

8. Fill in the Google form linked above to tell us the exact URL for the file as well as the password.

9. Make sure that when we visit the URL you have given, there is actually a file to be downloaded. Test your URL on a browser window to verify that you have not submitted a corrupted URL.

10. Do not host your ZIP submission file on file-sharing services like GitHub or Dropbox or Google drive. Host it on IITK servers only. We will autodownload your submissions and GitHub, Dropbox and Google Drive servers often send us an HTML page (instead of your submission) when we try to download your file. Thus, it is best to host your code submission file locally on IITK servers.

11. While filling in the form, you have to provide us with the password to your ZIP file in a designated area. Write just the password in that area. For example, do not write "Password: helloworld" in that area if your password is "helloworld". Instead, simply write "helloworld" (without the quotes) in that area. Remember that your password should contain only alphabets and numerals, no spaces, special or punctuation characters.

12. While filling the form, give the complete URL to the file, not just to the directory that contains that file. The URL should contain the filename as well.

    (a) Example of a proper URL:
    `https://web.cse.iitk.ac.in/users/purushot/mlassn1/my_submit.zip`

(b) Example of an improper URL (file name missing):
`https://web.cse.iitk.ac.in/users/purushot/mlassn1/`

(c) Example of an improper URL (incomplete path):
`https://web.cse.iitk.ac.in/users/purushot/`

13. We will use an automated script to download all your files. If your URL is malformed or incomplete, or if you have hosted the file outside IITK and it is difficult for us to download automatically.

14. Make sure you fill in the Google form with your file link before the deadline. We will close the form at the deadline.

15. Make sure that your ZIP file is actually available at that link at the time of the deadline. We will run a script to automatically download these files after the deadline is over. If your file is missing, we will treat this as a blank submission.

16. We will entertain no submissions over email, Piazza etc. All submissions must take place before the stipulated deadline over the Gradescope and the Google form. The PDF file must be submitted on Gradescope at or before the deadline and the ZIP file must be available at the link specified on the Google form at or before the deadline.

Problem 2.1 (HexaCAPTCHA). CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) are popular ways of preventing bots from attempting to log on to systems by extensively searching the password space. In its traditional form, an image is given which contains a few characters (sometimes with some obfuscation thrown in). The challenge is to identify what those characters are and in what order. In this assignment, we wish to crack these challenges.

Your Data. Each CAPTCHA image in this assignment will be 500 × 100 pixels in size. Each image will contain a code that is a 4 digit hexadecimal number. The font of all these characters would be the same, as would be the font size. The Latin characters A-F will always be in upper case. However, each character may be rotated (degree of rotation will always be either $0°, \pm 10°, \pm 20°, \pm 30°$ and each character may be rendered with a different color. The background color of each image can also change. However, all background colors are light in shade. Each image also has some stray lines in the background which are of varying thickness, varying color and of a shade darker than that of the background. These stray lines are intended to make the CAPTCHAs more "interesting" and realistic.

Hexadecimal numbers are written in base 16 i.e. there are 16 "digits" instead of the usual 10. These digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The last six digits have values going from 10 to 15 i.e. A = 10, B = 11, C = 12 etc. Thus, the hexadecimal number 10 is equal to the decimal number $1 \times 16^1 + 0 \times 16^0 = 16$ and the hexadecimal number DECAF is equal to the decimal number $D \times 16^4 + E \times 16^3 + C \times 16^2 + A \times 16^1 + F \times 16^0 = 13 \times 16^4 + 14 \times 16^3 + 12 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 912559$. Your task is to figure out whether the 4 digit hexadecimal number in the image is odd or even.

Your task is to predict for each image, whether the hexadecimal number in that image is even or odd. If the number is even, your output should be the string "EVEN" (without quotes) else your output should be the string "ODD" (without quotes). Take care not to make spelling or case mistakes. To take the above two numbers as examples, the hexadecimal number 10 is even since it corresponds to the decimal number 16 which is even whereas the hexadecimal number DECAF is odd since it corresponds to the decimal number 912559 which is odd.

You have been provided with 2000 training images (see the subdirectory `train`) in the assignment package. The correct parities (odd/even) for the hex number present in each image are collected in the file labels.txt. In addition to this you have also been given reference images (see the subdirectory `reference` in the assignment archive) for all 16 characters 0-9, A-F in a standardized form with a white background. This is to simply give you a handy reference on what unrotated images of characters look like if all color information is removed. You may use these reference images in any way you wish, including as training or validation data i.e., there is no restriction on their usage.

Some Observations. Some of the following observations may be useful in solving the problem. The following facts are true both of the train images that you have been provided with the training package as well as the (secret) test images on which we will evaluate your submissions.

1. Exact identification of the entire hex number is not necessary to decide its parity (oddness/evenness).

2. The border color of a character is always a darker version of the fill color of that character.

3. Although the characters may be rotated, these rotations are not with arbitrary angles. In fact all rotations are known to be either $0°, \pm 10°, \pm 20°, \pm 30°$ in both train and test images.

4. The obfuscating lines in the background are of significantly less thickness than the thickness of the characters themselves.

5. All these images were generated using the Python Imaging Library (PIL) and no professional CAPTCHA code was used to generate data for this assignment. Thus, the number of obfuscating transformations possible were pretty limited and as such it should be possible to invert those to recover the true code in each image.

6. The dimension of each image is 500 × 100 pixels.

Your Task. Your task is to design an algorithm that can take a set of such 500 × 100 images and return the parity of the hexadecimal number in that image.

1. Describe the method you used to find out what characters are present in the image. Give all details such as details of the model being used (e.g. linear, decision tree, neural network, kernel etc), the training algorithm used, including hyperparameter search procedures, validation procedures. You have to give a detailed explanation even if you used an algorithm/implementation from the internet – make sure to give proper credit to the person/source from where you took code/algorithm. There is no penalty for using someone else's code/algorithm but there would be heavy penalty for doing so without giving proper credit to that person/source.

2. Train your chosen method on the train data we have provided (using any validation technique you feel is good). You may or may not wish to use the reference images – it is your wish. Store the model you obtained in the form of any number of binary/text/pickled/compressed files as is convenient and write a prediction method in Python in the file `predict.py` that can take new images and use your model files you have stored to make predictions on that image. Include all the model files, the `predict.py` file, as well as any library files that may be required to run the prediction code, in your ZIP submission. You are allowed to have subdirectories within the archive.

Evaluation Measures and Marking Scheme . Part 1 needs to be answered in the PDF file itself and part 2 needs to be answered in the ZIP file submission. We will evaluate your method on our secret test data which will have 500 × 100 images, each having a 4 digit hexadecimal number with similar kinds of rotations and background obfuscations as the training images. The 60 marks for part 2 will be awarded as follows:

1. Total size of the submission, once unzipped (smaller is better): 10 marks

2. Average time taken to process a test image (smaller is better): 10 marks

3. Parity match (procedure given below): 40 marks

Marks for parts 1 and 2 would be awarded in a relative manner after it is known how all groups have performed in terms of model size and prediction time. However, marks for part 3 would be awarded in an absolute sense. We will simply take the fraction of test images on which your method gave the correct parity and multiply that by 40 to give you marks for part 3.

1. Your method must output, for a test image, the parity of the 4 digit hexadecimal number in that image. The parity must be spelt out in upper case Latin. Please see the `labels.txt` to see how to output the parity correctly. Make sure there are no spelling mistakes while writing the parity

2. Please make sure your code does run with the validation script on Google Colab. Try running your evaluation model and `predict.py` file with a small set of 100-200 images from the train set to check for coding glitches or other errors such as pickling errors or model read errors.

Suggested Methods. Various techniques may be used to successfully solve this problem. Take care to use powerful and nonlinear techniques e.g. kernels, deep learning as the final step. Expecting a deep network to take the whole image and solve the problem in one shot may be tempting, but also unnecessary and may end up giving larger model size and prediction times. We leave the final decision on the method to you, but offer some useful steps below.

1. **Identify background pixels**:
   (a) Notice that background pixels are always a light shade whereas the obfuscating lines are always a darker shade. Can you learn from the training set what is the distribution of shade/color of the background pixels?
   (b) To do so first you will have to extract the background color of an image so that you can perform estimation. However, this should be easy since the corners of the image are almost always background.
   (c) You may want to represent pixel color in the HSV/HSL format instead of the more popular RGB format to easily identify shade.
2. **Identify pixels that belong to obfuscating lines**:
   (a) Can you similarly learn a distribution on the shade/color of the obfuscating lines?
   (b) Another technique that may help eliminate obfuscating lines is a step called *erosion*. Check this step out in the opencv library. This step will thin out lines so that the obfuscating lines may get very thin or even disappear. You will have to learn a good value of erosion parameter.
   (c) Using any method (including but not limited to the above), once you have found a way to identify which pixels in an image belong to obfuscating lines, use your method(s) to eliminate obfuscating lines in images.
3. **Segment image into pieces, each containing one char**:
   (a) You may want to perform *segmentation* (a form of pixel clustering in images to club together adjacent pixels that are similar) to find out how many characters are present in an image. There are several segmentation techniques available (several are preimplemented) in the opencv library.
   (b) Another technique that may help you segment the image into pieces is by looking for vertical columns of pixels that contain very few non-background pixels. This should be easy once we have learnt to identify background pixels in the previous stage. The reason this helps is that regions where there is a char, a lot of the pixels are non-background which makes it clear that a char is present.
4. **Use ML models to identify the char in each piece – hint: identifying every piece is not necessary to solve the problem**:
   (a) You may want to remove color information from the image at this stage since colors cannot help you identify which char is present in a piece (or can they?). Also, try to trim the piece so that it contains mostly the char and not too much blank space.

(b) The reference images may help here since the rotations which have been applied to the chars are not arbitrary but small multiples of 10 degrees.

(c) To exploit the above, why not create rotated versions of the reference images and see which one most closely matches the char within a given piece?

(d) An alternative is to instead attempt to rotate back the piece itself so that it matches one of the standard reference images.

(e) Try simple (linear and/or non-linear) techniques to classify every piece into one of the 16 characters. May treat this as a multiclass problem.

Resources The OpenCV library `https://opencv.org/` provides routines for most operations you may need to solve this problem. OpenCV is well integrated into Python and available via `pip`. More challenging versions of this DeCAPTCHA problem are problems of identifying license number plates on cars (e.g. `https://sod.pixlab.io/articles/license-plate-detection.html`) and identifying house numbers from images of doors (e.g. `https://www.kaggle.com/stanfordu/street-view-house-numbers`) etc. You may search for approaches people have used to solve those problems to get motivation. However, be warned that the solutions to those problems may be an overkill for our much simpler problem and blindly using those techniques may result in large model sizes or prediction times, both of which would cause you to lose marks.

Validation on Google Colab. Before making a submission, you must validate your submission on Google Colab using the script linked below.

Link: `https://colab.research.google.com/drive/1B8ZkTIM21_Ug1Ls5HST5-xMDOMreMhjR?usp=sharing`

Validation ensures that your `predict.py` does work with the automatic judge and does not give errors due to file formats etc. Please use IPYNB file at the above link on Google Colab and the dummy test directory (details below) to validate your submission.

Please make sure you do this validation on Google Colab itself. **Do not download the IPYNB file and execute it on your machine – instead, execute it on Google Colab itself.** This is because most errors we encounter are due to non-standard library versions etc on students personal machines. Thus, running the IPYNB file on your personal machine defeats the whole purpose of validation. You must ensure that your submission runs on Google Colab to detect any library conflict. **Please note that there will be penalties for submissions which were not validated on Google Colab and which subsequently give errors with our automated judge.**

Dummy Submission File and Dummy Test Data. In order to help you understand how we will evaluate your submission using the evaluation script, we have included dummy test data, a dummy model and a dummy submission file in in the assignment package itself. However, note that the dummy test data is just a subset of the training data.

Using Internet Resources. You are allowed to refer to textbooks, internet sources, research papers to find out more about this problem. However, if you do use any such resource, cite it in your PDF file. There is no penalty for using external resources so long as they are acknowledged but claiming someone else's work (e.g. a book or a research paper) as one's own work without crediting the original author will attract penalties.

Restrictions on Code Usage. You are allowed to use all standard Python libraries e.g. numpy, sklearn, scipy, libsvm, keras, tensorflow as well as 3rd party libraries. However, if you use someone else's code, do give them proper credit by citing them prominently in your PDF file. There is no penalty for using external code so long as they are acknowledged. **However, if you are using a non-standard library that is not available via pip and which is essential for prediction on test data, you must supply that library to us in your submission.**

(40 + 60 marks)

2 How to Prepare the PDF File

Use the following style file to prepare your report.
`https://neurips.cc/Conferences/2023/PaperInformation/StyleFiles`

For an example file and instructions, please refer to the following files
`https://media.neurips.cc/Conferences/NeurIPS2023/Styles/neurips_2023.tex`
`https://media.neurips.cc/Conferences/NeurIPS2023/Styles/neurips_2023.pdf`

You must use the following command in the preamble

```
\usepackage[preprint]{neurips_2023}
```

instead of `\usepackage{neurips_2023}` as the example file currently uses. Use proper LaTeX commands to neatly typeset your responses to the various parts of the problem. Use neat math expressions to typeset your derivations. Remember that all parts of the question need to be answered in the PDF file. All plots must be generated electronically - no hand-drawn plots would be accepted. All plots must have axes titles and a legend indicating what the plotted quantities are. Insert the plot into the PDF file using proper LaTeX `\includegraphics` commands.

3 How to Prepare the Python File

The assignment package contains a skeleton file `predict_skeleton.py` which you should fill in with the code of your prediction method. This method should first load the model file that you should have included in your submission, and then make predictions.

1. Before submitting your code, make sure you validate on Google Colab to confirm that there are no errors etc.

2. You do not have to submit the evaluation script to us – we already have it with us. We have given you access to the Google Colab evaluation script just to show you how we would be evaluating your code and to also allow you to validate your code.