Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning

Instructor: Purushottam Kar
Date: October 30, 2019
Total: 120 marks

ASSIGNMENT 3

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:
`http://web.cse.iitk.ac.in/users/purushot/courses/ml/2019-20-a/material/assn3.zip`
Deadline for all submissions: 23 November, 9:59PM IST
There is no provision for "late submission" for this assignment

1.1 How to submit the PDF 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. Gradescope allows you to submit in groups - please use this feature to make a group submission.

3. To clarify the above, for example, there are, as of today i.e. Oct 30 2019, 47 assignment groups. We wish to have only 47 submissions on Gradescope, i.e. one submission per assignment group, not one submission per student.

4. URL link for groups `https://web.cse.iitk.ac.in/users/purushot/courses/ml/2019-20-a/material/assn_groups_16Oct2019.pdf`

5. 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.

6. 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.

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

1.2 How to submit the ZIP file

1. 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. Name your ZIP file `submit.zip`. Specify the file name properly in the Google form.

2. 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.

3. Make sure that the ZIP file does indeed unzip when used with that password (try
   `unzip -P your-password file.zip`
   on Linux platforms). If we are unable to unzip your file, you will lose marks.

4. 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`).

5. Fill in the following Google form to tell us the exact path to the file as well as the password
   `https://forms.gle/44nU8vpKzf3hBB9C6`

6. Do not host your ZIP submission file on file-sharing services like Dropbox or Google drive. Host it on IITK servers only. We will be using a script to autodownload your submissions and if not careful, Dropbox or Google Drive URLs may send us an HTML page (instead of your submission) when we try to autodownload your file. Thus, it is best to host your code submission file locally within IITK servers.

7. 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.

8. 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/mlassn3/submit.zip`

   (b) Example of an improper URL (file name missing):
   `https://web.cse.iitk.ac.in/users/purushot/mlassn3/`

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

9. 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, then we will not bother to make any efforts to manually locate your file or else contact you for clarifications. We will simply give your group a zero in these cases.

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

11. 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 simply assign your group zero marks.

12. We will entertain no submissions or mercy appeals 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 Python file must be available on the link specified on the Google form at or before the deadline.

Problem 3.1 (DeCAPTCHA). 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 600 × 150 pixels in size. Each image will contain a code composed of either 3 or 4 characters, all of which would be uppercase Latin alphabet characters (i.e. no numerals, punctuation marks or lowercase characters). The font of all these characters would be the same, as would be the font size. However, each character may be rotated a bit 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 problem more interesting.

You have been provided with 2000 training images (see the subdirectory `train`) in the assignment package. The true code present in each image is the filename of that file itself. For instance, if the characters present in an image are "M", "L", "O" and in that order, then the filename of that image is `MLO.png` itself. Thus, the filename of training images gives the true code present in those images. Note however, that this is just for sake of your convenience and we will obviously not have the codes present in the filenames of the test images while testing your submissions.

In addition to this you have also been given reference images (see the subdirectory `reference` in the assignment archive) for all 26 characters 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

1. The border color of a char is always a darker version of the fill color of that char.
2. Although the chars may be rotated, these rotations are not with arbitrary angles. In fact all rotations are known to be small (randomly chosen) multiples (clockwise or counterclockwise) of 10 degrees. Can you guess which multiples were used in the dataset? The same set of multiples were used to generate train data and test data.
3. The obfuscating lines in the background are of significantly less thickness than the thickness of the characters themselves.
4. 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 the image.

Your Task. Your task is to design an algorithm that can take a set of such 600 × 150 images and return two pieces of information for each image: the number of characters in the code in that image. This will be either 3 or 4, and the code present in that image.

1. Describe the method you used to solve both the problems i.e. counting how many chars are there in the image and then finding out what those characters are in what order. Give all details such as 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 (of number of characters and actual characters) 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.

(40 + 80 marks)

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 600 × 150 images, each having 3 or 4 characters with similar kinds of rotations and background obfuscations as the training images. The 80 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. Predicting the number of characters: 20 marks

4. Code 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 parts 3 and 4 would be awarded in an absolute sense, as outlined below.

To evaluate how correctly did your code predict the number of characters in an image, we will calculate the fraction of test images in which your code correctly predicted the number of characters. We will multiply that fraction by 20 to award you marks out of 20 for part 3.

To evaluate how correct is the code returned by your method for an image, we will take the following steps

1. If the code returned by your routine for an image contains more than 4 characters, we will only take the first 4 characters and discard the rest (thus, we will truncate your code at 4 characters).

2. We will then compute the length of the longest common subsequence between the true code and your (truncated) code. Suppose this length is $\hat{l}$ and the true length of the code is l then you will receive a (fractional) score of $\frac{\hat{l}}{l}$ for this image.

3. We will sum these scores over all test images, divide this sum by the number of test images and multiply that by 40 to award you marks out of 40 for part 4.

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. **Learn to 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. **Learn to 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. **Identify the char in each piece**:
    - (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 (including non-linear) techniques to classify every piece into one of the 26 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.

Code Usage from the Internet. You have full freedom to download code available online (written by others) for training without any penalty (even if you are able to directly use someone's code without making any changes to it). However **you must cite all code that you take from others** by giving names of authors and name of the URLs from where you obtained said code. There is no penalty for using code for which you have cited the original author. However, there will be heavy penalty for using someone's work and not giving them credit.

2 How to Prepare the PDF File

Use the following style file to prepare your report.
`https://media.neurips.cc/Conferences/NeurIPS2019/Styles/neurips_2019.sty`

For an example file and instructions, please refer to the following files
`https://media.neurips.cc/Conferences/NeurIPS2019/Styles/neurips_2019.tex`
`https://media.neurips.cc/Conferences/NeurIPS2019/Styles/neurips_2019.pdf`

Use the command `\usepackage[preprint]{neurips_2019}` in the preamble instead of `\usepackage[final]{neurips_2019}`. Use proper LaTeX commands to neatly typeset your responses to the various parts of the problem. Use neat math expressions to typeset any mathematical derivations. 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 ZIP File

Your submission ZIP archive must contain a file called `predict.py` which must contain a method by the name `decaptcha` which we will call to decode the CAPTCHA images. We will pass a list of strings to the method `decaptcha`, each string corresponding to the path to a file name. The method must return a `numpy` array of numbers (containing predictions for the number of characters in each image) as well as a list of strings (containing the predictions for codes present in those images). The assignment package contains a sample submission file `sample_submit.zip` which shows you how you must structure your submission as well as showing you the way in which your prediction code `predict.py` must accept input and return decodings. Do not change this input output behavior otherwise our autograder will not be able to process your recommendations and award you zero marks.

1. Your ZIP archive itself must contain a python file named `predict.py` which should not be contained inside any other directory within the archive. Apart from this file your ZIP archive may contain any number of files or even directories and subdirectories.

2. Do not change the name `predict.py` to anything else. The file must contain a method called `decaptcha` which must take in a list of filepaths to images and return a `numpy` array of integers and a list of strings.

3. There are no "non-editable regions" in any of the files for this assignment. So long as you preserve the input output behavior, feel free to change the structure of the file as you wish. However, do not change the name of the file `predict.py` to anything else.

4. Code you download from the internet may be in C/C++. You are free to perform training in C/C++ etc. However, your prediction code must be a python file called `predict.py`. This file may internally call C/C++ code but it is your job to manage that using extension modules etc. Do not expect us to call anything other than the single file `predict.py`

5. The assignment package also contains a file called `eval.py` which is an example of the kind of file we will be using to evaluate the code that you submit. Before submitting your code, make sure to run `eval.py` and confirm that there are no errors etc.

6. Do not submit `eval.py` to us – we already have it with us. We have given you access to the file `eval.py` just to show you how we would be evaluating your code.

7. Make sure that running `predict.py` does not require us to install any Python libraries which are not available using `pip`. Libraries that are installable using `pip` are fine. However, if your prediction routine requires using external code not available using `pip`, you must supply all that external code to us within your ZIP archive – we should not have to install it. To be sure, we will not install or download any libraries from places like GitHub, personal homepages, or other repositories (even the repository mentioned in the question itself) even if you want us to. Your archive itself should contain all such files.

8. Other than the above restrictions, you are free to import libraries as you wish,e.g. pil, sklearn, scipy, pandas. Files for libraries not available on `pip` must be provided by you.

9. Once we have installed `pip` libraries required by your code, we should just be able to call `predict.py` and get predictions i.e. your prediction code must be self contained. We should not have to create any new directories, move files around or rename them. All of that must be done by you while creating the archive (remember, you are allowed to have as many files and (sub)directories apart from the file `predict.py` as you want).

10. We do not need your training code for part 2. We simply need your prediction code for part 2. Do not increase your submission file size (remember there are marks for submission size too) by including unnecessary files.

11. The system on which we will run your prediction code will **not have a GPU**. Make sure your code can run simply on a CPU. Thus, use the CPU version of deep learning libraries if you are using them and not the GPU version of those libraries e.g. keras.

12. You may use a GPU (if you can find one) to train your method but your prediction method must still work on a CPU. To be sure, we will not offer your prediction code a GPU even if you ask us to. Do not use any libraries that require GPU access during prediction. There are no restrictions on using GPU for training but we will not provide your code GPU access during prediction.

13. We will test your submitted code on a secret dataset which would have images of the same size as the training images. All test images will contain either 3 or 4 characters each of which would be an uppercase Latin alphabet character.