CS771 Assignment 3

VINAY AGRAWAL, ARABINDA KARMAKAR, MANISH KUMAR GHILDIYAL, AYUSH KOTHIYAL, VAMSHIKIRAN BHASKAR MORLAWAR

TOTAL POINTS

94 / 100

QUESTION 1

1 Method Description 40 / 40

✓ **+ 40 pts** **Method description**

- 15 pts Insufficient method description

+ 0 pts No submission or else very sparse description

QUESTION 2

2 Experimental Results 54 / 60

+ 0 pts Correct

+ 54 **Point adjustment**

GROUP NO: 30

Grading scheme for code:
Submission size s (in KB): s < 128 (10 marks), 128 <= s < 512 (8 marks), 512 <= s < 1024 (6 marks), s > 1024 (4 marks)
Inference time t (in sec): t < 5 (10 marks), 5 <= t < 20 (8 marks), 20 <= t < 40 (6 marks), t > 40 (4 marks)
Code match c: floor(c * 40) marks

s = 3903.91 KB: 4 marks
s = 1.47 sec: 10 marks
score = 1.0 : 40 marks
TOTAL: 54 marks

CS771 : ASSIGNMENT 3

Group Members
Arabinda Karmakar (22111011)
Ayush Kothiyal (22111015)
Manish Kumar Ghildiyal (22111039)
Vamshikiran Morlawar (22111066)
Vinay Agrawal (22111068)

1 QUESTION:

Describe the method you used to find out what characters are present in the image. Give all details such as algorithm used including hyperparameter search procedures, validation procedure.

SOLUTION:

Firstly, while training we have a set of 2000 CAPTCHA images (150×500 pixels) each containing 3 upper case Greek characters with each one of them being rotated at an angle of either one of $0°, \pm 10°, \pm 20°, \pm 30°$.

Also there are some obfuscating lines in the background which are of comparatively darker shade than the background colour (which might also vary). So, first of all some preprocessing of image data is needed to be done in order to extract the 3 individual Greek characters from the CAPTCHA image which is also free from any stray lines in the background.

1. Preprocessing of Image (Data Cleaning):

So, firstly to remove the background colour and then to remove the obfuscating lines from the background we have used 2 functions:-

- The first function takes the 'BGR' image as input and returns an image with pure white background. To do this it first converts the image into 'HSV' format. Then it takes the 'S' value and removes any value which is less than its half. Then it uses S as mask into a single Foreground. Then we invert the foreground to get the background, convert the background back into the BGR form and then apply foreground map to the original image. Finally, we combine the foreground and the background to get the image with white background which contains stray lines.
- Now, since we have an image with no background colour, we use the second function to remove the obfuscating lines. This is also a similar function to the first one. The only difference is that instead of using 'S', it uses 'V' i.e. intensity value of HSV image. Any value where V > 170 will be part of mask (this threshold value 170 was found by hit and trial i.e. by seeing for which value the stray lines were nicely removed without tampering the letters). Then V is used as mask into a single foreground. Then we follow similar procedure as above and combine foreground and background to get final image which is free from stray lines.
- Website referred for above two functions i.e. to remove the background colour and obfuscating lines :

  "https://www.freedomvc.com/index.php/2022/01/17/basic-background-remover-with-opencv/"

Now, we have an image (150×500 pixels) consisting 3 Greek upper case characters with clear background and no obfuscating lines. The next step is to segment the image into 3 parts each containing a single character. The steps followed for it are as follows:-

- Firstly, we convert the image into Black and White so that each pixel value is either 0 or 255.
- Then we calculate the average pixel intensity value at each of 500 columns.
- With the average pixel value across each column we get to know that, where average = 255, there is no letter and when average < 255, there is a letter.
- Now, we can markup those 6 points across columns where each point corresponds to start and end of each of 3 characters.
- Finally, we divide a 150×500 image into 3 parts each containing one character. Each of these 3 image is of size ($150 \times k$ pixels) where k is some constant value which is different of each image.

After this, since each of the image is of different size, we 'Down Sample' each of the image into (25×25 pixels) so that we have same size feature vector for all 3 images. Hence, after flattening it we get a **feature vector of size 625** corresponding to each of the 3 letters of an image. So, now since we have divided each image into 3 parts we have the total data set of 2000*3 = 6000 single character images with 625 features each.

Then we assigned labels (0 to 23) to each of 24 characters which forms the 24 classes. After this, labelling of data set is done i.e. we assigned these numbered labels to each of the CAPTCHA image in the data set. Just to note each of this image can also be rotated to a certain angle in multiples of $10°$ between $\pm 30°$.

2. Deciding Technique to Solve the Problem:

- First, we tried using non-linear technique which is using **deep learning** to solve the given problem since it was easy and there were various resources available that used the same approach to solve the similar problem. Also the accuracy achieved in this case was very high as it was almost equal to 100%. But the problem here was that the size of the model was coming around **136 MB** which is very huge.

  Note : This accuracy was checked on the random split of the 6000 single character images where 80% of image data was used as training set and rest 20% was used as validation set. This train data was split just to check for accuracy in a better way. In final model, we will use the train images only to train and use the 2 CAPTCHA's available in test folder for testing.
- Since, the model size was very large, we thought of using SVM(Support Vector Machine) to solve the problem after referring to the website :

  "https://www.kaggle.com/code/nishan192/mnist-digit-recognition-using-svm"

  By author : "NISHAN PATEL", which used SVM for similar looking problem of Digit Recognition.
- So, then we tried using C-Support Vector Classification (SVC), with **kernel = 'rbf'** and all the other hyperparametes as default. This was done on a random train test split of 80% and 20%. The accuracy was exactly 100%. The prediction time was also significantly less than the previous approach. But, we found out that the complexity of RBF kernel grows with the size of training set.
- Hence, we also used SVC with **kernel = 'linear'** and all other hyperparameters as default i.e. **C= 1.0, degree =3, class_weight = None and max_iter = -1 (no limit)**. Here, on same train/test split as above we again got accuracy of 100% with almost same prediction time. Since, the accuracy was very good we didn't change the values of any hyperparameters.
- **Hence, final technique used : C-Support Vector Classification (SVC) with kernel = 'linear'.**
- Here, when we trained on 1600 single character images the model size was around **7.9 MB**. So, with an idea of trying to reduce the model size further, we thought of reducing the size of training set. Then we trained our model on 20% data (i.e. 1200 images) and tested on the

rest of 80% of the data. The accuracy did not drop and it was still 100%. Plus the model size was now around **3.89 MB**, which is considerably very small.

Hence, our final model was trained on the 1200 (20%) random single character noise free images (which might be rotated) and we tested it on the 2 test images given and achieved 100% accuracy. Final Model Size = 3.893 MB.

2 Question

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.

Solution:

Code submitted in Zip File.

1 Method Description 40 / 40

- ✓ **+ 40 pts** **Method description**
- **- 15 pts** Insufficient method description
- **+ 0 pts** No submission or else very sparse description

rest of 80% of the data. The accuracy did not drop and it was still 100%. Plus the model size was now around **3.89 MB**, which is considerably very small.

Hence, our final model was trained on the 1200 (20%) random single character noise free images (which might be rotated) and we tested it on the 2 test images given and achieved 100% accuracy. Final Model Size = 3.893 MB.

2 Question

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.

Solution:

Code submitted in Zip File.

2 Experimental Results 54 / 60

+ **0 pts** Correct

+ **54 Point adjustment**

GROUP NO: 30

Grading scheme for code:
Submission size s (in KB): s < 128 (10 marks), 128 <= s < 512 (8 marks), 512 <= s < 1024 (6 marks), s > 1024 (4 marks)
Inference time t (in sec): t < 5 (10 marks), 5 <= t < 20 (8 marks), 20 <= t < 40 (6 marks), t > 40 (4 marks)
Code match c: floor(c * 40) marks

s = 3903.91 KB: 4 marks
s = 1.47 sec: 10 marks
score = 1.0 : 40 marks
TOTAL: 54 marks