A Deep CNN Model For Student Learning Pedagogy Detection Data Collection Using OCR

Arnab Sen Sharma
Computer Science and Engineering
Shahjalal University of Science & Technology
Sylhet-3114, Bangladesh
Email: arnab.api@gmail.com

Maruf Ahmed Mridul
Computer Science and Engineering
Shahjalal University of Science & Technology
Sylhet-3114, Bangladesh
Email: mridul133@gmail.com

Marium-E-Jannat
Computer Science and Engineering
Shahjalal University of Science & Technology
Sylhet-3114, Bangladesh
Email: jannat-cse@sust.edu

Md Saiful Islam
Computer Science and Engineering
Shahjalal University of Science & Technology
Sylhet-3114, Bangladesh
Email: saiful-cse@sust.edu

***Abstract*—Student learning pedagogy detection requires a huge amount of data from students. Efficient process to collect the data is a major fact here. This paper proposes an approach based on Convolutional Neural Network (CNN) for Optical Character Recognition (OCR) and mainly shows a method to use this OCR system to extract information of a student filled in a specialized form. This form contains 170 cells. Some of these cells are to be filled with capital English alphabets and others are to be filled with English numerals. This paper discusses a method for feature extraction and use of CNN to identify each cell. Using this method we could predict 96.87% of numeric data and 94.36% of alphabetic data accurately.**

***Keywords*—optical character recognition, feature extraction, convolutional neural network(CNN), pedagogy , EMNIST , SUST:BNHD.**

I. Introduction

Handwritten character recognition has become a major research interest because of its wide range of applications. Digitization of handwritten documents as well as camera captured documents require Optical Character Recognition (OCR) the most. As these tasks are very often needed, automation of these can't be denied any more. In this paper, we discuss how we propose to extract information of students filled in a special form. These information will be used for student learning pedagogy detection. That means, identifying the best way to teach students based on the information they've given. We propose a method to extract characters of 37 classes (26 uppercase letters, 10 digits, and space) from a specially designed form and finally to recognize those. Each of the forms yield 170 cells of these 37 classes

In a visual recognition task like this, the main concern is how the features are extracted and which method is used. There are some methods like Support Vector Machines (SVMs), Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Kth nearest neighbor algorithms etc. As it turned out, CNNs work good with images where pixel values of an image is used as features.

For the recognition task, a deep CNN (Convolutional Neural Network) is trained with various open source datasets we found online as well as our collected data. An elegant preprocessing system of two steps (form-preprocessing and character-preprocessing) is used for achieving a higher accuracy rate.

To the best of our knowledge till now, character extraction and recognition from a form of this size is a completely new work. We suppose, this work will be useful in future for similar works in a larger context.

A. Literature Review

Perhaps the most popular OCR engine available to us is ***Tesseract*** [1]. This software was originally developed at Hewlett-Packard Laboratories Bristol and recognizes characters of more than 100 languages. This performs really good on printed clean images but does not perform very good in images of handwritten characters.

Dipti Singh , Mohd. Aamir Khan, Atul Bansal , Neha Bansal used SVM for optical character recognition [2]. They extracted the boundary of the character using Moore Neighborhood tracing and then applied chain code. C. De Stefano, F. Fontanella , C. Marrocco, A. Scotto di Freca proposed a Genetic Algorithm based feature extraction process to recognize handwritten characters [3].

Chunpeng Wu, Wei Fan, Yuan He, Jun Sun, Satoshi Naoi proposed a handwritten character recognition method using *Relaxation Convolutional Neural Networks(R-CNN)* and *Alternately Trained Relaxation Convolutional Neural Network(ATR-CNN)* [4]. R-CNN increases the total number of parameters and ATR-CNN regularizes the parameters during training process. They achieved an error rate of 0.254% on MNIST numerical dataset using this approach. Muhammad

978-1-5386-8207-4/18/$31.00

Naeem Ayyaz, Imran Javed, Waqar Mahmood proposed a hybrid feature extraction process [5]. They used a multiclass SVM for the recognition task an achieved an accuracy of 96.5% for english numeral and 96% for english alphabet.

Durjoy Sen Maitra, Ujjwal Bhattacharya and Swapan K. Parui used a 5-layer CNN as feature extractor and multiclass SVM to perform the classification or recognition task [6]. For english numeral they achieved an accuracy of 99.10%. Li Chen, Song Wang, Wei Fan, Jun Sun, Satoshi Naoi proposed CNN approach [7]. They trained multiple CNN based models and used voting of these models for the final recognition task. Md. Musfiqur Rahman Sazal, Sujan Kumar Biswas, Md. Faijul Amin, and Kazuyuki Murase proposed a *Deep Belief Network(DBN)* for recognizing handwritten Bangla numerals [8]. Parshuram M. Kamble, Ravinda S. Hegadi [9] used Rectangle Histogram Oriented Gradient representation as the basis for feature extraction. Then they applied *SVM* and *Feed Forward Artificial Neural Network* techniques.

Ahnaf Farhan Rownak, Md Fazle Rabby, Sabir Ismail, and Md Saiful Islam [10] proposed an efficient way for segmentation of Bangla characters in printed document using curved scanning. Bishwajit Purkaystha, Tapos Datta, and Md Saiful Islam [11] used Deep Convolutional Neural Network for Bengali handwritten character recognition. They achieved a maximum of 98.66% accuracy for Bangla numerals with 10 classes and minimum of 89.93% for all Bangla characters with 80 classes. Shuvanon Razik, Evan Hossain, Sabir Ismail, and Md Saiful Islam [12] prepared a database of bangla handwritten numerals named SUST:BNHD. They used deep CNN for the recognition task.

B. Convolutional Neural Network (CNN)

CNNs are responsible for major breakthroughs in image classification and at the core of most of the Computer vision technologies today. The input layer of CNN expects the input to be a multi-dimensional vector. So, CNN has some convolution layers at the beginning. At each of the convolution layers *convolution* is applied to the multi-dimensional vector. *Convolution* is a process where multiple *filters* are slided on the convolution layer. This filters learn/extract different features from the multidimensional vector.

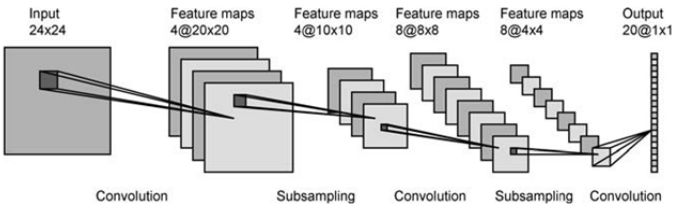


Fig. 1: Convolutional Neural Network[13]

Then the results are *feed-forwarded* in local connections. Output of a filter acts as the input vector of next layer. Each layer applies different filters and combines their result and passes the result to the next layer. This process is called *pooling*. Then a number of normal hidden layers of neural networks are added. They learn the feature vectors and performs the classification task.

II. Proposed Method

The whole method is divided into the following sub-parts - *A.Form Preprocessing, B. Character Extraction, C. Character Preprocessing, D. Feature Extraction, E. Recognition* and *F. Preparing Output*.

A. Form Preprocessing

In this part, the image of the form is aligned vertically, necessarily cropped and reshaped to an image of a general resolution. The form contains four circular markers in the four corners. First, we detect the markers and then crop the rectangle formed by the markers. Then we align it vertically using perspective transformation. Finally the form is reshaped into $1200px \times 1600px$.

1) Marker Detection: The image is converted to gray-scale and reshaped into $600px \times 800px$. Then average pixel values of each $20px \times 20px$ grid (as the diameter of the circular markers is $20px$ in the $600px \times 800px$. image) is calculated for the $200px \times 200px$ portions of four corners. For this task, cumulative sum (sum of the pixel values from the cell (1, 1) to the cell (i, j), for each cell (i, j)) is precalculated. Then, the total pixel values of a $20px \times 20px$ grid is calculated using-

$$tot = mat[RBx][RBy] - mat[RBx][LTy - 1] - mat[LTx - 1][RBy] + mat[LTx - 1][LTy - 1]$$

where,
mat = matrix containing the cumulitive sums
RBx = row number of right bottom pixel
RBy = column number of right bottom pixel
LTx = row number of left top pixel
LTx = column number of left top pixel

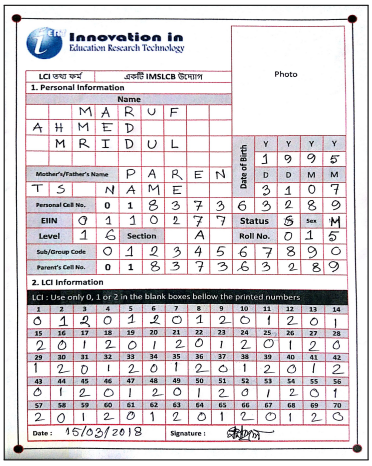
Finally, the average pixel value is calculated by dividing tot by $20 * 20 = 400$.

The $20px \times 20px$ grid with the minimum average value is considered as the **black** circular markers. Actually 10 grids from each of the four corners with the lowest averages are taken for safety and finally the $convexhull$ is taken for the desired four markers.

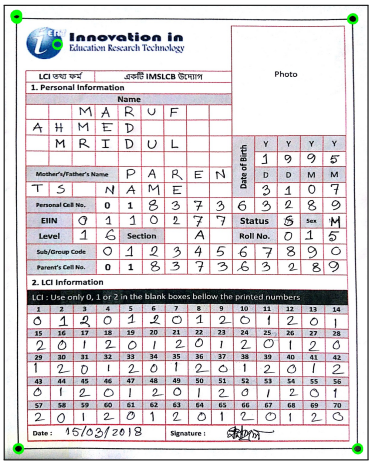
Then the co-ordinates are scaled up by the factor 2, as all operations are performed in a $600px \times 800px$ image whereas the expected resolution is $1200px \times 1600px$. The reason behind operating on the smaller image is time efficiency.

Finally, the four markers are detected on the $1200px \times 1600px$ image.

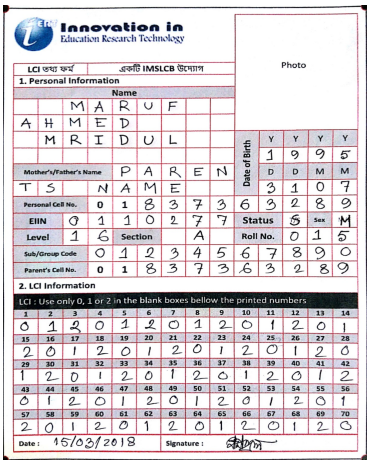
2) Cropping and Aligning: In this part, the rectangle formed by the four markers is cropped and perspective transformation is used to align the form vertically. *getPerspectiveTransform* and *warpPerspective* functions of *OpenCV* is used for this task.



(a) Initial Form



(b) Corner detection



(c) Cropped and Aligned

Fig. 2: Form Preprocessing

B. Character Extraction

The character cells are simply cropped from the upper left pixel to the lower right pixel. First, the upper left and lower right pixels of each row is defined manually. Then, corner pixels of each cell is determined by dividing the rows by the number of columns. Finally, needed cells are taken to count.

C. Character Preprocessing

Each extracted character is preprocessed trough these steps.

1) Noise Reduction: The input character image first undergoes a noise reduction process. *fastNlMeansDenoisingColored* of *OpenCV* is used for this purpose with *h_luminance = 10, photo_render = 10, search_window = 21* and *block_size = 7.*

2) Border Removal: The cell borders of the form is kept red with an intention of border removal. The character images may contain some borders in their backgrounds These red borders are removed in this step. The image is masked only with blackish pixels such that other colors go out. The range for the blackish pixels is [0, 0, 0] to [10, 10, 10] in terms of RGB values.

3) Grayscale Conversion: In this step the character image is converted to grayscale.

4) Thresholding and Background Elimination: The grayscaled image undergoes a thresholding and background elimination process simultaneously. The *threshold* function of *Python's OpenCV* is used with a *MaxValue* of 255 and of the type *THRESH_BINARY*. The image gets more bold and clear after this process. This can also be called *binarization*. *THRESH_BINARY* returns a binary image (Pixel values are 0 or 1 that means black or white) .

5) Cropping: The binarized image is then cropped in a way (from the leftmost black to the rightmost black and from the topmost black to the bottommost black) such that the presence of extra or unwanted border marks are minimized.

6) Adding Border: A white border of same size around the image is added with a view to keeping the character in the middle of the image.

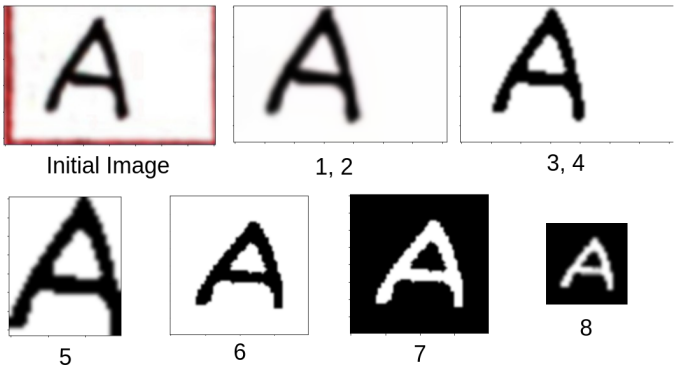


Fig. 3: Character Preprocessing

7) Color Inversion: Color is inverted such that the character becomes white in black background. This is just for following the formats of standard datasets we used to train our model.

8) Resizing: Images are then resized to a standard shape of $28px \times 28px$. This is done for maintaining a same size for all the images.

D. Feature Extraction

The only feature for the process is the pixel values of the preprocessed image. The pixel values are normalized by dividing by 255. Then we round the value(if the value is smaller than 0.5 then it is replaced by 0, otherwise 1). So, now we have an array of size 28*28 containing only 0s and 1s.

E. Recognition

We used CNN to recognize which character a character image represents. The recognition process is described in detail in the next chapter.

F. Preparing Output

We save the picture of the persons and their signatures as images and the data we extracted using OCR in a text file.

III. Optical Character Recognition

A. Expected data

In this part we discuss how we extracted the characters from images. The types of data expected from the form are shown in the table.

TABLE I: DETAILS OF THE FORM FIELDS

Name of the field	Number of blocks	Expected Characters
Name	36	A-Z
Father's/Mother's Name	14	A-Z
Date of Birth	8	0-9
Personal Cell No	9	0-9
EIIN	7	0-9
Level	2	0-9
Sub/Group Code	11	0-9
Parent's Cell No	11	0-9
Status	1	S or N
Sex	1	M or F
LCI	70	0 or 1 or 2

1) Identifying blank blocks: In the *Name* and *Father's/Mother's Name* fields, it is expected that some fields would be left blank. We identify those blocks in the preprocessing phase. We do not expect a block to be completely blank, because during cropping the individual cells, some part of the borderlines of adjacent cells may come. Also, there could be unintentional spots/marks. What we did is, after cropping the individual cells we took the middle 20*20 pixels and got the average value of pixels in that area. If that average value is below a predefined value we considered the cell as blank.

B. Models

From the table, it is clear that if a block expects a character from English alphabet we assume that the block will never contain a character from English numerals. Same thing is true for the blocks to be filled by English numerals.

So, instead of building a model that can identify all 36 characters(26 alphabets + 10 digits), we built two models, one for identifying alphabets and another for identifying digits. In that way we hoped to get higher accuracy. We also observed that the *LCI* field contains 70 blocks and the expected characters are only 0, 1 and 2. So, for this field we need not consider all the 10 numerals. So, for the sake of improving accuracy we actually built 3 models.

1) Model for identifying alphabets
2) Model for identifying digits
3) Model for identifying 0, 1, 2 (for LCI field)

1) Model for identifying alphabets:

a) Structure of the model: We tried several different CNN models for this task. Among this models one model performed relatively better than other. The structure of the model is given below.

- Input layer : 2D Convolutional layer with 32 filters which takes data of 28X28 grayscaled image. Activation method is *RelU*. *ReLU* stands for Rectified Linear Unit which represents the function below.

$$ReLU(x) = max(0, x)$$

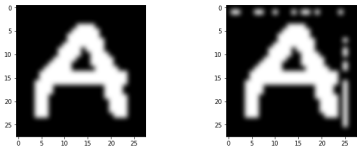
- Another 2D Convolutional layer of 32 filters which takes the features extracted by the input layer as input and normalizes it further.
- Pooling layer of 2X2 which scans throughout the image. This layer extract features and passes these features to the next layer using *ReLU* activation function.
- Allowed dropout 0.25 to avoid overfitting. Dropout is a technique which randomly omits some units or datapoints of a layer in a iteration.
- A dense hidden layer of 512 neurons. This layer learns the features and performs the actual classification task.
- Another dropout of 0.5 to reduce overfitting.
- The final layer is the output layer, where the number of neurons are the number of classes. In this case the number of classes is the number of letters in English alphabet. The activation function used in this layer is *softmax*. Softmax activation function is given below.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} x_j}$$

This function distributes the probability over all the classes. For one character sample, we take the class which has the maximum probability value.

b) Dataset: We trained and tested our model on two datasets. One of them we collected from online and we collected another dataset from students of CSE(Computer Science and Engineering department), SUST(Shahjalal University of Science and Technology). Below is a short description of those datasets.

- A-Z Handwritten Alphabets in .csv format from kaggle [14]. This dataset contains 372048 images of capital handwritten alphabets. Images of this dataset was clean with shape 28*28 where the character is centered at middle 20*20 space. But in our case it is expected that some of the border or fragments of adjacent cells to come into the 28*28 image scope. So, we modified this dataset a little bit to match our expected data. We randomly added border and other noises.



(a) Original (b) Border Added

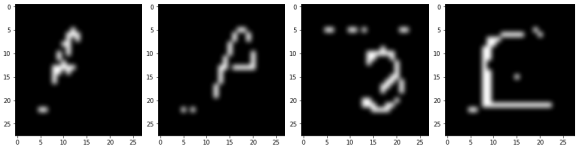
Fig. 4: Alphabet Data

- We collected data from 150 students of CSE department, SUST. This dataset contains more than 20,000 pictures of images of alphabets, more than 750 images per character.

c) Training and Testing: We combined these datasets. Split this combined dataset in two parts. 80% data was used for training and other 20% for validation. We do not test our model using the dataset we preserved for training and validation because the bulk portion of this dataset contains *clean* images we got from online datasets and testing it using this dataset will give us a biased result.

So for testing , we filled 40 forms each containing 6 images per letter(240 images per letter , total 6240 images) to see how our model performs in real time.

d) Result and Analysis: During training the model showed us an validation accuracy of 98.27%. But this accuracy was deceptive, because majority of the images were from the dataset we gained from Kaggle. Even after adding borders, these images was comparatively much cleaner from our extracted images. When we tested our model against the 40 forms we kept as our testing dataset. We got an accuracy of only 94.36%. The performance of this model for the task at hand was not very impressive. During extraction of cells and preprocessing we lost some data and some images got very unclear.



(a) Actual: A, 1st pred: F, 2nd pred: Q (b) Actual: A, 1st pred: P, 2nd pred: R (c) Actual: B, 1st pred: J, 2nd pred: O (d) Actual: E, 1st pred: C, 2nd pred: F

Fig. 5: Badly Affected Alphabets

This could happen for a varity of reasons. The camera quality of the smartphone may be not good enough, there may be some shadow over the effected part, glitch of pen, unintentional marks left during the fill up of the form etc. This effected some of the data badly.

Sometimes even we could not retrieve letters from images. These noises hampered the performance of the model, but still deep CNN outperformed all the traditional machine learning approaches.

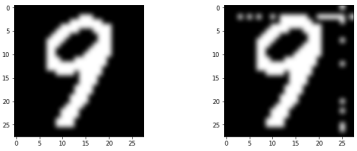
2) Model for identifying digits(10 digits + 3 digits(for LCI field)): We needed two models for digit recognition. One model recognizes all 10 digits, whereas other model recognizes only 3 digits(0, 1, 2). As the structure of these two models are almost same and we used almost same dataset to train these models here we discuss these two model in one scope.

a) Structure of the model:

- Input layer : 2D Convolutional layer with 64 filters which takes data of 28X28 grayscaled image. Activation method *ReLU*.
- Pooling layer of 2X2 which scans throughout the image. This layer extract features and passes these features to the next layer using *ReLU* activation function.
- Allowed dropout 0.2 to avoid overfitting.
- A dense hidden layer of 180 neurons. This layer learns the features and performs the actual classification task.
- The final layer is the output layer, where the number of neurons are the number of classes(10 for normal digit recognition , 3 for the digits in LCI field). Like the alphabet model the activation function used in this layer is *softmax*.

b) Dataset: We trained and tested our model on two datasets. Below is a short description of those datasets.

- EMNIST dataset [15]. This dataset contains 240000 images of 10 digits. Each image of size 28*28 pixels. Like the Kaggle alphabet dataset we added border to the images of this dataset too.



(a) Original (b) Border Added

Fig. 6: Digit Data

- We collected data from 62 students of CSE department, SUST. This dataset contains more than 9,000 pictures of images of alphabets, more than 860 images per character. We used our specialized form for collecting this images.

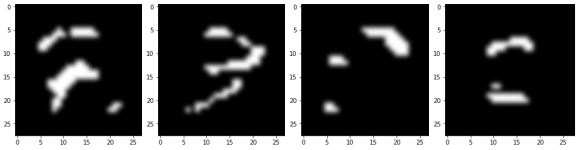
c) Training and Testing: We combined these datasets. Like alphabet recognition model we split this combined dataset in two parts. 80% data was used for training and other 20% for validation.

Like the alphabet model we did not test the model using the dataset for training. We filled 20 forms each containing 14 images per letter(280 images per digit, total 2800 images) to see how our model performs in a real time scenario.

d) Result and Analysis: During training the model showed us an accuracy of 99.38%. But like alphabet model, this accuracy was deceptive too. Because here majority of the images were from the dataset we gained from EMNIST.

We got an accuracy of only 96.87% for 0-9 and 98.77% for 0-2, when we tested the model against the images we collected.

The reason for this drop of performance is same as the alphabet model. Some of the images were distorted badly.



(a) Actual: 8 1st pred: 7, 2nd pred: 8 (b) Actual: 9, 1st pred: 3, 2nd pred: 9 (c) Actual: 9, 1st pred: 0, 2nd pred: 9 (d) Actual: 2, 1st pred: 1, 2nd pred: 3

Fig. 7: Badly Affected Digits

IV. Our models vs State of the art models

We used traditional CNN as our base model. But there are some state of the art CNN models available like ***Xception***, ***InceptionV3***, ***ResNet50***, ***DenseNet*** etc. These models have a very complex structure and they perform very well on the standard datasets. We did not use any of these models but tried our own models for recognition task for two reasons.

1) Because of the complex structure, the state of the art models are very heavy and resource hungry. We aimed for a model which would works on data from at least 10 thousand students. If we used the complex models we may have gotten a better result but they would surely consume much more time and resources for the processing of each form.
2) All these models expect the input image vectors to have 3 filters(Red, Green, Blue) whereas in our case both the online and offline datasets had images of only one filter (grayscaled images). Though this obstacle could be easily overcome by simply adding 2 dummy filter vectors to each image vectors, there is another problem. Most of the state of the art models require images of a minimum resolution to work. ***InceptionV3*** requires the images to have atleast a resolution of 50×50. If we take images of the form using smartphone, it is nearly impossible to ensure each extracted cell to have such high resolution. This problem could also be resolved by modifying the images using zero padding or image enhancement techniques. But we thought this would be an overkill and decided to make our own models for recognition task.

V. Conclusion

There are 170 cells in each of our forms. Using these forms we intend to collect data from around ten thousands students. Analyzing the data collected via these forms teachers would be able take better decisions on how to teach or help their students better. Since we are collecting data from so many students, if our model takes too much time processing one cell it would unacceptable. That is why we could not implement a more sophisticated preprocessing approach and a more complex model for the recognition task. Simple CNN with our preprocessing approach produced reasonably good results for us. If the image is good our system could identify all cells correctly. On our test data we could predict 96.87% of numeric data (0-9), 98.77% of numeric data (0-2) and 94.36% of alphabetic data accurately.

References

[1] (2018, Mar.) Tesseract open source ocr engine. [Online]. Available: https://github.com/tesseract-ocr/tesseract

[2] D. Singh, M. A. Khan, A. Bansal, and N. Bansal, "An application of svm in character recognition with chain code," in *Communication, Control and Intelligent Systems (CCIS), 2015*. IEEE, 2015, pp. 167–171.

[3] C. De Stefano, F. Fontanella, C. Marrocco, and A. S. Di Freca, "A ga-based feature selection approach with an application to handwritten character recognition," *Pattern Recognition Letters*, vol. 35, pp. 130–141, 2014.

[4] C. Wu, W. Fan, Y. He, J. Sun, and S. Naoi, "Handwritten character recognition by alternately trained relaxation convolutional neural network," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014, pp. 291–296.

[5] M. N. Ayyaz, I. Javed, and W. Mahmood, "Handwritten character recognition using multiclass svm classification with hybrid feature extraction," *Pakistan Journal of Engineering and Applied Sciences*, 2016.

[6] D. S. Maitra, U. Bhattacharya, and S. K. Parui, "Cnn based common approach to handwritten character recognition of multiple scripts," in *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*. IEEE, 2015, pp. 1021–1025.

[7] L. Chen, S. Wang, W. Fan, J. Sun, and S. Naoi, "Beyond human recognition: A cnn-based framework for handwritten character recognition," in *Pattern Recognition (ACPR), 2015 3rd IAPR Asian Conference on*. IEEE, 2015, pp. 695–699.

[8] M. M. R. Sazal, S. K. Biswas, M. F. Amin, and K. Murase, "Bangla handwritten character recognition using deep belief network," in *Electrical Information and Communication Technology (EICT), 2013 International Conference on*. IEEE, 2014, pp. 1–5.

[9] P. M. Kamble and R. S. Hegadi, "Handwritten marathi character recognition using r-hog feature," *Procedia Computer Science*, vol. 45, pp. 266–274, 2015.

[10] A. F. Rownak, M. F. Rabby, S. Ismail, and M. S. Islam, "An efficient way for segmentation of bangla characters in printed document using curved scanning," in *2016 International Conference on Informatics, Electronics and Vision (ICIEV)*. IEEE, 2016, pp. 938–943.

[11] B. Purkaystha, T. Datta, and M. S. Islam, "Bengali handwritten character recognition using deep convolutional neural network," in *Computer and Information Technology (ICCIT), 2017 20th International Conference of*. IEEE, 2017, pp. 1–5.

[12] S. Razik, E. Hossain, S. Ismail, and M. S. Islam, "Sust-bhnd: A database of bangla handwritten numerals," in *Imaging, Vision & Pattern Recognition (icIVPR), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[13] (2018, Jun.). [Online]. Available: https://i.stack.imgur.com/keDyv.png

[14] S. Patel. (2018, Mar.) A-z handwritten alphabets in .csv format. [Online]. Available: https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format

[15] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.