Project Report : CS 7643

Chaewon Park
Georgia Institute of Technology
cpark382@gatech.edu

Yueqiao Chen
Georgia Institute of Technology
ychen3221@gatech.edu

Weilong Shen
Georgia Institute of Technology
wshen61@gatech.edu

Borun Song
Georgia Institute of Technology
bsong74@gatech.edu

Abstract

Recognizing texts from images through Optical Character Recognition (OCR) has always been a popular area of research in deep learning as it is an effective and efficient way to transform text in images into digital text. Performing such tasks can be challenging when there is high variance in the dataset which commonly appears in natural scenes. For example, the text can be of various sizes, fonts, or can be laid on a complex background with varying colors and textures. With this project, we aimed not only to implement an efficient model that can make predictions well enough even with noisy datasets, but also to expedite the training process. The report presents our findings from experiments with accuracy statistics along with thorough analysis into the results.

1. Introduction/Background/Motivation

Traditionally, Optical Character Recognition (OCR) is a well-researched topic in identifying texts in images and converting them into digital texts. However, recognizing texts in natural scenes is a far more difficult computer vision task since there are countless ways in which a text can appear in such environments. For example, texts in natural scenes can be in various fonts, sizes, colors, and orientations. For that reason, traditional OCR generally did not outperform on natural scene character recognition. Hence, several previous works have proposed neural network architectures that performed equally well with different types of texts. Such architectures are often considered remarkable especially when they perform well with degraded text images, such as those that have low resolution, blurring, distortion, luminance variance, complicated background etc [1].

Convolutional Neural Network (CNN) has gained much popularity in documentation recognition tasks since the early 2000s [2], as CNN is proved to be tolerant to distortion of input images. Saidane and Garcia proposed a CNN architecture that automatically learns text recognition without pre- and post-processing nor tunable parameters [3]. The proposed architecture achieved nearly 68% of recognition rate on severely distorted text images in ICDAR 2003 data sets. In 2012, Zhu et al. [1] incorporated convolutional neural network and bimodal image enhancement to improve the performance of existing CNN architecture. Unlike Saidane's work [3], they presented the impact of preprocessing images to increase the consistency of data. With their method, the average recognition rate on the same dataset was 85.96%, but the rate on distorted texts was not reported separately. Yin et al. [4] utilized Maximally Stable Extremal Regions (MSERs) based methods to recognize text in natural scene images, and achieved 86.29% precision rate on ICDAR 2011 dataset. Atienza [5] introduced a model designed with 36 data augmentation techniques. When tested with several models proposed in six different papers [6] [7] [8] [9] [10] [11], the proposed method increased the recognition accuracy by 1.36% on average.

The true value of building a model to convert characters in natural scene images into plain digital text lies in its potential to make significant improvements in existing real-world applications. An accurate OCR model will help digitize texts from various images and make it more accessible and searchable which will bring positive changes to multiple domains. For example, it can be implemented in fields such as law enforcement, where the ability to quickly and accurately identify text from images could be critical for solving crimes. Moreover, many businesses could benefit from the accurate OCR model in that it can extract data from images such as invoices.

Hence, combining all the findings from previous research, we addressed challenges in natural scene text recognition tasks by developing a novel neural network architecture that synthesizes and enhances existing architectures proposed in multiple papers. We not only refined network architecture but also improved data consistency by exploring several data preprocessing techniques.

The dataset we used was from ICDAR 2003 Robust Character Recognition [12], which was initially for ICDAR 2003 Robust Reading competitions. It consists of images of characters with high variant image sizes. To better accommodate the CNN structures without crashing because of extreme image size, like images with one length or width of 1 pixel, we tried different resize methods and other data augmentation approaches for the pictures in the experiments. The dataset includes three folders of the images: one for the sample set with 845 images in the char_1 folder, one for the training set with 6185 images in the char_2 folder, and the last char_3 folder for the testing set with 5430 images. The labels for each folder are stored in the XML files accordingly. Through our implementation, we have found that there are 77 distinct labels in the label set. The labels are case-sensitive and could contain special characters. We choose the dataset because we are willing to try how to use CNN to solve street recognition with a high variance of image size. Most previous works used images of the same size, so we are looking into experimenting with different resizing methods.

2. Approach

Our approach is inspired by [13], which used the CNN approach on the same dataset with high accuracy, and [14], which proposed a CNN approach for scene text detection of the Urdu language. According to the two papers, we designed a convolutional neural network (CNN) with four layers for solving street scene character recognition.

We believe that this architecture can be successful because CNN architectures have shown success in previous research to be effective for street character recognition tasks. The use of convolutional layers to extract features from the input images is proven to be effective. Additionally, max pooling operations and ReLU activation functions help further enhance the extracted features' discriminative power.

The new research we carried out is about how to resize the images through images with high variances of image size. We tried out the three different resize methods. One is to stretch the images to squares with the desired size, and the other is to pad the images to the desired size. The third method is to use Torchvision transforms. It is shown in our experiments that resizing method 3 gives higher performance. The resizing methods will be discussed in the experiments section below.

In addition to a traditional fixed-sized CNN, we have also proposed a generic CNN with the aid of adaptive pooling to accept arbitrarily-sized input images. One concern we had when we design our CNN model is the horizontal and vertical resolution of images we should resize the input to. We have noticed that the resolution and aspect ratio of input varies greatly due to the fact that the original dataset was obtained from cropping street view images. The dataset contains images of extremely low resolution in one dimension or both, such as an image of size $W \times H = 1 \times 135$ (Figure 2) and $W \times H = 2 \times 16$ (Figure 3), and images of significantly higher resolution, with a largest image found in training dataset of resolution 589×898 (Figure 4).

Even though the size of convolution layers are invariant to the input image size, the fully connected layers that take in the output from the last convolution layer indeed depend on the size of the output, which indirectly depends on the resolution of input images. The number of input channels to the liner layers in our fully connected layer must be predefined and remain constant. In other words, the size of the flattened tensor that passed to the first linear layer must remain unchanged. This poses a challenge to our model as the resolution of the dataset varies greatly, which requires us to prepare the data so that the shape of the tensor reaching the fully connected layer is constant.

There are normally two approaches to tackle this issue, which are padding and resizing with interpolation [15]. To avoid the convolution layers from learning the padded information, zero padding is usually used for this scenario and all images are padded to the same size. Resizing with interpolation would resize all input images to the same resolution, and thus, when those images reach the fully connected linear layer, all images have the same size.

We have determined that padding is not a very practical solution, considering the lack of flexibility and enormous difference in size from the smallest image to the largest image. We could have padded all images to the resolution of the largest image in our dataset, which is of resolution 589×898, but the padding introduces a great number of futile works on the padded area that does not help training at all while wasting the computing power pointlessly.

Resizing is a more common solution, which has been adopted by all of those six aforementioned papers [6][7][8][9][10][11]. We have devoted most of our focus on resized images, as it is very easy to resize and has delivered considerably good accuracy for this given dataset. However, one key concern we have in our mind is that resizing would inevitably introduce artifacts and changes in aspect ratio. Since patterns presented in graphics of characters are mostly simple and contrastive, artifacts are not our main concern here. However, we do have concerns about losing spatial information from the change in aspect ratio. For example, it is fairly intuitive for humans to guess that a very thin and tall character is likely to be an "I" or "l". However, after resizing, we are concerned as the shape of the image has been changed significantly, especially for those images that have an aspect ratio W/H very far away from a square of $1/1$.

Consequently, we decide to attempt to add an adaptive

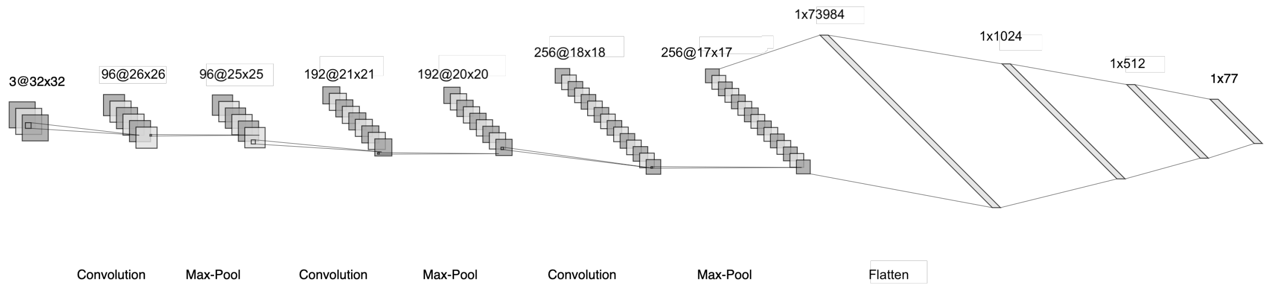

Figure 1. Proposed Model Architecture

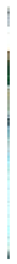

Figure 2. Image of shape 1×135

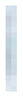

Figure 3. Image of shape 2×16 (Not to scale)

Figure 4. Image of shape 589×898 (Not to scale)

max pooling layer between the output of the last convolution layer and the input of the first linear layer in the fully connected layer. Adaptive max pooling would complete a certain number max pooling based on a fixed specified output size regardless of the input, and thus allows for variably sized input image and fixed size requirement of linear layer. Owing to limited computing power, we have decided to enforce the output size to be 20×20 to keep the model reasonably small and without losing too much information from max pooling. Considering the low training efficiency for enforcing variable sized input with non-batched training, we have also proposed variable sized batch approach, in which all images with the same size would be packed to the same batch to further accelerate training.

2.1. Network Layers

2.1.1 Layer 1

In the first layer, we used a convolutional operation with a kernel size of 7×7, an output channel of 96, and a stride of 1. It is followed by a rectified linear unit (ReLU) activation function and a max pooling operation with a kernel size of 2x2 and a stride of 1. This layer was designed to extract low-level features from the input images.

2.1.2 Layer 2

The second layer also used a convolutional operation with a kernel size of 5x5, an output channel of 192, and a stride of 1, followed by a ReLU activation function and a max pooling operation with the same kernel size (2x2) and stride of 1 as the previous layer. This layer was designed to extract median-level features from the input images.

2.1.3 Layer 3

In the third layer, we used a convolutional operation with a kernel size of 3x3, an output channel of 256, and a stride of 1, followed by a ReLU activation function and a max pooling layer with a kernel size of 2x2 and a stride of 1 as the previous layers. This layer is for extracting higher-level features from the input images.

2.1.4 Fully-connected Layer

Finally, we used a fully connected layer with a flattened output from the previous layer, followed by two linear layers with 1024 and 512 output units, respectively. We used ReLU activation functions after each linear operation and a dropout operation with a rate of 0.5 after the first linear operation to prevent overfitting. The final linear operation had 77 output units corresponding to the number of possible characters in our dataset.

2.1.5 Adaptive Pooling Layer

When the AdaptiveMaxPool layer is in use, we would temporarily remove the MaxPool layer in layer 3. As the pooling is changed, the input size to the FC layer has changed to 102400 to reflect the 20×20 max pool.

For sake of batched training, it is very crucial to ensure that all images in the same batch share the same resolution, so that the shape of the batched tensor would be $B \times C \times W \times H$, which effectively enforces all image to be unified to a shape of $W \times H$.

2.1.6 Variable Sized Batches

For sake of batched training, it is very crucial to ensure that all images in the same batch share the same resolution, so that the shape of the batched tensor would be $B \times C \times W \times H$, which effectively enforces all image to be unified to a shape of $W \times H$. Unfortunately, out of the 6185 images from the training data, most of them have a different resolution, and thus training batched data with a reasonably large batch size is virtually impossible. The easiest approach is to change the batch size to one, leaving each image being passed to the model individually. However, this has significantly limited the training performance as tiny batches may not fully utilize the parallelism and thus drags the training time from minutes to hours.

We have proposed a variably sized batch size by grouping all images with the same resolution into the same batch. Out of the 6185 images from training data, there are 2983 different sets of resolutions, which means that we could batch them into 2983 batches, each batch with a variable batch size depending on the number of images that share the same resolution. We have not yet seen any previous work on this procedure, and we have decided to compare its performance against a non-batched approach. We expect to see a training time reduced by a factor of two, but we are more interested in how our model tackles variably sized batches.

We have customized the get_item function in the dataloader to achieve customized batching. When initializing the dataloader, we iterate through all images in the given dataset and create a unique set of image files by resolution. This set is represented as a python dictionary object, each containing a list of images. Whenever an image of the same size is found, it would be added to the list corresponding to this image shape. Otherwise, a new list is created for images of this shape, which would be encoded with the tuple (W, H) as the key to the dictionary. Inside of the get_item function, we use the key array from the dictionary as the index to the batch, and we pack all images inside the list given the corresponding key to be one batch. We have effectively created variable sized batches by grouping all images of the same size to the same batch.

Owing to images with a number of pixels that are less than 16 in either dimension, it is possible that the shape of small images would shrink to 0 in one dimension, and thus we have decided to resize those smaller images to a size that is large enough to prevent this from happening while maintaining the aspect ratio W/H. We calculate the new shape of those images to resize by multiplying its original shape $W \times H$ by a factor of $32//MIN(W, H))$, resulting in a new image tensor of size $C, W \times 32//MIN(W, H)), H \times 32//MIN(W, H))$.

2.2. Model Design Analysis

During the design and implementation process, we anticipated potential problems with overfitting, as well as challenges related to tuning the hyper-parameters of the network. We encountered difficulties with our initial CNN which fluctuated at about 60% accuracy. We experimented by adding the number of filters, changing the kernel size, and adding the dropout layers, and we finally improved the performance by 10% accuracy in our final CNN architectures mentioned before. The dropout idea is from [14], which uses fully connected layers with a dropout layer. We tried adding the dropout layer to the first linear layer, which gives a more stable high performance at the end.

Overall, our approach to street scene character recognition using a CNN architecture with multiple convolutional and fully connected layers was successful in achieving high accuracy on our task, and we believe that the use of these layers and associated operations helped to effectively extract and discriminate between the features of the input images.

3. Experiments and Results

With the CNN structure proposed in the previous section, we conducted a series of experiments to optimize the performance of the model. We compared prediction accuracies on the test dataset to determine the optimal input image resizing method, hyperparameters as well as data augmentation techniques. First, we tuned hyperparameters of the model to achieve the highest accuracy possible with a fixed input preprocessing method (3.1.1). We then investigated the effect of different resizing methods explained in 3.1.2. Based on the results, we selected the best resizing method and hyperparameters to be used in subsequent experiments. In the next phase, we incorporated data augmentation techniques into the model and compared the accuracy of the model with and without data augmentation. Finally, we examined how our proposed model works with both case-sensitive and case-insensitive classification tasks. Our experimental variables and results are summarized below.

We will then present our attempt on modification to our original model with the aid of adaptive pooling in a separate section as slight modification to the model was involved. No major modification was made to our original model, especially for the convolutional layers. Due to the time limit, it is plausible that our model has not been fine-tuned to its best performance.

3.1. Independent Variables

3.1.1 Hyperparameters

See Table 1 for hyperparameters. Before feeding into the proposed model, we preprocessed input images to be of the same size (32x32 pixels) by stretching and shrinking. (Method 2 in section 3.I.B)

3.1.2 Resizing Methods

Resizing Method 1 Resize input images to be of the same size (Pixel 32x32). For portrait/landscape images, it stretches images to make it square-sized.

Resizing Method 2 Resize input images while keeping the (Height:Weight) ratio unchanged. Instead of stretching an image to make it square-sized, pad a solid-colored background to fill the empty space.

Resizing Method 3 Resize input images using Pytorch built-in method. (transforms.Resize)

3.1.3 Data Augmentation

See Table 2 for data augmentation methods as well as parameter values for each method used in the experiments.

3.2. Results

3.2.1 Hyperparameters

See Table 3 for accuracy of the model on test dataset with fine-tuned parameters. Refer to 1 for Experiment IDs and hyperparameter values.

3.2.2 Resizing Methods

See Table 4 for accuracy of the model depending on the resizing method used for data preprocessing.

3.2.3 Data Augmentation

See Table 5 for accuracy of the model for each data augmentation method. Hyperparameters and resizing method were chosen based on previous experiment results (3.2.1, 3.2.2).

3.2.4 Case-sensitive/insensitive Classification

Model was tested on both case-sensitive and case-insensitive classification tasks. See Table 6 for model accuracy on both tasks.

3.2.5 Adaptive Pooling and Variable Sized Batching

See Table 7 for accuracy of the model using Adaptive Max Pooling proposed in 2.1.5

3.3. Analysis & Conclusion

Before we start diving into experimenting with different data preprocessing methods, we would like to finetune our model so that it achieves the optimal performance. We found that hyperparameters play a considerable role in the model performance. With experiments 1, 2, 3, and 4, we found 5e-4 to be the optimal learning rate, and we observed that the model does not learn anything with a learning rate that is too high, and reducing the learning rate under the optimal rate increases the training time but does not significantly improve the performance. With experiments 8, 9, and 10, we found 64 to be the optimal batch size, and we observed that a low batch size increases the training time due to less vectorization while a high batch size causes the model to underfit. Additionally, we introduced weight decay and dropout to reduce overfitting. We observed that dropout is greatly effective in improving the model performance as adding a dropout layer with dropout probability 0.5 increases the test accuracy by 3%, as in experiments 10 and 11. On the other hand, weight decay regularization had no obvious influence on the model performance, so we decided to remove regularization in our model.

Once the model is finetuned, we move on to feed the model with data that are preprocessed differently. We tested on data that are preprocessed with 3 different methods, with the first two methods involving re-encoding the resized images to JPEG file form. We found out that stretching the images to a square before resizing performs slightly better on our model than the padding method, possibly due to the fact that padding introduces irrelevant information to the images. We also noticed that the data, after resized by the built-in PyTorch transform, performs slightly better than the first two possibly due to the loss of information introduced by re-encoding into JPEG. We also experimented with transforming the images with data augmentation techniques outlined above, but no performance improvements were observed.

We are surprised to see that no significant improvement was brought by adaptive pooling, but it even performs slightly worse than the regular approach if we choose to use padding. Our hypothesis is that the output size we set to the adaptive max pooling layer is smaller than that for our original model. In addition, in the case that there are

Experiment ID	1	2	3	4	5	6	7	8	9	10	11
Learning Rate	1e-3	5e-3	5e-4	1e-4	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4
Batch size	64	64	64	64	64	64	64	32	128	64	64
Weight Decay	1e-2	1e-2	1e-2	1e-2	0	1e-3	1e-1	0	0	0	0
Dropout	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0	0.5

Table 1. Hyperparameter Tuning

Method	Parameters
Color Jitter	Brightness = 0.5, Contrast = 0.5, Saturation = 0.5, Hue = 0.5
Gaussian Blur	Kernel size = 3, Sigma = 1
Random Affine	Degrees = (0, 30), Translate = (0.1, 0.1), Scale = (0.8, 1.2), Shear = (0, 20)

Table 2. Data augmentation methods & Parameters

Experiment ID	1	2	3	4	5	6	7	8	9	10	11
Accuracy	68.9	6.1	70.1	69.5	70.3	70.1	70.2	67.8	68.8	68.7	71.7

Table 3. Hyperparameter Tuning Results

Resizing Method #	1	2	3
Accuracy	71.7	72.3	72.5

Table 4. Resizing Methods Results

Data Augmentation Method	None	Color Jitter	Gaussian Blur	Random Affine
Accuracy	72.5	72.0	72.4	64.1

Table 5. Data Augmentation Results

Classifier	Case-sensitive	Case-insensitive
Accuracy	72.5	76.7

Table 6. Case-sensitive/insensitive Classification Results

Trail	1	2	3	4	5	6	7	8	9
Learning Rate	1e-5	5e-4	1e-5	1e-4	1e-4	5e-5	5e-5	5e-5	5e-5
Weight Decay	1e-3	1e-3	1e-3	1e-2	1e-3	1e-3	1e-3	1e-3	1e-3
Epoch	20	20	20	20	20	30	20	40	20
Adaptive Pooling Output	20×20	20×20	20×20	20×20	20×20	20×20	20×20	20×20	17×17
Adaptive batches	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Dropout	0.2	0.2	0.2	0.2	0.2	0.2	0.5	0.5	0.5
Padding	$\lvert kernel\rvert/2$	$\lvert kernel\rvert/2$	1	$\lvert kernel\rvert/2$	0	0	0	0	0
Accuracy	63.3%	6.1%	69.4%	65.3%	71.1%	71.2%	72.2%	72.6%	72.3%

Table 7. Adaptive Pooling and Variable Sized Batching Results

not enough kernels to pool from the last convolution layer, the adaptive max pooling may not be as effective as regular max pooling since it is not as selective. It is also plausible that this is the best a non-pretrained model could do due to the relatively small training size, especially when there are 77 unique characters, including non-English alphabets. It is also plausible that, due to the simplicity of characters, the large kernel in the first convolution layer is capable of encoding the spatial information, and the distortion caused by change in aspect ratio would not significantly hamper the classification result.

When the batch size is very small, the model is prone to overfitting easily. We have observed that, at a relatively high learning rate of 5e-4, the model stops learning and is stuck at a 6.1% with all later batches coming back with roughly the same accuracy. This is not unexpecting and we believe the root cause is that the model reaches a local minima too fast and bounces back and forth. We have used several regularization methods to prevent overfitting, including weight decay and dropout layers. Even though weight decay does not contribute to the model in our first approach we discussed above, we believe that weight decay is weakly effective for non-batched and variable sized batch approach. We have also observed a dropout layer of 0.5 probability boosting the testing accuracy by around 1%. Even though it is not significant, there is a consistent advantage over multiple training.

Adaptive batching indeed brings a benefit to training speed. Given the same model structure, by allowing for adaptive batching, we have seen an improvement on average training time from 3:14 per epoch to 2:56 per epoch, which saves around 20 seconds per epoch, a 6% improvement. Please note that, for an adaptive pooling of 20×20, the model for variably sized batch approach is slightly larger than that of non-batched approach since variable size batch outputs 102400 channels to the linear layer while non-adaptive pooling yields 73894 channels. We have also tested with a reduced output size for the adaptive pooling layer, which now only has dimensions of 17×17, or exactly 73894 channels after flattening. The improvement now is very significant, in which the training time is further reduced to 2:18 per epoch on average, a 28.8% improvement on training time. Owing to time constraints, we did not conduct thorough experiments on this smaller model, but we do not expect significant difference compared to those with 20×20 output size from adaptive pooling. We expected the variable sized batches to bring a more significant speedup, but it is very likely due to the overhead we added to the get_item method in the data loader, as it requires python query to dictionary and read images separately. However, we acknowledge that this result is obtained on two separate Google Colab instances at the same time, and thus direct comparison may not reflect the actual performance difference. Unfortunately, we do not have access to dedicated high performance GPUs to conduct a horizontal comparison. Throughout several attempts, we do see the variable batching completing faster than the non-batched model.

3.4. Future Prospective

We have observed an improvement from variable sized batching. However, since our customized dataloader is written in python and might not be best optimized, we might observe a further speedup by rewriting the dataloader in more efficient torch operation and even C level algorithm so that the overhead for retrieving images is minimized. Our model could be further improved if we train it with more training data so that our model is more robust.

4. Conclusions

In conclusion, we have experimented with various hyperparameters and data preprocessing methods to optimize the performance of our model. We found that a learning rate of 5e-4 and a batch size of 64 are optimal for our model. Additionally, adding a dropout layer with a probability of 0.5 greatly improved the model's performance, while weight decay regularization had no significant impact. We also tested different data preprocessing methods and found that stretching images to a square before resizing performs slightly better than padding or re-encoding into JPEG. Interestingly, we did not observe any significant improvement with adaptive pooling. We hypothesize that this may be due to the output size being smaller than that of our original model and the limited number of kernels available for pooling. Furthermore, we observed that a small batch size leads to overfitting with an inappropriate learning rate and that variable sized batching can improve training speed. While the variable sized batches did not bring as significant a speedup as we expected, we acknowledge that this may be due to overhead in the data loader method. Overall, our experiments demonstrate the importance of careful selection of hyperparameters and data preprocessing techniques to achieve optimal performance in machine learning models. Further training and optimization could bring even better accuracy and training speed.

5. Work Division and Code

Work distribution among team members has been provided in Table 8. Code is available on Gatech Github: `https://github.gatech.edu/bsong74/7643FinalProject`.

References

[1] Y. Zhu, J. Sun, and S. Naoi, "Recognizing natural scene characters by convolutional neural network and bimodal image enhancement," *CBDAR'11: Proceedings of the 4th international conference on Camera-Based Document Analysis and Recognition*, pp. 69–82, 2011. 1

[2] P. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pp. 958–963, 2003. 1

[3] Z. Saidane and C. Garcia, "Automatic scene text recognition using a convolutional neural network," pp. 958–963, 2007. 1

[4] X.-C. Yin, X. Yin, K. Huang, and H.-W. Hao, "Robust text detection in natural scene images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 5, pp. 970–983, 2014. 1

[5] R. Atienza, "Data augmentation for scene text recognition," 2021. 1

Student Name	Contributed Aspects	Details
Chaewon Park	Data preprocessing, Report write-up	Implemented two resizing methods for preprocessing dataset and wrote section 1 and 3 of the report.
Borun Song	Model finetuning, Experimenting, Report write-up	Drafted the CNN network structure, implemented adaptive pooling layer, implemented variable sized batching, and fine-tuned the hyperparameter for adaptive pooling model.
Weilong Shen	Model finetuning, Experimenting	Implemented the fixed-size CNN model, fine tuned it, and experimented with it. Analyzed their performance in the report.
Yueqiao Chen	Approaches, Layers, and Dataset with write-up	Implemented and designed CNN model with teammates and wrote section 1 and 2 of the report.

Table 8. Contributions of team members

[6] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *Trans on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, 2016. 1, 2

[7] C.-Y. Lee and S. Osindero, "Recursive recurrent nets with attention modeling for ocr in the wild," *The Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2231–2239, 2016. 1, 2

[8] J. Wang and X. Hu, "Gated recurrent convolution neural network for ocr," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 334–343, Curran Associates Inc., 2017. 1, 2

[9] F. Borisyuk, A. Gordo, and V. Sivakumar, "Rosetta: Large scale system for text detection and recognition in images," *CoRR*, vol. abs/1910.05085, 2019. 1, 2

[10] B. Shi, X. Wang, P. Lv, C. Yao, and X. Bai, "Robust scene text recognition with automatic rectification," *CoRR*, vol. abs/1603.03915, 2016. 1, 2

[11] J. Baek, G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. J. Oh, and H. Lee, "What is wrong with scene text recognition model comparisons? dataset and model analysis," *CoRR*, vol. abs/1904.01906, 2019. 1, 2

[12] S. Lucas, "ICDAR 2003 robust reading competitions," 2011. TC11, Jun. 28, 2011. http://www.iapr-tc11.org/mediawiki/index.php/ICDAR_2003_Robust_Reading_Competitions. 2

[13] X. Liu, T. Kawanishi, X. Wu, and K. Kashino, "Scene text recognition with high performance CNN classifier and efficient word inference," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1322–1326, 2016. 2

[14] A. Ali, M. Pickering, and K. Shafi, "Urdu natural scene character recognition using convolutional neural networks," *2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)*, pp. 29–34, 2018. 2, 4

[15] M. Hashemi, "Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation," *Journal of Big Data*, 2019. 2