Department of Computing Science **University of Alberta**

Winter 2021 CMPUT466 Final Report

List of group members (in random order)

- **Yuxi Chen**
- **Zijie Tan**
- **Lijiangnan Tian**
- **Ze Hui Peng**

Introduction & Related Work

Introduction/Motivation

Nowadays, CAPTCHA (the acronym for “Completely Automated Public Turing test to tell Computers and Humans Apart”) has been a popular method to distinguish robots from human users and has protected our internet security for a long time. From website login to transaction confirmation, CAPTCHA is always around us in our daily online life. Typically, CAPTCHAs can be divided into two categories: image-based CAPTCHA and text-based CAPTCHA. In the past few years, CAPTCHA also had a significant change in its form, from the combination of numbers and letters (i.e. alphanumeric characters) to selecting the correct images among several images. But as the CAPTCHA develops, the corresponding recognition technology is also evolving and improving constantly. Under this circumstance, we want to focus on the text-based CAPTCHA and implement some recognition models, from traditional algorithms to convolutional neural network models. We will make a comparison and analyze the detailed reasons for the different results. Meanwhile, we also plan to explore the impact of the training data sizes and features on our models.

Related Work

Traditionally, the text-based CAPTCHA recognition problems can be addressed by the OCR (optical character recognition) technology. For image-text recognition, the traditional method is to first make an object detection for the image, locate every single character, and then divide the image into single characters to recognize. In 2008, Yanping Lv and his team used this method to recognize the Microsoft CAPTCHA and finally achieved 60% accuracy [1]. As machine learning and convolutional neural network fields continue to evolve, better models can be trained to reach higher accuracy. In 2016, Yan and his team used a convolutional neural network model to achieve about 90% accuracy on the Simplified Chinese CAPTCHA dataset [2]. Later, the addition of the RNN (recurrent neural network) and LSTM (Long Short-Term Memory) made the text-based CAPTCHA obtain high accuracy while significantly reducing the model size, which was a huge milestone. In 2017, Baoguang Shi and his team added the recurrent layers into the convolutional neural network and named it the Convolutional Recurrent Neural Network (CRNN). This model had a size of only 8.3M but still achieved about 97% accuracy on different datasets.[3]

Those researches aim to improve the quality and efficiency of the text recognition models, but they only perform simple analyses of different models and focus on their models. Consequently, we intend to compare these models to analyze their differences and explain why they lead to the deficiencies of these previous analyses.

Data

Our data is currently comprised of ten thousand CAPTCHA images of size 200 pixels by 50 pixels, each of which has a dimmed background and in the centre contains a 6-character-long string — a combination of warped digits and deformed letters with some squiggle curves drawn across and alongside. The datasets are either from Kaggle or generated from `Google Kaptcha` by one of our group members, Ethan Yuxi Chen. Every image file is designated in the format `{content}_{index}.jpg` where `{content}` serves as the corresponding label and contains the actual string in the image which our models need to recognise. For this CAPTCHA recognition problem, which is a supervised classification problem, the labels of our data are multiclass, there are a total of 36 classes, which contains 10 digits and 26 case-insensitive letters. The following figure (**Figure 1**) contains one of the CAPTCHA images in the dataset.

A sample CAPTCHA image with content **882m62**

Figure 1: Sample CAPTCHA image from CAPTCHA-6-digits

While considering the accuracy of our models, our models may not predict all of the six characters in an image correctly, i.e., the prediction result may be partially correct with some characters accurately predicted and others wrongly. However, we notice that in the real world CAPTCHA is an all-or-nothing situation, one does not pass the test unless one gets all the characters correct. Therefore we use the rudimentary string equality instead of comparing the prediction result to the corresponding label character-wisely.

The interpretability of our models is not crucial since we do not need to know how our models come up with the predictions.

Analysis/Methodology

Most of the currently available text-based CAPTCHA recognition models can be classified into two categories: segmentation-based algorithms and segmentation-free algorithms, according to Thohbani et al.'s research [4]. We are interested in the differences between the effects of these models and algorithms, as well as the reasons behind these differences.

We applied some most commonly-used methods of both categories and their combinations to train text-based CAPTCHA recognition models, compare their performances, and investigate the factors that may affect their performances.

The algorithms we worked on are listed below:

1. Segmentation-free Algorithms

a) Convolutional neural network (CNN)

The first method we tried is the convolutional neural network. Based on the CNN model of Zahra Noury and Mahdi Rezaei [7], we built a similar model. The structure of the convolutional neural network model is depicted in **Figure 2**. The network starts with a convolutional layer with three input channels, the batch normalization layer, the ReLU (rectified linear unit) activation function, and 5×5 kernels, followed by a 2×2 max-pooling layer. This model has four parts similar to this one, and they have 32, 48, 64, 64 neurons respectively. Then the data will be flattened and passed through two fully connected layers. Finally, this model returns a sequence of length 216. This sequence will be divided into 6 equal segments each of which contains 36 values, corresponding to the aforementioned 36 classes. The model can select the index of the maximum value in each segment of the sequence and generate the predicted CAPTCHA by finding the corresponding character in the data table with the selected index.

In our experiments, we use the same method to measure performance. The details related to performance measurement are mentioned later in this section. For this model, we currently have five hyperparameters to tune, including learning rate, number of iterations, activation function, the number of the hidden layers, and the dropout function.

The learning rate controls the magnitude of the updates on the model. The number of iterations controls the progress of the model training. The activation function can provide nonlinear functions for the model and improve the performance of the model. But different activation functions have different characteristics. The number of hidden layers affects network performance and the computational cost of training this network. The dropout function prevents the model from overfitting.

But we need to pay attention that these hyperparameters are not “the bigger the better”. They usually incur some problems when they get large. Thus, we need to find suitable values for them. In this project, we tried three different learning rates, including 0.1, 0.01 and 0.001, and four different training epochs, including 10, 15, 20, and 25. Finally, we chose 0.001 as the learning rate and 15 as the number of iterations, because the model gets the best performance with these hyperparameters in the experiment.

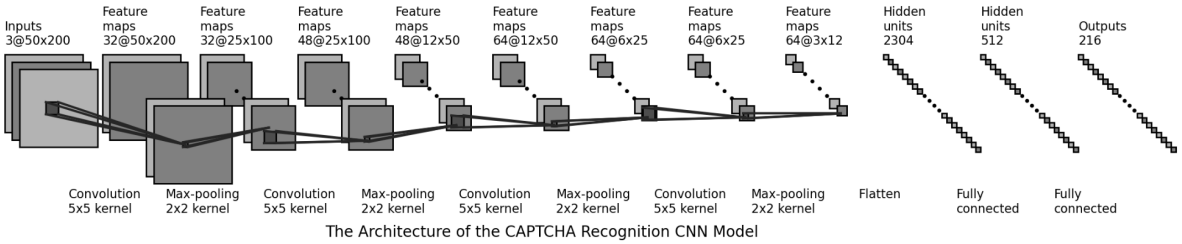

Figure 2: **The Architecture of CNN Model**: the convolutional layers and the pooling layers are arranged alternately, and the fully connected layers are added at the end. The model will output a sequence of length 216.

b) Recurrent neural network (RNN)

The second method that we are interested in is the recurrent neural network. **Fig. 3** shows the structure of the most basic neural cell in the RNN and the unfolded version. It uses the following equations to compute the output: $o_t = Wh_t$, $h_t = f(Ux_t + Vh_{t-1} + b_t)$, where f is the activation function, U, V, W are the 3 weight matrices, and b is the bias term. This model takes input sequentially and computes output recurrently, which provides it with the potential to solve variable-length sequence prediction problems. In our case, we hope that this architecture will allow us to extend the model to process and recognize variable-length CAPTCHA.

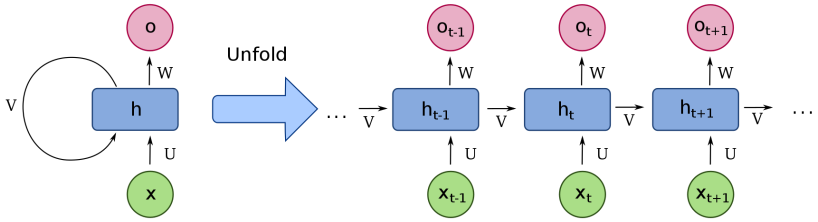

Figure 3: The structure of standard RNN cell and its data flow [6]

However, in practice, we found it quite difficult for the standard recurrent neural networks to efficiently complete this task. In our experiments, the naïve RNN barely converged, and its accuracy is extremely low (~0.02%). So we decided to resort to some other variants of the RNN network.

The text-based CAPTCHA recognition involves taking in an image, which is represented by tensor, as input and producing a string as output. So this task is a typical sequence-to-sequence task (seq2seq). Therefore, we decided to use the encoder-decoder RNN architecture, which has been proven an effective approach to such a seq2seq prediction problem [8].

The encoder-decoder RNN architecture consists of two RNN cores: one for encoding the input into a fixed-length vector, the other for mapping the vector representation back to a variable-length target sequence. We used `PyTorch` to implement an encoder-decoder RNN model that contains an input dense layer, an encoder layer, a decoder layer, and an output dense layer as one of our preliminary models. **Fig. 4** and **Fig. 5** demonstrate the architecture and the data flow, respectively.

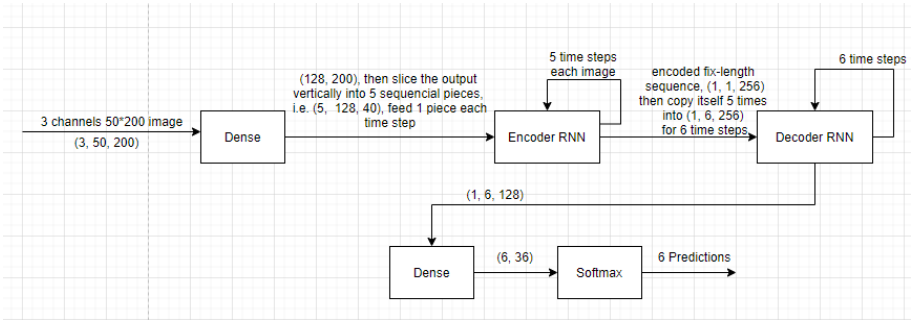

Figure 4: The architecture of our encoder-decoder RNN model

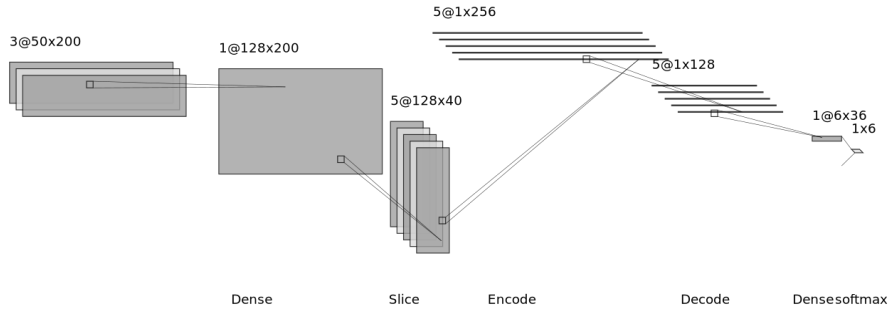

Figure 5: The data flow of our encoder-decoder RNN model, note that in the `Slice` step, it slices the image horizontally into 5 equal pieces so that they can be fed into the model as sequential inputs for 5 time-steps

2. Segmentation-based Algorithms

The communal step before applying the algorithms listed below is to apply image segmentation to separate every digit or letter in the image. If you check the sample image illustrated in **Figure 1**, you will notice that the image has noise lines which make it intricate to identify the digits and letters therein. The first step we planned is to apply median filters to remove some of the line noises, and after the line noises are removed, apply image segmentation algorithms to separate each one of the individual characters.

c) Support-vector machine (SVM)

The third method we explored is the support-vector machine model. As a segmentation-based algorithm, the first phase is to preprocess the image which helps to segment the CAPTCHA image into 6 smaller images each of which ideally contains the entirety of a single alphanumeric character. However, segmentation is also the most difficult and most important step among all the steps in the preprocessing phase, and it significantly impacts the performance of the SVM in the sequential recognition step. Hence, the preprocessing procedure is indeed much more complicated than the one in a segmentation-free algorithm. The current preprocessing procedure consists of seven steps: erosion, grayscale, binarisation, morphological transformation, shear transformation, horizontal stretch, and lastly, segmentation. All those steps before segmentation aim to remove noises, take out the curvilinear patterns in the background or foreground, and straighten the characters and increase the spacing between adjacent characters so that it would be easier to separate them in the segmentation step. The current segmentation step is rudimentary as we just separated the preprocessed CAPTCHA image into even pieces, which works not well when there are characters in the picture that are particularly wide such as `W` and `M`. A successful segmentation is illustrated in the SVM part of the following Results section. The SVM model we used is C-Support Vector Classification from `scikit-learn`. The SVM contains two hyperparameters: C, called the regularization parameter, which helps us tune the importance of misclassified data points (or equivalently, controls the strength of the regularization); and kernel types, which decides the kind of kernel inside the SVM.

d) k-nearest neighbours algorithms (KNN)

The fourth method that we explored is the k-nearest neighbours algorithm for classification. After the image has been segmented into individual characters, each digit or letter in each image from the training dataset will be classified as one of the 36 classes (10 digits and 26 letters, case-insensitive) that corresponds to its true label. Then for each digit/letter in each image from the test dataset, we calculate the Euclidean distances with all the data from the training dataset. After all the distances are calculated, data with the lowest k distances will be used for the classification, each

digit/letter will be classified as the class that has the largest occurrence among those k nearest neighbours. Here k is a hyperparameter that represents the number of closest distance data, we tried out many different k values to check out which k value will yield the best performance.

Measuring the Performance and Generalizability:

In general, this is a classification problem, and therefore it is natural to use classification accuracy/error to measure the performances of the models we train.

We focused on fixed-length 6-alphanumeric-characters CAPTCHA recognition. Another temporary assumption to simplify the preliminary models is that our CAPTCHAs contain only lowercase letters and numbers (so 36 possible characters in total).

We are using a database that contains 10,000 labelled CAPTCHA images at this point. For the consistency of model comparison, we decide to use the same data segmentation for all 4 models, that is, use 6000 images for training, 2000 images for validation, and the remaining 2000 images for testing (that is, we use a split ratio of 6:2:2 instead of 7:1:2).

Results

1. CNN

As shown in **Fig. 6**, the performance of the CNN model has been greatly improved after only a few training epochs, and it can reach an accuracy of about 97% after 15 training epochs. Although our measurement standard is to judge whether the model prediction and the actual characters are all the same, we still want to explore the prediction accuracy of a single character. According to the confusion matrix in **Fig. 7**, we can find that the data set is not balanced, and some values on the diagonal are 0, but according to the colour and other values, the confusion matrix only has a deep blue colour and a high value on the diagonal, while the other positions are almost 0, which means that the CNN model is very accurate for all characters. There is no low performance of some character samples. As for the case where the input of certain characters of the confusion matrix is 0, we will discuss further later.

For this result, we believe that this is related to the convolution operation inside the CNN model. Because the convolution operation can reduce large data volume to small data volume, while it can retain image features.

In this project, the convolution operation can extract the needed features in the data and reduce the impact of interference on character recognition, while still preserving the location information. At the same time, the normalization and pooling operations will

compress the information. Compared with characters, noise occupies a small proportion in the image, and the model can obtain image features more easily after compression.

However, due to some characteristics of convolutional neural networks, such as convolution operation, it is difficult to explain in detail.

Based on the above point of view, we believe that the CNN model should have a strong generalization ability. The same model may be suitable for CAPTCHA with different types of noise added. In order to verify the accuracy of this conjecture, we generated multiple data sets with different noises for this purpose, and trained and tested the CNN model. According to **Fig. 8(a)**, we found that for the CNN model, different noises will not have too much influence on the model. The performance of the model has always been maintained at around 95%, with only slight fluctuations.

2. RNN

Compared to our CNN model, the performance of our RNN model is still much lower after a relatively large amount of epochs, although we have applied many improvements to it, such as increasing the number of hidden layers, adding a dense layer, etc. It takes much more epochs to converge (nearly 60 epochs), and its final accuracy is much lower (73.1% against CNN's 96.75%). One of the possible reasons for this is that, unlike the CNN model, when the vector representation of the images is fed into the model sequentially, the spatial properties of the images are not reserved in the computation.

Another hypothesis we made is that when the RNN model takes the previous output as one of the inputs at every time step, the past information becomes more heavily involved in predicting the next character of the CAPTCHA string. This may have negative effects on the prediction process since there is no strong connection between any two consecutive characters in a CAPTCHA string. We still need some further experiments to prove or disprove this hypothesis. If the hypothesis is correct, adding a scalar to the previous output to lower its influence may improve our RNN model further.

There is one interesting property we discovered during the training process, that is if we use character-wise accuracy instead of CAPTCHA-wise accuracy, the accuracy reaches approximately 92%. In other words, although the model failed to predict most of the CAPTCHAs 100% correctly, it did succeed to correctly predict the majority of the characters, and in most of the failure cases, it got merely a single character wrong. According to the confusion matrix in **Fig. 7,** the RNN model has a weak ability to recognize the characters "A", "8", and "E". At the same time, the model will also cause a certain degree of confusion for the characters "M" and "N", or the characters "W" and "X".

Due to the poor interpretability of the RNN model, the reason behind this is still undercover.

We also used different datasets to train and test the RNN model. We found that the RNN model is less resistant to different noises, and the accuracy rate drops by 50% in some data sets. Therefore, for different noises, we may need different models to maintain a good performance.

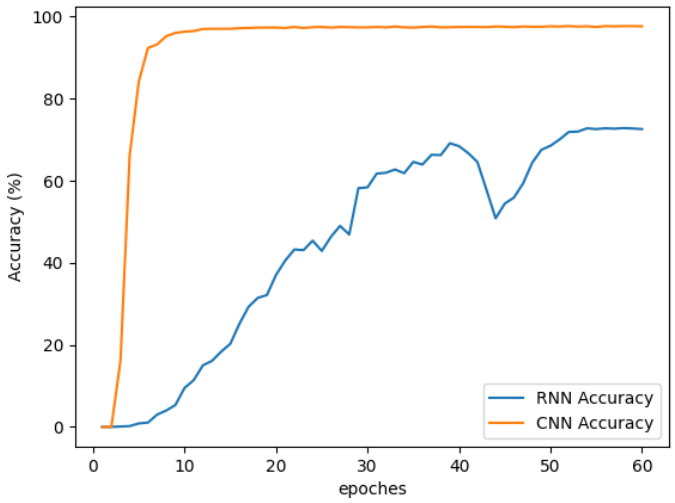

Figure 6: Accuracy chart for CNN and RNN Models

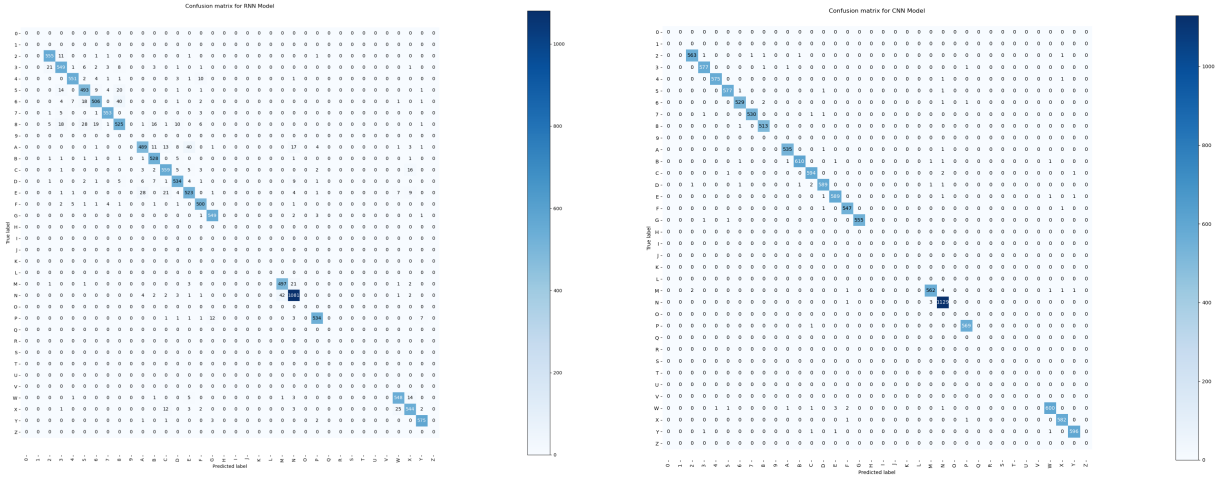

Figure 7: Confusion matrices for CNN (left) and RNN (right)

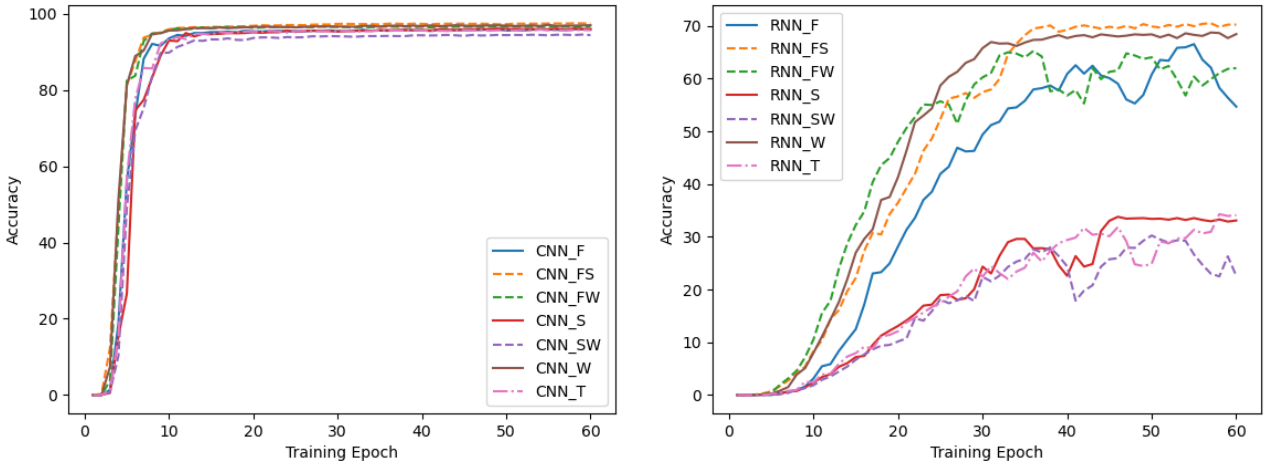

Figure 8: **(a)** For a dataset with different noises, the accuracy of the CNN model. **(b)** For a dataset with different noises, the accuracy of the RNN model. The "CNN" and "RNN" before the underscore in the label represent the type of model, the characters after the underscore in the label represent the type of noise, where "F" stands for fisheye noise, "S" stands for shadow noise, "W" stands for water ripple noise, and "T" stands for having all these three kinds of noise. Multiple characters represent the combination of these noises, such as "FS" for fisheye and shadow noise combined.

3.SVM

The SVM model we are using can predict most characters in a CAPTCHA image (segmented by our rudimentary segmentation method) correctly most of the time.

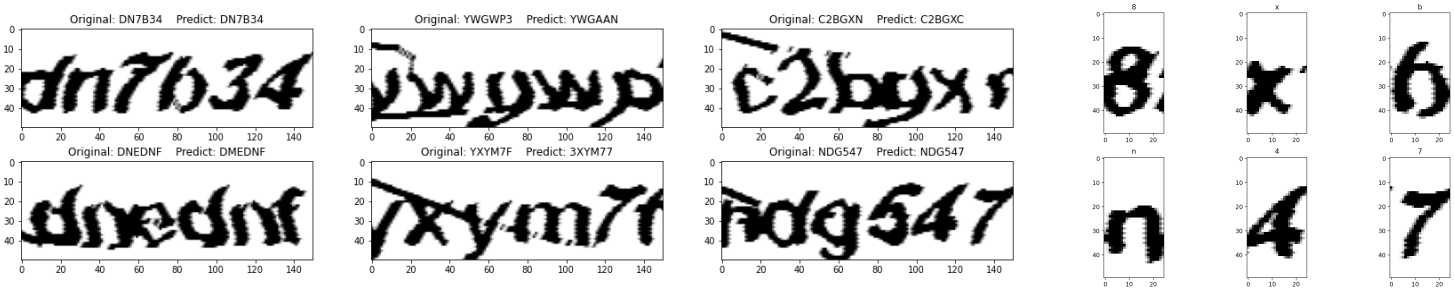

Figure 9: Samples of CAPTCHA with corresponding labels and predicted ones (left) and an example of image segmentation for CAPTCHA **8xbn47** (right)

The current rudimentary segmentation method works well when the alphanumeric characters in the CAPTCHA image have narrow and similar widths. After usual image preprocessing, the rudimentary image segmentation technique simply straightens the image and divides it into six parts evenly and ditches the remaining white background.

We explored a couple of different regularization parameters each with three kernel types: linear kernel, polynomial kernel and radial basis function (RBF) kernel. From the results shown in the following chart, we found out that selecting the suitable kernel type is of utmost importance. The kernel type that works the best is the RBF kernel. The regularization parameter does not matter much when it gets bigger than 1. The highest accuracy our SVM model can achieve is around 82%, equipped with the RBF kernel. The accuracy for recognising individual characters using the RBF kernel is 94.05%. The kernel matters so much because it is the kernel that maps the data into higher dimensions so that the transformed data becomes linearly separable. Hence, if the kind of kernel one chooses cannot eventually make the data linearly separable, the accuracy of the SVM will be very low, as the case for the linear kernel in the following chart. In addition, noises can make the SVM perform very poorly because our rudimentary image segmentation algorithm cannot deal with all kinds of image transformations that come along with noises, i.e., it can no longer separate the image into individual characters. For the confusion matrix in **Figure 10**, the SVM model has the best prediction (most true positives) on digit '4' (ignoring the letter 'n', see **Analysis of Confusion Matrix Abnormality** section below for reasoning), likely the reason is that '4' looks very different from the rest of the characters. On the other hand, the model makes a lot of prediction errors between the letter 'm' and letter 'n', which is because our basic image segmentation algorithm tends to divide the letter 'm' in half and will make the segmented images look like an 'n'.

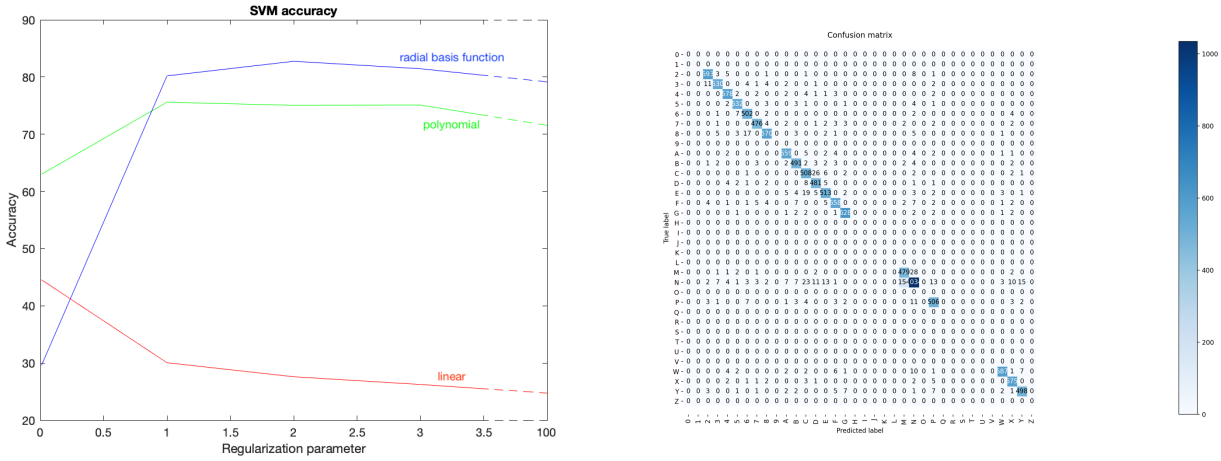

Figure 10: Support vector machine model accuracy chart (left) and confusion matrix for support vector machine with RBF kernel (right)

4.KNN

The highest accuracy KNN model achieved is ~72% with k = 20 neighbours, k is a hyperparameter that we tried with different values. **Figure 11** shows the performance chart of KNN from k = 1 through k = 30 neighbours. Although not included in the figure,

we also tried out $k$values from 50 to 250 in an increment of 25, but with higher k values the model starts to perform worse. The optimal k value is affected by the size of the training data, for example, if the training data only has 20 samples that correspond with the letter `m`, setting k to a high value will likely cause the model to include neighbours from another class, and if too many incorrect neighbours are included then the model will make incorrect predictions.

Figure 11 also shows that the model is very susceptible to noise, the noise in our dataset includes line noise, shadow, fisheyes, and water ripple. The performance with reduced noises is between 65%–72%, while the performance with noises is between 22%–34%. The reason for this performance gap is that noises created redundant pixels that result in segmented characters with the same class got placed far away from each other in the 2-dimensional place, in which a character's position in the space is determined solely by image pixels. Therefore when considering nearest neighbours it is less likely that neighbours with the same class are nearby, which means it is more likely to include neighbours of different classes for prediction and make an incorrect prediction. For the confusion matrix in **Figure 11**, the KNN model has the best prediction (most true positives) on digit '4' (ignoring letter 'n', see **Analysis of Confusion Matrix Abnormality** section below for reasoning), which is exactly the same as the SVM model. On the other hand, the model makes a lot of prediction errors between the letter 'm' and the letter 'n', as the two characters look alike. Also described in the SVM section, our segmentation algorithm sometimes will divide 'm' in half which makes it look like an 'n'.

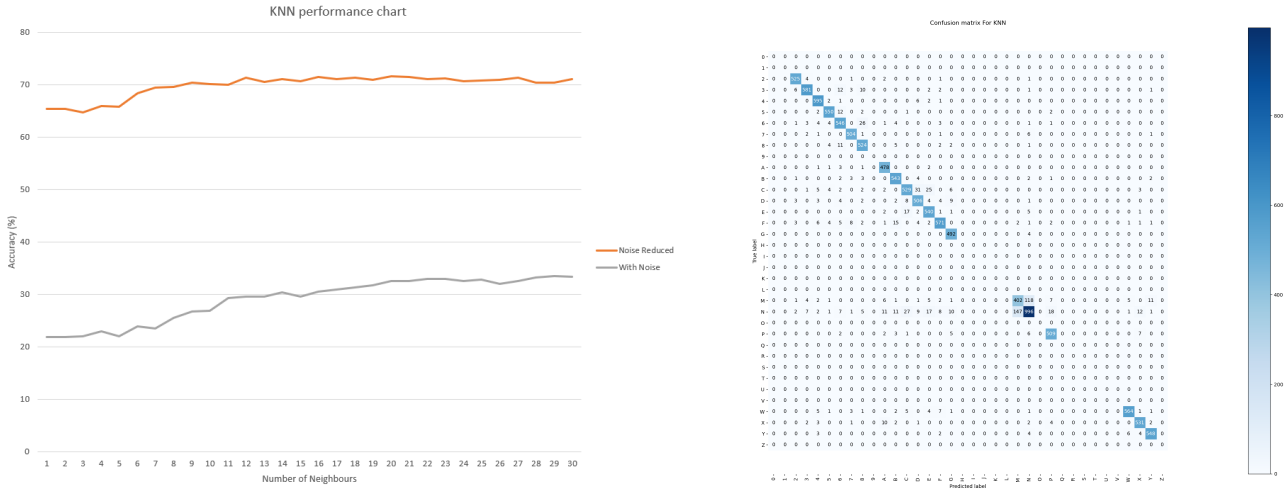

Figure 11: k-nearest neighbour model performance chart (left) and confusion matrix for KNN with k = 20 neighbours (right)

Analysis of Confusion Matrix Abnormality

According to the confusion matrices, our dataset is unbalanced, and some labels are missing, e.g., “0”, “1”, “i”, “j”. Meanwhile, compared with other characters, the number of characters “n” is about twice the number of other characters. For this phenomenon, we checked the source code of Google Kaptcha and found that in the process of CAPTCHA generation, the program has ignored some characters that can cause ambiguity. Furthermore, all CAPTCHA pictures are randomly generated based on the string "abcde2345678gfynmnpwx". (**Figure 12**)

```
public char[] getTextProducerCharString() {
    String paramName = "kaptcha.textproducer.char.string";
    String paramValue = this.properties.getProperty(paramName);
    return this.helper.getChars(paramName, paramValue, "abcde2345678gfynmnpwx".toCharArray());
}
```

Figure 12: The part of source code in Google Kaptcha library

What's interesting is that we found that there are two characters "n" in this string, so this is the reason why the number of characters "n" is twice the others. Thus, this dataset actually only contains 20 labels, which is different from the labels we set to the models. However, based on the analysis of the actual situation, it is impossible for us to know the source code of the CAPTCHA generator in real life, so we decided to keep our initial settings, that is to say, keep 36 labels to configure models.

Discussion, Conclusion, Future Work

High-level Summary & Takeaways

As a high-level summary, the algorithms/models we choose for this project are very common machine learning algorithms and most of the algorithms were discussed in class this semester, and the dataset we used are common 6-digit CAPTCHA images that are still being used today. The CNN algorithm had the best performance with ~97% accuracy for our dataset. The other three algorithms have a maximum accuracy between 70%–80%, which is not bad, but there is still room for improvement, such improvements will be discussed later in the Future Work section below.

Ethical Concerns

CAPTCHA is an interesting topic to discuss potential ethical concerns for machine learning, because CAPTCHA is used as a security tool for websites to distinguish robots from humans. Therefore if our model works well, it will defeat the main purpose of CAPTCHA as artificial intelligence has the ability to bypass this stage of security check. If the model does not work well then the web developers will get the benefit, as they need to worry less about non-human users trying to access their

website. The model could be misused by hackers that try to overload some website, which they can use the model and some kind of automated tool to register and login large amounts of accounts. Our data only consist of a fixed length of 6 digits and/or letters, in the real world it is more common to see CAPTCHA with different lengths, also it is now more common for websites to use image-based CAPTCHA instead of text-based because it is much harder to decode image-based CAPTCHAs. Another consideration is the noises in CAPTCHA images, in our dataset we only had four types of noises (line, fisheye, shadow, and water ripple), but in reality there are other noises to consider which can potentially make our models perform worse.

Future Work

If we had another semester, we plan to explore which noise each model is more sensitive to in order to obtain a certain level of interpretation. We also want to design some experiments to discover the reasons behind the aforementioned results and verify our hypotheses, for example, design experiments to verify the effect of the spatial properties of the images on the performance of our models. We also want to expand our model to work with variable-length CAPTCHA datasets, as our current model will only work for the length of 6. If people continue to work in this area for the next few years, one thing they can try is to develop models that can decode different types of CAPTCHAs, for example, image-based CAPTCHAs, 3-dimensional CAPTCHAs, mathematical/calculus CAPTCHAs, etc. On the other hand, people can also develop new libraries that can generate new types of CAPTCHAs that are fundamentally harder for artificial intelligence to decode, but not too arcane to solve for human users.

Task Distribution

Here is a screenshot of our work plan from Google Sheets:

Work Name	Estimated Hours	Expected Completion Date	Assignee	Estimated Progress	Complete	Additional Notes
Project Proposal	10	February 2, 2021	All	100%	☑	* The subsections were deleted to reduce length
Create Github Repository	0.5	February 3, 2021	Ze Hui Peng	100%	☑	* Should disable user directly pushing to main/master * Every Pull Request(PR) should have at least one approval before merging
Oraganize and/or Generate Training and Test Datasets	5	February 12, 2021	Yuxi Chen	100%	☑	* There are some datasets available on Kaggle * Yuxi also found out how to generate CAPTCHA images
Project Midterm Report	10	March 11, 2021	All	100%	☑	* The subsections were deleted to reduce length
Implement&Train model using CNN	25	March 19, 2021	Yuxi Chen	100%	☑	
Implement&Train model using RNN	25	March 19, 2021	Zijie Tan	100%	☑	
Implementing Image Segmentation	10	March 19, 2021	Lijiangnan Tian	100%	☑	* include removing image noise and character segmentation
Implement and train model using a combination of IS and KNN	10	March 26, 2021	Ze Hui Peng	100%	☑	
Implement and train model using a combination of IS and SVM	10	March 26, 2021	Lijiangnan Tian	100%	☑	
Test all available models	10	April 1, 2021	All	100%	☑	* Report the running time and accurency for each model using the same set of test data * test out different hyperparameters
Project Demo Video	5	April 6, 2021	All	100%	☑	* Slides preparation: Everyone * Voice Over: Ze Hui Peng
Project Final Report	10	April 16, 2021	All	100%	☑	* No changes in Introduction, Related Work, and Data section
↳ Analysis & Methodology Section	4	April 16, 2021	All	100%	☑	
↳ Results Section	4	April 16, 2021	All	100%	☑	
↳ Conclusion Section	1	April 16, 2021	All	100%	☑	
↳ Work Ditribution Section	0.5	April 16, 2021	All	100%	☑	
↳ Code Section	0.5	April 16, 2021	All	100%	☑	
Glossary:						
Covolutional Neural Network	CNN					
Recurrent Neural Network	RNN					
Image Segmentation	IS					
K-Nearest Neighbour	KNN					
Support-Vector Machine	SVM					

Figure 13: Screenshot of our work distribution

If the image is too small or you are having trouble viewing the image, you can view our up-to-date version of our project work plan from our Project Work Plan Google Sheet.

Code

GitHub repo containing all our code can be found here: Link
The dataset we used can be found here: CAPTCHA-6-digits

The Colaboratory notebook `ML_CAPTCHA_Images_Decoder.ipynb` in the repository contains everything we used for this project organized into different sections. Please consider using this notebook to run our code.

See the `README` file in our GitHub repository for more information if needed.

References

1. A low-cost attack on a Microsoft captcha. 2008. Accessed February 2, 2021. https://search-ebscohost-com.login.ezproxy.library.ualberta.ca/login.aspx?direct=true&db=edsoai&AN=edsoai.on1098300311&site=eds-live&scope=site

2. Lv Y, Cai F, Lin D, Cao D. Chinese character CAPTCHA recognition based on convolution neural network. 2016 IEEE Congress on Evolutionary Computation (CEC), Evolutionary Computation (CEC), 2016 IEEE Congress on. July 2016:4854-4859. doi:10.1109/CEC.2016.7744412

3. Shi B, Bai X, Yao C. An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39(11):2298-2304. doi:10.1109/TPAMI.2016.2646371

4. Thobhani A[1], Gao M[1], Hawbani A[2], Ali STM[3], Abdussalam A[4]. CAPTCHA recognition using deep learning with attached binary images. Electronics (Switzerland). 9(9):1-19. doi:10.3390/electronics9091522

5. Character-Based Handwritten Text Transcription with Attention Networks. 2017. Accessed February 2, 2021. https://search-ebscohost-com.login.ezproxy.library.ualberta.ca/login.aspx?direct=true&db=edsoai&AN=edsoai.on1106281594&site=eds-live&scope=site

6. Recurrent Network Structure File:Recurrent_neural_network_unfold.svg, Accessed March 6, 2021.

7. Noury Z, Rezaei M. Deep-CAPTCHA: a deep learning based CAPTCHA solver for vulnerability assessment. 2020. Accessed April 16, 2021. https://search-ebscohost-com.login.ezproxy.library.ualberta.ca/login.aspx?direct=true&db=edsarx&AN=edsarx.2006.08296&site=eds-live&scope=site

8. Brownlee J., Encoder-Decoder Long Short-Term Memory Networks. https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/