Chinese Digit Classifier

Yiming Feng, Rithika Krishna Perugupalli, Brian Li, Angela Zhou, Stephanie Hsia
University of California, Davis
Emails: {ymfeng, rperugupalli, bryli, ajzhou, sehsia}@ucdavis.edu

1. Introduction and Background

With over 1 billion total speakers worldwide, Chinese is the world's most widely used language; it also contains one of the world's most complex writing systems. Utilizing a non-alphabetic writing system, Chinese is composed of individual characters called hanzi that represent specific words or concepts. Hanzi can vary from simple characters made of a handful of strokes to more complex ones made up of multiple components. An estimated 3,000 hanzi are necessary for a Chinese speaker to be considered fluent. Because Chinese has such an intricate writing system made of characters with widely varying structures, written communication in the language is susceptible to confusion or misinterpretation. This is exacerbated if the communication is done through written Chinese, which can vary greatly depending on each individual's handwriting style. As Chinese use continues to grow internationally, miscommunications arising from its writing system may have potentially severe consequences.

Our project aimed to develop a machine learning-based system for identifying handwritten Chinese numbers, a specific subset of hanzi. There are countless practical applications for our project. It could be used as a tool for Chinese learners, a medium of communication between Chinese speakers and non-Chinese speakers, a method to quickly process large amounts of handwritten documents with poor legibility, or a way for elderly/visually impaired people to read Chinese.

For this project, we used the MNIST Handwritten Chinese Numbers dataset[5], originally developed by researchers Dr. Kianoush Nazarpour and Dr. Mingyang Chen at Newcastle University. For simplicity and scalability, we focused on number recognition as a proof of concept, which later could be scaled for other Chinese hanzi.

The first stage of this project was performing exploratory data analysis on the dataset to ensure it can lead to successful model development. Our EDA involved checking the class label distribution of the dataset, ensuring equal distribution of samples for each class in the training and testing datasets, visualizing pixel intensities across images, verifying that character consistency between samples, and performing principal component analysis to visualize any potential underlying structures.

To augment the input data, each image in the dataset was duplicated with a random rotation, making the input data more varied and model training more robust by simulating real-life characters and preventing overfitting.

After EDA, we developed and deployed three separate CNN models for character recognition. First, a VGG-16 model, a type of deep convolutional neural network whose depth is good for large datasets of complex images. Second, a LeNet model, a type of simple CNN that is specifically designed for handwritten character recognition. Lastly, a ResNet model, a type of deep CNN that utilizes skip connections to prevent the vanishing gradient problem and is good for datasets with many handwriting variations. Grid search was used to optimize model hyperparameters to find the optimal configuration.

Upon holistically evaluating model performance on precision, accuracy, recall, and F1 score, the VGG16 model was found to have the best performance, followed by ResNet, then finally LeNet – with all having significantly better than chance performance in classifying Chinese characters. We also deployed a demonstrative website, in which users are able to draw a Chinese numeral and see model predictions alongside their respective probabilities.

2. Literature review

"Chinese character recognition: history, status and prospects[1]" by Dai Ruwei, Liu Chenglin, and Xiao Baihua discusses the difficulty of creating classification models for Chinese because of the complexity of its characters. The article found that performing data preprocessing techniques like noise reduction and line density smoothing combined with augmenting the dataset with deformed samples increases the effectiveness of the model. The article demonstrates several successful machine learning techniques for Chinese character recognition, including classification networks, decision trees, Bayesian networks, single vector machines, and multi layer perceptron models. Additionally, it notes that a lack of quality samples and data preprocessing results in decreased chance of success. Taking inspiration from this article's findings, our project will include data augmentation to improve model training. However, unlike this article, our project will use randomly rotated images to augment the dataset instead of deformed images, start from a large enough dataset of quality samples, and use CNN models instead.

"Drawing and Recognizing Chinese Characters with Recurrent Neural Network[8]" by Xu-Yao Zhang, Fei Yin,

Yan-Ming Zhang, Cheng-Lin Liu, and Yoshua Bengio discusses the success that past projects on Chinese character recognition have had with CNN models. However, the article instead discusses the authors' attempt to develop a recurrent neural network (RNN) for Chinese character recognition and compare its effectiveness to CNN models. Before model training, the authors applied image normalization preprocessing. Additionally, the dataset was augmented by adding a large amount of randomly generated new images. Lastly, hyperparameters such as activation function, number of layers, regularization, and dropout were manipulated to find the optimal model. The report concludes that the RNN model was successful and is a good candidate for Chinese character recognition. For our project, we also intend to add a large amount of new samples for augmentation and tune hyperparameters during model training. However, we still intend to keep the CNN model as the focus of our project because of its high performance in this field .

"Research on Artificial Intelligence Machine Learning Character Recognition Algorithm Based on Feature Fusion[2]" from Dongdong He and Yaping Zhang defines the three types of character recognition: handwritten, printed, and world scene character recognition, of which handwritten is the most mature. He and Zhang underline the importance of data augmentation to prevent overfitting, which we implemented via random rotation of the input data. Their conclusions reveal that convolutional neural network models display high levels of accuracy when trained to recognize Chinese characters. In particular, they highlight a VGG-16 based network structure and Inception-ResNet-v2 models. Based on the success shown by this study – with precisions as high as 99.932% in the test set, we concluded that our project should also explore VGG-16 and ResNet as potential candidates for Chinese character recognition models .

3. Dataset Description and Exploratory Data Analysis

Our project uses the MNIST Handwritten Chinese Characters Dataset, developed at Newcastle University by researchers Dr. Kianoush Nazarpour and Dr. Mingyang Chen. This dataset originates from a study conducted with the help of 100 Chinese nationals; each volunteer provided 10 handwritten samples for each of the 15 Chinese numbers. Thus, each volunteer contributed 150 samples, with the entire dataset being 15,000 images total.

The first stage of our project's Exploratory Data Analysis is preparing the data for use. First, the dataset is imported from Kaggle using the kagglehub library, then converted to data frame format using the Panda library for easier handling. Each image in the MNIST dataset is characterized by 5 features: suite_id, which identifies which volunteer wrote it; sample_id; code, a number between 1 and 15 that indicates which character the sample is; value, the value of the character; and character, the 64x64 pixel image of the character.

Next, a class called ChineseMNISTdataset is created to give a way to easily access images from the dataset. ChineseMNISTdataset contains three functions. The init function accepts the dataframe and stores the image samples in a directory for our use. The len function returns the number of images in the dataset. The getitem function retrieves a specific image sample from the dataset given an index.

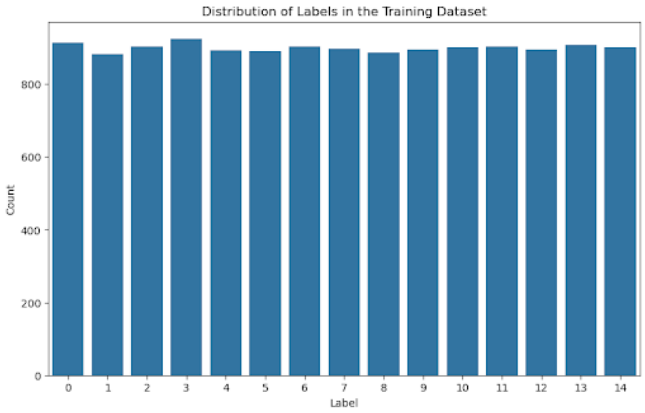

Figure 1. Class label distribution in training datasets

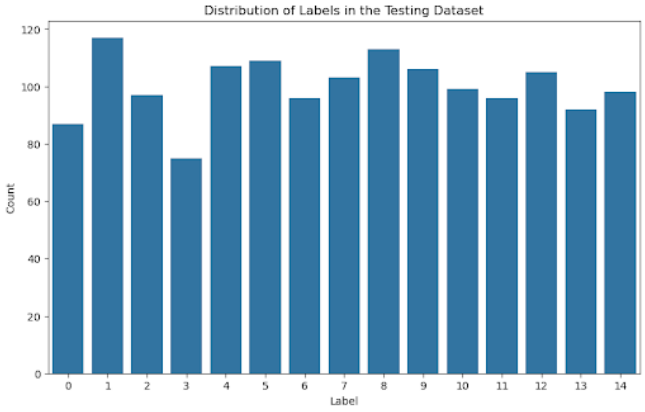

Figure 2. Class label distribution in testing datasets

The second stage involves splitting our data into training and testing datasets. To ensure that the 15 characters are equally distributed in both the training and testing datasets, we created two histograms visualizing the count for each character; these histograms confirm that no characters are over or underrepresented in one dataset over the other. As shown in Figure 1, the class label distribution in the training dataset is well balanced. Similarly, Figure 2 illustrates the distribution in the testing dataset..

Next, we visualized the pixel intensity of samples to observe the distribution of pixel values in images. For each number, we created a histogram for each number to visualize distribution of pixel intensities, see Figure 3. Pixel intensities are binned 0-7, with each bin representing a different range (8 equally-sized bins between 0-255) of pixel intensities.

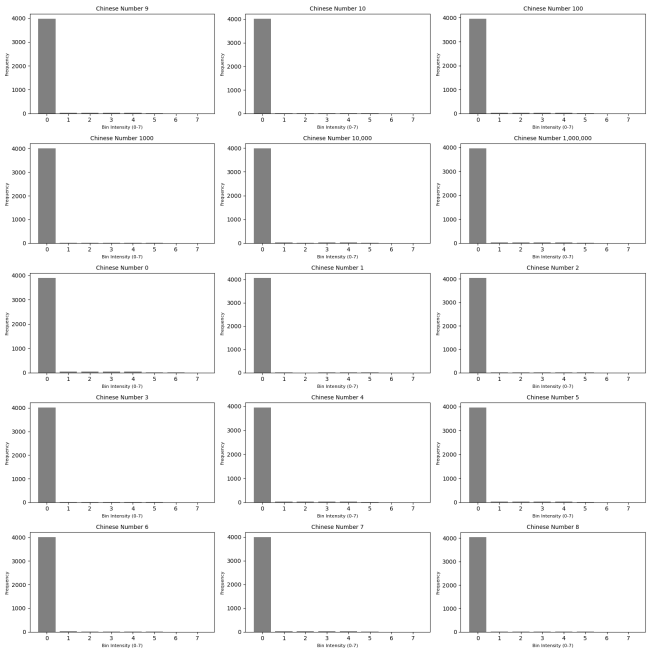

Figure 3. distribution of image pixel densities for each character

They show the distribution is heavily right-skewed, with the vast majority of pixels being white or close to white.

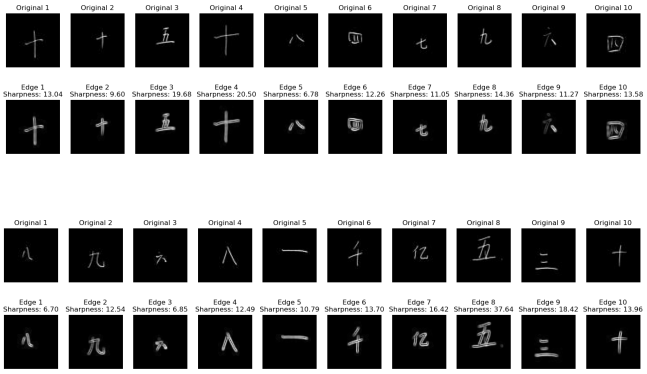

Figure 4. Comparison of original character with visualization of its edge sharpness

Next, we performed sharpness and edge detection on the data. We used Sobel edge detection filters to calculate the edge sharpness of the images. We compared the original image with the edge detection visualization and its sharpness score. Our results confirm that the images have high sharpness and no unnecessary noise is present, see Figure 4. Additionally, we plot a histogram of the edge sharpness of images from the training and testing dataset to ensure both datasets contain enough images of high sharpness, see Figure 5.

Next, we performed an intra character consistency check to see if the appearance of characters remains consistent across the data. We did this by calculating the mean and standard deviation of pixel intensities for each character. We compared the visualization of the mean to the visualization of the standard deviation for each character, see Figure 6. Because the standard deviation visualization does not differ that drastically from the mean, we concluded that characters remained consistent across samples.

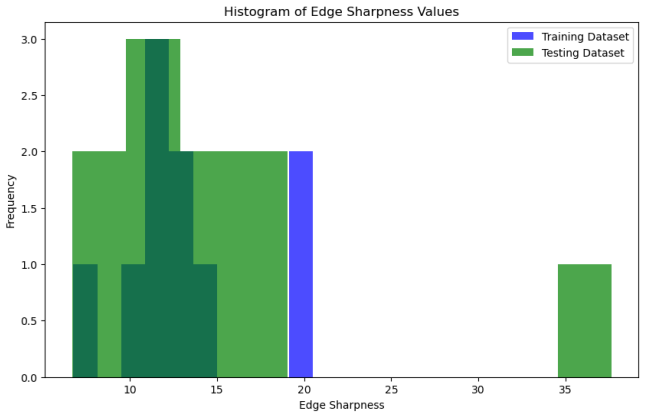

Figure 5. Histogram visualizing distribution of images' edge sharpness, separated by training and testing data

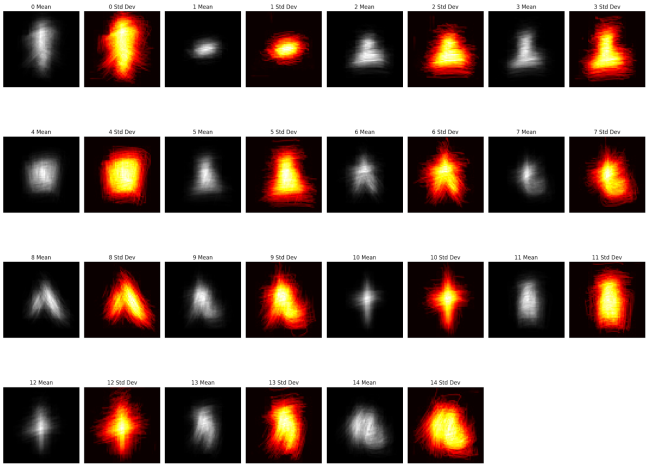

Figure 6. comparison between the visualizations of the mean pixel density and the standard deviation for each character

PCA was used to visualize the top two features in the flattened train and test datasets, i.e. the principal component with the most variance and thus the most information. After plotting the two principal components against each other, we found that there was no discernable pattern or underlying structure in the top two principal components. Attempting to visualize inherent manifolds in the data, we see the data is not clearly separable, see Figure 7. Thus, we chose not to perform dimensionality reduction techniques on our training dataset.

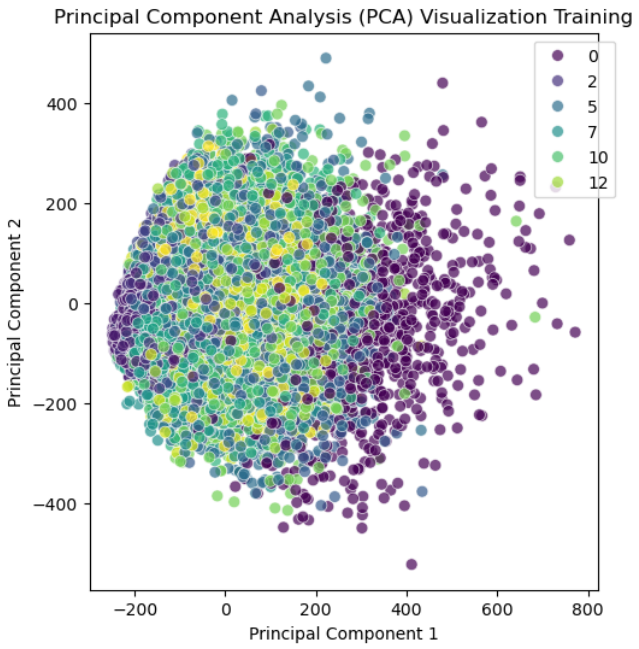

Figure 7. plot of two principal components found through PCA of data, confirms no need for further dimensionality reduction

In the final stage of EDA, we perform data augmentation. To enhance the robustness of our models and mitigate overfitting, we augmented our training and testing dataset. For each image, a random rotation from -30 to 30 degrees was created and added to the dataset. This increased the diversity of our training data and simulated real-world variations.

4. Proposed methodology

To address the challenge of recognizing Chinese numeral characters, three widely-known convolutional neural network (CNN) architectures were evaluated: LeNet, VGG-16, and ResNet-18. These models were selected to evaluate the effect of different network depths and architectural complexities on the classification of Chinese digits.

4.1. Model Architectures

4.1.1. LeNet. LeNet, also known as LeNet-5, was used as the simplest CNN designed for our handwritten character recognition project with decent accuracy. In LeNet, every convolutional layer includes convolution, pooling, and nonlinear activation functions. The model has one input channel, taking in a 64x64 grayscale image as the input and outputting 15 classes which are the 15 chinese number characters, and has 2 convolutional layers. In the first layer, it accepts 1 channel and outputs 64 channels, the size of the convolving kernel being 5x5 and a padding of 2. An activation function is applied, which ReLU for all hidden layers. Then, the pooling layer subsamples by using max pooling to select the brighter pixels of the image, which is what we are interested in. The second layer uses the same method. The convolutional layers are then flattened to prepare them for the fully connected layers to ensure compatibility. There are 3 fully connected layers, also using ReLU as the activation function, and a dropout layer is placed after each to prevent overfitting.

4.1.2. VGG-16. VGG-16 was selected as a medium between the simple LeNet and the complex ResNet. We adapted the model to only have one channel for grayscale input for the 64x64 image, and to only output 15 rather than 1000 classes. The first five blocks are for feature extraction/convolution. Each block has 4 main components:

1) Convolution: Convolve a 3x3 kernel over all image channels, with padding of 1 to maintain dimensionality.
2) Batch Normalization: Normalize output of convolution layer for each training batch with the following normalization equation. $y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}}$
3) ReLU(x)=max(0,x), adding non-linearity to output.
4) 2D pooling: halve feature map length and width via selecting max value in 2x2 kernel.

Blocks 1 and 2 of the feature extraction blocks include components 1→2→3→4.

Blocks 3 to 5 include 1→2→3→1→2→3→4.

Feature extraction blocks output final dimensions of 2x2x512.

This is fed into two fully connected linear classifying layers with ReLU activation, with dropout if applicable.

Dimensions evolves as follows:

- Layer 1 :512x2x2 →4096
- Layer 2: 4096 → 4096

4.1.3. ResNet-18. ResNet-18,the most complex model, uses residual blocks (2 convolutional layers with 3x3 filters, then batch normalization and ReLU activation) with identity skip connections (block input added to output of the second convolution). Global average pooling flattens dimensions after a residual block, after which a fully connected layer with softmax is used for classification.

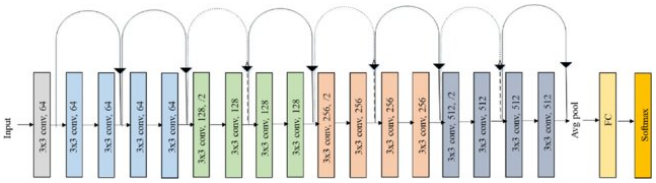

Figure 8. ResNet-18 architecture[6]

For our dataset, we modified ResNet to accept single-channel grayscale images by setting the number of input channels to 1 and adding a dropout layer before the output

layer name	output size	18-layer
conv1	112×112	7x7, 64, stride 2
conv2_x	56×56	3x3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$
	1×1	Average pool, 1000-d fc, softmax
FLOPs		1.8×10^9

Figure 9. ResNet-18 layer sizes, kernels, pooling, and dimensionalities, adapted from [3]

	LeNet	VGG-16	ResNet
Dropout rate	[0.3, 0.5]	[0.3, 0.5]	[0,0.3, 0.5]
Learning rate	[0.001, 0.01]	[0.001, 0.01]	[0.001, 0.01]
Batch size	[64,256]	[32,64]	[64,128]
Number of Epochs	[20,40]	[20,40]	[20,40]
Optimizer	['SGD', 'Adam']	['SGD', 'Adam']	['SGD', 'Adam']

TABLE 1. RESNET HAD AN ADDITIONAL 'INPUT SIZE' HYPERPARAMETER [64,128] AND VGG-16 HAD AN ADDITIONAL REGULARIZATION HYPERPARAMETER [0.0001, 0.001].

layer to prevent overfitting. The final fully connected layer was replaced with a linear layer having 15 output neurons. Input images were resized to one of two dimensions—the original 64×64 or upscaled 128×128— based on the hyperparameter input_size. Preprocessing included normalization with a mean of 0.5 and a standard deviation of 0.5 for the single input channel.

4.2. Training Procedure

Grid search was performed to optimize the hyperparameters.

4.3. Experimental Setup

Since three group members each trained a model, the experimental setup differed across models. PyTorch's deep learning framework was used to implement and train all models.

Model	CPU	GPU	Time
LeNet	8-CoreAMD Ryzen 7 3700X	NVIDIA GeForce RTX 3080	30 mins
VGG-16	Intel i9-10900K CPU @ 3.70 GHz	NVIDIA GeForce RTX 3070	10 hrs
ResNet	13th Gen Intel i9-13900HX	NVIDIA GeForce RTX 4060	3hrs

TABLE 2. WORKSTATION DETAILS

5. Experimental results and evaluation

To evaluate the performance of the three models—VGG-16, ResNet, and LeNet—we utilized precision, accuracy, F1-score, and recall as key metrics. An evaluation function was implemented to compare predictions against true labels from the prepared test data. This function calculated overall accuracy, generated a classification report including per-class precision, recall, and F1-scores, and produced a confusion matrix for visualizing model performance.

5.1. VGG-16 Model

Starting with the VGG-16 model, it achieved an impressive overall accuracy of 99.67% (see Figure 10) on the testing data. The classification report showed that the precision, recall, and F1-scores for most classes were nearly perfect. However, digits 3 and 11 exhibited slightly lower scores, likely due to structural similarities with other characters. For example, digit 3 often resembled digit 2 when strokes were incomplete or blurred. There were also two misclassifications between digits 10 and 12, which makes sense given their similar structures.

The F1-score, which balances both precision and recall, mirrored this trend, indicating that the VGG-16 model effectively minimized both false positives and false negatives. Further analysis of the macro average –which treats all classes equally without considering class distribution– and the weighted average–which accounts for the contribution of each class based on its number of samples, weighting each class by the number of samples in that class– revealed strong values over 0.99 for both metrics (see Figure 10). These results show the model's robustness across all classes.

5.2. ResNet

The ResNet model also performed exceptionally well, achieving an overall accuracy of 99.33% (see Figure 11). However, the precision scores for digits 10, 11, 12, and 13 showed a slight drop, again likely due to the similarity in their structures, as they all have a horizontal bar. Main findings from the confusion matrix were that digits 13and 6were also confused with each other twice, and digit 3

Model: VGG16
Overall Accuracy: 99.67%

Classification Report:

	precision	recall	f1-score	support
Digit 0	0.994286	1.000000	0.997135	174.000000
Digit 1	1.000000	1.000000	1.000000	234.000000
Digit 2	0.979798	1.000000	0.989796	194.000000
Digit 3	1.000000	0.973333	0.986486	150.000000
Digit 4	1.000000	0.995327	0.997658	214.000000
Digit 5	1.000000	0.995413	0.997701	218.000000
Digit 6	1.000000	0.994792	0.997389	192.000000
Digit 7	0.990385	1.000000	0.995169	206.000000
Digit 8	1.000000	1.000000	1.000000	226.000000
Digit 9	1.000000	1.000000	1.000000	212.000000
Digit 10	0.990000	1.000000	0.994975	198.000000
Digit 11	1.000000	0.994792	0.997389	192.000000
Digit 12	1.000000	0.990476	0.995215	210.000000
Digit 13	0.994595	1.000000	0.997290	184.000000
Digit 14	1.000000	1.000000	1.000000	196.000000
accuracy	0.996667	0.996667	0.996667	0.996667
macro avg	0.996604	0.996276	0.996414	3000.000000
weighted avg	0.996710	0.996667	0.996665	3000.000000

Figure 10. VGG 16 classification report

was misclassified as 2 thrice, further contributing to the drop in metrics. Despite these issues, the ResNet model demonstrated strong performance overall.

Model: ResNet
Overall Accuracy: 99.33%

Classification Report:

	precision	recall	f1-score	support
Digit 0	1.000000	1.000000	1.000000	174.000000
Digit 1	0.995745	1.000000	0.997868	234.000000
Digit 2	0.979695	0.994845	0.987212	194.000000
Digit 3	0.986486	0.973333	0.979866	150.000000
Digit 4	1.000000	0.995327	0.997658	214.000000
Digit 5	1.000000	0.986239	0.993072	218.000000
Digit 6	0.994737	0.984375	0.989529	192.000000
Digit 7	0.985646	1.000000	0.992771	206.000000
Digit 8	0.995536	0.986726	0.991111	226.000000
Digit 9	1.000000	0.995283	0.997636	212.000000
Digit 10	0.985075	1.000000	0.992481	198.000000
Digit 11	0.994819	1.000000	0.997403	192.000000
Digit 12	0.995215	0.990476	0.992840	210.000000
Digit 13	0.983871	0.994565	0.989189	184.000000
Digit 14	1.000000	0.994898	0.997442	196.000000
accuracy	0.993333	0.993333	0.993333	0.993333
macro avg	0.993122	0.993071	0.993072	3000.000000
weighted avg	0.993380	0.993333	0.993332	3000.000000

Figure 11. ResNet classification report

While accuracy provides a good overview of correct predictions, it can be misleading in multiclass classification tasks, as it does not account for how well the model performs across individual classes. In this context, metrics like precision, recall, and F1-score offered deeper insights into the model's strengths and weaknesses. Although ResNet was the most complex model, it fell slightly short of VGG-16's performance, illustrating the latter's superiority despite ResNet's skip-connection architecture.

5.3. LeNet

The LeNet model, being the simplest architecture used in our study, achieved an overall accuracy of 96.90% (Figure 12), which is slightly lower than both VGG-16 and ResNet.

Model: LeNet
Overall Accuracy: 96.90%

Classification Report:

	precision	recall	f1-score	support
Digit 0	0.994118	0.971264	0.982558	174.000
Digit 1	0.991489	0.995726	0.993603	234.000
Digit 2	0.941463	0.994845	0.967419	194.000
Digit 3	0.992857	0.926667	0.958621	150.000
Digit 4	0.985915	0.981308	0.983607	214.000
Digit 5	0.977064	0.977064	0.977064	218.000
Digit 6	0.978495	0.947917	0.962963	192.000
Digit 7	0.975728	0.975728	0.975728	206.000
Digit 8	0.969697	0.991150	0.980306	226.000
Digit 9	0.931818	0.966981	0.949074	212.000
Digit 10	0.927536	0.969697	0.948148	198.000
Digit 11	0.958974	0.973958	0.966408	192.000
Digit 12	0.984536	0.909524	0.945545	210.000
Digit 13	0.972678	0.967391	0.970027	184.000
Digit 14	0.964467	0.969388	0.966921	196.000
accuracy	0.969000	0.969000	0.969000	0.969
macro avg	0.969789	0.967907	0.968533	3000.000
weighted avg	0.969549	0.969000	0.968975	3000.000

Figure 12. LeNet classification report

Digits 10 and 12 were the most confused, at 14 misclassifications, with confusing between digits 2and 3. Digits 9and 14 were the third most confused. This makes sense, given that these characters have similar structures and LeNet has relatively shallow architecture, unable to capture more complex patterns as well. For more distinct characters, however, like digits 1 and 0 , the character had high precision scores (Figure 12).

Since we are doing multiclass classification where some of the digits look similar, evaluating based on a single metric will not capture all of the nuances. Thus, we will also look at per class precision , recall, and f1 for each model to get more insight to how the individual models perform.

Comparing the models to each other, we can see that the VGG16 model performed better than the rest on all of the metrics, outperforming ResNet with its deeper layers, despite ResNet's more advanced skip-connection architecture.

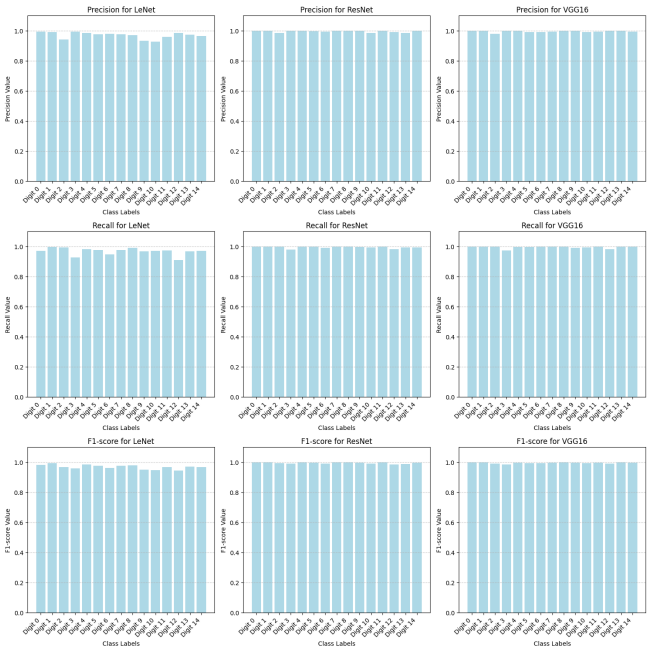

Figure 13. Precision, Recall, and F1-score for each model and class

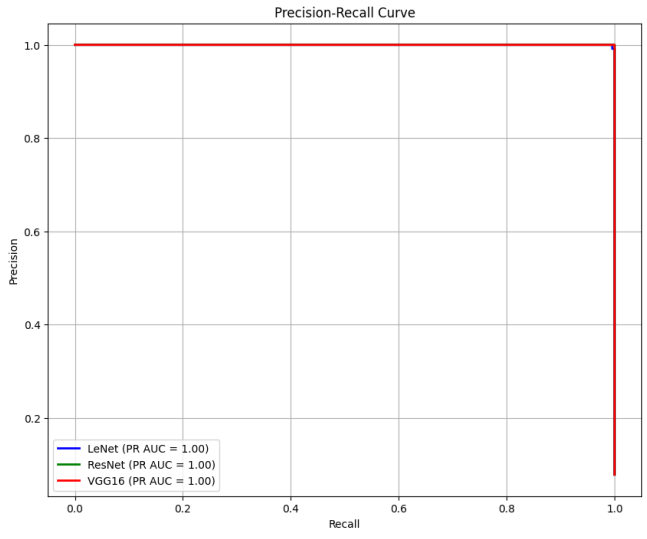

Figure 14. Precision Recall Curve (PRC) of the three models

The high metric values can be a sign of overfitting however; the PRC curves of the models (Figure 14), specifically used for unbalanced datasets, all having total area under the curve of 1 raises concerns about potential overfitting, particularly given the relatively small size of our dataset. To mitigate this, we employed data augmentation techniques, such as rotation, to introduce more variability into the training data. Although we took those measures, it seems we need to do more to prevent overfitting, such as increasing the dataset size and variability.

One notable aspect of the evaluation process was the tradeoff between performance and computational complexity. While the VGG-16 model demonstrated the highest accuracy and robustness, it also required significantly more computational resources and time to train compared to ResNet and LeNet, as it is a deeper model then the both. The ResNet model, despite being more complex than LeNet, achieved comparable results with a shorter training time due to its efficient use of skip connections. LeNet, being the simplest architecture, was the fastest to train but struggled with the more complex handwritten digits.

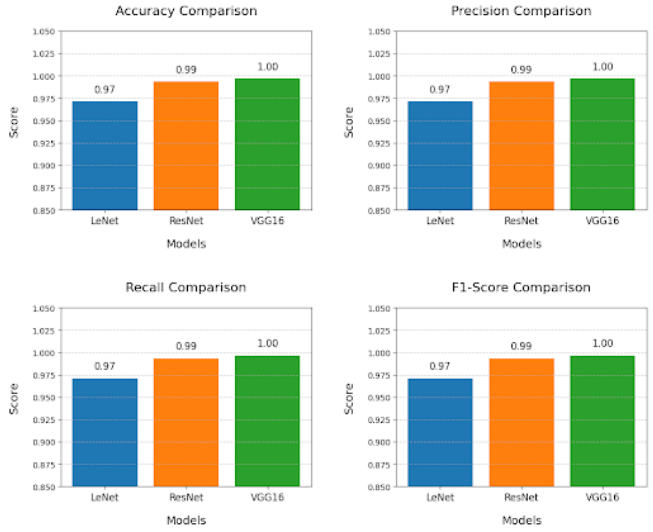

Figure 15. Bar Graph Comparison of Accuracy, Precision, Recall, and F1-Score

From the bar graph (Figure 15), we observe that VGG-16 consistently outperformed both ResNet and LeNet across all metrics. However, this performance came at a cost, as it required significantly more computational resources due to its large number of parameters. ResNet achieved a good balance between performance and efficiency, making it a suitable choice for scenarios with limited computational power. LeNet, while the simplest, demonstrated solid performance on simpler classes but lacked the capacity to handle complex digits effectively.

6. Conclusion and discussion

To achieve our goal of handwritten Chinese character recognition, we implemented and evaluated eminent CNN image classification models on a subset of Chinese numerals as a proof of concept—LeNet, VGG-16, and ResNet-18. The experimental results showed high accuracy for all models, with ResNet-18 and VGG-16 both outperforming LeNet. By exploring these three architectures, we were able to explore the trade-offs between training time, accuracy, and computational complexity.

Although our results on the 15 class (each with 1,000 images) numerical dataset were promising, it was

relatively small and simple compared to typical datasets used for deep learning, which may have constrained the performance of deeper models like ResNet-18. Furthermore, training ResNet-18 and VGG-16 – the more complex models – was computationally intensive, taking hours to train and perform grid search, even with GPU acceleration. In hindsight, these approaches may have been too complex for such a simple dataset. As such, given the already small scope of our project, expanding to other characters with the same approach would not be feasible, requiring many more computational resources.

As such, future directions could include exploring and evaluating more computationally simple approaches to Chinese character recognition, such as MobileNet or EfficientNet (lightweight CNN architectures). On our demonstrative website, characters were often misclassified when written off-center or larger/smaller than the training image set – implying that our models overfit to our training data. Thus, to prevent overfitting and improve accuracy, more data augmentation methods could be implemented, such as scaling, noise addition, distortion transformation, and translation.

Dimensionality reduction methods could also be further explored. Although principal component analysis (PCA) was performed on the data, it was only exploratory (for visualization purposes) and not used for model training. Furthermore, the number of principal components we chose to visualize, two, was much too low for the 4096 features (each pixel in the image). So to realistically reduce dimensionality, higher-dimensional latent spaces could be tested, perhaps on the scale of 100-1000 features. Afterwards, to potentially speed up training, post-PCA image data could be used as training data, and its performance compared to training with the whole image. Other dimensionality reduction techniques, like autoencoders, could also be explored to reduce the computational cost.

7. Github link and Project roadmap and any comments about assignment of tasks to team members

Our project was broken down into 4 main stages: finding a suitable dataset, exploratory data analysis (EDA), model implementation, and final report writing. Our group leader was Yiming Feng.

All team members collaborated in finding a dataset. We had our first meeting on October 1, 2024 to discuss our choice of dataset, initially deciding on the Chinese Character Dataset. However, we changed our minds on October 22, 2024, deciding to switch to the Social Media Menace dataset. However, after Yiming Feng finished EDA on the dataset, we found very little correlation between features; furthermore, when investigating further for dataset description, we found the dataset to be artificially generated .

Thus, we made the decision to return to the Chinese character dataset on October 29, 2024, this time choosing a simpler subset of 15 different handwritten Chinese characters. Exploratory Data Analysis tasks were then assigned to be completed by November 11 as follows: Pixel Intensity Analysis (Angela Zhou), Character Pattern Analysis (Rithika Krishna Perugupalli), Sharpness and Edge Detection (Yiming Feng), Intra-Character Consistency Check (Stephanie Hsia), and Data Augmentation (Brian Li).

These tasks were essential for moving forward with model development. Three models were chosen according to the literature review done by Brian Li (November 26, 2024) and each assigned to a group member to complete by November 26, 2024: VGG-16 (Yiming Feng), LeNet (Stephanie Hsia), and ResNet (Angela Zhou).

Once completed, we moved forward with preparing the final project report, demo web page, and five-minute video presentation, with a deadline of December 10, 2024. Responsibilities were assigned as follows:

- Yiming Feng: Webpage development, 5-minute video presentation, converting final report to LaTeX.
- Angela Zhou: Proposed methodology (VGG-16 and ResNet-18), conclusion and discussion, project roadmap, final editing.
- Stephanie Hsia: Proposed methodology (LeNet), references.
- Rithika Krishna Perugupalli: Model evaluation and benchmarking.
- Brian Li: Dataset description and exploratory data analysis of the dataset, introduction and background, literature review.

https://github.com/quiet98k/Chinese-Digit-Classifier

References

[1] Ruwei Dai, Chenglin Liu, and Baihua Xiao. "Chinese character recognition: history, status and prospects". In: *Frontiers of Computer Science in China* 1.2 (May 2007), pp. 126–136. ISSN: 1673-7466. DOI: 10.1007/s11704-007-0012-5. URL: http://dx.doi.org/10.1007/s11704-007-0012-5.

[2] Dongdong He and Yaping Zhang. "Research on Artificial Intelligence Machine Learning Character Recognition Algorithm Based on Feature Fusion". In: *Journal of Physics: Conference Series* 2136.1 (Dec. 2021), p. 012060. ISSN: 1742-6596. DOI: 10.1088/1742-6596/2136/1/012060. URL: http://dx.doi.org/10.1088/1742-6596/2136/1/012060.

[3] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[4] mashimo. *Chinese MNIST with simple PyTorch CNN step by step*. URL: https://www.kaggle.com/code/mashimo/chinese-mnist-with-simple-pytorch-cnn-step-by-step.

[5] Gabriel Preda. *Chinese MNIST*. URL: https://www.kaggle.com/datasets/gpreda/chinese-mnist.

[6] Farheen Ramzan et al. “A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer’s Disease Stages Using Resting-State fMRI and Residual Neural Networks”. In: *Journal of Medical Systems* 44.2 (Dec. 2019). ISSN: 1573-689X. DOI: 10.1007/s10916-019-1475-2. URL: http://dx.doi.org/10.1007/s10916-019-1475-2.

[7] Rohit Thakur. *Beginners guide to VGG16 implementation in keras*. URL: https://builtin.com/machine-learning/vgg16.

[8] Xu-Yao Zhang et al. “Drawing and recognizing Chinese characters with recurrent neural network”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 40.4 (Apr. 2018), pp. 849–862.