Kuzushiji Classification Three Ways

Ronan Bottoms

March 14, 2024

Abstract

A K-Nearest Neighbors Classifier, a Fully Connected Neural Network, and a Convolutional Neural Network are applied to the classification of a large, imbalanced dataset of images of kuzushiji. The analysis focuses on testing accuracy and testing runtime as well as the effects of dimension reduction on both accuracy and runtime.

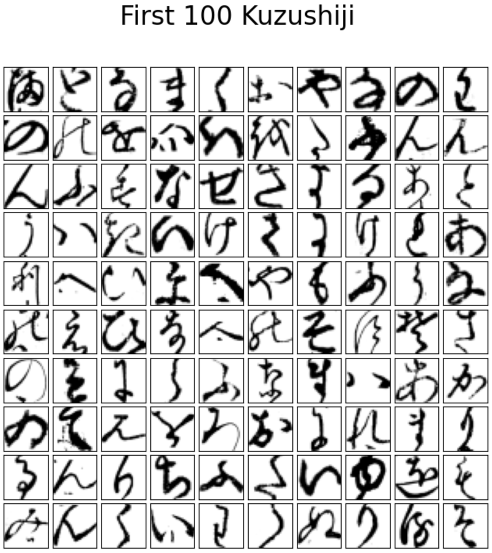

Figure 1: The first 100 kuzushiji of the training set from Kuzushiji49.

1 Introduction

The Japanese language uses two alphabets in addition to Chinese characters. “Letters” in these alphabets originally derived from Chinese characters themselves, often times with multiple different characters representing the same sound. For instance, before the alphabets were standardized, the “a” sound could be represented by not only 安 , but also 阿 and 愛 [2]. Cursive or sloppy penmanship can lead to abridged versions of the characters, like how 安 morphed into the modern day あ . These obsolete or antiquated short form Chinese characters are referred to as kuzushiji (崩し字 [lit. crumbling/destroyed character]).

Nowadays, only one form of each letter exists. But if one wants to read a historical document written prior to 1900, a knowledge of the many old forms of letters is necessary. For a layperson, or someone whose native language is not Japanese, learning these different forms may be difficult and time-consuming, especially if only one or two documents need to be read. The aim of this report is to create, test, and compare three different machine learning algorithms tasked with classifying the different kuzushiji that may appear.

The Kuzushiji49 dataset, part of the Kuzushiji-MNIST [1] collection was used to test the models. This dataset contains 270, 912 grayscale images of 49 different classes of kuzushiji (all 48 hiragana characters and one iteration mark). The dataset is imbalanced, so there may be some classes with greater or fewer samples than the average. The analysis used three different kinds of classifier: a K-Nearest Neighbors classifier (KNN), a Fully-Connected Neural Network (FCN), and a Convolutional Neural Network (CNN). The KNN classifier is a more elementary machine learning algorithm and represents a naive approach to solving the classification problem.

The analysis aimed to answer the following questions:

1. Which of the three models is the most accurate?

2. Which of the three models is the most accurate *in the least amount of time*?

3. How does dimension reduction affect accuracy and runtime in all three models?

Additionally, this analysis posed an opportunity to test a hyperparameter tuning of the FCN that was left unanswered in prior analyses. The following question was added as a secondary goal:

4. How does the number of neurons and layers affect testing accuracy and runtime?

Prior to analysis, the following hypothesis was proposed in response to the target questions:

1. The CNN will be the most accurate as it is known to be powerful for image data.

2. The CNN will not only be the most accurate, but also the fastest due to the above reason.

3. Dimension reduction will in general decrease both accuracy and testing time. However, since dimension reduced images still need to be projected back to 28×28 pixels to be passed through the CNN, dimension reduction will not affect the testing time of the CNN as greatly.

4. Increasing both neurons and layers will increase accuracy and testing time.

2 Methods

The analysis proceeded in two phases: testing the models using the "full", unreduced data, and testing them using the dimension reduced data. The first step of the analysis was then to determine how many dimensions to reduce to. The cumulative energy contained in different numbers of PCA modes was computed using the methods developed in class through the `sklearn.decomposition` package. A number of PCA modes that achieved a target of 90% of cumulative energy was chosen as the dimension to reduce the data to.

Testing of the models was performed using the dedicated testing data provided by Kuzushiji49. This dataset consisted of $38,547$ images. Testing times were computed as the wall-clock time required to classify each sample in the testing set.

The KNN was implemented using the `sklearn.neighbors` package. The model was tested for 1 through 10 neighbors and the testing accuracy and testing time were plotted as a function of the number of neighbors.

The FCN was implemented using the `PyTorch` package, building on the scalable model written for the author's submission for a prior homework. The model has an adjustable number of layers and neurons per layer, which allows for easy hyperparameter tuning to determine the effects of the number of neurons and layers on accuracy and speed. It used a learning rate of 0.001, ADAM as its optimizer, Cross Entropy Loss as the loss function, and was trained over 40 epochs. Batch normalization was applied after each layer, but dropout regularization was not used.

The CNN was also implemented using `PyTorch`. It was built with two convolutional layers, each max-pooled, and two linear layers. Batch normalization was applied at each step. For continuity with the FCN, it also used a learning rate of 0.001, ADAM as its optimizer, Cross Entropy Loss as the loss function, and was trained over 40 epochs.

Other additional packages used were `tqdm`, `numpy`, `pandas`, `matplotlib`, and `seaborn`.

Note that runtimes are incredibly machine specific. The machine used to perform the analysis for this report used a dedicated graphics card, which may have decreased runtimes compared to other machines using only a CPU.

3 Results

The first step of the analysis was to determine how many PCA dimensions were required to retain 90% of the energy the training data. The results are shown in Figure 2. The graph shows that 138 dimensions are required to attain 90% energy. This means that from data of 784 dimensions, 90% of the structure is captured using only 138, or 17.6%, of the dimensions. This reduction has the capacity to greatly reduce runtimes while keeping accuracy high. The reconstruction of the first 100 kuzushiji after being projected to 138 dimensions are shown in Figure 3. They look remarkably similar to those in Figure 1, despite some distortion in the backgrounds of the images.

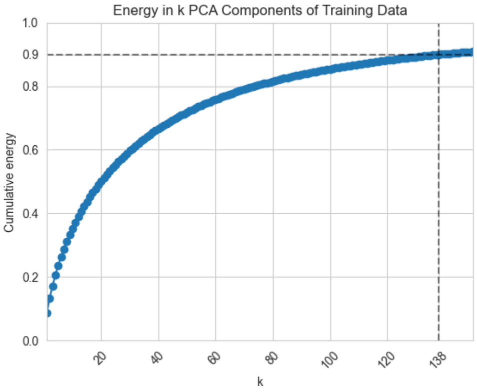

Figure 2: The cumulative energy contained in k PCA components of the training data.

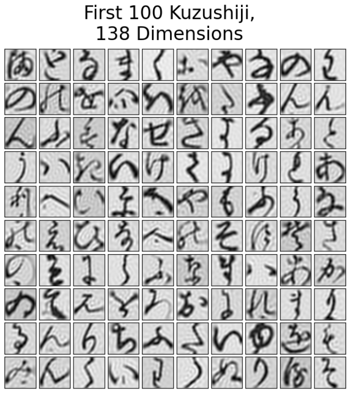

Figure 3: The first 100 kuzushiji reconstructed from projection to 138 dimensions.

3.1 KNN

Analysis of the KNN model immediately encountered a disruption – using the full data proved to be infeasible as a solution. Training for the model using full data and only one neighbor ran for nearly thirty minutes uninterrupted and did not finish before being forcibly terminated. This extremely long runtime ruled out the KNN model using full data as it took too long to train to considered as a valid solution.

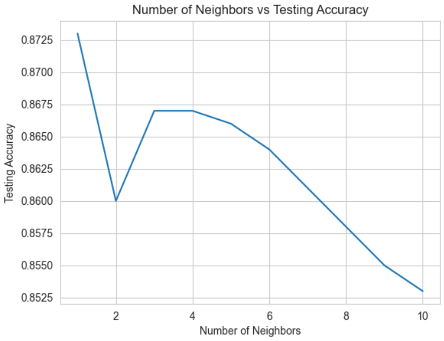

Figure 4: Testing accuracy of the KNN classifier at 138 dimensions vs number of neighbors.

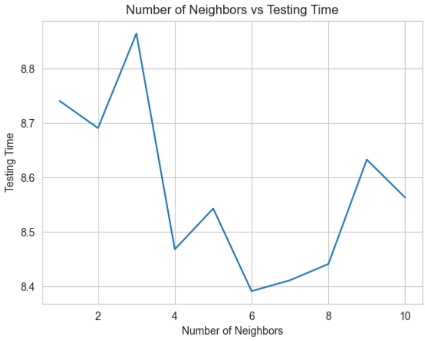

Figure 5: Testing time of the KNN classifier at 138 dimensions vs number of neighbors.

Using the reduced data, however, the model performed well. The results are shown in Figure 4 and

Figure 5. Overall, the model produced a reasonable accuracy between 85% and 88%. Surprisingly, the highest accuracy occurred with only a single neighbor, and the plot generally shows an inversely proportional relationship between accuracy and the number of neighbors, with an anomalous sharp decrease in accuracy occurring at two neighbors.

The testing time of the model, however, was quite slow. The fastest testing time, occurring at six neighbors, was about 8.468 seconds. The testing time shows no clear correlation between number of neighbors and testing time.

Table 1: Output table of KNN classification using reduced data.

Neighbors	Train	Test	Val Mean	Val STD	Test Time
1	1.0	0.873	0.914	0.001	8.741
2	0.959	0.86	0.902	0.001	8.691
3	0.955	0.867	0.912	0.001	8.865
4	0.948	0.867	0.912	0.001	8.468
5	0.943	0.866	0.912	0.001	8.543
6	0.939	0.864	0.91	0.001	8.391
7	0.935	0.861	0.909	0.001	8.411
8	0.932	0.858	0.907	0.001	8.441
9	0.929	0.855	0.906	0.001	8.633
10	0.927	0.853	0.904	0.001	8.563

3.2 FCN

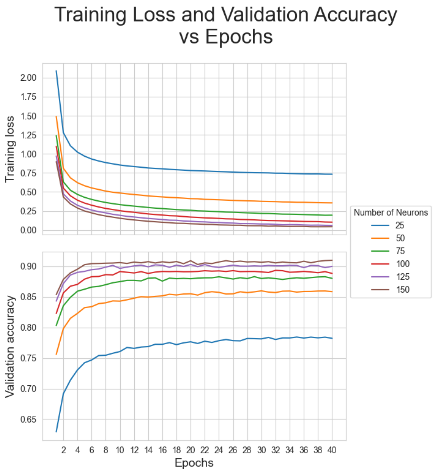

Figure 6: Loss vs validation accuracy for different number of neurons per layer using **full** data.

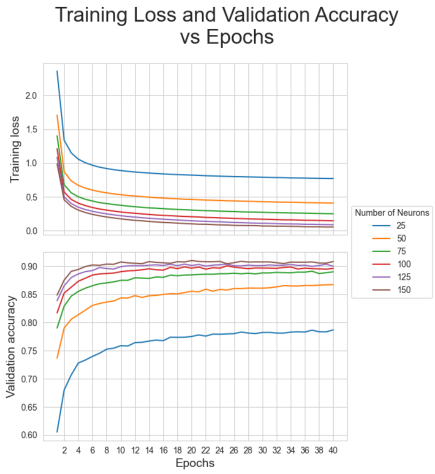

Figure 7: Loss vs validation accuracy for different number of neurons per layer using **reduced** data.

The training of the FCN proceeded without problems. The model was trained independently adjusting the number of neurons per layer across 3 hidden layers and the number of layers with 100 neurons per layer. The full outputs of each trial are given in Tables 2, 3, 4, and 5. The resulting curves for training loss and validation accuracy are given for full and reduced data in Figures 6 (left) and 7 (right) respectively. The data supports the initial hypothesis that more neurons per layer correlate with reduced loss and increase accuracy. Noticeably, there is no significant observable difference between the use of full data or reduced data; both sets produce similar training loss and validation accuracy. This is an unexpected result contrary to the hypothesis posed.

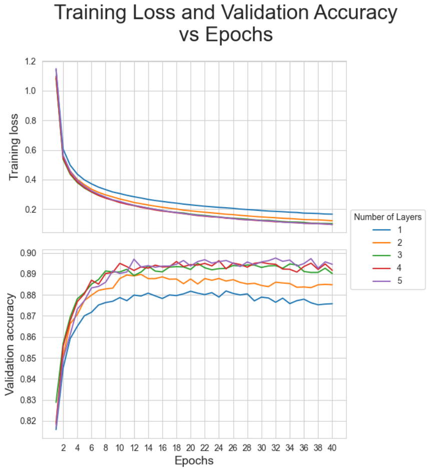

Figure 8: Loss vs validation accuracy for different number of layers using **full** data.

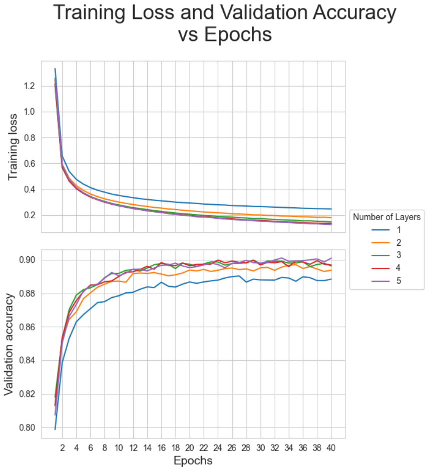

Figure 9: Loss vs validation accuracy for different number of layers using **reduced** data.

The results of tuning the layers are displayed in Figures 8 and 9. These plots show a similar result to the neurons, with accuracy values much closer in value than what was anticipated. The greatest validation accuracy was achieved in both with five layers at about 90%.

Next the testing accuracy for both neuron and layer tuning was considered. The testing accuracy as a function of neurons for both full and reduced data is shown in Figure 10 and the testing accuracy as a function of layers is shown in Figure 11. The plots show that the *reduced* data is slightly more accurate than the full data. This is an unexpected result contrary to the hypothesis. Intuition says that using “blurrier” images would produce a more inaccurate classification, as indeed the 100 reconstructed kuzushiji in Figure 3 have more noise than their non-dimension-reduced counterparts.

The FCN was slightly less accurate than the KNN algorithm using the largest value of neurons and layers. If the network were trained for longer or tuned more efficiently, it is probable that the FCN would achieve greater accuracy than the KNN.

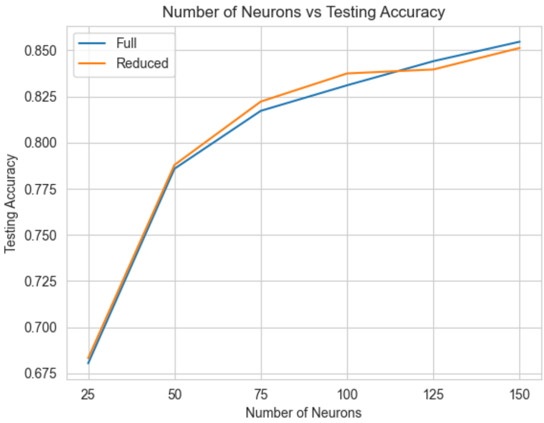

Figure 10: Testing accuracy for different numbers of neurons.

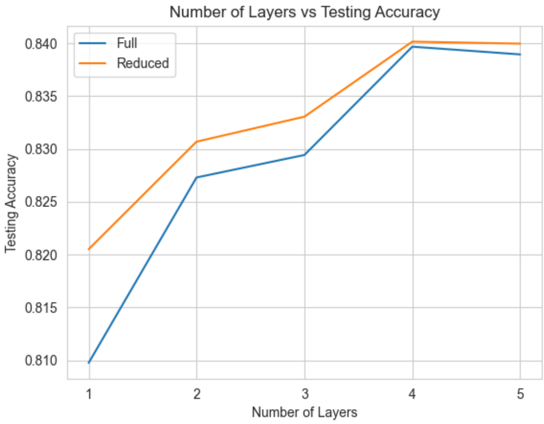

Figure 11: Testing accuracy for different numbers of layers.

The testing time as a function of neurons for both types of data is shown in Figure 12 and its counterpart for number of layers is shown in Figure 13. Of import is how much faster the FCN model is compared to KNN. The KNN with reduced data classified the training set on the order of 8 seconds; the FCN classified the testing data on the order of less than one second. This is consistent with intuition and empirical experience from prior homeworks. Additionally, once again the reduced data proves superior to the full data. However, while the reduced data is faster, it is not faster by much compared to the difference between KNN and FCN.

Finally, there is a loose trend that more neurons and layers correspond to longer test times, but not by a substantial amount.

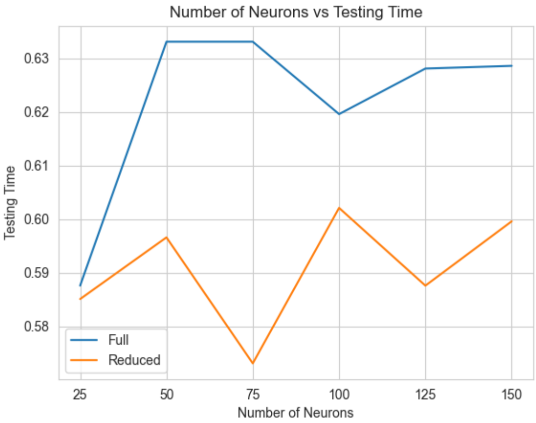

Figure 12: Testing time for different numbers of neurons.

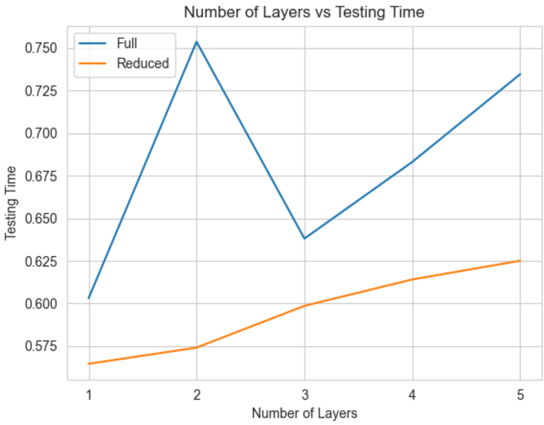

Figure 13: Testing time for different numbers of layers.

Full Data

Table 2: Output table of neuron tuning using full data (values rounded to 3 places).

Neurons	Layers	Final Loss	Final Val Accuracy	Test Accuracy	Test Time
25	3	0.733	0.782	0.680	0.588
50	3	0.357	0.859	0.786	0.633
75	3	0.196	0.880	0.817	0.633
100	3	0.104	0.888	0.831	0.620
125	3	0.061	0.900	0.844	0.628
150	3	0.0443	0.910	0.855	0.629

Table 3: Output table of layer tuning using full data (values rounded to 3 places).

Layers	Neurons	Final Loss	Final Val Accuracy	Test Accuracy	Test Time
1	100	0.1667	0.876	0.810	0.603
2	100	0.122	0.885	0.827	0.754
3	100	0.1027	0.890	0.829	0.638
4	100	0.0970	0.892	0.840	0.683
5	100	0.0967	0.895	0.839	0.735

Reduced Data

Table 4: Output table of neuron tuning using reduced data (values rounded to 3 places).

Neurons	Layers	Final Loss	Final Val Accuracy	Test Accuracy	Test Time
25	3	0.773	0.787	0.683	0.585
50	3	0.410	0.867	0.788	0.597
75	3	0.252	0.890	0.822	0.573
100	3	0.148	0.896	0.837	0.602
125	3	0.090	0.900	0.839	0.588
150	3	0.057	0.908	0.851	0.600

Table 5: Output table of layer tuning using reduced data (values rounded to 3 places).

Layers	Neurons	Final Loss	Final Val Accuracy	Test Accuracy	Test Time
1	100	0.248	0.889	0.821	0.565
2	100	0.181	0.894	0.831	0.574
3	100	0.148	0.896	0.833	0.599
4	100	0.136	0.897	0.840	0.614
5	100	0.128	0.901	0.840	0.625

3.3 CNN

The plot of CNN loss and training accuracy is shown in Figure 14. Training of the CNN experienced difficulties using the reduced model, and validation accuracy remained near zero. Testing accuracy and testing speed are shown in Table 6. Astoundingly, the testing speed of the CNN is vastly slower than that of the FCN, about eight times as slow. This completely disproves the analysis hypothesis. Testing accuracy was high as well at 91.2%, about $5 - 6\%$ higher than the FCN, but whether this is big enough an increase to warrant the longer testing time, especially if the FCN is tuned more optimally, remains to be seen.

A concern that arose is whether the max pool layers may have contributed to the failure of the model to train using reduced data. The model was altered to remove the max pool layers and rerun. The results are shown in Figure 15. The loss curve of for the reduced data is much better, decreasing from near 4 to below 0.5. The accuracy of the reduced data however did not increase. Interestingly the accuracy of the full data was affected as well, decreasing about 3%.

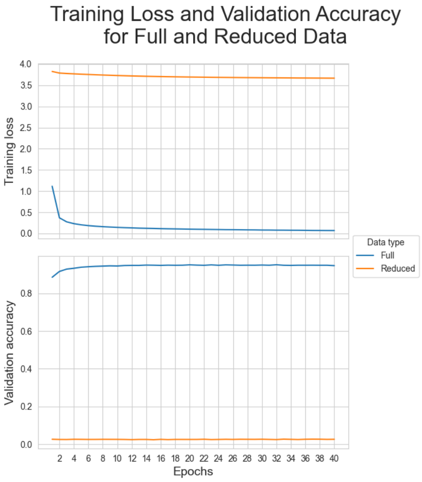

Figure 14: Training loss and validation accuracy for the CNN using full and reduced data **with** pool layers.

Table 6: Output table for testing of full and reduced data **with** max pool layers (values rounded to 3 places).

Data	Final Loss	Final Val Accuracy	Test Accuracy	Test Time
Full	0.069	0.947	0.912	4.198
Reduced	3.668	0.026	0.027	4.667

Table 7: Output table for testing of full and reduced data **without** max pool layers (values rounded to 3 places).

Data	Final Loss	Final Val Accuracy	Test Accuracy	Test Time
Full	0.0018	0.937	0.888	4.331
Reduced	0.185	0.0237	0.0241	4.750

4 Conclusion

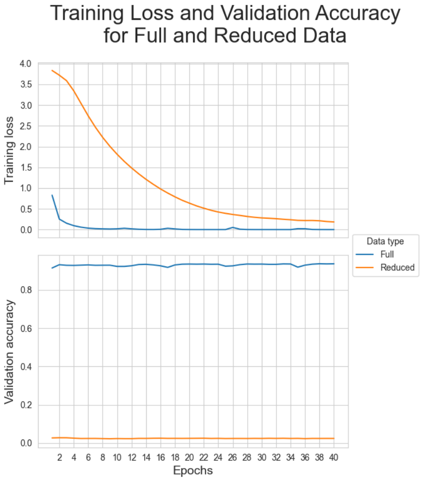

Figure 15: Training loss and validation accuracy for the CNN using full and reduced data **without** using pool layers.

In response to the target questions, the following conclusions were reached.

1. The CNN was the most accurate at about 5% more accurate than the FCN and KNN.
2. The FCN was the fastest algorithm. The CNN was twice as fast as the KNN, and the FCN eight times as fast as the CNN.
3. (a) Dimensionality reduction made the KNN algorithm feasible as it failed to train in a reasonable amount of time using full data.
   (b) The FCN was marginally faster and more accurate using reduced data.
   (c) The CNN failed to train using reduced data.
4. The addition of more neurons and more layers to the FCN generally increased both runtime and accuracy.

4.1 Extensions

The analysis above contains several shortcomings that could be extended in order to be more complete.

First, the FCN received relatively expansive hyperparameter tuning via investigation of the secondary goal. However, the convolutional neural network was tested for only two different configurations. That the CNN is significantly more accurate than the FCN ($> 5\%$) given optimal tuning is a real possibility that this analysis is insufficient in dispelling. A more robust hyperparameter tuning of the CNN would lead to a much more rounded investigation.

Further, while the FCN was tuned for number of layers and number of neurons, this tuning was rather exploratory and without the explicit goal of increasing accuracy or decreasing runtime. A more targeted tuning of the FCN would again lend itself to the completeness of the analysis.

Finally, plotting the confusion matrix for the different classifiers to see which characters they are misclassifying would be an enlightening extension that may help identify shortcomings to the model otherwise not visible.

In conclusion, the next time you need to read an Edo era manuscript and can't parse the kuzushiji, consider using a fully connected neural network to help you identify them.

Acknowledgements

I would like to acknowledge Professor Shlizerman for teaching this course and for his notes that proved helpful when implementing the routines required for this analysis.

References

[1] *Kuzushiji-MNIST*. https://www.kaggle.com/datasets/anokas/kuzushiji. DOI: `10.20676/00000341`.

[2] 国文学研究資料館. *くずし字って何？* 2018. URL: `https://www.nijl.ac.jp/koten/kuzushiji/post.html` (visited on 03/05/2024).