KMNIST Character Classification with Logistic Regression and Convolutional Neural Networks

Noureddin Lutfi
Texas A&M University
noorlutfe@tamu.edu

Aditya Rajiv
Texas A&M University
adiman@tamu.edu

Nuo Chen
Texas A&M University
nuochen@tamu.edu

Diviya Bhavaani M B
Texas A&M University
diviyabhavaani@tamu.edu

***Abstract*—This project studies handwritten Japanese character recognition on the Kuzushiji-MNIST (KMNIST) dataset. We evaluate the performance of a simple multinomial logistic regression baseline and compare it to deeper convolutional neural networks (CNNs). The dataset contains $60{,}000$ training and $10{,}000$ test images of size 28×28 across ten classes. Three CNN architectures are explored: a small two layer CNN, a shallow ResNet style model, and a deeper CNN with batch normalization and dropout, with hyperparameters tuned on a validation set. Logistic regression achieves about 60% accuracy, while the tuned CNN reaches 97%. Confusion matrices show that remaining errors are mostly visually ambiguous characters, highlighting the benefits of representation learning and validation based tuning.**

I. Introduction

Handwritten character classification is a standard benchmark in machine learning [1]. The MNIST dataset [2] of digits popularized convolutional neural networks (CNNs), and modern deep learning techniques are summarized in standard texts [3]. The KMNIST dataset replaces digits with cursive Japanese Kuzushiji characters, which are often visually similar even to humans [4]. This makes KMNIST a challenging testbed for model design and training strategies.

This project investigates supervised classification on KMNIST using classical and deep models. Key questions include:

- How well does multinomial logistic regression perform on flattened images?
- How much improvement is obtained with CNNs?
- How important is validation-based hyperparameter tuning compared to default settings?

Standard CNNs such as LeNet [2] and ResNets [5], often coupled with batch normalization [6] and dropout [7], achieve high accuracy on MNIST/KMNIST [4], [8]. Here, the goal is not state-of-the-art performance but to implement a few models from scratch and analyze the effects of architecture and training strategy.

The main contributions are:

- Comparison of logistic regression and several CNNs on the same KMNIST split.
- Hyperparameter tuning (learning rate, dropout, batch size) for a deeper CNN using a validation set.
- Error analysis via confusion matrices and inspection of misclassified examples.

II. Methodology

A. Dataset and Preprocessing

KMNIST consists of $70{,}000$ gray-scale 28×28 images, split into $60{,}000$ training images and $10{,}000$ test images [4]. There are ten classes, each containing $6{,}000$ training and $1{,}000$ test samples.

The images are normalized to the $[0, 1]$ range by dividing by 255 and reshaped to $(1, 28, 28)$ for CNN input. An $80\%/20\%$ split of the training data is used for training and validation, yielding $48{,}000$ training and $12{,}000$ validation images. For most experiments, the default batch size is 64.

B. Baseline Logistic Regression

A multinomial logistic regression model is trained on flattened 28×28 images, converted into 784-dimensional vectors. Features are standardized using `sklearn`'s `StandardScaler`. The model computes class logits, with predictions made using the softmax function. Training is done on a random subset of $10{,}000$ training samples, achieving a test accuracy of about 60%.

C. CNN Architectures

To better exploit spatial information, three CNN models are implemented in PyTorch:

1) Simple CNN: This model consists of:

- Conv1: 1 input channel, 32 output channels, kernel 3×3, padding 1, ReLU.
- Conv2: 32 input channels, 64 output channels, kernel 3×3, padding 1, ReLU.
- Max-pooling: 2×2 after Conv2, reducing feature map size from 28×28 to 14×14.
- Fully connected layer: $64 \cdot 14 \cdot 14$ to 128 units, ReLU.
- Output layer: 128 to 10 units (one per class).

The network uses cross-entropy loss and the Adam optimizer with a learning rate of 10^{-3} for 5 epochs [9].

2) ResNet-like CNN: This model replaces standard convolutions with residual blocks, each containing two 3×3 convolutions with a skip connection. The architecture is:

- ResidualBlock1: input 1 channel, output 32 channels, followed by ReLU.
- Max-pooling: 2×2.
- ResidualBlock2: input 32 channels, output 64 channels.
- Fully connected layer: $64 \cdot 14 \cdot 14$ to 128 units, ReLU.
- Output layer: 128 to 10 units.

This model is trained for 10 epochs.

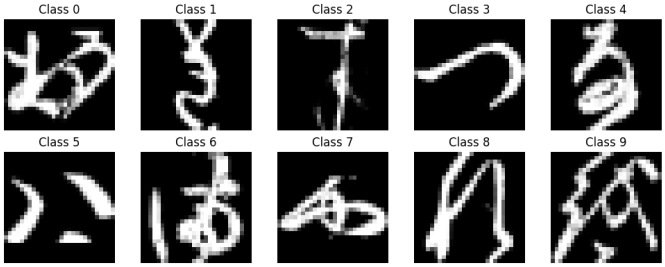


Fig. 1: Example 28×28 grayscale images from the ten KMNIST classes. The grid covers all categories and illustrates the diversity of stroke patterns and character complexity.

3) Improved CNN: This deeper model includes batch normalization and dropout. It is built from three *ImprovedCNNBlock* modules:

- Conv $3 \times 3 \rightarrow$ BatchNorm $\rightarrow$ ReLU.
- Conv $3 \times 3 \rightarrow$ BatchNorm.
- Residual addition (optional 1×1 convolution for channel mismatch).
- Final ReLU.

The full architecture is:

- Block 1: $1 \rightarrow 32$ channels, followed by 2×2 max-pooling.
- Block 2: $32 \rightarrow 64$ channels, followed by 2×2 max-pooling.
- Block 3: $64 \rightarrow 128$ channels, no pooling.
- Flatten to $128 \cdot 7 \cdot 7$.
- Fully connected layer: $128 \cdot 7 \cdot 7$ to 256 units, ReLU.
- Dropout (rate 0.4).
- Output layer: 256 to 10 units.

Batch normalization and dropout help stabilize training and prevent overfitting [6], [7]. The model is trained using Adam with learning rate 10^{-3} for 10 epochs.

D. Hyperparameter Tuning for TunedImprovedCNN

To optimize performance, a grid search is performed over the following hyperparameters:

- Learning rate: $\{0.001, 0.0005\}$,
- Dropout rate: $\{0.3, 0.4, 0.5\}$,
- Batch size: $\{64, 128\}$.

For each combination, the model is trained for 3 epochs, and the best configuration is selected based on validation accuracy. The best configuration (learning rate 0.001, dropout rate 0.4, batch size 128) is then used to retrain the model for 10 epochs, resulting in the final TunedImprovedCNN model.

III. Experimental Results and Discussion

A. Exploratory Analysis

To visualize the global structure, we flatten each image to 784 features, normalize to $[0, 1]$, and apply principal component analysis (PCA) [10] to two dimensions (Fig. 2). The resulting scatter plot shows substantial class overlap and no clear linear separation, indicating that a linear classifier on raw pixels is limited and motivating the use of nonlinear CNNs.

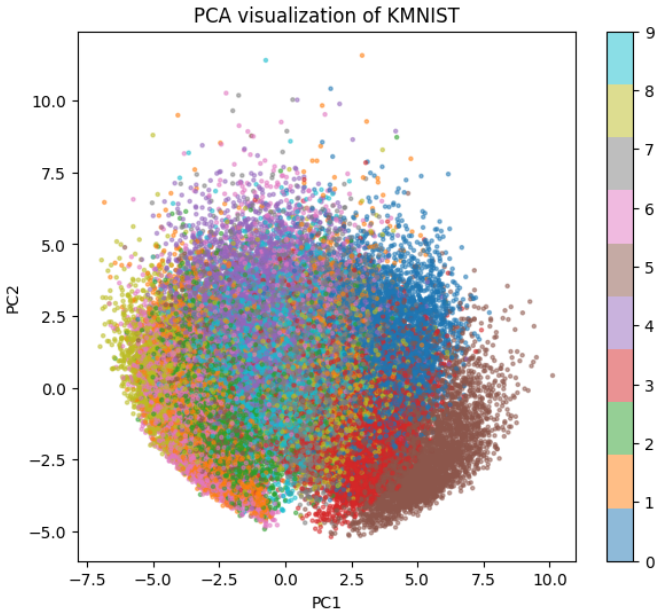


Fig. 2: PCA projection of the KMNIST training set onto the first two components. Each point is a training image, colored by its class label.

TABLE I: Test performance of all models on KMNIST.

Model	Acc.	Prec.	Rec.	F1
Logistic Regression	0.598	0.609	0.598	0.600
Simple CNN	0.934	0.935	0.934	0.934
ResNet-like CNN	0.943	0.943	0.943	0.943
Improved CNN	0.967	0.968	0.967	0.967
TunedImprovedCNN	0.970	0.970	0.970	0.970

B. Baseline vs CNN Performance

Table I summarizes the test performance of all models. The logistic regression baseline reaches about 60% test accuracy. The Simple CNN already improves this to about 93%, the ResNet-like CNN to about 94%, and the deeper Improved CNN to about 96.7%. After tuning the hyperparameters of the Improved CNN, the final model reaches about 97.0% accuracy.

The confusion matrix for logistic regression is shown in Fig. 3. The diagonal entries are visible but fairly small, and there are many off-diagonal cells with high counts. The model confuses many character pairs, which is expected because it can only draw linear decision boundaries in the 784-dimensional space. Combined with the overlapping clusters in Fig. 2, this explains why performance saturates around 60%.

Figure 4 compares confusion matrices for the Simple CNN and the Improved CNN. For the Simple CNN, the diagonal is much stronger than in the logistic regression matrix, but there are still noticeable confusions, especially among visually similar classes. The Improved CNN further sharpens the diagonal, and most off-diagonal counts become small. This suggests that adding extra depth, residual-style blocks, batch normalization, and dropout helps the model learn more robust features.

The ResNet-like CNN performs slightly better than the

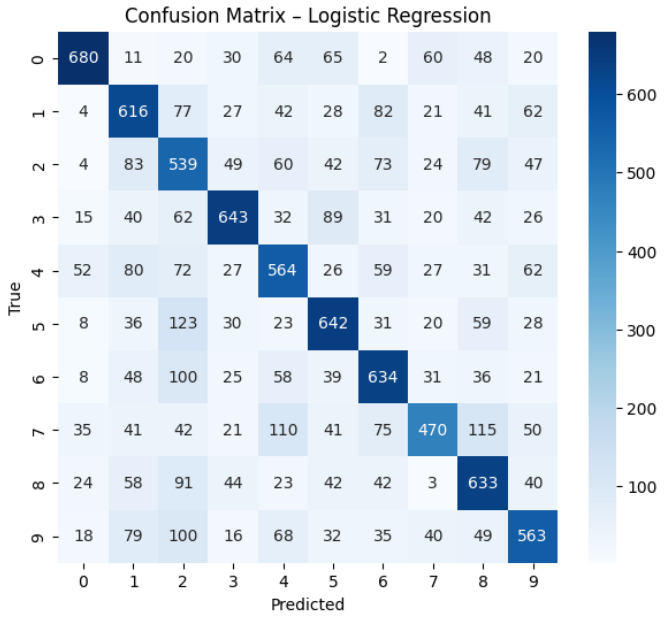


Fig. 3: Confusion matrix for the logistic regression baseline on the KMNIST test set. Numbers are counts of test images per true/predicted class pair.

Simple CNN but slightly worse than the Improved CNN. It probably cannot take full advantage of residual connections because the network is still relatively shallow and does not use batch normalization. Overall, the main jump in performance comes from moving from a linear model to even a small CNN, and then there is a smaller but still useful gain from increasing depth and regularization.

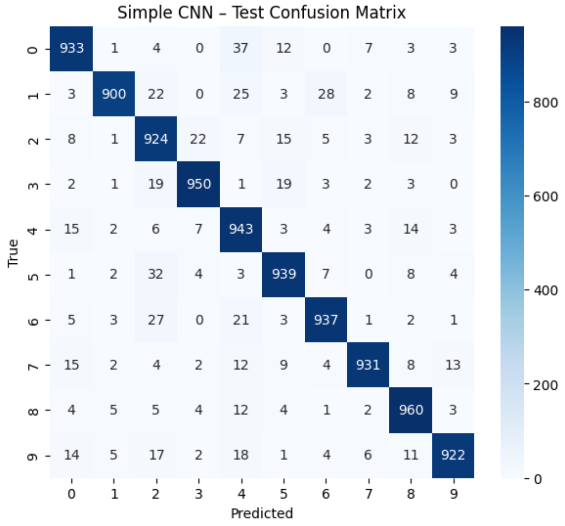
C. Tuned CNN and Error Analysis

The TunedImprovedCNN uses the hyperparameters selected from the validation search (learning rate 0.001, dropout rate 0.4, batch size 128). Figure 5 shows confusion matrices for both the training and test sets. On the training data, the model is almost perfectly accurate with nearly all mass on the diagonal. On the test set, the diagonal remains strong and off-diagonal counts are small, giving about 97% accuracy and a modest train–test gap that indicates some overfitting but generally good generalization.

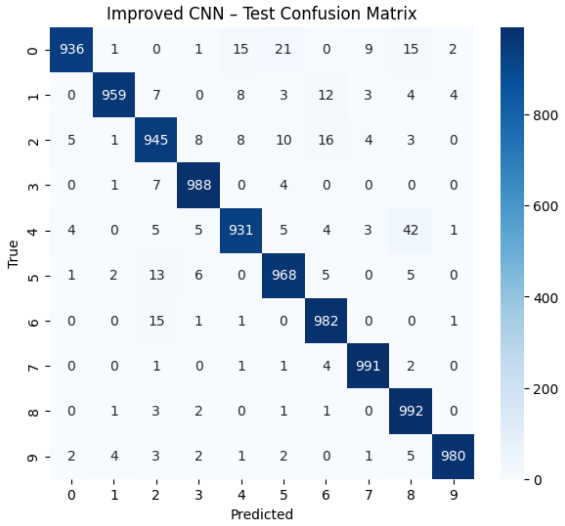
To better understand the remaining errors, we inspect misclassified test samples from the TunedImprovedCNN (Fig. 6). Many mistakes occur on characters with faint or broken strokes, or on visually ambiguous characters whose local stroke patterns resemble another class. Even at 97% accuracy, the model is therefore most challenged by low-quality or intrinsically similar characters, rather than by systematic failures on specific classes.

D. Interpretability

1) Model-Level Analysis: To understand how the TunedImprovedCNN makes decisions, we generate gradient-based saliency maps [11]. For a test image x with predicted class $\hat{y}$, we backpropagate the logit $f_{\hat{y}}(x)$ to the input and visualize



(a) Simple CNN



(b) Improved CNN

Fig. 4: Confusion matrices for two CNN models on the KMNIST test set. The deeper Improved CNN produces a sharper diagonal and fewer misclassifications.

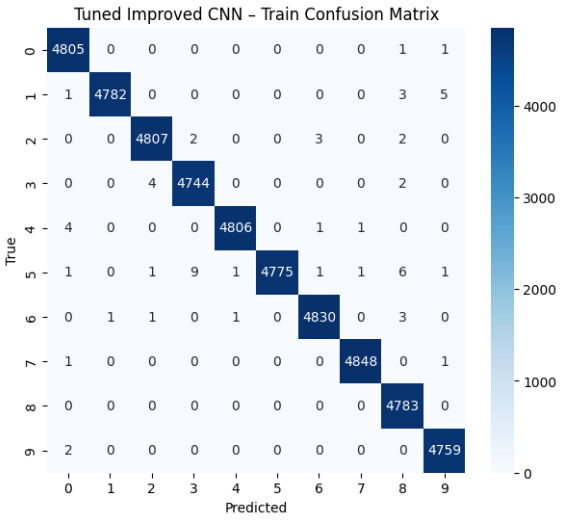
the maximum absolute gradient as a heatmap. Bright regions correspond to pixels that most influence the prediction.

As shown in Fig. 7, the model consistently highlights the main character strokes while largely ignoring the background, indicating that its decisions rely on meaningful structure. In harder cases, salient regions still follow plausible stroke shapes, but the characters may be faint or missing small strokes—explaining many of the remaining misclassifications [12].

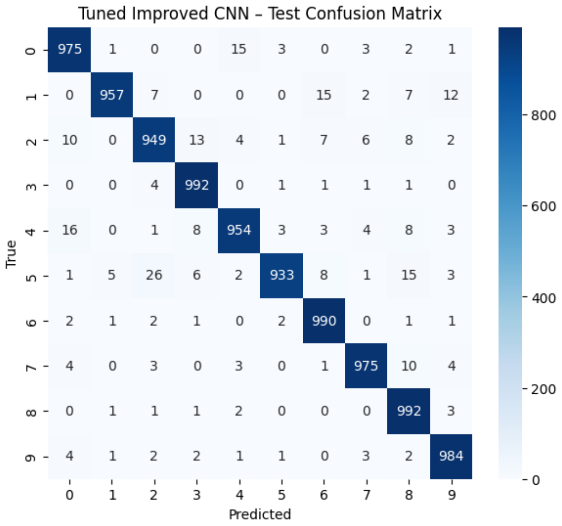
.

2) Business Insights: Handwritten Kuzushiji recognition is relevant in applications such as large-scale digitization of historical records, document archiving, and automated data extraction pipelines. The results in this project illustrate how model choice affects the feasibility of such systems.

A linear model such as logistic regression achieves only



(a) Train



(b) Test

Fig. 5: Confusion matrices for the TunedImprovedCNN on the training and test sets.

about 60% accuracy on KMNIST. In real deployments this would translate into a significant amount of manual correction and would be insufficient for automated document processing. In contrast, the TunedImprovedCNN reaches approximately 97% accuracy, reducing the number of errors by an order of magnitude. At this level of performance, most characters can be transcribed automatically, and human review can be focused on a small set of ambiguous cases.

A practical workflow could therefore combine automatic predictions with confidence-based routing: high-confidence outputs are accepted immediately, while samples from classes that commonly confuse the model, or predictions with low confidence, are flagged for manual verification. Such a hybrid approach reduces the overall annotation burden, speeds up throughput, and yields consistent transcription quality. These findings highlight the value of deploying a well-tuned deep model in real-world handwritten character recognition systems.

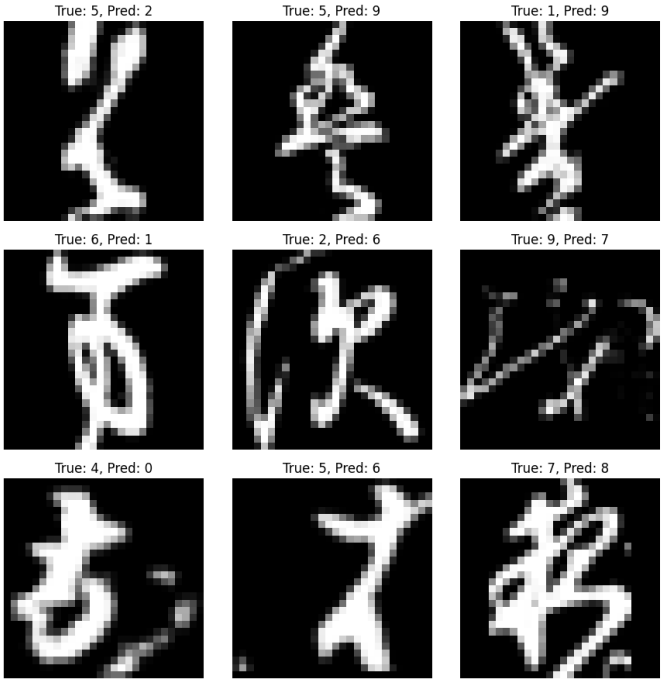


Fig. 6: Example misclassified test samples for the TunedImprovedCNN. The title above each image shows the true and predicted class.

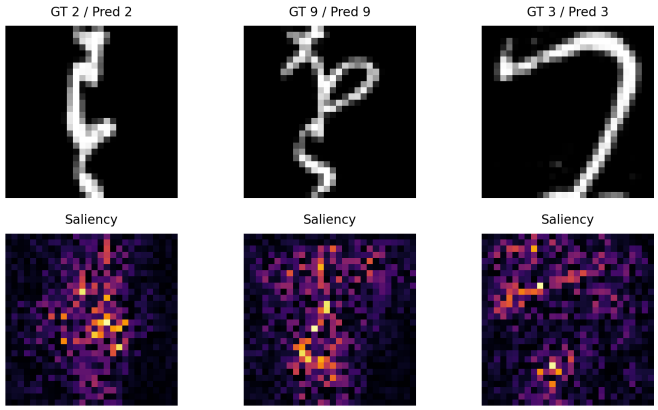


Fig. 7: Saliency maps for three KMNIST test images. The top row shows original inputs; the bottom row shows pixels most influential to the predicted class.

IV. Research Extension: Siamese Metric Learning for One-Shot Classification

To satisfy the Extra Credit Option 2 (research extension), we study a supervised metric-learning extension on top of our KMNIST classifiers. Instead of directly training another multi-class classifier, we learn a similarity function on handwritten characters and then reuse this learned metric in a 10-way one-shot recognition setting [13], in contrast to the fully supervised convolutional networks in the main experiments. The goal is to examine how much recognition performance can be recovered when only one labeled example per class is available at test

time, in contrast to the fully supervised convolutional networks in the main experiments.

Following the setup of Koch et al. [14], we use a Siamese convolutional architecture with two weight-sharing branches that each map a 28×28 grayscale image into a low-dimensional embedding vector [15]. Given a pair of images (x_i, x_j), the two embeddings are passed through a small comparison module that produces a scalar similarity score $g_\theta(x_i, x_j)$, where θ denotes all network parameters. This score is converted into a probability

$$\hat{p}_{ij} = \sigma\big(g_\theta(x_i, x_j)\big),$$

that the pair comes from the same class, and we supervise the model with the standard binary cross-entropy loss [16]

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(i,j)} \Big[y_{ij} \, \log \hat{p}_{ij} + \big(1 - y_{ij}\big) \, \log\big(1 - \hat{p}_{ij}\big) \Big]. \quad (1)$$

Here $y_{ij} \in \{0, 1\}$ is the same or different label for pair (x_i, x_j), with $y_{ij} = 1$ for pairs drawn from the same KMNIST class and $y_{ij} = 0$ otherwise, $\hat{p}_{ij}$ is the predicted probability that the pair is from the same class, $\sigma(\cdot)$ is the logistic sigmoid, N is the number of training pairs, and g_θ is implemented by a shared convolutional encoder followed by a linear classifier. During training we construct a balanced mixture of positive and negative pairs from the KMNIST training split and minimize $\mathcal{L}(\theta)$ with respect to θ. Class labels are used only to form the supervision pairs, and no additional annotation or modification of the dataset is required.

After training, we freeze the encoder and treat it as a learned metric for one-shot recognition. For each of the ten KMNIST classes, we randomly select a single support image, embed it together with all test images, and classify each test sample by the nearest support in Euclidean distance in the embedding space. On the full $10{,}000$-image test set, this nearest-neighbor one-shot classifier achieves an accuracy of 0.7387, with macro-averaged precision 0.8036, recall 0.7387, and F1-score 0.7338. We summarize these numbers alongside our logistic-regression and CNN baselines in a comparison table. Although the one-shot accuracy is lower than that of the best supervised CNN trained on all $60{,}000$ labeled training examples, it substantially improves over the logistic-regression baseline (accuracy 0.598) and demonstrates that a Siamese metric-learning encoder can support reasonably strong one-shot recognition on KMNIST under extreme label scarcity. The complete training and evaluation code for this extension is included in our project repository as part of the submitted materials.

V. Conclusion

This project compared several models for KMNIST character classification, from a multinomial logistic regression baseline to a tuned convolutional network. Logistic regression on flattened 28×28 images reached about 60% accuracy, while a small two-layer CNN exceeded 93% and an improved CNN with residual blocks, batch normalization, and dropout reached roughly 96.7%. A short grid search over learning rate, dropout, and batch size further increased test accuracy to about 97%, with confusion matrices and misclassifications indicating that remaining errors are concentrated on visually ambiguous characters. Overall, the results underline the importance of CNN-based representation learning and simple validation-driven tuning for this dataset; promising extensions include data augmentation [17], deeper residual architectures, and additional regularization techniques.

All code, trained models, and notebooks are available at our **GitHub Repository**, and a summary with additional visualizations is provided in the **Project Blog Post**.

Acknowledgment

The authors thank Dr. Peeples for guidance and feedback throughout the project and for providing the course framework for this work.

References

[1] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[4] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *arXiv preprint arXiv:1812.01718*, 2018.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[10] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed., ser. Springer Series in Statistics. New York, NY: Springer, 2002.

[11] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *Workshop at International Conference on Learning Representations (ICLR)*, 2014.

[12] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[13] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[14] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2, no. 1. Lille, 2015, pp. 1–30.

[15] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 3630–3638.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.