OCRing Javanese Characters with CNN

Aditya W. Mahastama 1, a), Lucia D. Krisnawati 1, b), Nathania Saphira 1, c) and Su-Cheng Haw 2, d)

1Universitas Kristen Duta Wacana, Jl. Dr. Wahidin Sudirohusodo 5-25, Yogyakarta, Indonesia

2*Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia*

b) Corresponding author: krisna@staff.ukdw.ac.id  
a)mahas@staff.ukdw.ac.id

*c)nathania.saphira@ti.ukdw.ac.id*

*d)sucheng@mmu.edu.my*

**Abstract.** The Javanese script, a traditional form of writing and part of Indonesia's cultural heritage, is now rarely used but continues to be preserved both physically and digitally. This study aims to support digital preservation of printed books written in Javanese script by developing an Optical Character Recognition (OCR) model using a Convolutional Neural Network (CNN) based on the GoogLeNet architecture. The model was trained on segmented characters acquired from scanned Javanese manuscripts, using character classes with at least ten samples – comprising a total of 11.920 characters. The combination of various parameters with and without normalization were experimented and tested. The performance evaluation -- performed using accuracy, precision, recall, and F1-scores -- showed that the best performance—though all metrics remained below 70%—was achieved with a batch size of 32, 100 epochs, and no Batch Normalization. Despite modest results, CNN proves reasonably effective for recognizing Javanese script, particularly when using weighted-averaged metrics due to class imbalance.

# introduction

Optical Character Recognition (OCR) is a field of study that converts digital images of text into editable documents that can be processed, edited, searched, saved, or copied using a computer application [1]. The OCR outputs have been used in various subfields of studies, such as for identifying text in images to recognize car license plates [2], branding names [3], or simply for extracting texts from images [4]. With the object of historical documents, OCR outputs are very beneficial for Information Retrieval and knowledge extraction [5]. It was applied also to digitize city manuals or to transcribe automatically diplomatic historical texts [6], or to preserve the classic literature written in Javanese script [1]. This explains why research in Optical Character Recognition (OCR) has advanced considerably, with recognition rates surpassing 98% accuracy in many domain applications. Consequently, OCR is often regarded as a solved problem [7]. This characterization holds true primarily for modern printed documents utilizing contemporary font styles. However, the performance of OCR systems remains significantly limited when applied to historical documents, particularly those written in Asian writing systems. In these contexts, accuracy rates fall short of expectations, indicating a continued need for methodological advancements and domain-specific adaptations. Unfortunately, contemporary OCR methods are often not adapted to the historical domain [5] which presents a range of additional challenges. These include poor scan quality, inconsistent use of typefaces [8], as well as physical degradation of the original materials – such as bleed-trough, mildew, ink-blotches – often caused by prolonged exposure to humidity and aging paper conditions [9]. A specific challenge for OCRing Javanese characters is set on its writing system, where some dependent characters will be written on the top or under an independent character. So line (horizontal) segmentation still plays a big role here.

This research aimed to enhance the performance of our OCR engine specifically designed for the digitization of printed historical texts written in Javanese script. Our previous studies addressed the problem of OCRing the historical documents on the physical degradation of the paper [9], data annotation and augmentation [10], horizontal and vertical segmentations [11], improving recognition using several machine learning models [1], as well as traditional statistical model [12]. Although our previous studies have addressed various challenges within the fundamental components of OCR system, enhancing the recognition accuracy of our OCR engine remains a critical objective. To this end, we explore the application of deep learning techniques, specifically Convolutional Neural Networks (CNNs), based on the assumption that such models can significantly improve the overall recognition performance of our system.

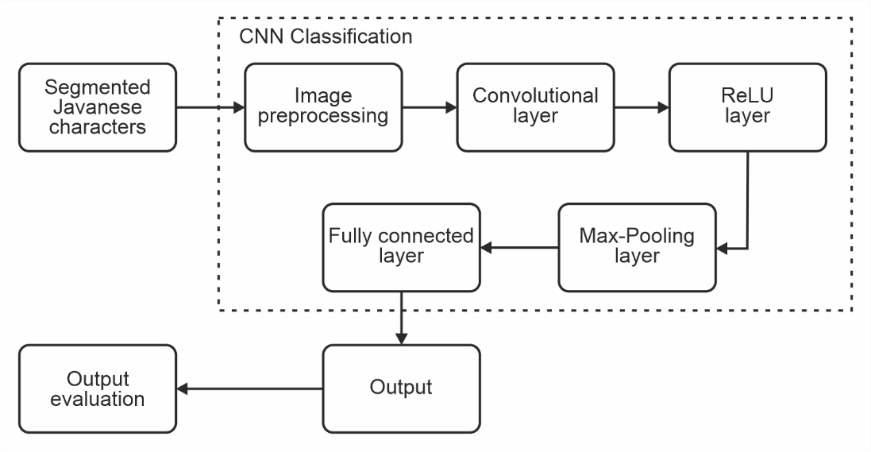
# related works

Javanese is recognized as one of the world’s classical languages [13], boasting a literary tradition that spans over a millennium. Unfortunately, the Javanese script is no longer in widespread use; it is now primarily found in educational materials and on street signs in major cities of Central Java. This decline underscores the importance of familiarizing the scripts as well as preserving classical literary works originally written in the Javanese script. Previous studies on Javanese character recognition have primarily focused on familiarizing users with the script itself, often limiting their scope to the recognition task alone as found in [14, 15, 16]. Only a limited number of studies have focused on the preservation of historical documents written in Javanese script [1, 12], or in Balinese script [17]. The historical documents take forms of Javanese books printed between 19th to 20th centuries [1, 9, 10], as well as Balinese handwritten Lontar manuscripts [17], which are traditionally inscribed on dried palm leaves. Their datasets predominantly consist of isolated Javanese characters, lacking contextual usage [14, 15, 18]. In some cases, limited context is provided in the form of words or short phrases [16], while only a few studies incorporate extended and complex contexts, such as full-length books [1, 12], or Lontar manuscripts [17]. Some of these studies concentrate exclusively on the recognition of handwritten Javanese characters [18, 19], while others utilize a combination of handwritten and digitally generated Javanese script [15, 20], or focus solely on printed characters [1, 14].

Our study focuses on enhancing the recognition accuracy of our previously developed OCR engines [1, 12] by utilizing the same dataset, which comprises scanned copies of *Serat Mangkunegaran IV* Books I and II—originally printed in the early 1900s—and educational textbooks published in the 1970s. Given the complexity of the textual context, this study faces several challenges, including the large number of unique Javanese characters—reaching up to 161 distinct classes—compared to the significantly smaller character sets used in prior studies, which range from 20 [14, 19, 20], to approximately 100 classes [16]. We assumed that Convolutional Neural Network with GoogleNet architecture will increase the performance of our OCR engine.

# research method

The research method applied in general follows the flow described in Figure 1. The input takes form of segmented Javanese characters undergoing image preprocessing, classification process, and the evaluation.



**Figure 1.** Research method

## Data Preparation

The dataset used for training was sourced from our previous works [1, 12], which contains 11,920 segmented and annotated images. These images are grouped by book page and stored in .PNG format, with filenames containing details like book source code, page number, line number, and character number for easy access and processing. Each image's metadata is stored in a corresponding .CSV file, which is combined with the image data during preprocessing. A new test dataset comprising 100 images was also prepared, following the same segmentation and annotation methods, with expert validation. The dataset labels, stored in the "training.csv" file, are used to identify transliterations and segment statuses, with only valid, non-damaged classes considered for training the model. The example of the results of preprocessing and the metadata of our dataset is presented in Figure 2.

The preprocessing workflow involves extracting labels from the training data using the getLabelName() function, filtering out damaged or redundant labels. The image data for each class is then collected, and the labels are converted into binary arrays to indicate the presence of a specific label. Image resizing is performed based on the largest image dimensions, ensuring consistency for the CNN model input. The data is processed into a format suitable for the model, where images are resized to the largest dimension (335x335) to avoid loss of script details. The image and label data are separated and augmented using ImageDataGenerator. The dataset is then split into training (75%) and validation (25%) sets using the train\_test\_split() function to ensure reproducibility.

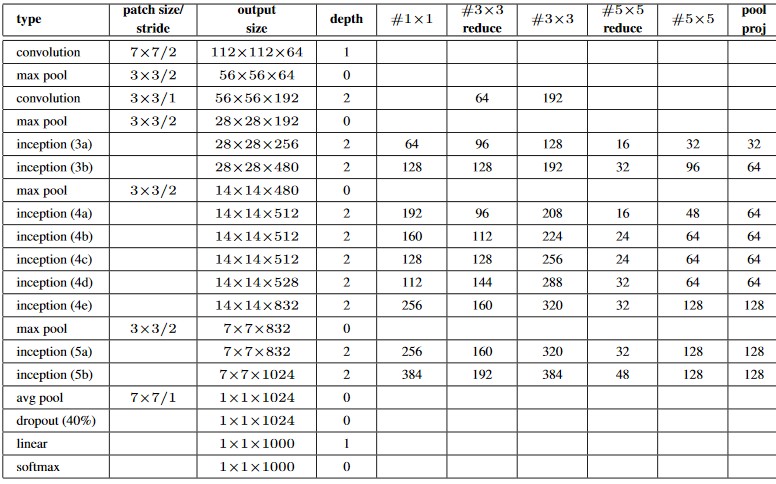
|  |  |
| --- | --- |
| (a) | (b) |

**Figure 2.** Examples of the dataset: (a) segmented images and (b) features and labels

## GoogLeNet Implementation

GoogLeNet utilizes an Inception network, which allows the model to dynamically select the most suitable filter size for each layer. To address the high computational cost of testing multiple filter types, GoogLeNet employs 1x1 convolutions to reduce filter dimensionality and computation. Unlike other architectures that use Fully Connected Layers, GoogLeNet incorporates Global Average Pooling (GAP) to minimize computational cost by averaging 7x7 features into a 1x1 output, reducing the number of parameters. The GoogLeNet architecture is implemented through a function that defines its layers, following the arrangement depicted in Figure 3, and applied prior to model training.

The inception layer in GoogLeNet consists of multiple sub-layers, including Conv and Concatenate layers, which require custom functions for implementation. This layer accepts tensor and filter parameters, which are manually set and may vary according to the architecture in Figure 3. The inception layer is divided into four paths: (1) 1x1 Conv, (2) 1x1 Conv followed by a 3x3 Conv, (3) 1x1 Conv followed by a 5x5 Conv, and (4) 3x3 Max-Pool followed by 1x1 Conv. The outputs of these paths are concatenated into a single tensor.

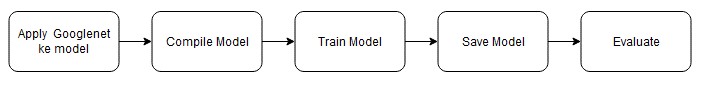


**Figure 3.** A description on layers in GoogLeNet architecture that was applied in this research

These inception layers, along with Conv and MaxPool layers, are organized into the GoogLeNet() function. The architecture begins with two Conv and MaxPool layers, followed by several inception layers separated by MaxPool layers, and concludes with a Fully Connected layer. The function accepts parameters for the number of classes and input image size (as specified by the img\_size parameter in preprocessing), outputting a GoogLeNet model ready for training.

## Model Training and Validation

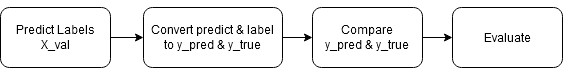
During training, the GoogLeNet functions are applied to the model and reconfirmed using the Keras library's model.summary() function. The model is then configured for evaluation metrics with the model.compile() function, utilizing the "adam" optimizer and categorical cross-entropy loss. Evaluation metrics include categorical accuracy, precision, recall, and F1-score, with the latter sourced from the TensorFlow-Addons library. Given the categorical nature of the training, the metrics are suited for categorical data, using Keras’ built-in functions for all but the F1-score. The model training flow is illustrated in Figure 4.



**Figure 4.** Process flow for model training

After setting the epoch and batch size, the model is trained using the model.fit() function for 100 epochs and a batch size of 32, with the training process taking approximately two days. Once trained, the model is saved as a .h5 file for future use, such as in the model and API testing stage. The evaluation results are visualized using Matplotlib functions and subsequently analyzed.

The trained model is validated using 25% of the training dataset (X\_val and y\_val), with predictions made using the Keras model.predict() function. The output is a list of class predictions for each data point, with each entry representing the model's probability for each of the 161 classes. The class with the highest probability is selected as the predicted class. These predictions (y\_pred) are compared to the true labels (y\_val) after converting them to class labels. The results are then evaluated using the classification\_report function, which calculates accuracy, precision, recall, and F1-score for the validation. The process of the validation is illustrated in Figure 5.



**Figure 5.** Process flow for model validation

## Model Testing

To optimize testing results, an additional dataset comprising 100 characters, which were annotated and validated by Javanese expert, was used. Like the training dataset, the test dataset consists of .png image files and a corresponding .csv file containing detailed information. The image filenames follow the same format as the training dataset.

The testing process closely mirrors the validation stage, with the primary difference being set on the preprocessing that took the same procedure as the training process. The model evaluation is performed using scikit learn's *classification\_report()* function, providing Macro-Averaged Precision (MAP), Recall (MAR), F1-score (MAF), and Weighted-Averaged Precision (WAP), Recall (WAR), and F1-score (WAF). After preprocessing, the model is loaded from the .h5 file, and predictions are made using the same procedure as the validation stage, as outlined in Figure 6.

# Results and Discussion

The model was trained on a dataset comprising 9,191 samples across 161 classes. During the training, the system outputs epoch-wise performance metrics including loss, categorial\_accuracy, precision, recall, and F1-score, as defined during the model compilation. For validation and testing, prediction performance is assessed using the classification\_report library, which compares predicted labels with true labels. This report provides MAP, MAR, MAF as well as WAP, WAR, and WAF scores. Accuracy is reported using the macro-average only.

This study conducted a series of experiments to determine the most effective training strategy. Initially, batch sizes of 16 and 32 were tested over 50 epochs, yielding accuracy rates of 6% and 56%, respectively. Given its superior performance, batch size 32 was further evaluated with 100 epochs. The results of the validation process in these various scheme is presented in Figure 6.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

**Figure 6.** (a) Accuracy rates during validation process, (b) results of model validation using macro-averaged metrics, (c) results of model validation using weighted-averaged metrics, (d) Validation results with and without batch normalization.

Following a series of validation experiments, the parameter combination that demonstrated the highest performance was identified: batch size 32, 100 epochs, and no batch normalization. This configuration was subsequently used in the testing phase. The model was evaluated on a test set of 100 character images, producing a single set of performance metrics, as presented in Table 1.

**TABLE 1.** Testing results with batch size 32, epoch 100, and without batch normalization

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Accuracy** | **Macro-Averaged** | | | **Weighted-Average** | | |
| **Precision** | **Recall** | **F1-Score** | **Precision** | **Recall** | **F1-Score** |
| 62% | 53% | 49% | 56% | 71% | 62% | 64% |

## Effects of Batch Normalization, Batch Size and Epoch Size

Batch normalization is a method used for speeding up training process. This study applied 2 batch normalization layers, one was implemented after the inception and another before the activation layer. During validation process, the implementation of batch normalization proved to increase the processing time by 9 minutes for each epoch but dropped the recognition accuracy to 4% in comparison to 94% accuracy rate for architecture without batch normalization. This showed that batch normalization is unsuitable when it is applied to CNN using GoogleNet architecture during training process. We hypothesize that the observed decline in accuracy in the batch-normalized model may stem from the evaluation procedure, wherein the same image was used for testing at each epoch. This consistent reuse of identical input data could have hindered the model’s capacity to generalize, indicating a potential overfitting issue and a lack of adaptability to novel, unseen samples. Therefore, this study trained and tested the final model without batch normalization.

Besides batch normalization, this study experimented with different batch sizes and epochs to improve model accuracy. Using 50 epochs, batch size 16 gave only 6% accuracy, while batch size 32 improved accuracy to 56%. With batch size 32, increasing epochs to 100 further boosted accuracy to 94%. Due to this significant improvement and to avoid overtraining, the author selected the model with batch size 32 and 100 epochs as the final model to be tested.

Although the model performed well during validation, it showed mediocre performance on a test set of 100 images, with a macro-averaged accuracy of 62%, MAP 53%, MAR 49% and MAF of 56% as shown in Table 1. However, weighted-average metrics performed better with 71% WAP, 62% WAR, and 64% WAF. From 11,920 data points, only 9,191 were usable for training, with class sample sizes of 10 to 585. The analysis results showed that classes with greater number of samples had better performance, while classes with few samples had poor, even zero scores, indicating the model struggles with underrepresented classes.

# CONCLUSION

The research concludes that using a Convolutional Neural Network with GoogleNet architecture, a batch size of 32, and 100 epochs (without Batch Normalization) yielded the best validation results (86% F1-score). While the test results were lower (49% F1-score), the approach is effective for recognizing Javanese scripts from scanned manuscripts, provided there is a large, evenly distributed dataset for training.

Our hypothesis that CNN with GoogLeNet architecture would enhance the previous OCR system is only partially validated. Comparing results from [1, 12], where the k-NN model achieved 70% accuracy and 74% WAP, the CNN model performed slightly lower, with a maximum accuracy of 62% (below k-NN and LDA) and a WAP of 71% (comparable to LDA). Thus, with 9,191 samples across 161 classes, CNN performance remains similar to traditional methods like k-NN and LDA [21]. To improve performance, increasing the training data and addressing class imbalance are necessary, which we consider for future work.

# REFERENCES

1. L. D. Krisnawati and A. W. Mahastama, “Building classifier models for on-off Javanese character recognition,” in Proceedings of the 21st International Conference on Information Integration and Web-Based Applications & Services (Association for Computing Machinery, Munich, Germany, 2019), pp. 25–34.
2. M. L. S. N. S. Lakshmi, P. J. Christydass, and S. Cheerla, “License plate detection using optical character recognition,” International Journal of Applied Engineering Research **5**, 103–111 (2020).
3. A. S. Timmaraju and V. Khanna, Detecting and Recognizing Text in Natural Images Using Convolutional Networks (Stanford University, 2015), pp. 1–7.
4. O. O. Patience, E. M. Amaechi, O. George, and O. N. Isaac, “Enhanced text recognition in images using Tesseract OCR within the Laravel framework,” Asian Journal of Research in Computer Science **17**, 58–69 (2024).
5. J. Martínek, L. Lenc, and P. Král, “Building an efficient OCR system for historical documents with little training data,” Neural Computing and Applications **32**, 17209–17227 (2020).
6. D. Fleischhacker, W. Goederle, and R. Kern, “Improving OCR quality in 19th century historical documents using a combined machine learning based approach,” (2024).
7. K. Kettunen and M. Koistinen, “Open source Tesseract in re-OCR of Finnish Fraktur from 19th and early 20th century newspapers and journals: Collected notes on quality improvement,” DHNB Publications **2**, 270–282 (2019).
8. A. Poncelas, M. Aboomar, J. Buts, J. Hadley, and A. Way, “A tool for facilitating OCR postediting in historical documents,” (2020).
9. A. W. Mahastama and L. D. Krisnawati, “Histogram peak ratio-based binarization for historical document image,” in 2017 International Conference on Smart Cities, Automation & Intelligent Computing Systems (IEEE, Yogyakarta, 2017), pp. 93–98.
10. L. D. Krisnawati and A. W. Mahastama, “JavAnote: Tool and strategies to annotate Javanese characters,” International Journal of Computer Trends and Technology **69**, 93–99 (2021).
11. A. W. Mahastama and L. D. Krisnawati, “Improving projection profile for segmenting characters from Javanese manuscripts,” in Proceedings of the 1st International Conference on Intermedia Arts and Creative Technology (SCITEPRESS – Science and Technology Publications, Yogyakarta, Indonesia, 2019), pp. 77–82.
12. A. W. Mahastama and L. D. Krisnawati, “Optical character recognition for printed Javanese script using projection profile segmentation and nearest centroid classifier,” in 2020 Asia Conference on Computers and Communications (IEEE, Singapore, 2020), pp. 52–56.
13. L. D. Krisnawati and A. W. Mahastama, “The scenario of Wikipedia usage as a supporting source for under-resourced language researches,” Prisma.Com **40**, 34–44 (2019).
14. M. F. Naufal, J. Siswantoro, and J. T. Soebroto, “Transliterating Javanese script images to Roman script using convolutional neural network with transfer learning,” International Journal of Informatics Visualization **8**, 1460 (2024).
15. D. U. K. Putri, D. N. Pratomo, and A. Azhari, “Hybrid convolutional neural networks-support vector machine classifier with dropout for Javanese character recognition,” TELKOMNIKA **21**, 346 (2023).
16. A. Susanto, I. U. W. Mulyono, C. A. Sari, E. H. Rachmawanto, and R. R. Ali, “Javanese character recognition based on K-nearest neighbor and linear binary pattern features,” KINETIK (2022).
17. N. Suciati, N. P. Sutramiani, and D. Siahaan, “LONTAR\_DETC: Dense and high variance Balinese character detection method in Lontar manuscripts,” IEEE Access **10**, 14600–14609 (2022).
18. C. K. Dewa, A. L. Fadhilah, and A. Afiahayati, “Convolutional neural networks for handwritten Javanese character recognition,” Indonesian Journal of Computing and Cybernetics Systems **12**, 83 (2018).
19. F. T. Anggraeny, Y. V. Via, and R. Mumpuni, “Image preprocessing analysis in handwritten Javanese character recognition,” Bulletin of Electrical Engineering and Informatics **12**, 860–867 (2023).
20. G. A. Robby, A. Tandra, I. Susanto, J. Harefa, and A. Chowanda, “Implementation of optical character recognition using Tesseract with the Javanese script target in Android application,” Procedia Computer Science **157**, 499–505 (2019).
21. S.C. Joseph, G. S., K. Suriyan, and S.P. K., “Performance Benchmarking: Pre-trained Models and Custom Convolutional Neural Networks in Deep Learning,” Journal of Informatics and Web Engineering **4**(2), 225–235 (2025).