Indonesian Journal of Artificial Intelligence and Data Mining (IJAIDM)
Vol 6, No.2, September 2023, pp. 243 – 251
p-ISSN: 2614-3372 | e-ISSN: 2614-6150

Javanese Script Letter Detection Using Faster R-CNN

[1]Muhammad Helmy Faishal, [2]Mahmud Dwi Sulistiyo, [3]Aditya Firman Ihsan
[1,2,3]School of Computing, Telkom University, Bandung, Indonesia
Email: [1]helmyfaishal@student.telkomuniversity.ac.id, [2]mahmuddwis@telkomuniversity.ac.id,
[3]adityaihsan@telkomuniversity.ac.id

Article Info
Article history:
Received Jul 11th, 2023
Revised Aug 1st, 2023
Accepted Aug 30th, 2023

Keyword:
CNN
Detection
Faster R-CNN
Javanese Script
Letter

ABSTRACT
The Javanese script is now rarely used, and some people no longer recognize it. Constructing a Javanese script recognition system based on digital image processing is one of its preservation efforts. This study proposes a model that can detect Javanese script using Faster R-CNN to help people who are not familiar with Javanese script. Object detection is chosen because it is not only able to detect the object, but also able to get the position of the object, and it can predict multiple objects simultaneously. Faster R-CNN was chosen for its higher accuracy and ability to detect small objects. Although Faster R-CNN performs well in text detection, its use in Javanese script detection is still unexplored, making its performance in this area unknown. This study aims to investigate Faster R-CNN's effectiveness in Javanese script detection. In this study, Faster R-CNN was able to show good performance by obtaining mean average precision (mAP) values up to 0.8381, accuracy up to 96.31%, precision up to 96.53%, recall up to 96.38 %, and F1-Score up to 96.41%. These results indicate that Faster R-CNN can detect Javanese script letters well.

Corresponding Author:
Mahmud Dwi Sulistiyo,
School of Computing, Telkom University, Bandung
Jl. Telekomunikasi No. 1, Terusan Buah Batu Bandung 40257, Jawa Barat, Indonesia
Email: mahmuddwis@telkomuniversity.ac.id
DOI: http://dx.doi.org/10.24014/ijaidm.v6i2.24641

1. INTRODUCTION

At present, Javanese script is not widely utilized and some individuals are not even familiar with it [1]. This could lead to the gradual disappearance of the Javanese script until it eventually becomes extinct. To contribute to the preservation of the Javanese script, this study proposes a model that can detect the letters of the Javanese script. An object detection approach is used in this study to identify objects and their localization. Moreover, this approach can simultaneously recognize multiple objects. The detection and recognition of letters are not a new thing; there is research that has been done using different methods, such as in the research [2], which uses the Projection Profile Segmentation and Nearest Centroid Classifier; research [3], which uses Nested Multi-Layer Perceptron network with Modified Direction Feature; and research [4], which uses Convolution Neural Network (CNN).

However, additional processes, such as using segmentation [2], [3] or response map [4], are still required to obtain the position of the object. To overcome this issue, this study uses one state-of-the-art object detection method, Faster R-CNN [5]. Faster R-CNN eliminates the additional process required to obtain the object's position. Faster R-CNN was selected for its high accuracy and ability to detect small objects [6], [7]. Moreover, the feature extractor or backbone in Faster R-CNN, based on the CNN architecture, can be modified or customized as desired, as in the research [8]–[11], which uses a different CNN architecture. Faster R-CNN has shown good performance in letter or text detection, as in the research [8] with kanji handwriting objects, research [9] with alphanumeric handwriting objects, and research [10] with scene text objects from the ICDAR 2013 and 2015 datasets.

Although it shows good performance in research [8]–[11], until the time of this study, the implementation of Faster R-CNN in detecting Javanese script letters has not been found, so it is not yet known how its ability to detect Javanese script letters. So, this study aims to implement Faster R-CNN using different

CNN architectures, then analyze and compare the performance of the Faster R-CNN model in detecting Javanese script letters on image-based data.

In short, the main contributions of this paper are as follows:

1. Proposing Faster R-CNN-based modeling for Javanese script letter detection;
2. Construct a dataset with multiple Javanese script letters in each single image;
3. Showing experimental results on the Faster R-CNN model using various backbones; and
4. Present the analysis results of Faster R-CNN's reliability in detecting Javanese script letters.

Additionally, this paper explains the dataset, the proposed model, and the experimental setup in section 2. The results and analysis of the experiments are discussed in section 3, while section 4 presents the conclusions.

2. RESEARCH METHOD

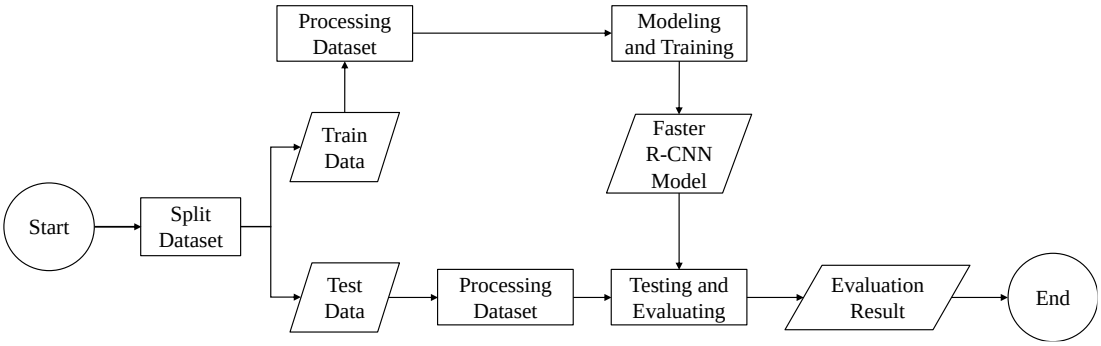

Figure 1. Overview of the research stages.

As seen in **Figure 1**, the stages in this study were divided into three stages, namely the Dataset Preparation Stage, Modeling and Training Stage, and Evaluation Stage, which will be explained in more detail in sections 2.4 to 2.6.

2.1. Dataset

This study uses a dataset that was generated using a dataset that contains images of Javanese script. This study uses only the basic Javanese script known as the Carakan or Nglegena script, which has 20 classes. Each class has 55 images, with 53 handwritten images and 2 digital font images. When generating the dataset, it is possible that the Javanese script image is resized and/or rotated. For resizing, it is done by 32 pixels to 144 pixels. Then, for rotation, it is up to 45 degrees. For illustration, the dataset construction can be seen in **Figure 2**. The dataset has a size of 576×576 pixels with 250 images.

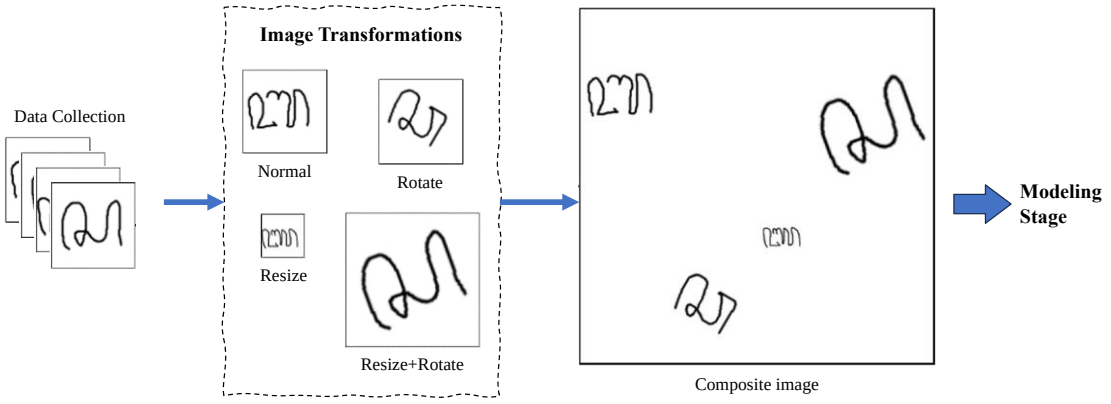

Figure 2. Illustration of the dataset construction; The rightmost part is the resulting composite image.

2.2. Faster R-CNN

Various methods can be used in text detection or object detection in general; One of these methods is Faster R-CNN [5]. Faster R-CNN is one of the regional-based CNN architectures used in object detection, which is developed from Fast R-CNN by introducing a Region Proposal Network (RPN) to replace the

Selective Search mechanism [5]. The structure of the Faster R-CNN is divided into four parts [9], [11] namely feature extractor, RPN, RoI pooling, and fully connected as shown in **Figure 3** with the following explanation:

a. Feature extractor
   In this part, feature extraction is carried out in the input image using CNN, which produces a feature map [9]. This section is also known as the backbone.
b. Region Proposal Network (RPN)
   In this part, RPN generates a proposal region contained in the input image, which determines the anchors in the background and foreground using SoftMax. Then get an accurate proposal with a bounding box regression [9], [11].
c. Region of Interest pooling (RoI pooling)
   In this part, RoI pooling performs pooling so that the scale and size ratio is fixed for all RoI obtained from the proposal region and feature map [9].
d. Fully-connected
   In this part, classify objects and determine the position of objects using bounding box regression on feature maps and candidate region proposals that have been processed by the RoI Pooling [9].

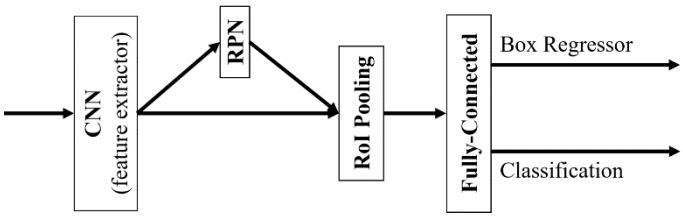

Figure 3. The architecture of the Faster R-CNN [5].

2.3. CNN Architecture

As mentioned in section 2.2, Faster R-CNN uses CNN. Convolutional Neural Network (CNN) is an evolution of artificial neural networks widely used in computer vision. CNN uses convolution to extract features from digital images. CNN architecture has been developed a lot and has its uniqueness; For example, SqueezeNet has a special structure that allows reducing the computation but still maintaining the level of accuracy [12], and EfficientNet can perform and balance scaling on several factors uniformly [13], ResNet has a structure that allows stacking layers to a great depth [14], and VGG has a deep convolution layer [15]. The CNN architecture of [12]–[15] can be seen in Figure 5 and the special blocks can be seen in Figure 4.

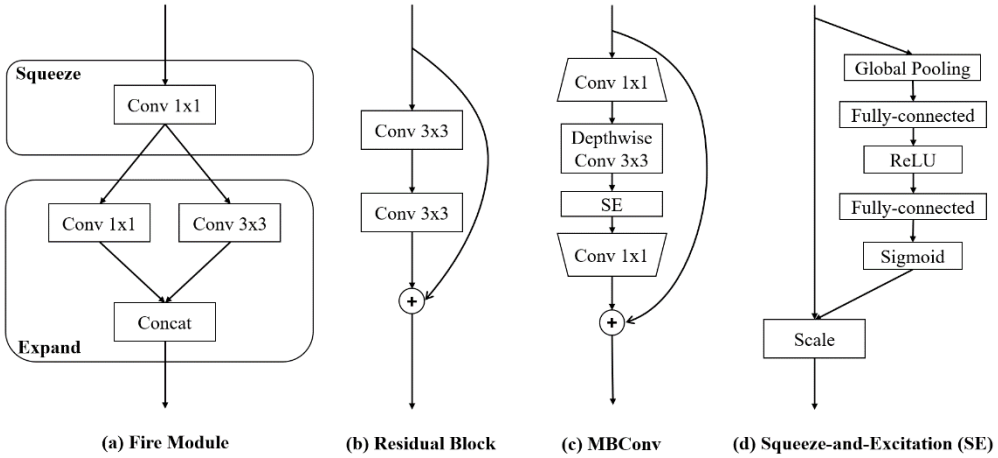

Figure 4. Structure of the special block options: (a) Fire Module used by SqueezeNet [12], (b) Residual Block used by ResNet [14], (c) MBConv used by EfficientNet [13], and (d) Squeeze-and-Excitation [16]

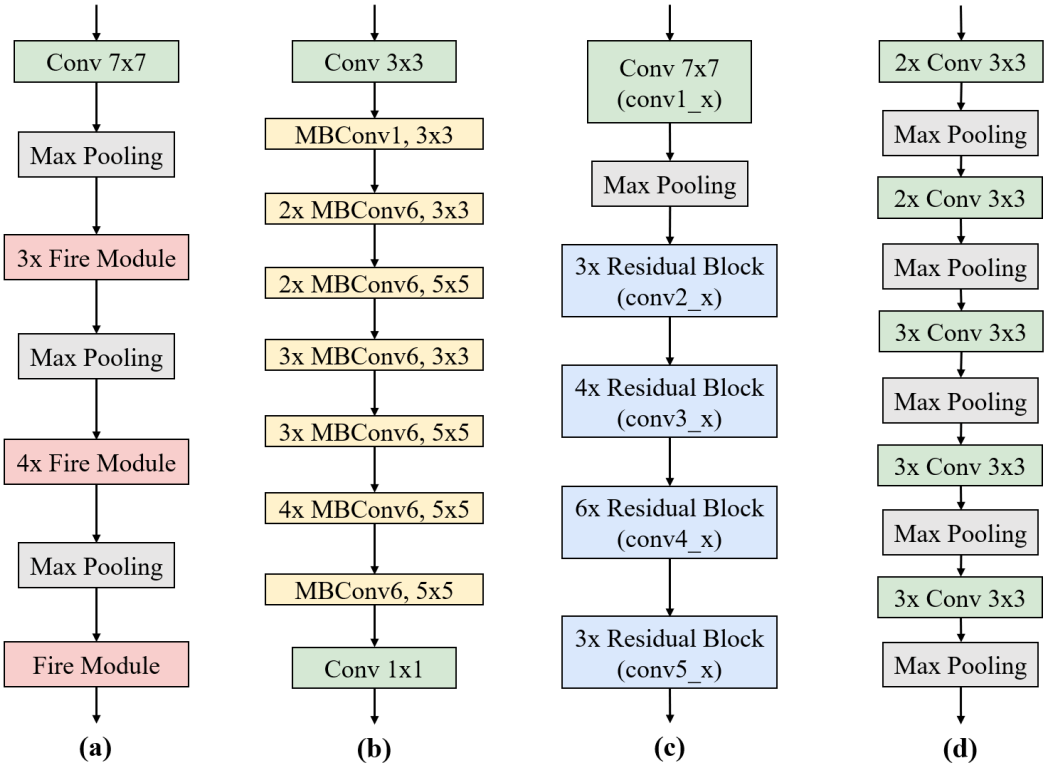

Figure 5. CNN architectures compared in this study: (a) SqueezeNet [12], (b) EfficientNet-B0 [13], (c) ResNet-34 [14], and (d) VGG-16 [15].

2.4. Dataset Preparation

In this stage, there are two steps. Firstly, the image dataset needs to be divided randomly into a training dataset and a test dataset. The training dataset is used to train the model, while the test dataset is used to evaluate the model. The total number of images in the training dataset is 200, which makes up 80% of the total dataset. On the other hand, the test dataset consists of 50 images, which makes up 20% of the total dataset. **Table 1** shows the distribution of Javanese script letters in both the training and test datasets, with an average of 12 letters per image in the training dataset and 11 letters per image in the test dataset. Additionally, the dataset loading process is carried out for each training and test dataset by converting the form into data that the model can process, while also including information about the image from the annotation data. Once finished, the training and test datasets can be used for either model training or model evaluation.

Table 1. Distribution of Javanese script letters in the dataset used in this study.

Dataset	Letters																				Total
	Ha	Na	Ca	Ra	Ka	Da	Ta	Sa	Wa	La	Pa	Dha	Ja	Ya	Nya	Ma	Ga	Ba	The	Nga	
Training	129	128	145	111	120	123	111	127	136	121	113	127	127	128	122	134	129	132	118	117	**2,498**
Testing	28	32	35	27	27	32	36	29	30	30	33	24	25	34	35	27	23	28	32	29	**596**

2.5. Modeling and Training

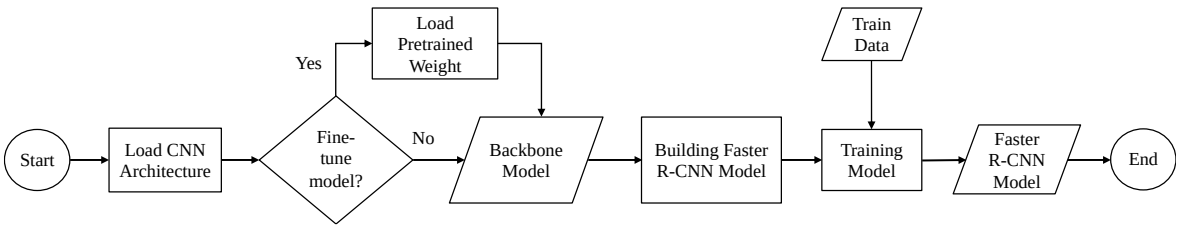

Figure 6. Flowchart of the model training.

At this stage, there are several stages, as shown in **Figure 6**. The first stage is loading the CNN model, which will be used as the Faster R-CNN backbone, As mentioned in section 2.3, this study uses four CNN architectures, namely SqueezeNet [12], EfficientNet-B0 [13], ResNet -34 [14], and VGG-16 [15]. For each CNN architecture, there are two versions of the CNN model, namely the Plain model and the Fine-tune model, with the following description:

a. **Plain model** is a model that has never been trained at all.
b. **Fine-tune model** is a model that uses a pre-trained model or a model that has been trained which is then adjusted so that the model can train new data.

The fine-tuning model uses pre-trained weights obtained from training on ImageNet [17]. The purpose of using a fine-tuned model is to improve model performance because the dataset used in this study is small [18]. After loading the CNN model, the next step is building the Faster R-CNN model. After the Faster R-CNN model is built, the model is trained using the prepared training data to make a trained model.

2.6. Testing and Evaluation

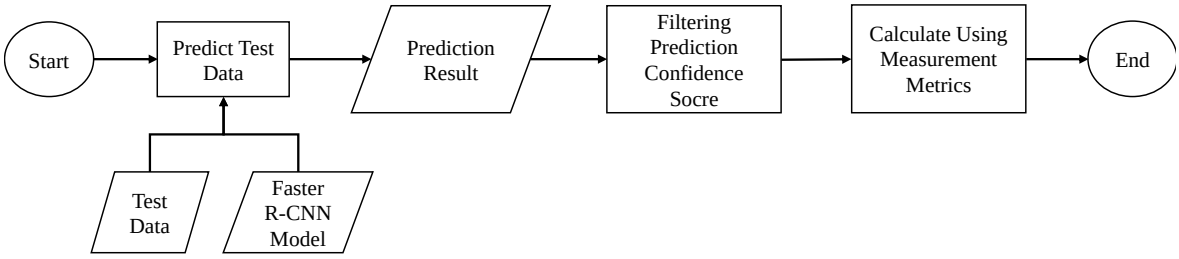

Figure 7. Flowchart of the model testing and evaluation.

At this stage, there are three stages carried out as shown in **Figure 7**. In the first stage, the model predicted the test data that was trained. The next step selected predictions with confidence greater than the threshold (0.5 in this study) using a confidence score-based filtering process. Finally, the last stage determined the model's performance by calculating the measurement metrics using the filtered prediction results from the previous stage.

2.7. Measurement Metrics

As explained in section 2.6, measurement metrics are needed to determine the performance of a model. This study uses measurement metrics including accuracy, precision, recall, F1-Score, and mean average precision (mAP).

In the mAP metric, the main threshold value used in this study is a threshold of 0.5 to 0.95 with a step of 0.05 which will be referred to as mAP@[0.5:0.95], this threshold is also used in measurement metrics on the Microsoft COCO dataset [19]. In addition, using other IoU thresholds such as a threshold of 0.5 which will be referred to as mAP@0.5, and a threshold of 0.75 which will be referred to as mAP@0.75.

2.8. Experimental Setup

In this study, the model was implemented using the Python programming language version 3.8.5 using the help of several Python libraries, one of these libraries is PyTorch [20] which was used to build the model used in this study. The models that have been built are trained using a computer with specifications, namely an Intel® Core™ i7-8750H processor, Nvidia GeForce GTX 1050 4GB GPU, and 16GB RAM.

The models were trained using the same optimizer as the Faster R-CNN research [5], namely the SGD optimizer with a learning rate of 0.001; momentum of 0.9; and weight decay of 0.0005 [5]. The models were trained for 100 epochs.

3. RESULTS AND ANALYSIS

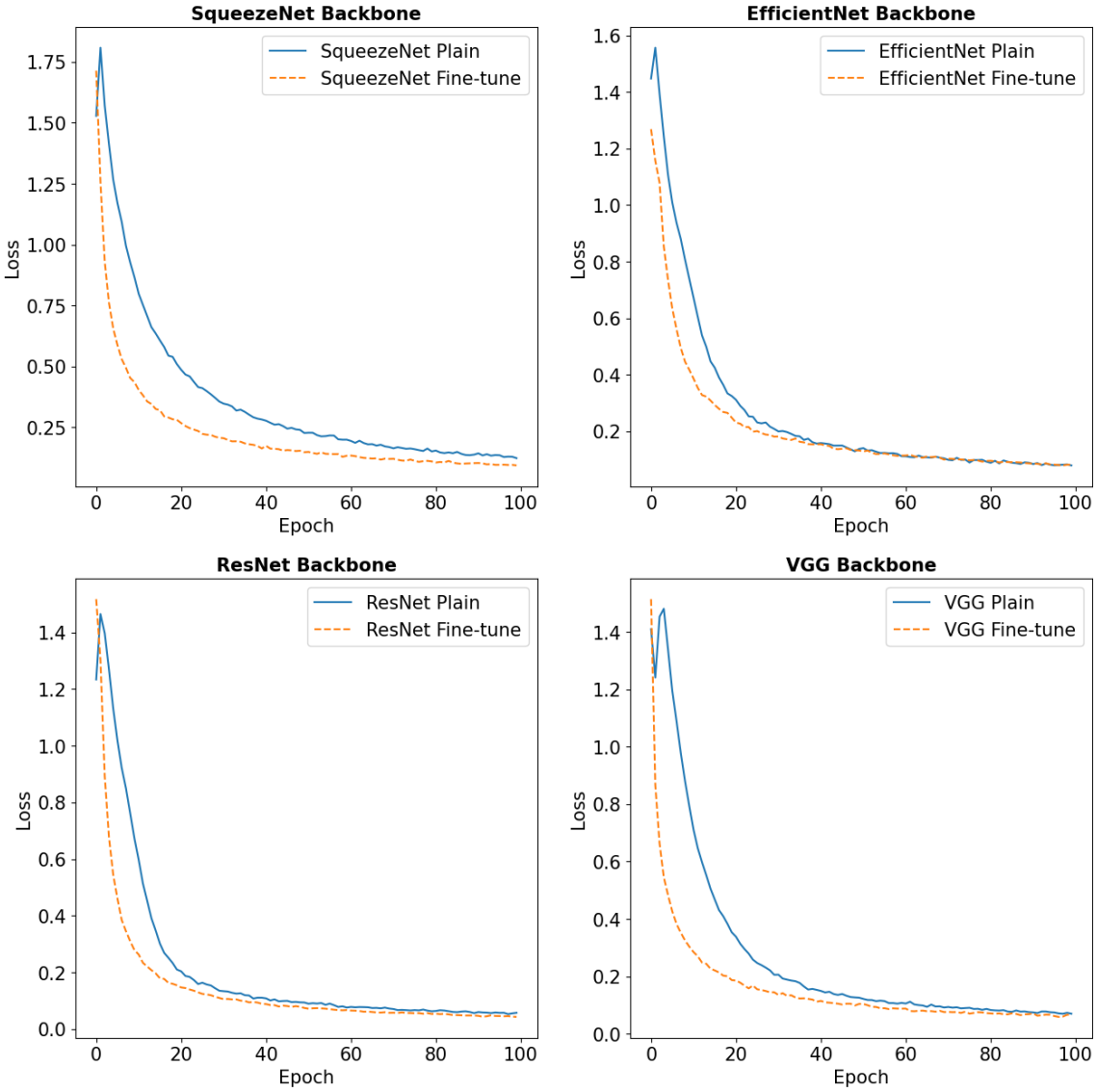

Figure 8. The average loss performance of each backbone model during the training stage.

Table 2. Comparison of measurement metric values across models.

Backbone Model		mAP@0.5	mAP@0.75	mAP@[0.5:0.95]	Accuracy	Precision	Recall	F1-Score
SqueezeNet	Plain	0.9330	0.9217	0.8061	0.9111	0.9203	0.9138	0.9157
	Fine-tune	0.9632	0.9563	0.8348	0.9631	0.9653	0.9638	0.9641
EfficientNet	Plain	0.6955	0.6862	0.6021	0.7097	0.7782	0.7190	0.7257
	Fine-tune	0.9241	0.9050	0.7841	0.9245	0.9441	0.9272	0.9338
ResNet	Plain	0.7471	0.7344	0.6294	0.7483	0.8012	0.7523	0.7699
	Fine-tune	0.9446	0.9369	0.8360	0.9413	0.9600	0.9423	0.9501
VGG	Plain	0.8655	0.8573	0.7470	0.8574	0.8836	0.8600	0.8696
	Fine-tune	0.9586	0.9560	0.8381	0.9547	0.9652	0.9557	0.9596

Based on **Figure 8** and **Table 2**, there is a faster decrease in the loss value in the fine-tuned model compared to the plain model. Not only that, but the fine-tuned model also performs better on all measurement metrics. This can happen because the fine-tuned model uses the weight of the pre-trained model as the initial training weight of the model so that the model can get faster loss reduction and better performance than the plain model. As shown in [18], this study also shows that the fine-tuning model can improve performance on a model with a small dataset that only has 250 images with 3,094 objects compared to ImageNet which has around 1.2 million images [17] and on MS COCO which has around 330 thousand images [19].

Table 3. Complexity and attribute of the backbone model.

Backbone Model	Parameter	Depth	Special Block
SqueezeNet [12]	~1.25 M	1+8	Fire module
EfficientNet-B0 [13]	~5.29 M	1+16+1	MBConv
ResNet-34 [14]	~21.8 M	1+16	Residual Block
VGG-16 [15]	~138.36 M	13	-

In addition, as seen in **Table 2** and **Table 3**, even though SqueezeNet is a lighter model compared to other models, it can get better performance. Based on this, we assume that lighter models can get better performance on small datasets, as described in [18], which explains that choosing a CNN architecture that fits the dataset can affect the performance of a model.

Unlike SqueezeNet, EfficientNet has lower performance compared to other models. This led us to assume that the dataset used in our study, which consisted of simple images of Javanese letters on a white background as shown in **Figure 2**, may not be suitable for the complex special blocks used in the construction of EfficientNet as shown in Figure 4. We assume that the performance of a model can be influenced by the characteristics of the dataset and that models with simpler architectures may perform better on datasets with simple characteristics.

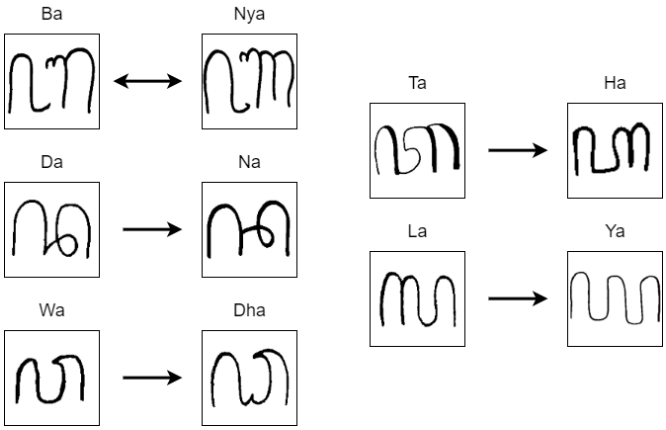

Figure 9. Some Javanese script pairs have considerably high similarities one each other.

Even though it has good results, there are still several letters that have the lowest accuracy in each model. These letters are the letters "Ta", "La", "Ba", "Da", "Nya", and "Wa". As seen in **Figure 9**, Some letters in the Javanese script share similarities, which can result in inaccurate classification. This is because the model makes prediction errors between these similar letters and other letters. As a result, some letters have lower accuracy compared to others.

Although Faster R-CNN has the advantage of better object detection accuracy, it has the disadvantage of slow speed [6]. In this study, the speed needed to predict each object for the SqueezeNet model is about 0.083 seconds or 12 frames per second (FPS), for EfficientNet about 0.166 seconds or 6 FPS, for ResNet about 0.125 or 8 FPS, and for VGG about 0.333 seconds or 3 FPS.

An example of the output of the predictions in this study can be seen in **Figure 10** with the green color as the correct prediction, the red color as the wrong prediction, and letters without boxes are letters that failed to predict. In the SqueezeNet model, the plain model has three prediction errors, but the fine-tuned model only has one prediction error. In the EfficientNet model, the plain model has four prediction errors and one prediction failure, but the fine-tuned model only has one prediction error. In the ResNet model, the plain model has three prediction errors, but the fine-tuned model has no prediction errors. In the VGG model, the plain model has three prediction errors, but the fine-tuned model has no prediction errors. Based on these results, it shows that the fine-tuned model has better results by having fewer errors and prediction failures than the plain model.

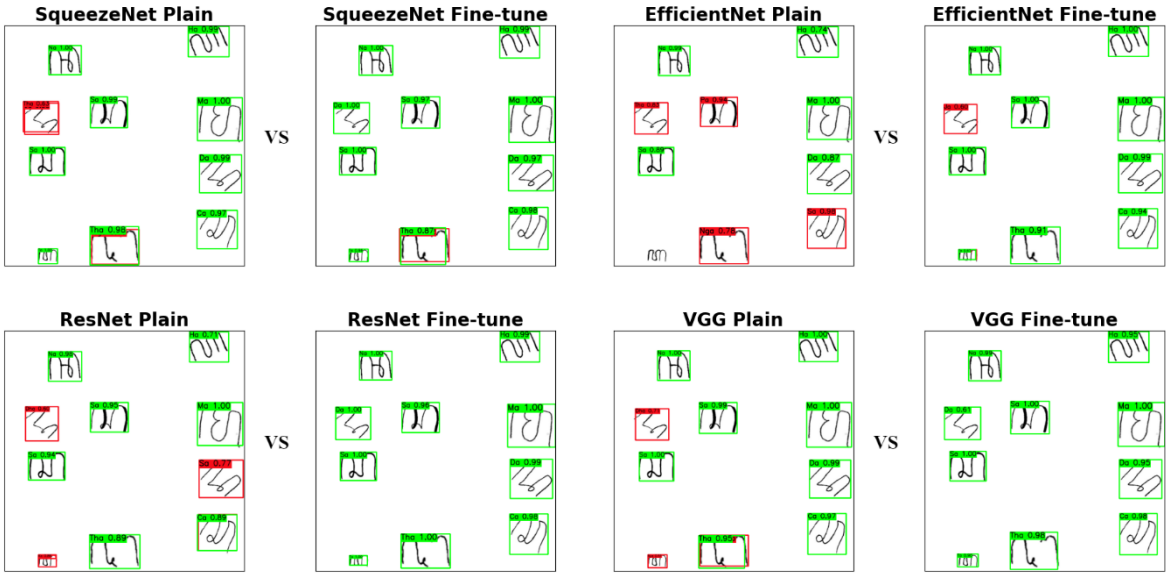

Figure 10. Some examples of script detection outputs from each model.

4. CONCLUSION

Based on the results of the evaluation and analysis of the model, it can be concluded that the Faster R-CNN can perform good detection and classification of Javanese script letters which are capable of obtaining mAP@[0.5:0.95] to 0.8381; mAP value@0.5 to 0.9632; mAP value@0.75 to 0.9563; precision up to 0.9653 or 96.53%; recall up to 0.9638 or 96.38%; and accuracy up to 0.9631 or 96.31%. In addition, it can be concluded that the fine-tuned model can provide increased performance by having better performance and results compared to the plain model. Even though they were able to get good results, there were still letters that had a low accuracy of 41.67%. This can still be improved by increasing the number of letter variations which in this study only had 55 letter variations.

There are some limitations to this study, specifically the use of a simple image dataset. Future studies should consider using more complex datasets, such as images or photos with Javanese script letters found in manuscripts or street signboards. In addition, despite having good accuracy, Faster R-CNN has a slow speed. To address this, further research could explore models with faster speed, such as the one-stage detector model.

REFERENCES

[1] "Dinas Kebudayaan (Kundha Kabudayan) Daerah Istimewa Yogyakarta." [Online]. Available: https://budaya.jogjaprov.go.id/berita/detail/753-aksara-jawa-menolak-punah. [Accessed: 06-Nov-2022].

[2] A. W. Mahastama and L. D. Krisnawati, "Optical character recognition for printed javanese script using projection profile segmentation and nearest centroid classifier," *2020 Asia Conf. Comput. Commun. ACCC 2020*, pp. 52–56, 2020, doi: 10.1109/ACCC51160.2020.9347895.

[3] M. D. Sulistiyo, D. Saepudin, and Adiwijaya, "Optical character recognition using modified direction feature and nested multi layer perceptrons network," *Proceeding - 2012 IEEE Int. Conf. Comput. Intell. Cybern. Cybern. 2012*, pp. 30–34, 2012, doi: 10.1109/CyberneticsCom.2012.6381611.

[4] M. L. Afakh, A. Risnumawan, M. E. Anggraeni, M. N. Tamara, and E. S. Ningrum, "Aksara jawa text detection in scene images using convolutional neural network," *Proc. - Int. Electron. Symp. Knowl. Creat. Intell. Comput. IES-KCIC 2017*, vol. 2017-Janua, pp. 77–82, 2017, doi: 10.1109/KCIC.2017.8228567.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[6] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 3296–3305, 2017, doi: 10.1109/CVPR.2017.351.

[7] L. Tan, T. Huangfu, and L. Wu, "Comparison of YOLO v3, faster R-CNN, and SSD for real-time pill identification," *arXiv*. Research Square, 2021, doi: 10.21203/rs.3.rs-668895/v1.

[8] A. Adole, E. Edirisinghe, B. Li, and C. Bearchell, "Investigation of Faster-RCNN Inception Resnet V2 on Offline Kanji Handwriting Characters," *ACM Int. Conf. Proceeding Ser.*, 2020, doi: 10.1145/3415048.3416104.

[9] J. Yang, P. Ren, and X. Kong, "Handwriting Text Recognition Based on Faster R-CNN," *Proc. - 2019 Chinese Autom. Congr. CAC 2019*, pp. 2450–2454, 2019, doi: 10.1109/CAC48633.2019.8997382.

[10] Q. Xie, K. Zhou, and X. Fan, "A scene text detection algorithm based on ResNet and faster R-CNN," *ACM Int. Conf. Proceeding Ser.*, pp. 826–829, 2019, doi: 10.1145/3349341.3349521.

[11] B. Liu, W. Zhao, and Q. Sun, "Study of object detection based on Faster R-CNN," *Proc. - 2017 Chinese Autom. Congr. CAC 2017*, vol. 2017-Janua, pp. 6233–6236, 2017, doi: 10.1109/CAC.2017.8243900.
[12] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," pp. 1–13, 2016.
[13] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *36th Int. Conf. Mach. Learn. ICML 2019*, vol. 2019-June, pp. 10691–10700, 2019.
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.
[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.
[16] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-Excitation Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, 2020, doi: 10.1109/TPAMI.2019.2913372.
[17] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," no. May 2014, pp. 248–255, 2010, doi: 10.1109/cvpr.2009.5206848.
[18] M. Shu, "Deep learning for image classification on very small datasets using transfer learning," *Creat. Components*, pp. 14–21, 2019.
[19] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014, doi: 10.1007/978-3-319-10602-1_48.
[20] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, no. NeurIPS, 2019.

BIOGRAPHY OF AUTHORS

Muhammad Helmy Faishal is a student of School of Computing at Telkom University, Bandung, Indonesia. He is an aspiring AI enthusiast. He gained hands-on experience as a Programmer at PT Telkom Indonesia, actively contributed to the RPLGDC Lab in 2020, and demonstrated his coding prowess in the Shopee Code League 2020. Through courses with Dicoding Indonesia and the Sanbercode Bootcamp, he mastered diverse skills, ranging from machine learning to web development and data science. Helmy's fervent dedication to artificial intelligence and technology leads him to be a promising contributor in this rapidly advancing field.

Mahmud Dwi Sulistiyo received his Bachelor and Master's degrees in Informatics Engineering of Telkom Institute of Technology, Indonesia, in 2010 and 2012, respectively. In 2017 he was awarded the Indonesia's LPDP Scholarship and started a doctoral program in Department of Intelligent Systems at Graduate School of Informatics, Nagoya University, Japan, with a research focusing on Attribute-aware Semantic Segmentation. He received his Doctor of Informatics in 2021. Since 2013, Mahmud has been a Lecturer at School of Computing of Telkom University, where he is currently the Artificial Intelligence Laboratory Supervisor. His research topics are related to artificial intelligence, machine learning, and computer vision. His main activities include lecturing, conducting research and community services. He has received numerous funding grants, both from internal of Telkom University sources, as well as from external sources, including the Ministry of Research, Technology, and Higher Education and the Indonesia Endowment Fund for Education (LPDP) under the Ministry of Finance. Currently, he becomes the members of IEEE and ACM.

Aditya Firman Ihsan obtained degrees of bachelor and master in mathematics department of Institut Teknologi Bandung (ITB) consecutively in 2016 and 2017. He was actively involved in many research projects in the Research Consortium OPPINET (Optimization of Pipeline Network) under Mathematical Modelling and Simulation Co- Laboratory in ITB from 2017 until present. He also teaches mathematics in the School of Computing in Telkom University from 2020. His main research interests are mathematical modelling, dynamical systems, and deep learning.