JSCE (Journal of System and Computer Engineering)
Vol.7, No.4, October 2026, pp. 498-509
ISSN (online): 2723-1240
DOI : https://doi.org/10.61628/jsce.v7i4.2904

Research Article Open Access (CC–BY-SA) ■

Javanese Script Classification with YOLO26 and Convolutional Filter Visualization

Harry Mardika [1,a,*]; Ericks Rachmat Swedia [2,b]
[1,2] Informatics, Faculty of Industrial Technology, Universitas Gunadarma, Jl. Margonda Raya No. 100, Depok 16424, Indonesia
[a] harrymardika@student.gunadarma.ac.id; [b] ericks_rs@staff.gunadarma.ac.id
** Corresponding author*

Abstract

Javanese script is a cultural heritage whose use is narrowing, while automatic recognition is held back by visual similarity between characters. This study evaluates the YOLO26 architecture on Javanese script classification, records the change in convolutional kernel weights across the eleven architecture blocks, and tracks prediction probabilities on visually similar characters. Secondary data of 2,659 Nglegena images from 20 classes came from Kaggle. Each image passed through Otsu's binarization, diacritic-safe bounding, resizing to 224x224 pixels and letterboxing, with augmentation on the training partition. The nano variant was trained in PyTorch with AdamW and stopped at the tenth epoch through early stopping. Testing on 25 images gave 96% accuracy with a macro average F1-score of 0.9733, and 18 of the 20 classes were recognised without error; the single error fell on the blurriest test image. Under the same data, split and metrics, the model stayed 20 points above a Random Forest over HOG features and matched a ResNet-18 to within one image while holding 7.2 times fewer parameters. Blocks 1 to 3 recorded edges and curves at larger weight values, blocks 4 to 9 produced denser representations with weights closer to zero, and block 11 converted them into class scores.

Keywords—*Javanese script, YOLO26, image classification, convolutional kernel weights, softmax probability*

1. Introduction

Javanese script, or Hanacaraka, is one of Indonesia's traditional writing systems. Its characters contain many repeated curved elements and a high degree of shape similarity between characters (Robby et al., 2019), which makes them harder to tell apart than Latin letters (Susanto et al., 2023a). Everyday use of the script has narrowed as the Latin alphabet has taken over, which places its digitisation alongside other preservation work.

Character recognition systems have developed as visual data has grown more complex. From the early to the middle of the 2010s, optical character recognition (OCR) kept localisation and feature extraction as separate, rigid steps (Robby et al., 2019), which handled standard printed text but lost accuracy on handwriting with wide shape variation. Towards the end of that decade the approach shifted to layered architectures, from standard convolutional neural networks (CNN) to models such as DenseNet, ResNet and VGG (Susanto et al., 2024; Sudewo et al., 2024b).

Several studies have applied these approaches to Javanese script. Random Forest combined with a Histogram of Oriented Gradients (HOG) reached 96.29% accuracy, while the highest accuracy in the same study, 97.77%, came from the configuration without HOG (Rasyidi et al., 2021). Random Forest combined with zoning reached 58.33%, which indicates that zoning-

based feature extraction is less effective for Nglegena characters (Graciello et al., 2025). Later work moved to CNNs that take up edges, textures and character shapes directly (Susanto et al., 2023b), and deeper architectures such as ResNet, DenseNet, GoogLeNet and VGG reached validation accuracy above 98% (Susanto et al., 2024); ResNet-18 has been applied to Hanacaraka character prediction (Sudewo et al., 2024a).

Single-stage detectors such as You Only Look Once (YOLO) followed (Redmon et al., 2016), locating and classifying objects in a single pass (Khanam & Hussain, 2024; Sapkota et al., 2026). A recent study applied YOLO11 to Javanese script detection and obtained 81.00% accuracy with a macro-averaged F1-score of 86.41%, with performance dropping on characters of ambiguous shape: 'ha' reached 68.09%, 'ya', 'ja' and 'ra' stayed below 80%, and every sample of the background class was misclassified (Darmawan & Ariatmanto, 2025).

Accuracy has risen across these studies, but they report final metrics without describing how a model arrives at the shape of a character, so a high accuracy figure says little about the process behind it. The question is sharper for YOLO, where a deeper convolutional stack makes the internal steps harder to follow.

This study applies the YOLO26 classification variant to Javanese script. Progressive Loss Balancing (ProgLoss) and Small-Target-Aware Label Assignment (STAL), reported for YOLO26, operate on the detection head and are not active in the classification variant used here, which retains the convolutional backbone and a linear classification head (Sapkota et al., 2026). The convolutional stack is the part this study observes. The aim here is not accuracy alone. The study also records the weight matrix of each convolutional layer during training, so that the extraction of the curved patterns of Javanese script can be followed from raw pixels to the final class decision, and tracks prediction probabilities across epochs with attention to classes whose shapes are close. The work has three objectives: to build and evaluate a Javanese script classifier based on YOLO26, to examine the change in convolutional kernel weights across the architecture blocks during training, and to observe the prediction probability dynamics for visually similar classes.

2. **Method**

2.1 Dataset

The study uses secondary data from the Kaggle repository phiard/aksara-jawa (Phiard, 2019). The object of study is Aksara Jawa Nglegena, the set of basic Javanese characters. The dataset covers 20 character classes: ba, ca, da, dha, ga, ha, ja, ka, la, ma, na, nga, nya, pa, ra, sa, ta, tha, wa and ya, giving 2,659 images in total. The source already separates the data into training, validation and test partitions, as shown in Table 1.

Table 1. Dataset Size and Partitioning.

Partition	Images	Per-class distribution
Training	2,154	108 per class; ba 114; ha and nga 102
Validation	480	24 per class
Test	25	1 per class; ma 3; ba, nga and ra 2
Total	**2,659**	**133 per class, from 127 (ha) to 140 (ba)**

Class sizes are close to balanced. Fifteen classes hold 108 training, 24 validation and 1 test image, giving 133 images per class; the remainder differ slightly with the amount of source data available, ba holding the largest training partition at 114 images and ha and nga the smallest at 102 each. The split gives 2,154 training samples, 480 validation samples used to monitor performance at each epoch, and 25 test samples used for the final evaluation.

2.2 Preprocessing and Augmentation

Raw images vary in size and proportion and carry background areas that are not relevant, so each image passes through preprocessing that isolates the character area and standardises its dimensions. The character is separated from the background with Otsu's binarization (Otsu, 1979), and the character boundary is set so that all components, including the *sandhangan*, stay intact when the image is cropped. Figure 1 shows the stages.

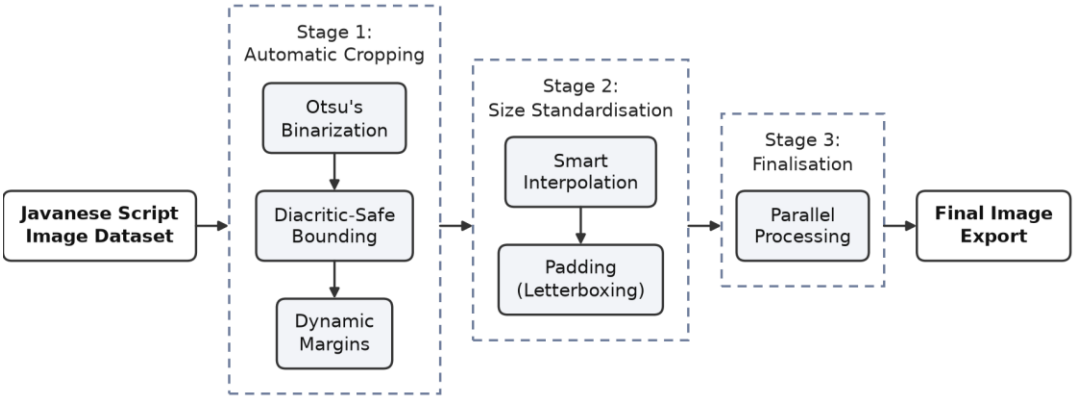


Figure 1. Preprocessing Stages.

Preprocessing runs in three steps. In the first, Otsu's binarization converts the image to binary form to separate the writing from the background. All writing pixels are detected with findNonZero so that *sandhangan* are included, then enclosed in a bounding box that is expanded by 1.618% of the longest side as a tolerance margin. In the second step, the cropped image is resized to 224×224 pixels, with area interpolation for reduction and cubic interpolation for enlargement so the character shape is not damaged (Gonzalez & Woods, 2018); remaining space is filled with white padding, which squares the image without altering its aspect ratio. In the third step all files are processed with multi-core parallel computation.

After preprocessing, images pass through augmentation, a common way to enlarge a small training set (Shorten & Khoshgoftaar, 2019; Yang et al., 2023). It runs dynamically in memory at each epoch and applies only to the training data, which prevents data leakage, so validation performance reflects unmodified images. Training images are rotated at random up to 15°, translated by 10%, and scaled between 80% and 120%, with areas left empty filled in white. The image is then converted to a tensor and normalised with the ImageNet statistics, mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225], so the input distribution matches the pretrained weights. Validation images receive no augmentation.

2.3 Architecture

The study uses the YOLO26 architecture modified for Javanese script classification, the main change being in the classification head: the base model, built for 1000 ImageNet classes, is adjusted to 20 output nodes. The backbone extracts image features in stages, from basic edge patterns to abstract representations, and is built from convolutional layers (O'Shea & Nash, 2015), C3k2 bottleneck modules derived from the cross stage partial design (Wang et al., 2019; Zhang et al., 2025), and a closing Cross Stage Partial with Spatial Attention (C2PSA) block that directs attention to the most relevant image areas (Woo et al., 2018; Wang et al., 2025). The architecture has no neck: multi-scale feature merging is needed to detect objects at varying sizes, while classification takes its output from the last backbone feature.

The network works in four stages, shown in Figure 2. Blocks 1 to 3 extract basic patterns: the convolutional stem in Block 1 reduces the image from 224×224 to 112×112 pixels, Block 2 reduces it to 56×56, and Block 3 applies a C3k2 module that holds 56×56 while raising the channel depth to 64. Blocks 4 to 6 handle intermediate semantics, taking the resolution to 28×28 and then 14×14 while the channels rise to 128. Blocks 7 to 10 cover abstraction and attention:

the resolution falls to 7×7 and the channels reach 256, and Block 10 applies the C2PSA module, whose self-attention mechanism links pixel correlations across the whole 7×7 map. Block 11 is the classification head: a convolutional layer expands the representation to 1280 channels, AdaptiveAvgPool2d reduces each channel to a single value, and the vector is flattened, passed through dropout, and mapped by a linear layer to scores for the 20 classes.

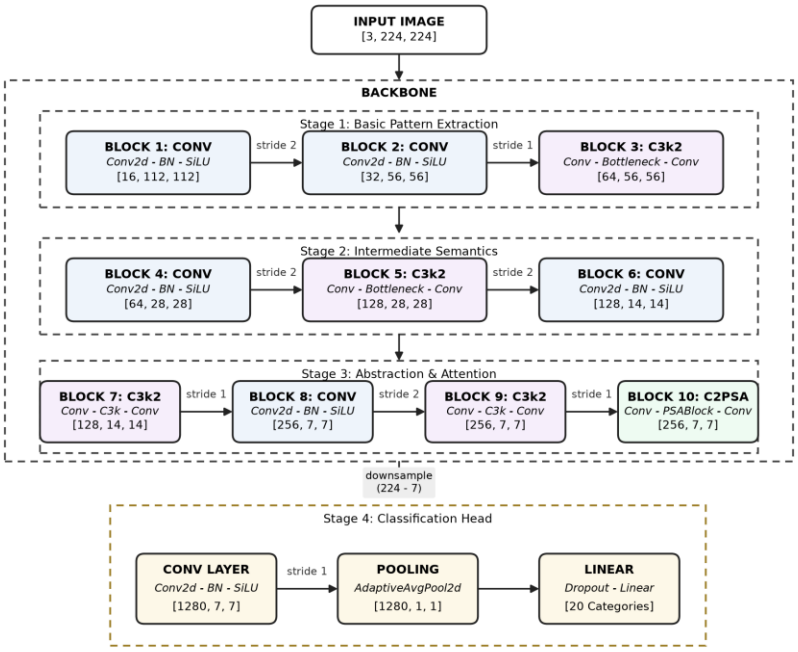


Figure 2. YOLO26 Architecture for Javanese Script Classification.

The network holds 1,556,724 parameters, all of them trainable, at a computational complexity of 6.21 GigaFLOPs, and the weight file is 6.23 MB. Training requires an estimated 658.12 MB of memory.

2.4 Training Setup

Training ran on an AMD Radeon RX 6800S GPU and an AMD Ryzen 9 6900HS processor under Linux Fedora 43, with PyTorch 2.11.0+rocm7.2 on the AMD ROCm backend, compiled through HIP/ROCm 7.2.26015. Scikit-learn and Matplotlib were used to compute and display evaluation metrics, and TensorBoard to monitor loss values during training. Table 2 lists the configuration.

Table 2. Training Parameters and Configuration.

Parameter	Value / setting	Parameter	Value / setting
Base model	YOLO26 nano variant	Learning rate	7×10^{-5}
Target classes	20	Weight decay	1×10^{-3}
Input dimension	[32, 3, 224, 224]	LR scheduler	CosineAnnealingLR
Batch size	32	Gradient clipping	max_norm = 1.0
Maximum epochs	30	Precision	Automatic Mixed Precision
Loss function	CrossEntropyLoss (class-weighted)	Stopping mechanism	Early stopping
Optimisation algorithm	AdamW	Patience	3 epochs

CrossEntropyLoss measures the prediction error across the 20 classes (Mao et al., 2023), weighted per class by N / (K · n_c), where N is the number of training images, K the number of

classes and n_c the number of images in class c, so that the small differences in class size do not bias the gradient. AdamW separates the weight decay term from the gradient update, which keeps the regularisation independent of the adaptive step size, and the initial learning rate of 7×10^{-5} is adjusted by the CosineAnnealingLR scheduler. Automatic Mixed Precision reduces memory use by running the forward pass in 16-bit precision while holding the weights at 32-bit. Training was capped at 30 epochs with a batch size of 32 and stopped automatically when the validation metric did not improve over three consecutive epochs; it stopped at the tenth epoch, and the weights were exported whenever validation performance improved on the previous epoch.

2.5 Evaluation and Kernel Extraction

The best model from training was tested on the held-out test set using accuracy, precision, recall and F1-score (Powers, 2011; Provost & Fawcett, 2013). The model runs inference over all test images and produces probability values for the 20 classes, summarised in a classification report giving precision, recall and F1-score per class, and in a confusion matrix comparing predictions against the ground truth.

To record how the parameters of the network change, the kernel values of every convolutional layer were read during training. The procedure walks the model with named_modules(), selects every layer of type nn.Conv2d, and copies the weight tensor from GPU to CPU at the end of each epoch so that graphics memory does not fill. This reads weights that already exist and does not alter training. The records are held in a pickle file covering initialisation to the end of training.

3. **Results And Discussion**

3.1 Preprocessing Result

Javanese characters often carry *sandhangan* attached at the top, bottom or side of the main character, and resizing can distort the shape, stretch it, or cut those elements off. Diacritic-safe bounding combined with letterboxing crops the excess background and standardises the resolution to a square without changing the aspect ratio of the writing, which is what allows the stroke characteristics to survive into the feature extraction stage. Figure 3 compares a character before and after preprocessing.

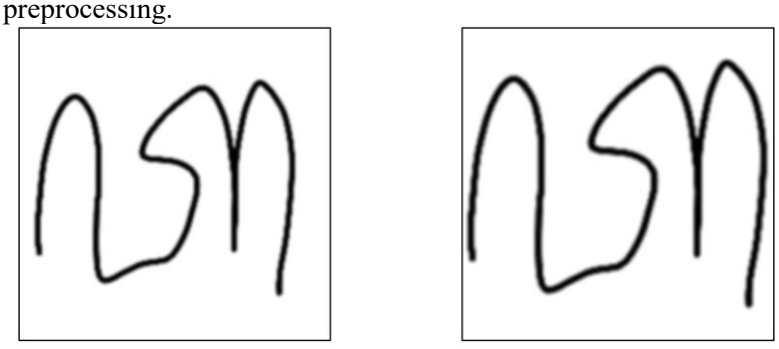


Figure 3. Javanese Character Before (a) and After (b) Preprocessing.

In Figure 3, the bounding box before preprocessing (a) is not uniform and leaves empty space around the character, while after preprocessing (b) the boundary follows the outer edge without cutting the ends of its curves. Letterboxing pads the shorter side, so the final image is square while the size and shape of the strokes stay as they were.

3.2 Classification Performance

Figure 4 shows the accuracy and loss curves for the training and validation data across the ten epochs.

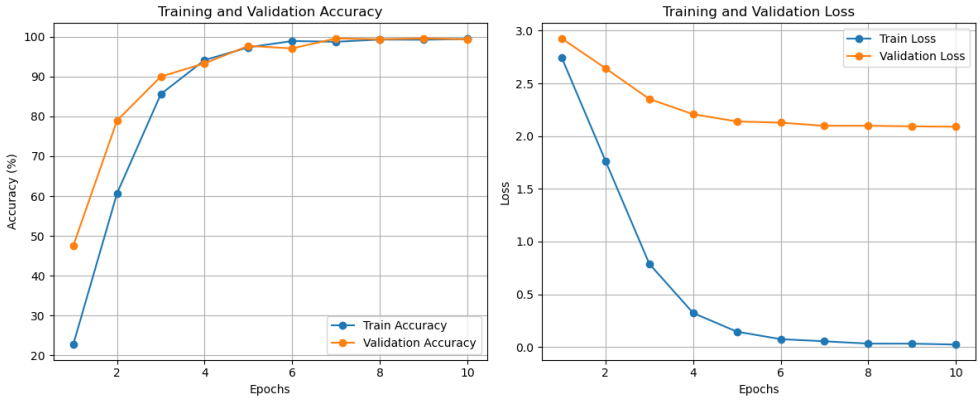


Figure 4. Training and Validation Accuracy and Loss.

The sharpest rise in accuracy occurs at the start. Training accuracy begins around 20% at epoch 1 and approaches 90% by epoch 3, and over the same range validation accuracy rises from around 48% to above 90%. Both curves settle close to 100% between epochs 8 and 10.

The two loss curves behave differently from each other. Training loss falls from about 2.75 to below 0.5 by epoch 4, then flattens towards 0. Validation loss falls from about 2.9 and levels off near 2.1 from epoch 6 onward, without following the training loss down. The two values were not computed on the same quantity: the classification head returns unnormalised class scores in training mode but a normalised probability vector in evaluation mode, and the loss function carries out a normalisation of its own, so the validation figure includes one normalisation step more than the training figure. For 20 classes that expression has a lower bound of 2.0782, which the curve reaches from epoch 6 and then holds, so the level section reflects that bound rather than the fit of the model. Recomputed without the additional step, the validation loss of the retained weights is 0.0312. The ordering of the class scores does not change under the extra normalisation, so validation accuracy, the confusion matrix and the class metrics below are unaffected.

Both accuracy curves had already flattened by epoch 8 and the validation loss had been level since epoch 6, so the run ended after the point where the metrics stopped moving. The weights kept for testing are those from the last epoch that improved on the one before it.

Table 3 gives precision, recall and F1-score for each class over the 25 test images.

Table 3. Classification Report per Class.

Class	Precision	Recall	F1-Score	Support
ma	1.0000	0.6667	0.8000	3
ca	0.5000	1.0000	0.6667	1
the other 18 classes	1.0000	1.0000	1.0000	1 each
Accuracy			**0.9600**	**25**
Macro average	**0.9750**	**0.9833**	**0.9733**	**25**
Weighted average	**0.9800**	**0.9600**	**0.9627**	**25**

Global accuracy is 0.9600 over the 25 test images, and the single prediction error appears in the class metrics: 'ma' records a recall of 0.6667 because one of its three samples was missed, and the precision of 'ca' falls to 0.5000 because it absorbed that false positive. The other 18 classes record 1.0000 throughout. The macro average F1-score is 0.9733 and the weighted average 0.9627. The test partition holds 25 images across 20 classes, so most per-class values rest on a single sample.

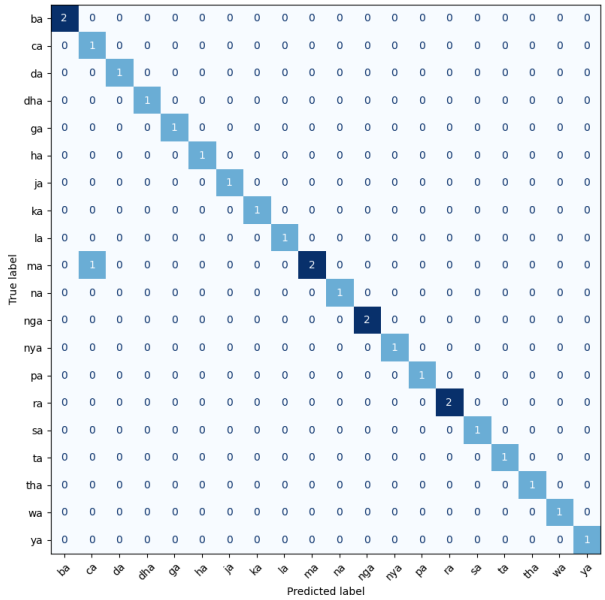


Figure 5. Confusion Matrix for the Test Partition.

Figure 5 maps the actual class labels against the highest-probability predictions. The main diagonal holds the correct predictions, and one point falls outside it: an image labelled 'ma' in the ground truth was predicted as 'ca'.

The single error was examined directly. The probability vector returned for the misclassified image divides across several classes: 'ca' 30.37%, 'ma' 18.50%, 'tha' 15.33%, 'sa' 9.65% and 'ta' 6.19%, a margin of 11.87 percentage points between the predicted and the true class. Across the 24 images classified correctly the highest probability ranges from 82.14% to 99.99%, with a median of 99.05%, and the median margin over the second-ranked class is 98.46 percentage points. The misclassified sample therefore sits 52 percentage points below the lowest value recorded on a correct prediction, and its probability mass stays divided across 'ca', 'ma' and 'tha'. These three characters are drawn from the same two elements, a rounded loop on the left joined to a tall arched stroke on the right, and differ in how far the loop is closed and where it meets the arch.

Two measurements separate the image from the rest of the partition. Its Laplacian variance, which responds to edge sharpness, is 47.1, the lowest of the 25 test images and about 56 times below the partition median of 2,634.0, and its ink coverage is 18.95% of pixels against a median of 11.19%. The second lowest sharpness value, 273.5, belongs to the other remaining 'ma' image, which also carries the heaviest ink coverage at 26.69%, so both extremes fall in the same class. Because the strokes are thick and the edges diffuse, the loop and its junction with the arch, where the two characters differ, are not resolved at the 224 × 224 input size. The error follows the condition of this one image rather than a systematic confusion between the classes: the other two 'ma' images were classified correctly, at 95.70% and 99.34%, and the 'ca' test image at 99.37%.

3.3 Comparison with Baseline Methods

Three further methods were trained and evaluated to place this result in context. All four runs use the same dataset, the same preprocessing, the same data split and the same evaluation metrics, and the three deep models share one training recipe: class-weighted CrossEntropyLoss, AdamW with weight decay 1×10^{-3}, CosineAnnealingLR, gradient clipping at 1.0, Automatic Mixed Precision and early stopping, with the seed fixed at 42. The Random Forest follows an established feature-based approach, with 500 trees over 8,100 HOG features (Rasyidi et al., 2021). The CNN holds three convolutional blocks and no pretrained weights, and was given up to 80 epochs, because early stopping with a patience of three ends a run from random

initialisation before its validation metric settles. ResNet-18 starts from ImageNet weights, an architecture already applied to Hanacaraka characters (Sudewo et al., 2024a). Table 4 lists the results.

Table 4. Comparison of Classification Methods on the Same Data.

Method	Parameters	Test accuracy	Test macro F1	Validation accuracy
CNN	460,308	0.5200	0.4286	0.6937
Random Forest + HOG	—	0.7600	0.6550	0.9146
ResNet-18 (ImageNet)	11,186,772	1.0000	1.0000	1.0000
YOLO26n-cls (this study)	**1,556,724**	**0.9600**	**0.9733**	**0.9958**

The two methods that do not start from pretrained weights fall furthest behind: the CNN reaches 0.5200 on the test partition and 0.6937 on the validation partition, and the Random Forest over HOG features reaches 0.7600 and 0.9146. A Random Forest over HOG features has been reported at 96.29% on a different dataset and partitioning (Rasyidi et al., 2021), which is not directly comparable; within this experiment, where the data and the split are held fixed, the feature-based method stays 20 percentage points below YOLO26 on the test partition and 8 points below it on the validation partition. ResNet-18 classified all 25 test images correctly, one more than YOLO26, using 11,186,772 parameters against 1,556,724, a factor of 7.2; on the 480-image validation partition the two separate by 0.42 percentage points, and the one differing test image is the blurred 'ma' sample described above. At 25 images a single sample moves accuracy by four percentage points, so the validation figures carry the firmer comparison.

3.4 Kernel Weights Across the Architecture Blocks

Weight values were recorded for every convolutional layer at the end of each epoch. The description below follows the four stages of the architecture, and Table 5 lists the recorded values at the first and the tenth epoch for one filter in each observed layer.

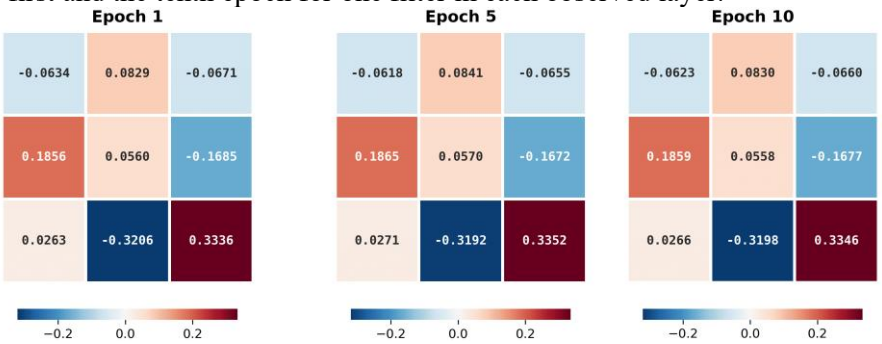


Figure 6. Weight Values of Layer model.0.conv at Epochs 1, 5 and 10.

In the first stage, Blocks 1 to 3, the weight matrices are 3×3 and hold values at the larger end of the recorded range. Figure 6 shows filter 0, channel 0 of layer model.0.conv at epochs 1, 5 and 10. The cell in the third row and second column moves from −0.3206 to −0.3198 and the cell in the third row and third column from 0.3336 to 0.3346, so the matrix carries a negative and a positive extreme at adjacent positions. A sign pattern of this kind measures the difference in intensity between neighbouring pixels, which marks the boundary between the dark pixels of the ink and the light pixels of the background. In Block 2 the values group along a diagonal while the lower-left quadrant stays negative, and in Block 3 most cells are negative.

In the second stage, Blocks 4 to 9, the numeric range narrows and most elements sit close to zero. Block 5 is the exception and holds a vertical contrast between its upper-middle and lower-middle areas. Blocks 7 and 9 record point-wise operations, where the weight is a single scalar rather than a 3×3 matrix, and Block 8 keeps a positive centre cell while its negative corner value shrinks.

In the third stage, Block 10 applies the C2PSA module, and its scalar stays negative across all ten epochs. In the fourth stage, Block 11 converts the feature map into class scores: its scalar stays positive, and the layer model.10.linear produces the raw logit scores that are normalised into probabilities for the 20 class labels. Across all observed layers the values move in small steps from one epoch to the next and change little after epoch 8. The magnitudes do not fall steadily with depth: the largest single values appear in Block 1, at about ±0.33, and in Block 5, at −0.3672 and 0.3136, while the smallest sit in the point-wise layers, where Block 7 stays between −0.0117 and −0.0147.

Table 5. Recorded Weight Values per Observed Layer at Epoch 1 and Epoch 10.

Block	Layer	Weight form	Recorded values, epoch 1 to epoch 10
1	model.0.conv	3×3 matrix	row 3 col 2 −0.3206 to −0.3198; row 3 col 3 0.3336 to 0.3346
2	model.1.conv	3×3 matrix	top-left 0.1137 to 0.1148; bottom-right 0.0869 to 0.0916
3	model.2.m.0.cv1.conv	3×3 matrix	centre −0.1237 to −0.1150
4	model.3.conv	3×3 matrix	all elements small, close to zero
5	model.4.m.0.cv1.conv	3×3 matrix	upper-middle −0.3672; lower-middle 0.3136
6	model.5.conv	3×3 matrix	bottom-right 0.0420; left column −0.0228
7	model.6.m.0.cv1.conv	scalar	−0.0117 to −0.0144 / −0.0147
8	model.7.conv	3×3 matrix	centre 0.1067; corner −0.0227 to −0.0174
9	model.8.m.0.cv1.conv	scalar	−0.0652 to −0.0625
10	model.9.m.0.attn.qkv.conv	scalar	−0.0545 to −0.0527 (epoch 7) to −0.0531
11	model.10.conv.conv	scalar	0.0927 to 0.0895

Table 5 brings together the readings for all eleven observed layers, including those not described individually above. The same filters can also be followed through their output rather than through their weight values.

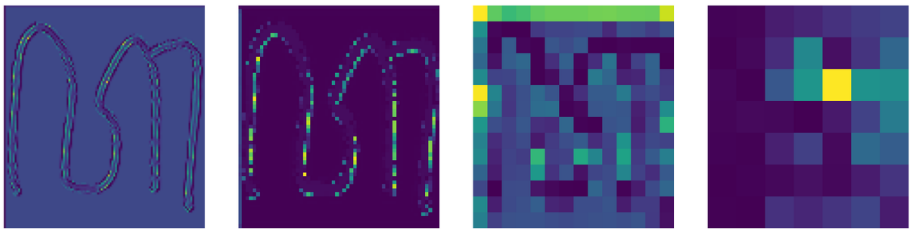


Figure 7. Output of the First Filter at Four Depths of the Network.

Figure 7 shows the output of the first filter at four depths. At Block 1 (a), at 112×112 pixels, the pixels along the outer contour of the strokes carry the highest intensity and the outline of the character is visible. At Block 3 (b), at 56×56 pixels, the continuous line breaks into discrete points along the curves. At Block 6 (c), at 14×14 pixels, the character appears as a grid of blocks, and at Block 10 (d), at 7×7 pixels, most of the grid is dark and one block carries a high value.

3.5 Prediction Probability Dynamics

The softmax function at the classification layer converts raw logit scores into probabilities between 0.0 and 1.0. Tracking these across epochs shows how the network handles Javanese characters whose shapes are close, such as 'tha' and 'nga' or 'ba' and 'nya', before the predictions settle. Figure 8 shows the shift for all 20 classes from epoch 1 to epoch 10.

Characters of this kind share most of their stroke pattern and differ over a small part of the shape, so probability can stay divided between them for several epochs before one is separated from the other. Reading the probability of each class at every epoch, rather than only the final

metric, shows at which point the network separates a pair and how long the division persists. The classes that resist separation longest here are also reported as the hardest in earlier work on the same script (Darmawan & Ariatmanto, 2025), although that study addressed detection with an additional background class over a test partition of different size, so the two sets of figures are not directly comparable.

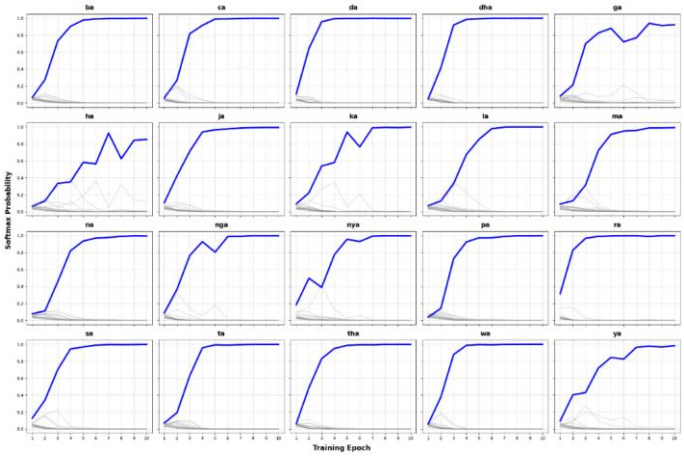


Figure 8. Probability Shift Across All 20 Classes.

Classes reach a stable probability at different rates. The curves for 'ba', 'da' and 'wa' rise steeply and settle near 1.0 after epoch 3, while classes with similar stroke shapes, among them 'ha', 'ka' and 'ga', fluctuate through the early and middle epochs, where the probability is still divided across several classes, before stabilising by epoch 10. Table 6 gives the values per epoch.

Table 6. Prediction Probability per Class and Epoch.

Class	1	2	3	4	5	6	7	8	9	10
ba	0.0645	0.2763	0.7312	0.9040	0.9781	0.9918	0.9972	0.9973	0.9985	0.9993
ca	0.0583	0.2673	0.8167	0.9161	0.9909	0.9932	0.9971	0.9990	0.9992	0.9986
da	0.1043	0.6429	0.9587	0.9954	0.9978	0.9711	0.9996	0.9985	0.9978	0.9985
dha	0.0486	0.4166	0.9200	0.9859	0.9942	0.9995	0.9996	0.9998	1.0000	0.9999
ga	0.0804	0.2113	0.6976	0.8262	0.8806	0.7214	0.7694	0.9403	0.9138	0.9238
ha	0.0620	0.1267	0.3329	0.3500	0.5812	0.5646	0.9268	0.6247	0.8422	0.8527
ja	0.1052	0.4229	0.7108	0.9407	0.9655	0.9767	0.9868	0.9920	0.9944	0.9942
ka	0.0910	0.2220	0.5384	0.5794	0.9394	0.7643	0.9887	0.9954	0.9928	0.9971
la	0.0696	0.1255	0.3300	0.6737	0.8536	0.9794	0.9973	0.9997	0.9994	0.9992
ma	0.0895	0.1274	0.3137	0.7194	0.9121	0.9512	0.9586	0.9880	0.9881	0.9921
na	0.0770	0.1113	0.4557	0.8200	0.9360	0.9709	0.9780	0.9914	0.9966	0.9951
nga	0.0873	0.3655	0.7640	0.9284	0.8051	0.9887	0.9915	0.9990	0.9974	0.9996
nya	0.1837	0.4971	0.3885	0.7741	0.9566	0.9304	0.9937	0.9972	0.9980	0.9968
pa	0.0369	0.1416	0.7277	0.9254	0.9741	0.9747	0.9899	0.9978	0.9967	0.9988
ra	0.3137	0.8284	0.9684	0.9913	0.9959	0.9983	0.9981	0.9901	0.9994	0.9997
sa	0.1274	0.3428	0.7006	0.9452	0.9686	0.9901	0.9965	0.9954	0.9965	0.9986
ta	0.0701	0.1920	0.6267	0.9586	0.9934	0.9907	0.9952	0.9982	0.9982	0.9989
tha	0.0609	0.4886	0.8300	0.9508	0.9868	0.9951	0.9938	0.9983	0.9982	0.9987
wa	0.0582	0.3788	0.8793	0.9871	0.9964	0.9937	0.9988	0.9993	0.9998	0.9998
ya	0.0954	0.4026	0.4302	0.7179	0.8450	0.8252	0.9652	0.9779	0.9677	0.9839

At epoch 1 most predictions sit below 0.1500, which indicates that the network has not yet taken up the pattern. Learning becomes visible at epoch 3, where several classes pass 0.70. The class 'ha' moves the slowest: it is held at 0.5646 at epoch 6 and rises to 0.8527 by epoch 10. Seventeen classes end above 0.9900. Two others, 'ga' at 0.9238 and 'ya' at 0.9839, approach but do not pass 0.99.

A single sample of class 'ta' shows the movement in more detail. At epoch 1 'ta' ranks third at 7.01%, below 'ya' at 10.17% and 'sa' at 7.09%. At epoch 2 'ta' moves to first place at 19.20%, with 'pa' and 'ma' as the nearest competitors, characters whose opening form and foot curve are close to it. The value reaches 62.67% at epoch 3 and 99.34% at epoch 5, and the split across similar classes ends there. At epoch 10 'ta' holds 99.89%. A second 'ta' image from the validation data gives 99.97%, the two differing slightly in stroke thickness and contrast.

4. Conclusions

The YOLO26 architecture classified Javanese script at 96% accuracy over 25 test images, with a macro average F1-score of 0.9733 and 18 of the 20 classes recognised without error; the only mistake was one 'ma' image classified as 'ca', the image with the lowest Laplacian variance in the test partition. The model holds about 1.5 million parameters with a weight file of 6.23 MB, so it can be trained on hardware with limited resources. Training and validation accuracy moved close together and stayed stable; the validation loss recorded during training levelled off near 2.1 because that figure carried one normalisation step more than the training figure, and recomputed consistently it is 0.0312, which leaves the accuracy figures unaffected. Against three methods trained on the same data, YOLO26 stayed 20 percentage points above a Random Forest over HOG features and 44 points above a CNN, and matched a ResNet-18 initialised from ImageNet weights to within one test image while holding 7.2 times fewer parameters. Observation of the eleven blocks showed the network building features in stages: larger weights forming clear sign patterns at edges and curves in Blocks 1 to 3, a denser representation with values closer to zero from Blocks 4 to 9, attention to class-separating parts in Block 10, and conversion to class scores in Block 11. Weight values changed in small steps and settled towards epoch 10, and most classes reached a high and stable probability after epoch 3, 'ha' being the slowest at 85%. Three limitations bound these results: the test set holds 25 images, so most per-class metrics rest on a single sample; the coverage is limited to the 20 Nglegena characters, without *sandhangan* or *pasangan*; and the weight observation covers one filter per layer. Observation of more filters, and evaluation on a larger test partition, would extend what can be said about how the network learns.

References

Darmawan, E. R., & Ariatmanto, D. (2025). Deteksi karakter aksara Jawa menggunakan YOLO11 pendekatan deep learning untuk pelestarian warisan budaya digital. *Jurnal Bangkit Indonesia*, *14*(2), 38–43. https://doi.org/10.52771/bangkitindonesia.v14i2.450

Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing* (4th ed.). Pearson.

Graciello, M. T., Handayani, A. N., & Wibawa, A. P. (2025). Optimization of Nglegena Javanese script recognition with machine learning based on zoning and normalization of feature extraction. *Indonesian Journal of Data and Science*, *6*(2), 281–293. https://doi.org/10.56705/ijodas.v6i2.256

Khanam, R., & Hussain, M. (2024). *YOLOv11: An overview of the key architectural enhancements* arXiv. https://arxiv.org/abs/2410.17725

Mao, A., Mohri, M., & Zhong, Y. (2023). Cross-entropy loss functions: Theoretical analysis and applications. In *Proceedings of the 40th International Conference on Machine Learning* (pp. 23803–23828). PMLR.

O'Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks* arXiv. https://arxiv.org/abs/1511.08458

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, *9*(1), 62–66.

Phiard. (2019). *Aksara Jawa* [Data set]. Kaggle. https://www.kaggle.com/datasets/phiard/aksara-jawa

Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *International Journal of Machine Learning Technology*, *2*(1), 37–63.

Provost, F., & Fawcett, T. (2013). *Data science for business: What you need to know about data mining and data-analytic thinking* (1st ed.). O'Reilly Media.

Rasyidi, M. A., Bariyah, T., Riskajaya, Y. I., & Septyani, A. D. (2021). Classification of handwritten Javanese script using random forest algorithm. *Bulletin of Electrical Engineering and Informatics*, *10*(3), 1308–1315. https://doi.org/10.11591/eei.v10i3.3036

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779–788).

Robby, A., Tandra, A., Susanto, I., Harefa, J., & Chowanda, A. (2019). Implementation of optical character recognition using Tesseract with the Javanese script target in Android application. *Procedia Computer Science*, *157*, 499–505. https://doi.org/10.1016/j.procs.2019.09.006

Sapkota, R., Cheppally, R. H., Sharda, A., & Karkee, M. (2026). *YOLO26: Key architectural enhancements and performance benchmarking for real-time object detection* arXiv. https://arxiv.org/abs/2509.25164

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, *6*, Article 60. https://doi.org/10.1186/s40537-019-0197-0

Sudewo, E. D. B., Biddinika, M. K., & Fadlil, A. (2024a). Javanese script Hanacaraka character prediction with ResNet-18 architecture. *JURTEKSI (Jurnal Teknologi dan Sistem Informasi)*, *10*(2), 363–370. https://doi.org/10.33330/jurteksi.v10i2.3017

Sudewo, E. D. B., Biddinika, M. K., & Fadlil, A. (2024b). DenseNet architecture for efficient and accurate recognition of Javanese script Hanacaraka character. *Matrik: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer*, *23*(2), 453–464.

Susanto, A., Mulyono, I. U. W., Sari, C. A., Rachmawanto, E. H., Setiadi, D. R. I. M., & Sarker, M. K. (2023a). Handwritten Javanese script recognition method based 12-layers deep convolutional neural network and data augmentation. *IAES International Journal of Artificial Intelligence*, *12*(3), 1448–1458. https://doi.org/10.11591/ijai.v12.i3.pp1448-1458

Susanto, A., Mulyono, I. U. W., Sari, C. A., Rachmawanto, E. H., Setiadi, D. R. I. M., & Sarker, M. K. (2023b). Improved Javanese script recognition using custom model of convolution neural network. *International Journal of Electrical and Computer Engineering*, *13*(6), 6629–6636. https://doi.org/10.11591/ijece.v13i6.pp6629-6636

Susanto, A., Sari, C. A., Rachmawanto, E. H., Mulyono, I. U. W., & Mohd Yaacob, N. (2024). A comparative study of Javanese script classification with GoogleNet, DenseNet, ResNet, VGG16 and VGG19. *Scientific Journal of Informatics*, *11*(1), 31–40. https://doi.org/10.15294/sji.v11i1.47305

Wang, C.-Y., Liao, H.-Y. M., Yeh, I.-H., Wu, Y.-H., Chen, P.-Y., & Hsieh, J.-W. (2019). *CSPNet: A new backbone that can enhance learning capability of CNN* arXiv. https://arxiv.org/abs/1911.11929

Wang, K., Liu, J., & Cai, X. (2025). *C2PSA-enhanced YOLOv11 architecture: A novel approach for small target detection in cotton disease diagnosis* [Unpublished manuscript]. College of Big Data, Yunnan Agricultural University.

Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). CBAM: Convolutional block attention module. In *Computer Vision – ECCV 2018* (pp. 3–19). Springer.

Yang, S., Xiao, W., Zhang, M., Guo, S., Zhao, J., & Shen, F. (2023). *Image data augmentation for deep learning: A survey* arXiv. https://arxiv.org/abs/2204.08610

Zhang, L., Deng, Y., & Zou, Y. (2025). Automatic road damage recognition based on improved YOLOv11 with multi-scale feature extraction and fusion attention mechanism. *PLOS ONE*, *20*(8), e0327387. https://doi.org/10.1371/journal.pone.0327387