Electronic Journal of Education, Social Economics and Technology
Vol. 5, No. 1, (2024), pp. 19~29
ISSN 2723-6250 (online)
DOI: https://doi.org/10.33122/ejeset.v5i1.310

SAINTIS publishing

Research Article

Signature Originality Verification Using A Deep Learning Approach

Muhammad Azi Saputra[1] & Ida Nurhaida[1,2*]

[1] Department of Informatics, Universitas Pembangunan Jaya, South Tangerang, 15413
[2] Center of Urban Studies, Universitas Pembangunan Jaya, South Tangerang, 15413
*Corresponding Author: ida.nurhaida@upj.ac.id | Phone: +6287876725326

ABSTRACT

The rapid advancement of digital technology has heightened the need for reliable methods to verify signature authenticity, a critical aspect of document and transaction security. This study uses a deep learning approach to develop a mobile application to verify the originality of paper and digital media signatures. The dataset comprises 1,060 signature images, including authentic and forged categories for both media types. The system employs the EfficientNetV2M model, trained with augmented data, to enhance robustness. Model evaluation demonstrates strong performance with an accuracy of 82.07%, a global precision of 81.31%, a global recall of 83.25%, and a global F1-score of 82.18%. The model is implemented in an Android-based mobile application, providing an intuitive interface for users to upload and verify signatures in real time. These results underscore the potential of EfficientNetV2M for mitigating signature fraud across various domains while highlighting areas for improvement, particularly in classifying paper-based signatures. Future work will focus on expanding the dataset and refining feature extraction techniques to enhance classification performance.

Keywords: Deep learning, Digital signature, EfficientNetV2M, Paper signature, Signature verification

1. INTRODUCTION

In the ever-evolving digital era, signature security has become a crucial issue. A signature is a person's identification mark to validate a document or show an agreement (Delvina, 2019; Octariadi, 2020). Signatures play a significant role in business and financial processes as a form of agreement and in verifying the authenticity of documents and transactions involving various parties, such as letters of agreement, Memorandum of Understanding (MoU), and documents related to legality (Andani & Satya Nugraha, 2020). Manual signature verification typically involves directly comparing original and forged patterns. Although frequently written signatures are similar, they are not always identical due to factors such as writing position, size, type of writing instrument used, and the individual's age or mental condition (Putra et al., 2023). However, technological advancements have made it easier to forge signatures, increasing document fraud with potentially severe consequences.

For instance, in 2022, irregularities were identified in the signatures submitted in a case brought by University of Lampung (UNILA) students to the Constitutional Court, indicating possible forgery (Ihsan, 2022). Similarly, in Samanera Village, South Sulawesi, forged signatures and stamps were used in a fraudulent proposal for government assistance (Pramono, 2024). These examples highlight the critical need for robust signature verification methods to prevent fraud and protect individual and organizational interests. Forgery of signatures can lead to legal sanctions under Article 263 of the Indonesian Criminal Code, which stipulates a maximum imprisonment of six years for such offenses (Izdihar Hulwa et al., 2023).

In response to these challenges, advancements in machine learning and deep learning offer promising solutions. One widely used method in signature verification research is Convolutional Neural Networks (CNN), known for effectively detecting visual patterns in images, including signatures (Rabbi et al., 2019). Various studies have explored different CNN architectures to enhance detection accuracy and efficiency. For instance, (Özyurt et al., 2023) combined MobileNetV2 with feature selection techniques such as Neighborhood Component Analysis (NCA) and Chi2, improving classification accuracy from 91.3% to 97.7%.

Other studies, such as those by (Jahandad et al., 2019), evaluated the performance of GoogleNet Inception-v1 and Inception-v3 for signature verification, revealing that Inception-v1 achieved higher validation accuracy (83%) compared to Inception-v3 (75%). This shows the importance of choosing the exemplary model architecture for signature verification. CNN was applied to offline handwritten signature verification, achieving 95.5% accuracy with 130 signature samples, further illustrating the strengths of CNN (Mosaher & Hasan, 2022).

Additionally, an approach combining Bidirectional Recurrent Neural Networks (BiRNN) with Discrete Fourier Transform (DFT) for feature extraction was proposed, showing the effectiveness of RNN-based architectures in reducing false rejection and acceptance rates (Nathwani, 2020). While these methods have demonstrated promising results, innovation still has the potential to improve both efficiency and accuracy. Although various approaches to signature verification have yielded promising results, there remains significant potential for further innovation to enhance both efficiency and accuracy. One of the most noteworthy advancements in deep learning is the implementation of EfficientNetV2M. This architecture provides distinct advantages, including improved computational efficiency and fewer parameters than earlier models, such as MobileNetV2 and GoogLeNet. As a state-of-the-art deep learning model, EfficientNetV2M has emerged as a compelling solution for signature verification. Its ability to optimize resource utilization while maintaining high accuracy makes it an ideal choice for addressing the demands of fast and reliable signature verification processes (Tan & Le, 2021).

Building on this background, this study aims to implement EfficientNetV2M to distinguish between genuine and forged signatures in paper and digital formats. The proposed approach accelerates the verification process, saves time, and minimizes human error. By leveraging EfficientNetV2M, the study seeks to contribute significantly to improving document security across various sectors, including business, finance, and academia. The findings may pave the way for developing more secure, efficient, and affordable signature verification technologies, addressing the growing challenges of signature security in the digital era (Widiawati & Suliswaningsih, 2022).

2. RESEARCH METHOD

Figure 1 describes the research flow, which includes nine main stages: data collection, preprocessing, data augmentation, model training, model evaluation, front-end development, back-end development, and model integration.

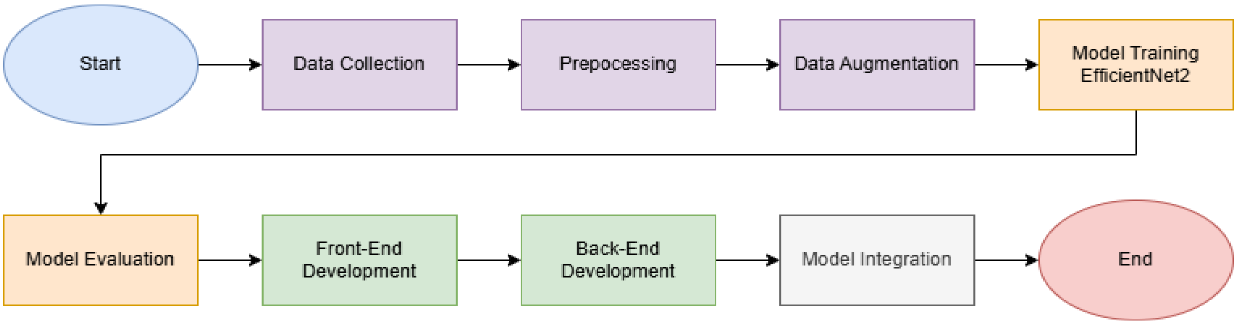


Figure 1. Research Flow

2.1 Data Collection

The data for this study were collected through a form distributed to Jaya University students, designed to gather examples of original signatures in both paper and digital formats. Five hundred thirty original signature images were obtained, evenly split between 265 images from paper media and 265 images from digital media, involving 53 participants. Five participants created forged signatures on both media types (paper and digital) to enhance data variation. This process added another 530 forged signature images, equally divided between 265 images from paper media and 265 images from digital media. The dataset comprised 1,060 images, consisting of original and forged signatures across paper and digital media. Table 1 presents a representative sample of the dataset used in this study, illustrating examples of genuine and forged signatures across both paper and digital media.

Table 1. Dataset Sample

Genuine Paper	Forged Paper	Genuine Digital	Forged Digital

2.2 Preprocessing and Augmentation

In the data preprocessing stage, several necessary steps are to prepare the signature images so that they are ready to be used in model training. First, the signature images are resized to 224x224 pixels to ensure dimensional consistency across all images. Next, the images are converted to grayscale format, and binary thresholding is applied to simplify irrelevant information so that the model can focus more on the pattern and shape of the signature (Pujianto et al., 2021; Rudiansyah et al., 2021). After preprocessing, the dataset was split into training (80%) and testing (20%) sets, resulting in 848 training and 212 testing images. Several data augmentation techniques were applied to generate five variations for each training image to enhance the training data, increasing the training dataset to 5,936 images. These techniques, explained in Table 2, include random rotation, horizontal flipping, random noise addition, brightness adjustment, and zoom transformations. Each method ensures that the model is exposed to diverse variations, improving its robustness in recognizing unseen data.

Table 2. Augmentation Thecniques

Technique	Description	Formula/Matrix
Rotation	Rotates the image randomly within the range of −15° to +15°	$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{bmatrix}$
Horizontal Flip	Mirrors the image along the vertical axis.	$F(x,y) = I(w - x - 1, y)$
Random Noise Addition	It adds Gaussian noise to simulate variations in image quality. The noise N is adjusted for each pixel intensity I	$N \sim N(0,\sigma^2)$ $I_{new} = I + N$
Brightness Adjustment	Adjusts the image's brightness randomly within 80% to 120% of the standard brightness	$I_{new} = \alpha . I$ $\alpha \in [0.8, 1.2]$
Zoom-in and Zoom-Out	Scales the image by a factor z, where z > 1 for zoom-in and z < 1 for zoom-out. For zoom-out, borders are padded with black pixels; for zoom-in, the central region is cropped.	$W_{new} = w . z$ $h_{new} = h . z$

These augmentation techniques were carefully selected to introduce realistic variations in the training data. For instance, random rotation ensures the model learns from rotated versions of signatures, while brightness adjustment and random noise simulate environmental conditions like lighting changes or noise in scanned images. Horizontal flipping adds diversity by creating mirror versions of the signatures, and zoom transformations alter the scale to ensure robustness against size variations. These techniques enhance the model's generalization capability, improving its performance on unseen data and making it more robust for signature verification tasks (Najda & Saeed, 2024). Figure 2 shows examples of the results after data preprocessing and augmentation, demonstrating how these techniques are applied to the original signature images.

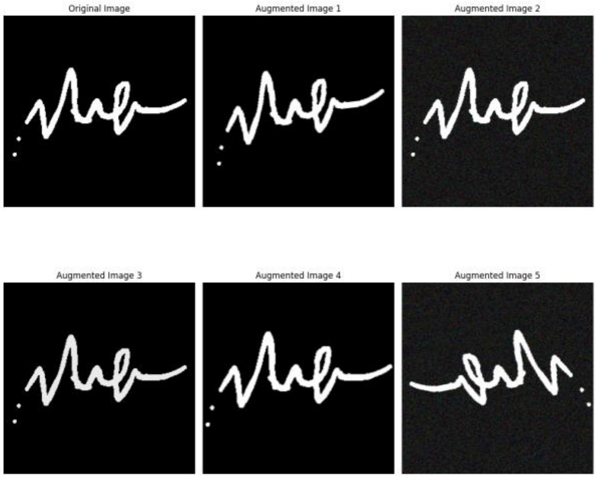


Figure 2. Data Preprocessing and Augmentation Techniques

2.3 Model Training EfficientNetV2M

Five thousand nine hundred thirty-six training data images were used to build the signature classification model. The EfficientNetV2M model as a base model froze the network layers up to the block7e_dwconv2 layer to utilize the features previously learned by the model. This approach allowed the model to adapt faster to the signature data by leveraging prior knowledge from the pre-trained features. The implementation used Keras and TensorFlow libraries, widely used frameworks for building and training deep learning models. The training was conducted on Kaggle's platform, utilizing NVIDIA Tesla T4 (2 GPUs) and the local system's 12 GB RAM. The Adam optimizer was employed for training with a learning rate of 0.0003, a batch size of 32, and a total of 30 epochs. The training aimed to teach the model to recognize key patterns and differences between genuine and forged signatures by focusing on features such as stroke order, pressure points, pen tilt, and angles, connecting lines, shaky hands, and retouching indicators (Tirumala et al., 2024).

Table 3. Pseudocode: Implementing EfficientNetV2M Model

Step	Description
Import Libraries	Import EfficientNetV2M, Dense, Dropout, Flatten, BatchNormalization, Adam, and utilities for building the model.
Load Pre-trained Model	Load the pre-trained EfficientNetV2M model with: - weights='imagenet' - include_top=False to exclude the fully connected layers. - Input shape (224, 224, 3) for RGB images.
Freeze Layers	Freeze initial layers up to block7e_dwconv2: - Iterate through layers of the base model. - Set trainable = False for layers before the specified block.
Add Custom Layers	Build the top layers for classification: - Add a flattened layer to convert feature maps to a 1D vector. - Add Dense layers with ReLU activation, L2 regularization, Batch Normalization, and Dropout: - First Dense: 256 units, Dropout 0.3 - Second Dense: 128 units, Dropout 0.25 - Third Dense: 64 units, Dropout 0.25 - Add the output layer: Dense with four units (for four classes) and Softmax activation.
Compile Model	Compile the model with: - Optimizer: Adam, with a learning rate of 0.0003, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and AMSGrad = True - Loss function: categorical_crossentropy. - Evaluation metric: Accuracy.
Train Model	Train the model using training data: - Use a validation split (e.g., 20% of training data). - Train for 30 epochs with a batch size of 32. - Apply callbacks such as ReduceLROnPlateau (reduce learning rate) and EarlyStopping (stop training when validation loss does not improve).

Table 3 outlines the pseudocode for implementing the EfficientNetV2M model, detailing each step, from importing necessary libraries to training the model. The process includes loading the pre-trained EfficientNetV2M model with specified configurations, freezing layers up to block7e_dwconv2 to utilize learned features, adding custom classification layers, compiling the model with the Adam optimizer, and defining essential callbacks like ReduceLROnPlateau and EarlyStopping. Each step presents a structured guideline that facilitates algorithm replication. This table effectively fulfills the requirement to include pseudocode for implementing the EfficientNetV2M model in the study.

2.4 Model Evaluation

After training, the model is tested using test data. Evaluation metrics include accuracy, precision, recall, and F1-score.

1) Accuracy, precision, recall, and f1-score
Accuracy measures the proportion of correct predictions from the total predictions, providing an overall picture of the model's performance. However, this metric may be distorted if the data is imbalanced. Precision indicates how accurately the model classifies genuine signatures by calculating the proportion of accurate optimistic predictions. Recall measures the model's ability to identify all genuine signatures, assessing how well the model detects hidden genuine signatures. F1-Score, the harmonic mean of precision and recall, provides a more balanced evaluation, particularly for imbalanced datasets. A high F1-Score reflects accuracy in optimistic predictions and the model's ability to identify the most positive examples (Krstinić et al., 2020; Sitarz, 2022).

2) Confusion Matrix
As shown in Table 4, the confusion matrix is a tool to assess the performance of a classification model and describe the presentation of the model's prediction performance by comparing the prediction results with the actual data values (Swaminathan & Tantri, 2024). The Confusion Matrix will calculate accuracy, precision, recall, and f1-score from the following equations: (1), (2), (3), and (4), respectively (Evidently AI, 2024).

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \quad (1)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} \quad (2)$$

$$\text{F1 score} = 2 \text{ x } \frac{\text{Precision x Recall}}{\text{Precision + Recall}} \quad (3)$$

$$\text{Accuracy} = \frac{\text{Correct Prediction}}{\text{All Prediction}} = \frac{TP + TN}{TP + FP + TN + FN} \quad (4)$$

Table 4. Confusion Matrix

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

2.5 Front-End Development

At this stage, the application interface was designed using the Flutter programming language. Flutter, developed by Google, is a free and open-source framework that supports multi-platform application development from a single codebase (Tashildar et al., 2020). Its direct-to-hardware rendering capabilities enhance graphic performance and responsiveness, making it an excellent choice for applications with complex and dynamic user interfaces (GÜLCÜOĞLU et al., 2021). Figure 3 illustrates the designed application interface.

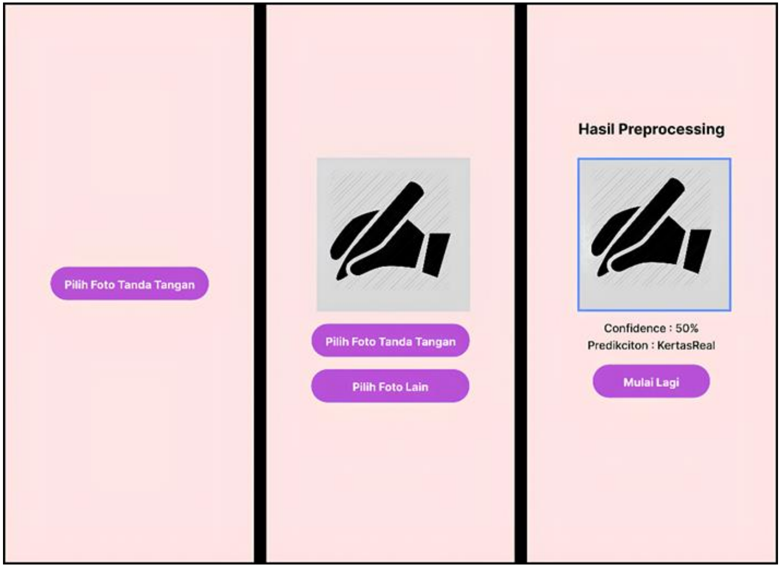


Figure 3. Application Interface Design Prototype

2.6 Back-End Development

Python and Flask were utilized as the frameworks for backend development. Flask, a simple yet flexible Python-based microweb framework, supports modular architecture such as Model-View-Controller (MVC). This framework simplifies routing management, facilitates API development, and allows for adding additional features through extensions (Ningtyas & Setiyawati, 2021). The back end was designed to handle image inputs, process them to generate pre-diction results, calculate confidence values, and produce preprocessed images. Furthermore, it includes an API endpoint that accepts user image inputs and returns JSON responses containing the prediction results, confidence values, and links to the preprocessed images.

3) RESULTS AND DISCUSSION

The EfficientNetV2M training process was conducted for 20 minutes using Python 3.11.8 with a Tesla T4 GPU. This process produced several findings.

3.1 Training Performance Matrics

The training accuracy consistently increases from 44.23% in the first epoch to 96.80% in the last epoch without significant decreases. Validation accuracy, on the other hand, starts at 59.82% and reaches a peak of 91.55%. Despite minor fluctuations in certain epochs—such as the 12th (79.47%) and 21st (83.60%)—the model demonstrates effective learning and strong generalization on validation data. These fluctuations are attributed to the nature of validation data or the model's sensitivity to specific data patterns.

Similarly, the loss metrics indicate a steady improvement in model performance. Initially, the training loss starts at 5.2875 and decreases to 0.7736 by the 30th epoch. Validation loss also shows a significant decline, starting at 4.4475 and reducing to 0.8598 in the final epoch. However, minor increases in validation loss occur during the 12th epoch (1.7781) and the 28th epoch (1.1281), reflecting minor adaptation challenges to validation data. The training accuracy and loss metrics are illustrated in Figure 4, respectively.

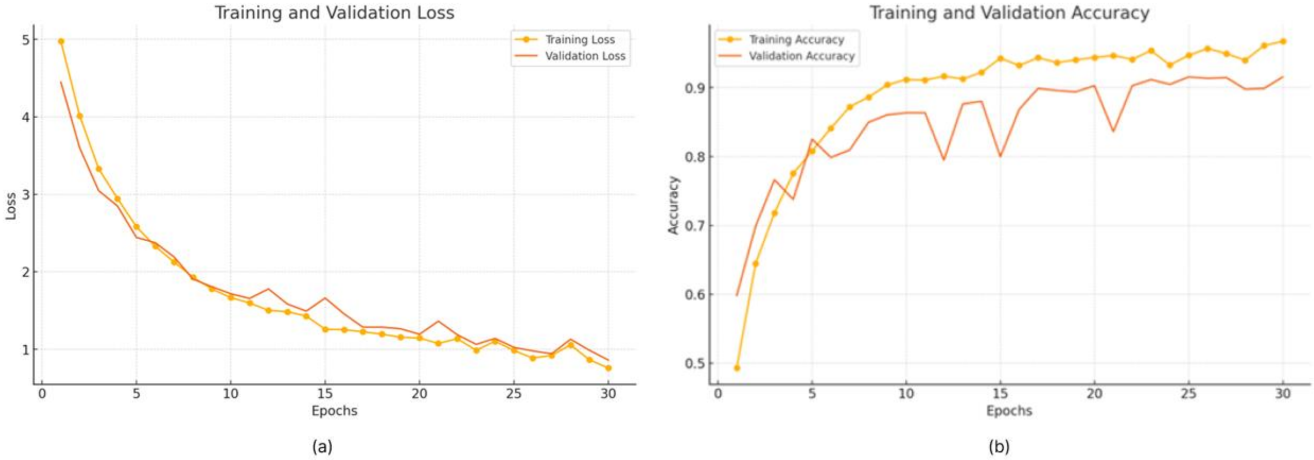


Figure 4. (a) Loss Graph on Training and Validation Data, and (b) Accuracy Graph on Training and Validation Data

3.1 Confidence Intervals for Metrics

Table 5 below presents the mean and confidence intervals for the accuracy and loss metrics during training and validation. The confidence intervals provide insights into the stability and reliability of the model's performance across different epochs, indicating high confidence in the reported mean values.

Table 5. Confidence Intervals for Accuracy and Loss Metrics

Metric	Mean	CI Lower	CI Upper
Training Accuracy	0.888346	0.848826	0.927865
Validation Loss	0.849738	0.822061	0.877415
Training Loss	1.688951	1.317453	2.060449
Validation Loss	1.723049	1.407757	2.038340

The mean values and confidence intervals for the accuracy and loss metrics during training and validation were calculated with a 95% confidence level, providing a range of values within which the actual mean performance is expected to lie. The training accuracy shows a high mean value of 88.83% with a confidence interval ranging from 84.88% to 92.79%, indicating consistent performance during training. The validation accuracy demonstrates the model's ability to generalize well to unseen data with a mean of 84.97% and a confidence interval of 82.21% to 87.74%. The overlap between the confidence intervals for training and validation accuracy indicates that the difference in performance between the two datasets is not statistically significant, supporting the stability of the model during training.

For the loss metrics, the training loss has a mean value of 1.689 and a confidence interval ranging from 1.317 to 2.060, while the validation loss has a mean value of 1.723 and a confidence interval of 1.408 to 2.038. These intervals reflect a steady reduction in the loss values over time, indicating that the model is learning from the training data. Although the validation loss is slightly higher than the training loss, the overlapping confidence intervals suggest that the model maintains a good balance between learning and generalizing to the validation data. The confidence intervals further assure the model's reliability by highlighting its consistent performance across different epochs. Despite minor fluctuations observed in some epochs, the narrow intervals demonstrate the model's stability and robustness throughout the training process.

3.3 Hyperparameter Tuning

The EfficientNetV2M model underwent hyperparameter tuning to optimize performance and balance generalization. Layers up to block7e_dwconv2 in the base model were frozen to retain pre-trained low-level features. In contrast, custom dense layers were added with progressively decreasing units (256, 128, 64) and l2 regularization (0.005) to prevent overfitting. Dropout rates of 0.3 and 0.25 were applied across the layers, paired with batch normalization to stabilize learning. The Adam optimizer was used with a reduced learning rate of 0.0003, and AMSGrad enabled efficient weight updates. Dynamic callbacks, including ReduceLROnPlateau and EarlyStopping, ensured optimal training by adjusting the learning rate and halting when validation loss plateaued. This tuning approach allowed the model to learn effectively while minimizing overfitting and maintaining robust performance across datasets.

3.4 Model Evaluation EfficientNetV2M

The evaluation of the EfficientNetV2M model includes analyzing its classification performance through a confusion matrix and detailed metrics such as precision, recall, and F1-score. An error analysis was also conducted to identify misclassification patterns and suggest improvements for better generalization. The resulting confusion matrix in Figure 5 shows the classification model's performance in determining the authenticity of signatures across four classes: 'genuine paper,' 'forged paper,' 'genuine digital,' and 'forged digital.' Each class contains 53 samples, for a total of 212 samples. The model successfully classified 174 samples correctly, resulting in % overall accuracy of 82.07%. Analysis of the evaluation metrics for each class highlights the model's capability to discriminate between the classes. Class 2 ('genuine digital') achieved the highest F1-score of 86.80%, demonstrating the model's strong ability to identify genuine digital signatures with minimal misclassification. On the other hand, Class 0 ('genuine paper') exhibited the lowest F1-score of 76.25%, primarily due to a higher number of false positives (FP) and false negatives (FN). These errors suggest challenges distinguishing genuine paper signatures from forged ones, likely due to their visual similarity or limited diversity in the training data.

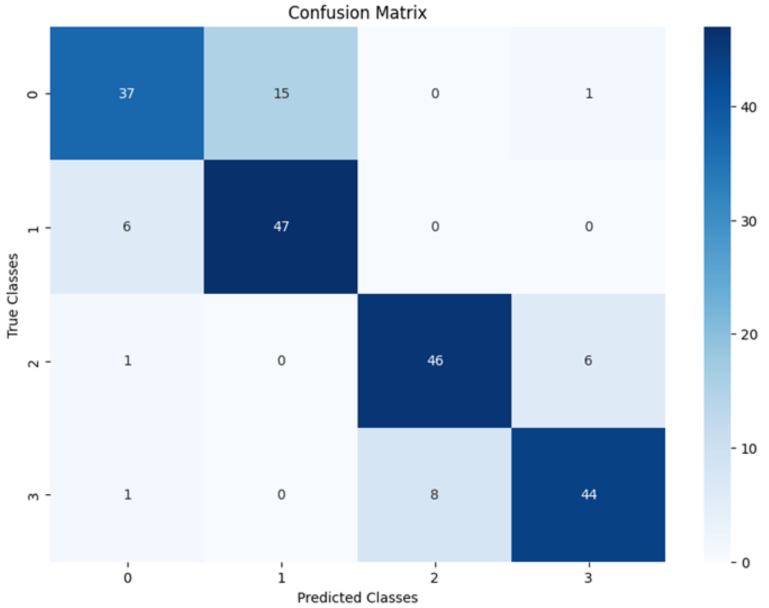


Figure 5. Confusion Matrix

Despite these challenges, the model demonstrates effective discrimination overall, with a global precision of 81.31%, recall of 83.25%, and F1-score of 82.18%. These metrics indicate that the model is generally reliable in classifying samples across all four classes. The confusion matrix also reveals that while digital signatures ('genuine digital' and 'forged digital') are easier to distinguish, paper-based signatures ('genuine paper' and 'forged paper') present more incredible difficulty due to overlapping features. Global performance metrics, including precision, recall, F1-score, and accuracy, are calculated as follows:

$$\text{precision} = \frac{174}{174+40} = 0.8131$$

$$\text{Recall} = \frac{174}{174+35} = 0.8325$$

$$\text{F1 score} = 2 \text{ x } \frac{(0.8131 \text{ X } 0.8325)}{0.8131+\ 0.8325} = \frac{2 \text{ X } 0.6769}{1.6456} = 0.8218$$

$$\text{Accuracy} = \frac{174}{212} = 0.8207$$

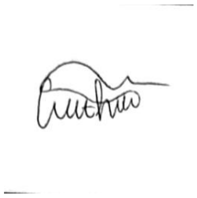
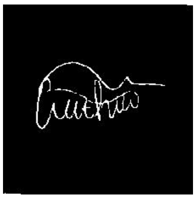
Table 6. Model Performance Evaluation Based on Precision, Recall, and F1-score for Each Class

Classes	Precision (%)	Recall (%)	F1-score (%)
Genuine Paper (0)	82	70	76
Forged Paper (1)	76	89	82
Genuine Digital (2)	85	87	86
Forged Digital (3)	86	83	85

Table 6 summarizes the performance metrics for each class, highlighting the model's ability to classify samples accurately. Precision, which measures the proportion of correct optimistic predictions, is highest for class 3 ('forged digital') at 86% and lowest for class 1 ('forged paper') at 76%. Recall, which measures the model's ability to detect true positives, is highest for class 1 ('forged paper') at 89% and lowest for class 0 ('genuine paper') at 70%. The F1-score, which balances precision and recall, is highest for class 2 ('genuine digital') at 86% and lowest for class 0 ('genuine paper') at 76%. These results indicate that the model performs consistently well across most classes but struggles with classifying genuine paper signatures accurately, which negatively affects its overall recall for this class.

An in-depth analysis of the model's errors highlights specific challenges in distinguishing between particular classes, particularly between 'genuine paper' (Class 0) and 'forged paper' (Class 1). False positives (FP) often occurred when genuine paper signatures were misclassified as forged paper, likely due to visual similarities in stroke patterns or a lack of sufficient variability in the training data for these classes. On the other hand, false negatives (FN) were commonly observed for genuine paper signatures, where they were misclassified as forged or digital signatures. This indicates that the model struggles to generalize well for the genuine paper class. In contrast, the model demonstrated stronger performance in detecting 'genuine digital' signatures (Class 2), exhibiting relatively low rates of FN and FP. This suggests that the model is particularly well-suited for handling digital signature data. Several strategies can address the issues observed with genuine paper signatures. Data augmentation techniques such as adding noise, rotation, and scaling can be applied to improve the diversity of the training samples. Furthermore, extracting more discriminative features and using class-balanced training strategies, such as oversampling or applying weighted loss functions, can help reduce misclassifications and improve the model's ability to generalize effectively. Table 7 presents examples of the model's errors, highlighting common misclassifications such as false positives and false negatives.

Table 7. Error Analysis for Signature Classification

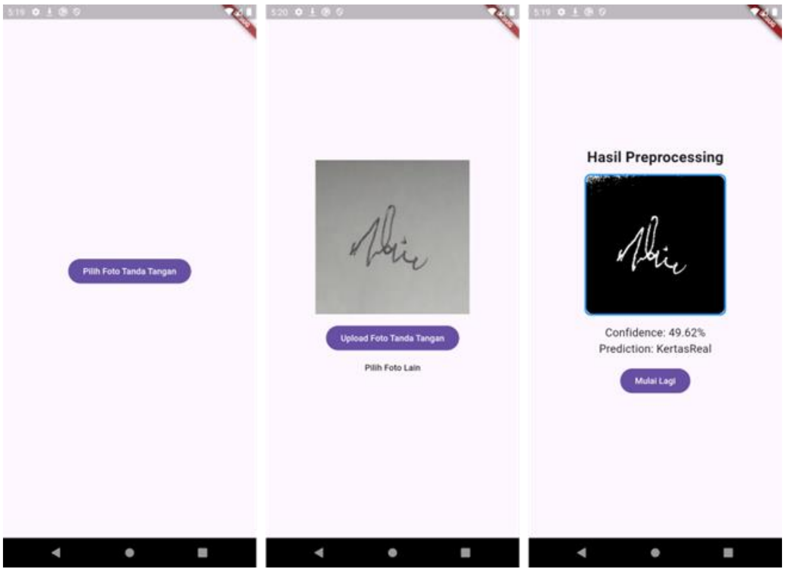
Example Image	Preprocessing	Actual Class	Predicted Class	Error Type
		Genuine Paper	Forged paper	False Positive (FP)
		Genuine Paper	Forged Digital	False Negative (FN)
		Forged paper	Genuine Paper	False Negatives

		Genuine Digital	Genuine Digital	Correct

3.5 Model Implementation in Mobile Application

After successfully developing the EfficientNetV2M model, the next step is to create a prediction function that processes input images and generates output from confidence values and class predictions. The model is initialized using weights from the `signature_classification_model_82%.h5` file. This function begins by loading the input image using TensorFlow Keras' `image.load_img` function, then applies preprocessing steps such as resizing the image to 224x224 pixels, converting it to grayscale with OpenCV, performing binary thresholding, and finally converting the image back to RGB format with three channels. Once preprocessing is completed, the model generates probability predictions for each class, and the confidence value is calculated based on the highest class probability. These prediction results and the confidence score are returned as a dictionary. The pre-processed image is also saved to the `static/preprocessed` folder with a unique filename and a timestamp.

This prediction function can be integrated into an image upload route, enabling it to process HTTP POST requests in a REST API-based application. The Flask-based service is designed to handle image files for classification through the `/upload` endpoint. Upon receiving an HTTP POST request with an image file, the service saves the image to the `static/uploads` directory. The image undergoes preprocessing, including resizing, grayscale conversion, and thresholding, to prepare it for the model. The model then predicts the class label of the image along with its confidence score. The output is returned as a JSON response, which includes the predicted class label, confidence percentage, and the file path to the preprocessed image. The communication between the mobile application and the backend follows a clear request/response structure: The client sends a POST request to the `/upload` endpoint, which includes the image file in the request body with the content type set to `multipart/form-data.` The backend responds with a JSON object containing the `preprocessed_image_path,` the predicted `prediction` class label (e.g., "Genuine Paper "), and the `accuracy` score (the confidence percentage). For instance, a response might include: `{"preprocessed_image_path":"static/preprocessed/Genuine Paper.png," "prediction": "Genuine Paper, ""accuracy": 49.62}`. Figure 6 illustrates the implementation of the mobile application that integrates this Flask-based service. This mobile application allows users to upload images, which are processed and classified through the API, providing real-time predictions and confidence scores.



Hasil Preprocessing

Confidence: 49.62%
Prediction: KertasReal

Figure 6. Implementation Application Mobile

3.6 Discussion

Previous studies have developed a system for verifying genuine or fake signatures using deep learning models but have not integrated them into a system. This study implements the model into a system as an application and utilizes the results of verifying genuine or fake signatures based on paper and digital media. Although the developed model shows less than optimal performance on the test dataset, the model still shows potential for classifying genuine or fake signatures from paper and digital media.

4) CONCLUSION

This study successfully developed the EfficientNetV2M model to authenticate paper and digital signatures. The model achieved a training accuracy of 96.80% and a peak validation accuracy of 91.55% during training, with a % accuracy on new data of 82.07%. While the evaluation results demonstrated good overall performance, the model showed weaknesses in the 'Genuine Paper' class, which had the lowest F1-score of 76.25%, compared to the highest F1-score of 86.80% in the 'Genuine Digital' class. This highlights the need for further improvements, such as expanding the dataset by 30% to include more diverse Genuine Paper signatures and applying additional data augmentation techniques to improve performance and reduce misclassifications of paper-based signatures. Additionally, model performance can be further enhanced by optimizing parameters and exploring feature engineering methods to improve generalization and consistency across different data distributions. The model has been successfully implemented in an Android-based application, allowing users to upload signature images and receive prediction results and confidence values. This application provides a practical solution for verifying digital and paper signatures, offering real-time predictions to end users. However, there are challenges regarding deploying this application in real-world scenarios. Issues such as scalability, device compatibility, and managing large volumes of requests need to be addressed for widespread adoption. Furthermore, user readiness must be considered, particularly regarding the user interface, ease of use, and ensuring that the application performs consistently across diverse conditions. Future research can focus on refining the deployment of the model in real-world applications by improving scalability, optimizing the user experience, and evaluating the system's performance on a larger scale to ensure its effectiveness in various environments.

REFERENCES

Andani, M. W., & Satya Nugraha, G. (2020). Signature Verification Using Feature of LBP and DCT With LVQ Classifier. *Jurnal Teknologi Informasi, Komputer, Dan Aplikasinya (JTIKa)*, *2*(2). https://doi.org/https://doi.org/10.29303/jtika.v2i2.107

Delvina, A. (2019). Penggunaan Tanda Tangan Elektronik dalam Pengajuan Pembiayaan berdasarkan Prinsip Syariah. *Jurnal Akuntansi Bisnis Dan Ekonomi*, *5*(1). https://doi.org/10.33197/JABE.VOL5.ISS1.2019.230

Evidently AI. (2024, October 1). *Accuracy, precision, and recall in multi-class classification.* https://www.evidentlyai.com/classification-metrics/multi-class-metrics

Gülcüoğlu, E., Ustun, A. B., & Seyhan, N. (2021). Comparison of Flutter and React Native Platforms. *Journal of Internet Applications and Management*. https://doi.org/10.34231/iuyd.888243

Ihsan, D. (2022, July 18). *Kasus Tanda Tangan Palsu, Rektor Unila Buka Suara* . https://www.kompas.com/edu/read/2022/07/18/163653471/kasus-tanda-tangan-palsu-rektor-unila-buka-suara

Izdihar Hulwa, S., Khairani Br Ginting, R., Merlani Purba, D., Stevani Siahaan, C., Gabriel Siahaan, P., & Pika Lb Batu, D. (2023). *Tindak Pidana Pemalsuan Tanda Tangan Akta Tanah Ditinjau Dari Pasal 263 KUHP (Putusan No. 55/Pid.Pra/2023/Pn. Medan)*. *03*(06), 799–807. https://j-innovative.org/index.php/Innovative

Jahandad, Sam, S. M., Kamardin, K., Amir Sjarif, N. N., & Mohamed, N. (2019). Offline signature verification using deep learning convolutional Neural network (CNN) architectures GoogLeNet inception-v1 and inception-v3. *Procedia Computer Science*, *161*, 475–483. https://doi.org/10.1016/j.procs.2019.11.147

Krstinić, D., Braović, M., Šerić, L., & Božić-Štulić, D. (2020). *Multi-label Classifier Performance Evaluation with Confusion Matrix*. 01–14. https://doi.org/10.5121/csit.2020.100801

Mosaher, Q. S., & Hasan, M. (2022). Offline Handwritten Signature Recognition Using Deep Convolution Neural Network. *European Journal of Engineering and Technology Research*, *7*, 44–47. https://doi.org/10.24018/ejeng.2022.7.4.2851

Najda, D., & Saeed, K. (2024). Impact of augmentation methods in online signature verification. *Innovations in Systems and Software Engineering*, *20*(3), 477–483. https://doi.org/10.1007/s11334-022-00464-4

Nathwani, C. (2020). Online Signature Verification Using Bidirectional Recurrent Neural Network. *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 1076–1078. https://doi.org/10.1109/ICICCS48265.2020.9121023

Ningtyas, D. F., & Setiyawati, N. (2021). Implementasi Flask Framework pada Pembangunan Aplikasi Purchasing Approval Request. *Jurnal Janitra Informatika Dan Sistem Informasi*, *1*(1), 19–34. https://doi.org/10.25008/janitra.v1i1.120

Octariadi, B. C. (2020). Pengenalan Pola Tanda Tangan Menggunakan Metode Jaringan Syaraf Tiruan Backpropagation. *Jurnal Teknoinfo*, *14*(1), 15–21. https://doi.org/10.33365/jti.v14i1.462

Özyurt, F., Majidpour, J., Rashid, T. A., & Koç, C. (2023). Offline Handwriting Signature Verification: A Transfer Learning and Feature Selection Approach. *Traitement Du Signal*, *40*(6), 2613–2622. https://doi.org/10.18280/ts.400623

Pramono, A. (2024). *Kades di Bone Geram Tanda Tangannya Dipalsukan untuk Bantuan Motor 3 Roda*. Detiksulsel. https://www.detik.com/sulsel/berita/d-7127627/kades-di-bone-geram-tanda-tangannya-dipalsukan-untuk-bantuan-motor-3-roda

Pujianto, R., Lestari, M., Wayan, N., Septiani, P., Raya, J., No, T., Gedong, K., Rebo, P., & Timur, J. (2021). Pengolahan Citra Dan Metode Support Vector Machine (Svm) Dalam Pengenalan Pola Tanda Tangan. *JRKT (Jurnal Rekayasa Komputasi Terapan)*, *01*(01), 2776–5873. https://doi.org/10.30998/jrkt.v1i01.4048

Putra, I. K. N., Dewi, N. P. D. A. S., Pusparani, D. A., & Mupu, D. N. (2023). Signature Identification using Digital Image Processing and Machine Learning Methods. *Jurnal Eltikom*, *7*(1), 29–37. https://doi.org/10.31961/eltikom.v7i1.618

Rabbi, Md. T. F., Rahman, S. M. T., Biswash, P., Kim, J., Sheikh, A., Saha, A. K., & Uddin, M. S. (2019). Handwritten Signature Verification Using CNN with Data Augmentation. *The Journal of Contents Computing*, *1*(1), 25–37. https://doi.org/http://dx.doi.org/10.9728/jcc.2019.12.1.1.25

Rudiansyah, Ryanto, S. S., Rojali, Pandia, H., Marwan, R., Yoshara, R., & Effendi, K. (2021). Aplikasi Deteksi Penyakit Tuberculosis (TB) Pada Balita Menggunakan Metode Pengolahan Citra Matlab. *Jurnal Teknik Informatika*, *1*(3), 20–25. https://doi.org/10.58794/jekin.v1i3.354

Sitarz, M. (2022). *Extending F1 metric, probabilistic approach*. https://doi.org/10.54364/AAIML.2023.1161

Swaminathan, S., & Tantri, B. R. (2024). Confusion Matrix-Based Performance Evaluation Metrics. *African Journal of Biomedical Research*, 4023–4031. https://doi.org/10.53555/AJBR.v27i4S.4345

Tan, M., & Le, Q. V. (2021). *EfficientNetV2: Smaller Models and Faster Training*. https://doi.org/10.48550/arXiv.2104.00298

Tashildar, A., Shah, N., Gala, R., Giri, T., & Chavhan, P. (2020). APPLICATION DEVELOPMENT USING FLUTTER. *International Research Journal of Modernization in Engineering Technology and Science @International Research Journal of Modernization in Engineering*, *02*(08). www.irjmets.com

Tirumala, M. G., Sowjanya, T., Lokesh, D., Lokesh, E., & Sheel, S. G. (2024). Advanced signature identification and verification: using digital image processing and machine learning. *Journal of Engineering Sciences*, *15*(04), 1724–1735.

Widiawati, C. R. A., & Suliswaningsih, S. (2022). Analisa Hasil Perbandingan Poly Kernel Dan Normalisasi Poly Kernel Pada Support Vector Machine Sebagai Metode Klasifikasi Citra Tanda Tangan. *Jurnal Informatika*, *9*(1), 71–77. https://doi.org/10.31294/inf.v9i1.11288