CNN-based Baybayin Character Recognition on Android System

Edric Castel C. Hao
Electronics and Computer Engineering Department
De La Salle University
Manila, Philippines
edric_hao@dlsu.edu.ph

Galvin Brice S. Lim
Electronics and Computer Engineering Department
De La Salle University
Manila, Philippines
galvin_lim@dlsu.edu.ph

Dr. Melvin K. Cabatuan
Electronics and Computer Engineering Department
De La Salle University
Manila, Philippines
melvin.cabatuan@dlsu.edu.ph

Dr. Edwin Sybingco
Electronics and Computer Engineering Department
De La Salle University
Manila, Philippines
edwin.sybingco@dlsu.edu.ph

Dr. Ann Dulay
Electronics and Computer Engineering Department
De La Salle University
Manila, Philippines
ann.dulay@dlsu.edu.ph

***Abstract*—Baybayin is an old Philippine script that has become part of Filipino heritage. It was used by the indigenous people during the pre-colonial times but was later replaced with the Latin Alphabet during colonialization. Because of this, only some tribes are using it nowadays. However, recently the government has been reviving it as a way to connect to the past and preserve indigenous culture. That is why this paper proposed an improved CNN model for classifying the Philippine Script Baybayin in order to help people to learn this old script. The model was systemically developed through experimentation and hyperparameter tuning with MobileNetV2 as the baseline. Sample handwritten Baybayin characters were obtained from Kaggle and Mendeley Data for training. It was evaluated that the proposed model has training and validation accuracies of 97.04% and 96.02%, respectively, in contrast to the MobileNetV2, which has slightly lower accuracies and larger fluctuations in its validation accuracy. Aside from that, this model only has total parameters of 892,255 and a size of 3575.944 kB. Lastly, through mobile deployment testing, it was concluded that only the proposed model is working, as the MobileNetV2 was not able to accurately detect the diacritics.**

Keywords—Baybayin Character Recognition, CNN, Hyperparameter Tuning, TensorFlow Lite, MobileNetV2, Android

I. INTRODUCTION

Baybayin is an old Philippine script that was being used since the 16th century until the late 19th century, wherein it was replaced with the Latin alphabet. However, recently, it has been slowly being reintroduced by the Philippine government by using this script in documents like passports. As shown in Fig. 1, it contains 63 characters that include slight modifications of characters with diacritics [1]. While it has been promoted as a way to showcase the Filipino heritage, as it holds their cultural identity, most people nowadays do not actually know how to read such characters, especially due to the differences can often be subtle, which discourages people to use it. A slightly extra stroke or diacritic would change the character. Hence, this paper aims to address the issue by creating a character recognition model that can distinguish between the letters and provide the user with instant feedback on the Latin alphabet equivalent of the letter and, at the same time, help them learn this script.

Fig. 1. Baybayin characters

II. RELATED WORKS

Optical Character Recognition, short for OCR, is an AI algorithm that allows a computer to automatically recognize handwritten or printed writings and convert them into text data for further processing [2]. The main principle behind it is analogous to that of the human eyes recognizing some patterns from writing and decoding them into their corresponding character. From a computer perspective, this can be done by acquiring training images first and then subjecting them into preprocessing to remove noise and retain only the writing itself. Afterward, the input images are then split into a set of features that are of critical characteristics, such as horizontal, vertical, and diagonal lines, as well as curves and dots. With the extracted feature, these are then fed into a neural network for training in order to recognize the characters[2]. While there are already various OCRs for different languages available on the internet, one is the Cloud Vision API developed by Google, which includes popular languages like English, Filipino, and Chinese; unfortunately, there is still no official OCR for Baybayin that is developed [3]. However, there is recent interest in Baybayin character recognition and even word recognition.

One is the paper in [4], which essentially utilizes a Convolutional Neural Network (CNN) composed of 32-, 64-, and 128-channel 3x3 filter convolutional layers with a 2x2 max pooling layer after each convolutional layer and then two fully connected dense layers. Here, 7000 handwritten Baybayin characters, including all the 63 characters, were used for training, and it was concluded that the optimal model developed in the paper was able to achieve a 94% validation accuracy. Moreover, it was recommended in the paper that a mobile app must be developed wherein the users can write a character and automatically get a classification result in order to aid people in learning the Baybayin script and fully re-introduce this old script which embedded the identity of the Filipino to the public [4].

A similar study was also done in [5], but this time, the paper utilized the VGG16 model, and it was able to obtain a higher training accuracy of 99.54% and testing accuracy of 98.84% using a 1080P camera to capture an image of the character. However, the limitation of the model is that it is only trained for 45 Baybayin characters only out of the 63 characters.

In another paper [6], the authors introduced two models, namely the Feed-Forward Neural Network with Dropout Method (FFNNDM), which is mainly composed of one (1), four (4), and another one (1) dense layers for the input, hidden, and output layers respectively, whereas the Convolutional Neural Network with Dropout Method (CNNDM) is basically made up of three (3) convolutional layers, followed by two (2) dense hidden layer and another one (1) output layer. From these two models, it was determined that the former model has a higher accuracy of 92.4% compared to the later model which only has an accuracy of 91.69%.

III. METHODOLOGY

A. *Preparation of Datasets*

In the development of the model, two Baybayin character image datasets were utilized for the training in order to have larger training samples. One is the Baybayin Handwritten Images dataset with 9833 sample images in Kaggle [7], and the other is the Handwritten Baybayin Symbols Dataset with 36000 sample images in Mendeley Data [8]. Both datasets have an imbalanced number of samples for each unique Baybayin character or class. Fig. 2 shows a sample of the Baybayin datasets.

Particularly, it is noted that only the former dataset includes Baybayin characters with diacritics; hence, have a total of 63 classes, in contrast to the latter dataset, which only involves the main character with a total of 17 classes, but it has a larger number of samples for each class. Fig. 3 presents the statistics of the overall dataset obtained, and it can be seen that there is a severe imbalance in the data, which can be detrimental to the training as the model will become more biased toward the classes that cover the majority of the data [9]. Hence, in order to resolve this problem, the classes with a low number of samples were upsampled by generating duplicate copies of the sample images. On the other hand, the classes with large samples were downsampled by only using some of the samples randomly. Following the typical 80:20 training and validation sample ratio, the authors decided to use 400 sample images for the training and 100 for the validation. Moreover, for simplicity, only the ‘ɛ’ and ‘ɔ’ equivalent of the characters were considered for labels of the main character with dots above and below, respectively.

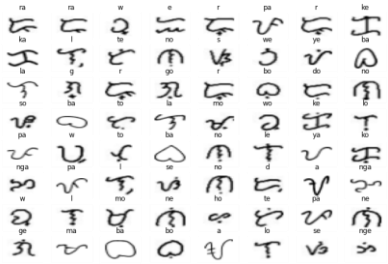

Fig. 2. Dataset samples.

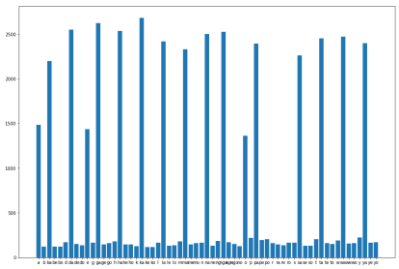

Fig. 3. Statistics of the dataset samples for each unique Baybayin character.

B. *Preprocessing of Datasets and Data Augmentations*

The Baybayin character recognition is essentially dealing with the horizontal, vertical, and diagonal lines as well as curves. The input image is converted to grayscale, as RGB information is not of concern, and the grayscale pixel is also inverted, similar to the MNIST, in order to have most of the pixels be black, that is, a pixel value of 0. To ensure that the training network will converge fast, the images are normalized to a range of 0 to 255 with a float 32 data type.

Data Augmentation layers at the input of the model are employed to cover the non-uniformity of size and position of human handwriting. It would also cover slightly slanted or lighter handwriting. Data Augmentation also increases the training and validation datasets. Given this idea, a Random Zoom layer with both height and width factor of -5% to 50%, a Random Rotation layer of about ±30°, a Random Transliteration layer with both height and width factor of -10% to 10%, and a Random Contrast layer with a factor of 0.1 are added to the model. However, prior to this, in order to ensure that the characters of the dataset samples would not be clipped after data augmentation, the image is first converted to a 23x23 size and is then padded to a 32x32 pixel image. Fig. 4 presents sample character images after data augmentations. Lastly, the labels were then converted to one-hot encoding.

C. *Modeling and Hyperparameter Tuning*

The training and validation data for Baybayin character recognition are both applied to MobileNetV2 and the proposed model. Fig. 5 presents the MobileNetV2 model. The implementation of the MobileNetV2 code is adapted from [10]. The proposed model is presented in Fig. 6. With the proposed based model, the hyperparameters of the model were then optimized. This was done by foremost using the Keras Tuner to tune the number of units and the dropout rate of both dense layers as well as the type of optimizer and its corresponding learning rate. This was done as the Keras Tuner has a faster tuning algorithm, as it does not train each model at once, but rather it trains a fraction of the total epochs first and then decides on which model to further train. For the optimizer, only the Adam and RMSprop were tried as both of these are deemed to be superior, especially since they can have a varying learning rate depending on the features. From

Fig. 4. Training samples after data augmentations

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Fig. 5. MobileNetV2 model [12].

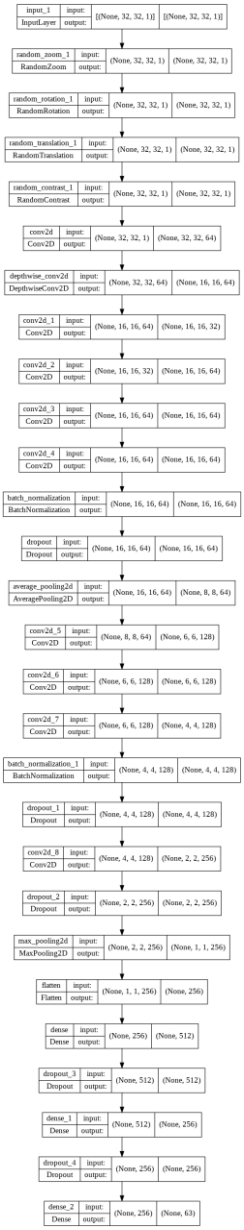

Fig. 6. Proposed model.

the resulting tuned model, the optimal model was then selected and further tuned with the Hparams dashboard. Here, the type of optimizer and its learning rate was again tuned, but this time together with the batch size. The top three (3) models with the highest accuracy were then selected and further evaluated.

D. Training and Evaluation

With the MobileNetV2 model and the three (3) candidate models obtained from the Hyperparameter Tuning, these models are compiled using Categorical Crossentropy loss function and accuracy metric and then fully trained according to the resulting optimal hyperparameters obtained for each model with 50 epochs. The training vs. validation accuracy and loss plots are obtained for initial evaluation. The confusion matrix and the classifcation report of both the MobileNetV2 and the final optimized model are further evaluated, including their model size and number of parameters.

E. Mobile Deployment

After designing and optimizing the model, both the MobileNetV2 and the proposed model are deployed to an Android system. In order to do this, the Tflite MNIST android application developed by [11], as shown in Fig. 7, is adapted. Sample inferences are obtained and the real performance of the model is assessed.

IV. Results and Discussion

A. Model Hyperparameter Tuning Results

Following the proposed systematic methodology in developing a model and tuning its hyperparameters, namely the number of units and the dropout rate of the two dense layers as well as the optimizer and learning rate to be used, the KerasTuner was utilized, Table I presents the top three (3) model with the highest validation accuracy. Particularly, as can be seen in the results, an increasing number of units for the two dense layers, which is not the usual case for a CNN model, coincidentally have a better performance. In addition to that, an equal number of units for the two dense layers, that is, having less trainable parameters in the overall model as well, also results in a higher validation accuracy in contrast to the usual model of having a narrowing network after the CNN feature extraction. The rationale here is that with the high-level features, it is a convention to reduce the number of neurons by decreasing the units of each dense layer and reaching the target number of outputs or classes.

In order to confirm the performance, both models were fully trained with 50 epochs, as shown in Fig. 8 and Fig. 9. From the figures, it can be seen that the model does not have satisfactory training results. For the model with 256- and 512-unit dense layers, the training accuracy reached 95.74%, its validation accuracy is fluctuating over different batches of images with an accuracy of as low as 61.90% and as high as 95.22%. The same performance can be seen in the model with 256- and 256-unit dense layers, but this time with lesser fluctuations in the validation accuracy ranging from 89.83% to 95.05%. Similarly, its training accuracy was also around 95.59%. Third model with 512- and 256-unit dense layers, respectively, produces better performance with higher training and validation accuracies as well as fewer fluctuations in the validation accuracy.

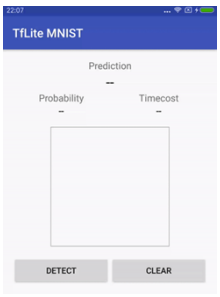

Fig. 7. Character recognition application

TABLE I. HYPERPARAMETER TUNING RESULTS USING KERASTUNER

Hyperparameters	Model		
	Top 1	***Top 2***	***Top 3***
Units1	256	256	512
Dropout1	0.1	0.05	0.05
Units2	512	256	256
Dropout2	0.15	0.1	0.15
Optimizer	RMSprop	adam	adam
Learning Rate	0.0001	0.001	0.001
Tuner/Epochs	30	30	30
Score	0.936349213	0.935079336	0.923968256

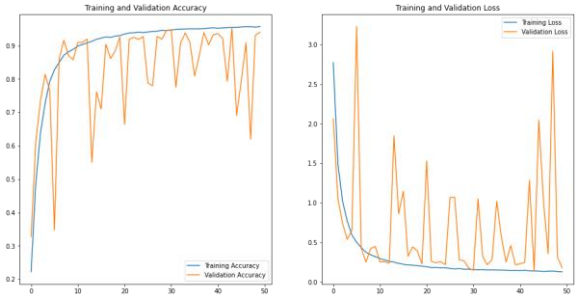

Fig. 8. Model with 256 and 512-unit dense layers, respectively.

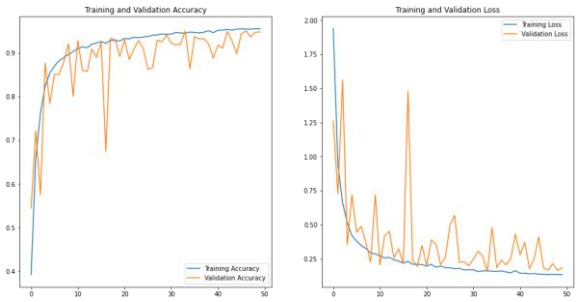

Fig. 9. Model with 256 and 256-unit dense layers, respectively.

With the base model being developed, it was optimized with the utilization of the Hparams dashboard. Specifically, the type of optimizer and its corresponding learning rate was again tuned together with the batch size in an attempt to reduce the fluctuations in the validation accuracy and, at the same time, increase the training and validation accuracy. It was found out that the root cause of the fluctuations in the validation accuracy is due to the dynamic of the datasets, especially given that data augmentations were applied. Hence, the batch size is varied in order to cover a larger percentage of the overall datasets and minimize the mini-batch stochastic gradient descent (SGD) from wandering aggressively. Furthermore, adaptive optimizers, namely Adam and RMSprop, were also tried as both of these can have a varying learning rate with respect to the features Fig. 10 presents the results of the hyperparameter tuning. Here, it was noted that the Adam optimizer with a learning rate of 0.0001 and batch size of 16 was able to achieve the highest accuracy, followed by the Adam optimizer with a learning rate of 0.001 and batch size of 64, and then the RMSprop optimizer with a learning rate of 0.0001 and batch size of 16.

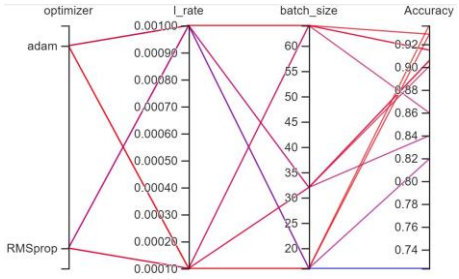

Fig. 10. Hyperparameter tuning results with Hparams dashboard

B. *Training Results*

With the three (3) candidate models obtained from the hyperparameter tuning, the MobileNetV2 was utilized as a reference for comparison. Fig. 11 presents the training vs. validation accuracy and loss plots of the MobileNetV2. From the figure, it can be seen that the model only was able to achieve a training accuracy of 94.65% after 50 epochs of training. Moreover, the validation accuracy also has a large fluctuation ranging from as low as 76.21% to 92.11%, considering the last ten (10) epochs. In fact, there is high overfitting manifested in the model with around a 2.5% difference between the training and validation accuracies. This can be primarily attributed to the complexity of the MobileNetV2 model itself, which has more than 2.3 million parameters; hence, it requires long training. In addition to that, this model is primarily used for large images with a typical size of 224 x 224 and RGB color, in contrast to the image of interest, which is only 32x32 and grayscale. Lastly, MobileNetV2 is designed for image classification, which extracts spatial, shape, and color features, whereas, for character recognition, it primarily focuses on the lines and curves of the writing as well as the presence of diacritics in the case of Baybayin characters. Due to the large number of parameters need by MobileNetV2, the proposed model has a more simplified model with lesser parameters and direct connection of layers, in order words, no skipping of layers in between, as illustrated in Fig. 6.

Given that, the three candidate models were trained with 50 epochs as well, and the training results are presented in Fig. 12, Fig. 13, and Fig. 14. From the figures, it can be seen that the first candidate model indeed achieves a higher accuracy with 97.05% for the training and around 95% to 96.02% for the validation but with an outlier accuracy of as low as 90.11% in the last ten (10) epochs. On the contrary, the second candidate model was only able to obtain a training accuracy of 95.84% and validation accuracy ranging from 86.60% to 95.08% and slightly lower for the third candidate model with 94.02% and 82.90% to 94.84%, respectively. Nevertheless, all these three (3) models exhibit lesser overfitting compared to

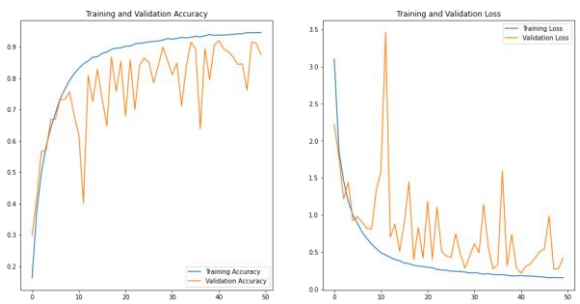

Fig. 11. Training vs. validation accuracy and loss plots for the MobileNetV2 model.

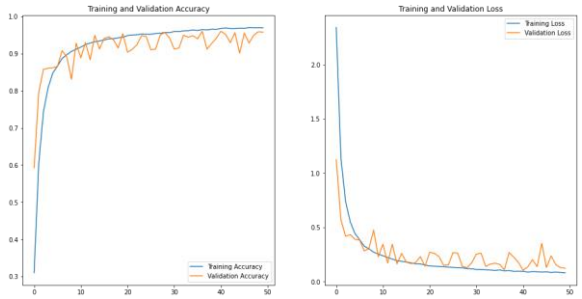

Fig. 12. Training vs. validation accuracy and loss plots for the proposed model.

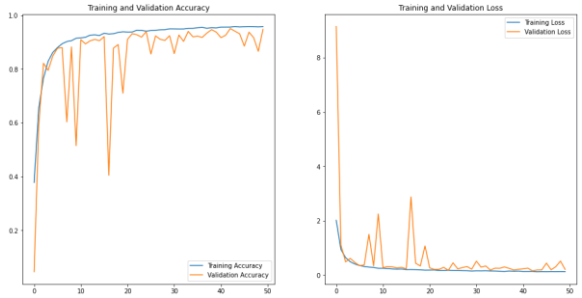

Fig. 13. Training vs. validation accuracy and loss plots for the 2nd candidate model

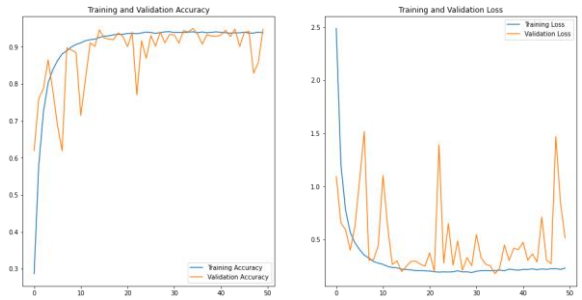

Fig. 14. Training vs. validation accuracy and loss plots for the 3rd candidate model.

the MobileNetV2. Therefore, the model using Adam optimizer with a learning rate of 0.0001 and batch size of 16 was deemed as the optimal model for Baybayin character recognition and was further analyzed and deployed to an Android system, which will be discussed later.

Moving on, Fig. 15 and Fig. 16 present the confusion matrix of the MobileNetV2 and the chosen proposed model. From these two figures, it can be clearly seen that the latter model performed better compared to the former, as it was able to recognize most of the characters correctly, as denoted by the degree of blue shade in the diagonal line. Delving into it further, it was determined that the proposed model has a higher accuracy of 95.76%, whereas the MobileNetV2 only has an accuracy of 87.65%. That is why it is deemed that the proposed model outperforms MobileNetV2.

C. Mobile Deployment

Afterward, the MobileNetV2 and the proposed models were then deployed to a mobile device, specifically to an Android system. In order to do this, the models were foremost converted to a tflite model—an optimized version of the model for mobile applications. Table II summarizes the specifications of these two models. Particularly, it can be seen that the proposed model only has 892,255 parameters, which

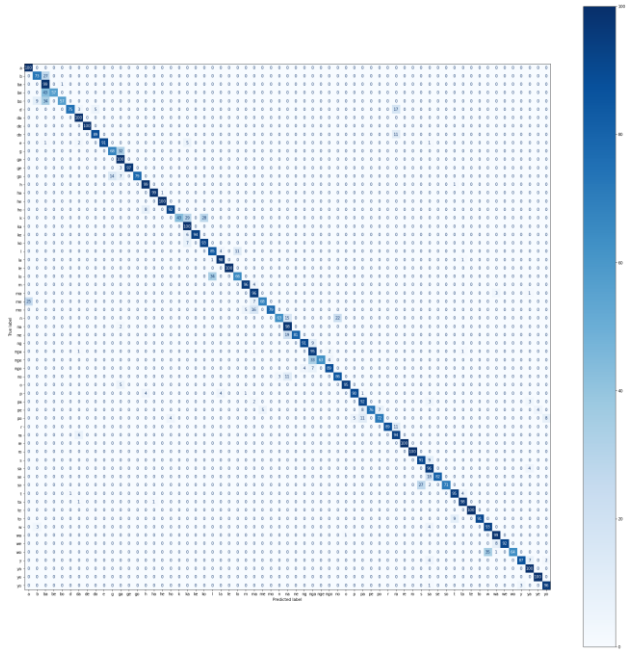

Fig. 15. Confusion matrix of the MobileNetV2 model.

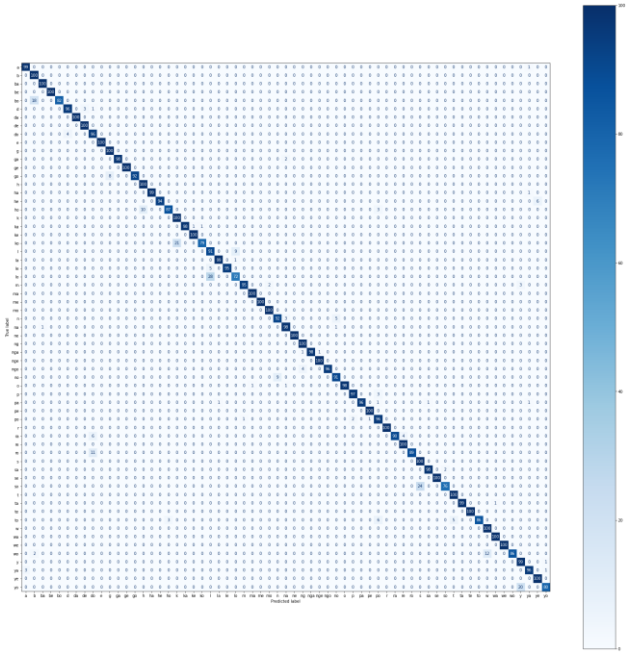

Fig. 16. Confusion matrix of the proposed model.

is about one-third of the size of the MobileNetV2. Hence, consequently, the model size is also reduced by one-third with a size of 3575.944 kB. Nevertheless, while the model is significantly smaller, it outperforms the MobileNetV2, as seen in their training and validation accuracies.

With that, Fig. 17 present some sample inferences obtained from the mobile deployment using both the proposed model and MobileNetV2. As illustrated in the figure, the proposed model indeed was able to successfully recognize the Baybayin characters, particularly it was able to distinguish the different equivalent characters with diacritics, like dots above or below the main character as well as plus sign or cross below the character. Moreover, as seen in the figure, even with varying sizes of the writing as well as orientation and position,

TABLE II. MOBILENETV2 VS. PROPOSED MODEL SPECIFICATIONS SUMMARY

Hyperparameters	Model	
	MobileNetV2	***Proposed Model***
Total Params	2,338,111	892,255
Training Accuracy	94.65%	97.04%
Validation Accuracy	92.11%	96.02%
Model Size	9178.852 kB	3575.944 kB
Inference Time	24~37 ms	<30 ms
Mobile Deployment Prediction (Diacritic Recognition)	NO	YES

(a) (b)

Fig. 17. Sample inferences obtained from mobile deployment using: (a) proposed model and (b) MobileNetV2.

it still was able to accurately recognize the character. This is mainly attributed to the data augmentation process integrated into the model training. Furthermore, it can be observed that the inference time of the model, including the application itself, denoted by the Time Cost, was essentially less than 30 ms.

Though it may seem that the MobileNetV2 will have comparable performance as it has a more complicated and larger model and even has a satisfactory confusion matrix, it can be seen in Fig. 17(b) that this was not able to produce accurate inferences, particularly it fails to recognize characters with diacritics. Moreover, it takes a longer inference time with an average time of around 24 to 37 ms. Again, this can be mainly attributed to the overcomplexity of the model for the problem itself, which instead produces a detrimental performance.

V. CONCLUSION

The model for Baybayin character recognition system that utilizes a simpler CNN model with a lesser number of parameters and direct connection between each layer and then undergoing a series of hyperparameter tuning, particularly the number of units and dropout rate of the two dense layers as well as the type of optimizer, learning rate, and the batch size is successfully developed. The Baybayin character recognition system has training and validation accuracies of 97.04% and 96.02%, respectively. Moreover, the proposed model only has total parameters of 892,255 and a size of 3575.944 kB, hence, making it more suitable for mobile deployment. With the reduction in the size of the model, the inference time also significantly decreases, making the system operate at a faster rate, especially given that a mobile system has limited computing power. In comparison with MobileNetV2, the proposed model exceeded the performance of MobileNetV2 where the model recognizes diacritics better and with better accuracy. For future studies, it is highly recommended to optimize Baybayin word recognition system with transliteration for mobile deployment in the hope that this can be utilized for the tourists to translate the Baybayin writings into artifacts during their museum tour as well as aid the archeologists in decoding Baybayin writings.

ACKNOWLEDGMENT

The authors of this paper would like to extend their sincere gratitude to their course professor, Dr. Melvin K. Cabatuan, and their research advisers, Dr. Edwin Sybingco and Dr. Ann Dulay, for imparting their knowledge on Deep Learning in Mobile Computing and guiding them throughout the research.

REFERENCES

[1] "Learning Baybayin: A Writing System From the Philippines | Baybayin, Filipino words, Meaningful word tattoos." https://www.pinterest.ph/pin/360499145173337709/ (accessed Jul. 08, 2022).

[2] "OCR Algorithm: Improve and Automate Business Processes – InData Labs." https://indatalabs.com/blog/ocr-automate-business-processes (accessed Jul. 13, 2022).

[3] "OCR Language Support | Cloud Vision API | Google Cloud." https://cloud.google.com/vision/docs/languages (accessed Jul. 13, 2022).

[4] J. A. Nogra, C. L. S. Romana, and E. Maravillas, "Baybáyin Character Recognition Using Convolutional Neural Network," *International Journal of Machine Learning and Computing*, vol. 10, no. 2, pp. 265–270, Feb. 2020, doi: 10.18178/IJMLC.2020.10.2.930.

[5] "Recognition of Baybayin (Ancient Philippine Character) Handwritten Letters Using VGG16 Deep Convolutional Neural Network Model," *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 9, pp. 5233–5237, Sep. 2020, doi: 10.30534/IJETER/2020/55892020.

[6] M. J. A. Daday, "Recognition of Baybayin Symbols (Ancient Pre-Colonial Philippine Writing System) using Image Processing," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 1, pp. 594–598, Feb. 2020, doi: 10.30534/IJATCSE/2020/83912020.

[7] "Baybayín (Baybayin) Handwritten Images | Kaggle." https://www.kaggle.com/datasets/jamesnogra/baybayn-baybayin-handwritten-images (accessed Jul. 13, 2022).

[8] M. J. Daday, "Handwritten Baybayin Symbols Dataset," vol. 1, 2020, doi: 10.17632/J6CGCFYS77.1.

[9] "What is Imbalanced Data | Techniques to Handle Imbalanced Data." https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/ (accessed Jul. 13, 2022).

[10] "Creating MobileNetsV2 with TensorFlow from scratch | by Sumeet Badgujar | Analytics Vidhya | Medium." https://medium.com/analytics-vidhya/creating-mobilenetsv2-with-tensorflow-from-scratch-c85eb8605342 (accessed Jul. 13, 2022).

[11] "GitHub - nex3z/tflite-mnist-android: MNIST with TensorFlow Lite on Android." https://github.com/nex3z/tflite-mnist-android (accessed Jul. 13, 2022).

[12] "MobileNetV2: Inverted Residuals and Linear Bottlenecks | by Paul-Louis Pröve | Towards Data Science." https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5 (accessed Jul. 13, 2022).