Android-Based Rice Variety Classifier (Arvac) Using Convolutional Neural Network

Erwil David A. Pasion, Joe G. Lagarteja

Abstract:— Rice is the world's most important cereal. It constitutes the world's guideline source of nourishment, being the essential grain for the planet's most significant populace. It is the staple nourishment and source of dietary vitality and protein. Within the current grain-handling frameworks, grain sort and quality are evaluated by mere visual review. This assessment handle is, be that as it may, dull and time-consuming. Decision-making capabilities can be genuinely influenced by a physical condition such as weakness in vision, the current mental state caused by predispositions and work weight, and working condition such as a disgraceful lighting condition. In this paper, an Android-based image recognition system using Convolutional Neural Network was used as a new method to classify and identify rice variety in terms of visual features such as size, color, shape, and texture of the seeds. Fifty (50) rice samples having eleven (11) rice varieties were used in the study for testing. Results in the study show an overall accuracy rate of 93.8%. Such high accuracy rate confirms that the Android-based Rice Variety Classifier can be used as a tool for classifying rice grains.

Index Terms:— Artificial Intelligence, Android, Classification, Convolutional Neural Network

1 Introduction

In recent years, the emergence of smartphones has changed the definition of mobile phones. The phone is no longer just a communication tool, but also an essential part of the people's communication and daily life. Various applications added unlimited fun in people's lives. Now, Android System in the electronics market is becoming more and more popular. Operations like listening to music, watching videos, tweeting and some others can be moved from computer to a phone. This much-inspired people to use Android system. Moreover, also, since it is an open source, plenty of Android-based mobile applications are created. Many have pondered on how to carry out the application of a machine learning algorithm on mobile phones for quality advancement [2] One machine learning algorithm widely used today is the Convolutional Neural Networks. These have a significant impact on image and pattern recognition. CNN's have provided an extra advantage which allows it to automatically learn the features from the training data which makes it especially potent for image recognition. It has become essential in recent times exhibiting many practical applications in agriculture, health, and security. Rice, in terms of total production and value, is the most important cereal in the world. It constitutes the world's guideline source of nourishment, being the fundamental grain for the planet's most significant populace. It is the staple nourishment and source of dietary vitality and protein. Today, more than a hundred rice varieties are identified in the Philippines [5].The quality of these different varieties varies considerably in size, color, shape, and texture. Within the current grain-handling frameworks, grain sort and quality are evaluated by mere visual review. This assessment handle is, be that as it may, dull and time-consuming. The decision-making capabilities of a grain examiner can be genuinely influenced by his/her physical condition such as weakness in vision, the current mental state caused by predispositions and work weight, and working conditions such as a disgraceful lighting condition, etc. As a result, proper identification of each variety has become a crucial matter, and fraudulent labeling of one variety as another is a primary concern in food and business industry.

- *Erwil David A. Pasion is from Aldersgate College, Neuva Vizcaya*
- *Joe G. Lagarteja is from Isabela State University, Echague IsabelaToday*

This paper proposes an Android-based image recognition system using Convolutional Neural Networks for rice seed variety identification which focuses on analyzing visual features such as size, color, shape, and texture of the seeds.

2 Materials and Methods

2.1 Requirements

The researchers had determined the requirements and the tools that are needed in the development of ARVAC. This stage is called the concept definition stage. The researcher has analyzed the app's specifications that represent all the app's inputs, processes, and outputs. The following steps were followed by the researcher in the development of ARVAC.

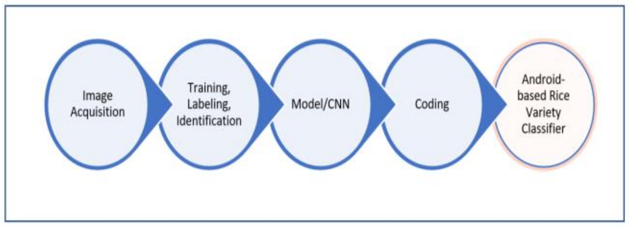
Image Acquisition
Training, Labeling, Identification
Model/CNN
Coding
Android-based Rice Variety Classifier

***Fig. 1**. ARVAC developmental process*.

There are five (5) steps in the developmental process of ARVAC. These are the following: (1) image acquisition, (2) training/labeling, (3) Model/CNN, (4) Coding and (5) the output which is the ARVAC.

2.2 Image Acquisition

The image acquisition system consisted of an Android smartphone (Samsung note 5/ SM9208) with a CPU of Octa-core (4x2.1 GHz Cortex-A57 & 4x1.5 GHz Cortex-A53) and GPU of Mali-T760MP8 that is hand-held and is operated in a controlled environment. A led light of nine (9) watts together with a coupon bond was also used.

2.3 Rough Rice Seed Samples

Three hundred (300) grams seed samples representing eleven (11) rice varieties that are popularly grown in the Philippines was obtained from the Department of Agriculture – Bureau of Plant Industry – National Seed Quality Control Services /

PhilRice. Twenty (20) grains in each variety were used for image analysis.

2.4 User Design
In this stage, the researcher systematically designed a mobile app layout that would meet the needs of the mobile application. The researcher focused on the simplicity of use of the mobile application. When a problem occurred, the researcher identified that problem and resolved it within Android Studio so that it would not prevent correct operation of the mobile application.

2.5 Build, Demonstrate and Refine
In this stage, the researcher systematically designed a mobile app layout that would meet the needs of the mobile application. The researcher focused on the simplicity of use of the mobile application. When a problem occurred, the researcher identified that problem and resolved it within Android Studio so that it would not prevent correct operation of the mobile application. The goal of the researcher was to take a new image that fell into a category of trained pictures and ran it through a command that would tell in which category the image fit. The researcher followed these steps: Training, Labelling, and Identification.

1. Labeling is the process of curing data on training. For rice, images of “NSIC RC 222” is dragged into the "NSIC RC 222" folder, “NSIC RC 360” into the "NSIC RC 360" folder, and so on, for as many different rice as desired. If “PSB RC 72 H” is never labeled, the classifier will never return "PSB RC 72 H." This requires many examples of each variety, so it takes time, and it is a necessary process.
2. Training is when the labeled data (images) is fed to the model. An instrument captures a random batch of images, uses the model to guess what type of rice is in each, tests the accuracy of the assumptions and repeats until most training data are used. The last batch of available images is used to calculate the effectiveness of the trained model.
3. Identification is using the model on new images. For example, input: IMG207.JPG, output: “NSIC RC 222”. This is the fastest, easiest and cheapest step.
   The CNN used in this project was implemented in Google’s TensorFlow using MobileNet version 1.0, which provided a high level of abstraction over TensorFlow.
   The CNN was trained on an ASUS PBH67-V computer that ran in Core i7-2600 CPU 3401 Mhz that took 10-20 minutes. The following script was used in the training process.

Since increased samples were used for input, they were highly correlated even if they increased the size of the dataset. This could have led to overfitting of the data. The overfitting problem has been solved by modulating the network's entropic capacity - the amount of information stored in the model that can store much information can be more accurate, but it also saves unnecessary features. In our case, we used a very trivial CNN with few layers and few filters per layer together with data increase and 0.5 dropouts. Dropout also helped reduce overfitting by preventing twice the same pattern from being seen by a layer. The model that stored less information therefore only saved the most important features found in the data and is likely to become morerelevant and more widespread.

2.6 Confusion Matrix
A confusion matrix is a table that is typically used to describe the performance of a classification model (or "classifier") on a set of test data known to the real values. A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class is the key to the confusion matrix. The confusion matrix shows how the classification model is confused when it makes predictions. It gives insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. Results of the test are calculated using a simple mathematical formula adapted from the work of Zhu (Zhu, Zeng, & Wang, 2010) wherein:

$$Accuracy = \frac{(TN + TP)}{(TN + TP + FN + FP)} = \frac{NUMBER\ OF\ CORRECT\ ASSESSMENT}{NUMBER\ OF\ ALL\ ASSESSMENTS}$$

WHERE: TP – True Positive, TN – True Negative, FN – False Negative, FP – False Positive
For example, we have,
- Class 1: Positive
- Class 2: Negative

Positive (P): Observation is positive (for example: is rice grain).
Negative (N): Observation is not positive (for example: is not a rice grain).
True Positive (TP): Observation is positive, and is predicted to be positive.
False Negative (FN): Observation is positive, but is predicted negative.
True Negative (TN): Observation is negative, and is predicted to be negative.
False Positive (FP): Observation is negative, but is predicted positive.
However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

3 Results and Discussions
The steps in the development of Android-based Rice Variety Classifier (ARVAC) is shown in Figure 2. After undergoing the process, result yields three (3) possible predictions; PSB RC 18 which has the confidence of 79.44, NSIC RC 222, which has the confidence of 13.16 and, NSIC RC 480 which has the confidence of 6.13. This means that the classifier can predict rice variety accurately. The process is similar to the study conducted by [15] where Convolutional Neural Networks greatly influenced pattern recognition. Features were manually picked, designed and then classified [16]. Were also able to design a framework using Convolutional Neural Networks for classification, localization, and detection. The process is effectively implemented by using a multiscale and sliding window approach.

3.1 Integration of CNN in the Mobile Application

- Camera Input The mobile application gets the camera input using the functions defined in the file CameraActivity.java. This file depends on AndroidManifest.xml to set the camera orientation. CameraActivity also contains code to capture user preferences from the UI and make them available to other classes via convenience methods.
- Classifier The file Classifier.java contains most of the complex logic for processing the camera input and running inference. Two subclasses of the file exist, in ClassifierFloatMobileNet.java and ClassifierQuantizedMobileNet.java, to demonstrate the use of both floating point and quantized models. The Classifier class implements a static method, create, which is used to instantiate the appropriate subclass based on the supplied model type (quantized vs. floating point).
- Load model and create an interpreterTo perform inference, we need to load a model file and instantiate an Interpreter. This happens in the constructor of the Classifier class, along with loading the list of class labels. Information about the device type and a number of threads are used to configure the Interpreter via the Interpreter. Options instance passed into its constructor. Note how that in the case of a GPU is available, a Delegate is created using GpuDelegateHelper. Pre-loading and memory mapping the model file offers faster load times and reduce the dirty pages in memory. The method loadModelFile does this, returning a MappedByteBuffer containing the model. The MappedByteBuffer is passed into the Interpreter constructor, along with an Interpreter. Options object. This object can be used to configure the interpreter, for example by setting the number of threads (.setNumThreads(1)) or enabling NNAPI (.setUseNNAPI(true)).
- Pre-process bitmap image Next, in the Classifier constructor, we take the input camera bitmap image and convert it to a ByteBuffer format for efficient processing. We pre-allocate the memory for the ByteBuffer object based on the image dimensions because Bytebuffer objects cannot infer the object shape. The ByteBuffer represents the image as a 1D array with three bytes per channel (red, green, and blue). We call to order(ByteOrder.nativeOrder()) to ensure bits are stored in the device's native order. The code in convertBitmapToByteBuffer pre-processes the incoming bitmap images from the camera to this ByteBuffer. It calls the method addPixelValue to add each set of pixel values to the ByteBuffer sequentially. In ClassifierQuantizedMobileNet, addPixelValue is overridden to put a single byte for each channel. The bitmap contains an encoded color for each pixel in ARGB format, so we need to mask the least significant 8 bits to get blue, and next 8 bits to get green and next 8 bits to get blue. Since we have an opaque image, alpha can be ignored. For ClassifierFloatMobileNet, we must provide a floating point number for each channel where the value is between 0 and 1. To do this, we mask out each color channel as before, but then divide each resulting value by 255.f.
- Run inference The method that runs inference, runInference, is implemented by each subclass of Classifier. The output of the inference is stored in a byte array labelProbArray, which is allocated in the subclass's constructor. It consists of a single outer element, containing one inner element for each label in the classification model. To run inference, we call run() on the interpreter instance, passing the input and output buffers as arguments.
- Recognize image Rather than call runInference directly, the method recognizeImage is used. It accepts a bitmap, runs inference, and returns a sorted List of Recognition instances, each corresponding to a label. The method will return several results bounded by MAX_RESULTS, which is three(3) by default. Recognition is a simple class that contains information about a specific recognition result, including its title and confidence. A PriorityQueue is used for sorting. Each Classifier subclass has a getNormalizedProbability method, which is expected to return a probability between 0 and 1 of a given class being represented by the image.
- Display results The classifier is invoked, and inference results are displayed by the processImage() function ClassifierActivity.java. ClassifierActivity is a subclass of CameraActivity that contains method implementations that render the camera image, run classification, and display the results. The method processImage() runs classification on a background thread as fast as possible, rendering information on the UI thread to avoid blocking inference and creating latency. Another important role of ClassifierActivity is to determine user preferences (by interrogating CameraActivity) and instantiate the appropriately configured Classifier subclass. This happens when the video feed begins (via onPreviewSizeChosen()) and when options are changed in the UI (via onInferenceConfigurationChanged()).
- TensorFlow-Android AAR This app uses a pre-compiled TFLite Android Archive (AAR). This AAR is hosted on jcenter. The following lines in the module's build.gradle file include the newest version of the AAR, from the TensorFlow bintray maven repository, in the project. We use the following block, to instruct the Android Asset Packaging Tool that .lite or .tflite assets should not be compressed. This is important as the .lite file will be memory-mapped, and that will not work when the file is compressed.
- Using theTFLite Java APIThe code interfacing to the TFLite is all contained in ImageClassifier.java.
- Run the model This method does three things. First, it converts and copies the input Bitmap to the imgData ByteBufferinput to the model. Then it calls the interpreter's run method, passing the input buffer and the output array as arguments. The interpreter sets the values in the output array to the probability calculated for each class. The input and output nodes are defined by the arguments to the toco conversion step that created the .lite model file earlier. During the inception of the mobile app, the output of the trained images in training, labeling and the identifying process is two files, the graph.pb, and labels.txt. These files are required by the app as these are the trained images that are being mapped by the mobile app. [16] in their study presented a Novel deep Learning Approach to localization by learning to predict boundaries where bounding boxes were gathered rather than concealed for it to be recognized better. [7] In their study proposed a method for identifying and segregating text in complex images and videos. Text lines were identified using complex-valued multilayer feed-forward network designed to detect text at a fixed scale and position. The output of the network is integrated into a single-saliency map, serving as a starting point for possible text lines.

3.2 Test the Mobile Application's Accuracy in Identifying Rice Variety

Rice grain kernels of five (5) varieties namely; NSIC RC 152, NSIC RC 216, NSIC RC 218, NSIC RC 222, NSIC RC 358,

Rice Variety	*Rice Samples*	*TP*	*TN*	*FP*	*FN*	*Accuracy*
NSIC RC 152	*50*	*45*	*0*	*3*	*0*	*90%*
NSIC RC 216	*50*	*46*	*0*	*7*	*0*	*92%*
NSIC RC 218	*50*	*47*	*0*	*6*	*0*	*94%*
NSIC RC 222	*50*	*48*	*0*	*2*	*0*	*96%*
NSIC RC 358	*50*	*46*	*0*	*15*	*0*	*92%*
NSIC RC 360	*50*	*47*	*0*	*10*	*0*	*94%*
NSIC RC 400	*50*	*46*	*0*	*4*	*0*	*92%*
NSIC RC 480	*50*	*48*	*0*	*7*	*0*	*96%*
PSB RC 18	*50*	*42*	*0*	*12*	*0*	*84%*
PSB RC 72 H	*50*	*47*	*0*	*11*	*0*	*94%*
TG 102 M	*50*	*48*	*0*	*2*	*0*	*96%*

NSIC RC 360, NSIC RC 400, NSIC RC 480, PSB RC 18, PSB RC 72 H, TG 102 M were taken up for classification. TABLE 1 Classification Results

Table 1 shows that forty-five (45) out of fifty (50) NSIC RC152 were classified correctly, forty-six (46) out of fifty (50) NSIC RC216, forty-seven (47) out of 50 NSIC RC218, and forty-two (38) out of fifty PSB RC 18 and PSB RC 72H respectively. This points to an overall accuracy rate of 93.8%. Confusion and misclassifications were attributed to lighting condition; this is also the case in the study of (Tan & Triggs, 2010).

4 Conclusion

The mobile app that was developed was based on the Rapid Application Development (RAD) model which consists of different phases such as Analysis and Quick Design, Testing and Implementation. The manual procedure for rice classification was studied to determine the requirements needed for the application. With the information obtained, ARVAC was designed. Furthermore, the developmental tools that have been used are Android Studio, google's TensorFlow and a smartphone that has a built-in camera. The training of the neural network was accomplished based on Figure 5. The training of the model took sixty-four hours. Based on the results, Android-based Rice Variety Classifier (ARVAC) using Convolutional Neural Network has an overall accuracy of 93.8% which leads to a 6.2% loss in rice image prediction. This means that the mobile app developed was good at classifying rice variety due to its high accuracy prediction. Based on the study, the following conclusions were drawn:
1. A mobile application developed can classify eleven (11) rice kernel varieties based on color and texture.
2. A Convolutional Neural Network specifically google's MobileNet version 1.0 is used in the integration of the mobile app. A machine vision system together with the developed neural network architectures could be used as a tool to achieve a more reliable, accurate and objective assessment of the rice quality at trading points in the rice marketing system with comparatively high classification accuracy.
3. Based on the result of the study, in the test dataset, the total classification accuracies were 93.8% which indicate high accuracy rating.

Acknowledgment

The authors wish to thank Aldersgate College and Isabela State University for the support of this study.

References

[1] And, N. C., & Deshmukh H R. (2012). ANDROID OPERATING SYSTEM. Software Engineering. https://doi.org/http://en.wikipedia.org/wiki/Android_(operating_system)
[2] Azizi, A., Abbaspour-Gilandeh, Y., Nooshyar, M., & Afkari-Sayah, A. (2015). Identifying Potato Varieties Using Machine Vision and Artificial Neural Networks. International Journal of Food Properties, 19(3), 618–635.

https://doi.org/10.1080/10942912.2015.1038834
[3] Bennett, J. K., Wheatley, J. K., & Walton, K. N. (1984). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning arXiv:1602.07261v2. Journal of Urology. https://doi.org/10.1007/s10236-015-0809-y
[4] Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): An empirical review. European Journal of Information Systems, 8(3), 211–223. https://doi.org/10.1057/palgrave.ejis.3000325
[5] Department of Agriculture, P. R. R. I. (n.d.). Pinoy Rice Knowledge Bank. Retrieved October 26, 2018, from http://www.pinoyrice.com/rice-varieties/
[6] Developers, A. (2014). Android, the world's most popular mobile platform. Android, the World's Most Popular Mobile Platform | Android Developers.
[7] Eggert, C., Brehm, S., Winschel, A., Zecha, D., & Lienhart, R. (2017). A closer look: Small object detection in faster R-CNN. In Proceedings - IEEE International Conference on Multimedia and Expo. https://doi.org/10.1109/ICME.2017.8019550
[8] Fei-Fei Li, J. J. and S. Y. (2016). CS 231N Course Slides and Notes", Stanford University. Retrieved from http://cs231n.stanford.edu/2017/syllabus
[9] Felt, A., Chin, E., & Hanna, S. (2011). Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security - CCS '11 (2011). https://doi.org/10.1145/2046707.2046779
[10] Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision. https://doi.org/10.1109/ICCV.2015.169
[11] Khush, G. S., Paule, C. M., & dela Cruz, N. M. (1979). Rice grain quality evaluation and improvement at IRRI. In In: Proceedings of Workshop in Chemical Aspects of Rice Grain Quality. Int. Rice Res. Inst., Los Baños, Laguna, Philippines.
[12] Kuo, T. Y., Chung, C. L., Chen, S. Y., Lin, H. A., & Kuo, Y. F. (2016). Identifying rice grains using image analysis and sparse-representation-based classification. Computers and Electronics in Agriculture, 127, 716–725.

https://doi.org/10.1016/j.compag.2016.07.020

[13] Liu, Z., Cheng, F., Ying, Y., & Rao, X. (2005). Identification of rice seed varieties using neural network. Journal of Zhejiang University SCIENCE. https://doi.org/10.1631/jzus.2005.b1095

[14] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. https://doi.org/10.1109/TPAMI.2016.2577031

[15] Sakr, G. E., Mokbel, M., Darwich, A., Khneisser, M. N., & Hadi, A. (2016). Comparing deep learning and support vector machines for autonomous waste sorting. In 2016 IEEE International Multidisciplinary Conference on Engineering Technology, IMCET 2016. https://doi.org/10.1109/IMCET.2016.7777453

[16] Sermanet, P., & LeCun, Y. (2011). Traffic sign recognition with multi-scale Convolutional Networks. In Neural Networks (IJCNN), The 2011 International Joint Conference on. https://doi.org/10.1109/IJCNN.2011.6033589

[17] T.Vizhanyo, J. F. (2000). Enhancing color differences in image of diseased mushrooms,”. Computer and Electronics in Agriculture, 26, 187–198.

[18] Tan, X., & Triggs, B. (2010). Recognition Under Difficult Lighting Conditions. IEEE Transactions on Image Processing(TIP). https://doi.org/10.1109/TIP.2010.2042645

[19] Van Dalen, G. (2004). Determination of the size distribution and percentage of broken kernels of rice using flatbed scanning and image analysis. Food Research International. https://doi.org/10.1016/j.foodres.2003.09.001

[20] Zhu, W., Zeng, N., & Wang, N. (2010). Sensitivity, specificity, accuracy, associated confidence interval and ROC analysis with practical SAS® implementations. Health Care and Life Sciences. https://doi.org/10.1080/14759551.2011.530745