Journal of Computational Innovations and Engineering Applications 3(1) 2018: 38–43

Real-Time Vehicle Classification Using MobileNet

Reagan L. Galvez,[1,*] Melvin K. Cabatuan,[2] and Argel A. Bandala[2]

***Abstract*—classification is an important part of vision systems and has several applications like autonomous cars and surveillance. This is a challenging task because computers see images differently from humans. This paper used the MobileNet model for training the data and tested it on an Android device. This model is lightweight and efficient compared with previous developed models. This was inspired by the sample code from Google Codelabs. Experiment results show that the Android application can accurately classify the type of vehicle in real time.**

Keywords: convolutional neural network, deep learning, MobileNet, vehicle classification

I. Introduction

Intelligent transport systems (ITSs) are developed to provide safe travel and ensure effective transportation [1]. To implement this, the ITS needs access to data like the type of vehicle. This will help authorities to identify criminals quickly. It can also be used to apprehend violators like trucks [2], during certain periods of time (ex., rush hour) when trucks are not allowed on the road. Vehicle classification maybe a simple problem for a human, but for a computer, it's a complex problem. That's why many researchers are exploring image processing [3] to solve this problem with high accuracy by creating different architectures/models. There are many techniques that can be used to classify images like artificial neural networks, decision tree, support vector machine, and fuzzy measure [4]. Currently, convolutional neural networks (CNNs) are used in image classification because they provide accurate performance in computer vision tasks. There are many pre-trained CNN based models that can be used for image classification like AlexNet [5], VGG16, VGG19 [6], ResNet50 [7], InceptionV2, InceptionV3 [8], Xception [9], and DenseNet [10]. On the other hand, models like R-CNN [11], Fast R-CNN [12], YOLO [13], YOLO9000 [14], SSD [15], and MobileNet [16] are commonly used object classification and detection models. The focus of this paper is to utilize MobileNet for object classification using android platform.

This paper is organized as follows: section 2 discusses the concept about MobileNet and its architecture. Section 3 describes the experiment setup and the dataset used for training. Section 4 shows the performance of the MobileNet model in vehicle classification and its deployment in an Android platform.

Reagan L. Galvezla, Bulacan State University, Malolos City, Philippines (e-mail: reagangalvez@gmail.com)

Melvin K. Cabatuan and Argel A. Bandala, De La Salle University–Manila, Philippines

II. MobileNet

MobileNet is an efficient model designed for mobile and embedded vision applications [16]. It uses depthwise separable convolutions to build lightweight and efficient deep neural networks. The depthwise convolution applies a single filter to each input channel. To combine the output of depthwise convolution, pointwise convolution applies 1×1 convolution. The combination of two convolutions results to a depthwise separable convolution. Two layers are formed by depthwise separable convolution. The first layer is used for filtering, and the second layer is used for combining. This is called factorization, and as a result, it reduces the model size and computation.

Table 1 shows the MobileNet architecture. The model consists of 28 layers. These layers are combinations of alternating depthwise and pointwise convolutions. Every layer is followed by batch normalization and rectified linear unit (ReLu) [17] function except the fully connected layer which is followed by a softmax classifier that gives actual probabilities in each class. Batch normalization speeds up the training [18], and the ReLu function is 0 for negative values and grows linearly for positive values. The notations s1 and s2 are the number of strides. Stride controls how the filter convolves around an input volume.

TABLE 1
MobileNet Architecture

Layer	Type/Stride	Input Size
1	Conv /s2	224 × 224 × 3
2	Conv dw /s1	112 × 112 × 32
3	Conv /s1	112 × 112 × 32
4	Conv dw /s2	112 × 112 × 64
5	Conv /s1	56 × 56 × 64
6	Conv dw /s1	56 × 56 × 128
7	Conv /s1	56 × 56 × 128
8	Conv dw /s2	56 × 56 × 128
9	Conv /s1	28 × 28 × 128
10	Conv dw /s1	28 × 28 × 256
11	Conv /s1	28 × 28 × 256
12	Conv dw /s2	28 × 28 × 256
13	Conv /s1	14 × 14 × 256
14–23	Conv dw/ s1 Conv /s1	14 × 14 × 512 14 × 14 × 512
24	Conv dw /s2	14 × 14 × 512
25	Conv /s1	7 × 7 × 512
26	Conv dw /s2	7 × 7 × 1024
27	Conv /s1	7 × 7 × 1024
	Avg. Pool /s1	7 × 7 × 1024
28	FC /s1	1 × 1 × 1024
	Softmax /s1	1 × 1 × 1000

III. Experiment Setup

The effectiveness of MobileNet in vehicle classification was tested in an android application. It is a simple camera application that runs a TensorFlow image recognition program to identify vehicles. This was inspired by the code from "TensorFlow for Poets 2" of Google Codelabs [19]. TensorFlow mobile was used to run the MobileNet and integrated it to mobile application. The Anaconda Prompt was used to initiate command in training and testing the model.

The first step in training the data is to install dependencies like TensorFlow. This is an open-source library for high-performance calculation, which allows easy deployment in different platforms and supports deep learning applications [20]. Figure 1 shows the block diagram of the vehicle classification structure. Next is to collect the data that will be used for training. The dataset is composed of five types of vehicles such as pickup, SUV, sedan, van, and truck. These images came from ImageNet, an image database. Table 2 shows the number of images per category of vehicle.

TABLE 2
Dataset

Vehicle Type	Number of Images
Pickup	400
SUV	400
Sedan	400
Van	400
Truck	400
Total	**2000**

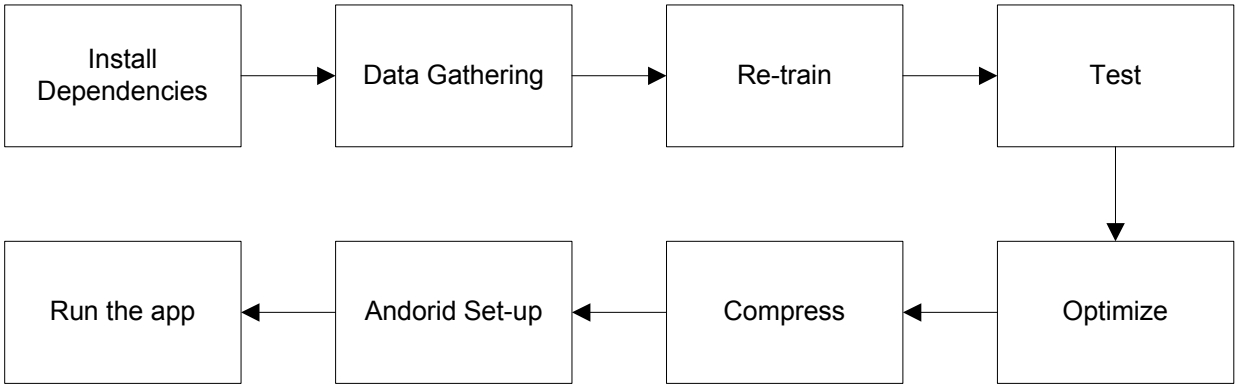

Fig.1. Block diagram.

Next is to re-train the data using the MobileNet model. The output of this is a `.pb` file; this is a single model file that contains graph variables frozen as constants. The input images can be configured using 4 different input sizes such as 128, 160, 192, and 224 pixels. The width multiplier (α) of the model can also be set such as 1.0, 0.75, 0.5, and 0.25. The width multiplier makes the model small and faster by reducing computational cost and number of parameters.

Then, test the model by using random image and compute its accuracy. After this, optimize the model by removing some nodes that are not needed in a given set of input and output. Next is to compress the model by quantizing the network weights to make it ideal for mobile applications. After compression, the model is ready to upload in an Android platform.

IV. RESULTS

A. Image Classification Performance

The performance of the MobileNet model was tested by calculating its validation accuracy, cross entropy, and evaluation time. Validation accuracy is the precision on a randomly selected group of images from a different set. Cross entropy is a loss function that shows how well the learning process is progressing. The ideal value of cross entropy is 0. Evaluation time is the time it takes to classify the test image. The total number of steps used in training the model was 2000 steps. This is enough to see if the model is learning. Table 3 shows the accuracy and cross entropy using different values of width multiplier (α) in MobileNet. It shows that when the MobileNet's width multiplier (α) decreases, the validation accuracy also decreases and the cross entropy increases.

TABLE 3
ACCURACY AND CROSS ENTROPY

Input Size	Width Multiplier (α)	Accuracy (%)	Cross Entropy
224	1.0	84.83	0.541196
224	0.75	84.36	0.554476
224	0.5	81.99	0.713274
224	0.25	72.04	2.38165

Figure 2 shows the training (orange) and validation accuracy (blue) during the training. This graph was from MobileNet using α = 1.0. As shown, the model was not learning anymore. In this case, the training can be stopped. The average validation accuracy is equal to 84.83%.

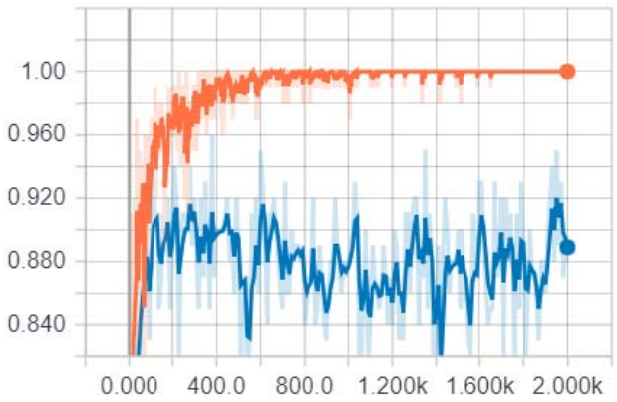

Fig. 2. Training steps vs. accuracy.

Figure 3 shows the training (orange) and validation cross entropy (blue) during the training. The ideal value of cross entropy is 0 because this is a loss function. The average cross entropy is equal to 0.541196.

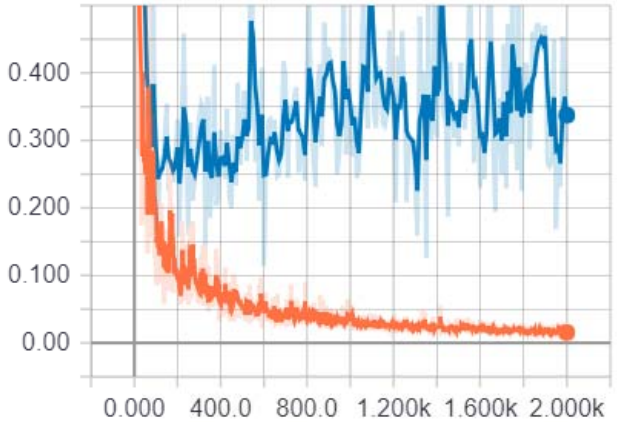

Fig. 3. Training steps vs. cross entropy.

Table 4 shows the time comparisons using different width multiplier (α) in the model. As the width multiplier (α) decreases, the time of evaluation also decreases. This is because the model became lightweight and has fewer computations.

Table 5 shows the model size comparisons for each different width multiplier (α) in the model. Using α = 1.0, there was 32.50% size reduction from the optimized to the compressed model. This compression can be useful in deploying the model for mobile applications because it can be downloaded easily.

Figure 4 shows the bubble chart of evaluation time versus accuracy. The size of the bubble is proportional to the size of the model. As we can see, there is a trade-off between accuracy and speed. If we need a faster model, α = 0.25 can be used, but the accuracy will decrease. For real-time application, α = 1.0 can still be used.

TABLE 4
Evaluation Time Comparisons

Input Size	Width Multiplier (α)	Time (Re-Trained) (s)	Time (Optimized) (s)	Time (Compressed) (s)
224	1.0	2.163	2.023	2.061
224	0.75	1.756	1.776	1.701
224	0.5	1.352	1.341	1.359
224	0.25	1.122	1.121	1.121

TABLE 5
Model Size Comparisons

Input Size	Width Multiplier (α)	Size (Re-Trained) (KB)	Size (Optimized) (KB)	Size (Compressed) (KB)
224	1.0	15,489	15,485	5,033
224	0.75	9,490	9,486	3,174
224	0.5	4,916	4,913	1,786
224	0.25	1,766	1,763	708

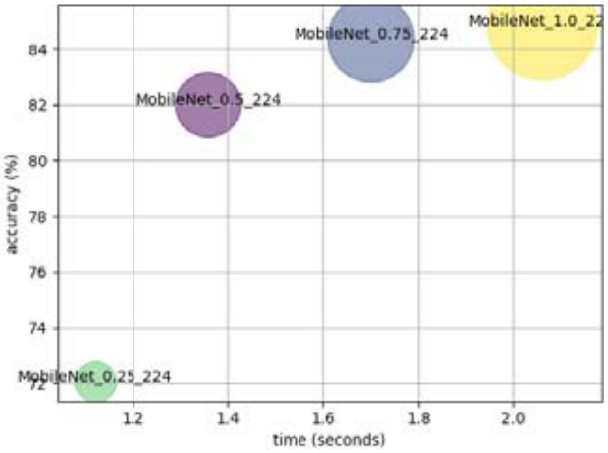

Fig. 4. Evaluation time vs. accuracy.

B. Deployment in Android Platform

The trained model was deployed in an Android platform to test its performance in real-time application. TensorFlow mobile was used to prepare the model for mobile deployment. Figure 5 shows the screenshot from the TensorFlow application using a Samsung Galaxy S7 Flat phone. The testing images were randomly selected to test if the application can detect unseen data. The actual images in the figure were both sedan and correctly identified as sedan.

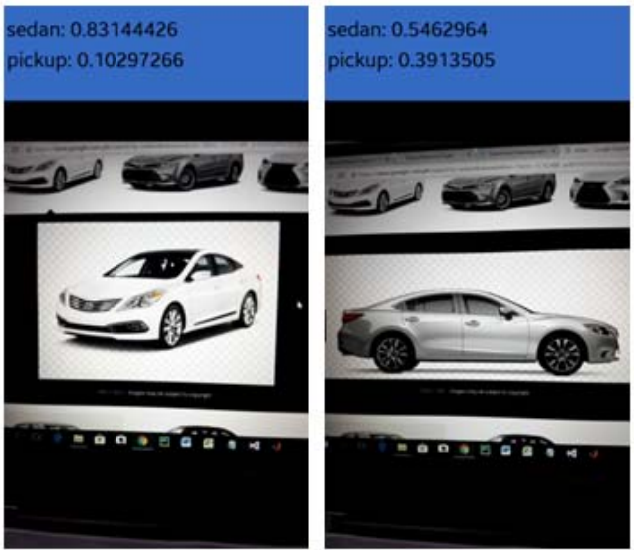

Fig. 5. Screenshot from TensorFlow application (sedan).

Another set of images was tested as shown in Figure 6. These images were a truck and correctly identified as truck by the TensorFlow application.

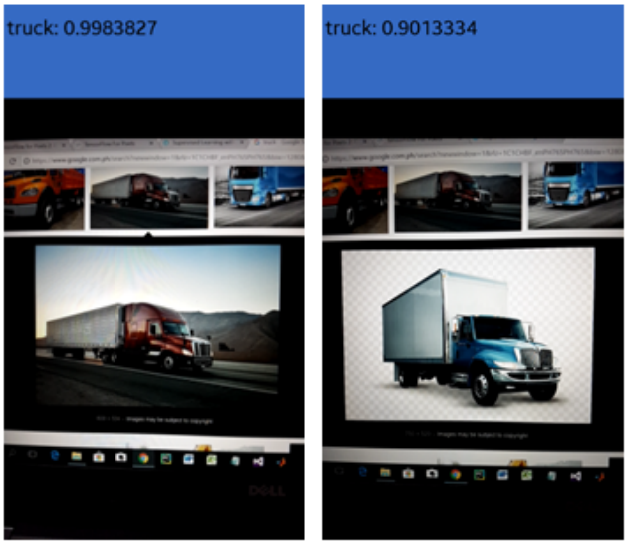

Fig. 6. Screenshot from TensorFlow application (truck).

Figure 7 shows an image of an SUV. The TensorFlow application failed to classify the first image as SUV; it was predicted as sedan. This is due to the similar features of an SUV and a sedan.

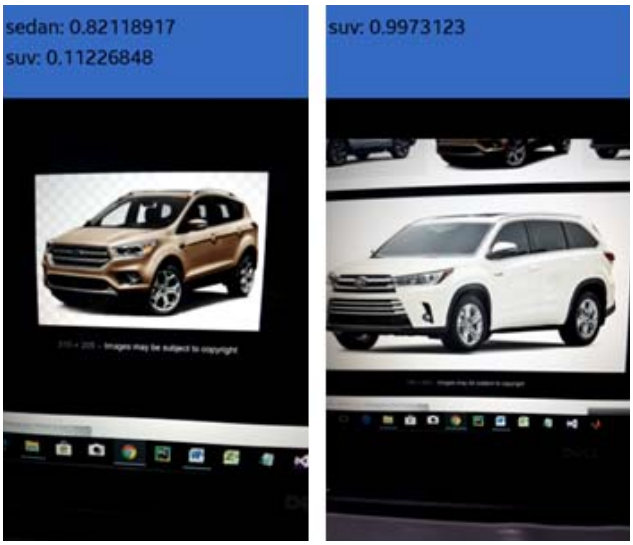

Fig. 7. Screenshot from TensorFlow application (SUV).

V. CONCLUSION

The implementation of real-time vehicle classification was successfully tested using the Android platform. The experiment result shows that the MobileNet can classify vehicle type up to 84.83% accuracy. TensorFlow mobile is a good deep learning solution for a mobile platform like Android. Although in testing an SUV image, the model cannot consistently classify the image, other types of vehicles were accurately identified. This is due to the physical structure of the SUV that is not totally different from the other types of vehicles because some SUVs are similar to a sedan, van, and pickup. As a solution, it is recommended to increase the dataset and remove some images that are confusing.

For future work, the application of MobileNet can be upgraded to track and identify vehicle type using CCTV videos in real-time.

ACKNOWLEDGMENTS

The author would like to thank the Department of Science and Technology—Engineering Research and Development for Technology (DOST-ERDT) and De La Salle University for the financial support while doing this research.

REFERENCES

[1] A. Ferdowsi, U. Challita and W. Saad, "Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems," *arXiv:1712.04135v1,* December 2017.

[2] R. K. C. Billones, A. A. Bandala, E. Sybingco, L. A. G. Lim and E. P. Dadios, "Intelligent system architecture for a vision-based contactless apprehension of traffic violations," *2016 IEEE Region 10 Conference (TENCON),* pp. 1871 - 1874, 2016.

[3] R. K. C. Billones, A. A. Bandala, E. Sybingco, L. A. G. Lim, A. D. Fillone and E. P. Dadios, "Vehicle Detection and Tracking using Corner Feature Points and Artificial Neural Networks for a Visionbased Contactless Apprehension Syste," *Computing Conference 2017,* pp. 688-691, 2017.

[4] P. Kamavisdar, S. Saluja and S. Agrawal, "A Survey on Image Classification Approaches and Techniques," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. II, no. 1, pp. 1005-1009, January 2013.

[5] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition," in *ICLR*, 2015.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference Publication,* pp. 1-12, 2015.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842v1,* pp. 1-12, September 2014.

[9] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *IEEE Conference Publication,* pp. 1-8, 2017.

[10] G. Huang, Z. Liu, L. Van der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[11] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv:1311.2524v5 [cs.CV],* pp. 1-21, 2014.

[12] R. Girshick, "Fast R-CNN," *arXiv:1504.08083v2 [cs.CV],* 2015.

[13] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once:Unified, Real-Time Object Detection," *arXiv:1506.02640v5 [cs.CV],* pp. 1-10, 2016.

[14] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv:1612.08242v1 [cs.CV],* pp. 1-9, 2016.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," *arXiv:1512.02325v5,* pp. 1-17, 2016.

[16] A. G. Howard, M. Zhu, B. Chen and D. Kalenichenko, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision," 2017.

[17] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010.

[18] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167v3,* pp. 1-11, March 2015.

[19] Codelabs, Google, "Tensorflow for Poets 2," December 2017. [Online]. Available: https://github.com/googlecodelabs/tensorflow-for-poets-2.

[20] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: A system for large-scale machine learning," *arXiv:1605.08695v2,* pp. 1-18, May 2016.