Abstract
In this paper I developed an android application to classify waste into recyclable and not recyclable. I retrained the MobileNetV2 model originally trained on the ImageNet dataset. I used the TrashNet dataset that consist of 2527 images classified into glass, paper, cardboard, plastic, metal and trash. The retrained model has a tendency to overfit. To reduce overfitting, I augmented the data, used L2 regularization and used dropout. Then, I converted the retrained model into a TensorFlowLite model and incorporated it into the application. The compressed model successfully achieved 99% training accuracy with a loss of 0.02 and an 80% testing accuracy with 1.0 loss. The mobile application's size is 42.11MB and uses, on average, 100KB of memory/hour. It takes 0.3245 seconds to classify and image.

Introduction

Recycling is an essential part of earth conservation efforts. Oftentimes, individuals attempting to recycle are confused on what is and isn't recyclable. A simple android application that allows users to take a picture of an object, then shows them whether an object is recyclable, could go a long way in helping those interested in recycling but confused about how to do it.

One of the challenges that face creating a waste classification system is the absence of a large enough dataset that could be used to obtain high accuracy. To compact this challenge, I'm using transfer learning to re-train an already exiting image classification model.

Another challenge is that waste sorting is a complicated process. For example, a pizza cartoon is recyclable; but if the cartoon has oil stains, it becomes unrecyclable. For this paper, I'm using a simple model that assumes that all cardboard, glass, metal, paper and plastic are recyclable.

Related Work

Gary Thung, and Mindy Yang from Stanford University created a trash classifier (2016). They built a dataset named TrashNet that composes of 2527 waste images and classified it using an AlexNet-like architecture. It's composed of 7*7 convolutional, Dropout, Fully connected layer, and log softmax. The network also used Adam gradient descent optimization. The authors ran 50 epochs of training with a 0.0075 learning rate and batch size of 25 with a 70/30 training/testing split and achieved a 23% training accuracy and 27% test accuracy.

Another attempt used ResNet34 architecture on the same dataset described above(Ching, 2019). The developer used learning rate of 5.13e-03 and 20 epochs. They achieved a 92.1% test accuracy and 91.4% validation accuracy.

There have been no attempts to develop a trash classifier for android phones using deep learning.

Data description

I used the TrashNet dataset. It's composed of images belonging to six classes: glass, paper, cardboard, plastic, metal, and trash. TrashNet consists of 2527 images: 501 glass, 594 paper, 403 cardboard, 482 plastic, 410 metal, and 137 trash.

To take the pictures, the authors placed each object on a white posterboard and used sunlight and/or room lighting. The devices used were Apple iPhone 7 Plus, Apple

iPhone 5S, and Apple iPhone SE (Thung and Yang, 2016). The size of the dataset is 3.5GB. The authors then resized thedataset and compressed it resulting in a dataset of size 40.9 MB

I created another dataset that's 400% the size of the original TrashNet through augmentation. The augmentation method used was rotation (by 90°, 180° and 270°). The size of the augmented dataset is 122 MB.

Method description

I used transfer learning to re-train the MobileNetV2 model developed by Google, and pre-trained on the ImageNet dataset, on the Tashnet Dataset. Then, I converted the classifier to TFLite and used it in an android mobile application. My mobile application is based on TensorFlow's Android Application Example.

I decided to retrain an already existing model because the TrashNet dataset is relatively small (2527 images). Therefore, retraining will allow me to take advantage of features learned by the MobileNetV2 model. To do so, I followed the steps described below. Those steps are adapted from TensorFlowLite's tutorial named Recognize Flowers with TensorFlow on Android. The code for steps 1 to 6 is in the file named trashClassifier.py while the code for the android application (steps 7 and 8) is in folder named Android

1. Set up
    a. Download the TrashNet dataset
    b. Rescale the images using ImageDataGenerator
    c. Create the training generator
    d. Specify image size, batch size and the directory of training dataset directory.
    e. Create the validation generator
    f. Save the labels into a file
2. Create the base model from the pre-trained model
    a. Create an instance of MobileNetV2 without including the top layer
3. Extract features
    a. Freeze the convolutional base created in step B and use it for feature extraction
    b. Add a classifier as a top layer
    c. Compile the model
    d. Train the top-level classifier.
4. Fine tune the model
    a. Un-freeze the top layers of the model
    b. Train the weights of the newly unfrozen layers alongside training the weights of the top-level classifier
5. Convert to TFLite
    - Use TFLiteConverter
6. Download the converted model and labels
7. Develop the android application
8. Incorporate the Converted Model into Android Studio application (Moroney,2018)

a. Add the TensorFlowLite libraries to my app
b. Import a TensorFlow Lite interpreter
c. Create an instance of an Interpreter
d. Load it with a MappedByteBuffer
e. call the run method on the Interpreter

Model Description

I use a modified MobileNetV2 architecture. The original MobileNetV2 architecture is based on an inverted residual structure; the input and output of the residual block are thin bottleneck layers. To filter features in the intermediate expansion layer, MobileNetV2 uses lightweight depthwise convolutions. In addition, MobileNetV2 doesn't include non-linearities in the narrow layers (Sandler, Howard, Zhu, Zhmoginov and Chen, 2018). Figure 1 shows an overview of MobileNetV2 Architecture (Sandler and Howard, 2018)

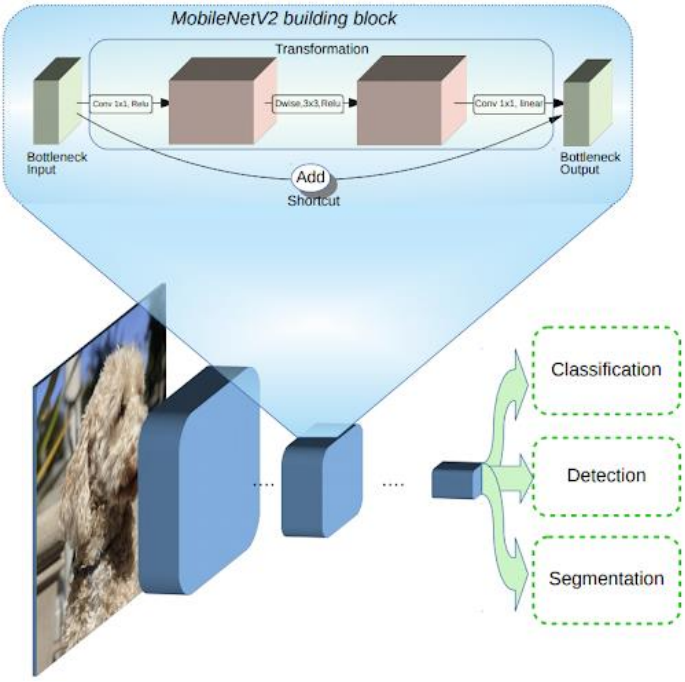

Figure 1: Simplified MobileNetV2 Architecture

In the modified version, the topmost layer of the MobileNetV2 model is replaced with a fully connected classifier. The pre-trained MobileNetV2 model (trained on the ImageNet dataset) is frozen and only the weights of the top-layer classifier get updated. After this, the top few layers of the original are unfrozen, and the model is further trained. Table 1 shows a summery of the modified model.

Table 1 Summery of the modified model

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Model)	(None, 7, 7, 1280)	2257984
conv2d_1 (Conv2D)	(None, 5, 5, 32)	368672
dropout_1 (Dropout)	(None, 5, 5, 32)	0
global_average_pooling2d_1 (	(None, 32)	0
dense_1 (Dense)	(None, 6)	198

Total params: 2,626,854
Trainable params: 2,231,462
Non-trainable params: 395,392

Experimental Procedure

The validation, training, and testing accuracy were measured while varying the number of epochs (from 1 to 100), the learning rate(1e-3,1e-5 and 1e-7), optimizer (Adam, SGD and RMSprop) and the freezing ratio (0%, 30%, 60%, 100%).

In addition, I measured the effect of using augmentation (turn right, turn left and flip vertically) to increase the size of the dataset by 300%. The augmented dataset consists of 10108 images.

I also experimented with adding L2 regulizer (through keras.regularizers.l2) to the dense layer and the convolutional layer. In addition, I experimented with and changing the drop out ratio.

For the mobile application, I will test the amount of time it takes the application to classify an image and the amount of storage and memory it takes up.

Results

As shown in figure 2, increasing the number of epochs increased the training accuracy and decreased the loss until about 30 epochs. After that, increasing the number of epochs has minimal impact on the training accuracy and the loss.

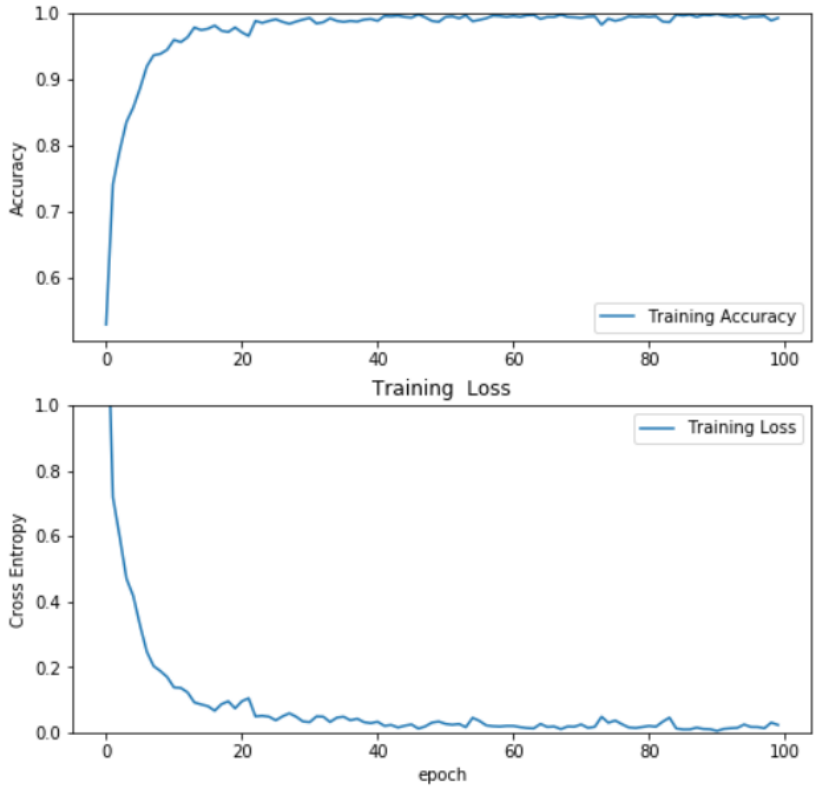

Figure 2 effect of increasing epochs on training accuracy loss

Changing the learning rate yielded considerable changes in the testing accuracy and loss as shown in Table 2. All those values were calculated at number of epochs = 50, freezing ratio = 100 and optimizer = SGD.

Table 2 effect of changing learning rate on test accuracy and loss

Learning Rate	Testing accuracy	Loss
1e-5	0.5010	4.6834
1e-3	0.2520	10.210
1e-7	0.2782	9.342

Changing the optimizer yielded minimal change in the testing accuracy and loss as shown in Table 3. All those values were calculated at number of epochs = 50, freezing ratio = 100 and learning rate 1e-5.

Table 3 effect of changing optimizer on testing accuracy and loss.

Optimizer	Testing accuracy	Loss
Adam	0.5010	4.6834
SGD	0.5250	2.9354
RMSprop	0.5070	2.8835

Changing the freezing ratio yielded minimal change in the testing accuracy and loss as shown in Table 3. All those values were calculated at number of epochs = 50, optimizer = SGD and learning rate 1e-5

Table 4 effect of changing the freezing on testing accuracy and loss.

Freezing ratio	Testing accuracy	Loss
0	0.5109	3.6927
100	0.5250	2.9354
155	0.5288	3.7587

Augmenting the dataset caused the average training accuracy of the last 3 epochs to increase to 0.9886 and the average loss to decrease to 0.0250. In addition, it caused the testing accuracy to increase to 0.6158 and the average loss to decrease to 2.4746

Adding an L2 regulizer to the Dense layer yielded varying effects depending on the value of l as shown in table 5. All those values were calculated at number of epochs = 50, optimizer = SGD and learning rate 1e-5 and with dropout ratio of 0.2.

Table 5 effect of adding a regulizer to the dense layer

Value of l	Average training accuracy for the last three epochs	Average training loss for the last three epochs	Testing accuracy	Testing loss
0.001	0.2291	1.7906	0.2353	2.9406
0.01	0.2582	1.8958	0.2425	1.7903
0.1	0.9598	0.1014	0.5621	1.8806
0.5	0.9853	0.0936	0.2873	7.1961

Adding an L2 regulizer to the convolutional layer yielded varying effects depending on the value of l as shown in table 6. All those values were calculated at number of epochs = 50, optimizer = SGD, learning rate 1e-5 and a regulizer at the Dense layer with l = 0.1 and with dropout ratio of 0.2.

Table 6 effect of adding a regulizer to the dense layer

Value of l	Average training accuracy for the last three epochs	Average training loss for the last three epochs	Testing accuracy	Testing loss
0.01	0.987	0.2247	0.5765	1.8806
0.1	0.9881	1.2980	0.6107	1.2980
0.2	0.9974	0.0929	0.4017	3.4829
0.35	0.9561	1.1572	0.6307	1.0980

Changing the dropout ratio yielded varying effects as shown in table 7. All those values were calculated at number of epochs = 10, optimizer = SGD, learning rate 1e-5, a regulizer at the Dense layer with l = 0.1 and a regulizer at the convolutional layer with l = 0.35

Table 7 effect of changing the dropout ratio

Dropout ratio	Average training accuracy for the last three epochs	Average training loss for the last three epochs	Testing accuracy	Testing loss
0.2	0.9561	1.1572	0.6307	1.0980
0.25	0.8125	1.0548	0.6043	1.2847
0.27	0.9974	0.0241	0.8350	1.0104
0.3	0.9981	0.0223	0.6627	1.5889
0.33	0.9979	0.0429	0.7235	1.4256

Adding another dropout layer after the dense layer had varying effects as shown in table 8. All those values were calculated at number of epochs = 10, optimizer = SGD, learning rate 1e-5, a regulizer at the Dense layer with l = 0.1, a regulizer at the convolutional layer with l = 0.35 and 0.27 dropout ratio for the first dropout layer.

Table 8 effect of changing the dropout ratio

Dropout ratio	Average training accuracy for the last three epochs	Average training loss for the last three epochs	Testing accuracy	Testing loss
0.05	0.2487	2.5487	0.2357	1.8547
0.1	0.6542	6.2548	0.5187	1.0320
0.2	0.6592	6.8518	0.1586	9.2548

The mobile application classifies an image in 0.3245 seconds. Its size is 42.11MB and, on average, it uses 100KB of memory/hour. Figure 3 shows screenshots of the application.

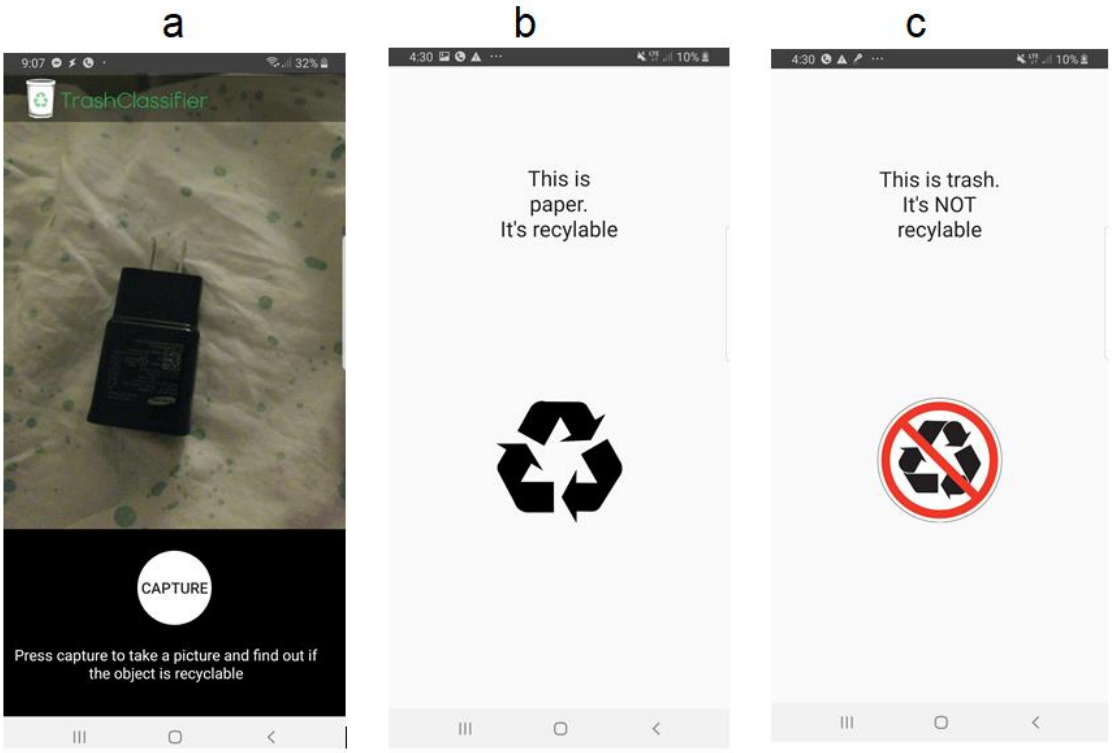

Figure 3 screenshots of the mobile application. (a) is the main interface of the application. It contains a camera view and a capture button. The remaining figure show screenshots of the application after the user captures an image of something the application deems (b) recyclable, (c) not recyclable.

Conclusion

The compressed model successfully achieved a 99% training accuracy with a loss of 0.02 and an 80% testing accuracy with 1.0 loss. This was achieved when using the augmented data, with number of epochs = 10, L2 regularization added to the dense layer (l = and the convolutional layer and a dropout ratio of 0.27.

The model initially performed poorly because of over fitting. Before augmenting the data and adding regulations, the model was only able to achieve 56% accuracy with 1.8 loss - even though the training accuracy was as high as 99% and the training loss was as low as 0.02. To compact the effect of overfitting, I augmented the data to increase the number of images by 300%. In addition, I added L2 regularization to the dense layer and the convolutional layer.

Adding an additional dropout layer after the dense layer reduced the training accuracy to a maximum 0.65, thus limiting the possibilities for test accuracy. Therefore, I avoided using it in the final model.

The over fitting is caused by the small dataset (2527 images) and the high noise in the dataset. Figure 4 shows a few of the images of glass in the dataset. The glass pieces are only occupying almost 30%- 50% of the images while the other 70% is noise.

Figure 4 sample images from the TrashNet dataset

Acknowledgements
Great thanks to Dr. Bo Yuan for designing this course and providing an engaging and supportive learning environment.

References

Thung, G. Yang M. 2016, Classification of Trash for Recyclability Status.

Ching, C. 2019, How to build an image classifier for waste sorting. Towards Data Science. Available at https://towardsdatascience.com/how-to-build-an-image-classifier-for-waste-sorting-6d11d3c9c478

Moroney, L. 2018, Using TensorFlow Lite on Android. A Medium Corporation. Available at http://www.library.uq.edu.au/training/citation/harvard.html

2018, Recognize Flowers with TensorFlow on Android. Available at https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android/#0

Sandler, M. Howard, A. Zhu, M. Zhmoginov, A. Chen L. 2018, MobileNetV2: Inverted Residuals and Linear Bottlenecks. Cornell University.

Sandler,M. Howard,A. 2018, MobileNetV2: The Next Generation of On-Device Computer Vision Networks. Google AI Blog. Available At https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html