FruitVegCNN: Power- and Memory-Efficient Classification of Fruits & Vegetables Using CNN in Mobile MPSoC

Somdip Dey* Suman Saha†
Amit Singh*, Klaus McDonald-Maier*,

* *Embedded and Intelligent Systems Laboratory, University of Essex, UK.*
† *Nosh Technologies, UK.*

ABSTRACT

Fruit and vegetable classification using Convolutional Neural Networks (CNNs) has become a popular application in the agricultural industry, however, to the best of our knowledge no previously recorded study has designed and evaluated such an application on a mobile platform. In this paper, we propose a power-efficient CNN model, FruitVegCNN, to perform classification of fruits and vegetables in a mobile multi-processor system-on-a-chip (MPSoC). We also evaluated the efficacy of FruitVegCNN compared to popular state-of-the-art CNN models in real mobile platforms (Huawei P20 Lite and Samsung Galaxy Note 9) and experimental results show the efficacy and power efficiency of our proposed CNN architecture.

KEYWORDS: convolutional neural network (CNN), multi-processor system-on-a-chip (MPSoC), fruits, vegetables, classification, flutter, mobile device, application, energy efficiency

1 Introduction and Motivation

In recent years, computer vision based Convolutional Neural Networks (CNNs) [CSJC10, CL14,SZ14] approaches have become very popular to solve several real-life challenges such as traffic categorization [LJLS15,DKS+18,DSPMM20], human rights violation [K+17], weather forecasting [Z+98], fruits and vegetables' classification [MO18,PSRGC18], etc due to its high prediction accuracy/categorization in the aforementioned target applications. Moreover, we can notice a steady growth in using CNNs in agriculture, especially for use cases such as automatic fruit harvesting, fruit sorting machines, and fruit scanning in supermarkets. Out of several applications of using CNNs in agriculture, classification of fruits and vegetables are an important one due to the fact that such an application could be used to automate and ret-

[1]Corresponding E-mail: somdip.dey@essex.ac.uk; dey@nosh.tech
[2]This work is supported by Nosh Technologies [nosh/agri-tech-000001] and the UK Engineering and Physical Sciences Research Council EPSRC [EP/R02572X/1 and EP/P017487/1].

rospectively improve productivity in agri-food system. For example, when a shopper goes to supermarket to buy fruits and vegetables, instead of typing all the items manually to create an inventory, the shopper can just take a picture of the bought fruits and vegetables and it automatically classifies the bought items and input them in the inventory.

On the other hand, battery operated mobile devices are becoming more affordable due to the improvement in chip manufacturing technology and also come equipped with multiple processing elements to cater for performance requirement of the executing applications [SDB+20, DGB+19]. Such a mobile platform, utilizing multi-processor system-on-a-chip (MPSoC) [SDB+20, DGB+19, DSPMM19, DSWMM19, IDSMM19, DSS+19, DSWMM20, DSMM19], implements different types of processing elements such as CPU and GPU, capable of computing demanding computer vision applications on the device. Given the importance of adopting CNN based approaches in agricultural industry and the increase in popularity of using mobile devices, it is very important to study, design and implement CNN models for fruits and vegetables' classification in mobile devices. A consumer can use their mobile device to make an inventory of their bought fruits and vegetables using such CNNs. However, implementing CNNs in mobile platforms come with their own challenges. Such challenges include implementing a power-efficient CNN, which is capable of accurately classifying fruits and vegetables while consuming the least power and memory on the device.

In this paper, we propose a CNN model, named FruitVegCNN, which can be utilized in mobile devices for fruits and vegetables' classification, and to the best of our knowledge this is the first work on designing and implementing a CNN model for the same task in a mobile platform. To this end, this paper makes the following contributions:

1. Proposal of FruitVegCNN, a light weight CNN model, capable of fruits and vegetables' classification in battery operated mobile device.

2. Comparative study between FruitVegCNN and implementation of different CNN models on real mobile devices (Huawei P20 Lite [p20] and Samsung Galaxy Note 9 [gal] mobile devices) to show the difference in memory consumption, power consumption, CPU load and GPU load.

3. Evaluation of FruitVegCNN on Huawei P20 Lite and and Samsung Galaxy Note 9 mobile devices.

2 Preliminaries

2.1 Convolutional Neural Networks and Deep Learning

A Deep Learning (DL) model [K+12] consists of an input layer, several intermediate (hidden) layers stacked on top of each other and an output layer. In the input layer, which is the first layer of the model, the raw values of data features are fed into it. In each of the hidden layers a mathematical operation called convolution is applied to extract specific features, which is then utilized to predict the label of the raw data in the last (output) layer of the DL network. Most of the time, if a model utilize an input layer, a hidden layer and an output layer then the model is denoted as Convolutional Neural Network (CNN) model or simply, CovNet. If such a model uses a lot of stacked hidden layers only then it is denoted as a DL model or Deep Neural Networks (DNN).

2.2 Pre-trained Networks and Transfer Learning

A conventional approach to enable training of DNN/CNN on relative small datasets is to use a model pre-trained on a very large dataset, and then use the CNN as an initialization for the applicative task of interest. Such a method of training is called "*transfer learning*" [P+09] and we have followed the same principle. The chosen CNN models mentioned in Sec. 4 are pre-trained on ImageNet. For the propose of classifying frtuis and vegetables on the mobile device, we have utilized the following popular pre-trained CNN models: VGG (VGG19) [SZ14], ResNet (ResNet152v2) [HZRS16], MobileNet (MobileNetv2) [H+17], NASNet (NASNetMobile and NASNetLarge) [ZVSL18] and Inception-ResNet [SIVA17].

3 Related Work

There has been several studies to utilize CNNs and DNNs for fruit classification [SPL19, WC18, ZDC+19, Lu16, PSRGC18, MO18, KSPS18] and fruit detection [BU17, CSD+17].

In [Lu16], Lu implemented CNN models with data expansion techniques to select images in ten-class food items from the ImageNet to compare its method against bag-of-feature and and support vector machine models. In [ZDC+19], Zhang et al. proposed a 13 layer CNN for fruit classification, and compared the effects of different types of data augmentation approach and max-pooling techniques on the prediction accuracy.

Wang et al. in [WC18] proposed an 8 layer CNN by using a parametric rectified linear unit (ReLU) and placing a dropout layer before each fully connected (FC) layer. Kausar et al. [KSPS18] proposed another fruit classification methodology, Pure-CNN, consisting of 7 convolutional layers. The CNN models proposed in [WC18] only uses fully connected layers at the end of the architecture, and [KSPS18] only utilizes convolutional layers in the architecture with one fully connected layer in the last layer of the CNN for classification purposes. Such models perform poorly for occluded images, which is the case for real life scenarios of fruit classification. In a CNN while using convolutional layer, features extracted from convolutional layer are spatially local [OL17] and due to use of only convolutional layers for occluded images some important features defining the object in the image could be lost, hence, failing to classify occluded images of fruits & vegetables properly during testing. Therefore, using a mixture of fully connected layers along with convolutional layers could resolve issues related to occlusion in images. On the other hand, fully connected layers are computationally expensive and consumes more power because each neuron is connected to the other neuron of the input and output of the layer. Therefore, in this paper we design a CNN architecture, using a combination of convolutional layers and fully connected layers, which overcomes the limitations of each other.

In [SPL19], Steinbrener et al. utilized a CNN model pre-trained for RGB image data to classify fruits and vegetables. Whereas, Mureşan et al. [MO18] introduced a new dataset of fruit images called Fruits-360 and utilized AlexNet and GoogleNet, pre-trained CNN models, to classify fruits' images. In [PSRGC18], Patino-Saucedo et al. implemented AlexNet CNN to classify tropical fruits.

In [BU17], Bargoti et al. proposed a framework based on CNN model to detect and count apples and mangoes. This work was implemented on a robotic vehicle, however, the platform was not a low powered device such as mobile phone. In another study, Chen et al. [CSD+17] used a blob detector based on a fully convolutional network (FCN) to count

total number of fruits in the input image.

In none of the aforementioned studies, designing and implementation of the CNN was performed by keeping the power-efficiency of executing the CNN in a mobile platform for classification purposes into consideration. This paper solves the aforementioned challenge and proposes a CNN model catering for: performance (prediction accuracy), power consumption and memory constraint of battery operated mobile platform.

4 Fine-tuning pre-trained CNNs for fruit and vegetable classification on mobile MPSoC

It is very common to choose a pre-trained CNN model and fine-tune the model to train on a target application [MO18, PSRGC18]. Fine-tuning is the process of taking the weights of a pre-trained CNN and using it as initialization for a new model being trained on a dataset from the same domain. This approach is used to speed up the training process while being able to train on small dataset. Since, pre-trained models such as VGG (VGG19) [SZ14], ResNet (ResNet152v2) [HZRS16], MobileNet (MobileNetv2) [H$^+$17], NASNet (NASNetMobile and NASNetLarge) [ZVSL18] and Inception-ResNet [SIVA17] are initially trained on ImageNet, which coincides with several of the fruits and vegetables' classes, hence, choosing these CNN models for fine-tuning to train to classify fruits and vegetables based on our dataset, as proposed in Sec. 4.1.

4.1 Dataset

For our fruits and vegetables' classification task we have utilized the Fruits-360 dataset introduced by Mureşan et al. [MO18]. The dataset consists of a total of 90483 images (67692 images for training and 22688 images for validation) for 131 different types (class labels) of fruits and vegetables. Each image in this dataset contains only one fruit or vegetable per image. We have also used the dataset introduced by Patino-Saucedo et al. [PSRGC18], which consists of a total of 2633 images of 15 different types of tropical fruits. Fig. 1 shows some of the representational images of different types of fruits and vegetables.

Since, the two dataset [MO18] & [PSRGC18] have some common fruits, we combined the two dataset to create our own dataset to train the CNN for the classification task. The common fruits/vegetables labels between the dataset are potato, peach, apple, melon, kiwi, nectarine, onion, orange, plum, pear, lime and watermelon, which comprised of 14 out of 15 classes in Patino-Saucedo et al.'s dataset [PSRGC18]. Therefore, we consolidated the images of the common classes/labels between the dataset, and the total number of class labels after consolidation became 132. Since, the [PSRGC18] did not come with separate training and validation images, we randomly select 25% of the dataset to be the validation set and the rest 75% as the training set. In order to test the prediction accuracy of trained CNN models, we also randomly chose 5 images per class label from the validation dataset and kept it separate. Therefore, the total number of testing images were 660.

4.2 Training a pre-trained CNN

For our classification task we chose the following popular pre-trained CNNs: VGG (VGG19), ResNet (ResNet152v2), MobileNet (MobileNetv2), NASNet (NASNetMobile and NASNet-

(a) Granny Smith Apple (b) Agata Potato (c) Braeburn Apple (d) Nectarine

Figure 1: Graphical representation of some of the classes from our dataset, as mentioned in Sec. 4.1

Table 1: Comparison between CNN models based on disk size and parameters

CNN Model	Size	Parameters
ResNet152v2	232 MB	60,380,648
NASNetMobile	23 MB	5,326,716
NASNetLarge	343 MB	88,949,818
VGG19	549 MB	143,667,240
MobileNetv2	14 MB	3,538,984
Inception-ResNet	215 MB	55,873,736

Large) and Inception-ResNet. Out of these CNNs MobileNet and NASNetMobile are specifically developed for mobile platforms. We chose these CNNs to evaluate their performance on our experimental mobile device to deduce which pre-trained CNN is most suitable for such classification task. Table 1 shows the difference in the memory (disk) size of the CNN models (non mobile version) along with their total number of parameters.

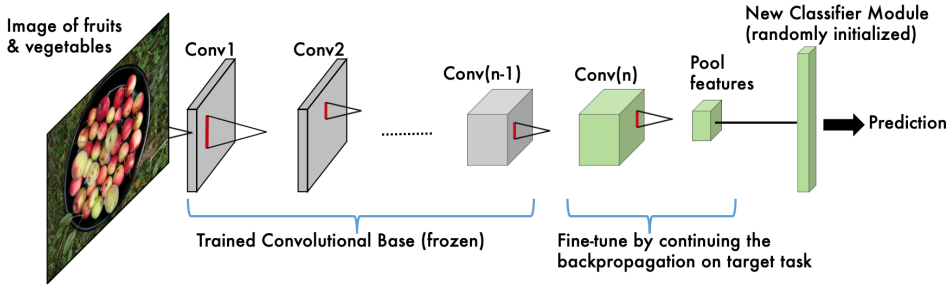

Figure 2: Network architecture used for fine-tuning pre-trained CNN

We fine-tuned our pre-trained CNN models by adding our a new randomly initialized classifier, and training the last fully connected layer by freezing all the layers of the base model (frozen layers represented with gray colour in Fig. 2) and unfreezing the last fully connected layer (unfrozen layers represented with green colour in Fig. 2). Given the computational constraint of mobile devices we perform the training of our CNN model on a general purpose computer using Tensorflow [ABC+16] backend for processing. Since, the trained CNN models are in Tensorflow format (non mobile version), they can not be directly implemented in mobile devices and have to be converted to Tensorflow Lite [LAD+19] for-

mat, which is a machine learning framework for on-device inference, to be implemented on the mobile device.

4.3 Hardware and software setup for training

The training was performed on a general purpose computer with 4 Intel(R) Xeon(R) Gold 6134 CPUs and CUDA enabled Nvidia Tesla P100 GPU with 12GB memory, which is utilized to significantly accelerate the training of the CNN models. The training system was running on Ubuntu version 16.04.6 LTS.

4.4 Hardware and software setup on mobile platforms

4.4.1 Huawei P20 Lite

We utilized Huawei P20 Lite [p20] smart-phone, which employs the HiSilicon Kirin 659 [kir] MPSoC, to evaluate different trained CNN models. Kirin 659 MPSoC is based on ARM's big.LITTLE technology and contains a cluster of 4 big CPU cores and a cluster of 4 LITTLE CPU cores. But the big.LITTLE implementation of this MPSoC is unique, since it uses the same type of CPU core for big as well as LITTLE. Kirin 659 MPSoC uses Cortex A-53 as both big and LITTLE CPU cores, which implements ARMv8-A ISA, supporting 64 bit instruction set and is userspace compatible with 32-bit ARMv7-A architecture.

This MPSoC also provides dynamic voltage frequency scaling feature per cluster, where the big core cluster has 5 frequency scaling levels ranging from 1402 MHz to 2362 MHz (at the following frequencies: 1402 MHz, 1805 MHz, 2016 MHz, 2112 MHz, 2362 MHz), and the LITTLE core cluster has 4 frequency scaling levels ranging from 480 MHz to 1709 MHz (at the following frequencies: 480 MHz, 807 MHz, 1306 MHz, 1709 MHz). If we consider P as the dynamic power consumption, V as the operating voltage, f as the operating frequency of the processing core, dynamic voltage and frequency scaling (DVFS) [DGB+19,SDB+20] is used to reduce the dynamic power consumption ($P \propto V^2 f$) by executing the workload over extra time at a lower voltage and frequency, which could be accounted for reduced power consumption of the device. Kirin 659 MPSoC also comes equipped with 2 Mali-T830 MP2 GPUs and 4GB RAM. Huawei P20 Lite comes with a non-removable 3000 milliamp Hour (mAh) battery.

4.4.2 Samsung Galaxy Note 9

We also chose Samsung Galaxy Note 9 [gal] to observe the computing resource utilizing of the trained CNNs. Note 9 is a recent powerful mobile device from Samsung and utilizes the Exynos 9810 MPSoC [exy]. Exynos 9810 has two CPU clusters (based on ARM's big.LITTLE technology), one for big CPU cores consisting of 4 Mongoose 3 CPU cores, and the other cluster for LITTLE CPU cores consisting of 4 Cortex A-55 CPU cores. The Mongoose 3 CPU cores allow cluster wise DVFS and has 18 frequency scaling levels ranging from 650 MHz to 2704 MHz (2704 MHz, 2652 MHz, 2496 MHz, 2314 MHz, 2106 MHz, 2002 MHz, 1924 MHz, 1794 MHz, 1690 MHz, 1586 MHz, 1469 MHz, 1261 MHz, 1170 MHz, 1066 MHz, 962 MHz, 858 MHz, 741 MHz, 650 MHz). Whereas, the LITTLE Cortex-A55 CPU cores has 10 frequency scaling levels ranging from 455 MHz to 1794 MHz (1794 MHz, 1690 MHz, 1456 MHz, 1248 MHz, 1053 MHz, 949 MHz, 832 MHz, 715 MHz, 598 MHz, and 455 MHz). The

Exynos 9810 MPSoC comes equipped with 18 Mali-G72 MP18 GPUs and 6GB RAM. Galaxy Note 9 comes equipped with a non-removable 4000 mAh battery.

4.4.3 Software setup & Profilers

Huawei P20 Lite smart-phone has power sensors as well as 13 thermal sensors, but due to lack of documentation from the vendor on the positioning of the thermal sensors it is not feasible to associate all the installed temperature sensors with specific cores/cluster. However, we were able to track one specific thermal sensor which is placed on the battery and in our evaluation we report the thermal behaviour of the battery. On Samsung Galaxy Note 9 we also observe the thermal sensor on the battery. To record different factors such as power consumption, CPU load, GPU load, memory (RAM) consumption and battery temperature, we used Trepn Profiler by Qualcomm (version 6.2) [tre] and the Profiler app by Tomas Chladek (version 1.5.5) [pro]. Trepn Profiler doesn't have support for GPU load profiling and hence, we utilize Chladek's Profiler app to observe the GPU load. The Kirin 659 MPSoC was running on the Android 8.0.0.168 (Oreo) [anda] OS (utilizing Linux Kernel: 4.4.23+ #1 SMP). The Galaxy Note 9 was running on Android 9 (Pie) [andb] OS utilizing Linux kernel version 4.9.59.

To implement the trained CNN model (in Tensorflow Lite format) on the mobile device we developed a mobile application using Flutter [flu], which is a cross-platform mobile application development framework by Google. In the Flutter app, we implemented a continuosly streaming camera module, which inputs the image from the camera and continuously streams the images to the trained CNN for inference. Fig. 3 shows the user-interface of our Flutter application with an overlay of the Profiler app while profiling the computing resources during inference (classification).

4.5 Evaluation of trained CNN models

After the training of the CNNs completed on our dataset, we evaluated the validation accuracy of the respective models. In a trained CNN, validation accuracy and testing accuracy reflects the performance of the CNN for the target application and hence, we observe the respective values to compare the CNNs in Table 2 and Table 6. The testing accuracy and comparison based on the same is discussed in details in Sec. 6 and in Table 6.

From Table 2 it could be noticed that VGG19 performed the best for validation testing for fruits and vegetables' classification, whereas, both the NASNet CNNs (NASNetMobile & NASNetLarge) performed the worst. The NASNet CNN architecture is massive, consisting of two different types of layers (Normal cell and Reduction cell) [ZVSL18] and training such an architecture requires a lot of computational resources. Fig. 4 shows the improvement in validation accuracy achieved during the training period (training epoch) of the respective CNNs.

4.6 Evaluation of trained CNN models on the mobile device

After converting the trained models to Tensorflow Lite format, the memory (disk) size of the trained VGG (VGG19), ResNet (ResNet152v2), MobileNet (MobileNetv2), NASNet (NASNetMobile and NASNetLarge) and Inception-ResNet models got reduced. Table 3 shows

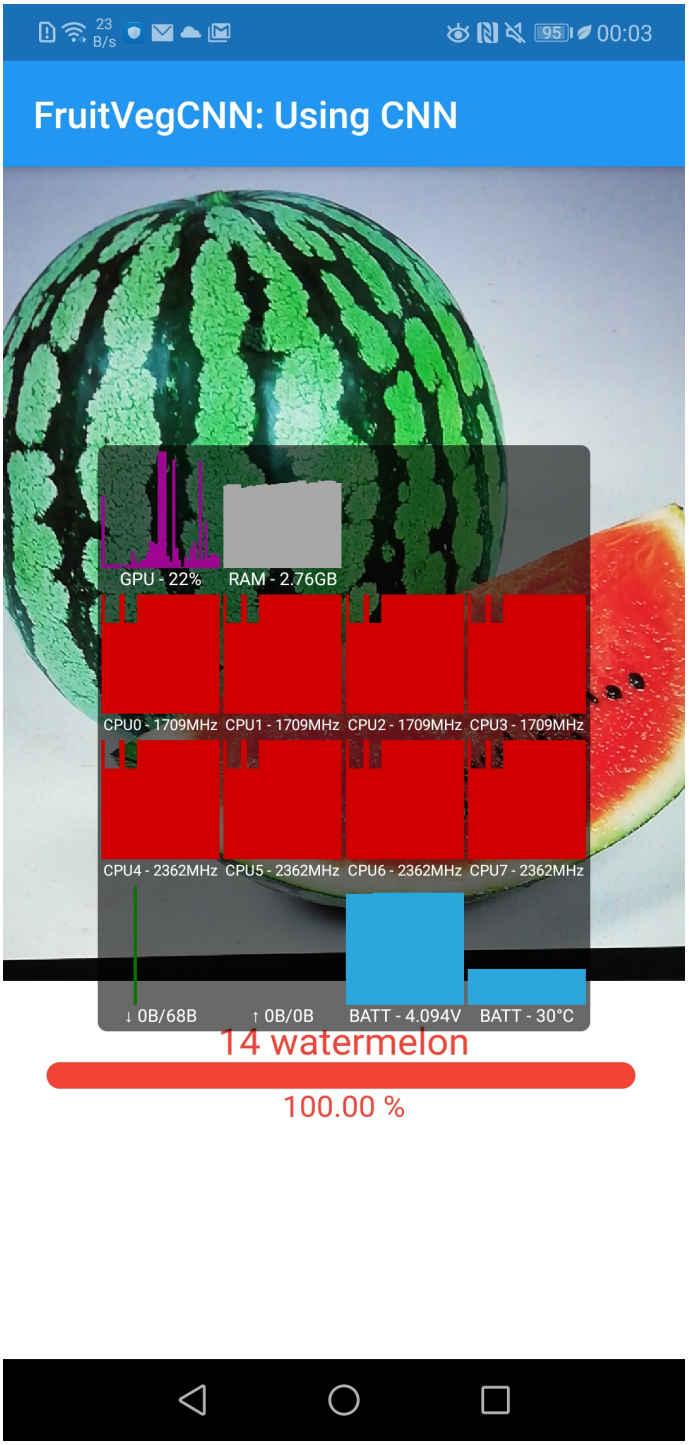

Figure 3: An illustration of our Flutter app implementing the trained CNN (VGG19) model for fruits & vegetables classification while the Profiler app [pro] runs overlaying the Flutter app in Huawei P20 Lite

Table 2: Comparison between trained CNN models on validation accuracy (%)

CNN Model	Validation Accuracy (%)
ResNet152v2	95.027
NASNetMobile	10.108
NASNetLarge	10.054
VGG19	98.918
MobileNetv2	97.892
Inception-ResNet	98.432

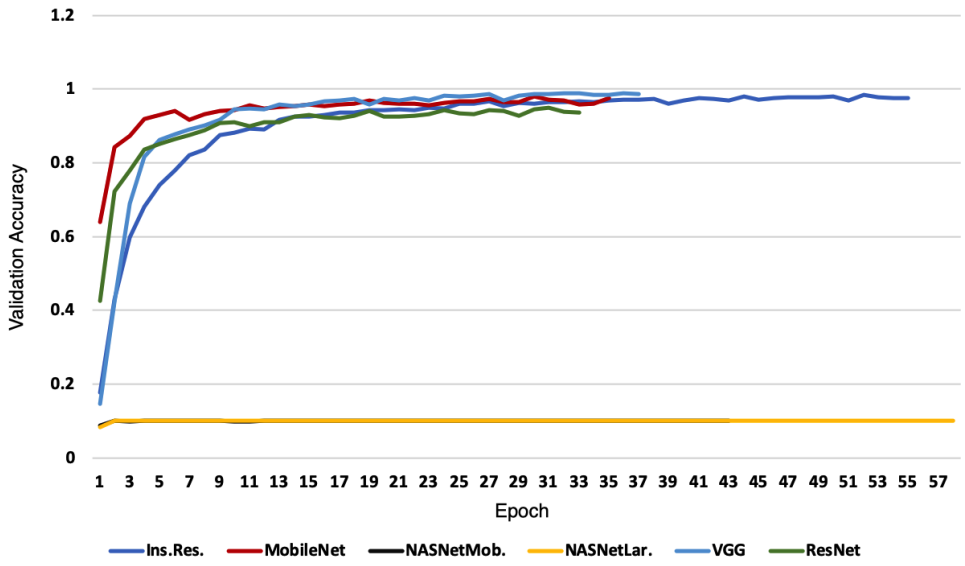

Figure 4: Validation accuracy of trained VGG, ResNet, MobileNet, NASNetLarge (NASNetlar.), NASNetMobile (NASNetMob.) and Inception-ResNet (Ins.Res.) over training epochs

Table 3: Comparison between trained CNN models in Tensorflow Lite format based on disk size

CNN Model	Size
ResNet152v2	335.5 MB
NASNetMobile	20.9 MB
NASNetLarge	352.5 MB
VGG19	182.9 MB
MobileNetv2	23.3 MB
Inception-ResNet	340.8 MB

Table 4: Comparison between trained CNN models in Tensorflow Lite format based on Ld. time, RAM, CPU%, GPU%, Power and Bat. Temp. in Huawei P20 Lite

CNN Model	Ld. time (sec)	RAM (GB)	CPU%	GPU%	Power (mW)	Bat. Temp. (° C)
ResNet152v2	16	0.65	64	21	2150	37
NASNetMobile	13	0.53	61	15	3302	37
NASNetLarge	14	0.69	68	11	3475	33
VGG19	17	0.78	72	22	3458	30
MobileNetv2	11	0.54	59	13	2746	36
Inception-ResNet	12	0.62	59	22	2880	38

the reduced disk size of the trained CNNs in Tensorflow Lite format for mobile implementation.

To compare the trained CNNs on mobile device we observe the following factors: Average memory (RAM) consumption (denoted as *RAM*), Loading time of the CNN model in seconds (denoted as *Ld. time*), average CPU load which is normalized across 8 CPUs of Kirin 659 & Exynos 9810 MPSoCs (denoted as *CPU%*), average GPU load (denoted as *GPU%*), average power consumption in milliwatt (mW) (denoted as *Power*) and average battery temperature in ° centigrades (denoted as *Bat. Temp.*). The Flutter application, implementing the respective trained CNN, was executed for 60 seconds while profiling, which includes the loading time of the model and inference of stream images from the camera. Table 4 shows the comparative study between different CNNs in the Huawei P20 Lite based on Ld. time (sec), RAM (GB), CPU%, GPU%, Power (mW) & Bat. Temp. (° C), whereas, Table 5 shows the comparative study between different CNNs in the Galaxy Note 9.

Table 5: Comparison between trained CNN models in Tensorflow Lite format based on Ld. time, RAM, CPU%, GPU%, Power and Bat. Temp. in Samsung Galaxy Note 9

CNN Model	Ld. time (sec)	RAM (GB)	CPU%	GPU%	Power (mW)	Bat. Temp. (° C)
ResNet152v2	4	0.72	56	1	4052	35
NASNetMobile	6	0.83	56	0	3953	34
NASNetLarge	19	1.39	54	1	4026	36
VGG19	3	0.9	62	1	7952	35
MobileNetv2	2	0.73	53	0	4019	35
Inception-ResNet	5	0.74	55	1	4042	35

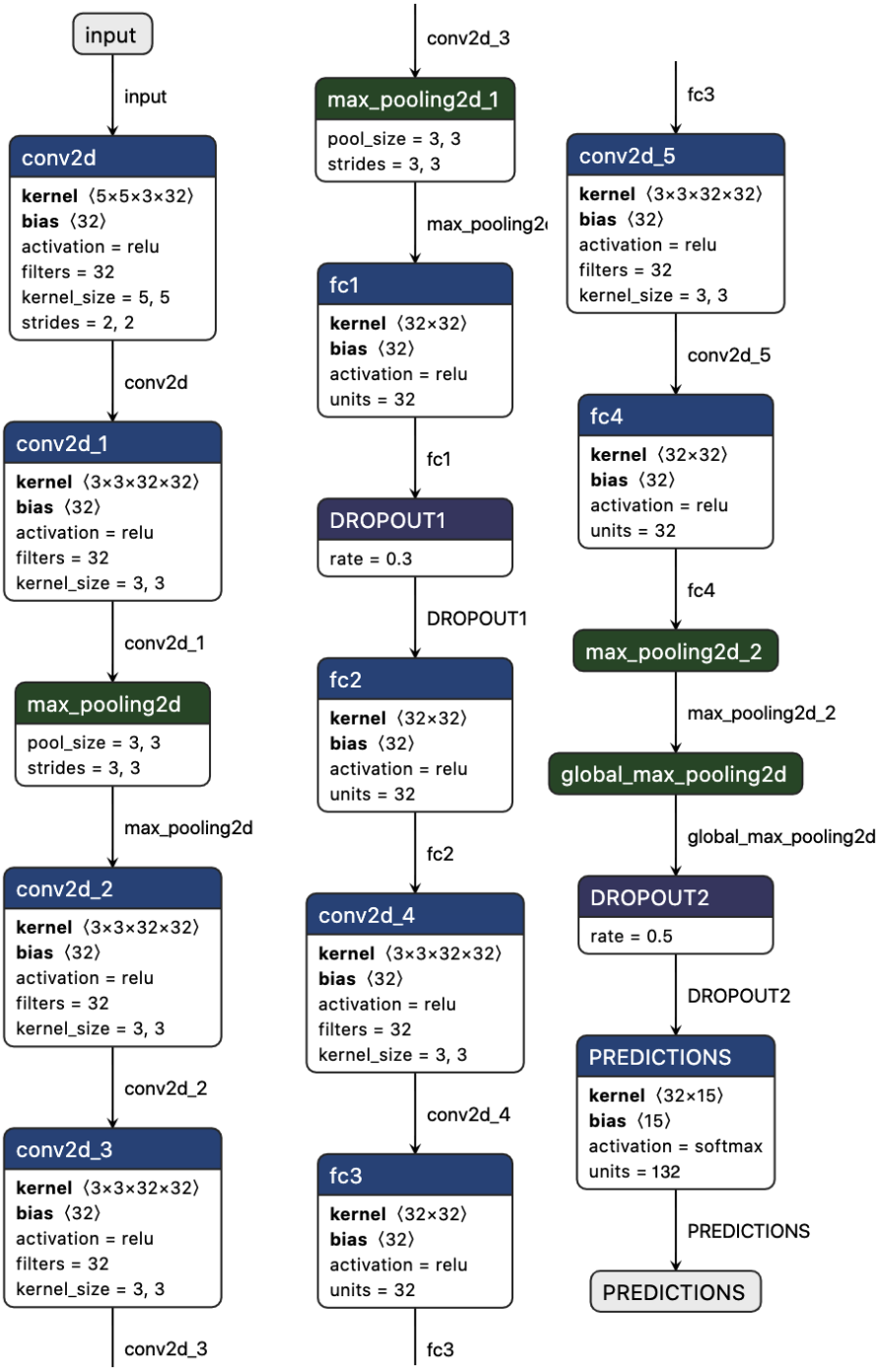

Figure 5: An illustration of the architecture of our FruitVegCNN

5 The proposed architecture: FruitVegCNN

Based on the comparative study of different trained CNNs as shown in Tables 4 and 5 we wanted to develop a CNN model, which has less number of trainable parameters and utilizes less computing resources comparatively while performing almost similar to existing popular CNNs. In this section, we introduce FruitVegCNN, a 10 layered network for fruit and vegetable classification.

5.1 Overall architecture

The overall architecture of our proposed network, FruitVegCNN, for fruits and vegetables' classification is illustrated in Fig. 5. FruitVegCNN consist of 10 learned layers with weights: 6 convolutional layers and 4 fully connected layers. The output of the last fully-connected layer is fed to a 132-way softmax which produces a distribution over the 132 class labels. The first convolutional layer filters the 224 x 224 x 3 input image with a kernel (neuron) of size 5 x 5 x 3 with a stride of 4 pixels. The second convolutional layer takes the output of the first convolutional layer and filters it with a kernel of size 3 x 3 x 32. The third convolutional layer takes the response-normalized and pooled output of the second convolutional layer as input and filters it with a kernel of size 3 x 3 x 32. In a CNN architecture the pooling layer summarizes the outputs of neighboring groups of neurons in the same kernel map. The fourth convolutional layer is connected to the first fully connected layer, whose input is the response-normalized and pooled output of fourth convolutional layer, and filters it with a kernel of size 32. The fourth, fifth and sixth convolutional layers, all has a kernel of size 3 x 3 x 32. In the learned layers we have utilized Rectified Linear Units (ReLUs) as the activation function of the neurons. The standard way to model a neuron's output (f) as a function of its input (x) is with $f(x) = max(0, x)$, which is the activation function using ReLU.

5.2 Reducing overfitting

Our neural network architecture has 53,391 parameters and it makes it insufficient to learn so many parameters without overfitting. To resolve this challenge we use the following two approaches:

5.2.1 Data augmentation

One of the popular and easy way to reduce overfitting on the image data is to artificially enhance the dataset using label-preserving transformations of the images. In our training period we have utilized combination of the following data augmentation approaches: rotation, random width and height shift, random zoom, horizontal flip, salt & pepper and coarse dropout. The augmentation techniques of salt & pepper and coarse dropout are used to mimic occlusion in images [OL17]. Fig. 6 shows the output of aforementioned data augmentation approaches on an image of braeburn apple (as shown in Fig. 1.(c)). The augmentation/transformation of the images are done during the training period and the transformed images does not have to be stored separately on the disk. Augmentation was performed during the time of the training.

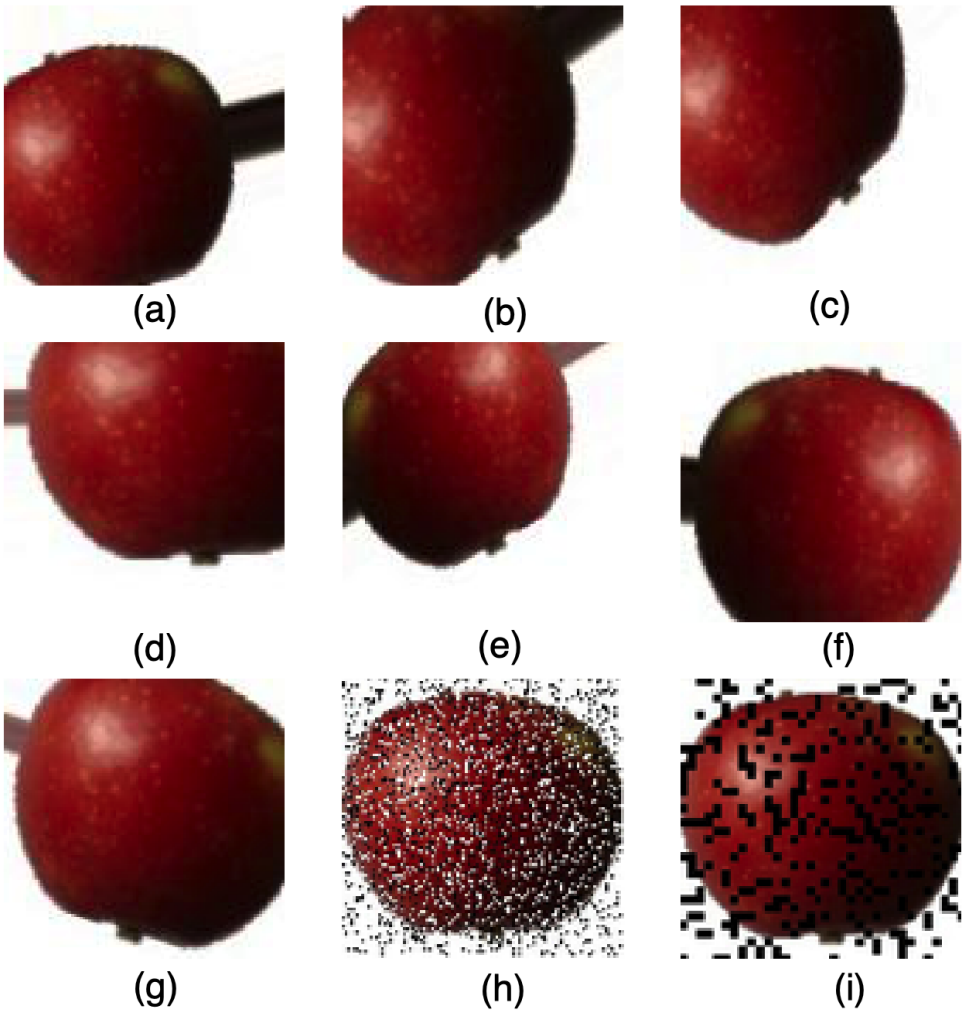

Figure 6: Data augmentation output of Fig. 1.c - braeburn apple: (a) rotation + width shift; (b) rotation + zoom; (c) rotation + width shift + zoom (d) zoom; (e) horizontal flip + width shift; (f) horizontal flip + zoom; (g) rotation + height shift + zoom; (h) salt & pepper; (i) coarse dropout

Table 6: Comparison between trained CNN models on testing accuracy (%)

CNN Model	Testing Accuracy (%)
ResNet152v2	68.18
NASNetMobile	2.5
NASNetLarge	1.515
VGG19	**72.72**
MobileNetv2	70
Inception-ResNet	70.45
FruitVegCNN	**71.36**

5.2.2 Dropout layer

To improve generalization and reduce overfitting in a CNN model, *dropout layers* [KSH12] can be used. In a dropout layer neurons which are dropped out, do not contribute to the forward pass and back propagation. In a dropout layer, the output of each hidden neuron is set to zero with a probability of p. We used a dropout layer, which is denoted as DROPOUT1 in Fig. 5, of 0.3 probability (p) between the first fully connected layer (fc1 in Fig. 5) and the second fully connected layer (fc2 in Fig. 5). DROPOUT1 is placed between fc1 and fc2 to improve learning generalization such that parts of the feature map produced from the convolutional layers conv2d to conv2d_3 in Fig. 5 could be forgotten, hence, improving learning and reducing overfitting. We have utilized another dropout layer of 0.5 p right before the prediction (output) layer as well.

6 Evaluation of FruitVegCNN and comparative study

When we implemented FruitVegCNN on Huawei P20 Lite & Samsung Galaxy Note 9 the disk storage of the model in Tensorflow Lite format was 220 KB, which is a very small fraction of the disk size of other popular CNNs as shown in Table 3. The validation accuracy achieved by FruitVegCNN during the training was 95.081% which is comparable to other CNNs as shown in Table 2. When we evaluated FruitVegCNN against other popular CNNs for the testing accuracy, FruitVegCNN performed very close comparatively and the evaluation results are illustrated in the Table 6. The Ld. time (sec), RAM (GB), CPU%, GPU%, Power (mW) & Bat. Temp. (° C) for FruitVegCNN in Huawei P20 Lite are 9, 0.34, 58, 14, 1560 and 33 respectively. In Samsung Galaxy Note 9, the Ld. time (sec), RAM (GB), CPU%, GPU%, Power (mW) & Bat. Temp. (° C) for FruitVegCNN are 1, 0.5, 45, 0, 3092 and 32 respectively. If we compare the respective values for FruitVegCNN with other CNNs as specified in Table 4 and Table 5, we can definitely observe that FruitVegCNN performs most power efficiently while occupying the least memory (disk and RAM). Compared to VGG, FruitVegCNN consumes 54.88% less power, 56.41% less RAM memory and loads 47.06% faster in Huawei P20 Lite. Whereas, in Samsung Galaxy Note 9, compared to VGG, FruitVegCNN consumes 61.12% less power, 44.44% less RAM memory and loads 66.67% faster.

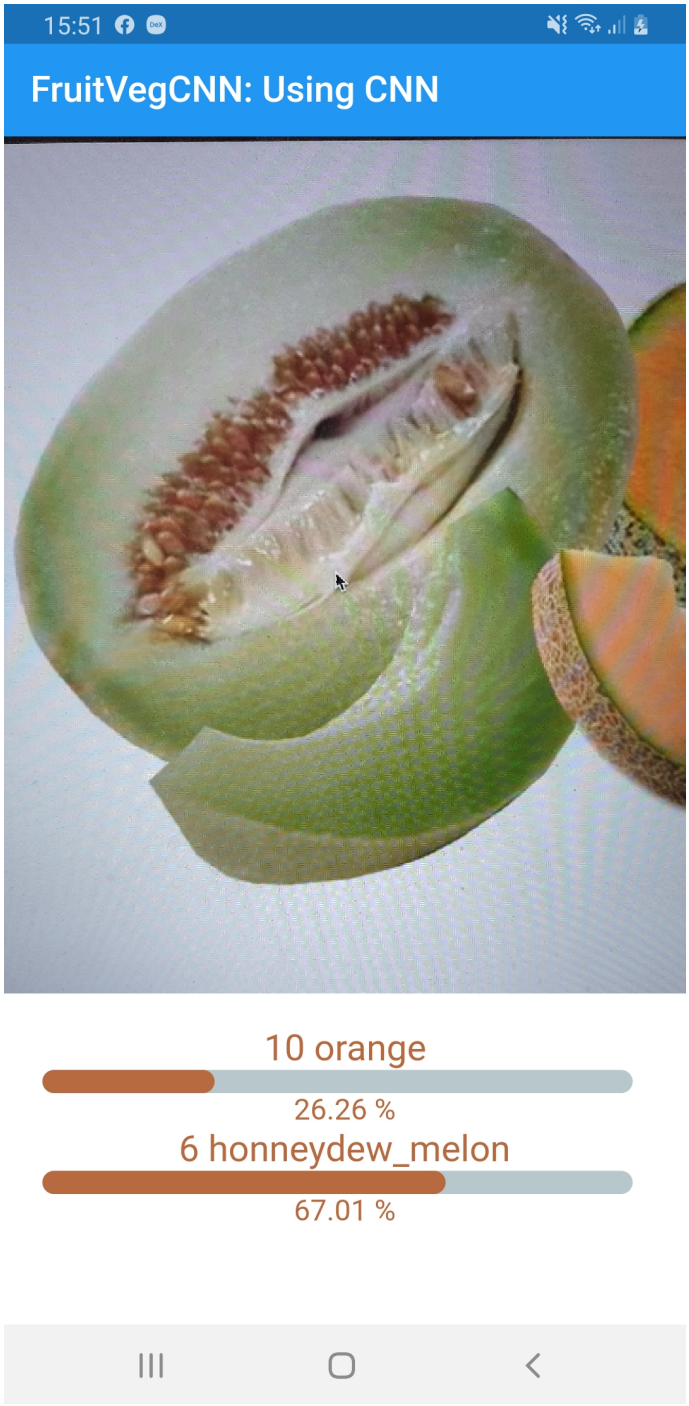

Figure 7: An illustration of our Flutter app implementing the FruitVegCNN model for fruits & vegetables classification in Samsung Galaxy Note 9

7 Conclusions

In this paper, we proposed a novel CNN architecture, FruitVegCNN, to perform fruit and vegetable classification on Huawei P20 Lite and Samsung Galaxy Note 9 mobile devices in a power-efficient manner. We also provided a comparative study of FruitVegCNN with current state-of-the-art CNNs such as VGG, ResNet, MobileNet, NASNet and Inception-ResNet. Comparative study and experimental evaluation shows the efficacy, power- and memory-efficiency of our proposed CNN architecture.

8 Code availability

The program source code to implement and train each of the CNN models including FruitVeg-CNN can be accessed from here: #Code will be made available upon acceptance#.

References

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[anda] Android 8 oreo. `https://www.android.com/versions/oreo-8-0/`. Accessed: 2018-01-31.

[andb] Android 9 pie. `https://www.android.com/versions/pie-9-0/`. Accessed: 2018-01-31.

[BU17] Suchet Bargoti and James Underwood. Deep fruit detection in orchards. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3626–3633. IEEE, 2017.

[CL14] Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[CSD+17] Steven W Chen, Shreyas S Shivakumar, Sandeep Dcunha, Jnaneshwar Das, Edidiong Okon, Chao Qu, Camillo J Taylor, and Vijay Kumar. Counting apples and oranges with deep learning: A data-driven approach. *IEEE Robotics and Automation Letters*, 2(2):781–788, 2017.

[CSJC10] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 247–257. ACM, 2010.

[DGB+19] Somdip Dey, Enrique Zaragoza Guajardo, Karunakar Reddy Basireddy, Xiaohang Wang, Amit Kumar Singh, and Klaus McDonald-Maier. Edgecoolingmode: An agent based thermal management mechanism for dvfs enabled

heterogeneous mpsocs. In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pages 19–24. IEEE, 2019.

[DKS+18] Somdip Dey, Grigorios Kalliatakis, Sangeet Saha, Amit Kumar Singh, Shoaib Ehsan, and Klaus McDonald-Maier. Mat-cnn-sopc: Motionless analysis of traffic using convolutional neural networks on system-on-a-programmable-chip. In *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 291–298. IEEE, 2018.

[DSMM19] Somdip Dey, Amit Kumar Singh, and Klaus Dieter McDonald-Maier. P-edgecoolingmode: an agent-based performance aware thermal management unit for dvfs enabled heterogeneous mpsocs. *IET Computers & Digital Techniques*, 13(6):514–523, 2019.

[DSPMM19] Somdip Dey, Amit Kumar Singh, Dilip Kumar Prasad, and Klaus Dieter Mcdonald-Maier. Socodecnn: Program source code for visual cnn classification using computer vision methodology. *IEEE Access*, 7:157158–157172, 2019.

[DSPMM20] Somdip Dey, Amit Kumar Singh, Dilip Kumar Prasad, and Klaus Dieter Mcdonald-Maier. Iron-man: An approach to perform temporal motionless analysis of video using cnn in mpsoc. *IEEE Access*, 8, 2020.

[DSS+19] Somdip Dey, Amit Kumar Singh, Sangeet Saha, Xiaohang Wang, and Klaus Dieter McDonald-Maier. Rewardprofiler: A reward based design space profiler on dvfs enabled mpsocs. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 210–220. IEEE, 2019.

[DSWMM19] Somdip Dey, Amit Kumar Singh, Xiaohang Wang, and Klaus Dieter McDonald-Maier. Deadpool: Performance deadline based frequency pooling and thermal management agent in dvfs enabled mpsocs. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 190–195. IEEE, 2019.

[DSWMM20] Somdip Dey, Amit Singh, Xiaohang Wang, and Klaus McDonald-Maier. User interaction aware reinforcement learning for power and thermal efficiency of cpu-gpu mobile mpsocs. In *2020 DATE*, pages 1728–1733. IEEE, 2020.

[exy] Exynos 9 series (9810). `https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9810`. Accessed: 2019-01-27.

[flu] Flutter: the first ui platform designed for ambient computing. `https://developers.googleblog.com/2019/12/flutter-ui-ambient-computing.html`. Accessed: 2020-06-23.

[gal] Galaxy note9. `https://www.samsung.com/global/galaxy/galaxy-note9/`. Accessed: 2018-01-27.

[H+17] Andrew G Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IDSMM19] Samuel Isuwa, Somdip Dey, Amit Kumar Singh, and Klaus McDonald-Maier. Teem: Online thermal-and energy-efficiency management on cpu-gpu mpsocs. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 438–443. IEEE, 2019.

[K+12] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[K+17] Grigorios Kalliatakis et al. Detection of human rights violations in images: Can convolutional neural networks help? *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP*, 2017.

[kir] Hisilicon kirin 650 (659). `http://www.hisilicon.com/en/Solutions/Kirin`. Accessed: 2018-07-23.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KSPS18] Asia Kausar, Mohsin Sharif, Jinhyuck Park, and Dong Ryeol Shin. Pure-cnn: A framework for fruit images classification. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 404–408. IEEE, 2018.

[LAD+19] Marcia Sahaya Louis, Zahra Azad, Leila Delshadtehrani, Suyog Gupta, Pete Warden, Vijay Janapa Reddi, and Ajay Joshi. Towards deep learning using tensorflow lite on risc-v. In *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2019.

[LJLS15] Zhiming Luo, Pierre-Marc Jodoin, Shao-Zi Li, and Song-Zhi Su. Traffic analysis without motion features. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 3290–3294. IEEE, 2015.

[Lu16] Yuzhen Lu. Food image recognition by using convolutional neural networks (cnns). *arXiv preprint arXiv:1612.00983*, 2016.

[MO18] Horea Mureşan and Mihai Oltean. Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica*, 10(1):26–42, 2018.

[OL17] Elad Osherov and Michael Lindenbaum. Increasing cnn robustness to occlusions by reducing filter support. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 550–561, 2017.

[p20] Huawei p20 lite. `https://consumer.huawei.com/uk/phones/m/p20-lite/`. Accessed: 2018-07-23.

[P+09] Sinno Jialin Pan et al. A survey on transfer learning. *IEEE TKDE*, 22(10), 2009.

[pro] Profiler by tomas chladek. `https://play.google.com/store/apps/details?id=cz.chladek.profiler`. Accessed: 2020-06-23.

[PSRGC18] Alberto Patino-Saucedo, Horacio Rostro-Gonzalez, and Jorg Conradt. Tropical fruits classification using an alexnet-type convolutional neural network and image augmentation. In *International Conference on Neural Information Processing*, pages 371–379. Springer, 2018.

[SDB+20] Amit Kumar Singh, Somdip Dey, Karunakar Reddy Basireddy, Klaus McDonald-Maier, Geoff V Merrett, and Bashir M Al-Hashimi. Dynamic energy and thermal management of multi-core mobile platforms: A survey. *IEEE Design & Test*, 2020.

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[SPL19] Jan Steinbrener, Konstantin Posch, and Raimund Leitner. Hyperspectral fruit and vegetable classification using convolutional neural networks. *Computers and Electronics in Agriculture*, 162:364–372, 2019.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[tre] Trepn profiler by qualcomm. `https://www.apkmirror.com/apk/qualcomm-innovation-center-inc/trepn-profiler`. Accessed: 2020-06-23.

[WC18] Shui-Hua Wang and Yi Chen. Fruit category classification via an eight-layer convolutional neural network with parametric rectified linear unit and dropout technique. *Multimedia Tools and Applications*, pages 1–17, 2018.

[Z+98] Guoqiang Zhang et al. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

[ZDC+19] Yu-Dong Zhang, Zhengchao Dong, Xianqing Chen, Wenjuan Jia, Sidan Du, Khan Muhammad, and Shui-Hua Wang. Image based fruit category classification by 13-layer deep convolutional neural network and data augmentation. *Multimedia Tools and Applications*, 78(3):3613–3632, 2019.

[ZVSL18] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.