Comparison of Convolutional Neural Network Models for Mobile Devices

Vivian Kimie Isuyama[1], **Bruno de Carvalho Albertini** [1]

[1]Escola Politécnica – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

mimi.vki@gmail.com, balbertini@usp.br

Abstract. *In recent years mobile devices have become an important part of our daily lives and Deep Convolutional Neural Networks have been performing well in the task of image classification. Some considerations have to be made when running a Neural Network inside a mobile device such as computational complexity and storage size. In this paper, common architectures for image classification were analyzed to retrieve the values of accuracy rate, model complexity, memory usage, and inference time. Those values were compared and it was possible to show which architecture to choose from considering mobile restrictions.*

1. Introduction

Convolutional Neural Networks (CNNs) are a class of Artificial Neural Networks that have been performing well in the area of image classification [Rawat and Wang 2017]. CNNs got popular for image classification ever since the 2012 ImageNet Large Scale Visual Challenge (ILSRVC), a competition that evaluates algorithms for object detection and image classification at a large scale [Russakovsky et al. 2015]. The winner of the 2012 competition used a CNN called AlexNet [Krizhevsky et al. 2012] and got a difference of 11.1% on the top-5 test error rate compared to the runner-up. The top-5 error rate is the percentage of test images for which the correct label is amongst the top 5 most probable predictions made by the Neural Network. In the subsequent years, most entries have been of CNNs [Rawat and Wang 2017].

There are a number of different architectures within the CNN class, notable AlexNet [Krizhevsky et al. 2012], ZFNet [Zeiler and Fergus 2014], Inception [Szegedy et al. 2015], ResNet [He et al. 2016], and SENet [Hu et al. 2018], the winners on the ILSRVC object classification challenge. In the midst of them, some topologies have been following the tendency of adding more layers in the network to improve accuracy. However, this trade-off of layers/accuracy raises the computational complexity of the network, since it requires more operations [Iandola et al. 2016]. To absorb this computational cost, powerful computers and specialized hardware such as GPU and CPU clusters are used.

Nonetheless, not all systems can deliver this high computational power. Mobile devices, for example, are intrinsically restricted regarding memory, processing speed, energy consumption, among others, making it costly to run these multi-layered architectures. Although mobile devices have the mentioned limitations, they have become an important part of our daily lives. Also, it is believed that deep learning will play an important role in the evolution of mobile applications [Chen et al. 2016]. So, in this paper, performance

parameters relevant for image classification inside mobile devices were identified and compared between CNN architectures.

Section 2 gives a brief introduction about the following topics: running neural networks on mobile devices, Artificial Neural Networks, Convolutional Neural Networks, and transfer learning. Section 3 overviews other papers related to CNNs being used in devices with limited resources. Section 4 describes the workbench and conditions for training and collecting results from different CNN architectures that were studied in this paper. Section 5 presents all obtained data from the CNNs training and benchmarks for classification. Section 6 discuss which CNN architecture had the best results for each parameter measured in this paper.

2. Background

2.1. Mobile Devices

Mobile devices have limited storage and computation capability [Chen et al. 2016], mostly due to the power source nature (battery), which should be considered when deploying a deep neural network on them. There are two different ways that a deep learning inference can be done with a mobile phone: cloud-based (1) and on-device (2). Cloud-based deep learning inference, also called Edge, is done by calling cloud-exposed APIs that host a pre-trained model to run an inference algorithm. The latter is done by using the mobile CPU and GPU to run the inference and its memory to store the model [Guo 2018]. A new trend is to use custom hardware or co-processor to accelerate machine learning, which is present on new processors [Qasaimeh et al. 2021]. This paper focuses on low-power devices without hardware acceleration support, which represents $2/3$ of the current market share [Haas and Davies 2020].

The advantages of on-device inference are: being able to work without internet access, offering data privacy, and having no cloud hosting costs. However, mobile devices are limited in resource capabilities compared to cloud servers, creating the necessity for a lightweight neural network.

2.2. Artificial Neural Networks

Artificial Neural Networks (ANNs) are machine learning systems inspired by a biological model of the brain composed of interconnected neurons. These neurons are stacked into layers, often an input layer a hidden layer, and an output layer. ANNs can infer rules for pattern recognition through examples, being able to do algorithmically difficult tasks. One example of a difficult task is to recognize handwritten numbers through images [Nielsen 2015].

The way that ANNs can infer rules is by adjusting weights that are present in the neurons. The weights are adjusted with examples of input and expected outputs through a learning process. The neurons in the input layer transmit information from the input to the neurons on the hidden layer. The hidden layer neurons receive a summation of the input layer information transformed by an activation function. Then, the values are multiplied by their respective neuron weight and transmitted to the output layer. The values go through another summation applying an activation function resulting in the final output [García-Alba et al. 2019].

2.3. Convolutional Neural Networks

CNNs are a specialized class of artificial neural networks that are leading on most of the tasks of image classification, detection, and recognition [Rawat and Wang 2017]. They are similar to ANNs as they are made of interconnected neurons with adjustable (learnable) weights. The difference resides in a convolution operation that is used in at least one of its layers, hence their name. The convolution is a mathematical operation that creates a function using two other functions as input. The convolution operation used in the convolutional layer of a CNN extracts features from the input.

Compared to ANNs, CNNs take into consideration invariances of two-dimensional shapes and require fewer free parameters [LeCun et al. 1998], benefiting CNNs over ANNs for image classification tasks.

2.4. Transfer Learning

Transfer learning for Convolutional Neural Networks is when a model is trained on two or more datasets, keeping the learned features (neuron weights) from the previous dataset when training using the next one. It helps the next dataset to reach better results with less training according to the degree of similarities between the datasets [Li et al. 2020].

This is possible due to the first layers behaving almost always the same way for a specific input type, having very similar weights. For natural images, for example, the first layers always resemble either Gabor filters or color blobs [Yosinski et al. 2014]. The similarity on the first layers allows reducing the training time for a given dataset since the weights have fewer adjustments to be done to the network to reach higher accuracy. Hence, transfer learning is used for training smaller datasets and reducing the training time for a given dataset.

3. Related Work

CNN Image Recognition on Mobile Phones

Recently, CNNs have been used for image recognition in mobile devices with reasonable results, such as in [Qayyum and Şah 2018], [Rattani et al. 2018] and [Elhassouny and Smarandache 2019], selected from a literature search due to their similarity to this work.

In [Qayyum and Şah 2018] an IOS application for food image recognition was developed. It was tested to identify 101 classes of different food types and the range of accuracy obtained was from 86 to 97%. Inference time ranged from 5 to 7 seconds.

In [Rattani et al. 2018] CNNs were used for biometric authentication. A fusion of biometric information from the left eye, the right eye, and the gender were used as input. A total of 550 volunteers provided the input dataset and they were able to obtain a 96.9% accuracy for the authentication using a custom-developed CNN.

Finally, in [Elhassouny and Smarandache 2019] a CNN architecture inspired in MobileNet was trained with 7176 images with 10 different classes of tomato leaf diseases. The trained CNN was used to create a mobile application to classify those diseases and they were able to obtain a 90,3% accuracy.

4. Experimental Setup

In this paper, a dataset containing 829 butterfly images [Wang et al. 2009] was used to train and test all CNN architectures provided by the Keras APIs [Chollet et al. 2015] in TensorFlow (TF). Keras was used for its popularity as a framework that abstracts the complexity of dealing directly with TF . It allows developers to create and run deep learning models without having to deal with all complex details of neural networks such as mathematical operations.

These CNNs were all pre-trained with the ImageNet dataset and then the classification layers were replaced for layers that would support the classes from the butterfly dataset. The rest of the layers were left frozen since the chosen dataset was small and the number of parameters large, which would result in overfitting in case they were not frozen [Yosinski et al. 2014].

The models were then trained through 20 epochs with a custom dataset and saved into HDF5 format files since there were no custom objects created in these models. In case there were custom objects, it would have been preferable to save the models into the TF SavedModel format that has support for that. The HDF5 files contained the weight values, model's architecture, model's training architecture, and the optimizer and its state aggregated in a single file per model.

Once the CNNs were trained and saved, the accuracy, memory profile, model complexity, inference time, and model size were measured for each different architecture. Later on, the HDF5 files were converted to the TensorFlow lite (TFL) format which has better compatibility with mobile devices, and all parameters were measured again using TFL to enable the comparison with TF. Some of the parameters such as the inference time and memory profile were measured with the TFL Android benchmark app [Abadi et al. 2015a] natively on a mobile device. The specifics of how the study was conducted are explained in the next subsections.

4.1. Dataset

As mentioned before, the selected dataset contains 829 images of butterflies [Wang et al. 2009]. Within those images, there are 10 species of butterflies with a range of 55 to 100 images of each. We partitioned the dataset so 664 (80%) images were used for training and 165 (20%) images were used for validation. All images were resized to a 128px square to fit almost all CNNs input layers architectures from Keras API [Abadi et al. 2015b].

4.2. CNN Architectures

All available CNN architectures in the Keras API inside TF except for the NASNetLarge and NASNetMobile were used: VGG16, VGG19, DenseNet121, DenseNet169, IceptionResNetV2, InceptionV3, MobileNet, MobileNetV2, NASNetLarge, NASNetMobile, ResNet50, Xception, EfficientNet. NASNetLarge and NASNetMobile were not evaluated in this paper since their input format differs from all the other architectures. Changing the input format would imply on not all models being tested under the same conditions.

4.3. Training

Each CNN was trained with the described dataset through 20 epochs with 32 size batches. They were preloaded with ImageNet weights and had their classification layers at the top

of the network substituted for other ones to have the custom output for 10 butterfly species. All layers except for the classification layers were frozen during the training following TF 's recommendation for small datasets [Abadi et al. 2015c].

4.4. Performance Comparison

The metrics chosen for performance comparison were accuracy rate, model complexity, memory usage, and inference time. These metrics can be found being used in [Rattani et al. 2018], [Tan et al. 2019],[Cheng et al. 2017], [Zhang et al. 2018], [Qin et al. 2018], [Howard et al. 2017], [Yanai et al. 2016] for CNN's comparison under an environment with memory, energy, and processing capacity constraints as shown in Table 1. They can also be found in [Bianco et al. 2018] for CNN benchmarking.

Table 1. Measured parameters in related works

Top-1 Accuracy rate	Top-5 Accuracy rate	Parameter file size (MB)	Memory usage	Computational complexity: FLOPs	Inference Time	Number of trainable parameters	Number of multiply-add operations	Inference Latency	Storage	Paper
X	X	X	X	X	X					[Rattani et al. 2018]
	X	X					X	X	X	[Tan et al. 2019]
X	X		X		X				X	[Cheng et al. 2017]
X				X	X					[Zhang et al. 2018]
X	X			X						[Qin et al. 2018]
	X						X	X		[Howard et al. 2017]
	X	X		X		X				[Yanai et al. 2016]
	X	X	X	X	X	X				[Bianco et al. 2018]

Since we want to compare all feasible architecture, we decided to evaluate the same metrics that we found in the literature. Some of those metrics are directly related, so we choose a representative subset, explained in the following subsections.

Accuracy Rate

The accuracy rate measures the models' accuracy for classifying images in a specific category. It is the number of correct predictions divided by the total number of predictions. In this study, the top-1 accuracy rate was measured with the test set. The top-1 accuracy rate is the accuracy rate obtained considering the models' answer is the expected answer.

Each one of the images from the test set was classified by both the TF and TFL models. The result obtained is the number of images that were correctly classified divided by the total number of images in the test set for each model.

The top-5 accuracy was not measured since the used dataset had only 10 classes.

The top-5 accuracy would be high since it is the percentage of times that the correct answer is in the top five highest probable answers of the model's output.

Model Complexity

The model complexity measures how much processing the model needs. It can be obtained by counting the total amount of parameters, so the models' file sizes were measured in this study since they are directly related.

For a mobile device that has limited processing capabilities and battery, a smaller file is better for less processing and memory usage.

Memory Usage

Memory usage measures the amount of memory needed for classifying an image. In this study, the measured memory usage was the overall memory usage.

With the TFL model, the overall memory usage was measured with the TFL Android benchmark app in a OnePlus 5 android mobile phone. As for the TF model the memory usage was measured with the sum of what the Python [Fabian Pedregosa 2011] measured through the use of the `load_model` method and `predict` from the Keras model API in TF [Abadi et al. 2015b] methods. The `predict` method was used with a single image from the test set on a Windows 10 computer.

For a mobile device that has limited memory available, a smaller result is better.

Inference Time

Inference time measures how long the model takes to classify one image. In this study, it was measured the average time to classify the whole image test set.

For the TFL model, the inference time was measured with the TFL Android benchmark app in a OnePlus 5 android mobile phone. As for the TF model, the inference time was measured with timestamp difference between right before the `predict` method from the Keras model API in TF and after right after. The `predict` method was used with a single image from the test set on a Windows 10 computer.

A low inference time is better in a mobile application for a better user experience, so the user does not have to wait a long time to obtain the classification.

4.5. Infrastructure Characterization

All metrics were obtained using either Python3.8.5 with the Keras framework API available in TF in a computer with an Intel(R) Core(TM) i7-4771 CPU @ 3.50GHz, NVIDIA GeForce GTX 760 graphics processing unit, 16.0GB (RAM), running a Microsoft Windows 10 (OS build 19041.508) Pro operational system or a OnePlus 5 mobile phone with v10.0.0 OxygenOS, Qualcomm Snapdragon 835 CPU, Adreno 540 GPU and 8GB LPDDR4X RAM.

Table 2. Results Table

CNN	Top-1 Accuracy		Inference Time (ms)		Overall Memory Usage (MB)		Model File Size (KB)	
	TF	TFL	TF	TFL	TF	TFL	TF	TFL
DenseNet121	**98.18**	**98.18**	1746	168	214.8	46.25	28249	27289
DenseNet169	96.36	96.36	2251	208	298	67.87	50521	48949
DenseNet201	**98.18**	**98.18**	2727	252	393.3	90.37	72907	70889
EfficientNetB0	94.54	94.54	1127	145	74.3	26.68	16348	15721
EfficientNetB1	95.75	95.75	1607	197	146.6	37.82	26360	25511
EfficientNetB2	95.75	95.75	1581	210	130.6	45.16	31037	30145
EfficientNetB3	96.96	96.96	1589	273	180.2	62.62	42894	41830
EfficientNetB4	94.54	94.54	2311	396	268.5	92.44	69969	68570
EfficientNetB5	93.93	93.93	2760	556	452.4	139.47	112493	110685
EfficientNetB6	89.09	89.09	4142	725	606.7	192.39	161274	159048
EfficientNetB7	91.51	91.51	4975	946	905.7	280.25	251896	248989
InceptionResNetV2	92.72	92.72	2762	329	737	248.18	213611	212262
InceptionV3	89.09	89.09	1062	156	267.5	110.88	85904	85173
MobileNet	95.15	95.15	0415	182	40	29.76	12900	12547
MobileNetV2	94.54	94.54	0648	**29**	**25**	**17.42**	**9271**	**8713**
ResNet101	95.15	95.15	1545	335	546.2	191.25	167488	165986
ResNet101V2	93.93	93.93	1457	334	541.1	191.81	167343	166253
ResNet152	92.72	92.72	2342	464	783.3	247.92	229158	227051
ResNet152V2	92.12	92.12	2251	475	743.7	251.45	228978	227443
ResNet50	94.54	94.54	799	191	276.5	118.05	92703	91854
ResNet50V2	92.12	92.12	856	188	279	118.38	92599	91979
VGG16	91.51	91.51	**208**	508	166.7	175.74	57607	57510
VGG19	88.48	88.48	263	649	228.8	196.10	78357	78253
Xception	89.09	89.09	2451	201	269.8	124.23	81990	81309

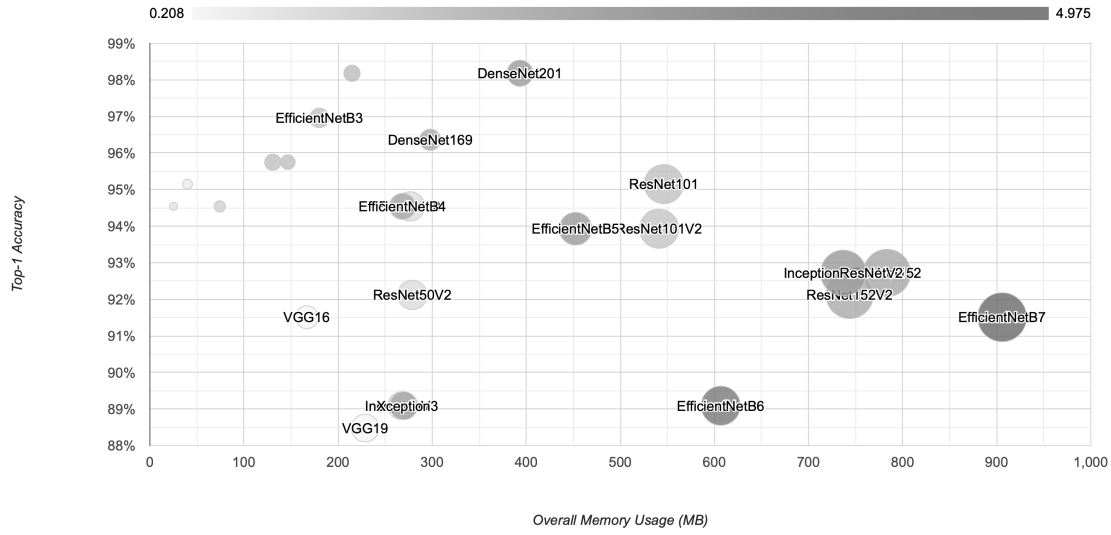

Figure 1. Bubble chart of the TensorFlow CNN models

5. Results

Accuracy Rate

It is possible to see in Table 2 that the Top-1 accuracy rate remained the same through the conversion between TF and TFL for all architectures, which was expected since a difference would mean implementation disparities, preventing the remaining metrics comparisons. Two models had the highest accuracy: DenseNet121 and DenseNet201 with 98.18%. We would like to notice that the accuracy is application-dependent, so a lower accuracy does not mean that the architecture should be discarded.

Model Complexity

It was possible to identify MobileNetV2 as the model with the lowest complexity, an expected result since it was developed to be efficient specifically for mobile and embedded vision applications that have limited memory and computational power.

Overall Memory Usage

MobileNetV2 was the model with the lowest overall memory usage for running both TF and TFL models. It is important to notice that this architecture is comparable only to two EfficientNet variations, being the remaining architectures 5 to 10 times memory-eager.

Inference Time

Inference time was the only measured parameter that had a significant difference between the TF model that was run on a desktop and the TFL model that was run on a mobile device. The difference was due to architectural differences between the environments creating a discrepancy between the latency of different operations. The TF model with the lowest inference time was the VGG16 and the TFL model with the lowest inference time was the MobileNetV2.

6. Conclusion

The analyzed parameters varied between the CNN models but between TF and TFL models of a single architecture, and the conversion between TF and TFL models did not change the model file sizes in a significant way, thus it was possible to identify a few similarities in the accuracy and computational complexity.

The inference time and overall memory usage varied a lot between the TF and TFL models with the same architecture in terms of which ones had the lowest and highest results, however, the accuracy did not change between TF and TFL models within the same architecture, as expected. Our results point out MobileNetV2 as the best model overall for running in a mobile device, with the best results in inference time, overall memory usage, and computational complexity. Although it did not have the highest accuracy, it is acceptable for most image recognition tasks.

Future work should include a comparison with the next generation of machine learning accelerators, being deployed as co-processors by major processor manufacturers. Although MobileNet was developed with constrained devices as target platforms, we believe that we still lack a framework specifically targeting low power devices, that could not even count on a floating-point multiplier. Future works could also focus on low power implementation, enabling devices such as wearables or extremely low power embedded or mobile systems to benefit from image recognition.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015a). TensorFlow: Performance measurement, android benchmark app. `https://www.tensorflow.org/lite/performance/measurement#android_benchmark_app`. (accessed 17 September 2020).

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015b). TensorFlow: Model. `https://www.tensorflow.org/api_docs/python/tf/keras/Model`. (accessed 17 September 2020).

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015c). TensorFlow: Transfer learning and fine-tuning. `https://www.tensorflow.org/tutorials/images/transfer\_learning`. (accessed 17 September 2020).

Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277.

Chen, C.-F., Lee, G. G., Sritapan, V., and Lin, C.-Y. (2016). Deep convolutional neural network on ios mobile devices. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 130–135. IEEE.

Cheng, J., Wu, J., Leng, C., Wang, Y., and Hu, Q. (2017). Quantized cnn: A unified approach to accelerate and compress convolutional networks. *IEEE transactions on neural networks and learning systems*, 29(10):4730–4743.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Elhassouny, A. and Smarandache, F. (2019). Smart mobile application to recognize tomato leaf diseases using convolutional neural networks. In *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, pages 1–4. IEEE.

Fabian Pedregosa, P. G. (2011). Memory profiler, python module for monitoring memory consumption. `https://github.com/pythonprofilers/memory_profiler`.

García-Alba, J., Bárcena, J. F., Ugarteburu, C., and García, A. (2019). Artificial neural networks as emulators of process-based models to analyse bathing water quality in estuaries. *Water research*, 150:283–295.

Guo, T. (2018). Cloud-based or on-device: An empirical study of mobile deep inference. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 184–190. IEEE.

Haas, R. and Davies, J. (2020). What's powering artificial intelligence? Technical report, ARM (White Paper).

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, X., Grandvalet, Y., and Davoine, F. (2020). A baseline regularization scheme for transfer learning with convolutional neural networks. *Pattern Recognition*, 98:107049.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA.

Qasaimeh, M., Denolf, K., Khodamoradi, A., Blott, M., Lo, J., Halder, L., Vissers, K., Zambreno, J., and Jones, P. H. (2021). Benchmarking vision kernels and neural network inference accelerators on embedded platforms. *Journal of Systems Architecture*, 113:101896.

Qayyum, O. and Şah, M. (2018). Ios mobile application for food and location image prediction using convolutional neural networks. In *2018 IEEE 5th international conference on engineering technologies and applied sciences (ICETAS)*, pages 1–6. IEEE.

Qin, Z., Zhang, Z., Zhang, S., Yu, H., and Peng, Y. (2018). Merging-and-evolution networks for mobile vision applications. *IEEE Access*, 6:31294–31306.

Rattani, A., Reddy, N., and Derakhshani, R. (2018). Multi-biometric convolutional neural networks for mobile user authentication. In *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6. IEEE.

Rawat, W. and Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828.

Wang, J., Markert, K., and Everingham, M. (2009). Learning models for object recognition from natural language descriptions. In *BMVC*, volume 1, page 2.

Yanai, K., Tanno, R., and Okamoto, K. (2016). Efficient mobile implementation of a cnn-based object recognition system. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 362–366.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856.