university of groningen / faculty of science and engineering

Is machine learning energy-efficient enough to be used in mobile fashion applications?

Bachelor's Project Thesis

Paul Pintea, s3593673, p.pintea@student.rug.nl,
Supervisors: dr. C.P. Lawrence and prof. dr. H. Jaeger

Abstract: This report aims to analyze the energy efficiency of readily available mobile neural networks: MobileNet-V2, EfficientNet-lite0 and ResNet-50. MobileNet-V2 represents a lightweight neural network created for efficiency. EfficientNet-lite0 is a neural network built on a modern architecture that uses active scaling. ResNet-50 is a powerful neural network with 50 layers and high computational costs. All neural networks have been retrained on a modified version of the Fashion Product Images Dataset (Aggarwal, 2019). Testing focused on the energy consumption of the neural networks under an image classification task. The task was performed in a custom fashion-oriented application. Results show that MobileNet-V2 is the most energy-efficient, followed by EfficientNet-lite0 and then ResNet-50. A mean CPU usage of 26.5 to 28% was recorded for all three networks, while running on a Samsung A8 (2018). This represents a 12 to 13% increase from the simple version of the application. From the computational costs, ResNet-50 was expected to perform 10 times slower than the other two neural networks. This was not the case, from testing ResNet-50 ran 2 times longer, not 10 times, compared to the other two neural networks. This might be caused by the way multi-threaded mobile architectures work. Less demanding tasks occupy fewer threads and are not optimized as well, whilst more demanding tasks are split across multiple threads to improve energy efficiency. Even so, all three NNs seem energy efficient enough for deployment in different mobile environments.

1 Introduction

The fashion industry has changed drastically in the last 20 years. In current times, the industry has already shifted towards the "fast fashion" approach. This was realized by creating more "fashion seasons" compared to the classic 4 seasons present in past decades. The most important aspect of this type of shift is that it impacts consumers' purchasing habits and their view on trends (Bhardwaj and Fairhurst, 2010). Due to fashion quickly changing, consumers might be inclined to shop compulsively which only leads to misspend time and a lot of waste that could be avoided. As a response, Fashion-oriented applications which use machine learning might be used to help and guide consumers' buying habits. Neural Networks have already become a widespread tool for solving complex tasks in many fields. Furthermore, many previous papers have tested different types of neural networks on how well are they able to label and detect fashion items (Bhatnagar, Ghosal, and Kolekar, 2017; Hara, Jagadeesh, and Piramuthu, 2016). In these articles, the main analysis focus is the accuracy of the neural networks, but in mobile environments, the energy efficiency might be even more important. So, are neural networks energy efficient enough for mobile use?

The main interest of this report is to analyze and show the differences in the energy efficiency of different well known mobile neural networks. MobileNet-v2 is built by using the classical approach of using depth-wise separable convolutions.To improve efficiency, the model also uses an inverted residual structure, built on a $narrow \longrightarrow wide \longrightarrow narrow$ approach. This way, each inverted block takes an input with a reduced number of channels and its output will also have a reduced number of channels (Sandler, Howard, Zhu, Zhmoginov, and Chen, 2018). EfficientNet-Lite0 is

built using a new approach of scaling in three dimensions: resolution, depth, width, by using a compound coefficient. The result is a new model which is supposed to yield better efficiency and accuracy compared to the classical neural networks (Tan and Le, 2019). ResNet-50 represents a powerful neural network with 50 total layers which is built by using classic residual bottleneck layers. The result is a big neural network with great accuracy, but with a big computational cost as well (He, Zhang, Ren, and Sun, 2015).

Transfer Learning has been used to retrain the neural networks to accommodate the focus of this study. The training has been done on a modified version of the Fashion Product Images Dataset (Small) (Aggarwal, 2019). The original dataset included items such as "accessories" which had to be eliminated since they did not accommodate the study. Furthermore, the dataset had to be rearranged to fit the requirements of the transfer learning procedure.

For testing, a custom Android application has been created. The application uses the CameraX module from Android Jetpack to take a picture. That picture is then labelled using one of the mentioned neural networks. The correctness of the labelling and the confidence level are taken as measurements for comparing the models.

Other measurements such as the CPU, memory, battery usage and run-time are also taken as variables in the comparison. To record the variables, the Resource Profiler pre-implemented in Android Studio is used.

The methods section will explain in detail the main structure of the project, how training was performed and all the terms which were introduced. In the results section, not only will the energy consumption be presented, but a new "efficiency score" will be introduced. This score comes as an aid for the reader to better visualize which neural network is the most efficient, without any fine-tuning. The discussion section will explain the results and it will present the positive and negative aspects of each neural network. Furthermore, this section will also offer suggestions for improvements in future studies.

2 Methods

This project is divided into three important blocks. The first block is filtering and rearranging the dataset to accommodate the retraining procedure. The second block is retraining the neural networks using transfer learning. The final block is the custom application used for testing and running the neural networks. A visual representation of the project's architecture is depicted in Figure 2.1. First of all, in this section, the architecture of each neural network will be presented in detail. After, the project's three blocks and their procedures will be explained.

2.1 The MobileNet-V2 architecture

The MobileNet-v2 neural network family still maintains the classical approach of using depth-wise separable convolutions, which was used in the first generation family of MobileNet-v1 (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto, and Adam, 2017). In addition, the second generation (MobileNet-v2) is using an inverted residual structure which increases the accuracy of the model (Sandler et al., 2018). The inverted block first widens the input by using a 1x1 convolution layer, then a 3x3 depth-wise convolution layer is used (which decreases the number of parameters) and then another 1x1 convolution layer is used to shrink the number of channels. A graphical representation of the convolutional blocks can be seen in Figure 2.2. Overall, MobileNet-v2 uses a full convolutional layer with 32 filters as its starting layer, followed by 19 inverted bottleneck blocks. The full model architecture is depicted in Table 2.1. Each bottleneck sequence can either start with the Stride 1 or 2 blocks (seen in Figure 2.2) and they are then continued with the Stride 1 block. The starting block is also mentioned in Table 2.1. The model's computational cost for one image is around 300 million floating point operations per second (FLOPS) (Tan and Le, 2019).

2.2 The EfficientNet-lite0 architecture

EfficientNet-lite0 is built on a modern architectural design that improves both the accuracy and the computational cost. The model also uses inverted

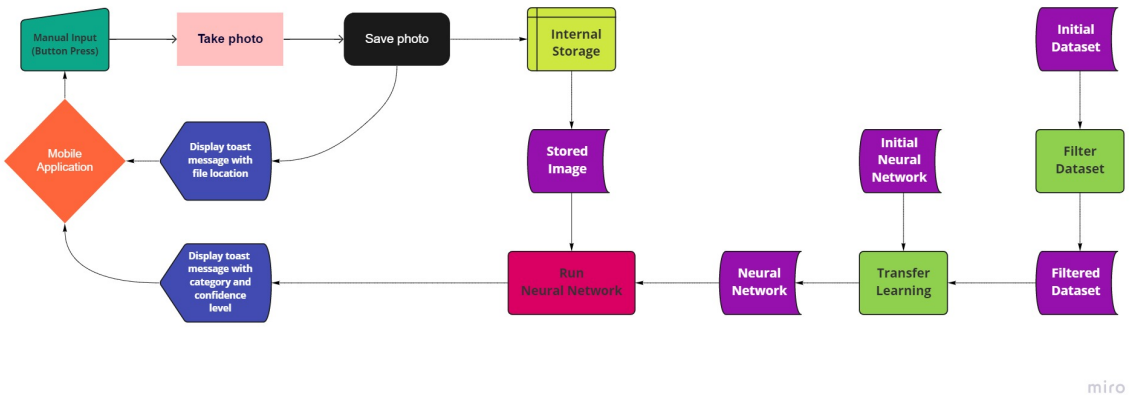

Figure 2.1: Flowchart of how the project functions. On the far right, the dataset filtering block and the neural network retraining block are depicted. On the left, the application block is depicted.

Table 2.1: MobileNet-V2 complete architecture, reproduced from Sandler et al. (2018)

Operator	No. Layers	Starting Stride
conv2d	1	2
bottleneck	1	1
bottleneck	2	2
bottleneck	3	2
bottleneck	4	2
bottleneck	3	1
bottleneck	3	2
bottleneck	1	1
conv2d 1x1	1	1
avgpool 7x7	1	-
conv2d 1x1	-	

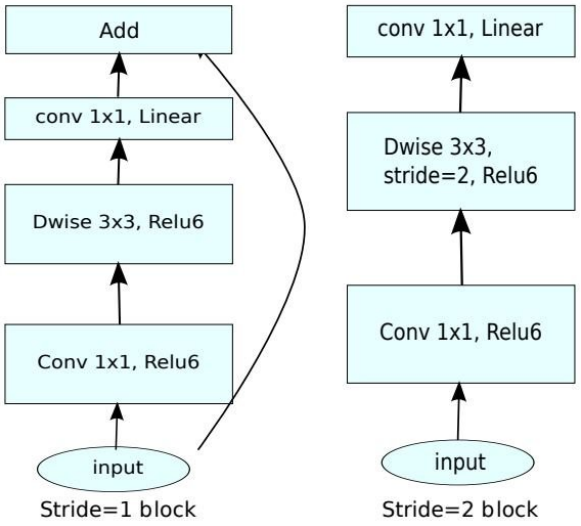

Figure 2.2: Graphical representation of the convolutional blocks in MobileNet-v2. Taken from Sandler et al. (2018)

bottleneck Convolutional layers (also called MBconv blocks) for its primary block (Sandler et al., 2018), see Figure 2.2 for graphical reference. In addition EfficientNet-lite0 also uses compound coefficients to uniformly scale on all dimensions of resolution/depth/width, to improve the accuracy and computational costs of the network. This scaling works on the concept that higher quality pictures need more layers to better define fine-grained patterns. So, compound scaling, instead of just increasing the number of layers (increasing the depth of the network), it also increases the resolution of the layers and the width of the layers, just like it is depicted in Figure 2.3. If EfficientNet-lite0 needs to use 2^N more resources the constant scaling coefficients will be raised to the power N as well, so: α^N, β^N and γ^N. For the base model (EfficientNet-lite0, also named EfficientNet-B0), the optimal scaling coefficients have been determined to be $\alpha = 1.2$ for depth, $\beta = 1.1$ for width and $\gamma = 1.15$ for resolution (Tan and Le, 2019). The scaling coefficients can be determined using a small grid search on the model. Overall, the full model architecture is presented in Table 2.2. The model's computational cost for one image is around 407 million FLOPS (Tan and Le, 2019).

2.3 The ResNet-50 architecture

ResNet-50 is the most classic of all the neural networks used. As the base building blocks, ResNet-50 uses simple residual bottleneck layers. Compared to the smaller neural networks in the ResNet family, ResNet-50 uses 3 layers in one block instead of 2, see Figure 2.4. These types of blocks work on a *wide* $\longrightarrow$ *narrow* $\longrightarrow$ *wide* approach. The ResNet-50 blocks start with a 1x1 convolution layer which compresses the input, followed by a 3x3 convolution layer and then another 1x1 convolution layer which restores the number of channels for the output (He et al., 2015). A visual representation is depicted on the right side of Figure 2.4. In total, ResNet-50 has 50 layers and the full architecture is depicted in Table 2.3. The model's computational cost for one image is around 4.1 billion FLOPS (Tan and Le, 2019).

Table 2.2: EfficientNet-lite0 complete architecture, reproduced from Tan and Le (2019)

Operator	No. Layers
Conv 3x3	1
MBConv1, k3x3	1
MBConv6, k3x3	2
MBConv6, k5x5	2
MBConv6, k3x3	3
MBConv6, k5x5	3
MBConv6, k5x5	4
MBConv6, k3x3	1
Conv1x1 & Pooling & FC	1

2.4 The dataset

Fashion Product Images Dataset(Small) (Aggarwal, 2019) contains 44000 fashion items with a resolution of 60x80 pixels. These images are stored in the same folder and a table with all the information about the images was offered. The dataset had to be filtered because items such as "watches" are not important for the task. A new dataset was created, only from the "Apparel" and "Footwear" categories. From these categories, the 'Innerwear', 'Loungewear and Nightwear' and 'Socks' subcategories have been eliminated. after the filtering process the dataset was reduced to 28321 images. To accommodate TensorflowLite Model-Maker the images had to be divided into separate folders based on their category. 5 categories have been created: "bottoms" with 2693 items, "complete" (images which represent entire outfits) with 106 items, "dress" with 905 items, "shoes" with 9219 items and tops with 15398 items. See algorithm 2.1 for a pseudo-code explanation of the procedure. After empirically analyzing the dataset, the "complete" category was dropped before the retraining procedure, due to its relatively small size in compari-

Table 2.3: ResNet-50 complete architecture, reproduced from He et al. (2015)

Operator	No. Layers
conv1 7x7	1
maxpool 3x3	1
3layer block	3
3layer block	4
3layer block	6
3layer block	3
avgpool, 1000-d fc, softmax	1

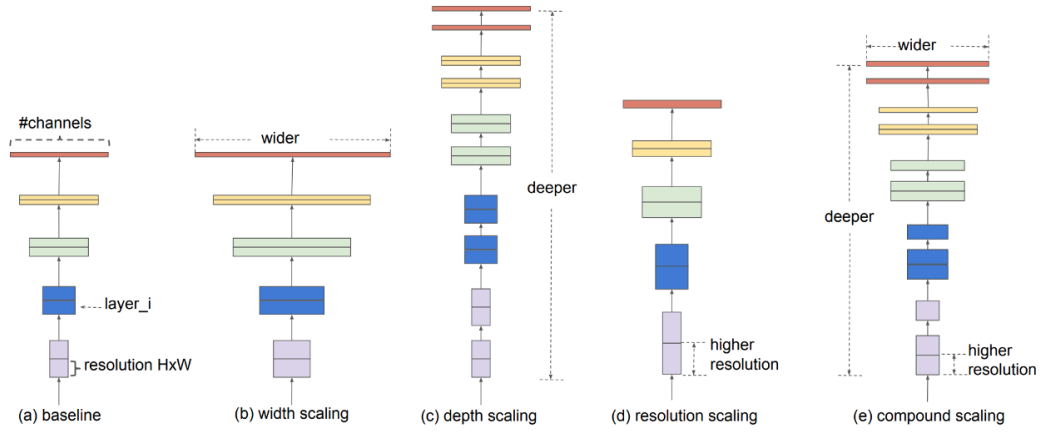

Figure 2.3: Graphical representation of the compound scaling structure, compared to other simple types of scaling. Taken from Tan and Le (2019)

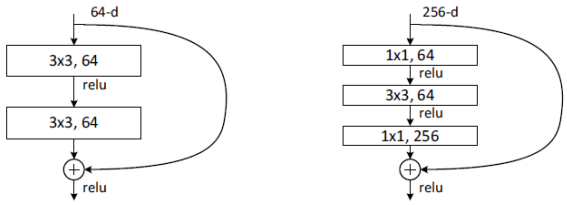

Figure 2.4: Graphical representation of the building blocks for the ResNet family. Left: architecture for smaller networks from the family. Right: architecture for ResNet-50 and bigger networks in the family. Taken from He et al. (2015)

Algorithm 2.1 Dataset Processing

Require: *dataset* $\Leftarrow$ Fashion Product Images Dataset(Small) (Aggarwal, 2019)
Ensure: Filter *dataset*
{Separate the dataset into categories}
for all *items* in *dataset* == "$clothing_{type}$" **do**
Save *items* in "$clothing_{type}$" folder
end for

son with the other categories. Therefore, the final dataset included a total of 28215 items.

2.5 Transfer Learning

To better fit the task of this study, the neural networks had to be retrained. Transfer learning is a procedure in which the knowledge of the model (such as features and weights) is kept, but the input layers are changed and retrained on a new dataset. To perform this task TensorflowLite Model-Maker is used. Model-Maker already has in its library all three neural networks: EfficientNet-lite0, MobileNet-v2 and ResNet-50. A new model is created by inputting the new train and validation data, and which neural network should be used by the image classifier function in Model-Maker. 80% of the entire dataset is reserved for training, 10% for validation and 10% for testing. After retraining, the model is tested for performance on the test data and then saved, see algorithm 2.2.

2.6 The Android application

The testing phase will be performed in a custom Android application. The main activity represents a camera view implemented with the CameraX package from Android Jetpack. Furthermore, a simple "Add Item" button is implemented at the bottom of the view, see Figure 2.5.

The "Add Item" button is used to take a photo and save it to the device's internal storage. Imme-

Algorithm 2.2 Transfer Learning procedure

Require: TensorflowLite Model-Maker
Ensure: $dataset \Leftarrow new\_data$
$train \Leftarrow 80\% dataset$
$validate \Leftarrow 10\% dataset$
$test\_data \Leftarrow 10\% dataset$
$img\_class = Model - Maker.image\_classifier$
$Model \Leftarrow img\_class(train, model_{type}, validate)$
Test the $Model$ on $test\_data$
Save $Model$

Figure 2.5: The View of the MainActivity.

Figure 2.6: The View of the MainActivity after the image is processed and the label and confidence level appear on the screen.

Algorithm 2.3 Image Processing

Require: *model* ⇐ Neural Network, Jetpack CameraX, ML Kit's Image Labeler
Ensure: OnButtonClick(SaveImageToStorage)
threshold ⇐ Threshold acceptance level
labeler ⇐ ImageLabeling(*model*, *threshold*)
labeler.process(*image*)
Print label and confidence level

diately after, the image labeller from Android's ML kit starts processing that image, using one of the three neural networks. The neural network has to be manually specified in the code of the application. The acceptance threshold has been set to 30% confidence, to ensure that the application will yield a choice. After the image is processed a toast message appears showing the first choice label and the confidence level for that label, see Figure 2.6 for reference. A pseudo-code explanation of the process is offered in algorithm 2.3.

2.7 Testing Phase

All testing has been performed manually by repeatedly taking images of clothing items. For each of the four categories, three separate clothing items have been photographed. Each item was processed 5 times (5 photos), to account for differences in positioning and hand movement. Some of the images taken can be seen in Figure 2.7. In total, 180

Figure 2.7: Illustration of some of the images taken during testing.

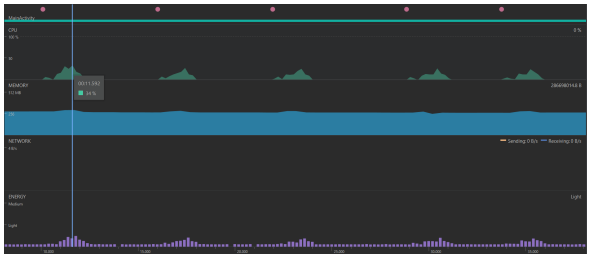

Figure 2.8: Illustration of the main view in the Android Studio Profiler.

data-points have been manually recorded for each neural network. These data-points are equally split between CPU usage, runtime and confidence level. For reference, 60 data-points were also recorded for a version of the application which does not run a neural network. The CPU usage and the run-time have been recorded by observing the measurements given by Android Studio's Profiler (see Figure 2.8). If a classification failure occurred (an item was misclassified), a 0 was recorded as the confidence level, instead of the actual confidence level. Furthermore, testing was performed on an Android device, not in a virtual environment. Samsung's A8 2018 phone has been used. This device is promoted on multiple websites to have a computational power of 29 GFLOPS (Samsung Galaxy A8 (2018): Specs, Benchmarks, and User Reviews, n.d.).

3 Results

3.1 Confidence Levels and accuracy

All three neural networks have performed poorly in the accuracy area. The neural networks had many misclassifications and none of the neural networks have managed to correctly classify the "dress" category. This complete misclassification, might have been caused by the relatively small number of items present in the "dress" category compared to the other three categories. Furthermore, dresses usually have similar features to many of the items that could be found in the "tops" category. This aspect might have impacted the choice of the models.

For the "tops" category, MobileNet-v2 and EfficientNet-lite0 have completely misclassified two out of the three items, whilst ResNet-50 has correctly classified all the items at least one time. In the "bottoms" category ResNet-50 has completely misclassified one out of the three items, whilst the other two neural networks have correctly classified all the items at least once. The "shoes" category showed the best result with higher confidence levels compared to the rest of the categories. Only EfficientNet-lite0 has completely misclassified one of the items, whilst the other two neural networks have correctly classified all the items, multiple times.

The mean confidence levels of the neural networks can also reflect their accuracy. MobileNet-v2 showed the highest mean confidence scores in the "shoes" category, followed by the "bottoms" and then "tops" category. EfficientNet-lite0 yielded the highest mean confidence in the "bottoms" category, followed by the "shoes" and then the "tops" category. The overall mean confidence level was around 24% for the MobileNet-v2 and 31% for the EfficientNet-lite0. ResNet-50 also showed the highest mean confidence level in the "shoes" category, but it was followed by the "tops" and then the "bottoms" category. Its overall mean confidence level was recorded at around 28%. For the exact mean confidence levels, see Table 3.1.

Table 3.1: Mean confidence levels yielded by each of the neural networks for each category.

	MobileNet	EfficientNet	ResNet
Shoes	53.85%	39.79%	65.59%
Tops	13.20%	2.68%	33.17%
Bottoms	29.58%	83.12%	12.31%
Dress	0	0	0
Overall	24.16%	31.39%	27.77%

3.2 Resource Consumption

All neural networks show mostly the same mean CPU usage. MobileNet-v2 mean CPU usage ranges from around 25% to 29%, between the four categories. The overall mean CPU usage of MobileNet-v2 is computed at around 26.5%. The mean CPU usage of EfficientNet-lite0 ranges from 26% to 28%, across the categories. Its overall mean CPU usage is recorded at around 26.5%. ResNet-50 has a mean CPU usage ranging from 27% to 29%, between the categories. ResNet-50 overall mean CPU usage is calculated at around 28%. For the detailed mean CPU usage levels, see Table 3.2.

As a point of reference, the CPU usage of the application was recorded without running any of the neural networks. The mean CPU usage of this simple version of the application was calculated at around 14.5%. Overall, the neural networks showed an increase between 12% and 13% in the CPU usage, when compared to the simple version of the application, see Figure 3.1

Furthermore, the measured RAM usage and battery usage for the three neural networks were mostly the same. The battery usage was always recorded at the lowest level "Light". The RAM usage stayed mostly constant at around 256MB of memory. Both the battery and RAM usage showed small spikes in correlation to the CPU usage spikes, see Figure 2.8 for reference.

Since similar resource utilization was recorded across the models, the CPU usage had to be normalized by multiplying it to the run-time of the neural network, see Equation 3.1. For a more precise estimation, the area under the CPU usage would have to be calculated, but the simple multiplication from Equation 3.1, should offer a good base for comparing the models. Even if the run-time was recorded in milliseconds, in the equations it was

Table 3.2: CPU usage levels yielded by each of the neural networks for each category.

	MobileNet	EfficientNet	ResNet
Shoes	25.93%	27.53%	28.86%
Tops	29.4%	25.8%	28.13%
Bottoms	24.6%	26%	26.53%
Dress	26.53%	26.06%	27.8%
Overall	26.61%	26.35%	27.83%

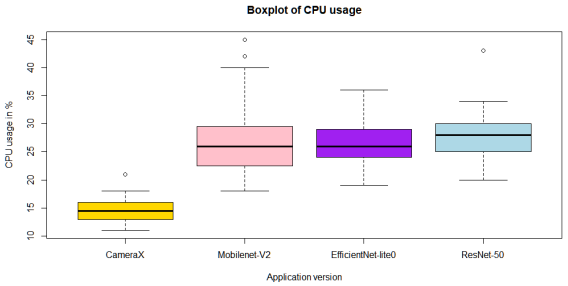

Figure 3.1: CPU usage of the simple application and the neural networks' versions of the application.

used in seconds. MobileNet-V2 measured a mean run-time of around 207 milliseconds. EfficientNet-lite0 on average ran for about 228 milliseconds and ResNet-50 for around 487 milliseconds, see Table 3.4 for exact numbers. From the normalized CPU usage, ResNet-50 took more than two times the resources, to process one image, compared to MobileNet-V2 and EfficientNet-lite0. Check Figure 3.2 for a graphical representation.

$$normalizedCPU[i] = CPU[i] * meanRuntime \tag{3.1}$$

The theoretical expectations were also computed as a reference point against the normalized CPU usage. These expectations represent the CPU usage level, that each neural network should consume during testing. The theoretical CPU usage was calculated by dividing each model's theoretical computational cost (see Table 3.3) by the total power of the device (29 billion FLOPS). The result was then multiplied by 100 to compute percentages (see Equation 3.2). From the theoretical expectations, ResNet-50 was supposed to take around ten times more resources, to process one image, compared to MobileNet-V2 and EfficientNet-lite0. Check Figure 3.2 for a graphical representation.

$$expectations = 100 * \frac{computationalCost}{devicePower} \tag{3.2}$$

The efficiency score is used to better visualize how did the neural networks perform, without any fine-tuning. The score was computed by taking the best average Confidence level which was divided

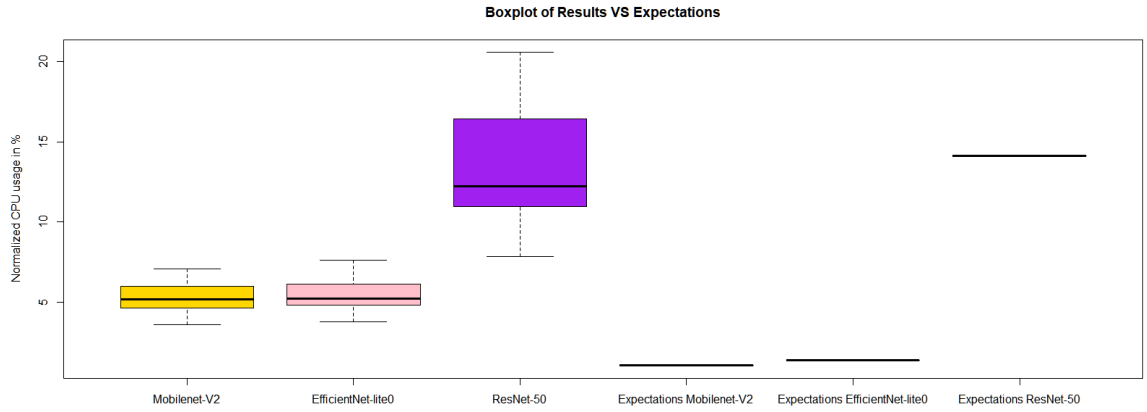

Figure 3.2: Normalized CPU usage and Theoretical expectations for the neural networks.

by the mean CPU usage and the mean run-time, for each neural network (see Equation 3.3). Overall, EfficientNet-lite0 yielded the best and highest score, followed by MobileNet-v2 and then ResNet-50. See Table 3.4 for exact values.

$$effScore = \frac{bestAvgConfidenceLevel}{meanCPU * meanRuntime} \tag{3.3}$$

Table 3.3: Computational cost for each of the neural networks.

	MobileNet	EfficientNet	ResNet
FLOPS	300 Million	407 Million	4.1 Billion

Table 3.4: All the measurements needed to calculate the Efficiency Score.

	MobileNet	EfficientNet	ResNet
Best confLevel	33.23%	37.14%	48.73%
CPU	26.61%	26.35%	27.83%
runtime	206.93ms	228.03ms	486.5ms
effScore	0.006034	0.006181	0.003598

4 Discussion

4.1 Model Comparison

Overall, the neural networks have performed poorly in recognizing the items(see Table3.1 for exact results). Many misclassifications were recorded for all the models. Furthermore, none of the models managed to recognize the "dress" category. This was to be expected since the neural networks were retrained on the default settings and no fine-tuning was performed. The main focus of the paper was not the accuracy, but rather the energy efficiency of the models and whether they can be deployed in mobile applications. Even so, recording and observing the accuracy and the models' choices have offered more insight on how the neural networks behaved. MobileNet-V2 was the most equilibrated out of the models. EfficientNet-lite0 showed a high affinity for misclassifying other items as "bottoms". In the same manner, ResNet-50 showed an affinity for the "tops" category, but not as pronounced as EfficientNet-lite0.

When it comes to energy consumption, all three models measured a mean CPU usage between 26.5% and 28% (see Table 3.2 for exact measurements). This represents just a small increase of 12 to 13% compared to running the application without any of the models. Since the increase was relatively small it implies that all three neural networks could be used in mobile environments without problems. Even so, to better distinguish be-

tween the energy performance of each model, the CPU usage had to be normalized using the runtime.

By normalizing the CPU usage the models can also be compared to the theoretical expectations. MobileNet-V2 and EfficientNet-lite0 had higher CPU usage levels compared to the expectations for the device that was used. ResNet-50 has mostly stayed in line with the expectations regarding its CPU usage levels. From their computational costs, ResNet-50 was expected to perform around ten times slower than the other models. From testing, it was observed that actually, ResNet-50 performed only two times slower compared to the other two neural networks. This might indicate that models with a higher computational cost could be used in mobile environments without a major loss in energy-efficiency. Even so, MobileNet-v2 and EfficientNet-lite0 are more energy efficient, compared to ResNet-50, which indicates that they might be better suited for mobile environments. As an extra point for comparison, the models' overall performance was analyzed as well.

The efficiency score indicates which neural network has performed the best overall, without any fine-tuning. EfficientNet-lite0 and MobileNet-V2 show mostly the same performance. Still, EfficientNet-lite0 is ahead by a small margin. ResNet-50 is almost two times less efficient, due to its longer run-times and similar accuracy to the other models. Therefore, the efficiency score shows that EfficientNet-lite0 might be the most optimal solution for mobile applications.

4.2 Future Studies

Overall, the energy consumption did not vary drastically across the different categories and items. This would indicate that increasing the number of categories or items, would not have an impact on the conclusions made in regards to energy consumption. In contrast, the confidence levels and the number of misclassifications vary a lot between the categories, items and models. Therefore, in the future, it would be important to see what impact does fine-tuning have on the overall performance of the neural networks. To start, future studies should try to augment the dataset to reduce the discrepancy between the number of items in each of the categories. This way, the models would be less likely to form affinities for one of the categories. In addition, by also focusing and recording the error-rate of the models a more complex efficiency analysis could be performed. This type of analysis would better show which model is the most optimal for mobile environments.

Furthermore, in the future, larger neural networks could be added to the analysis. One example of such models would be EfficientNet-B4 which has similar computational costs to ResNet-50 (Tan and Le, 2019). This way, it could be observed if the discrepancy between the theoretical expectations and the actual results can be observed for other models as well, not just for ResNet-50.

Finally, to be able to ensure that future studies will have the possibility to reproduce all the results given in this report, a GitHub repository has been created which includes all the necessary data and scripts important for testing and recreating the results Pintea (2022).

4.3 Conclusion

All three neural networks seem to be energy efficient enough to be deployed in mobile environments. All models have their benefits and downsides. MobileNet-V2 consumes the least amount of energy due to its short run-times and low CPU usage. The trade-off is made when it comes to accuracy. MobileNet-V2 has the smallest confidence levels and some misclassifications. EfficientNet-lite0 seems to be a middle ground and the most optimal model for mobile applications. It is very energy efficient and it has relatively high confidence levels, but it also has an affinity for misclassifying items as "bottoms". ResNet-50 should be the easiest model to implement. The neural network offers very high confidence levels without fine-tuning, but it also consumes the most amount of energy and it has misclassifications.

Finally, to create a more complex efficiency analysis, the error-rate should also be taken into consideration to better illustrate the most optimal neural network. The dataset should also be augmented to have similar numbers of items in each category. This way, the models would be less prone to form affinities for a specific category. Larger networks could also be included to be able to inspect the discrepancy between the theoretical expectations and the actual results, shown by ResNet-50.

References

Samsung Exynos 7885 Specs, Features and performance score 28th January 2022. URL https://www.giznext.com/mobile-chipsets/samsung-exynos-7885-chipset-gnt.

Samsung Galaxy A8 (2018): specs, benchmarks, and user reviews. URL https://nanoreview.net/en/phone/samsung-galaxy-a8-2018.

P. Aggarwal. Fashion product images (small), 2019. URL https://www.kaggle.com/paramaggarwal/fashion-product-images-small/metadata.

Vertica Bhardwaj and Ann Fairhurst. Fast fashion: response to changes in the fashion industry. *The international review of retail, distribution and consumer research*, 20(1):165–173, 2010.

Shobhit Bhatnagar, Deepanway Ghosal, and Maheshkumar H. Kolekar. Classification of fashion article images using convolutional neural networks. In *2017 Fourth International Conference on Image Information Processing (ICIIP)*, pages 1–6, 2017. doi: 10.1109/ICIIP.2017.8313740.

Kota Hara, Vignesh Jagadeesh, and Robinson Piramuthu. Fashion apparel detection: the role of deep convolutional neural network and pose-dependent priors. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9. IEEE, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Paul Pintea. GitHub Repository for WardrobeAI, a Bachelor Thesis, 02 2022. URL https://github.com/Paul0808/WardrobeAI.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL http://arxiv.org/abs/1905.11946.