Neural Network Project - MobileNet v3

Salvatore Cognetta 1874383

January 2022

MobileNetV3 implementation for Neural Network course project, Artificial Intelligence and Robotics, Sapienza.

Contents

1 Introduction

In the last decade the mobile device, like smartphones, computational power is rapidly increased, almost reaching power of low-budget computer. This led to the study of the possibility of using neural networks on these devices.
As it is known, using machine learning techniques is high demanding in terms of computation, for this reason it's needed to make some assumptions and observations before go into this applications. At the current state is unthinkable to train a complex neural network on a single mobile device with so low computational power, compared to dedicated hardware and data center typically used for this kind of task. Even if edge computing and the possibility of using a group of devices simultaneously also for neural network train can be an interesting field of study, this work is focused on the application and implementation of lightweight networks, expressly engineered, that are trained on dedicated hardware but can be used for inference on mobile devices, like smartphones.

In this sense, the aim of this work is the implementation and uses of what is, at current state, the state of the art for this kind of networks: MobileNetV3 [1]. The authors proposed two kind of architectures, called MobileNetV3 Small and Large. The first one is the direct evolution of MobileNetV2 [2], while the second one is based on the work expressed in MnasNet-A1 [3].

The goal of the reference paper is to develop the best possible mobile computer vision architectures optimizing the accuracy-latency trade off on mobile devices. To accomplish this they introduced (1) complementary search techniques, (2) new efficient versions of nonlinearities practical for the mobile setting, (3) new efficient network design, (4) a new efficient segmentation decoder.
The solution proposed in MobileNetV3 is therefore implemented and applied to two different dataset for image detection and classification: MNIST [4] and CIFAR10 [5].

2 Related works

This work, as already said, is an evolution of MobileNetV2, trying to find a quasi-optimal model for devices for low computational power. To realize this solution a trade-off between accuracy of the model and its complessity during inference (efficiency) is obviously needed. Both novel handcrafted structures and algorithmic neural architecture search have played important roles in advancing this field.

For example, as stated by the authors, in SqueezeNet [6] 1x1 convolutions with squeeze and expand modules are extensively used, primarily focusing on reducing the number of parameters. More recent works shifts the focus from reducing parameters to reducing the number of operations (MAdds) and the actual measured latency.
MobileNetV1 [7] employs depthwise separable convolution to substantially improve computation efficiency; while MobileNetV2 [2] expands on this by introducing a resource-efficient block with inverted residuals and linear bottlenecks.
More studies on the subject, like ShuffleNet [8], CondenseNet [9] and ShiftNet [10] were taken into account by the authors for the uses of group convolution at training phase and shift operation with point-wise convolutions in order to replace expensive spatial convolutions.

Another major study performed regards the automatization of the architecture design search, which can be done using Reinforcement Learning to search efficient architectures with interesting and very good accuracy.
Recently, the authors in MnasNet [3] explored a block-level hierarchical search space allowing different layer structures at different resolution blocks of a network.

3 Solutions

All these above mentioned ideas and studies are taken into account by the authors and used, together with other innovative proposal, reaching state of the art network for mobile devices, as explained in this section.

3.1 MobileNetV3 Block

In the previous work done in MobileNetV2 the authors introduced a linear bottleneck and inverted residual structure in order to make an efficient layer structures by leveraging the low rank nature of the problem. This structure is defined by an 1x1 expansion convolution followed by depthwise convolutions and a 1x1 projection layer. The input and output are connected with a residual connection if and only if they have the same number of channels.
Using these linear bottleneck in combination with lightweight attention modules based on squeeze and excitation into the bottleneck structure, introduced by MnasNet architecture, the final bottleneck output is defined in Figure 1.

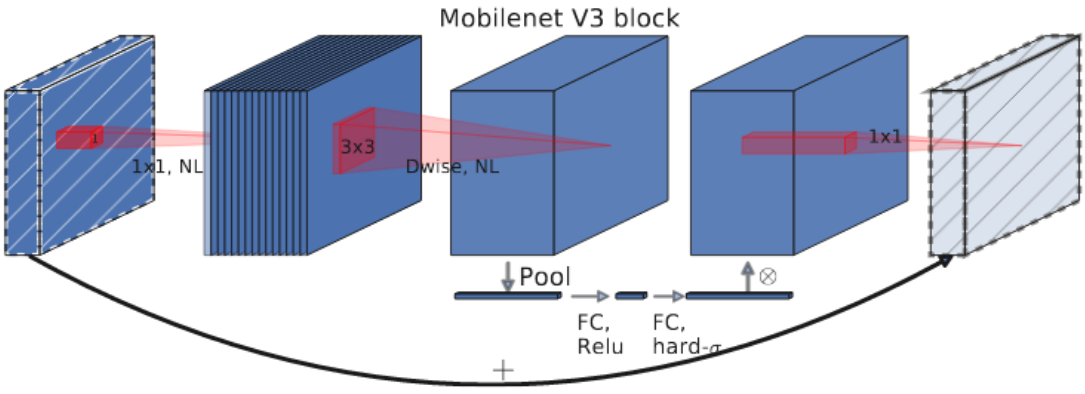

Figure 1: MobileNetV3 botlle-neck

3.2 Network search

Another technology imployed by the authors for the architecture definition, is platform-aware Neural Architecture Search (NAS) to find the global network structures. The authors simply reuse the same MnasNet-A1 architecture as initial Large mobile model, and then apply NetAdapt and other optimizations on top of it.

3.3 Nonlinearities

As a replacement for the ReLU function, the swish nonlinearity is taken into account. It can be defined as

$$swish(x) = x \cdot \sigma(x)$$

This nonlinearities improves the overall accuracy, however it's more expensive because of the sigmoid function, difficult to compute on mobile devices.
For this reason it's replaced with its piece-wise linear hard analog:

$$hswish(x) = x\frac{ReLU6(x+3)}{6}$$

This hard version have no big differeces in terms of accuracy. A comparison beetwen them can be seen in figure 2

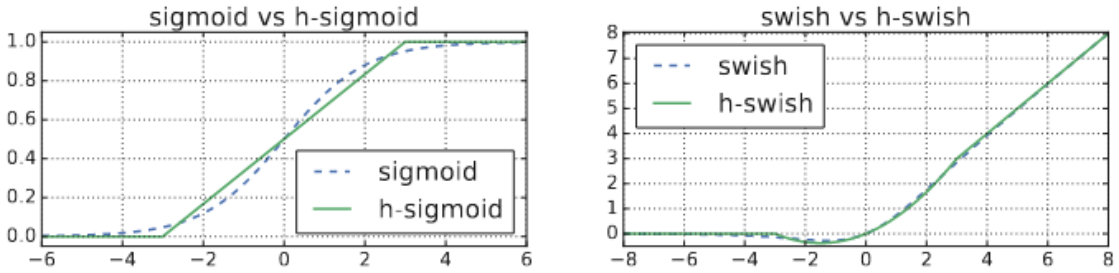

Figure 2: Comparision between Sigmoid and swish function with their hard corrispective

4 Implementation

The MobileNetV3 consists of two different design architectures, one used for high resources and another one for low resources, respectively Large and Small models. In this work both of them are implemented from scratch using pytorch and pytorch-lightning, framework used essentially to simplify multi-gpu training, using Distributed Data Parallel paradigm.
Initially the implementation was done using Jupyter Notebooks, however cause of a bug regarding multi-gpu training, the implementation was completed using classical python script files.
Both proposed model, small and large, were trained using the structure defined by the authors and replicated in the Figure 3.

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

(a) Large model

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

(b) Small model

Figure 3: MobileNetV3 specifications for Large and Small model

These specifications are stored inside two different json files, loaded before training the neural network to create the model architecture on the fly. Every row of the table 3 is associated to an object of the json file, containg:

- the input and output size
- kernel size
- the stride size
- expansion dimension

- if the squeeze and excite must be used
- the type of non-linearity to be used (ReLU or hard-swish)

4.1 Adroid app

After the training all the models are transformed in order to use them in an android workspace. The workflow followed for this operation can be seen in figure 4, which consists in the scripting of the model with the *just in time* built-in pytorch compiler and the transformation of this to a model for pytorch lite.

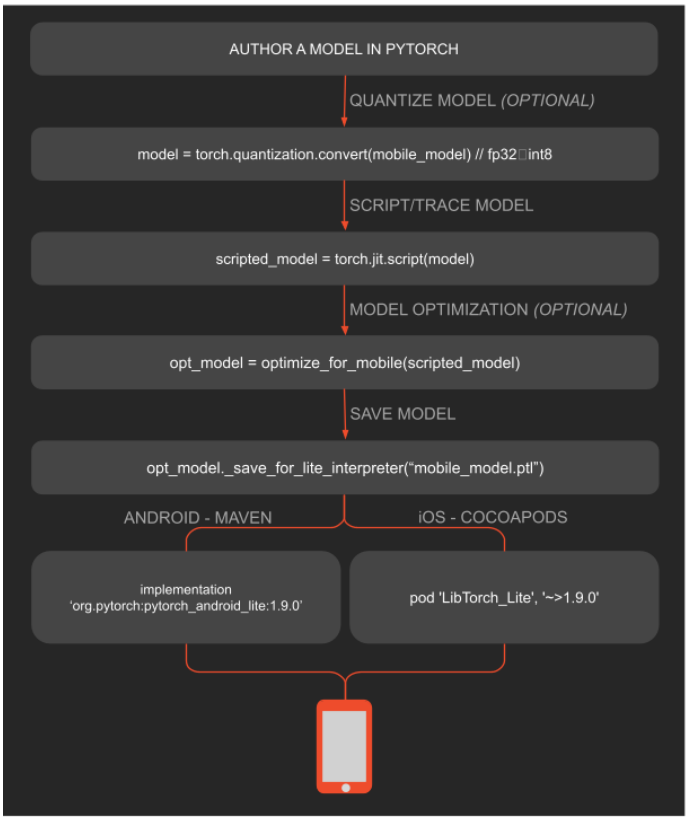

Figure 4: Mobile deployment workflow: https://pytorch.org/mobile/home/

After this workflow as output we have a model that is ready to use inside a Android or iOS app. To make a demo, we created a simple app in Android Studio using Kotlin, in order to test the mobile version of the trained models.
The app is very simple, as shown in the figure 5. In the main view the image to

be predicted is shown, while on the bottom the prediction is set in a light grey box. There are two buttons, always on the bottom, one for Mnist and another one for Cifar10 image prediction. Both of them upload the needed model, put inside the asset folder, while a new image is randomly chosen inside the associated asset directory (`asset/MNIST` or `asset/Cifar10`).

Figure 5: Android demo app

Even if this is only a demo app, it's noticeable that, without using model quantization and suitable optimization, there is a significant drop in model accuracy, underlying moreover the improvements that can be done using model inference on mobile devices.

5 Experiments and results

The implementation of MobileNetV3 in this work is trained and test using two different datasets: `MNIST` and `Cifar10`.
Both of them are trained using normalization of the channels using Gaussian Normalization. Upsampling to a dimension of 224 width and 224 height is also needed for the images, considering the low quality of the networks.
All the datasets are splitted in training and validation set, in a classical way to evaluate the network on never seen images and trying to not occurr in overfitting.

5.1 MNIST

The MNIST dataset consist in a collection of black and white (one channel) images of hand written digit.
The classification of the MNIST dataset is a very simple task for this kind of networks, for this reason only the small version of MobileNetV3 is trained on this dataset.
In this case a Gaussian Normalization with mean and standard deviation in 0.5 is firstly applied, with some data augmentation using the transforms built-in function of the torchvision framework: random crop and random horizontal flip is applied to training dataset.
The network is trained for 50 epochs.
As we can see from Figure 6 the accuracy during training reaches high values after only 10 epochs, while in Figure 7 we can notice a complementar pattern for the training loss.

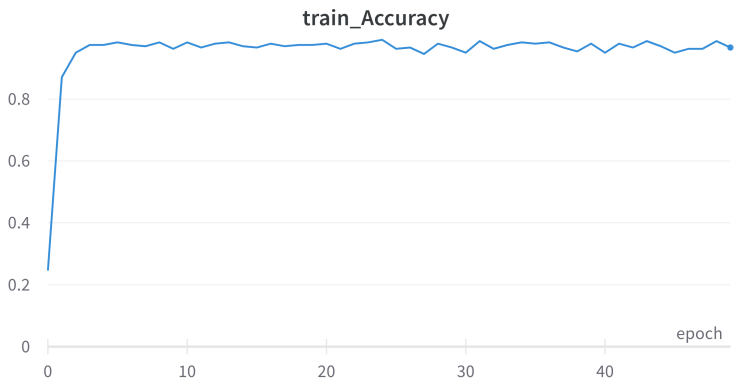

Figure 6: Mnist train accuracy

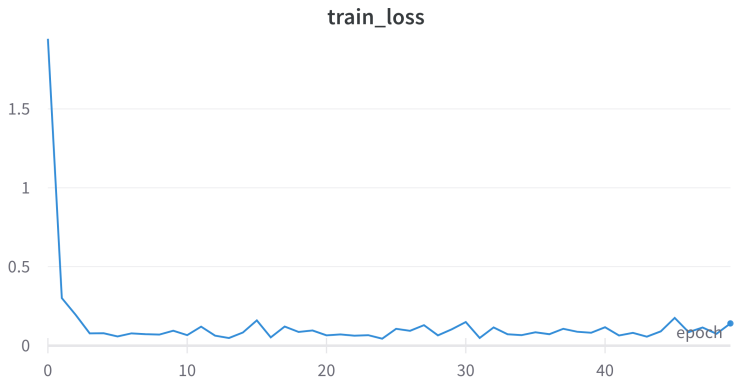

Figure 7: Mnist train loss

In the confusion matrix plotted in Figure 8 we can notice that the network has an high sensitivity with the number 1 while it struggles more with number 5, confused often with numbers 2 and 3.

After training the test dataset is used for evaluation, in this case we notice an overall test accuracy very high, about 0.95%. The associated classification report with all the evaluation metrics is shown in Figure 9.

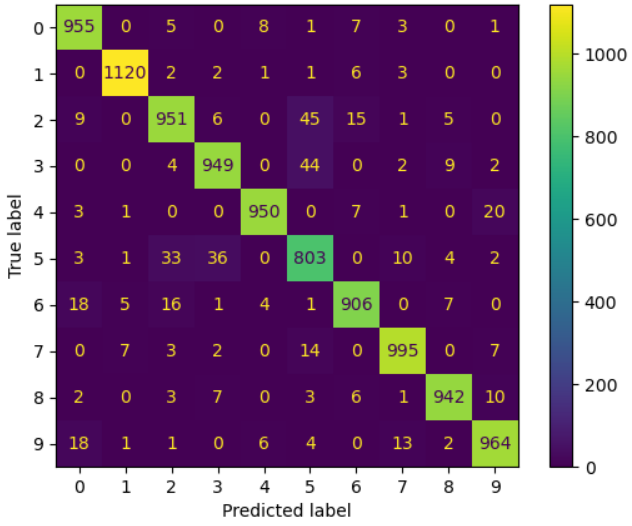

Figure 8: Mnist confusion matrix

```
Accuracy: 0.95

Micro Precision: 0.95
Micro Recall: 0.95
Micro F1-score: 0.95

Macro Precision: 0.95
Macro Recall: 0.95
Macro F1-score: 0.95

Weighted Precision: 0.95
Weighted Recall: 0.95
Weighted F1-score: 0.95

Classification Report
```

	precision	recall	f1-score	support
0 - zero	0.95	0.97	0.96	980
1 - one	0.99	0.99	0.99	1135
2 - two	0.93	0.92	0.93	1032
3 - three	0.95	0.94	0.94	1010
4 - four	0.98	0.97	0.97	982
5 - five	0.88	0.90	0.89	892
6 - six	0.96	0.95	0.95	958
7 - seven	0.97	0.97	0.97	1028
8 - eight	0.97	0.97	0.97	974
9 - nine	0.96	0.96	0.96	1009
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000

Figure 9: Mnist classification report

5.2 Cifar10

The Cifar10 dataset consist in a collection of rgb (three channels) images belonging to 3 different categories.
This classification task is more complex than the MNIST one, for this reason in this work the train is done with both the networks architectures, Small and Large.
Also in this case a Gaussian Normalization is used with mean in $[0.485, 0.456, 0.406]$ and standard deviation in $[0.247, 0.243, 0.261]$ is firstly applied. Like before, some data augmentation is applied: random crop and random horizontal flip is applied to training dataset.
The network is trained for 25 epochs.

Small As we can see from Figure 10 the accuracy during training reaches not so good values, while in Figure 11 we can notice a complementar pattern for the training loss.

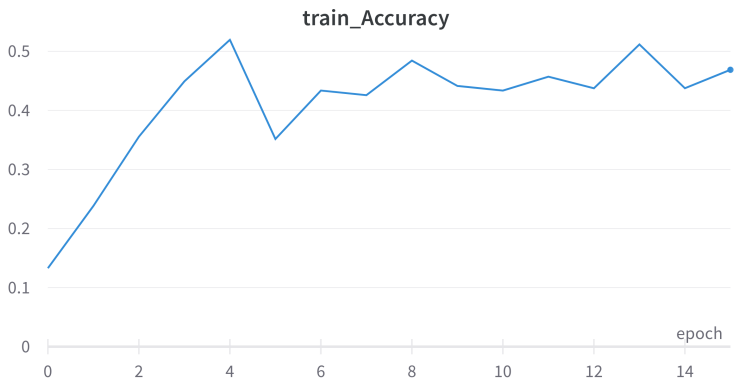

Figure 10: Cifar10 train accuracy using MobileNetV3 small and stride 2

Figure 11: Cifar10 train loss using MobileNetV3 small and stride 2

This low training accuracy and loss is due to the datasets itself, it has very low image quality. An attempt to cut out this problem is done changing the stride of the first two convolutional layers.
In the architecture proposed by the authors, trained on the ImageNet dataset, the stride is set to two, loosing some informations in the convolution operation. For this type of dataset the stride is changed to one resulting in less information loss and better results.
In fact, as we can see from Figure 12 and 13 the loss and accuracy gets better during training with this small change.

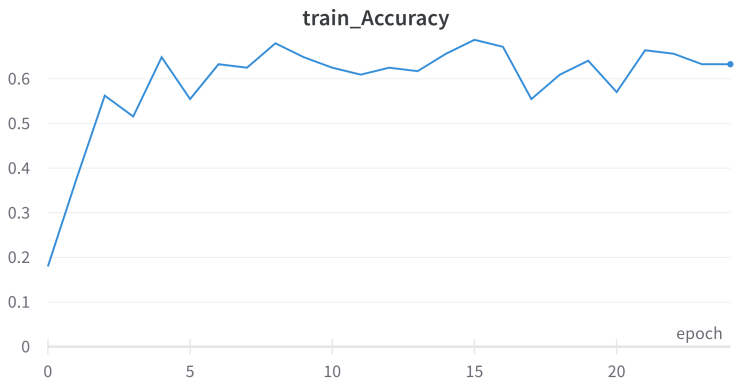

Figure 12: Cifar10 train accuracy using MobileNetV3 small and stride 1

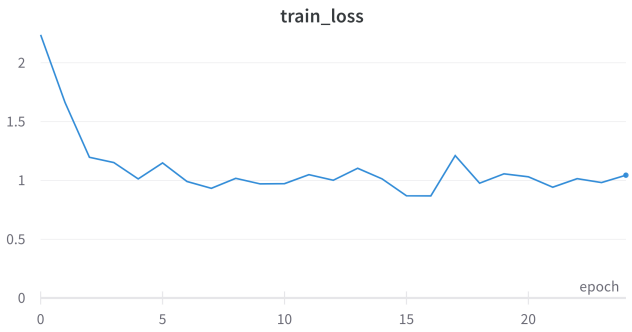

Figure 13: Cifar10 train loss using MobileNetV3 small and stride 1

In the confusion matrix plotted in Figure 14 we can notice that the network has low sensitivity with the class associated to label $2, 3, 4$ associated respectively to cat, deer and dog images. The network, therefore, is often confused in the identification of images belonging to these classes.
As before, after training the test dataset is used for evaluation. In this case we notice an overall test accuracy quite good, about 0.59%. The associated classification report with all the evaluation metrics is shown in Figure 15.

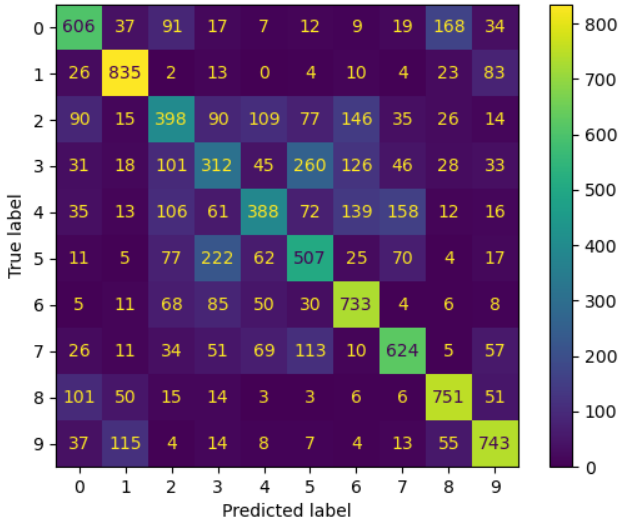

Figure 14: Cifar10 confusion matrix using MobileNetV3 Small and stride 1

```
Accuracy: 0.59

Micro Precision: 0.59
Micro Recall: 0.59
Micro F1-score: 0.59

Macro Precision: 0.58
Macro Recall: 0.59
Macro F1-score: 0.58

Weighted Precision: 0.58
Weighted Recall: 0.59
Weighted F1-score: 0.58

Classification Report
              precision    recall  f1-score   support

    airplane       0.63      0.61      0.62      1000
  automobile       0.75      0.83      0.79      1000
        bird       0.44      0.40      0.42      1000
         cat       0.35      0.31      0.33      1000
        deer       0.52      0.39      0.45      1000
         dog       0.47      0.51      0.49      1000
        frog       0.61      0.73      0.66      1000
       horse       0.64      0.62      0.63      1000
        ship       0.70      0.75      0.72      1000
       truck       0.70      0.74      0.72      1000

    accuracy                           0.59     10000
   macro avg       0.58      0.59      0.58     10000
weighted avg       0.58      0.59      0.58     10000
```

Figure 15: Cifar10 classification report using MobileNetV3 Small and stride 1

Large Similarly to the Small version of MobileNetV3, the dataset is used to train MobileNet Large.
This time, using stride 1 for training, we can notice in Figure 16 an increases to training accuracy with respect to the small counterpart.

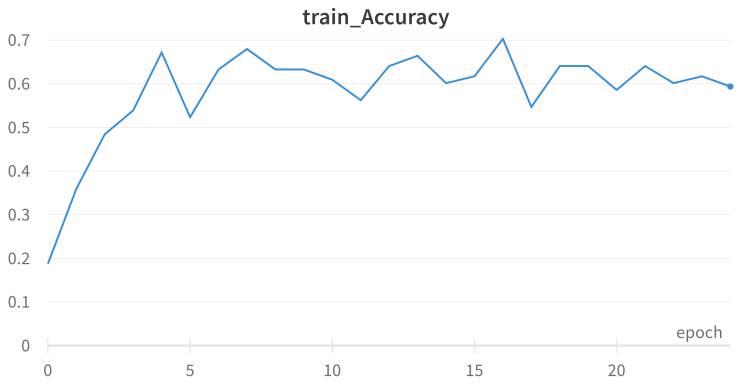

Figure 16: Cifar10 train accuracy using MobileNetV3 Large and stride 1

Figure 17: Cifar10 train loss using MobileNetV3 Large and stride 2

Using the Large architecture on this dataset, we notice a slight increase in performance by the network on the evaluation set, as expected and stated by the authors. In fact, we the overall test accuracy about 0.6%. The associated classification report with all the evaluation metrics is shown in Figure 19.

In the confusion matrix plotted in Figure 18 we can notice that the network has increased its sensitivity within the cat, deer and dog images, continuing to struggle more for the deer class.

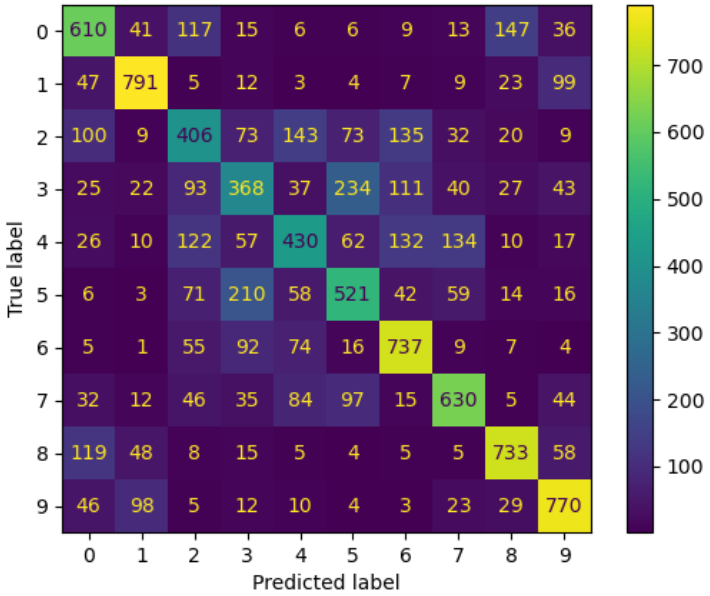

Figure 18: Cifar10 confusion matrix using MobileNetV3 Large and stride 1

```
Accuracy: 0.60

Micro Precision: 0.60
Micro Recall: 0.60
Micro F1-score: 0.60

Macro Precision: 0.59
Macro Recall: 0.60
Macro F1-score: 0.60

Weighted Precision: 0.59
Weighted Recall: 0.60
Weighted F1-score: 0.60

Classification Report
```

	precision	recall	f1-score	support
airplane	0.60	0.61	0.61	1000
automobile	0.76	0.79	0.78	1000
bird	0.44	0.41	0.42	1000
cat	0.41	0.37	0.39	1000
deer	0.51	0.43	0.46	1000
dog	0.51	0.52	0.52	1000
frog	0.62	0.74	0.67	1000
horse	0.66	0.63	0.64	1000
ship	0.72	0.73	0.73	1000
truck	0.70	0.77	0.73	1000
accuracy			0.60	10000
macro avg	0.59	0.60	0.60	10000
weighted avg	0.59	0.60	0.60	10000

Figure 19: Cifar10 classification report using MobileNetV3 Large and stride 1

References

[1] Andrew Howard et al. "Searching for MobileNetV3". In: (2019). arXiv: 1905.02244 [cs.CV].

[2] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: (2019). arXiv: 1801.04381 [cs.CV].

[3] Mingxing Tan et al. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: (2019). arXiv: 1807.11626 [cs.CV].

[4] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[5] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[6] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size". In: (2016). arXiv: 1602.07360 [cs.CV].

[7] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: (2017). arXiv: 1704.04861 [cs.CV].

[8] Xiangyu Zhang et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: (2017). arXiv: 1707.01083 [cs.CV].

[9] Gao Huang et al. "CondenseNet: An Efficient DenseNet using Learned Group Convolutions". In: (2018). arXiv: 1711.09224 [cs.CV].

[10] Zhaoyi Yan et al. "Shift-Net: Image Inpainting via Deep Feature Rearrangement". In: (2018). arXiv: 1801.09392 [cs.CV].