Disclaimer: This paper is a work of fiction written from the perspective of a 2020 researcher traveling back in time to late 2012 to share some 2020 network design, training and implementation ideas; references to credit the actual inventors of the various ideas is provided at the end

MobileNetV2: The Same but Better

Nancy Dominguez
nsd093020@utdallas.edu
Dec 31, 2012

Abstract

Existing current methods are limited by image resolution, computational complexity, or by large memory usage. Most will sacrifice one for the other. However, MobileNetV2 is a network that strikes an optimal balance between accuracy and memory/speed.

Even though MobileNetV2 is a deep convolutional network designed to process high resolution images, it controls memory usage and computations by replacing the standard convolution with a depthwise convolution and pointwise convolution. It maintains the accuracy by including an expansion factor and adding a residual connection. Which leads to many levels composed of the network's basic building block- the bottleneck depthwise separable convolution with inverted residuals.

The expansion factor is a hyperparameter that will need to be adjusted based on the size of the dataset. A way to speed up training time is to use a linear warmup followed by cosine decay, and a way to improve model accuracy during training is to normalize and standardize batches. In this paper, we train two versions of MobileNetV2- the standard version for higher resolution images and a reduced version for lower resolution images.

Out of AlexNet, VGG-16, and ResNet-18, MobileNet V2 requires the least amount of memory and computation time while maintaining a similar, if not better accuracy.

1 Introduction

Humans classify images every day, and now computers can do the same! Image classification started out as classifying black and white low resolution images of written numbers and now it has the practical application of ATMs accepting checks instead of requiring a bank teller. Now, there are networks that can classify high resolution images such as people and cars and could be used in the self-driving cars of the future.

Current object classification methods are very accurate, however they have grown in size and computational complexity, and can be slow to carry out the image processing. As such, these networks would not be apt for devices that have limited memory and processing units such as mobile phones or tasks that require immediate image processing such as a self-driving car. For example, just to sit on disk, without running, VGG16 requires a disk size of 553MB which is due to the 138M parameters required for the network whereas MobileNetV2 requires a disk size of 17.2MB and has 4.27M parameters. Once training occurs, these memory size requirements will increase. [3]

In Section 2, we will go over the implementation for object classification of current existing methods and their limitations that MobileNetV2 aims to fix. In Section 3, we will describe in detail how the design of MobileNetV2 works to combat the limitations and show that the model can be parameterized to be even more efficient in regard to particular types of datasets. In Section 4, we go over the MobileNetV2 training methods that work the best with datasets of lower and higher resolutions. Finally, in Section 5, we go over the memory and compute time of the implementation of MobileNetV2.

2 Related Work

2.1 Design

The first network to perform object classification was LeNet which was developed in 1998. It was used to classify black and white low resolution (32x32) objects. LeNet has 60k parameters and is composed of 5 layers- 3 convolution layers, each followed by average pooling, 2 two fully connected layers. The most expensive part

of the network are these fully connected layers and so it is not recommended to use datasets like ImageNet that has many classes.

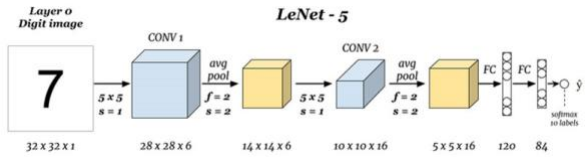

Figure 1: LeNet architecture [4]

AlexNet was later developed in 2012. It was based on the LeNet architecture but could work with datasets that have high resolution images (224x224) and multiple classes. AlexNet is much bigger with 60M parameters and is composed of 7 layers- 4 unique convolution layers, along with some max pooling layers, and 3 fully connected layers. Even though AlexNet is much bigger in size than LeNet, its activation function is ReLU instead of Sigmoid or Tanh functions which speed up the network by more than 5 times faster with same accuracy.

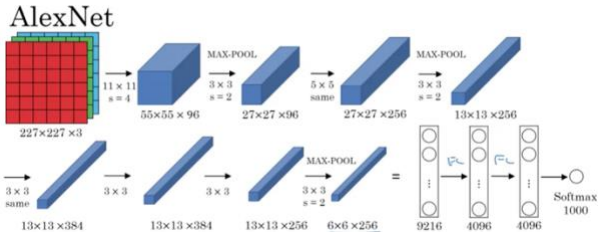

Figure 2: AlexNet architecture [5]

As we can see, the trend is that the network increases in size, complexity, and computation time when our dataset grows more complex with many classes and higher resolution color images. This is in part because we will inevitably require more parameters for larger inputs. Standard convolution is essentially a depthwise convolution and a pointwise convolution done all in one step. The problem with this is that the number of parameters, which impacts memory size, can blow up.

In this paper, we aim to combat these limitations by creating a network that has a good balance between accuracy and memory/speed.

2.2 Training

Mini batch stochastic gradient descent is a mix of stochastic gradient descent and batch gradient descent. Stochastic gradient descent updates the model for every example in the dataset which tends to lead to faster learning. Whereas batch gradient updates the model at the end of an epoch when all examples have passed through the network making batch gradient much more computationally efficient than stochastic gradient descent. However, this comes at a cost of higher memory usage and lower model update accuracy.

Mini batch gradient descent tries to find a balance between the two. When using mini batch gradient descent, the training data is split into batches and so the model updates after every batch. Controlling the number of model updates with batches will save computation time, yet having more model updates will avoid getting stuck at local minima like batch gradient descent tends to. However, a new hyperparameter of batch size will add training complexity to our model as it will have to be yet another hyperparameter that will need to be fine tuned.

2.3 Implementation

Intel x86 CPUs and GPUs have greatly increased the memory that networks can work with accomadating the memory requirement of LeNet but requiring a slight rework of AlexNet. CPUs have a larger memory than GPUs but don't have a lower processing speed while GPUs have a smaller memory possibly requiring data to be transferred to and from the CPU. However, this can be slow. We will see that MobileNetV2 is so small in memory that this will not pose a problem.

3 Design

A standard convolution can be slow to perform a process, but if we instead break it into two simpler processes via depthwise separable convolution, then image processing can use less parameters and be a lot faster. In standard convolution, the application of filters throughout and across all input channels are done in one single step. Depthwise Separable Convolution on the other hand, breaks this down into two steps: depthwise convolution, which applies convolution to a single channel at a time, and pointwise convolution, which linearly combines all the channels of the previous resulting depthwise convolution.[6] Since depthwise separable convolution is adding the computational complexity of two processes instead of multiplying everything all at once, depthwise separable convolution

will use 8 to 9 times less computation than standard convolution. I leave the details of computational efficiency calculations for the reader to review in MobileNet V1 publication.[2]

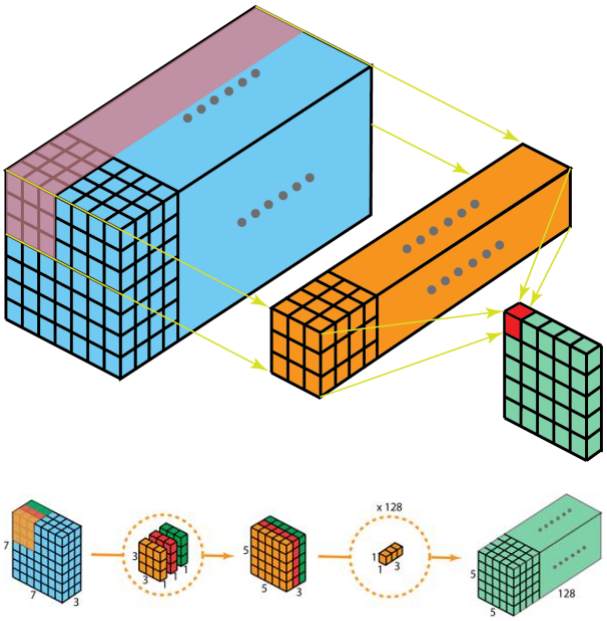

Figure 3: Top is standard convolution where the application of filters throughout and across all input channels are done in one single step, and bottom is depthwise separable convolution where a convolution is applied at each channel first (depthwise) and then another convolution is applied across all channels at each point (pointwise). [7]

Convolutional Networks will most likely require the use nonlinear activation functions such as, ReLu, as they are very effective at extracting features from data. However whenever nonlinearities are used on a channel, some information is lost. Therefore, by having many channels, this loss of information can be mitigated if there are other existing channels that might still be preserving the information. However, one of the goals of this network is to reduce the number of parameters the network requires, i.e. the network size, so we want to pass low dimensional tensors from building block to building block. In order to get the best of both worlds, we will implement linear bottlenecks where the number of channels of the input and output are lower, and the number of channels within the building block are expanded by a factor of t. Nonlinearities will be applied within the building block where we have more channels.

Furthermore, we can prevent the loss of information due to these nonlinearities by including an inverted residual connection that adds the information found at the bottlenecks of each building block. Since this residual connection helps with information loss, it in turn helps with having a smoother and more accurate gradient descent in the backward pass for better generalization.[8]

The basic building block of the network is a bottleneck depthwise separable convolution with inverted residuals. Here we use three distinct instances of that block. 1) Special Identity block – used when there are different number of channels at both ends of the block 2) Identity block – used when both ends of the block have the same number of channels 3) Downsample block – used when we wish to downsample the input size.

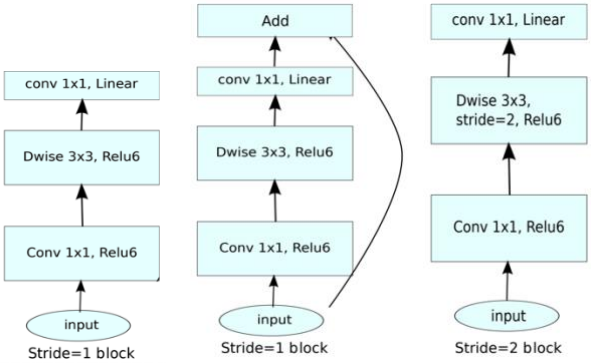

Figure 4: Special Identity block, Identity block, Downsample block in respective order.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: Input to output transformation for each sub-block with stride s, and expansion factor t. [1]

The network is comprised of a tail (encoder), a body, and a head (decoder). The tail is one standard convolution, the body is composed of 7 levels with a total of 17 blocks, and the head is composed of a pointwise convolution followed by a global average pooling.

Standard MobileNetV2						
Level	Input	Operator	*t*	*c*	*n*	*s*
tail	224^2x3	conv2D	-	32	1	2
1	112^2x32	special	1	16	1	1
2	112^2x16	downsample	6	24	1	2
	56^2x24	identity	6	24	1	1
3	56^2x24	downsample	6	32	1	2
	28^2x32	identity	6	32	3	1
4	28^2x32	downsample	6	64	1	2
	14^2x64	identity	6	64	3	1
5	14^2x64	special	6	96	1	1
	14^2x96	identity	6	96	2	1
6	14^2x96	downsample	6	160	1	2
	7^2x160	identity	6	160	2	1
7	7^2x160	special	6	320	1	1
head	7^2x320	conv2D 1x1	-	1280	1	1
	7^2x1280	global avg pool	-	256	1	-
	1^2x256	softmax	-	k	1	-

Number of Parameters: 4,278,048

*Table 2: Sequence of bottleneck depthwise separable convolution blocks (building blocks) that make up MobileNetV2. The expansion factor **t** is the factor by which the channels will expand within the block. The number of output channels at each block is given by **c**, the stride for each block is given by **s**, and the number of repetitions per block is given by **n**. [1]*

To create a reduced version of MobileNetV2 that will be more accurate and efficient with datasets of lower resolution, remove the levels that came before the resolution of the dataset images and downsample until the channel is an 8x8. The expansion factor might need to be tuned down if the dataset is small. The number of repetitions at the identity block should also decrease as the level increases.

Reduced MobileNetV2						
Level	Input	Operator	*t*	*c*	*n*	*s*
tail	32^2x3	conv2D	-	32	1	2
1	16^2x32	special	6	64	1	1
	16^2x64	identity	6	64	3	1
2	16^2x64	downsample	6	96	1	2
	8^2x96	identity	6	96	2	1
3	8^2x96	special	6	160	1	1
	8^2x160	identity	6	160	2	1
head	8^2x160	conv2D 1x1	-	256	1	1
	8^2x256	global avg pool	-	256	1	-
	1^2x256	softmax	-	k	1	-

Number of parameters: 709,120

Table 3: A smaller possible variation of MobileNetV2 that can be used for smaller sized datasets. Uses less memory and is faster than the standard MobileNetV2 while maintaining accuracy.

4 Training

State why to each statement:
In this experiment, we will create a model for image classification using a CIFAR10 dataset. The training models accepts batches of 32 images with each image resized to 224x224 or 32x32 depending on the model version. Image batching allows our model to process more than one image at a time so if we have adequate computational resources on our machine, each epoch during training will finish faster. [3]

After every block, we use batch normalization with momentum of 0.99 and epsilon of 0.001 Batch normalization is another technique that allows for faster training by stabilizing the learning process and dramatically reducing the number of training epochs required to converge. [9]

A simple image augmentation is used to randomly flip images and to standardize each image using CIFAR10 image dataset mean and standard deviation. Data augmentation is a good strategy to use to add diversity to a dataset so that our model can generalize better at test time, and it's especially effective when the dataset is small. [10]

The network uses a max learning rate of .001, learning scale of .01, and initial learning rate of max*scale. At first, the learning rate scales linearly for the first 5 epochs followed by cosine decay for the rest. This is another good method make our training speed faster because we can quickly approach the minimum with a learning rate that grows linearly and then refine our learning rate by decaying it slowly as we get closer to the minimum. [11]

Both versions were first trained with the CIFAR10 dataset and running 60 epochs, but according to Figure 5, accuracy and cross entropy tapered off around 20-30 epochs. Therefore, in order to prevent overfitting, it would be advisable to limit epochs to no more than 30. CIFAR10 has 50,000 train and 10,000 validation examples.

In addition, both models were also trained using a subset of ImageNet called Imagenette to test how the models would respond to higher resolution images. This dataset is smaller than CIFAR10 with 9,469 train and 3,925 validation examples. [12]

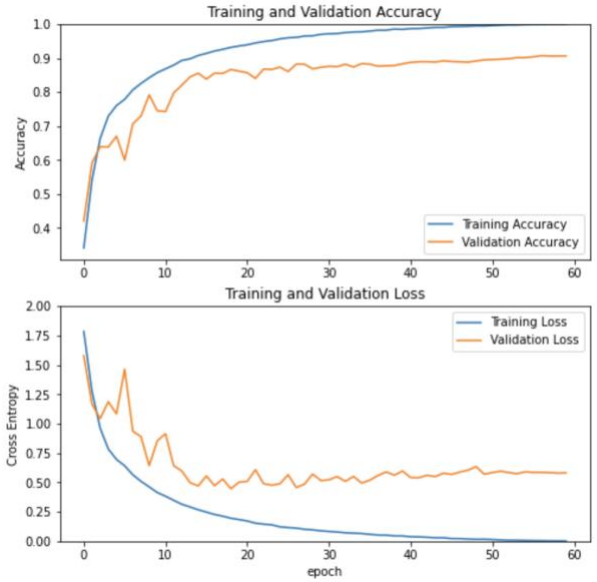

Figure 5: Training curves using CIFAR10 during 60 epochs for Reduced MobileNetV2

Test loss:	0.5812219977378845
Test accuracy:	0.9063000082969666

Table 4: Test Results for Reduced MobileNetV2

Figure 6: Training curves using CIFAR10 during 30 epochs for Reduced MobileNetV2

Test loss:	0.4659130871295929
Test accuracy:	0.9043999910354614

Table 5: Test Results using CIFAR10 Reduced MobileNetV2 using

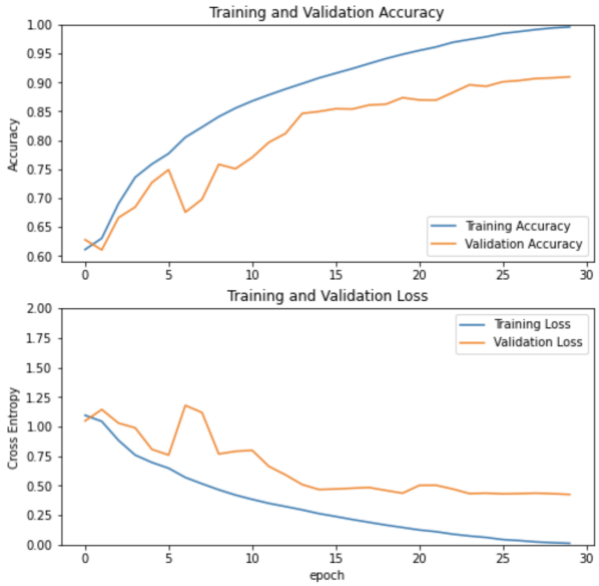

Figure 7: Training curves using CIFAR10 during 30 epochs for Standard MobileNetV2

Test loss:	0.4254923462867737
Test accuracy:	0.9097999930381775

Table 6: Test Results using CIFAR10 for Standard MobileNetV2

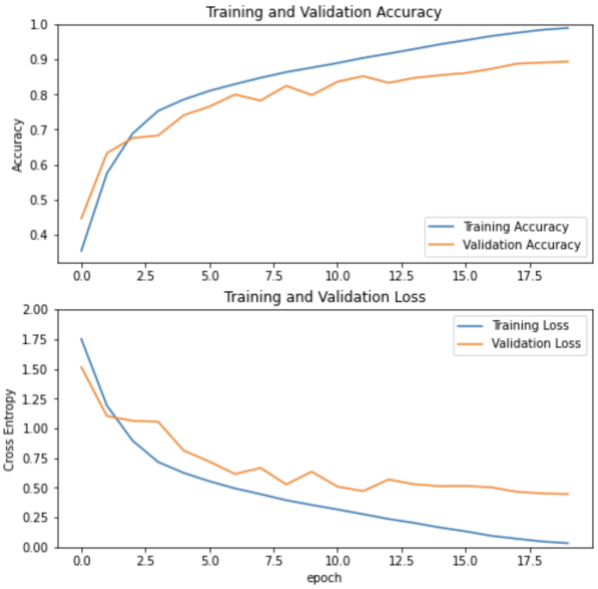

Figure 8: Training curves using imagenette during 20 epochs for Reduced MobileNetV2

Test loss:	0.4467226564884186
Test accuracy:	0.8935999870300293

Table 7: Test Results using imagenette during 20 epochs for Reduced MobileNetV2

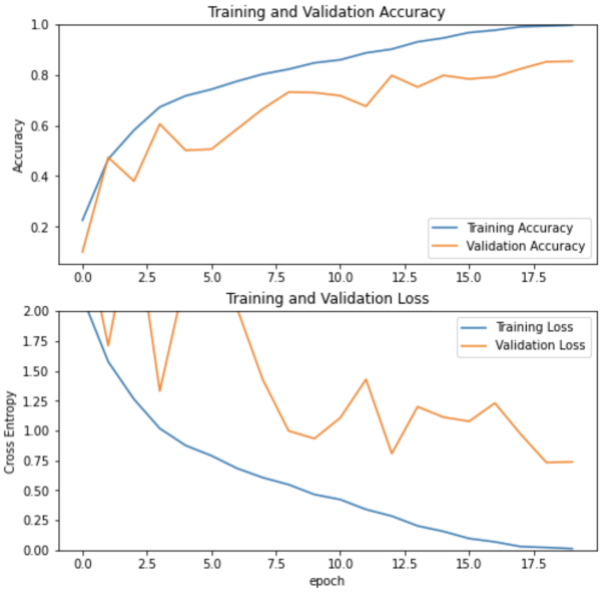

Figure 9: Training curves using imagenette during 20 epochs for Standard MobileNetV2

Test loss:	0.7345989942550659
Test accuracy:	0.8539999723434448

Table 8: Test Results using imagenette during 20 epochs for Standard MobileNetV2

As seen in Figure 9, the model is overfitting the data. It is possible that the standard version of MobileNetV2 might be too complex of a model for a small dataset like imagenette. A recommendation to fix this would be to lower the expansion factor from 6 to 4. The results of this action are seen in Figure 10.

Figure 10: Training curves using imagenette during 20 epochs and t=4 for Standard MobileNetV2

Test loss:	0.3790670931339264
Test accuracy:	0.9041000008583069

Table 9: Test Results using imagenette during 20 epochs and t=4 for Standard MobileNetV2

5 Implementation

This paper offers a standard version and a reduced version of MobileNetV2. The standard version is meant for large datasets such as ImageNet and the reduced version is meant for smaller datasets such as CIFAR10. The reduced version will require much less memory space and will run much faster. That being said, standard MobileNetV2 is much smaller in size and faster than the average current method while maintaining comparable accuracy.

ImageNet Object Classification Accuracy	
Network	Accuracy %
MobileNetV2	71.81
AlexNet	57.2
VGG-16	71.5
ResNet-18	69.52

Table 10: Accuracies using Imagenet dataset [13]

5.1 Memory

The minimum amount of memory required to run the standard version is 47MB while for the reduced version, the minimum amount is 4.6MB. This is calculated by adding the memory to hold all parameters and their gradient plus the memory to hold the largest feature map, filter and output size. The amount of memory per layer can be found in Table A1 in the Appendix. Since MobileNetV2 is so small in size, there are no present day CPU or GPU limitations which have an input memory of 3GB.

$$(Parameters + Parameter\ Gradients + Feature\ map + Filter\ map\ size) * 4 = Memory$$

$$(709120 + 709120 + 196608 + 2983984) * 4 = 18395328$$

$$(4278048 + 4278048 + 196608 + 2983984) * 4 = 46946752$$

As seen in Figure 11, MobileNetV2 memory usage is comparable to AlexNet but is able to handle large datasets like VGG16 is able to.

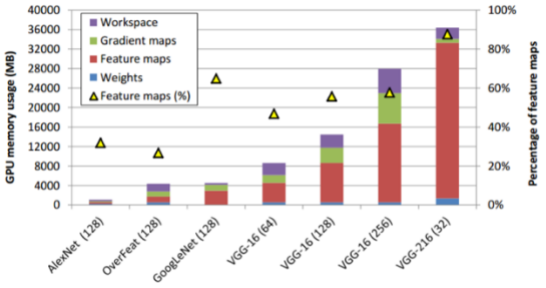

Figure 11: Amount of memory required to run current methods for object classification. MobileNetV2 standard version is 47MB while for the reduced version, the minimum amount is 4.6MB. [14]

As shown in Figure 12, through accuracy density, we can see that MobileNetV2 strikes the best balance between memory and accuracy compared to AlexNet, VGG-16, and ResNet-18.

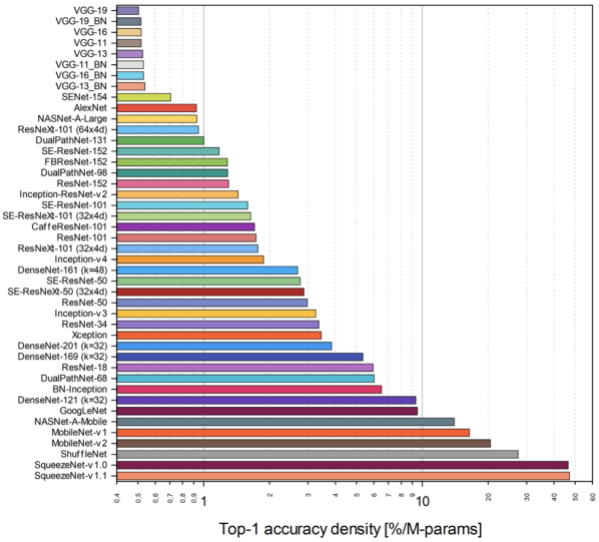

Figure 12: Accuracy density comparisons [13]

5.2 Compute Time

In each block, due to its bottleneck nature, the number of MACs increases and then decreases as image data flows through it. Within each block, the max number of MACs can be found at the 3x3 depth wise convolution that happens right after the 1x1 expansion layer. MACs tend to increase as data moves through each level in the network. This is because even though each channel is downsampled as we go through the network, the number of output channels at each layer keeps increasing (refer to Table 1 and Table 2). In both the standard and reduced version, the maximum number of MACs in the network occur at the last level (found in Table A2). Therefore, the number of repetitions should be limited at this block.

Network MACs	
Network	MACs (M)
MobileNetV2	531
AlexNet	720
VGG-16	15300
ResNet-18	1725

Table 11: MAC comparisons [13]

MACs are directly proportional to computation time and this can be seen empirically by the time required for a single forward pass. A forward pass during testing through the standard version averages about 467s while the reduced version averages about 4s. In addition, one epoch during training in the standard version averages at 467s and the reduced version averages 51s.

As shown in Figure 13, we can see that MobileNetV2 also strikes the best balance between speed and accuracy compared to AlexNet, VGG-16, and ResNet-18.

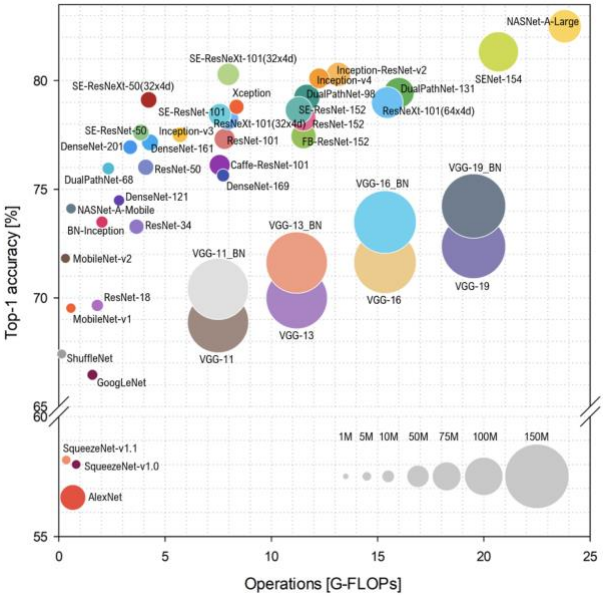

Figure 13: Accuracy vs Number of Operations [13]

6 Conclusion

Existing current methods are limited by image resolution, computational complexity due to standard convolutions, or by large memory usage due to creating large deep networks with millions of parameters to tackle dataset of higher resolutions. MobileNetV2 does away with these limitations by creating a network that has a good balance between accuracy and memory/speed.

MobileNet is a deep convolutional network to process high resolution images, but instead of using standard convolutions that blow up the number of parameters, it instead splits a standard convolution into a depthwise

convolution and pointwise convolution to limit the number of parameters required to generalize the data. We can increase accuracy by including an expansion factor and adding a residual. The expansion factor creates more channels to prevent information loss and the residual connection allows for a smoother gradient descent. Therefore, MobileNetV2 is composed of multiple layers of our basic building block, the bottleneck depthwise separable convolution with inverted residuals.

Fast effective methods that we used to train MobileNetV2 are batch normalization after every block and a learning rate with a linear warmup followed by cosine decay. In addition, an important image preprocessing step to follow is batch standardizing. If the dataset is small, consider image augmentation and keep in mind that the expansion factor hyperparameter might need to be tuned down.

Out of networks mentioned in this paper that can process large datasets with high resolution images (AlexNet, VGG-16, and ResNet-18), MobileNet V2 requires the least amount of memory and computation time while maintaining a similar accuracy, if not better accuracy.

References

Note: As stated above, this paper is a work of fiction. The following are the actual inventors of the ideas described in this paper.

[1] M. Sandler, et. al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," arXiv:1801.04381v4, 2019.

[2] A. Howard, et. al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861v1, 2017.

[3] C. Mandy, et. al., "MobileNet Image Classification With Keras," 2018.

[4] E. Amor, et. al., "4 CNN NETWORKS EVERY MACHINE LEARNING ENGINEER SHOULD KNOW," 2020.

[5] X. Prabhu, et. al., "CNN Architectures — LeNet, AlexNet, VGG, GoogLeNet and ResNet," 2018.

[6] C. Emporium, et. al., "Depthwise Separable Convolution - A FASTER CONVOLUTION!," 2018.

[7] K. Bai, et. al., "A Comprehensive Introduction to Different Types of Convolutions in Deep Learning," 2019.

[8] L. Gonzales, et. al., "A Look at MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019.

[9] J. Brownlee, et. al., "A Gentle Introduction to Batch Normalization for Deep Neural Networks," 2019.

[10] D. Ho, et. al., "1000x Faster Data Augmentation," 2019.

[11] C. Wang, et. al., "A Newbie's Guide to Stochastic Gradient Descent With Restarts," 2018.

[12] imagenette

[13] S. Bianco, et. al., "Benchmark Analysis of Representative
Deep Neural Network Architectures," arXiv:1810.00736v2, 2018.

[14] H. Zhu, et. al., "How to Train a Very Large and Deep Model on One GPU?" 2019.

[15] Reduced MobileNetV2:
https://colab.research.google.com/drive/1iSYC47xVF8seO3wodrHTSxCRXHCsYXTU

[16] Standard MobileNetV2:
https://colab.research.google.com/drive/1AfoZ94womOzAs5uP2oAwo6cHl98z0REA

…

Appendix

Table A1: Feature Map Size and Memory per layer. Maximum feature map size is in black.

		Feature Map Size $=L_r \times L_c \times N_i$	Feature Map Memory $=(N_i + N_o) \times L_r \times L_c$
Level 5:	conv(1x1)	8x8x160	(160+256)*8*8 = 26,624
		8x8x960	(960+160)*8*8 = 71,680
		8x8x960 = 61,440	(960+960)*8*8 = 122,880
Level 4:	identity	8x8x160 = 10,240	(160+960)*8*8 = 71,680
		8x8x576	(576+160)*8*8 = 47,104
		8x8x576	(576+576)*8*8 = 73,728
Level 3-4:	special_id_block	8x8x96	(96+576)*8*8 = 43,008
		8x8x576	(576+96)*8*8 = 43,008
		8x8x576 = 36,864	(576+576) *8*8 = 73,728
Level 3:	identity	8x8x96 = 6,144	(96+576)*8*8 = 43,008
		16x16x384	(384+96)*16*16 = 122,880
		16x16x384	(384+384)*16*16 = 196,608
Level 2-3:	conv_block	16x16x64	(64+384)*16*16 = 114,688
		16x16x384	(384+64)*16*16 = 114,688
		16x16x384 = 98,304	(384+384)*16*16 = 196,608
Level 2:	identity	16x16x64 = 16,384	(64+384)*16*16 = 114,688
		16x16x192	(192+64)*16*16 = 65,536
		16x16x192 = 49,152	(192+192)*16*16 = 98,304
Level 1-2:	special_id_block	16x16x32 = 8,192	(32+192)*16*16 = 57,344
Level 0-1:	conv(3x3/2s)	32x32x3 = 3,072	(3+32)*32*32 = 35,840

Table A2: Filter Coefficient Size and Memory per layer. Largest filter memory is in green and smallest filter memory is in orange.

		Filter Coefficient Size & Filter Coefficient Memory $=F_r \times F_c \times N_i \times N_o$	
Level 5:	conv(1x1)	1x1x160x256 =	40,960
		1x1x960x160 =	153,600
		3x3x960x960 =	8294,400
Level 4:	identity	1x1x160x960 =	153,600
		1x1x576x160 =	92,160
		3x3x576x576 =	2,985,984
Level 3-4:	special_id_block	1x1x96x576 =	55,296
		1x1x576x96 =	55,296
		3x3x576x576 =	2,985,984
Level 3:	identity	1x1x96x576 =	55,296
		1x1x384x96 =	36,864
		3x3x384x384 =	1,327,104
Level 2-3:	conv_block	1x1x64x384 =	24,576
		1x1x384x64 =	24,576
		3x3x384x384 =	1,327,104
Level 2:	identity	1x1x64x384 =	24,576
		1x1x192x64 =	12,288
		3x3x192x192 =	331,776
Level 1-2:	special_id_block	1x1x32x192 =	6,144
Level 0-1:	conv(3x3/2s)	3x3x3x32 =	864

Table A2: MACs per layer. Max number of MACs is in green and minimum number of MACs is in orange.

		MACs = $F_r \times F_c \times N_i \times N_o \times L_r \times L_c$	
Level 5:	conv(1x1)	1x1x160x256 x8x8 =	2,621,440
		1x1x960x160 x8x8=	9,830,400
		3x3x960x960 x8x8=	530,841,600
Level 4:	identity	1x1x160x960 x8x8 =	9,830,400
		1x1x576x160 x8x8 =	5,898,240
		3x3x576x576 x8x8 =	191,102,976
Level 3-4:	special_id_block	1x1x96x576 x8x8 =	3,538,944
		1x1x576x96 x8x8 =	3,538,944
		3x3x576x576 x8x8 =	191,102,976
Level 3:	identity	1x1x96x576 x8x8 =	3,538,944
		1x1x384x96 x16x16 =	9,437,184
		3x3x384x384 x16x16 =	339,738,624
Level 2-3:	conv_block	1x1x64x384 x16x16 =	6,291,456
		1x1x384x64 x16x16 =	6,291,456
		3x3x384x384 x16x16 =	339,738,624
Level 2:	identity	1x1x64x384 x16x16 =	6,291,456
		1x1x192x64 x16x16 =	3,145,728
		3x3x192x192 x16x16 =	84,934,656
Level 1-2:	special_id_block	1x1x32x192 x16x16 =	1,572,864
Level 0-1:	conv(3x3/2s)	3x3x3x32 x32x32 =	884,736