Deep Learning's Potential — Drawing the Line Between Concepts and Tangible Objects

Pratik Bhusal
pratik.bhusal@utdallas.edu
July 27, 2021

Max Xie
max.xie@utdallas.edu
July 27, 2021

***Abstract*—Hello World**

I. Introduction

II. Related Work

A. *ResNet V2 (ResNet 50)*

B. *Training*

III. Design

A. *Depthwise Separable Convolutions*

To extarct strong features in an image classification task, 2-D convolution layers are used in the encoder of the model. As the data goes through the network, the weak features found in the original image become more pronounced as the receptive field size and thus the number of feature maps increases. Figure 1 depicts how a standard convolution takes place in a 3×3×3 region gets convolved with a 3×3×3 to generate a 1×1 receptive field of the output.

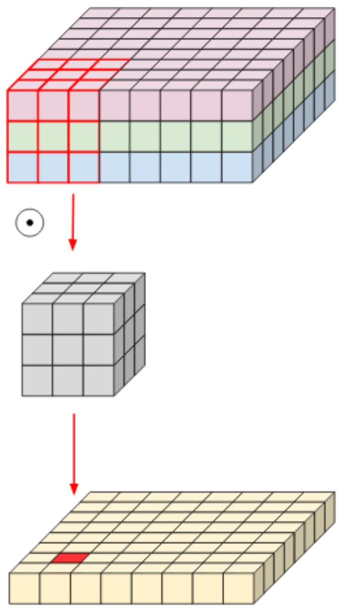

Fig. 1. Standard 2D Convolution [1]

However, because we are mixing across *and* within channels, there is a high computational complexity for each convolution. To remedy this issue, we split the mixings into 2 insolated steps. First, depth-wise convolution is applied.

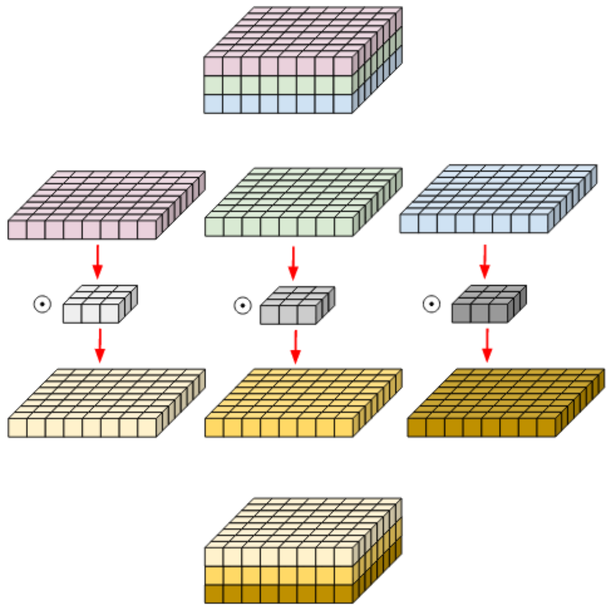

Fig. 2. Depth-Wise Convolution [1]

Each channel is convolved separately in Figure 2. Then, they are stacked thereafter to complete the intermediate step of depthwise separable convolutions. This intermediate phase is where "mixing within channels" occurs. Next, pointwise convolution is applied.

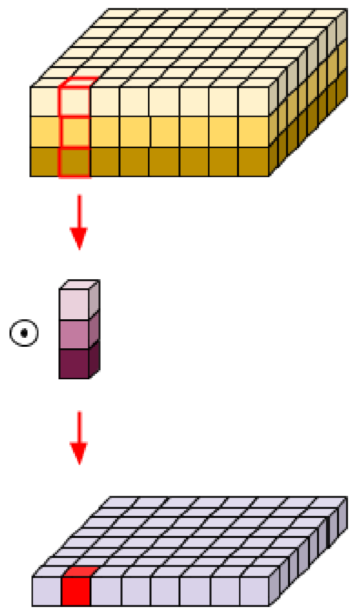

Fig. 3. Depth-Wise Convolution [1]

Figure 3 depicts the final phase of “mixing across channels”. A point wise convolution is applied to each $1\times1\times3$ region to generate the same result as a standard 2D convolution layer.

By using this design for convolutions, it greatly reduces the number of trainable parameters [2].

$$\frac{1}{N} + \frac{1}{D_k^2} \tag{1}$$

Equation 1 empherically defines the computational cost reduction of depthwise separable convolutions. Here, N represents the number of output feature maps and D_K^2 represents the demensions of the filter.

B. Inverted Residual Bottleneck Blocks

This block structure uses the depthwise separable convolutions defined in the previous scection III-A. These blocks provide a middle ground between the complexity of deeper networks while also keeping some of the simplicity and computational ease of shallower network designs. Most importantly, they help overcome the limitations of mini-batch SGD implementations mention in Section II-B.Figure 4 provides a visual example of the residual bootleneck block.

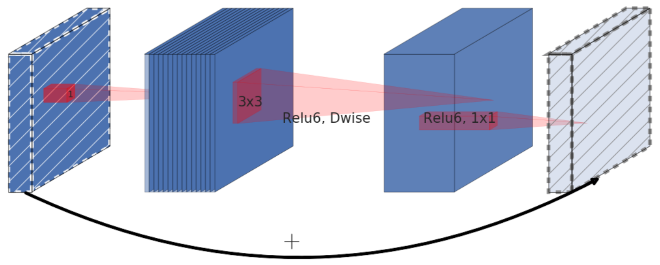

Fig. 4. Inverted Residual BottleNeck Block [3, p. 3]

The “inversion” comes from the fact that we create an expansion bottleneck. By increasing the number of feature maps by an expansion factor t before we perform depth wise convolutions. Now, only a select few channels pass into a depthwise separable convolution block.

Finally, a residual connection is added together to the output of the depthwise separable convolution block to get the final output of the inverted residual bottleneck block. By doing so it helps with the propagation of the output of the depthwise separable convolution block and thus alleviates the issue of vanishing and exploding gradients [4]. With this implementation, it keeps the number of channels low while also maintaining a high accuracy [3].

C. Architecture Overview

The following subsections go into deatail of the Encoder-Decoder style architecture of MobileNet. This is the reference point for the CIFAR-10 implementation itself will be based off of and is heavily inspired by a paper of a similar design [3]. This particular model has 1,184,746 trainable parameters. Some noteworthy points are that the ReLU-6 activation function is used instead of standard ReLU to allow learning sparse features. ReLU-6 can be defined as $\min(6, ReLU(\mathbf{x}))$.

D. Encoder — Tail Architecture

	Input Dimensions		Expansion			Output
Layer	Feature Map	Size	Factor	Repeats	Stride	Channels
Image	3	28×28	—	—	—	—
Convolution	3	28×28	—	1	1	32

TABLE I
MOBILENET V2 TAIL FOR CIFAR 10 DATASET.

The single 2D convolution layer exists to match the number of output feature maps with the input to the first residual bottleneck block. It also provides the beginning of the feature extraction propagation for the remainder of the body.

E. Encoder — Body Architecture

	Input Dimensions		Expansion			Output
Layer	Feature Map	Size	Factor	Repeats	Stride	Channels
Residual Bottleneck	32	28×28	6	4	2	64
Residual Bottleneck	64	14×14	6	3	1	96
Residual Bottleneck	96	14×14	6	3	2	160
Residual Bottleneck	160	7×7	6	1	1	320

TABLE II
MOBILENET V2 BODY FOR MNIST DATASET.

The body contains the residual bottleneck blocks. We repeat each block a cetain amount of times and then downsample to the next layer in the network. Note that each residual block has 3 layers to compute to. They are not shown in Table II for brevity.

F. Decoder — Head Architecture

Layer	Input Dimensions		Expansion Factor	Repeats	Stride	Output Channels
	Feature Map	Size				
Global Average Pool	320	7×7	—	1	—	—
Dense (softmax)	1	320	—	1	—	10

TABLE III
MOBILENET V2 HEAD FOR CIFAR 10 DATASET.

Finally, the decoder takes in the encoded information and categorises the strong features derived fromt he encoder and passed it to a global average pooling layer followed by a fully connected layer to get the most likely classification for the given input image. A softmax operation is applied to transform the output into a probability mass function.

IV. TRAINING

A. Batch Normalization

Internal covarience shift is always a concern when trying to have networks converge as quickly and as efficiently as possible. It occurs during training when the input distribution changes throughout the model. For neural networks in particular, each layer's input is affected by the parameters of *all* other input layers. When there are significant differences in the results on a per-batch level, the model will have a difficult time optimizing and converging to a minima of error.

To combat the issues, we add another layer to our model after each convolution layer that normalizes each feature map by their respective mean and variance [5]. This technique that normalizes the batches is known as "batch normalization" (batch norm). Depending on the size of the batches, the model generalizes more cleanly overall. Because of these advantages, we do not need to use Dropout layers.

B. Learning Rate Scheduling

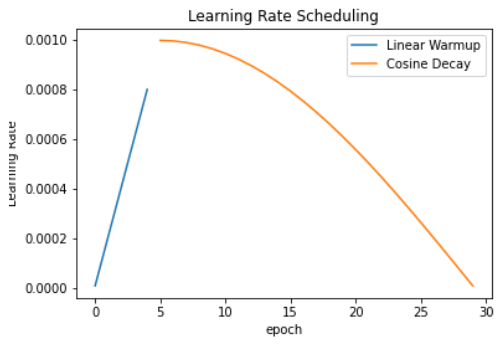

Fig. 5. Learning Rate Scheduling

A slightly modified version of simulated anneling takes place with the learning rate. As seen in Figure 5, the initial learning rate linearly increases until a certain threshhold is met. Once passed it, the learning rate follows a cosine decay pattern to allow for the model to converge to a local minima of error.

C. Adam Optimizer

Finally, instead of using mini-batch SGD, we utilize the RMSProp optimizer. In essence it still calculates gradiends similar to mini-batch SGD, but it tracks an average of the square of the calculated gradients. When taking the square roots of these stored values, we mitigate the magnitue of the weight update and prioritize the direction of convergence more than how much the model has moved.

V. CONCLUSION

MobileNet V2 is a new take on neural network design that is optimized for mobile and embedded systems. Using new techniques such as inverted residuals, depth wise convolution, learning rate scheulding, and batch normalization help in not only keeping the amount of computations low but also maintaining a high accuracy. The experiments shown have provided validation accuracy results that far exceed its contemporary models.

REFERENCES

[1] A. Pandey. (Sep. 2018). "Depth-wise convolution and depth-wise separable convolution," [Online]. Available: https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[3] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. arXiv: 1801.04381. [Online]. Available: http://arxiv.org/abs/1801.04381.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167.