Training convolutional neural networks on Fashion-MNIST

I describe three strategies for training convolutional neural networks on the Fashion-MNIST dataset: i) a simple neural network combining convolution and max-pooling layers, ii) transfer learning on ImageNet pre-trained MobileNet [2] and iii) reduced MobileNet architecture leading to only small drop in the accuracy but reducing the number of parameters by a factor of ~ 20 at the same time.

Simple CNN

The first model is a simple convolutional neural network inspired by the VGG [1] architecture. It consists of three consecutive blocks, each containing of two convolution layers (with ReLU activation) and one max-pool layer (see Table 1). The blocks are followed by a dense layer and softmax layer. The subsequent application of convolutional and max-pooling layers gradually reduces the spatial resolution of the input and increasing the number of features. The number of filters in each convolution layers was chosen to minimize the number of total parameters while preserving the performance.

In order to avoid overfitting, I apply dropout before the dense layer (relatively high dropping probability $p = 0.5$ was found to perform the best). A randomly selected small fraction (10%) of the training data is used as a validation set for hyperparameter tuning while the provided test set is used to evaluate the final performance of the network after all parameters have been fixed. An early stopping scheme is applied, with training iterations stopping when the value of the loss function on the validation set is no longer decreasing. Using the default Adam optimizer with learning rate 0.001, the iterations typically stop after 30-40 epochs.

Table 1: Architecture of the simple CNN with the estimate for the number of computational operations per single forward step.

layer / stride	shape	input size	$\sim$ no. ops. / step
Conv2D / s1	$3 \times 3 \times 1 \times 16$	$28 \times 28 \times 1$	112896
Conv2D / s1	$3 \times 3 \times 16 \times 16$	$28 \times 28 \times 16$	1806336
MaxPool2D / s2	2×2	$28 \times 28 \times 16$	12544
Conv2D / s1	$3 \times 3 \times 16 \times 32$	$14 \times 14 \times 16$	903168
Conv2D / s1	$3 \times 3 \times 32 \times 32$	$14 \times 14 \times 32$	1806336
MaxPool2D / s2	2×2	$14 \times 14 \times 32$	6272
Conv2D / s1	$3 \times 3 \times 32 \times 32$	$7 \times 7 \times 32$	451584
Conv2D / s1	$3 \times 3 \times 32 \times 32$	$7 \times 7 \times 32$	451584
MaxPool2D / s2	2×2	$7 \times 7 \times 32$	1568
Flatten		$3 \times 3 \times 32$	
Dropout	keep prob. $= 0.5$		
Dense	128×288	288	36864
SoftMax	10×128	128	1280
Total	**73K** parameters		**5.6M**

Table 1 also estimates the number of algebraic operations per single forward step. Applying convolutional filter of shape $K \times K \times C_{\text{in}} \times C_{\text{out}}$ (K is the kernel size and C_{in}, C_{out} the number of input and output channels) on the input of size $D \times D \times C_{\text{in}}$ requires roughly (s denotes the stride)

$$K \times K \times \frac{D \times D}{s^2} \times C_{\text{in}} \times C_{\text{out}} \tag{1}$$

algebraic operations while computational complexity of the dense layers corresponds to its number of parameters. The number of steps during the backward propagation is similar to those required

for one backward step and thus the total number of 5.6 million operations / step serves as an estimate for the time complexity. Training on Tesla K80 GPU required approx. 7s per epoch.

Table 2 shows the final accuracy evaluated on the test set. The training dataset (without the validation part) was also augmented by horizontal flipping of the images and randomly rotating each image by an angle drawn from a normal distribution (mean 0, variance $\pi/8$), although this data augmentation showed up not to be beneficial.

Transfer learning with MobileNet

The second strategy uses the MobileNet architecture [2] pre-trained on ImageNet dataset. Instead of standard convolutions, it deploys depth-wise convolutions to reduce the computational complexity. In depth-wise convolutions, a single filter of size $K \times K$ is first applied to each channel, followed by 1×1 convolution with C_{out} output channels. The computational cost of this procedure is

$$K \times K \times \frac{D \times D}{s^2} \times C_{\text{in}} + \frac{D \times D}{s^2} \times C_{\text{in}} \times C_{\text{out}} \tag{2}$$

For the input, images are first up-scaled to the size 56×56. Since the weights are already pre-trained, starting with smaller learning rate $lr = 0.0003$ and reducing it by a factor of 0.8 after each epoch improves the performance. The resulting accuracy exceeds 94% when training with the augmented dataset, but at the cost of higher number of operations and execution time (see Table 2).

Simplified MobileNet architecture

Finally, I try to simplify the MobileNet architecture while preserving the accuracy. I found that the accuracy drops only slightly when keeping the layers up to the 5^{th} depthwise convolution and applying global average pooling followed by dropout layer and softmax immediately afterwards. Again, the pre-trained ImageNet weights are used for the convolution layers. This model improves the accuracy compared to the simple CNN but is still relatively light-weight, with only twice as many parameters.

Table 2: Different network architectures with number of training parameters, operations required for the forwards step and performance on the test set after applying different data augmentation techniques (horizontal flipping and rotation).

method	parameters	ops. / step	time / epoch	data augmentation		
				none	flip	flip and rot.
simple CNN	**73K**	**5.6M**	**4s**	92.52	91.87	91.16
MobileNet	3239K	242M	20s	93.46	**94.13**	93.87
simplified MobileNet	140K	93M	11s	92.94	92.69	92.49

Discussion

The reported results show a tendency of increasing performance and computational complexity when going from the simple CNN through the simplified version of the MobileNet to the full version. The exact performance on the test dataset may slightly vary after independent training runs. A more systematic evaluation of the results (e.g. by averaging performance of multiple training runs), tuning of hyperparameters e.g. by Bayesian optimization or improving the final performance on the test set by bagging was not possible due to limited computational power (I was testing the architectures in Google Colab where running stops after approx. 60 mins.).

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[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.