VGG network

ABSTRACT

In this work it was investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. The main contribution is
a thorough evaluation of networks of increasing depth using an architecture with
very small (3×3) convolution filters, which shows that a significant improvement
on the prior-art configurations can be achieved by pushing the depth to 16–19
weight layers.

1 INTRODUCTION

With ConvNets becoming more of a commodity in the computer vision field, a number of attempts have been made to improve the original architecture of Krizhevsky et al. (2012) in a bid to achieve better accuracy. In this paper, it was addressed another important aspect of ConvNet architecture design – its depth. To this end, the authors fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3×3) convolution filters in all layers.
As a result, the model became significantly more accurate and not only achieve the state-of-the-art accuracy on ILSVRC classification and localisation tasks, but are also applicable to other image recognition datasets, where they achieve excellent performance even when used as a part of a relatively simple pipelines.

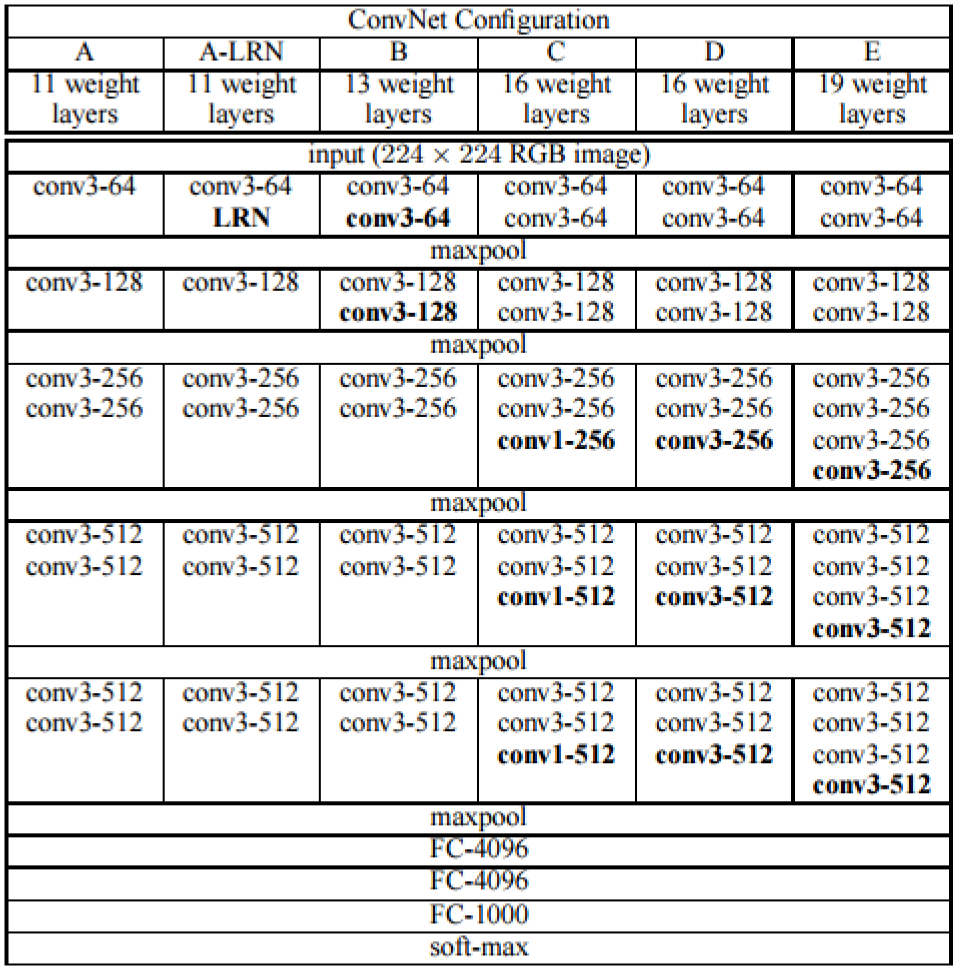
2 CONVNET CONFIGURATIONS

2.1 ARCHITECTURE

- Input is a fixed-size 224 × 224 RGB image.
- The only preprocessing step is subtracting the mean RGB value, computed on the training set, from each pixel.
- The image is passed through a stack of convolutional layers, where we use 3 × 3 filters (which is the smallest size to capture the notion of left/right, up/down, center)
- In one of the configurations we also utilise 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity).
- The convolution stride is fixed to 1 pixel.
- The spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. Layers.
- Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling).
- Max-pooling is performed over a 2 × 2 pixel window, with stride 2.
- A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class).

- The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.
- All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity.

Table 1: ConvNet configurations (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv receptive field size-number of channels”.
The ReLU activation function is not shown for brevity.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
	LRN	**conv3-64**	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
		conv3-128	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
			conv1-256	**conv3-256**	conv3-256
					conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	**conv3-512**	conv3-512
					conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

We will be implementing Network A, Number of parameters: 133
The layers have a 7 × 7 effective receptive field. So what have we gained by using, for instance, a stack of three 3×3 conv. layers instead of a single 7×7 layer? First, we incorporate three non-linear rectification layers instead of a single one, which makes the decision function more discriminative.Second, we decrease the number of parameters: assuming that both the input and the output of a three-layer 3 × 3 convolution stack has C channels
This can be seen as imposing a regularization on the 7 × 7 conv. filters, forcing them to have a decomposition through the 3 × 3 filters (with non-linearity injected in between).

The incorporation of 1 × 1 conv. layers is a way to increase the non linearity of the decision function without affecting the receptive fields of the conv. layers. Even though in our case the 1 × 1 convolution is essentially a linear projection onto the space of the same dimensionality (the number of input and output channels is the same), an additional non-linearity is introduced by the rectification function.

3.1 TRAINING

The training is carried out by optimizing the multinomial logistic regression objective using mini-batch gradient descent with momentum. The batch size was set to 256, momentum to 0.9. The training was regularized by weight decay (the L2 penalty multiplier set to 5 * 10^−4) and dropout regularization for the first two fully-connected layers (dropout ratio set to 0.5).
The learning rate was initially set to 10^−2, and then decreased by a factor of 10 when the validation set accuracy stopped improving. In total, the learning rate was decreased 3 times, and the learning was stopped after 74 epochs. We conjecture that in spite of the larger number of parameters and the greater depth of our nets, the nets required less epochs to converge due to (a) implicit regularization imposed by greater depth and smaller conv. filter sizes.
(b) pre-initialisation of certain layers.
The initialisation of the network weights is important, since bad initialisation can stall learning due to the instability of gradient in deep nets. To circumvent this problem, we began with training the configuration A, shallow enough to be trained with random initialisation. Then, when training deeper architectures, we initialized the first four convolutional layers and the last three fully connected layers with the layers of net A (the intermediate layers were initialized randomly). We did not decrease the learning rate for the pre-initialised layers, allowing them to change during learning.
For random initialisation, we sampled the weights from a normal distribution with the zero mean and 10^−2 variance. The biases were initialized with zero. It is worth noting that after the paper submission we found that it is possible to initialize the weights without pre-training by using the random initialisation procedure of Glorot & Bengio (2010).

3.2 TESTING

At test time, given a trained ConvNet and an input image, it is classified in the following way. First, it is isotropically rescaled to a pre-defined smallest image side, denoted as Q (we also refer to it as the test scale). We note that Q is not necessarily equal to the training scale S (as we will show later, using several values of Q for each S leads to improved performance). Then, the network is applied densely over the rescaled test image. Namely, the fully-connected layers are first converted to convolutional layers (the first FC layer to a 7 × 7 conv. layer, the last two FC layers to 1 × 1 conv. layers). The resulting fully-convolutional net is then applied to the whole (uncropped) image. The result is a class score map with the number of channels equal to the number of classes, and a variable spatial resolution, dependent on the input image size. Finally, to obtain a fixed-size vector of class scores for the image, the class score map is spatially averaged (sum-pooled).
the soft-max class posteriors of the original and flipped images are averaged to obtain the final scores for the image.

3.3 IMPLEMENTATION DETAILS

Our implementation is derived from the publicly available C++ Caffe toolbox (Jia, 2013), but contains a number of significant modifications, allowing us to perform training and evaluation on multiple GPUs installed in a single system, as well as train and evaluate on uncropped images at multiple scales. Multi-GPU training exploits data parallelism, and is carried out by splitting each batch of training images into several GPU batches, processed in parallel on each GPU. After the GPU batch gradients are computed, they are averaged to obtain the gradient of the full batch. Gradient computation is synchronous across the GPUs, so the result is exactly the same as when training on a single GPU. While more sophisticated methods of speeding up ConvNet training have been recently proposed, which employ model and data parallelism for different layers of the net, we have found that our conceptually much simpler scheme already provides a speedup of 3.75 times on an off-the-shelf 4-GPU system, as compared to using a single GPU. On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2–3 weeks depending on the architecture.