CV701 Assignment 4 - Lighter Stacked Hourglass

Jose Renato Restom   Sara Pieri
Mohamed Bin Zayed University of Artificial Intelligence, UAE
{jose.viera, sara.pieri}@mbzuai.ac.ae

1. Introduction

Human pose estimation (HPE) can be described as the process of approximating the configuration of the body from an image. To perform said task, models learn to detect and track semantic *key-points (points of interest)* on the human body, such as the knees, shoulders, and head, among others. The applications of HPE span several computer vision tasks, including but not limited to action recognition and object tracking, making it one of the key challenges in the area.
In this paper, we focus on the approach proposed by Newell et al. [5], called the stacked-hourglass architecture, which uses several up-scaling and down-sampling blocks repeated several times across the network to capture the local features and their positioning in the global context. Our goal is to modify the stacked-hourglass network (with two hourglasses) and improve the total number of parameters and the accuracy of the model.

2. Architecture Design

A stacked Hourglass [5] is a convolutional network architecture for human pose estimation that combines multiple hourglasses stacked together. Each hourglass contains a bottom-up and top-down section allowing the network to process the input at different resolutions to capture the spatial relationships across all scales of the images.
In the top-down section, the features are lowered down using convolution and max pooling. Additionally, after each max pooling layer, the network branches off and applies more convolutions to the original pre-pooled resolution. After reaching the lowest resolution, in the top-down section, nearest neighbor upsampling is applied and the two sets of features are added elementwise to capture information across two adjacent resolutions. Finally, two 1x1 convolutions are applied to produce the predictions. The network's output is a set of heatmaps reporting the probability of a joint occurring at pixel-level.
The vital elements of the hourglass architecture are bottleneck building blocks that perform convolution in residual learning modules ("Inception"-based designs) as pre-

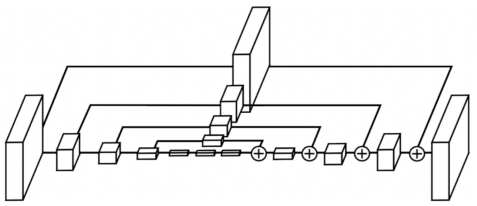

Figure 1. The architecture of a single hourglass. The input is downsampled using maxppoling in the top-down section and upscaled using NN interpolation in the botton-up section.

sented by He et al. [2]. Operating at the full input resolution (256x256 pixels) requires a significant amount of GPU memory. Therefore, bottleneck blocks that do not use filters greater than 3x3, reduce the number of parameters at each layer and curtail total memory usage. An example of the original bottleneck building block is shown in Fig. 7.a. The original paper used eight hourglasses stacked. Our baseline architecture consists of two hourglasses and 31 bottleneck blocks in total.
In this study, we explored and modified different bottleneck designs to investigate potential performance boosts while reducing the number of network parameters.

3. Literature Review

There are numerous works on how to make deep learning models smaller, faster and more accurate. Here we divide the different approaches in two main categories, as established by Berthelier et al. [1], *compression techniques* and *architecture optimization*. The first kind focuses on how to modify established models to optimize them (a training or pretraining phase is necessary), while the latter takes a different route by creating optimized architectures from scratch. Figure 2 conveys different methods for this purpose.

3.1. Compression techniques

3.1.1 Knowledge distillation

Knowledge distillation consists of using transfer learning from larger, deeper networks to shallower ones. The main

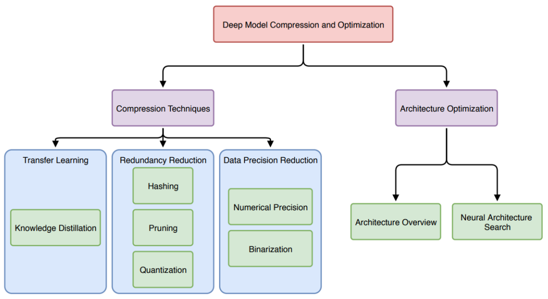

Figure 2. Compression and Optimization approaches [1]

approach is what's called the teacher-student approach. We train a deep neural network and create a synthetic dataset by relabeling the data with this network, then we train the shallow "student" network to mimic the deeper net by using the synthetic data as training set.

3.1.2 Pruning

Pruning techniques are used to remove parameters of a network that are not necessarily good for inference. In a general sense, pruning erases some of the connections from the network by deleting their weights and biases. Typically, a 3-step approach is taken; firstly, the entire network is trained; secondly, the less important weights are established and dropped; finally, the new pruned network is fine-tuned. The process can be further improved by making it iterative, as shown in Figure 3.

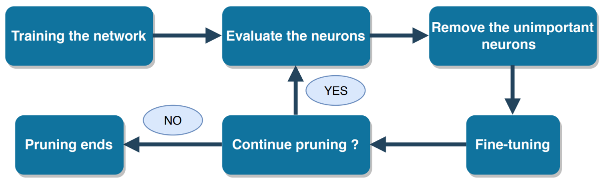

Figure 3. Iterative pruning [1]

3.1.3 Hashing

Hashing techniques are used to regroup the data in a neural network into clusters, reducing the number of bits required to represent every weight. The grouped connections shared the same value and as a result, every connection that in the *ith* cluster has the same weight value *W*.

3.1.4 Reducing numerical precision and Binarization

This solutions aims to reduce the model's complexity by limiting the numerical precision of the data. Training is typically done using a 32-bit floating-point precision, the goal here is to decrease the number of bits used (16, 8 or even less) and to change from floating-point to a fixed-point representation.

Technique	Method	Pros	Cons
Knowledge distillation	Using a deep CNN to train a smaller CNN.	Small models with comparable performances.	Models can only be trained from scratch; Difficult for the tasks other than classification.
Hashing	Indexing neurons into a hash table.	Better parallelization; Better data dispersion; Less computation time.	Considerably slower if the model is too sparse.
Pruning	Deleting neurons that have minor influence on the performance.	Significant speed up and size reduction; Compression rate is 10x to 15x (up to 30x).	Pruning process is time-consuming; Less interesting for too sparse model.
Quantization	Reducing the number of distinct neurons by gathering them into clusters.	High compression rate : 10x to 15x; Can be coupled with pruning.	Considerably slower if the model is too sparse.
Numerical Precision	Decreasing the numerical precision of the neurons.	High compression rate and speed up.	Higher precision is needed during the parameters update; Could require specific hardwares.
Binarization	Decreasing the numerical precision of the data to 2 bits.	Very high compression rate (30x) and speed up (50x to 60x).	Higher precision is needed during the parameters update.

Table 1: Summary of different compression methods.

Figure 4. Pros and cons of different compression methods

3.2. Architecture optimization

This approach centers it's attention on creating optimized architectures from the ground-up instead of finding methods to optimize them afterwards.

3.2.1 Modules and Separable convolutions

Modules arrange multiple convolutional layers with different kernel sizes into a block. The main goal as an optimization method is to decrease the number of parameters of the network by taking advantage of the several kernels to create cheap *bottlenecks*. For example, a large convolutional block (7x7, 5x5) could be replaced with a less computationally expensive block that stacks smaller convolutions together (1x1, 3x3).

3.2.2 Neural architecture search

The advances in understating deep neural networks and the creation of new modules and architectures have made it possible to tackle challenges that years prior were not feasibly for deep learning. However, since the options for mounting an architecture have increased significantly, choosing by hand what to use is now a more complex task. Neural architecture search was born as an attempt automate the process of choosing the blocks and functions that compose a neural net.

4. Implementation

4.1. Building blocks

The basic operations used for the modifications to the baseline are explained bellow:

4.1.1 Dilated convolution

The dilation factor in a convolution increases the area covered by the kernel without increasing the number of operations we need to compute. The Figure 5 shows how the dilation factor affects the kernel.

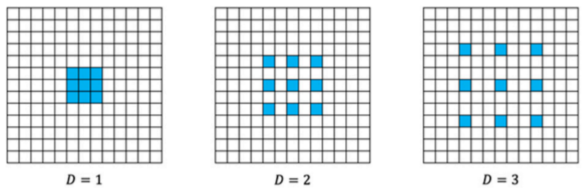

Figure 5. Kernels with different dilation factors [3]

4.1.2 Depth separable-wise convolution and the point-wise convolution

In order to reduce the numbers computations required in a standard convolution Francois Chollet [] introduced the Depth-wise separable convolution. It uses one filter per each channel in the input unlike the traditional one and then combines the channel-wise relations by following it with a point-wise convolution (1x1), which also helps resize the dimensions of the output to the desired one. The Figure 6 shows the concept for this operation.

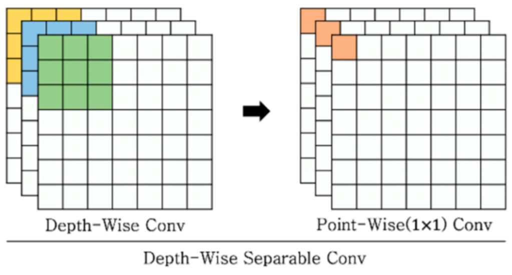

Figure 6. Depth-wise separable convolution followed by the point-wise convolution [3]

4.2. Multidilated Residual Bottleneck

The original stacked hourglass constructed the bottleneck using standard convolutional blocks as shown in Figure 7.a. Each of the blocks combine in sequence Batch Normalization, ReLu, and Convolution. However, due to their structure, residual blocks perform better in local features where the receptive field is small. Moreover, the original bottleneck require many parameters resulting in high computation cost. To improve the original structure, the multidilated light residual block Figure 7.b was introduced to reduce the number of parameters while expanding the receptive field. The network was made lighter by using depthwise separable convolutions and combining the outputs of three convolutional blocks with different values of dilation as shown in Fig 7.b.

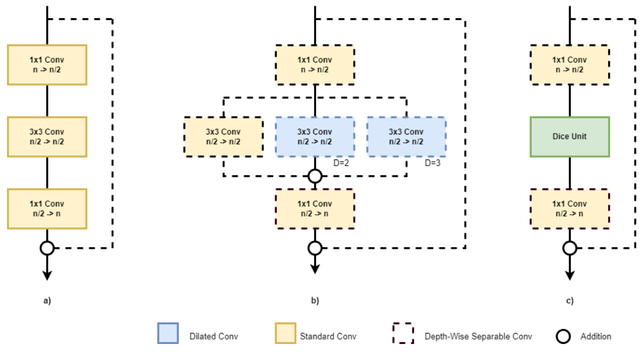

Figure 7. Structure of the compared models. From left to right: the original bottleneck, the base multidilated bottleneck, the bottleneck obtained using DiCE unit.

4.3. Expanded Multidilated Residual Bottleneck

We decided to extend the idea used for the Multidilated Residual Bottleneck by increasing the number of parallel blocks. The dilation factors used were D = 5 and D = 7 for the newly added blocks.

4.4. Dice Residual Bottleneck

Depth-wise convolutions are efficient but do not take into account relation information across channels. The DiCE unit [4] was proposed as a possible solution to this problem. DiCE applies convolution across each dimension of the input to learn local dimension-wise representations and then combines them to incorporate global information (see Fig.4.4). We incorporated a dice unit into the structure proposed by the original paper, resulting in an architecture that, as far as we know, has never been presented.

5. Discussion

The first experiment performed was the multidilated residual block. Using this idea, we replaced the standard convolutions in the baseline's bottleneck; and achieved an increase in the accuracy of the model by 0.5% while simultaneously decreasing the number of parameters by 40%.

Next, we proceeded to expand on this idea by appending more parallel units onto the multidilated block. According

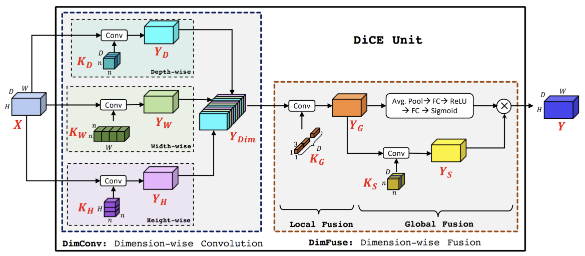

Figure 8. DiCE combines the input spatial and channel-wise information in two stages. The output is prodced by first applying dimension-wise convolutions (DimConv) and than dimension-wise fusion (DimFuse).

to the results in our experiments, adding a block with an increased dilation (D = 5) enhanced the model's performance. However, when the dilation factor became too large (D=7), too much zero padding caused the network to fail to learn the spatial features, resulting in a loss in the ability to localize joints.

The experiments using the DiceUnit showed that the number of parameters is clearly reduced, 2.6 M for 1 unit (Dice1) and 3.7 for 2 units (Dice2). However, the final accuracy dropped by when more than 1% compared to the baseline. Although, it is important to mention that we were not able to incorporate in the model the same number of units per bottleneck as in the baseline and the multidilated models, since we ran into a memory constrain with the GPU. The table bellow summarizes some metrics for all experiments.

Model	NP	Train Acc	Val Acc	Mean PCKh	IT	TT
Baseline	6,730,912	70.76	73.42	75.94	**01:16**	**2:53**
Multi	3,882,912	71.01	73.60	76.44	01:19	2:55
Multi5	4,420,704	71.63	74.04	**76.73**	01:18	2:56
Multi7	4,959,264	**71.84**	**74.12**	76.67	01:18	2:57
Dice1	**2,670,512**	69.22	72.12	74.68	01:20	3:49
Dice2	3,704,560	70.39	72.23	74.88	01:32	6:15

Table 1. Methods Statistics.

References

[1] Anthony Berthelier, Thierry Chateau, Stefan Duffner, Christophe Garcia, and Christophe Blanc. Deep model compression and architecture optimization for embedded systems: A survey. *Journal of Signal Processing Systems*, pages 1–16, 2020. 1, 2

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[3] Seung-Taek Kim and Hyo Jong Lee. Lightweight stacked hourglass network for human pose estimation. *Applied Sciences*, 10(18):6497, 2020. 3

[4] Sachin Mehta, Hannaneh Hajishirzi, and Mohammad Rastegari. Dicenet: Dimension-wise convolutions for efficient networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 3

[5] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016. 1