An Investigation of 2D Keypoints Detection on Challenging Scenarios using Depthwise Separable Convolutions: A Hand Pose Estimation Case Study (supplementary material)

Sibgrapi paper ID: 104

I. Training

In this section, we detail the training procedures for each module, describing loss and training hyperparameters such as learning rate and optimizers. For all configurations, we follow the training procedure described by Zimmermann and Brox [1].

HandSegNet training: We train *HandSegNet* for hand segmentation on ***R-train*** for a total of $40,000$ iterations on the dataset, using a standard softmax cross-entropy loss and a batch size of 8. We have used the Adam optimizer [2] with a learning rate scheduler, beginning with 10^{-5} for the first $20,000$ iterations on the dataset, and decaying by a scale of 10 each $10,000$ iterations. The architecture proposed for *HandSegNet* is exposed in Table I.

TABLE I
SUMMARY OF THE HANDSEGNET ARCHITECTURE, WHICH RECEIVES A 256X256 RGB IMAGE AS INPUT AND OUTPUTS THE CORRESPONDENT HAND MASK.

#	Layer type	Layer shape
-	Input image	(256, 256, 3)
1	Convolution + ReLU	(256, 256, 64)
2	Convolution + ReLU	(256, 256, 64)
3	Maxpool	(128, 128, 64)
4	Convolution + ReLU	(128, 128, 128)
5	Convolution + ReLU	(128, 128, 128)
6	Maxpool	(64, 64, 128)
7	Convolution + ReLU	(64, 64, 256)
8	Convolution + ReLU	(64, 64, 256)
9	Convolution + ReLU	(64, 64, 256)
10	Convolution + ReLU	(64, 64, 256)
11	Maxpool	(32, 32, 256)
12	Convolution + ReLU	(32, 32, 512)
13	Convolution + ReLU	(32, 32, 512)
14	Convolution + ReLU	(32, 32, 512)
15	Convolution + ReLU	(32, 32, 512)
16	Convolution + ReLU	(32, 32, 512)
17	Convoluition	(32, 32, 2)
18	Upsampling	(256, 256, 2)
19	Argmax	(256, 256, 1)
-	Output hand mask	(256, 256, 1)

PoseNet training: We initialize the first 17 layers with the weights of the Convolutional Pose Machines model [3], and initialize all other layers randomly. We begin by training *PoseNet* on ***R-train*** for $30,000$ iterations using a batch size of 8 and a L_2 loss. Again, we use the Adam optimizer with a learning rate scheduler, beginning with 10^{-4} for the first $10,000$ iterations, 10^{-5} for the next $10,000$ iterations and 10^{-6} until the end. We then proceed to train *PoseNet* on ***S-train***, following the same hyperparameters as training in ***R-train***. The architecture for *PoseNet* is exposed on Table II.

II. Environment

We conducted the experiments listed in this work on a virtual environment created with the Anaconda distribution. Virtual environments serve to help manage dependencies and to isolate projects from the operating system, meaning that updates and changes to the system's settings will not affect the environment configuration. The virtual environment uses the Python programming language in the version 3.7.4 in an Ubuntu GNU/Linux 16.04 x64 host operating system.

We configured the virtual environment to use CUDA, a parallel computing platform, and a programming model developed by NVIDIA to accelerate the execution of applications through GPU usage. For our setup, while we execute part of the workload in CPU (processing and filtering images, for example), the core of the operations are performed on GPU (tensor operations, for example). The development environment for the usage of CUDA is the CUDA Toolkit, and the version used for the experiments is the 10.0.130.

Alongside CUDA, we also configured our environment to use NVIDIA cuDNN, the Deep Neural Network library, a GPU-accelerated library of primitives for deep neural networks. These implementations have highly optimized standard routines such as convolutions, pooling, and activation. Using cuDNN allows us to focus on the architectural problems and to train the models instead of spending time on low-level GPU performance tuning. During the experiments, the cuDNN version used was the 7.6.0.

The main framework used for training and evaluating was Tensorflow [4], an open-source software library for machine learning. This framework was chosen mainly due to the optimizations found in the depthwise separable convolution operation. We list the main packages installed in the environment related to the applications' performance in Table III.

We executed the codes for training and evaluating all proposed models in a desktop computer powered with an Intel Core i7-4790K CPU, with 32 gigabytes of RAM. We executed

TABLE II

SUMMARY OF THE ORIGINAL AND PROPOSED *PoseNet* ARCHITECTURE. LAYERS FROM 1 TO 17 HAVE THEIR WEIGHTS PRE-LOADED FROM CONVOLUTIONAL POSE MACHINES [3] AND WERE NOT SWAPPED FOR DEPTHWISE SEPARABLE CONVOLUTIONS.

#	Original configuration		Mixed configuration (ours)	
	Layer type	**Layer shape**	**Layer type**	**Layer shape**
-	Input from HandSegNet	(256, 256, 3)	Input from HandSegNet	(256, 256, 3)
1	Convolution + ReLU	(256, 256, 64)	Convolution + ReLU	(256, 256, 64)
2	Convolution + ReLU	(256, 256, 64)	Convolution + ReLU	(256, 256, 64)
3	MaxPooling2D	(128, 128, 64)	MaxPooling2D	(128, 128, 64)
4	Convolution + ReLU	(128, 128, 128)	Convolution + ReLU	(128, 128, 128)
5	Convolution + ReLU	(128, 128, 128)	Convolution + ReLU	(128, 128, 128)
6	MaxPooling2D	(64, 64, 128)	MaxPooling2D	(64, 64, 128)
7	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
8	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
9	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
10	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
11	MaxPooling2D	(32, 32, 256)	MaxPooling2D	(32, 32, 256)
12	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
13	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
14	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
15	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
16	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
17	Convolution	(32, 32, 21)	Convolution	(32, 32, 21)
18	Concat(16, 17)	(32, 32, 533)	Concat(16, 17)	(32, 32, 533)
19	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
20	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
21	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
22	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
23	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
24	Convolution	(32, 32, 21)	Depthwise Sep. Convolution	(32, 32, 21)
25	Concat(16, 17, 24)	(32, 32, 554)	Concat(16, 17, 24)	(32, 32, 554)
26	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
27	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
28	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
29	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
30	Convolution + ReLU	(32, 32, 128)	Depth. Sep. Conv. + ReLU	(32, 32, 128)
31	Convolution	(32, 32, 21)	Depthwise Sep. Convolution	(32, 32, 21)
-	Output heatmaps	(32, 32, 21)	Output heatmaps	(32, 32, 21)

TABLE III

OVERVIEW OF THE PACKAGES INSTALLED IN THE VIRTUAL ENVIRONMENT THAT ARE PERFORMANCE-RELATED.

Package name	**Version**
cudatoolkit	10.0.130
cudnn	7.6.0
keras-gpu	2.2.4
numpy	1.16.4
opencv-python	4.1.1.26
python	3.7.4
scipy	1.3.1
tensorflow	1.13.1

TABLE IV

Train set	**Ratio**	**original**			**mixed**		
		Mean	Median	AuC	Mean	Median	AuC
Standard	-	**16.958**	**6.095**	**0.733**	17.608	6.313	0.723
Trained on S-train	0.1	**18.372**	**6.207**	**0.722**	19.135	6.427	0.711
	0.2	**22.085**	**6.412**	**0.697**	22.902	6.663	0.686
	0.3	**28.583**	**6.731**	**0.662**	29.854	6.980	0.651
Trained on S-train*	0.1	**18.725**	**6.426**	**0.717**	18.939	6.663	0.710
	0.2	21.444	6.497	0.696	**21.222**	**6.563**	**0.699**
	0.3	27.783	6.843	0.667	**27.717**	**6.800**	**0.670**

Quantitative evaluation of **dense** and **depthwise** architectures on ***S-val**** processed with occlusion, following the metrics proposed by Zimmermann and Brox [1].

the CUDA operations on an NVIDIA RTX 2080 Ti, with 12 gigabytes of RAM using the driver version 410.93.

III. EXAMPLES FROM THE AUGMENTED DATASET

In this section, we show a few examples of cases created on the augmented dataset. In Figure 1, we show an example of how the images are affected by the defocused lens simulation. In Figure 2, we show examples of images with the motion blur filter applied, simulating motion. In Figure 3, we show the proposed pipeline to generate images with occluded regions and on Figure 4 we show examples created using this pipeline. Finally, on Figure 5 and Figure 6 we show examples from the Gaussian Noise and Salt & Pepper sets compared with the original images.

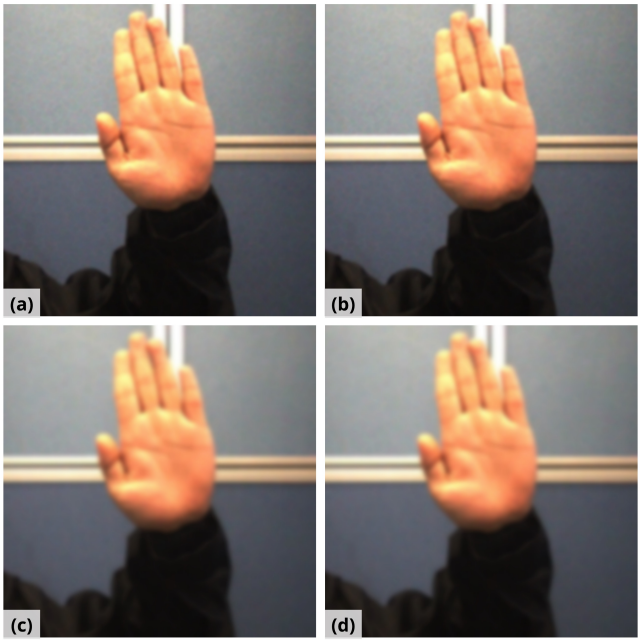

Fig. 1. An image from the ***S-val**** set that was selected and processed using Gaussian blur. **(a)** Original image, without processing. **(b)** Original image blurred with Gaussian blur with kernel size of 7×7, **(c)** 13×13 and **(d)** 21×21.

Fig. 2. An image from the ***S-val*** set that was selected and processed using horizontal motion blur. **(a)** Original image, without processing. **(b)** Original image blurred with horizontal motion blur with kernel size of 15×15, **(c)** 30×30 and **(d)** 45×45.

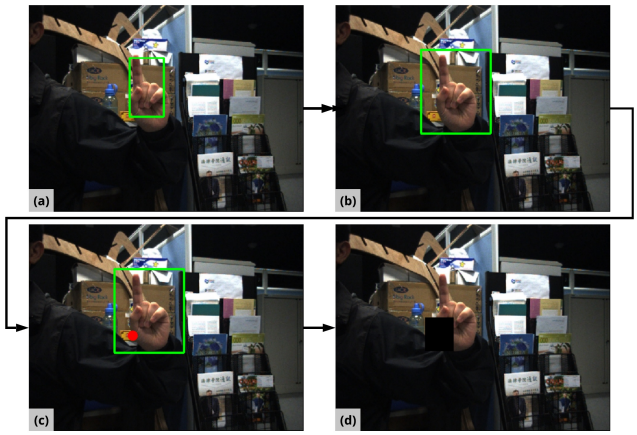

Fig. 3. Full pipeline of the occlusion process. **(a)** We estimate the bounding-box using ground-truth annotations. **(b)** We expand the bounding-box with a 15% margin. **(c)** We select a random point inside the bounding-box and use it to **(d)** draw a square over the image.

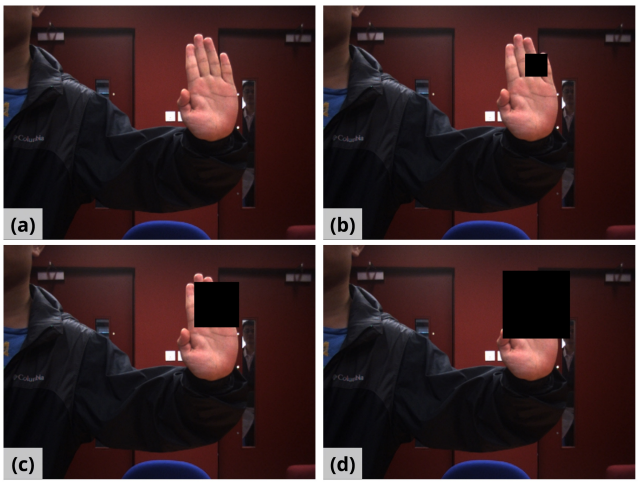

Fig. 4. An image from the ***S-val*** set that was selected and processed adding occlusion. **(a)** Original image, without occlusion. **(b)** Square with the 0.1, **(c)** 0.2 and **(d)** 0.3 parameters that control the size of the square.

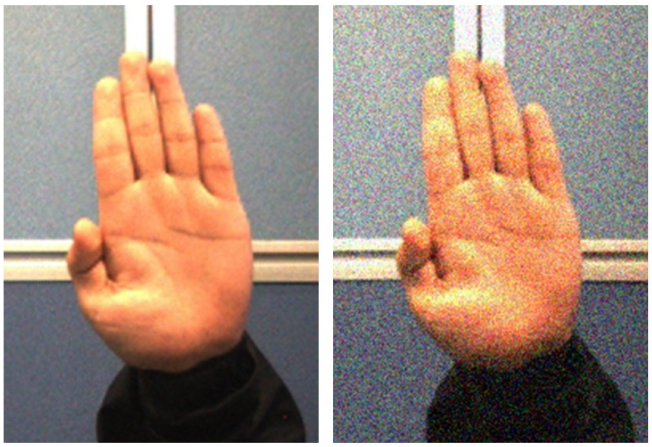

Fig. 5. Original image from the training set against its augmentation with Gaussian noise.

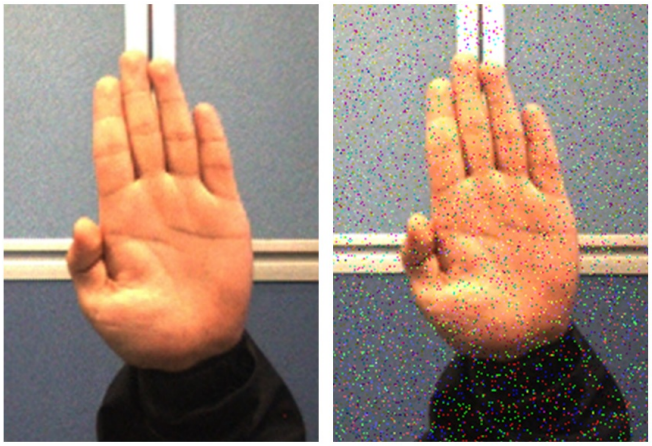

Fig. 6. Original image from the training set against its augmentation with S&P noise.

REFERENCES

[1] C. Zimmermann and T. Brox, "Learning to Estimate 3D Hand Pose from Single RGB Images," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-October, pp. 4913–4921, 2017.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4724–4732.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/