Sea lion-ception

Konrad Magnusson, Tim Olsson, Victor Ähdel

KTH - DD2424

Abstract. Counting occurrences of an object in an image is a task that is demanding for humans, especially if there are very many objects to be counted or if the objects overlap. We investigate a previously proposed deep learning-based method for counting bacterial cells in microscopy images, and modify it to count sea lions in aerial photography. The results from models we propose in this paper are not on par with human performance, but certainly show potential. With further tuned models, above-human performance could possibly be achieved.

1 Introduction

Counting objects or individual occurrences of an object in an image is a task where humans often fail due to fatigue or loss of focus. Automating the counting process would alleviate the workload and hopefully improve accuracy as a result. In this report, the focus is to create a machine learning model that counts sea lions in aerial photography. The data set was provided by an online competition hosted by Kaggle [1]. The sea lions present in these images are of varying age, sex, and size, and counting these manually can be an extremely difficult task due to many of them blending in very well with the rocky environment. In addition, the sea lion pups in the images are so small as to barely be noticeable, and some images contain overlapping sea lions.

Fig. 1. An aerial photograph of sea lions in their natural habitat. Taken from the NOAA Kaggle dataset.

The original challenge authored by NOAA included to also classify the sea lions into 5 classes based on age and gender, but the time period allocated to this project did not allow for more complex models to be trained, and so a simple total count was settled on as the final goal. In this report we aim to elucidate how well the deep learning-based method proposed by Cohen et al. [2] can be adapted to the sea lion counting problem.

2 Background

Counting objects in digital images has been tackled in several different ways. Notable ways of counting are 1. using object detection algorithms [3] such as YOLO [4] and SSD [5], 2. density estimation as in the original microscopy paper [6], and 3. redundant counting methods like Count-ception[2], which is the method of choice for this project.

If the images are very homogeneous and the target objects somewhat clearly separated, a simple approach using segmentation/connected components labeling has been seen to be sufficient [7]. Such methods also rely quite heavily on the objects having a certain brightness value or color that is distinct from other objects in the image, if the objects to count are not themselves grouped into some distinct area [8]. This can make the approach inefficacious for many types of images that you might want to count objects in. The necessary image transformations are also usually handcrafted, which implies that some image processing knowledge might be necessary to adapt the adapt the method to a new domain, which can be a downside for small businesses.

SSD and YOLO find objects by splitting the image into areas of interest, in which they look for features learned from labeled training data. The idea of counting using object detection algorithms is the most natural: recognizing objects in an image, and then simply counting the occurrences. In this way, counting is already a part of the algorithm, since one can essentially just threshold at a certain confidence value in order to get a count. However, there are flaws in this approach which become clear in the case of counting sea lions. Object detection-based algorithms do not handle overlapping objects very well. This implies that a group of sea lions close to each other may be counted as one. Here, methods of counting using density maps come in handy.

The modern ideas of counting using density maps are based on an approach described in [9]. In this approach, they are given images with point-annotated objects from which they construct density maps. Retrieving the count of objects is then achieved by integrating over the density maps. In [9], dense SIFT features are used as input to a linear-regression to predict density maps.

Deep learning methods have also built on this approach. In [6], a convolutional neural network is used to regress a cell spatial density over the image. Here, a 100x100 input image is convolved to a 100x100 density map that is then integrated over in the aforementioned manner to compute the final cell count.

Count-ception [2] is based on the density map idea, especially the methods that implement deep learning, with a few changes. Instead of predicting a density

map, count-ception proposes to predict a count map which contains redundant counts based on receptive fields. In count-ception a small receptive field is processed over the whole image by a convolutional neural network. By padding the input image with blank pixels, the method allows a fully convolutional counting, implying that every pixel in the input image is accounted for by the whole convolution kernel. This is referred to in the paper as *redundant counting*, and is the main difference between the two methods.

Count-ception's network architecture is based on the inception family of networks. Inception networks compute convolutions of different kernel sizes in parallel and then append the resulting tensors [10][11]. Both count-ception and microscopy test their methods on a dataset consisting of images of bacterial cells, i.e. spherical blobs of very similar sizes in a monochrome image.

3 Approach

We followed the approach very much outlined in count-ception[2], which is described in section 2. The authors made some questionable choices however, so we modified the net where necessary. We also had to modify the net a bit just to adapt it to the new domain.

3.1 The data

The training data from Kaggle consists of 947 raw training images as well as 947 dot-annotated target images. Furthermore, there are some 18000 images without matching labels, which unfortunately therefore are useless to us. In these images, there are five different classes of sea lions: adult males, juvenile males, adult females, juveniles, and pups. Because of time and resource constraints, we do not count the occurrence of each class, but instead aggregate the classes together to count the occurrence of a sea lion. In this aggregation we skip the pups since they are quite distinct from the other classes.

To create target images, we difference the original image with the dotted images, which transforms it into an image where it is easy to detect the dots. It is not entirely perfect though, and occasionally misses some dots. After that, boxes of 32x32 pixels with value one are added to a label image around each dot, corresponding to the target box for that sea lion. In order for the integration at the end to be correct, we have to ensure that this box is entirely inside the image, so the label images are 32 pixels larger than the actual images. If the sea lions are close enough that these boxes overlap, this implies that they add up, and it is not uncommon to see some pixels with values of 3 and upwards, in tightly grouped packs.

Kaggle images are also very large (between 16-20 megapixels), which is too large to handle reasonably with a deep network using current hardware. So we decided to split them into images of size 256x256 (which is the same as in count-ception [2]). We do not handle the edges between these small images, so sea lions that end up very close to the edges will occasionally match poorly. This

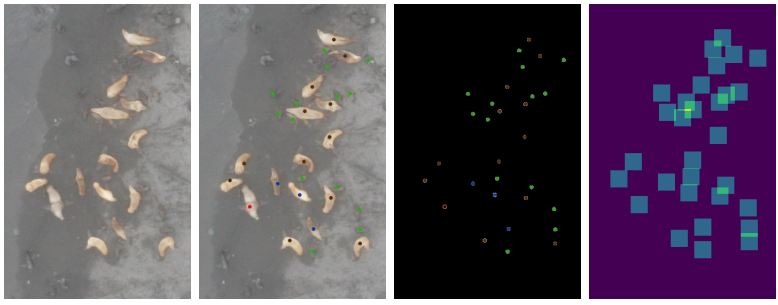

Fig. 2. Left to right: Original image, Dot annotated image, Dot-image thresholded on difference image, Targets

also means that entire-image counts may be off, but we test mostly on the smaller (split) images.

Out of the 947 images, the first 900 are used as the training set, and the rest are split into validation and testing sets. When split into smaller images, each image produces about 330 smaller ones, but almost 95% don't contain a single sea lion, which would affect the training significantly. In order to alleviate that, we take only 5% as many negative (empty) images as images with sea lions. The findings in [12] suggests that a value between 16-18% might be optimal, but it is likely sufficient with 5% for us since most images are sparse in sea lions to begin with. After this, there are about 18000 small images in total. The small validation/test images are shuffled (with a fixed seed, so test set is always the same), and 20% of those end up in validation, and 80% in testing. The somewhat low percentage of validation images are so that validation does not take too much time when training, since it is actually quite a lot of images in total.

3.2 Count-ception details

The exact network structure of count-ception is in table 1. There, a "Conv" layer consists of a convolution followed by a batch normalization layer, then fed through a Leaky ReLU layer (leakyness of 0.01). An inception layer consists of a concatenation of a 1x1 and a 3x3 version of such a layer. The total number of parameters (excluding batch normalization) is 1560337.

The theoretical receptive field for this is $(3-1)\times 6+(15-1)+(17-1)+1 = 43$, i.e. a square of 43x43 pixels. But the effective receptive field is probably smaller, and shaped more like a Gaussian bell [13]. The size of the effective receptive field seems sufficient for the synthetic dataset used in the original paper, since the blobs' sizes are very homogeneous, and it fits it well. But in order to count objects that are somewhat larger than those, one might need to increase the receptive field.

Type	Filters	Params	Output size
Conv (3x3)	64	1972	318x318x64
Inception	16+16	10282	318x318x32
Inception	16+32	9775	318x318x48
Conv (14x14)	16	172816	304x304x16
Inception	112+48	8864	304x304x160
Inception	64+32	56416	304x304x96
Inception	40+40	38480	304x304x80
Inception	32+96	71808	304x304x128
Conv (17x17)	32	1183776	288x288x32
Conv (1x1)	64	2112	288x288x64
Conv (1x1)	64	4160	288x288x64
Conv (1x1)	1	65	288x288x1

Table 1. Original model. All inceptions in this net are a combination of 1x1 and 3x3 Convs. Note that we have skipped a spurious increment from the original code (In their code they also tried out different strides, which may have required the extra pixel for correctness, but we do not need it.), and that 288 is $256 + 32$.

3.3 Modifications

Modifications from the original network were necessary because of the increase of complexity in the problem of counting sea lions in real life scenarios compared to similar-looking blobs in simple environments. Because of time constraints, we were only able to try out every single combination of model changes systematically, but rather rapidly iterated on the most promising networks on a subset of the data. From the results of those trials, we decided to use the model described in section 4.2 as our baseline. From this baseline, we then tried out some more configurations that seemed reasonable, as well as testing to remove some of the modifications to verify that they did not hamper the net. The adjustments made to the original count-ception network, to create the new baseline network, are described below.

The main way of increasing the complexity of the network was done by adding more filters to the layers. Another way of increasing the complexity was to follow the inception family of models[11] more closely. By scaling up the network in clever ways using inception layers[11], one can achieve more effective complexity with fewer parameters. In our network, we added 5x5 convolutions to the inception layers, as well as 1x1 convolutions before the large convolutional layers. The 5x5 convolutions were added in order to increase the effective receptive field, hence possibly matching better on large sea lions without needing to increase the label box size. The purpose of the 1x1 convolutions is to reduce the filter size of that layer, before the large (and very costly) 15x15 and 17x17 convolutions. Since the new features from the 1x1 layer is a nonlinear combination of the features from the previous layer, it leads to a comparably powerful model in terms of representational power. At the same time, it reduces the parameters used by the following layer by a factor equal to the ratio of output filters compared to

input filters. For us, this simple modification lead to a decrease in parameters from 1.5 million to 0.9 million, which means that the parameters for the large convolutions are of a similar scale to those used in the inception layers.

Another significant modification to the network is the swap of the order of the batch normalization layers from before the activation functions to after them. This goes against the original paper on batch normalization [14], but has been proven to be successful in later benchmarks [15]. One explanation might be that it is better to normalize after the ReLU because it will then not take into consideration all the negative values which will be squashed to zero (or close, for leaky) anyways.

The authors choice to include batch normalization also on the output layer was unmotivated, and our intuition was that it would not aid the training, so we skipped it for the baseline layer.

Pixel-wise MSE was used as the loss function. The original authors went with MAE as they got the best results with that, and claimed that they possibly needed the additional regularization. We did not find that we needed more regularization though, and so MSE performed better.

Besides the major changes, hyperparameters had to be adjusted to fit the new network model and problem. A learning rate of 0.0001 was used together with the Adam optimizer[16]. 50 epochs were run and each epoch ran through 500 steps of batches, where each batch consists of four images. Once trained, the model with the best validation accuracy was used as our final model.

4 Experiments/Results/Conclusions

4.1 Practicalities

All training and testing was performed on AWS [17], using one nVidia Tesla K80 [18]. With that hardware, most models converged after approximately 20 hours of training. For creating the network, we used the Python library Keras [19] with the Tensorflow [20] backend.

The final MSE presented is count-MSE, meaning the MSE of predicted amount of sea lions per sub-image. The loss function used in the training process was pixel-wise MSE, so the two error metrics are not directly comparable. The reason for this divergence in loss metrics is that pixel-wise MSE makes more sense for count maps, while a count-MSE is more intuitive when comparing final results. An improvement in pixel-wise MSE should improve the count-wise MSE, but not necessarily vice-versa. Since there should not actually be any half-counted sea-lions, we also look at the MSE that we would get if we first rounded the count-wise MSE to the nearest natural number. As a baseline, we calculated the mean of counts of the training data, and calculated the MSE that we would get if we just guessed that, which was 26.5.

4.2 Models

Fig. 3. Sample original images from the test set

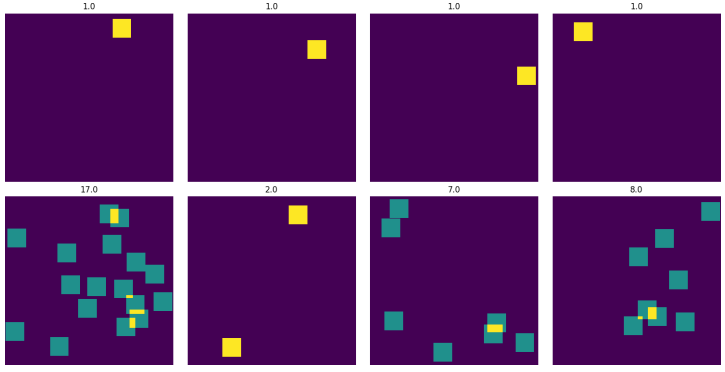

Fig. 4. Sample target images from the test set

Model 1: Default The first model used all the modifications in section 3.3. The exact net parameters are described in table 2. The total number of parameters are 998589. The baseline performs significantly better than the mean baseline, and seems to notice most of the sea-lions, as visible in the predictions in figure 5.

- MSE: 8.099
- Rounded-count MSE: 8.197

Model 2: More filters Since we did not notice much of an effect from overfitting, a straight up more complex model might perform better. Model 2 is therefore like model 1 but with the number of filters and such as described in table 3. The total number of parameters were 1294405. The increased number of filters seemed to have improved on the model, and the resulting MSE is somewhat lower than the default net.

- MSE: 7.604

Type	Filters	Params	Output size
Conv (3x3)	64	1792	318x318x64
Inception	24 + 20 + 10	29110	318x318x54
Inception	24 + 20 + 10	24570	318x318x54
Conv (1x1)	32	1760	318x318x32
Conv (17x17)	32	230432	304x304x32
Inception	112 + 40 + 20	31276	304x304x172
Inception	48 + 40 + 20	156284	304x304x108
Inception	48 + 40 + 20	98172	304x304x108
Inception	64 + 48 + 20	118504	304x304x136
Conv (1x1)	32	4384	304x304x32
Conv (17x17)	32	295968	288x288x32
Conv (1x1)	64	2112	288x288x64
Conv (1x1)	64	4160	288x288x64
Convolutional (1x1)	1	65	288x288x1
Leaky ReLU		0	288x288x1

Table 2. Model 1. The inceptions are all a concatenation of 1x1, 3x3, and 5x5 Convs.

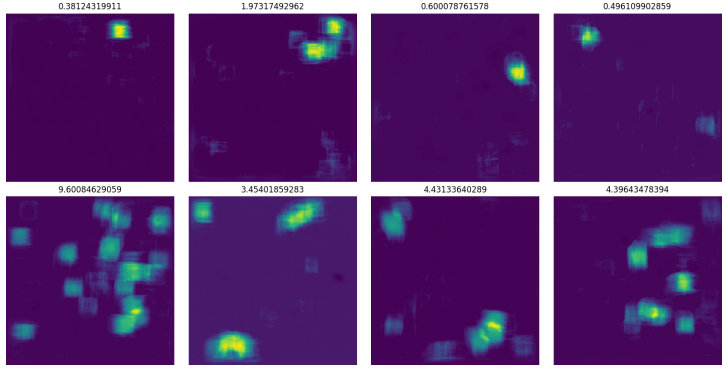

Fig. 5. Predicted images from model 1. These, and the following similar images, are the results of prediction on the images in figure 3, so they can be compared with each other and with figure 4.

– Rounded-count MSE: 7.710

Model 3: Batch normalization on output Model 3 is similar to model 1, except for the fact that the final layer ends with a batch normalization layer after the activation function. The resulting MSE is slightly better than the MSE of model 1. It was quite unexpected that the MSE would be lower, but perhaps the change aided the flow of the gradient backwards, even though it is on the very last layer. Still, the difference was low enough that it is also possible that it was a fluke.

Type	Filters	Params	Output size
Conv (3x3)	64	1792	318x318x64
Inception	24 + 32 + 16	45640	318x318x72
Inception	24 + 32 + 16	51336	318x318x72
Conv (1x1)	32	2336	318x318x32
Conv (17x17)	32	230432	304x304x32
Inception	112 + 40 + 20	31276	304x304x172
Inception	48 + 60 + 30	230274	304x304x138
Inception	48 + 60 + 30	184782	304x304x138
Inception	64 + 72 + 32	208824	304x304x168
Conv (1x1)	32	5408	304x304x32
Conv (17x17)	32	295968	288x288x32
Conv (1x1)	64	2112	288x288x64
Conv (1x1)	64	4160	288x288x64
Convolutional (1x1)	1	65	288x288x1
Leaky ReLU		0	288x288x1

Table 3. Model 2: more filters. The inceptions are all a concatenation of 1x1, 3x3, and 5x5 Convs.

– MSE: 7.563
– Rounded-count MSE: 7.626

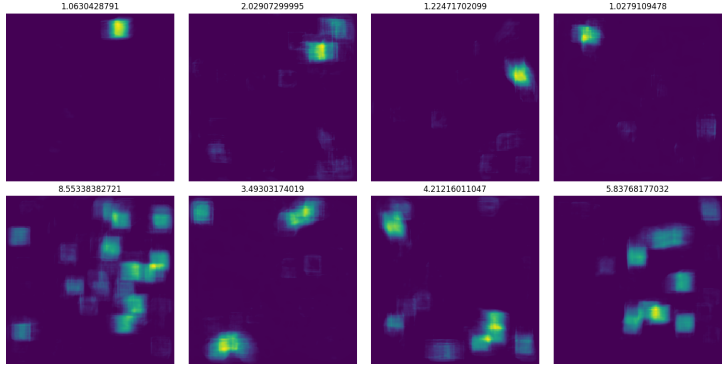

Fig. 6. Predicted images from model 3

Model 4: More negative images Model 4 is the same as model 3, but uses 17% negative images instead of 5%, as suggested in [12]. By adding more negative images we got a somewhat better MSE. It is quite interesting that the results were improved by using the found optimal percentage from [12], as the domains are quite different.

- MSE: 7.081
- Rounded-count MSE: 7.162

Model 5: Batch normalization before activation Model 5 has the same network structure as model 1, but swaps the order of the batch normalization layers. For whatever reason, that lead it to performing significantly worse. We expected that batch normalization after the activation function would have a slight edge, as per [15], but not this large. It may be due to the fact that much of our target images are zero, and the ReLU before the batch normalization removes some negative activations that would otherwise affect the statistics.

- MSE: 11.675
- Rounded-count MSE: 11.778

Model 6: MAE This model is simply model 1, but as loss function the mean average error is used instead of MSE. The results are awful, and while the output is very slightly different around sea lions, most of the image is close to zero. As we had occasionally noticed in early testing, the MAE is not penalized sufficiently from just guessing the most common value, and so it often converges to guessing zero everywhere. In other words, the regularization from MAE is too strong for our domain. It is perhaps possible to make MAE work with significantly different hyperparameters, but it seems that MSE is just better.

- MSE: 42.712
- Rounded-count MSE: 42.702

Model 7: Bigger target boxes Model 7 was trained with target boxes of 48x48 pixels. The idea here was to be able to use larger target boxes, in order to better cover the sea lions. However, with the same general idea of using two large convolution layers surrounded by multiple inception layers, the resulting convolution layers would be too large to fit in memory. There are two ways of surpassing this problem: 1) using fewer filters in the convolution layers, or 2) using more than two convolution layers, but smaller. Option 1 meant the amount of filters for the convolution layers had to be less than 8, which resulted in larger errors than a mean-of-training-data guess. Going on to option 2, by splitting the two large convolutions to three smaller ones, we could use the same amount of filters with a slightly smaller batch size and still fit the model in memory. The results after training showed potential, but were still worse than some of the other models.

- MSE: 10.975
- Rounded-count MSE: 11.063

Type	Filters	Params	Output size
Conv (3x3)	64	1792	350x350x64
Inception	24+20+10	29110	350x350x54
Inception	24+20+10	24570	350x350x32
Conv (1x1)	32	1760	350x350x32
Conv (15x15)	32	230432	336x336x32
Inception	112+40+20	31276	336x336x172
Inception	48+40+20	156284	336x336x108
Conv (1x1)	32	3488	336x336x108
Conv (17x17)	32	295968	320x320x32
Inception	48+40+20	29164	320x320x108
Inception	64+48+24	112204	320x320x136
Conv (1x1)	32	4384	320x320x32
Conv (17x17)	32	295968	304x304x32
Conv (1x1)	64	2112	304x304x64
Conv (1x1)	64	4160	304x304x64
Convolutional (1x1)	64	65	304x304x1
Leaky ReLU		0	304x304x1

Table 4. Model 7: bigger target boxes. The inceptions are all a concatenation of 1x1, 3x3, and 5x5 Convs.

4.3 Discussion/Conclusions

All in all, it seems to be a very promising approach for solving the counting problem. Even with the different shapes of sea lions that are in the images, the predictions end up like a "heat map" of them. The count-MSEs are somewhat less impressive, but they are at least significantly better than a mean guess. They are also likely to improve with more optimal parameters and longer training times.

It was interesting to see how the accuracy of the model would change with different network structures in the experiments. However, it's hard to draw any definite conclusions from these results as the same hyperparameters were used for all experiments which may not be suitable for the changed network structures and their respective error surfaces.

As mentioned in the approach, the target images are not always perfect, since dots are occasionally missed. About 9% of the sea lions lack targets due to this. This could affect our final test MSE since it would sometimes learn to not put a target where there is a sea lion. In addition, the test images may suffer from the same missing targets, and our net might therefore over-estimate. It was however quite interesting to see how robust a convolutional neural network can be to noisy data.

Because of time and resource constraints, there are a few possible improvements we could not try out. Firstly, it would be interesting to actually count the occurrence of each class of sea lion, this could perhaps allow the network to be more certain of its predictions, especially since the sea lions vary so much in size. Using this network approach one could possibly have one channel per image

class as the final output, and target image. Secondly, allocating more resources, to allow for more complex models, and bigger receptive fields could probably be beneficial for this quite complex problem. Finally, since we miss about 9% of the sea lions in our preprocessing step, it would probably be useful to preprocess in a more robust way.

References

1. NOAA: Noaa fisheries steller sea lion population count. https://www.kaggle.com/c/noaa-fisheries-steller-sea-lion-population-count (2017)
2. Cohen, J.P., Lo, H.Z., Bengio, Y.: Count-ception: Counting by fully convolutional redundant counting. arXiv preprint arXiv:1703.08710 (2017)
3. Hao Jiang, S.W.: Object detection and counting with low quality videos. (2016)
4. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR **abs/1506.02640** (2015)
5. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. CoRR **abs/1512.02325** (2015)
6. Xie, W., Noble, J.A., Zisserman, A.: Microscopy cell counting with fully convolutional regression networks. In: MICCAI 1st Workshop on Deep Learning in Medical Image Analysis. (2015)
7. Yella, S., Dougherty, M., et al.: Image processing technique to count the number of logs in a timber truck. In: IASTED conference on Signal and Image processing, Dallas, Texas, USA, 14-16 december, 2011, ACTA Press (2011)
8. Haralick, R.M., Shapiro, L.G.: Image segmentation techniques. Computer vision, graphics, and image processing **29**(1) (1985) 100–132
9. Lempitsky, V., Zisserman, A.: Learning to count objects in images. In: Advances in Neural Information Processing Systems. (2010) 1324–1332
10. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 1–9
11. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 2818–2826
12. Cernazanu-Glavan, C., Holban, S.: Determining the optimal percent of negative examples used in training the multilayer perceptron neural networks. In: Proceedings of the 10th WSEAS International Conference on NEURAL NETWORKS (NN'09), Prague. (2009)
13. Luo, W., Li, Y., Urtasun, R., Zemel, R.: Understanding the effective receptive field in deep convolutional neural networks. In: Advances in Neural Information Processing Systems. (2016) 4898–4906
14. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015)
15. Mishkin, D., Sergievskiy, N., Matas, J.: Systematic evaluation of CNN advances on the ImageNet. ArXiv e-prints (June 2016)
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014)
17. Amazon: Amazon web services. https://aws.amazon.com/
18. Nvidia: Nvidia tesla k80. http://www.nvidia.com/object/tesla-k80.html
19. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
20. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org.