ADVANCING THE RATE-DISTORTION-COMPUTATION FRONTIER FOR NEURAL IMAGE COMPRESSION

David Minnen & Nick Johnston

Google Research, Mountain View, CA 94043, USA

ABSTRACT

The rate-distortion performance of neural image compression models has exceeded the state-of-the-art for non-learned codecs, but neural codecs are still far from widespread deployment and adoption. The largest obstacle is having efficient models that are feasible on a wide variety of consumer hardware. Comparative research and evaluation is difficult due to the lack of standard benchmarking platforms and due to variations in hardware architectures and test environments. Through our rate-distortion-computation (RDC) study we demonstrate that neither floating-point operations (FLOPs) nor runtime are sufficient on their own to accurately rank neural compression methods. We also explore the RDC frontier, which leads to a family of model architectures with the best empirical trade-off between computational requirements and RD performance. Finally, we identify a novel neural compression architecture that yields state-of-the-art RD performance with rate savings of 23.1% over BPG (7.0% over VTM and 3.0% over ELIC) without requiring significantly more FLOPs than other learning-based codecs.

***Index Terms*—** image compression, neural networks, FLOPs, runtime

1. INTRODUCTION

The best neural image compression models are able to outperform leading non-learned image codecs in rate-distortion (RD) performance. However, neural image compression typically requires significantly more decode time and powerful, specialized hardware to achieve feasible runtimes for real-world applications. So far, no standard for providing runtime characteristics in neural image compression research has been adopted. While some studies report FLOPs, others report runtime on a particular device. We will demonstrate that neither of these alone is sufficient to meaningfully compare with other research works.

In this paper, we first show the results of a large architecture and parameter sweep over common neural image compression architectures. Next, we show how decoder runtime and decoder FLOPs (shortened to simply *runtime* and *FLOPs* below) are correlated across different model architectures and hardware platforms. Then, using the architecture study, we highlight a family of models that empirically have the best trade-off between FLOPs and RD performance. This family includes a new state-of-the-art model compared to BPG (HEVC Still Picture format), VTM (the VVC test model), and previous neural codecs. Finally, we make recommendations on how researchers should report the runtime characteristics of their models to more easily understand the trade-offs between compute, runtime, and RD performance.

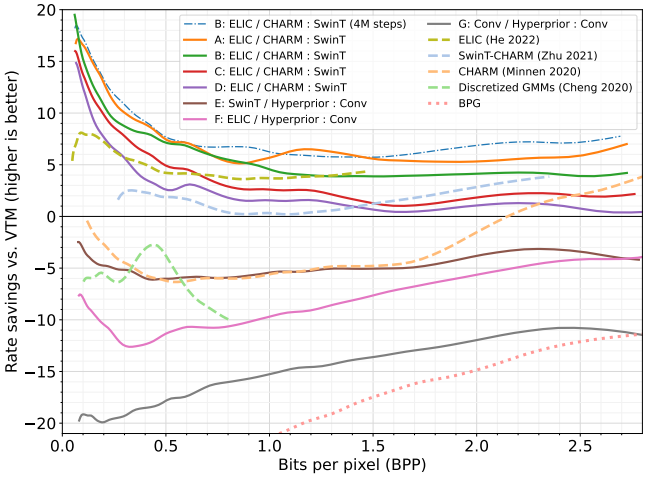

Fig. 1: Our model ("B") trained for 4M steps achieves state-of-the-art RD performance for MSE-optimized compression (see Fig. 2 for a FLOPs comparison). This graph shows rate savings relative to VTM v13.0 (higher is better) averaged over the Kodak dataset [1].

2. PRIOR WORK

Our architecture study builds on several previous model innovations: a hyperprior for entropy modeling [2], channel-wise autoregressive models (CHARM) [3], ELIC's use of convolutional layers, residual blocks, and attention blocks for transforms [4], and the Swin Transformer (SwinT) applied to compression models [5, 6, 7]. Johnston et al. produced a survey across different rates and loss functions, but focused on a single architecture [8]. Yang et al. [9] and Wu et al. [10] showed increased RD performance using instance-adaptive methods

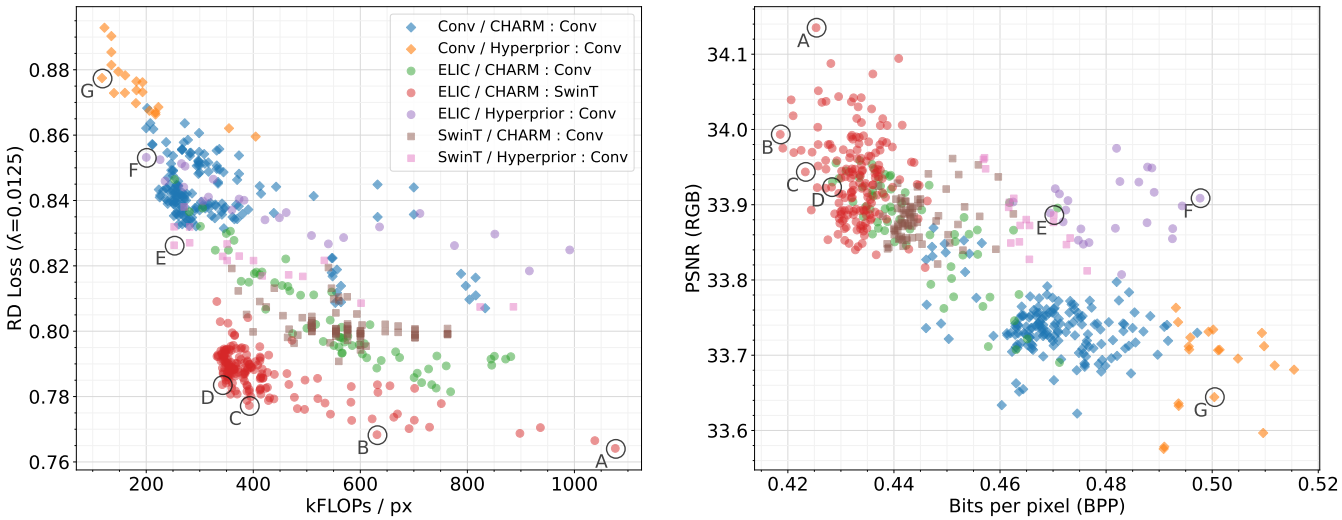

Fig. 2: The left plot shows model quality (RD loss) vs. complexity (kFLOPs per pixel) for 571 architecture variations. The best models compress well (smaller RD loss) and require fewer FLOPs. The plot on the right shows the same data but visualizes reconstruction quality (PSNR) vs. bit rate (BPP). Each color represents an architecture family labeled according to the modeling choice for the three main components: ⟨ analysis & synthesis transforms⟩ / ⟨ entropy model⟩ : ⟨ hyperprior transforms⟩.

that significantly increase image encoding time but do not slow down decoding. Le et al. demonstrates the importance of research and engineering co-design to optimize the runtime of end-to-end neural compression systems [11]. And Deghani et al. demonstrate discrepancies between FLOPs, runtime, and parameter counts in computer vision models and highlight the importance of accurate reporting of multiple metrics [12].

Previous research on neural image compression includes work that boosts RD performance without considering computational implications, *e.g.*, Cheng et al. used Gaussian mixture models for better entropy models [13], and several papers explored spatial autoregressive (AR) context models which can be very slow in practice since they do not effectively utilize massively parallel hardware [14, 15, 16]. Other research puts more emphasis on efficiency. The channel-wise autoregressive model (CHARM) was developed to improve parallelization and reduce decode time compared to spatial AR context models [3]. He et al. introduced ELIC, which improved CHARM by using uneven channel-wise groups and a spatial checkerboard decomposition [4]. And a different line of research explored vector quantization (VQ) for the latent representation [17, 18] since it typically leads to faster entropy models than scalar quantization.

3. ARCHITECTURE STUDY

We use L2 loss for distortion and a fixed RD trade-off of $\lambda = 0.0125$ which leads to an average of 0.46 bits per pixel on the Kodak image set used for evaluation [1] (see Fig. 2). We trained different combinations of analysis and synthesis transforms (stacked convolutions [2], ELIC-style [4], and SwinT [5, 6, 7]), entropy models (hyperprior [2] and CHARM [3]), and hyperprior transforms (stacked convolutions and SwinT) along with parameter sweeps across channel depth and number of layers. All models were trained using TensorFlow for 2M steps with a batch size of 32 (or 16 when memory constraints required it).

Fig. 2 (*left*) shows the results of our architecture sweeps and highlights seven models along the RDC frontier (labeled A–G, see Fig. 1 and Table 1 for details). Although we train with a single λ to target a fixed RD trade-off, there is some variation in the final bit rates and reconstruction quality levels as shown in Fig. 2 (*right*). Several interesting trends emerge: models with stacked convolutions (labeled as *Conv*) generally lead to the lowest compute but worst RD loss. This is not surprising since there are only four deconvolution layers in the synthesis transform for these models. Moving from a basic hyperprior to CHARM for the entropy model increases both the complexity and quality of the model (*e.g.*, compare orange vs. blue diamonds, purple vs. green dots, and pink vs. brown squares). Updating the transforms from stacked convolutions to ELIC makes the models more complex but also provides a significant gain in compression quality (compare blue diamonds to green dots). Comparing ELIC to SwinT for the transforms (green dots vs. brown squares) is more complex since relative RD performance flips for low vs. high FLOP counts. Finally, we see that using SwinT in the hyperprior leads to a more powerful entropy model with better RD performance across a wide complexity range (red dots).

Many of the interesting points on the frontier correspond to models that use ELIC transforms, a CHARM entropy model, and SwinT hyper-transforms. This architecture pro-

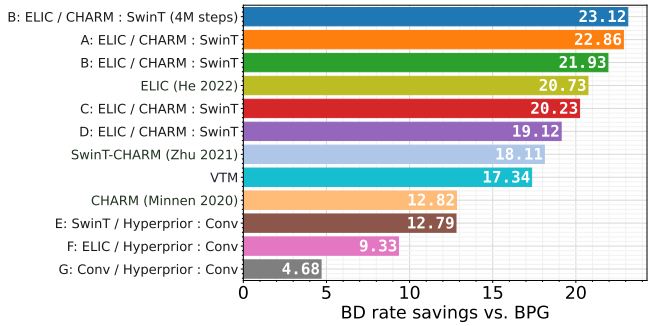

Fig. 3: Bjøntegaard Delta (BD) chart [19] showing average rate savings compared to BPG [20]. All BD rates were calculated over the largest quality range covered by all models (31.6 dB – 40.8 dB). For a more fine-grain comparison, Fig. 1 shows how rate savings varies across bit rates.

vides a strong baseline for further research into high-quality neural image compression but is at least 4x slower than smaller models that use simpler stacked convolutions and a basic hyperprior. To better evaluate the frontier models, relative rate savings are shown in Fig. 1 & 3, and runtime information is provided in Fig. 4 and Table 1.

4. RELATIONSHIP OF FLOPS TO RUNTIME

The FLOPs used by a neural model is not always proportional to runtime [12]. Several factors come into play here including the computational architecture, hardware and software used, and even the environment. The computational architecture, or how the FLOPs are arranged, is one factor that can affect runtime parallelism. For example, we can have one largely parallelized matrix multiplication that may complete in one time unit on a GPU, or we could have five very small matrix multiplications that cannot be fused together and therefore need to be computed serially. In this simple example, the one large operation might have more FLOPs yet still run more quickly than the five smaller operations.

Whether it's a TPU, GPU, FPGA or CPU, the design of each hardware platform can be optimized for certain operations or network architectures. Hardware availability, end-of-life, machine learning libraries, CUDA drivers, and even the operating system can affect the availability and performance of benchmarking. To provide more detailed information for our architecture study, we explored runtimes on a Nvidia A100 GPU [21] and a Google TPU v4 [22] for models on the empirical RDC frontier.

In Fig. 4 we see our models plotted with kFLOPs per pixel on the x-axis and decode time on an A100 GPU on the y-axis averaged over Kodak (*left*) and a 16 image CLIC subset (*center*). We provide timing data without entropy coding since optimizing the entropy coder is not the focus of this paper. Our unoptimized entropy coder leads to a relatively constant overhead across models and thus masks runtime differences due to changes in the neural architecture. If FLOPs is a good indicator of runtime, there should be a strong linear correlation in these scatterplots. While we do see linear correlation for some architectures, others show a very different relationship, *e.g.*, the `Conv/CHARM:Conv` (blue diamonds) and `Conv/Hyperprior:Conv` models (orange diamonds) have a very strong grouping around 20 ms. This is likely due to the A100 being under utilized. Thus, increasing the compute (by more than 6x in our study) added FLOPs in a way that was parallelizable with existing operations and thus did not have a linear effect on runtime. In general, the models with a CHARM entropy model show that you can have small changes in the model's FLOPs but a large multiplier on runtime. The resolution of the images across the two datasets also did not change these trends. Thus, while FLOP counts are platform agnostic, they are not sufficient for comparing relative runtime across model architectures.

Finally, we wanted to compare the relative performance between the A100 and TPU v4 hardware platforms. In Fig. 4 (*right*), we rank the runtime of all of our models on an A100 GPU and on a TPU v4. We see that across accelerator platforms it is not guaranteed to maintain the same set of ranking for different models. None of the model architecture families we explored were immune to this device-dependent variation.

While our study shows that comparisons across architectures and across devices may lead to false equivalences in neural compression research, it is important to keep the goal in mind. Runtime on a particular device can be useful for determining which models are "faster", though the claim of speed may not hold outside of that device and environment. For researchers more interested in complexity, FLOPs is an indicator within a particular architecture but can mask actual complexity when comparing across different architectures.

In the past, CLIC [23] ran many methods in the same environment, but that environment still changed each year. A global, year-round leaderboard, like ImageNet [24] and MS-COCO [25], would allow compression researchers to consistently compare progress over time. In the absence of a persistent leaderboard for measuring RD and runtime data, both FLOPs and runtime on device should be reported along with a more detailed discussion on how those FLOPs are distributed when they are compared across different architectures.

5. IMPROVED RDC MODELS

The architecture that gave the best RD results with moderate complexity ("B" in Table 1) used ELIC-style analysis and synthesis transforms [4] with a CHARM-based entropy model [3]. We found that two changes to the published ELIC transforms improved the RDC trade-off in our model: first, we reduced the number of residual blocks from three to two, and second we changed the channel depths in the analysis transform from [192, 192, 192] to [128, 256, 256] (with an

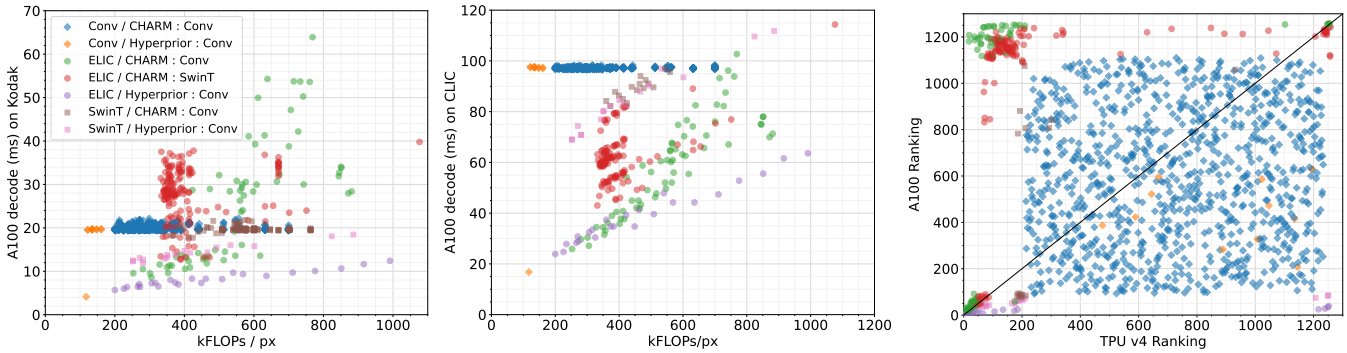

Fig. 4: Decode time vs. FLOPs (averaged over Kodak (*left*) and over sixteen CLIC images (*center*)) shows a mostly linear relationship for models using a simple hyperprior [2] (purple dots, pink squares, and orange diamonds). This trend breaks down for CHARM-based entropy models [3] where we see virtually no speed-up despite a 50% reduction in FLOPS (blue diamonds) and see wide runtime variance despite consistent FLOPs (red circles). We rank our methods fastest to slowest and compare rankings between an A100 GPU and a TPU v4 (*right*). Models below the diagonal line run relatively better on an A100, and models above the line run relatively better on a TPU v4. This makes ranking runtimes difficult, as it is very unlikely one can reproduce the relative ordering of methods on the particular hardware available to that researcher. This plot also reinforces the importance of hardware-software co-design if one wants to optimize directly for a particular device's runtime.

Model	Transforms (ana/syn)	Entropy Model	Hyperprior Transforms	Parameters ↓ (millions)	kFLOPS/px ↓ (decode-only)	Decode MP/s (Kodak) ↑		Decode MP/s (CLIC) ↑		Rate Savings ↑ vs. BPG
						V100	A100	V100	A100	
A	ELIC	CHARM	SwinT	38.0	1076.4	7.4	9.9	10.1	27.5	22.86
B	ELIC	CHARM	SwinT	31.6	631.1	8.8	10.7	14.8	35.3	21.93
C	ELIC	CHARM	SwinT	32.2	392.5	11.0	10.8	18.2	40.1	20.23
D	ELIC	CHARM	SwinT	22.4	342.1	11.7	11.1	19.6	43.4	19.12
E	SwinT	Hyperprior	Conv	17.3	252.1	20.5	31.5	23.9	45.5	12.79
F	ELIC	Hyperprior	Conv	19.8	200.1	41.0	69.0	51.7	131.6	9.33
G	Conv	Hyperprior	Conv	12.8	117.4	43.7	95.9	44.3	187.2	4.68

Table 1: Each row represents a model on the RDC frontier (see Fig. 2) listed in descending order of complexity. Columns 2–4 describe the model architecture, while the remaining columns list the number of parameters, FLOPs per pixel, decode speed in megapixels per second, and rate savings over BPG. Each Kodak image has $512 \times 768 = 393{,}216$ pixels, and the runtime numbers for the CLIC 2021 Professional dataset are averaged over 16 images each with $2048 \times 1365 = 2{,}795{,}520$ pixels.

equivalent change for the synthesis transform).

We also modified CHARM in three key ways. First, we found that moving from stacked convolutions to SwinT layers in the hyperprior improved RD performance. Second, we changed how the latent residual prediction (LRP) values were merged with decoded latents by switching from addition to concatenation followed by a 1×1 convolutional layer. And third, following Zhu et al. (see Fig. 12 in Appendix A.1 of [5]), we transformed the output of the hyper-synthesis transform into N smaller tensors using convolutional layers, where N is the number of channel-wise AR steps (following [3, 5], $N = 10$ for all of our models). Although this adds N extra layers to the decoding network, these layers can run in parallel, and the total computation is reduced due to using smaller tensors when predicting entropy parameters and LRP values.

6. CONCLUSION

Ranking the performance of neural image compression models is difficult due to the inversions demonstrated by our runtime vs. FLOPs plots and by the variability in runtime across platforms. We recommend that both FLOPs and runtime be reported, and ideally open-source implementations are also provided to facilitate comparisons by other researchers. We will have our best model open-sourced at `https://github.com/tensorflow/compression/tree/master/models/rdc`.

The result of our study is a model with state-of-the-art RD performance that falls on the RDC frontier of our architecture exploration and which outperforms BPG, VTM, ELIC and SwinT-CHARM baselines. We are optimistic that the compression community can create more efficient and effective models, evaluation methods, and hardware to make neural im-

age compression feasible for widespread deployment.

7. REFERENCES

[1] Eastman Kodak, "Kodak lossless true color image suite (PhotoCD PCD0992)," .

[2] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston, "Variational image compression with a scale hyperprior," *6th Int. Conf. on Learning Representations*, 2018.

[3] David Minnen and Saurabh Singh, "Channel-wise autoregressive entropy models for learned image compression," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 3339–3343.

[4] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang, "Elic: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5718–5727.

[5] Y. Zhu, Y. Yang, and T. Cohen, "Transformer-based transform coding," in *International Conference on Learning Representations*, 2022.

[6] Renjie Zou, Chunfeng Song, and Zhaoxiang Zhang, "The devil is in the details: Window-based attention for image compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17492–17501.

[7] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021.

[8] N. Johnston, E. Eban, A. Gordon, and J. Ballé, "Computationally efficient neural image compression," 2019.

[9] Yibo Yang, Robert Bamler, and Stephan Mandt, "Improving inference for neural image compression," *Advances in Neural Information Processing Systems*, vol. 33, pp. 573–584, 2020.

[10] Yuyang Wu, Zhiyang Qi, Huiming Zheng, Lvfang Tao, and Wei Gao, "Deep image compression with latent optimization and piece-wise quantization approximation," in *Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 1926–1930.

[11] H. Le, L. Zhang, A. Said, G. Sautiere, Y. Yang, P. Shrestha, F. Yin, R. Pourreza, and A. Wiggers, "Mobilecodec: neural inter-frame video compression on mobile devices," in *Proceedings of the 13th ACM Multimedia Systems Conference*, 2022, pp. 324–330.

[12] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay, "The efficiency misnomer," 2021.

[13] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7939–7948.

[14] Jan P. Klopp, Yu-Chiang Frank Wang, Shao-Yi Chien, and Liang-Gee Chen, "Learning a code-space predictor by exploiting intra-image-dependencies," in *British Machine Vision Conf.*, 2018.

[15] Jooyoung Lee, Seunghyun Cho, and Seung-Kwon Beack, "Context-adaptive entropy model for end-to-end optimized image compression," in *Int. Conf. on Learning Representations (ICLR)*, 2019.

[16] David Minnen, Johannes Ballé, and George D Toderici, "Joint autoregressive and hierarchical priors for learned image compression," in *Advances in Neural Information Processing Systems*, 2018, pp. 10771–10780.

[17] Xiaotong Lu, Heng Wang, Weisheng Dong, Fangfang Wu, Zhonglong Zheng, and Guangming Shi, "Learning a deep vector quantization network for image compression," *IEEE Access*, vol. 7, pp. 118815–118825, 2019.

[18] Xiaosu Zhu, Jingkuan Song, Lianli Gao, Feng Zheng, and Heng Tao Shen, "Unified multivariate gaussian mixture for efficient neural image compression," 2022.

[19] Gisle Bjøntegaard, "Calculation of average PSNR differences between RD-curves," Doc. VCEG-M33, ITU-T SG16/Q6 VCEG, Austin, TX, USA, Apr. 2001.

[20] F. Bellard, "BPG image format," `bellard.org/bpg` Accessed: 2017-01-30.

[21] Nvidia, "Nvidia A100," Accessed: 2023-02-22 `https://www.nvidia.com/en-us/data-center/a100/`.

[22] Google Cloud, "System architectures tpu," `cloud.google.com/tpu/docs/system-architecture-tpu-vm`.

[23] George Toderici, Nick Johnston, Eirikur Agustsson, Fabian Mentzer, Johannes Ballé, Radu Timofte, Zeina Sinno, Andrey Norkin, Krishna Rapaka, Erfan Noury, Ross Cutler, Luca Versari, and Fabien Fabien Racapé, *CLIC 2022: Challenge on Learned Image Compression*, 2022, `http://compression.cc`.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár, "Microsoft COCO: Common objects in context," 2014.