5 Important Deep Learning Research Papers You Must Read In 2020

These papers provide a breadth of information about Deep Learning that is generally useful and interesting from a data science perspective.

Contents

- Human-level control through deep reinforcement learning
- DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images
- Semi-Supervised Learning with Ladder Network
- Fast Graph Representation Learning With Pytorch Geometric

- High-Fidelity Image GenerationWith Fewer Labels

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems[4,5], the former evidenced by a wealth of neural data revealing notable parallels between the phasic signals emitted by dopaminergic neurons and temporal difference reinforcement learning algorithms[3]. While reinforcement learning agents have achieved some successes in a variety of domains[6–8], their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. Here we use recent advances in training deep neural networks[9–11] to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. We tested this agent on the challenging domain of classic Atari 2600 games[12]. We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters. This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.

We set out to create a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks—a central goal of general artificial intelligence[13] that has eluded previous efforts[8,14,15]. To achieve this, we developed a novel agent, a deep Q-network (DQN), which is able to combine reinforcement learning with a class of artificial neural network[16] known as deep neural networks. Notably, recent advances in deep neural networks[9–11], in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data. We use one particularly successful architecture, the deep convolutional network[17], which uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields—inspired by Hubel and Wiesel's seminal work on feedforward processing in early visual cortex[18]—thereby exploiting the local spatial correlations present in images, and building in robustness to natural transformations such as changes of viewpoint or scale.

We consider tasks in which the agent interacts with an environment through a sequence of observations, actions and rewards. The goal of the agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed experience replay[21–23] that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution (see below for details). Second, we used an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target.

While other stable methods exist for training neural networks in the reinforcement learning setting, such as neural fitted Q-iteration[24], these methods involve the repeated training of networks *de novo* on hundreds of iterations. Consequently, these methods, unlike our algorithm, are too inefficient to be used successfully with large neural networks. We parameterize an approximate value function $Q(s,a;\theta_i)$ using the deep convolutional neural network shown in Fig. 1, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i. To perform experience replay we store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step t in a data set $D_t = \{e_1, \ldots, e_t\}$. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s,a,r,s') \sim U(D)$, drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

in which γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters (θ_i) every C steps and are held fixed between individual updates (see Methods).

To evaluate our DQN agent, we took advantage of the Atari 2600 platform, which offers a diverse array of tasks ($n = 49$) designed to be

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK.
*These authors contributed equally to this work.

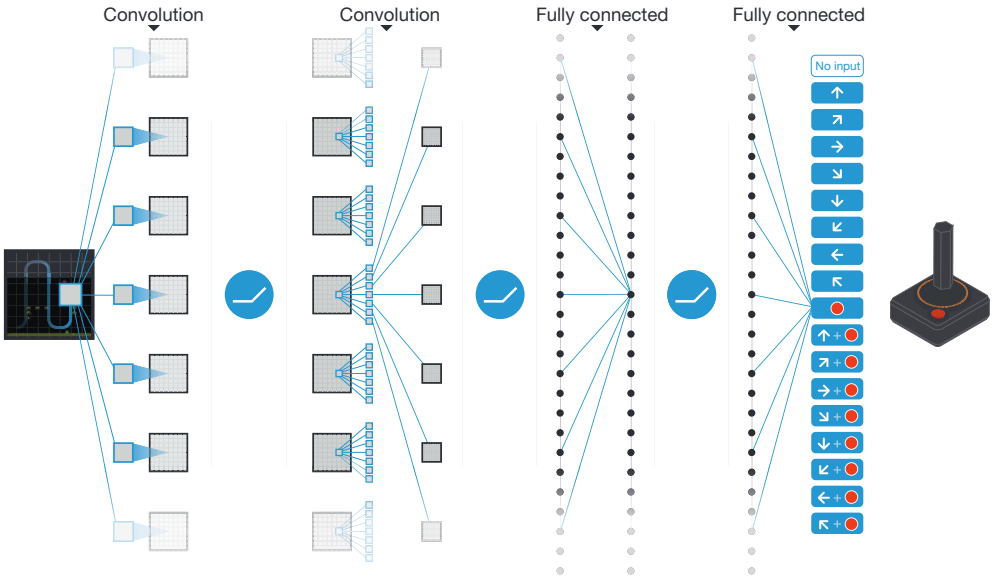


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

difficult and engaging for human players. We used the same network architecture, hyperparameter values (see Extended Data Table 1) and learning procedure throughout—taking high-dimensional data (210×160 colour video at 60 Hz) as input—to demonstrate that our approach robustly learns successful policies over a variety of games based solely on sensory inputs with only very minimal prior knowledge (that is, merely the input data were visual images, and the number of actions available in each game, but not their correspondences; see Methods). Notably, our method was able to train large neural networks using a reinforcement learning signal and stochastic gradient descent in a stable manner—illustrated by the temporal evolution of two indices of learning (the agent's average score-per-episode and average predicted Q-values; see Fig. 2 and Supplementary Discussion for details).

We compared DQN with the best performing methods from the reinforcement learning literature on the 49 games where results were available[12,15]. In addition to the learned agents, we also report scores for a professional human games tester playing under controlled conditions and a policy that selects actions uniformly at random (Extended Data Table 2 and Fig. 3, denoted by 100% (human) and 0% (random) on y axis; see Methods). Our DQN method outperforms the best existing reinforcement learning methods on 43 of the games without incorporating any of the additional prior knowledge about Atari 2600 games used by other approaches (for example, refs 12, 15). Furthermore, our DQN agent performed at a level that was comparable to that of a professional human games tester across the set of 49 games, achieving more than 75% of the human score on more than half of the games (29 games;

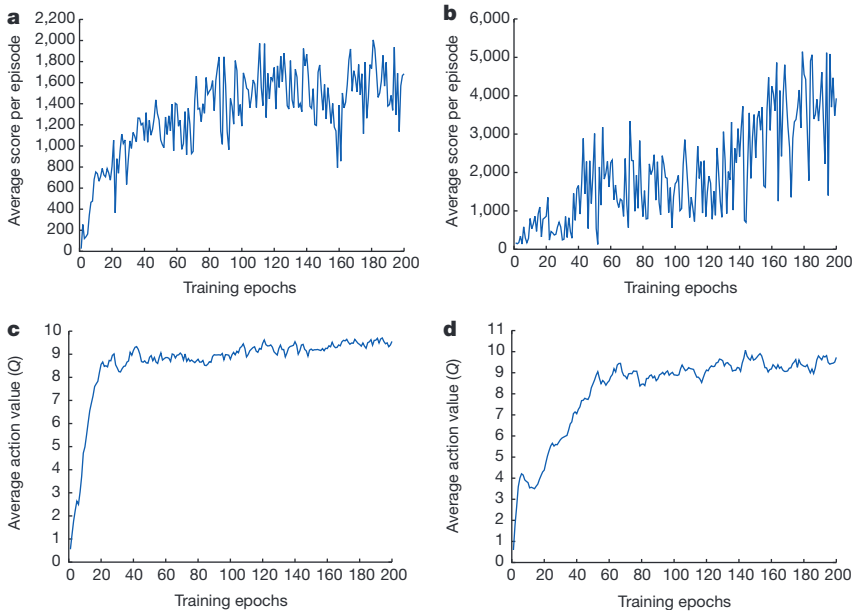


Figure 2 | Training curves tracking the agent's average score and average predicted action-value. **a**, Each point is the average score achieved per episode after the agent is run with ε-greedy policy ($\varepsilon = 0.05$) for 520 k frames on Space Invaders. **b**, Average score achieved per episode for Seaquest. **c**, Average predicted action-value on a held-out set of states on Space Invaders. Each point on the curve is the average of the action-value Q computed over the held-out set of states. Note that Q-values are scaled due to clipping of rewards (see Methods). **d**, Average predicted action-value on Seaquest. See Supplementary Discussion for details.

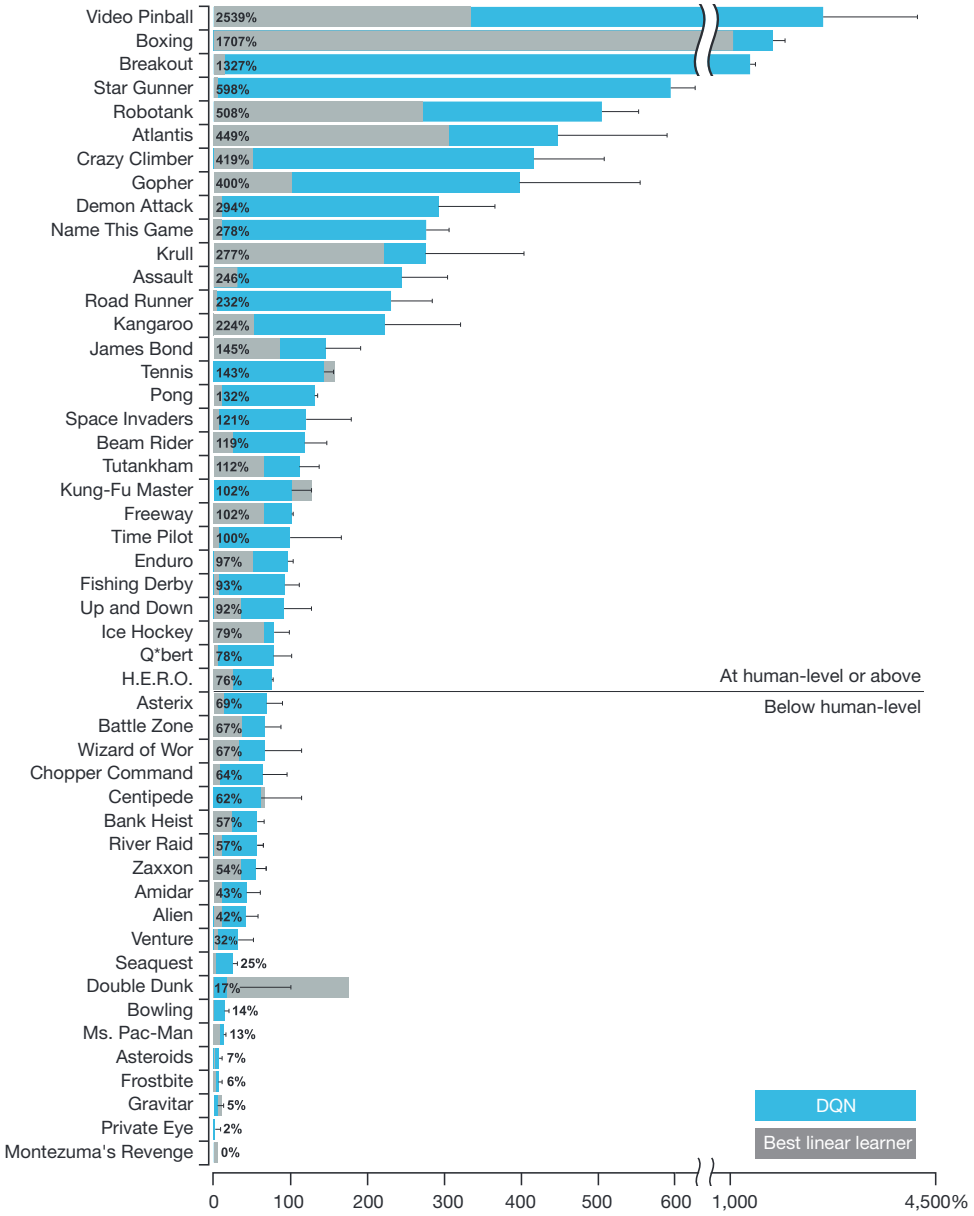


Figure 3 | Comparison of the DQN agent with the best reinforcement learning methods[15] in the literature. The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level) and random play (that is, 0% level). Note that the normalized performance of DQN, expressed as a percentage, is calculated as: 100 × (DQN score − random play score)/(human score − random play score). It can be seen that DQN outperforms competing methods (also see Extended Data Table 2) in almost all the games, and performs at a level that is broadly comparable with or superior to a professional human games tester (that is, operationalized as a level of 75% or above) in the majority of games. Audio output was disabled for both human players and agents. Error bars indicate s.d. across the 30 evaluation episodes, starting with different initial conditions.

see Fig. 3, Supplementary Discussion and Extended Data Table 2). In additional simulations (see Supplementary Discussion and Extended Data Tables 3 and 4), we demonstrate the importance of the individual core components of the DQN agent—the replay memory, separate target Q-network and deep convolutional network architecture—by disabling them and demonstrating the detrimental effects on performance.

We next examined the representations learned by DQN that underpinned the successful performance of the agent in the context of the game Space Invaders (see Supplementary Video 1 for a demonstration of the performance of DQN), by using a technique developed for the visualization of high-dimensional data called 't-SNE'[25] (Fig. 4). As expected, the t-SNE algorithm tends to map the DQN representation of perceptually similar states to nearby points. Interestingly, we also found instances in which the t-SNE algorithm generated similar embeddings for DQN representations of states that are close in terms of expected reward but perceptually dissimilar (Fig. 4, bottom right, top left and middle), consistent with the notion that the network is able to learn representations that support adaptive behaviour from high-dimensional sensory inputs. Furthermore, we also show that the representations learned by DQN are able to generalize to data generated from policies other than its own—in simulations where we presented as input to the network game states experienced during human and agent play, recorded the representations of the last hidden layer, and visualized the embeddings generated by the t-SNE algorithm (Extended Data Fig. 1 and Supplementary Discussion). Extended Data Fig. 2 provides an additional illustration of how the representations learned by DQN allow it to accurately predict state and action values.

It is worth noting that the games in which DQN excels are extremely varied in their nature, from side-scrolling shooters (River Raid) to boxing games (Boxing) and three-dimensional car-racing games (Enduro).

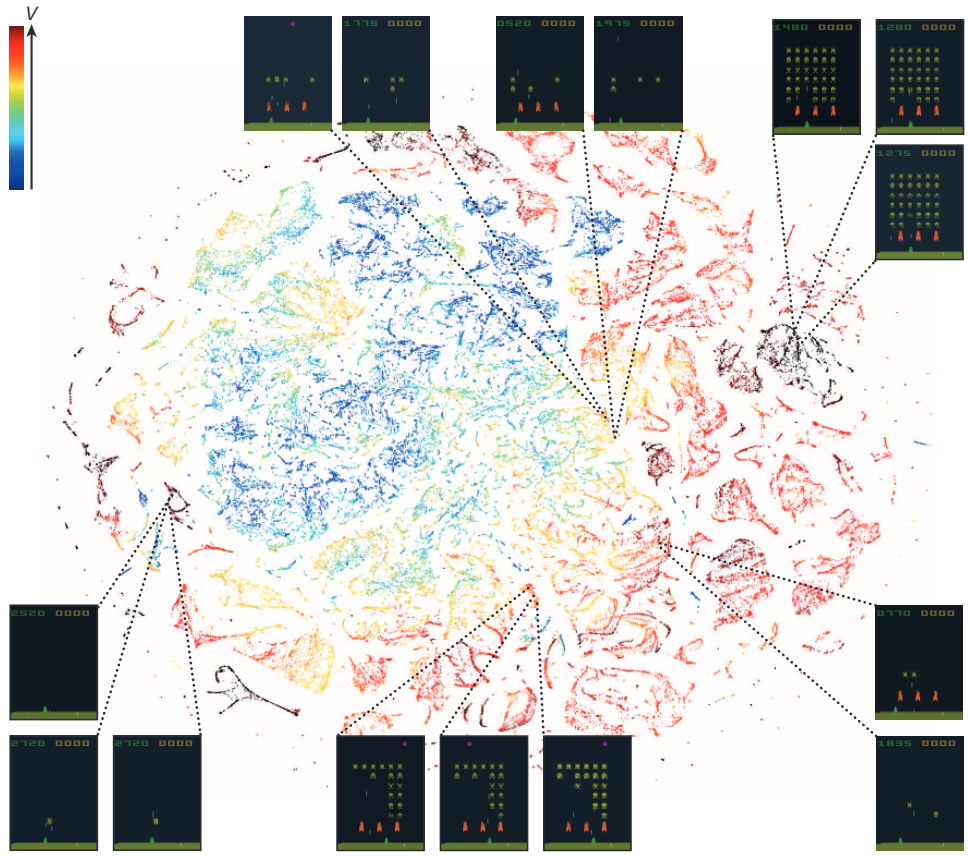


Figure 4 | Two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced while playing Space Invaders. The plot was generated by letting the DQN agent play for 2 h of real game time and running the t-SNE algorithm[25] on the last hidden layer representations assigned by DQN to each experienced game state. The points are coloured according to the state values (*V*, maximum expected reward of a state) predicted by DQN for the corresponding game states (ranging from dark red (highest *V*) to dark blue (lowest *V*)). The screenshots corresponding to a selected number of points are shown. The DQN agent predicts high state values for both full (top right screenshots) and nearly complete screens (bottom left screenshots) because it has learned that completing a screen leads to a new screen full of enemy ships. Partially completed screens (bottom screenshots) are assigned lower state values because less immediate reward is available. The screens shown on the bottom right and top left and middle are less perceptually similar than the other examples but are still mapped to nearby representations and similar values because the orange bunkers do not carry great significance near the end of a level. With permission from Square Enix Limited.

Indeed, in certain games DQN is able to discover a relatively long-term strategy (for example, Breakout: the agent learns the optimal strategy, which is to first dig a tunnel around the side of the wall allowing the ball to be sent around the back to destroy a large number of blocks; see Supplementary Video 2 for illustration of development of DQN's performance over the course of training). Nevertheless, games demanding more temporally extended planning strategies still constitute a major challenge for all existing agents including DQN (for example, Montezuma's Revenge).

In this work, we demonstrate that a single architecture can successfully learn control policies in a range of different environments with only very minimal prior knowledge, receiving only the pixels and the game score as inputs, and using the same algorithm, network architecture and hyperparameters on each game, privy only to the inputs a human player would have. In contrast to previous work[24,26], our approach incorporates 'end-to-end' reinforcement learning that uses reward to continuously shape representations within the convolutional network towards salient features of the environment that facilitate value estimation. This principle draws on neurobiological evidence that reward signals during perceptual learning may influence the characteristics of representations within primate visual cortex[27,28]. Notably, the successful integration of reinforcement learning with deep network architectures was critically dependent on our incorporation of a replay algorithm[21–23] involving the storage and representation of recently experienced transitions. Convergent evidence suggests that the hippocampus may support the physical realization of such a process in the mammalian brain, with the time-compressed reactivation of recently experienced trajectories during offline periods[21,22] (for example, waking rest) providing a putative mechanism by which value functions may be efficiently updated through interactions with the basal ganglia[22]. In the future, it will be important to explore the potential use of biasing the content of experience replay towards salient events, a phenomenon that characterizes empirically observed hippocampal replay[29], and relates to the notion of 'prioritized sweeping'[30] in reinforcement learning. Taken together, our work illustrates the power of harnessing state-of-the-art machine learning techniques with biologically inspired mechanisms to create agents that are capable of learning to master a diverse array of challenging tasks.

Online Content Methods, along with any additional Extended Data display items and Source Data, are available in the online version of the paper; references unique to these sections appear only in the online paper.

Received 10 July 2014; accepted 16 January 2015.

1. Sutton, R. & Barto, A. Reinforcement Learning: An Introduction (MIT Press, 1998).
2. Thorndike, E. L. Animal Intelligence: Experimental studies (Macmillan, 1911).
3. Schultz, W., Dayan, P. & Montague, P. R. A neural substrate of prediction and reward. Science **275,** 1593–1599 (1997).
4. Serre, T., Wolf, L. & Poggio, T. Object recognition with features inspired by visual cortex. Proc. IEEE. Comput. Soc. Conf. Comput. Vis. Pattern. Recognit. 994–1000 (2005).
5. Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybern. **36,** 193–202 (1980).

6. Tesauro, G. Temporal difference learning and TD-Gammon. Commun. ACM **38,** 58–68 (1995).
7. Riedmiller, M., Gabel, T., Hafner, R. & Lange, S. Reinforcement learning for robot soccer. Auton. Robots **27,** 55–73 (2009).
8. Diuk, C., Cohen, A. & Littman, M. L. An object-oriented representation for efficient reinforcement learning. Proc. Int. Conf. Mach. Learn. 240–247 (2008).
9. Bengio, Y. Learning deep architectures for AI. Foundations and Trends in Machine Learning **2,** 1–127 (2009).
10. Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. **25,** 1106–1114 (2012).
11. Hinton, G. E. & Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. Science **313,** 504–507 (2006).
12. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The arcade learning environment: An evaluation platform for general agents. J. Artif. Intell. Res. **47,** 253–279 (2013).
13. Legg, S. & Hutter, M. Universal Intelligence: a definition of machine intelligence. Minds Mach. **17,** 391–444 (2007).
14. Genesereth, M., Love, N. & Pell, B. General game playing: overview of the AAAI competition. AI Mag. **26,** 62–72 (2005).
15. Bellemare, M. G., Veness, J. & Bowling, M. Investigating contingency awareness using Atari 2600 games. Proc. Conf. AAAI. Artif. Intell. 864–871 (2012).
16. McClelland, J. L., Rumelhart, D. E. & Group, T. P. R. Parallel Distributed Processing: Explorations in the Microstructure of Cognition (MIT Press, 1986).
17. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. Proc. IEEE **86,** 2278–2324 (1998).
18. Hubel, D. H. & Wiesel, T. N. Shape and arrangement of columns in cat's striate cortex. J. Physiol. **165,** 559–568 (1963).
19. Watkins, C. J. & Dayan, P. Q-learning. Mach. Learn. **8,** 279–292 (1992).
20. Tsitsiklis, J. & Roy, B. V. An analysis of temporal-difference learning with function approximation. IEEE Trans. Automat. Contr. **42,** 674–690 (1997).
21. McClelland, J. L., McNaughton, B. L. & O'Reilly, R. C. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. Psychol. Rev. **102,** 419–457 (1995).
22. O'Neill, J., Pleydell-Bouverie, B., Dupret, D. & Csicsvari, J. Play it again: reactivation of waking experience and memory. Trends Neurosci. **33,** 220–229 (2010).
23. Lin, L.-J. Reinforcement learning for robots using neural networks. Technical Report, DTIC Document (1993).
24. Riedmiller, M. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. Mach. Learn.: ECML, **3720,** 317–328 (Springer, 2005).
25. Van der Maaten, L. J. P. & Hinton, G. E. Visualizing high-dimensional data using t-SNE. J. Mach. Learn. Res. **9,** 2579–2605 (2008).
26. Lange, S. & Riedmiller, M. Deep auto-encoder neural networks in reinforcement learning. Proc. Int. Jt. Conf. Neural. Netw. 1–8 (2010).
27. Law, C.-T. & Gold, J. I. Reinforcement learning can account for associative and perceptual learning on a visual decision task. Nature Neurosci. **12,** 655 (2009).
28. Sigala, N. & Logothetis, N. K. Visual categorization shapes feature selectivity in the primate temporal cortex. Nature **415,** 318–320 (2002).
29. Bendor, D. & Wilson, M. A. Biasing the content of hippocampal replay during sleep. Nature Neurosci. **15,** 1439–1444 (2012).
30. Moore, A. & Atkeson, C. Prioritized sweeping: reinforcement learning with less data and less real time. Mach. Learn. **13,** 103–130 (1993).

Supplementary Information is available in the online version of the paper.

Acknowledgements We thank G. Hinton, P. Dayan and M. Bowling for discussions, A. Cain and J. Keene for work on the visuals, K. Keller and P. Rogers for help with the visuals, G. Wayne for comments on an earlier version of the manuscript, and the rest of the DeepMind team for their support, ideas and encouragement.

Author Contributions V.M., K.K., D.S., J.V., M.G.B., M.R., A.G., D.W., S.L. and D.H. conceptualized the problem and the technical framework. V.M., K.K., A.A.R. and D.S. developed and tested the algorithms. J.V., S.P., C.B., A.A.R., M.G.B., I.A., A.K.F., G.O. and A.S. created the testing platform. K.K., H.K., S.L. and D.H. managed the project. K.K., D.K., D.H., V.M., D.S., A.G., A.A.R., J.V. and M.G.B. wrote the paper.

Author Information Reprints and permissions information is available at www.nature.com/reprints. The authors declare no competing financial interests. Readers are welcome to comment on the online version of the paper. Correspondence and requests for materials should be addressed to K.K. (korayk@google.com) or D.H. (demishassabis@google.com).

METHODS

Preprocessing. Working directly with raw Atari 2600 frames, which are 210 × 160 pixel images with a 128-colour palette, can be demanding in terms of computation and memory requirements. We apply a basic preprocessing step aimed at reducing the input dimensionality and dealing with some artefacts of the Atari 2600 emulator. First, to encode a single frame we take the maximum value for each pixel colour value over the frame being encoded and the previous frame. This was necessary to remove flickering that is present in games where some objects appear only in even frames while other objects appear only in odd frames, an artefact caused by the limited number of sprites Atari 2600 can display at once. Second, we then extract the Y channel, also known as luminance, from the RGB frame and rescale it to 84 × 84. The function ϕ from algorithm 1 described below applies this preprocessing to the m most recent frames and stacks them to produce the input to the Q-function, in which $m = 4$, although the algorithm is robust to different values of m (for example, 3 or 5).

Code availability. The source code can be accessed at https://sites.google.com/a/deepmind.com/dqn for non-commercial uses only.

Model architecture. There are several possible ways of parameterizing Q using a neural network. Because Q maps history–action pairs to scalar estimates of their Q-value, the history and the action have been used as inputs to the neural network by some previous approaches[24,26]. The main drawback of this type of architecture is that a separate forward pass is required to compute the Q-value of each action, resulting in a cost that scales linearly with the number of actions. We instead use an architecture in which there is a separate output unit for each possible action, and only the state representation is an input to the neural network. The outputs correspond to the predicted Q-values of the individual actions for the input state. The main advantage of this type of architecture is the ability to compute Q-values for all possible actions in a given state with only a single forward pass through the network.

The exact architecture, shown schematically in Fig. 1, is as follows. The input to the neural network consists of an 84 × 84 × 4 image produced by the preprocessing map ϕ. The first hidden layer convolves 32 filters of 8 × 8 with stride 4 with the input image and applies a rectifier nonlinearity[31,32]. The second hidden layer convolves 64 filters of 4 × 4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64 filters of 3 × 3 with stride 1 followed by a rectifier. The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied between 4 and 18 on the games we considered.

Training details. We performed experiments on 49 Atari 2600 games where results were available for all other comparable methods[12,15]. A different network was trained on each game: the same network architecture, learning algorithm and hyperparameter settings (see Extended Data Table 1) were used across all games, showing that our approach is robust enough to work on a variety of games while incorporating only minimal prior knowledge (see below). While we evaluated our agents on unmodified games, we made one change to the reward structure of the games during training only. As the scale of scores varies greatly from game to game, we clipped all positive rewards at 1 and all negative rewards at −1, leaving 0 rewards unchanged. Clipping the rewards in this manner limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games. At the same time, it could affect the performance of our agent since it cannot differentiate between rewards of different magnitude. For games where there is a life counter, the Atari 2600 emulator also sends the number of lives left in the game, which is then used to mark the end of an episode during training.

In these experiments, we used the RMSProp (see http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf) algorithm with minibatches of size 32. The behaviour policy during training was ε-greedy with ε annealed linearly from 1.0 to 0.1 over the first million frames, and fixed at 0.1 thereafter. We trained for a total of 50 million frames (that is, around 38 days of game experience in total) and used a replay memory of 1 million most recent frames.

Following previous approaches to playing Atari 2600 games, we also use a simple frame-skipping technique[15]. More precisely, the agent sees and selects actions on every kth frame instead of every frame, and its last action is repeated on skipped frames. Because running the emulator forward for one step requires much less computation than having the agent select an action, this technique allows the agent to play roughly k times more games without significantly increasing the runtime. We use $k = 4$ for all games.

The values of all the hyperparameters and optimization parameters were selected by performing an informal search on the games Pong, Breakout, Seaquest, Space Invaders and Beam Rider. We did not perform a systematic grid search owing to the high computational cost. These parameters were then held fixed across all other games. The values and descriptions of all hyperparameters are provided in Extended Data Table 1.

Our experimental setup amounts to using the following minimal prior knowledge: that the input data consisted of visual images (motivating our use of a convolutional deep network), the game-specific score (with no modification), number of actions, although not their correspondences (for example, specification of the up 'button') and the life count.

Evaluation procedure. The trained agents were evaluated by playing each game 30 times for up to 5 min each time with different initial random conditions ('no-op'; see Extended Data Table 1) and an ε-greedy policy with $\varepsilon = 0.05$. This procedure is adopted to minimize the possibility of overfitting during evaluation. The random agent served as a baseline comparison and chose a random action at 10 Hz which is every sixth frame, repeating its last action on intervening frames. 10 Hz is about the fastest that a human player can select the 'fire' button, and setting the random agent to this frequency avoids spurious baseline scores in a handful of the games. We did also assess the performance of a random agent that selected an action at 60 Hz (that is, every frame). This had a minimal effect: changing the normalized DQN performance by more than 5% in only six games (Boxing, Breakout, Crazy Climber, Demon Attack, Krull and Robotank), and in all these games DQN outperformed the expert human by a considerable margin.

The professional human tester used the same emulator engine as the agents, and played under controlled conditions. The human tester was not allowed to pause, save or reload games. As in the original Atari 2600 environment, the emulator was run at 60 Hz and the audio output was disabled: as such, the sensory input was equated between human player and agents. The human performance is the average reward achieved from around 20 episodes of each game lasting a maximum of 5 min each, following around 2 h of practice playing each game.

Algorithm. We consider tasks in which an agent interacts with an environment, in this case the Atari emulator, in a sequence of actions, observations and rewards. At each time-step the agent selects an action a_t from the set of legal game actions, $\mathcal{A} = \{1, \ldots, K\}$. The action is passed to the emulator and modifies its internal state and the game score. In general the environment may be stochastic. The emulator's internal state is not observed by the agent; instead the agent observes an image $x_t \in \mathbb{R}^d$ from the emulator, which is a vector of pixel values representing the current screen. In addition it receives a reward r_t representing the change in game score. Note that in general the game score may depend on the whole previous sequence of actions and observations; feedback about an action may only be received after many thousands of time-steps have elapsed.

Because the agent only observes the current screen, the task is partially observed[33] and many emulator states are perceptually aliased (that is, it is impossible to fully understand the current situation from only the current screen x_t). Therefore, sequences of actions and observations, $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$, are input to the algorithm, which then learns game strategies depending upon these sequences. All sequences in the emulator are assumed to terminate in a finite number of time-steps. This formalism gives rise to a large but finite Markov decision process (MDP) in which each sequence is a distinct state. As a result, we can apply standard reinforcement learning methods for MDPs, simply by using the complete sequence s_t as the state representation at time t.

The goal of the agent is to interact with the emulator by selecting actions in a way that maximizes future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step (γ was set to 0.99 throughout), and define the future discounted return at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, in which T is the time-step at which the game terminates. We define the optimal action-value function $Q^*(s,a)$ as the maximum expected return achievable by following any policy, after seeing some sequence s and then taking some action a, $Q^*(s,a) = \max_\pi \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi]$ in which π is a policy mapping sequences to actions (or distributions over actions).

The optimal action-value function obeys an important identity known as the Bellman equation. This is based on the following intuition: if the optimal value $Q^*(s',a')$ of the sequence s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' maximizing the expected value of $r + \gamma Q^*(s',a')$:

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

The basic idea behind many reinforcement learning algorithms is to estimate the action-value function by using the Bellman equation as an iterative update, $Q_{i+1}(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q_i(s',a') \mid s,a]$. Such value iteration algorithms converge to the optimal action-value function, $Q_i \to Q^*$ as $i \to \infty$. In practice, this basic approach is impractical, because the action-value function is estimated separately for each sequence, without any generalization. Instead, it is common to use a function approximator to estimate the action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$. In the reinforcement learning community this is typically a linear function approximator, but

sometimes a nonlinear function approximator is used instead, such as a neural network. We refer to a neural network function approximator with weights θ as a Q-network. A Q-network can be trained by adjusting the parameters θ_i at iteration i to reduce the mean-squared error in the Bellman equation, where the optimal target values $r+\gamma \max_{a'} Q^*(s',a')$ are substituted with approximate target values $y=r+\gamma \max_{a'} Q(s',a';\theta_i^-)$, using parameters θ_i^- from some previous iteration. This leads to a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i,

$$L_i(\theta_i)=\mathbb{E}_{s,a,r}\left[(\mathrm{E}_{s'}[y|s,a]-Q(s,a;\theta_i))^2\right]$$
$$=\mathbb{E}_{s,a,r,s'}\left[(y-Q(s,a;\theta_i))^2\right]+\mathrm{E}_{s,a,r}[\mathrm{V}_{s'}[y]]:$$

Note that the targets depend on the network weights; this is in contrast with the targets used for supervised learning, which are fixed before learning begins. At each stage of optimization, we hold the parameters from the previous iteration θ_i^- fixed when optimizing the ith loss function $L_i(\theta_i)$, resulting in a sequence of well-defined optimization problems. The final term is the variance of the targets, which does not depend on the parameters θ_i that we are currently optimizing, and may therefore be ignored. Differentiating the loss function with respect to the weights we arrive at the following gradient:

$$+_{\theta_i}L(\theta_i)\quad=\mathbb{E}_{s,a,r,s'}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-)-Q(s,a;\theta_i)\right)+_{\theta_i}Q(s,a;\theta_i)\right]:$$

Rather than computing the full expectations in the above gradient, it is often computationally expedient to optimize the loss function by stochastic gradient descent. The familiar Q-learning algorithm[19] can be recovered in this framework by updating the weights after every time step, replacing the expectations using single samples, and setting $\theta_i^-=\theta_{i-1}$.

Note that this algorithm is model-free: it solves the reinforcement learning task directly using samples from the emulator, without explicitly estimating the reward and transition dynamics $P(r,s'|s,a)$. It is also off-policy: it learns about the greedy policy $a=\mathrm{argmax}_{a'} Q(s,a';\theta)$, while following a behaviour distribution that ensures adequate exploration of the state space. In practice, the behaviour distribution is often selected by an ε-greedy policy that follows the greedy policy with probability $1-\varepsilon$ and selects a random action with probability ε.

Training algorithm for deep Q-networks. The full algorithm for training deep Q-networks is presented in Algorithm 1. The agent selects and executes actions according to an ε-greedy policy based on Q. Because using histories of arbitrary length as inputs to a neural network can be difficult, our Q-function instead works on a fixed length representation of histories produced by the function ϕ described above. The algorithm modifies standard online Q-learning in two ways to make it suitable for training large neural networks without diverging.

First, we use a technique known as experience replay[23] in which we store the agent's experiences at each time-step, $e_t=(s_t,a_t,r_t,s_{t+1})$, in a data set $D_t=\{e_1,\ldots,e_t\}$, pooled over many episodes (where the end of an episode occurs when a terminal state is reached) into a replay memory. During the inner loop of the algorithm, we apply Q-learning updates, or minibatch updates, to samples of experience, $(s,a,r,s')\sim U(D)$, drawn at random from the pool of stored samples. This approach has several advantages over standard online Q-learning. First, each step of experience is potentially used in many weight updates, which allows for greater data efficiency. Second, learning directly from consecutive samples is inefficient, owing to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates. Third, when learning on-policy the current parameters determine the next data sample that the parameters are trained on. For example, if the maximizing action is to move left then the training samples will be dominated by samples from the left-hand side; if the maximizing action then switches to the right then the training distribution will also switch. It is easy to see how unwanted feedback loops may arise and the parameters could get stuck in a poor local minimum, or even diverge catastrophically[20]. By using experience replay the behaviour distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters. Note that when learning by experience replay, it is necessary to learn off-policy (because our current parameters are different to those used to generate the sample), which motivates the choice of Q-learning.

In practice, our algorithm only stores the last N experience tuples in the replay memory, and samples uniformly at random from D when performing updates. This approach is in some respects limited because the memory buffer does not differentiate important transitions and always overwrites with recent transitions owing to the finite memory size N. Similarly, the uniform sampling gives equal importance to all transitions in the replay memory. A more sophisticated sampling strategy might emphasize transitions from which we can learn the most, similar to prioritized sweeping[30].

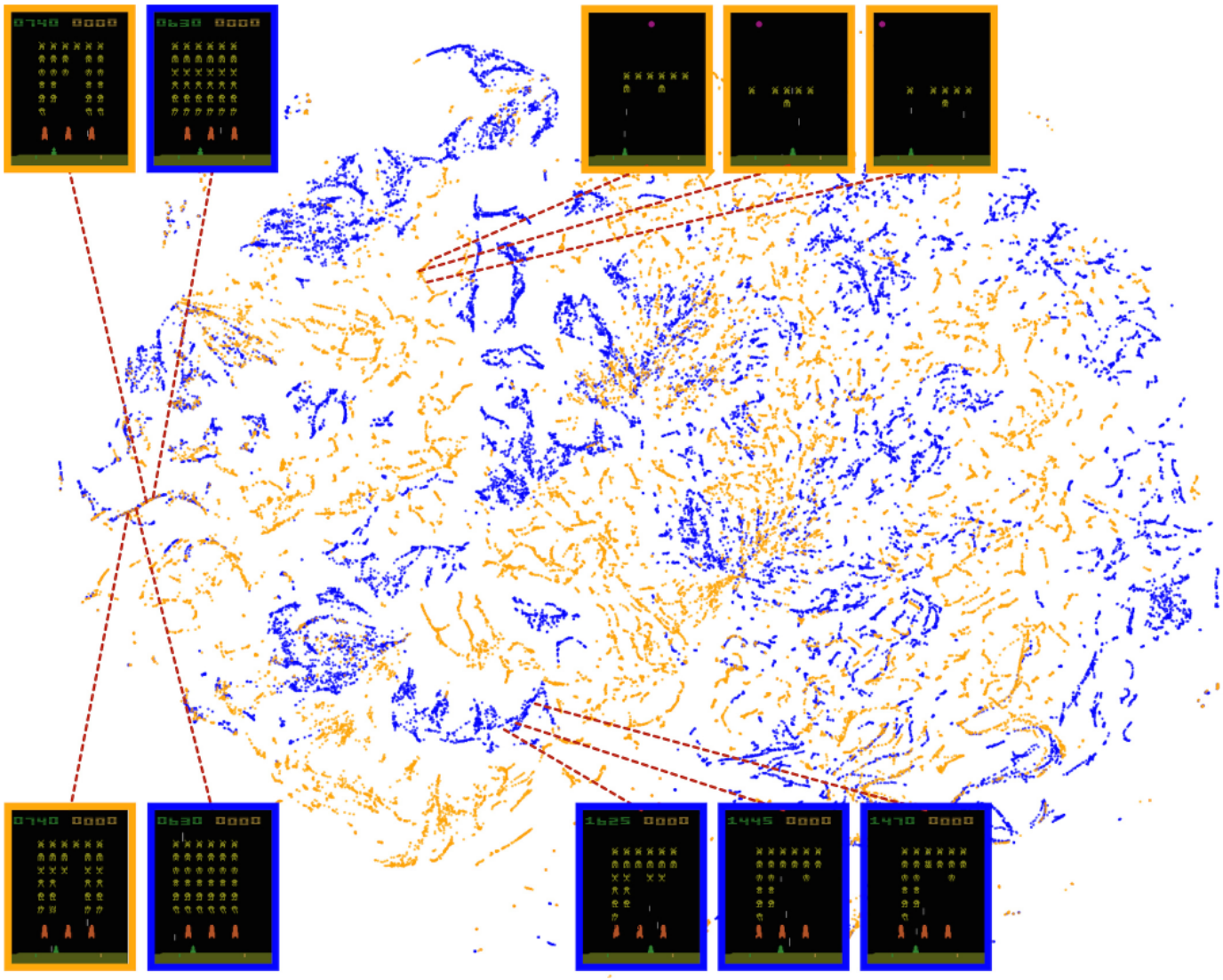
The second modification to online Q-learning aimed at further improving the stability of our method with neural networks is to use a separate network for generating the targets y_j in the Q-learning update. More precisely, every C updates we clone the network Q to obtain a target network $\hat{Q}$ and use $\hat{Q}$ for generating the Q-learning targets y_j for the following C updates to Q. This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t,a_t)$ often also increases $Q(s_{t+1},a)$ for all a and hence also increases the target y_j, possibly leading to oscillations or divergence of the policy. Generating the targets using an older set of parameters adds a delay between the time an update to Q is made and the time the update affects the targets y_j, making divergence or oscillations much more unlikely.

We also found it helpful to clip the error term from the update $r+\gamma \max_{a'} Q(s',a';\theta_i^-)-Q(s,a;\theta_i)$ to be between -1 and 1. Because the absolute value loss function $|x|$ has a derivative of -1 for all negative values of x and a derivative of 1 for all positive values of x, clipping the squared error to be between -1 and 1 corresponds to using an absolute value loss function for errors outside of the $(-1,1)$ interval. This form of error clipping further improved the stability of the algorithm.

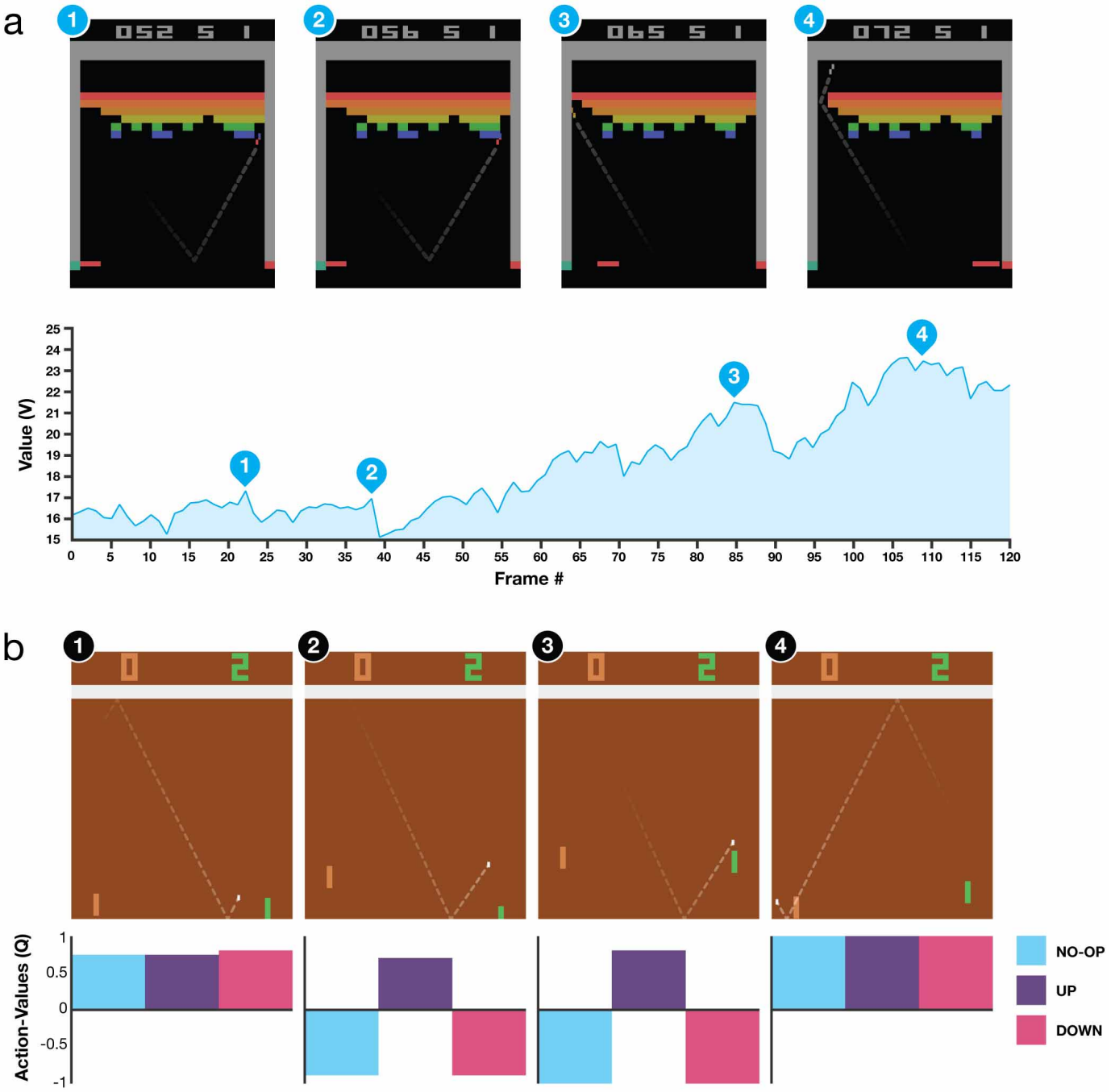
Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^-=\theta$
For episode = 1, M **do**
  Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$
  For $t=1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t=\mathrm{argmax}_a Q(\phi(s_t),a;\theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1}=s_t,a_t,x_{t+1}$ and preprocess $\phi_{t+1}=\phi(s_{t+1})$
    Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from D
    Set $y_j=\begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j+\gamma \max_{a'} \hat{Q}(\phi_{j+1},a';\theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j-Q(\phi_j,a_j;\theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q}=Q$
  End For
End For

31. Jarrett, K., Kavukcuoglu, K., Ranzato, M. A. & LeCun, Y. What is the best multi-stage architecture for object recognition? Proc. IEEE. Int. Conf. Comput. Vis. 2146–2153 (2009).
32. Nair, V. & Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. Proc. Int. Conf. Mach. Learn. 807–814 (2010).
33. Kaelbling, L. P., Littman, M. L. & Cassandra, A. R. Planning and acting in partially observable stochastic domains. Artificial Intelligence **101,** 99–134 (1994).



Extended Data Figure 1 | Two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced during a combination of human and agent play in Space Invaders. The plot was generated by running the t-SNE algorithm[25] on the last hidden layer representation assigned by DQN to game states experienced during a combination of human (30 min) and agent (2 h) play. The fact that there is similar structure in the two-dimensional embeddings corresponding to the DQN representation of states experienced during human play (orange points) and DQN play (blue points) suggests that the representations learned by DQN do indeed generalize to data generated from policies other than its own. The presence in the t-SNE embedding of overlapping clusters of points corresponding to the network representation of states experienced during human and agent play shows that the DQN agent also follows sequences of states similar to those found in human play. Screenshots corresponding to selected states are shown (human: orange border; DQN: blue border).



Extended Data Figure 2 | Visualization of learned value functions on two games, Breakout and Pong. a, A visualization of the learned value function on the game Breakout. At time points 1 and 2, the state value is predicted to be ~17 and the agent is clearing the bricks at the lowest level. Each of the peaks in the value function curve corresponds to a reward obtained by clearing a brick. At time point 3, the agent is about to break through to the top level of bricks and the value increases to ~21 in anticipation of breaking out and clearing a large set of bricks. At point 4, the value is above 23 and the agent has broken through. After this point, the ball will bounce at the upper part of the bricks clearing many of them by itself. **b**, A visualization of the learned action-value function on the game Pong. At time point 1, the ball is moving towards the paddle controlled by the agent on the right side of the screen and the values of all actions are around 0.7, reflecting the expected value of this state based on previous experience. At time point 2, the agent starts moving the paddle towards the ball and the value of the 'up' action stays high while the value of the 'down' action falls to −0.9. This reflects the fact that pressing 'down' would lead to the agent losing the ball and incurring a reward of −1. At time point 3, the agent hits the ball by pressing 'up' and the expected reward keeps increasing until time point 4, when the ball reaches the left edge of the screen and the value of all actions reflects that the agent is about to receive a reward of 1. Note, the dashed line shows the past trajectory of the ball purely for illustrative purposes (that is, not shown during the game). With permission from Atari Interactive, Inc.

Extended Data Table 1 | List of hyperparameters and their values

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter *C* from Algorithm 1).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.
no-op max	30	Maximum number of "do nothing" actions to be performed by the agent at the start of an episode.

The values of all the hyperparameters were selected by performing an informal search on the games Pong, Breakout, Seaquest, Space Invaders and Beam Rider. We did not perform a systematic grid search owing to the high computational cost, although it is conceivable that even better results could be obtained by systematically tuning the hyperparameter values.

Extended Data Table 2 | Comparison of games scores obtained by DQN agents with methods from the literature[12,15] and a professional human games tester

Game	Random Play	Best Linear Learner	Contingency (SARSA)	Human	DQN (± std)	Normalized DQN (% Human)
Alien	227.8	939.2	103.2	6875	3069 (±1093)	42.7%
Amidar	5.8	103.4	183.6	1676	739.5 (±3024)	43.9%
Assault	222.4	628	537	1496	3359(±775)	246.2%
Asterix	210	987.3	1332	8503	6012 (±1744)	70.0%
Asteroids	719.1	907.3	89	13157	1629 (±542)	7.3%
Atlantis	12850	62687	852.9	29028	85641(±17600)	449.9%
Bank Heist	14.2	190.8	67.4	734.4	429.7 (±650)	57.7%
Battle Zone	2360	15820	16.2	37800	26300 (±7725)	67.6%
Beam Rider	363.9	929.4	1743	5775	6846 (±1619)	119.8%
Bowling	23.1	43.9	36.4	154.8	42.4 (±88)	14.7%
Boxing	0.1	44	9.8	4.3	71.8 (±8.4)	1707.9%
Breakout	1.7	5.2	6.1	31.8	401.2 (±26.9)	1327.2%
Centipede	2091	8803	4647	11963	8309(±5237)	63.0%
Chopper Command	811	1582	16.9	9882	6687 (±2916)	64.8%
Crazy Climber	10781	23411	149.8	35411	114103 (±22797)	419.5%
Demon Attack	152.1	520.5	0	3401	9711 (±2406)	294.2%
Double Dunk	-18.6	-13.1	-16	-15.5	-18.1 (±2.6)	17.1%
Enduro	0	129.1	159.4	309.6	301.8 (±24.6)	97.5%
Fishing Derby	-91.7	-89.5	-85.1	5.5	-0.8 (±19.0)	93.5%
Freeway	0	19.1	19.7	29.6	30.3 (±0.7)	102.4%
Frostbite	65.2	216.9	180.9	4335	328.3 (±250.5)	6.2%
Gopher	257.6	1288	2368	2321	8520 (±3279)	400.4%
Gravitar	173	387.7	429	2672	306.7 (±223.9)	5.3%
H.E.R.O.	1027	6459	7295	25763	19950 (±158)	76.5%
Ice Hockey	-11.2	-9.5	-3.2	0.9	-1.6 (±2.5)	79.3%
James Bond	29	202.8	354.1	406.7	576.7 (±175.5)	145.0%
Kangaroo	52	1622	8.8	3035	6740 (±2959)	224.2%
Krull	1598	3372	3341	2395	3805 (±1033)	277.0%
Kung-Fu Master	258.5	19544	29151	22736	23270 (±5955)	102.4%
Montezuma's Revenge	0	10.7	259	4367	0 (±0)	0.0%
Ms. Pacman	307.3	1692	1227	15693	2311(±525)	13.0%
Name This Game	2292	2500	2247	4076	7257 (±547)	278.3%
Pong	-20.7	-19	-17.4	9.3	18.9 (±1.3)	132.0%
Private Eye	24.9	684.3	86	69571	1788 (±5473)	2.5%
Q*Bert	163.9	613.5	960.3	13455	10596 (±3294)	78.5%
River Raid	1339	1904	2650	13513	8316 (±1049)	57.3%
Road Runner	11.5	67.7	89.1	7845	18257 (±4268)	232.9%
Robotank	2.2	28.7	12.4	11.9	51.6 (±4.7)	509.0%
Seaquest	68.4	664.8	675.5	20182	5286(±1310)	25.9%
Space Invaders	148	250.1	267.9	1652	1976 (±893)	121.5%
Star Gunner	664	1070	9.4	10250	57997 (±3152)	598.1%
Tennis	-23.8	-0.1	0	-8.9	-2.5 (±1.9)	143.2%
Time Pilot	3568	3741	24.9	5925	5947 (±1600)	100.9%
Tutankham	11.4	114.3	98.2	167.6	186.7 (±41.9)	112.2%
Up and Down	533.4	3533	2449	9082	8456 (±3162)	92.7%
Venture	0	66	0.6	1188	380.0 (±238.6)	32.0%
Video Pinball	16257	16871	19761	17298	42684 (±16287)	2539.4%
Wizard of Wor	563.5	1981	36.9	4757	3393 (±2019)	67.5%
Zaxxon	32.5	3365	21.4	9173	4977 (±1235)	54.1%

Best Linear Learner is the best result obtained by a linear function approximator on different types of hand designed features[12]. Contingency (SARSA) agent figures are the results obtained in ref. 15. Note the figures in the last column indicate the performance of DQN relative to the human games tester, expressed as a percentage, that is, 100 × (DQN score − random play score)/(human score − random play score).

Extended Data Table 3 | The effects of replay and separating the target Q-network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN agents were trained for 10 million frames using standard hyperparameters for all possible combinations of turning replay on or off, using or not using a separate target Q-network, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).

Extended Data Table 4 | Comparison of DQN performance with linear function approximator

Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3

The performance of the DQN agent is compared with the performance of a linear function approximator on the 5 validation games (that is, where a single linear layer was used instead of the convolutional network, in combination with replay and separate target network). Agents were trained for 10 million frames using standard hyperparameters, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).

DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images

Yuying Ge[1], Ruimao Zhang[1], Lingyun Wu[2], Xiaogang Wang[1], Xiaoou Tang[1], and Ping Luo[1]

[1]The Chinese University of Hong Kong
[2]SenseTime Research

Abstract

*Understanding fashion images has been advanced by benchmarks with rich annotations such as DeepFashion, whose labels include clothing categories, landmarks, and consumer-commercial image pairs. However, DeepFashion has nonnegligible issues such as single clothing-item per image, sparse landmarks (4∼8 only), and no per-pixel masks, making it had significant gap from real-world scenarios. We fill in the gap by presenting DeepFashion2 to address these issues. It is a versatile benchmark of four tasks including clothes detection, pose estimation, segmentation, and retrieval. It has 801K clothing items where each item has rich annotations such as style, scale, viewpoint, occlusion, bounding box, dense landmarks (*e.g. *39 for 'long sleeve outwear' and 15 for 'vest'), and masks. There are also 873K Commercial-Consumer clothes pairs. The annotations of DeepFashion2 are much larger than its counterparts such as 8× of FashionAI Global Challenge. A strong baseline is proposed, called Match R-CNN, which builds upon Mask R-CNN to solve the above four tasks in an end-to-end manner. Extensive evaluations are conducted with different criterions in DeepFashion2. DeepFashion2 Dataset will be released at : https://github.com/switchablenorms/DeepFashion2*

1. Introduction

Fashion image analyses are active research topics in recent years because of their huge potential in industry. With the development of fashion datasets [20, 5, 7, 3, 14, 12, 21, 1], significant progresses have been achieved in this area [2, 19, 17, 18, 9, 8].

However, understanding fashion images remains a challenge in real-world applications, because of large deformations, occlusions, and discrepancies of clothes across domains between consumer and commercial images. Some challenges can be rooted in the gap between the recent benchmark and the practical scenario. For example, the existing largest fashion dataset, DeepFashion [14], has its own drawbacks such as single clothing item per image, sparse landmark and pose definition (every clothing category shares the same definition of $4 \sim 8$ keypoints), and no per-pixel mask annotation as shown in Fig.1(a).

Figure 1. Comparisons between (a) DeepFashion and (b) DeepFashion2. (a) only has single item per image, which is annotated with $4 \sim 8$ sparse landmarks. The bounding boxes are estimated from the labeled landmarks, making them noisy. In (b), each image has minimum single item while maximum 7 items. Each item is manually labeled with bounding box, mask, dense landmarks (20 per item on average), and commercial-customer image pairs.

To address the above drawbacks, this work presents DeepFashion2, a large-scale benchmark with comprehensive tasks and annotations of fashion image understanding. DeepFashion2 contains 491K images of 13 popular clothing categories. A full spectrum of tasks are defined on them including clothes detection and recognition, landmark and pose estimation, segmentation, as well as verification and retrieval. All these tasks are supported by rich annota-

tions. For instance, DeepFashion2 totally has 801K clothing items, where each item in an image is labeled with scale, occlusion, zooming, viewpoint, bounding box, dense landmarks, and per-pixel mask, as shown in Fig.1(b). These items can be grouped into 43.8K clothing identities, where a clothing identity represents the clothes that have almost the same cutting, pattern, and design. The images of the same identity are taken by both customers and commercial shopping stores. An item from the customer and an item from the commercial store forms a pair. There are 873K pairs that are 3.5 times larger than DeepFashion. The above thorough annotations enable developments of strong algorithms to understand fashion images.

This work has three main **contributions**. (1) We build a large-scale fashion benchmark with comprehensive tasks and annotations, to facilitate fashion image analysis. DeepFashion2 possesses the richest definitions of tasks and the largest number of labels. Its annotations are at least 3.5× of DeepFashion [14], 6.7× of ModaNet [21], and 8× of FashionAI [1]. (2) A full spectrum of tasks is carefully defined on the proposed dataset. For example, to our knowledge, clothing pose estimation is presented for the first time in the literature by defining landmarks and poses of 13 categories that are more diverse and fruitful than human pose. (3) With DeepFashion2, we extensively evaluate Mask R-CNN [6] that is a recent advanced framework for visual perception. A novel Match R-CNN is also proposed to aggregate all the learned features from clothes categories, poses, and masks to solve clothing image retrieval in an end-to-end manner. DeepFashion2 and implementations of Match R-CNN will be released.

1.1. Related Work

Clothes Datasets. Several clothes datasets have been proposed such as [20, 5, 7, 14, 21, 1] as summarized in Table 1. They vary in size as well as amount and type of annotations. For example, WTBI [5] and DARN [7] have 425K and 182K images respectively. They scraped category labels from metadata of the collected images from online shopping websites, making their labels noisy. In contrast, CCP [20], DeepFashion [14], and ModaNet [21] obtain category labels from human annotators. Moreover, different kinds of annotations are also provided in these datastes. For example, DeepFashion labels 4∼8 landmarks (keypoints) per image that are defined on the functional regions of clothes (*e.g.* 'collar'). The definitions of these sparse landmarks are shared across all categories, making them difficult to capture rich variations of clothing images. Furthermore, DeepFashion does not have mask annotations. By comparison, ModaNet [21] has street images with masks (polygons) of single person but without landmarks. Unlike existing datasets, DeepFashion2 contains 491K images and 801K instances of landmarks, masks, and bounding boxes, as well as 873K pairs. It is the most comprehensive benchmark of its kinds to date.

	WTBI	DARN	DeepFashion	ModaNet	FashionAI	DeepFashion2
year	2015[5]	2015[7]	2016[14]	2018[21]	2018[1]	now
#images	425K	182K	800K	55K	357K	491K
#categories	11	20	50	13	41	13
#bboxes	39K	7K	×	×	×	801K
#landmarks	×	×	120K	×	100K	801K
#masks	×	×	×	119K	×	801K
#pairs	39K	91K	251K	×	×	873K

Table 1. Comparisons of DeepFashion2 with the other clothes datasets. The rows represent number of images, bounding boxes, landmarks, per-pixel masks, and consumer-to-shop pairs respectively. Bounding boxes inferred from other annotations are not counted.

Fashion Image Understanding. There are various tasks that analyze clothing images such as clothes detection [2, 14], landmark prediction [15, 19, 17], clothes segmentation [18, 20, 13], and retrieval [7, 5, 14]. However, a unify benchmark and framework to account for all these tasks is still desired. DeepFashion2 and Match R-CNN fill in this blank. We report extensive results for the above tasks with respect to different variations, including scale, occlusion, zoom-in, and viewpoint. For the task of clothes retrieval, unlike previous methods [5, 7] that performed image-level retrieval, DeepFashion2 enables instance-level retrieval of clothing items. We also present a new fashion task called clothes pose estimation, which is inspired by human pose estimation to predict clothing landmarks and skeletons for 13 clothes categories. This task helps improve performance of fashion image analysis in real-world applications.

2. DeepFashion2 Dataset and Benchmark

Overview. DeepFashion2 has four unique characteristics compared to existing fashion datasets. (1) *Large Sample Size.* It contains 491K images of 43.8K clothing identities of interest (unique garment displayed by shopping stores). On average, each identity has 12.7 items with different styles such as color and printing. DeepFashion2 contained 801K items in total. It is the largest fashion database to date. Furthermore, each item is associated with various annotations as introduced above.

(2) *Versatility.* DeepFashion2 is developed for multiple tasks of fashion understanding. Its rich annotations support clothes detection and classification, dense landmark and pose estimation, instance segmentation, and cross-domain instance-level clothes retrieval.

(3) *Expressivity.* This is mainly reflected in two aspects. First, multiple items are present in a single image, unlike DeepFashion where each image is labeled with at most one item. Second, we have 13 different definitions of landmarks and poses (skeletons) for 13 different categories. There is

Figure 2. **Examples of DeepFashion2.** The first column shows definitions of dense landmarks and skeletons of four categories. From (1) to (4), each row represents clothes images with different variations including 'scale', 'occlusion', 'zoom-in', and 'viewpoint'. At each row, we partition the images into two groups, the left three columns represent clothes from commercial stores, while the right three columns are from customers. In each group, the three images indicate three levels of difficulty with respect to the corresponding variation, including (1) 'small', 'moderate', 'large' scale, (2) 'slight', 'medium', 'heavy' occlusion, (3) 'no', 'medium', 'large' zoom-in, (4) 'not on human', 'side', 'back' viewpoint. Furthermore, at each row, the items in these two groups of images are from the same clothing identity but from two different domains, that is, commercial and customer. The items of the same identity may have different styles such as color and printing. Each item is annotated with landmarks and masks.

23 defined landmarks for each category on average. Some definitions are shown in the first column of Fig.2. These representations are different from human pose and are not presented in previous work. They facilitate learning of strong clothes features that satisfy real-world requirements.

(4) *Diversity.* We collect data by controlling their variations in terms of four properties including scale, occlusion, zoom-in, and viewpoint as illustrated in Fig.2, making DeepFashion2 a challenging benchmark. For each property, each clothing item is assigned to one of three levels of difficulty. Fig.2 shows that each identity has high diversity where its items are from different difficulties.

Data Collection and Cleaning. Raw data of DeepFashion2 are collected from two sources including DeepFashion [14] and online shopping websites. In particular, images of each consumer-to-shop pair in DeepFashion are included in DeepFashion2, while the other images are removed. We further crawl a large set of images on the Internet from both commercial shopping stores and consumers. To clean up the crawled set, we first remove shop images with no corresponding consumer-taken photos. Then human annotators are asked to clean images that contain clothes with large occlusions, small scales, and low resolutions. Eventually we have 491K images of 801K items and 873K commercial-consumer pairs.

Variations. We explain the variations in DeepFashion2. Their statistics are plotted in Fig.3. (1) ***Scale.*** We divide all clothing items into three sets, according to the proportion of an item compared to the image size, including 'small' ($< 10\%$), 'moderate' ($10\% \sim 40\%$), and 'large' ($> 40\%$). Fig.3(a) shows that only 50% items have moderate scale. (2) ***Occlusion.*** An item with occlusion means that its region is occluded by hair, human body, accessory or other items. Note that an item with its region outside the im-

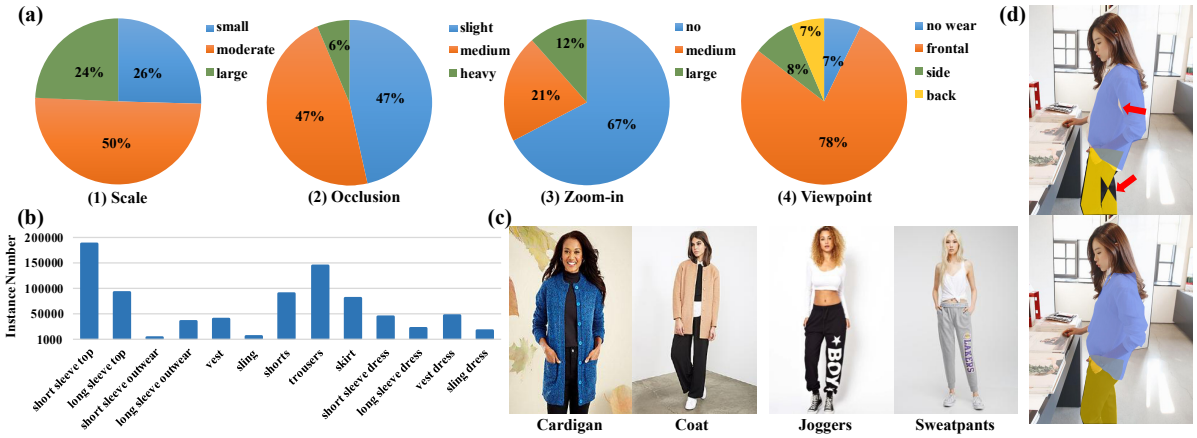


Figure 3. (a) shows the statistics of different variations in DeepFashion2. (b) is the numbers of items of the 13 categories in DeepFashion2. (c) shows that categories in DeepFashion [14] have ambiguity. For example, it is difficult to distinguish between 'cardigan' and 'coat', and between 'joggers' and 'sweatpants'. They result in ambiguity when labeling data. (d) **Top:** masks may be inaccurate when complex poses are presented. **Bottom:** the masks will be refined by human.

age does not belong to this case. Each item is categorized by the number of its landmarks that are occluded, including 'partial occlusion'($< 20\%$ occluded keypoints), 'heavy occlusion' ($> 50\%$ occluded keypoints), 'medium occlusion' (otherwise). More than 50% items have medium or heavy occlusions as summarized in Fig.3. (3) ***Zoom-in.*** An item with zoom-in means that its region is outside the image. This is categorized by the number of landmarks outside image. We define 'no', 'large' ($> 30\%$), and 'medium' zoom-in. We see that more than 30% items are zoomed in. (4) ***Viewpoint.*** We divide all items into four partitions including 7% clothes that are not on people, 78% clothes on people from frontal viewpoint, 15% clothes on people from side or back viewpoint.

2.1. Data Labeling

Category and Bounding Box. Human annotators are asked to draw a bounding box and assign a category label for each clothing item. DeepFashion [14] defines 50 categories but half of them contain less than 5‰ number of images. Also, ambiguity exists between 50 categories making data labeling difficult as shown in Fig.3(c). By grouping categories in DeepFashion, we derive 13 popular categories without ambiguity. The numbers of items of 13 categories are shown in Fig.3(b).

Clothes Landmark, Contour, and Skeleton. As different categories of clothes (*e.g.* upper- and lower-body garment) have different deformations and appearance changes, we represent each category by defining its pose, which is a set of landmarks as well as contours and skeletons between landmarks. They capture shapes and structures of clothes. Pose definitions are not presented in previous work and are significantly different from human pose. For each clothing item of a category, human annotations are asked to label landmarks following these instructions.

Moreover, each landmark is assigned one of the two modes, 'visible' or 'occluded'. We then generate contours and skeletons automatically by connecting landmarks in a certain order. To facilitate this process, annotators are also asked to distinguish landmarks into two types, that is, contour point or junction point. The former one refers to keypoints at the boundary of an item, while the latter one is assigned to keypoints in conjunction *e.g.* 'endpoint of strap on sling'. The above process controls the labeling quality, because the generated skeletons help the annotators reexamine whether the landmarks are labeled with good quality. In particular, only when the contour covers the entire item, the labeled results are eligible, otherwise keypoints will be refined.

Mask. We label per-pixel mask for each item in a semi-automatic manner with two stages. The first stage automatically generates masks from the contours. In the second stage, human annotators are asked to refine the masks, because the generated masks may be not accurate when complex human poses are presented. As shown in Fig.3(d), the mark is inaccurate when an image is taken from side-view of people crossing legs. The masks will be refined by human.

Style. As introduced before, we collect 43.8K different clothing identities where each identity has 13 items on average. These items are further labeled with different styles such as color, printing, and logo. Fig.2 shows that a pair of clothes that have the same identity could have different styles.

2.2. Benchmarks

We build four benchmarks by using the images and labels from DeepFashion2. For each benchmark, there are

391K images for training, 34K images for validation and 67K images for test.

Clothes Detection. This task detects clothes in an image by predicting bounding boxes and category labels. The evaluation metrics are the bounding box's average precision $\mathrm{AP}_{\mathrm{box}}$, $\mathrm{AP}_{\mathrm{box}}^{\mathrm{IoU}=0.50}$, and $\mathrm{AP}_{\mathrm{box}}^{\mathrm{IoU}=0.75}$ by following COCO [11].

Landmark Estimation. This task aims to predict landmarks for each detected clothing item in an each image. Similarly, we employ the evaluation metrics used by COCO for human pose estimation by calculating the average precision for keypoints $\mathrm{AP}_{\mathrm{pt}}$, $\mathrm{AP}_{\mathrm{pt}}^{\mathrm{OKS}=0.50}$, and $\mathrm{AP}_{\mathrm{pt}}^{\mathrm{OKS}=0.75}$, where OKS indicates the object landmark similarity.

Segmentation. This task assigns a category label (including background label) to each pixel in an item. The evaluation metrics is the average precision including $\mathrm{AP}_{\mathrm{mask}}$, $\mathrm{AP}_{\mathrm{mask}}^{\mathrm{IoU}=0.50}$, and $\mathrm{AP}_{\mathrm{mask}}^{\mathrm{IoU}=0.75}$ computed over masks.

Commercial-Consumer Clothes Retrieval. Given a detected item from a consumer-taken photo, this task aims to search the commercial images in the gallery for the items that are corresponding to this detected item. This setting is more realistic than DeepFashion [14], which assumes ground-truth bounding box is provided. In this task, top-k retrieval accuracy is employed as the evaluation metric. We emphasize the retrieval performance while still consider the influence of detector. If a clothing item fails to be detected, this query item is counted as missed. In particular, we have more than 686K commercial-consumer clothes pairs in the training set. In the validation set, there are $10,990$ consumer images with $12,550$ items as a query set, and $21,438$ commercial images with $37,183$ items as a gallery set. In the test set, there are $21,550$ consumer images with $24,402$ items as queries, while $43,608$ commercial images with $75,347$ items in the gallery.

3. Match R-CNN

We present a strong baseline model built upon Mask R-CNN [6] for DeepFashion2, termed Match R-CNN, which is an end-to-end training framework that jointly learns clothes detection, landmark estimation, instance segmentation, and consumer-to-shop retrieval. The above tasks are solved by using different streams and stacking a Siamese module on top of these streams to aggregate learned features.

As shown in Fig.4, Match R-CNN employs two images I_1 and I_2 as inputs. Each image is passed through three main components including a Feature Network (FN), a Perception Network (PN), and a Matching Network (MN). In the first stage, FN contains a ResNet-FPN [10] backbone, a region proposal network (RPN) [16] and RoIAlign module. An image is first fed into ResNet50 to extract features, which are then fed into a FPN that uses a top-down architecture with lateral connections to build a pyramid of feature maps. RoIAlign extracts features from different levels of the pyramid map.

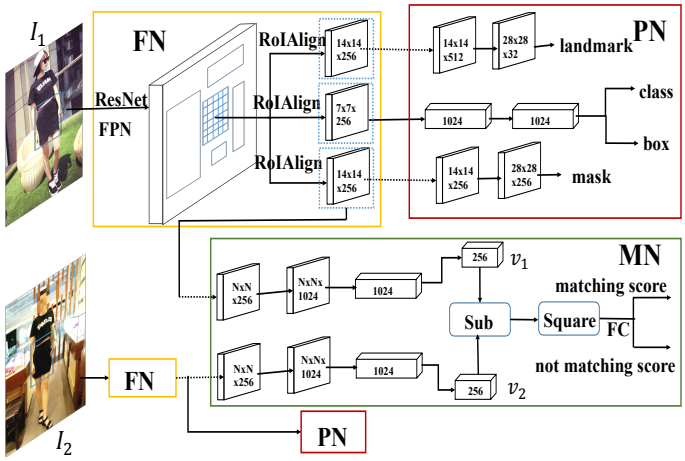


Figure 4. **Diagram of Match R-CNN** that contains three main components including a feature extraction network (FN), a perception network (PN), and a match network (MN).

In the second stage, PN contains three streams of networks including landmark estimation, clothes detection, and mask prediction as shown in Fig.4. The extracted RoI features after the first stage are fed into three streams in PN separately. The clothes detection stream has two hidden fully-connected (fc) layers, one fc layer for classification, and one fc layer for bounding box regression. The stream of landmark estimation has 8 'conv' layers and 2 'deconv' layers to predict landmarks. Segmentation stream has 4 'conv' layers, 1 'deconv' layer, and another 'conv' layer to predict masks.

In the third stage, MN contains a feature extractor and a similarity learning network for clothes retrieval. The learned RoI features after the FN component are highly discriminative with respect to clothes category, pose, and mask. They are fed into MN to obtain features vectors for retrieval, where v_1 and v_2 are passed into the similarity learning network to obtain the similarity score between the detected clothing items in I_1 and I_2. Specifically, the feature extractor has 4 'conv' layers, one pooling layer, and one fc layer. The similarity learning network consists of subtraction and square operator and a fc layer, which estimates the probability of whether two clothing items match or not.

Loss Functions. The parameters Θ of the Match R-CNN are optimized by minimizing five loss functions, which are formulated as $\min_{\Theta} \mathcal{L} = \lambda_1 \mathcal{L}_{cls} + \lambda_2 \mathcal{L}_{box} + \lambda_3 \mathcal{L}_{pose} + \lambda_4 \mathcal{L}_{mask} + \lambda_5 \mathcal{L}_{pair}$, including a cross-entropy (CE) loss $\mathcal{L}_{cls}$ for clothes classification, a smooth loss [4] $\mathcal{L}_{box}$ for bounding box regression, a CE loss $\mathcal{L}_{pose}$ for landmark es-

	scale			occlusion			zoom-in			viewpoint			overall
	small	moderate	large	slight	medium	heavy	no	medium	large	no wear	frontal	side or back	
AP_{box}	0.604	**0.700**	0.660	**0.712**	0.654	0.372	**0.695**	0.629	0.466	0.624	**0.681**	0.641	0.667
$AP_{box}^{IoU=0.50}$	0.780	**0.851**	0.768	**0.844**	0.810	0.531	**0.848**	0.755	0.563	0.713	**0.832**	0.796	0.814
$AP_{box}^{IoU=0.75}$	0.717	**0.809**	0.744	**0.812**	0.768	0.433	**0.806**	0.718	0.525	0.688	**0.791**	0.744	0.773

Table 2. **Clothes detection** of Mask R-CNN [6] on different validation subsets, including scale, occlusion, zoom-in, and viewpoint. The evaluation metrics are AP_{box}, $AP_{box}^{IoU=0.50}$, and $AP_{box}^{IoU=0.75}$. The best performance of each subset is bold.

Figure 5. (a) shows failure cases in clothes detection while (b) shows failure cases in clothes segmentation. In (a) and (b), the missing bounding boxes are drawn in red while the correct category labels are also in red. Inaccurate masks are also highlighted by arrows in (b). For example, clothes fail to be detected or segmented in too small scale, too large scale, large non-rigid deformation, heavy occlusion, large zoom-in, side or back viewpoint.

timation, a CE loss $\mathcal{L}_{mask}$ for clothes segmentation, and a CE loss $\mathcal{L}_{pair}$ for clothes retrieval. Specifically, $\mathcal{L}_{cls}$, $\mathcal{L}_{box}$, $\mathcal{L}_{pose}$, and $\mathcal{L}_{mask}$ are identical as defined in [6]. We have $\mathcal{L}_{pair} = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(\hat{y}_i) + (1-y_i)log(1-\hat{y}_i)]$, where $y_i = 1$ indicates the two items of a pair are matched, otherwise $y_i = 0$.

Implementations. In our experiments, each training image is resized to its shorter edge of 800 pixels with its longer edge that is no more than 1333 pixels. Each minibatch has two images in a GPU and 8 GPUs are used for training. For minibatch size 16, the learning rate (LR) schedule starts at 0.02 and is decreased by a factor of 0.1 after 8 epochs and then 11 epochs, and finally terminates at 12 epochs. This scheduler is denoted as 1x. Mask R-CNN adopts 2x schedule for clothes detection and segmentation where '2x' is twice as long as 1x with the LR scaled proportionally. Then It adopts s1x for landmark and pose estimation where s1x scales the 1x schedule by roughly 1.44x. Match R-CNN uses 1x schedule for consumer-to-shop clothes retrieval. The above models are trained by using SGD with a weight decay of 10^{-5} and momentum of 0.9.

In our experiments, the RPN produces anchors with 3 aspect rations on each level of the FPN pyramid. In clothes detection stream, an RoI is considered positive if its IoU with a ground truth box is larger than 0.5 and negative otherwise. In clothes segmentation stream, positive RoIs with foreground label are chosen while in landmark estimation stream, positive RoIs with visible landmarks are selected. We define ground truth box of interest as clothing items whose style number is > 0 and can constitute matching pairs. In clothes retrieval stream, RoIs are selected if their IoU with a ground truth box of interest is larger than 0.7. If RoI features are extracted from landmark estimation stream, RoIs with visible landmarks are also selected.

Inference. At testing time, images are resized in the same way as the training stage. The top 1000 proposals with detection probabilities are chosen for bounding box classification and regression. Then non-maximum suppression is applied to these proposals. The filtered proposals are fed into the landmark branch and the mask branch separately. For the retrieval task, each unique detected clothing item in consumer-taken image with highest confidence is selected as query.

4. Experiments

We demonstrate the effectiveness of DeepFashion2 by evaluating Mask R-CNN [6] and Match R-CNN in multiple tasks including clothes detection and classification, landmark estimation, instance segmentation, and consumer-to-

	scale			occlusion			zoom-in			viewpoint			overall
	small	moderate	large	slight	medium	heavy	no	medium	large	no wear	frontal	side or back	
AP_{pt}	0.587	**0.687**	0.599	**0.669**	0.631	0.398	**0.688**	0.559	0.375	0.527	**0.677**	0.536	0.641
	0.497	**0.607**	0.555	**0.643**	0.530	0.248	**0.616**	0.489	0.319	0.510	**0.596**	0.456	0.563
$AP_{pt}^{OKS=0.50}$	0.780	**0.854**	0.782	**0.851**	0.813	0.534	**0.855**	0.757	0.571	0.724	**0.846**	0.748	0.820
	0.764	**0.839**	0.774	**0.847**	0.799	0.479	**0.848**	0.744	0.549	0.716	**0.832**	0.727	0.805
$AP_{pt}^{OKS=0.75}$	0.671	**0.779**	0.678	**0.760**	0.718	0.440	**0.786**	0.633	0.390	0.571	**0.771**	0.610	0.728
	0.551	**0.703**	0.625	**0.739**	0.600	0.236	**0.714**	0.537	0.307	0.550	**0.684**	0.506	0.641

Table 3. **Landmark estimation** of Mask R-CNN [6] on different validation subsets, including scale, occlusion, zoom-in, and viewpoint. Results of evaluation on visible landmarks only and evaluation on both visible and occlusion landmarks are separately shown in each row. The evaluation metrics are AP_{pt}, $AP_{pt}^{OKS=0.50}$, and $AP_{pt}^{OKS=0.75}$. The best performance of each subset is bold.

Figure 6. (a) shows results of landmark and pose estimation. (b) shows results of clothes segmentation. (c) shows queries with top-5 retrieved clothing items. The first column is the image from the customer with bounding box predicted by detection module, and the second to the sixth columns show the retrieval results from the store. (d) is the retrieval accuracy of overall query validation set with (1) detected box (2) ground truth box. Evaluation metrics are top-1, -5, -10, -15, and -20 retrieval accuracy.

shop clothes retrieval. To further show the large variations of DeepFashion2, the validation set is divided into three subsets according to their difficulty levels in scale, occlusion, zoom-in, and viewpoint. The settings of Mask R-CNN and Match R-CNN follow Sec.3. All models are trained in the training set and evaluated in the validation set.

The following sections from 4.1 to 4.4 report results for different tasks, showing that DeepFashion2 imposes significant challenges to both Mask R-CNN and Match R-CNN, which are the recent state-of-the-art systems for visual perception.

4.1. Clothes Detection

Table 2 summarizes the results of clothes detection on different difficulty subsets. We see that the clothes of moderate scale, slight occlusion, no zoom-in, and frontal viewpoint have the highest detection rates. There are several observations. First, detecting clothes with small or large scale reduces detection rates. Some failure cases are provided in Fig.5(a) where the item could occupy less than 2% of the image while some occupies more than 90% of the image. Second, in Table 2, it is intuitively to see that heavy occlusion and large zoom-in degenerate performance. In these two cases, large portions of the clothes are invisible as shown in Fig.5(a). Third, it is seen in Table 2 that the clothing items not on human body also drop performance. This is because they possess large non-rigid deformations as visualized in the failure cases of Fig.5(a). These variations are not presented in previous object detection benchmarks such as COCO. Fourth, clothes with side or back viewpoint, are much more difficult to detect as shown in Fig.5(a).

4.2. Landmark and Pose Estimation

Table 3 summarizes the results of landmark estimation. The evaluation of each subset is performed in two settings, including visible landmark only (the occluded landmarks are not evaluated), as well as both visible and occluded landmarks. As estimating the occluded landmarks is more difficult than visible landmarks, the second setting generally provides worse results than the first setting.

In general, we see that Mask R-CNN obtains an overall

	scale			occlusion			zoom-in			viewpoint			overall
	small	moderate	large	slight	medium	heavy	no	medium	large	no wear	frontal	side or back	
AP_{mask}	0.634	**0.700**	0.669	**0.720**	0.674	0.389	**0.703**	0.627	0.526	0.695	**0.697**	0.617	0.680
$AP^{IoU=0.50}_{mask}$	0.831	**0.900**	0.844	**0.900**	0.878	0.559	**0.899**	0.815	0.663	0.829	**0.886**	0.843	0.873
$AP^{IoU=0.75}_{mask}$	0.765	**0.838**	0.786	**0.850**	0.813	0.463	**0.842**	0.740	0.613	0.792	**0.834**	0.732	0.812

Table 4. **Clothes segmentation** of Mask R-CNN [6] on different validation subsets, including scale, occlusion, zoom-in, and viewpoint. The evaluation metrics are AP_{mask}, $AP^{IoU=0.50}_{mask}$, and $AP^{IoU=0.75}_{mask}$. The best performance of each subset is bold.

	scale			occlusion			zoom-in			viewpoint			overall		
	small	moderate	large	slight	medium	heavy	no	medium	large	no wear	frontal	side or back	top-1	top-10	top-20
class	0.513	**0.619**	0.547	**0.580**	0.556	0.503	**0.608**	0.557	0.441	0.555	**0.580**	0.533	0.122	0.363	0.464
	0.445	**0.558**	0.515	**0.542**	0.514	0.361	**0.557**	0.514	0.409	0.508	**0.529**	0.519	0.104	0.321	0.417
pose	0.695	**0.775**	0.729	**0.752**	0.729	0.698	**0.769**	0.742	0.618	0.725	**0.755**	0.705	0.255	0.555	0.647
	0.619	**0.695**	0.688	**0.704**	0.668	0.559	**0.700**	0.693	0.572	0.682	**0.690**	0.654	0.234	0.495	0.589
mask	0.641	**0.705**	0.663	**0.688**	0.656	0.645	**0.708**	0.670	0.556	0.650	**0.690**	0.653	0.187	0.471	0.573
	0.584	**0.656**	0.632	**0.657**	0.619	0.512	**0.663**	0.630	0.541	0.628	**0.645**	0.602	0.175	0.421	0.529
pose+class	0.752	**0.786**	0.733	**0.754**	0.750	0.728	**0.789**	0.750	0.620	0.726	**0.771**	0.719	0.268	0.574	0.665
	0.691	**0.730**	0.705	**0.725**	0.706	0.605	**0.746**	0.709	0.582	0.699	**0.723**	0.684	0.244	0.522	0.617
mask+class	0.679	**0.738**	0.685	**0.711**	0.695	0.651	**0.742**	0.699	0.569	0.677	**0.719**	0.678	0.214	0.510	0.607
	0.623	**0.696**	0.661	**0.685**	0.659	0.568	**0.708**	0.667	0.566	0.659	**0.676**	0.657	0.200	0.463	0.564

Table 5. **Consumer-to-Shop Clothes Retrieval** of Match R-CNN on different subsets of some validation consumer-taken images. Each query item in these images has over 5 identical clothing items in validation commercial images. Results of evaluation on ground truth box and detected box are separately shown in each row. The evaluation metrics are top-20 accuracy. The best performance of each subset is bold.

AP of just 0.563, showing that clothes landmark estimation could be even more challenging than human pose estimation in COCO. In particular, Table 3 exhibits similar trends as those from clothes detection. For example, the clothing items with moderate scale, slight occlusion, no zoom-in, and frontal viewpoint have better results than the others subsets. Moreover, heavy occlusion and zoom-in decreases performance a lot. Some results are given in Fig.6(a).

4.3. Clothes Segmentation

Table 4 summarizes the results of segmentation. The performance declines when segmenting clothing items with small and large scale, heavy occlusion, large zoom-in, side or back viewpoint, which is consistent with those trends in the previous tasks. Some results are given in Fig.6(b). Some failure cases are visualized in Fig.5(b).

4.4. Consumer-to-Shop Clothes Retrieval

Table 5 summarizes the results of clothes retrieval. The retrieval accuracy is reported in Fig. 6(d), where top-1, -5, -10, and -20 retrieval accuracy are shown. We evaluate two settings in (c.1) and (c.2), when the bounding boxes are predicted by the detection module in Match R-CNN and are provided as ground truths. Match R-CNN achieves a top-20 accuracy of less than 0.7 with ground-truth bounding boxes provided, indicating that the retrieval benchmark is challenging. Furthermore, retrieval accuracy drops when using detected boxes, meaning that this is a more realistic setting.

In Table 5, different combinations of the learned features are also evaluated. In general, the combination of features increases the accuracy. In particular, the learned features from pose and class achieve better results than the other features. When comparing learned features from pose and mask, we find that the former achieves better results, indicating that landmark locations can be more robust across scenarios.

As shown in Table 5, the performance declines when small scale, heavily occluded clothing items are presented. Clothes with large zoom-in achieved the lowest accuracy because only part of clothes are displayed in the image and crucial distinguishable features may be missing. Compared with clothes on people from frontal view, clothes from side or back viewpoint perform worse due to lack of discriminative features like patterns on the front of tops. Example queries with top-5 retrieved clothing items are shown in Fig.6(c).

5. Conclusions

This work represented DeepFashion2, a large-scale fashion image benchmark with comprehensive tasks and annotations. DeepFashion2 contains 491K images, each of which is richly labeled with style, scale, occlusion, zooming, viewpoint, bounding box, dense landmarks and pose, pixel-level masks, and pair of images of identical item from consumer and commercial store. We establish benchmarks covering multiple tasks in fashion understanding, including clothes detection, landmark and pose estimation, clothes segmentation, consumer-to-shop verification and retrieval. A novel Match R-CNN framework that builds upon Mask R-CNN is proposed to solve the above tasks in end-to-end manner. Extensive evaluations are conducted in DeepFash-

ion2.

The rich data and labels of DeepFashion2 will definitely facilitate the developments of algorithms to understand fashion images in future work. We will focus on three aspects. First, more challenging tasks will be explored with DeepFashion2, such as synthesizing clothing images by using GANs. Second, it is also interesting to explore multi-domain learning for clothing images, because fashion trends of clothes may change frequently, making variations of clothing images changed. Third, we will introduce more evaluation metrics into DeepFashion2, such as size, runtime, and memory consumptions of deep models, towards understanding fashion images in real-world scenario.

References

[1] Fashionai dataset. http://fashionai.alibaba.com/datasets/.

[2] H. Chen, A. Gallagher, and B. Girod. Describing clothing by semantic attributes. In *ECCV*, 2012.

[3] Q. Chen, J. Huang, R. Feris, L. M. Brown, J. Dong, and S. Yan. Deep domain adaptation for describing people based on fine-grained clothing attributes. In *CVPR*, 2015.

[4] R. Girshick. Fast r-cnn. In *ICCV*, 2015.

[5] M. Hadi Kiapour, X. Han, S. Lazebnik, A. C. Berg, and T. L. Berg. Where to buy it: Matching street clothing photos in online shops. In *ICCV*, 2015.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, 2017.

[7] J. Huang, R. S. Feris, Q. Chen, and S. Yan. Cross-domain image retrieval with a dual attribute-aware ranking network. In *ICCV*, 2015.

[8] X. Ji, W. Wang, M. Zhang, and Y. Yang. Cross-domain image retrieval with attention modeling. In *ACM Multimedia*, 2017.

[9] L. Liao, X. He, B. Zhao, C.-W. Ngo, and T.-S. Chua. Interpretable multimodal retrieval for fashion products. In *ACM Multimedia*, 2018.

[10] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[12] K.-H. Liu, T.-Y. Chen, and C.-S. Chen. Mvc: A dataset for view-invariant clothing retrieval and attribute prediction. In *ACM Multimedia*, 2016.

[13] S. Liu, X. Liang, L. Liu, K. Lu, L. Lin, X. Cao, and S. Yan. Fashion parsing with video context. *IEEE Transactions on Multimedia*, 17(8):1347–1358, 2015.

[14] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *CVPR*, 2016.

[15] Z. Liu, S. Yan, P. Luo, X. Wang, and X. Tang. Fashion landmark detection in the wild. In *ECCV*, 2016.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[17] W. Wang, Y. Xu, J. Shen, and S.-C. Zhu. Attentive fashion grammar network for fashion landmark detection and clothing category classification. In *CVPR*, 2018.

[18] K. Yamaguchi, M. Hadi Kiapour, and T. L. Berg. Paper doll parsing: Retrieving similar styles to parse clothing items. In *ICCV*, 2013.

[19] S. Yan, Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Unconstrained fashion landmark detection via hierarchical recurrent transformer networks. In *ACM Multimedia*, 2017.

[20] W. Yang, P. Luo, and L. Lin. Clothing co-parsing by joint image segmentation and labeling. In *CVPR*, 2014.

[21] S. Zheng, F. Yang, M. H. Kiapour, and R. Piramuthu. Modanet: A large-scale street fashion dataset with polygon annotations. In *ACM Multimedia*, 2018.

Semi-Supervised Learning with Ladder Network

Antti Rasmus
Nvidia, Finland

Harri Valpola
ZenRobotics, Finland

Mikko Honkala
Nokia Technologies, Finland

Mathias Berglund
Aalto University, Finland

Tapani Raiko
Aalto University, Finland

Abstract

We combine supervised learning with unsupervised learning in deep neural networks. The proposed model is trained to simultaneously minimize the sum of supervised and unsupervised cost functions by backpropagation, avoiding the need for layer-wise pretraining. Our work builds on top of the Ladder network proposed by Valpola (2015) which we extend by combining the model with supervision. We show that the resulting model reaches state-of-the-art performance in various tasks: MNIST and CIFAR-10 classification in a semi-supervised setting and permutation invariant MNIST in both semi-supervised and full-labels setting.

1 Introduction

In this paper, we introduce an unsupervised learning method that fits well with supervised learning. The idea of using unsupervised learning to complement supervision is not new. Combining an auxiliary task to help train a neural network was proposed by Suddarth and Kergosien (1990). By sharing the hidden representations among more than one task, the network generalizes better. There are multiple choices for the unsupervised task, for example, reconstructing the inputs at every level of the model (e.g., Ranzato and Szummer, 2008) or classification of each input sample into its own class (Dosovitskiy *et al.*, 2014).

Although some methods have been able to simultaneously apply both supervised and unsupervised learning (Ranzato and Szummer, 2008; Goodfellow *et al.*, 2013a), often these unsupervised auxiliary tasks are only applied as pre-training, followed by normal supervised learning (e.g., Hinton and Salakhutdinov, 2006). In complex tasks there is often much more structure in the inputs than can be represented, and unsupervised learning cannot, by definition, know what will be useful for the task at hand. Consider, for instance, the autoencoder approach applied to natural images: an auxiliary decoder network tries to reconstruct the original input from the internal representation. The autoencoder will try to preserve all the details needed for reconstructing the image at pixel level, even though classification is typically invariant to all kinds of transformations which do not preserve pixel values. Most of the information required for pixel-level reconstruction is irrelevant and takes space from the more relevant invariant features which, almost by definition, cannot alone be used for reconstruction.

Our approach follows Valpola (2015) who proposed a Ladder network where the auxiliary task is to denoise representations at every level of the model. The model structure is an autoencoder with skip connections from the encoder to decoder and the learning task is similar to that in denoising autoencoders but applied to every layer, not just inputs. The skip connections relieve the pressure to represent details at the higher layers of the model because, through the skip connections, the decoder can recover any details discarded by the encoder. Previously the Ladder network has only been demonstrated in unsupervised learning (Valpola, 2015; Rasmus *et al.*, 2015a) but we now combine it with supervised learning.

The key aspects of the approach are as follows:

Compatibility with supervised methods. The unsupervised part focuses on relevant details found by supervised learning. Furthermore, it can be added to existing feedforward neural networks, for example multi-layer perceptrons (MLPs) or convolutional neural networks (Section 3). We show that we can take a state-of-the-art supervised learning method as a starting point and improve the network further by adding simultaneous unsupervised learning (Section 4).

Scalability due to local learning. In addition to supervised learning target at the top layer, the model has local unsupervised learning targets on every layer making it suitable for very deep neural networks. We demonstrate this with two deep supervised network architectures.

Computational efficiency. The encoder part of the model corresponds to normal supervised learning. Adding a decoder, as proposed in this paper, approximately triples the computation during training but not necessarily the training time since the same result can be achieved faster due to better utilization of available information. Overall, computation per update scales similarly to whichever supervised learning approach is used, with a small multiplicative factor.

As explained in Section 2, the skip connections and layer-wise unsupervised targets effectively turn autoencoders into hierarchical latent variable models which are known to be well suited for semi-supervised learning. Indeed, we obtain state-of-the-art results in semisupervised learning in MNIST, permutation invariant MNIST and CIFAR-10 classification tasks (Section 4). However, the improvements are not limited to semi-supervised settings: for the permutation invariant MNIST task, we also achieve a new record with the normal full-labeled setting.[1]

2 Derivation and justification

Latent variable models are an attractive approach to semi-supervised learning because they can combine supervised and unsupervised learning in a principled way. The only difference is whether the class labels are observed or not. This approach was taken, for instance, by Goodfellow *et al.* (2013a) with their multi-prediction deep Boltzmann machine. A particularly attractive property of hierarchical latent variable models is that they can, in general, leave the details for the lower levels to represent, allowing higher levels to focus on more invariant, abstract features that turn out to be relevant for the task at hand.

The training process of latent variable models can typically be split into inference and learning, that is, finding the posterior probability of the unobserved latent variables and then updating the underlying probability model to better fit the observations. For instance, in the expectation-maximization (EM) algorithm, the E-step corresponds to finding the expectation of the latent variables over the posterior distribution assuming the model fixed and M-step then maximizes the underlying probability model assuming the expectation fixed.

In general, the main problem with latent variable models is how to make inference and learning efficient. Suppose there are layers l of latent variables $\mathbf{z}^{(l)}$. Typically latent variable models represent the probability distribution of all the variables explicitly as a product of terms, such as $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ in directed graphical models. The inference process and model updates are then derived from Bayes' rule, typically as some kind of approximation. Often the inference is iterative as it is generally impossible to solve the resulting equations in a closed form as a function of the observed variables.

There is a close connection between denoising and probabilistic modeling. On the one hand, given a probabilistic model, you can compute the optimal denoising. Say you want to reconstruct a latent z using a prior $p(z)$ and an observation $\tilde{z} = z + \text{noise}$. We first compute the posterior distribution $p(z \mid \tilde{z})$, and use its center of gravity as the reconstruction $\hat{z}$. One can show that this minimizes the expected denoising cost $(\hat{z} - z)^2$. On the other hand, given a denoising function, one can draw samples from the corresponding distribution by creating a Markov chain that alternates between corruption and denoising (Bengio *et al.*, 2013).

[1] Preliminary results on the full-labeled setting on permutation invariant MNIST task were reported in a short early version of this paper (Rasmus *et al.*, 2015b). Compared to that, we have added noise to all layers of the model and further simplified the denoising function g. This further improved the results.

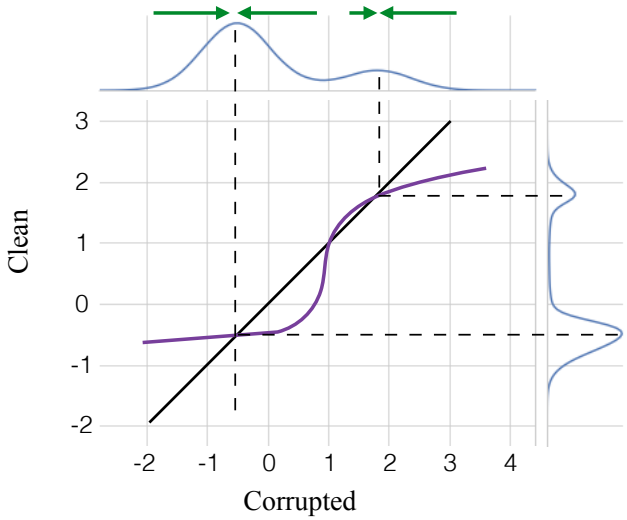


Figure 1: A depiction of an optimal denoising function for a bimodal distribution. The input for the function is the corrupted value (x axis) and the target is the clean value (y axis). The denoising function moves values towards higher probabilities as show by the green arrows.

Valpola (2015) proposed the Ladder network where the inference process itself can be learned by using the principle of denoising which has been used in supervised learning (Sietsma and Dow, 1991), denoising autoencoders (dAE) (Vincent *et al.*, 2010) and denoising source separation (DSS) (Särelä and Valpola, 2005) for complementary tasks. In dAE, an autoencoder is trained to reconstruct the original observation $\mathbf{x}$ from a corrupted version $\tilde{\mathbf{x}}$. Learning is based simply on minimizing the norm of the difference of the original $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$ from the corrupted $\tilde{\mathbf{x}}$, that is the cost is $\|\hat{\mathbf{x}} - \mathbf{x}\|^2$.

While dAEs are normally only trained to denoise the observations, the DSS framework is based on the idea of using denoising functions $\hat{\mathbf{z}} = g(\mathbf{z})$ of latent variables $\mathbf{z}$ to train a mapping $\mathbf{z} = f(\mathbf{x})$ which models the likelihood of the latent variables as a function of the observations. The cost function is identical to that used in a dAE except that latent variables $\mathbf{z}$ replace the observations $\mathbf{x}$, that is, the cost is $\|\hat{\mathbf{z}} - \mathbf{z}\|^2$. The only thing to keep in mind is that $\mathbf{z}$ needs to be normalized somehow as otherwise the model has a trivial solution at $\mathbf{z} = \hat{\mathbf{z}} =$ constant. In a dAE, this cannot happen as the model cannot change the input $\mathbf{x}$.

Figure 1 depicts the optimal denoising function $\hat{z} = g(\tilde{z})$ for a one-dimensional bimodal distribution which could be the distribution of a latent variable inside a larger model. The shape of the denoising function depends on the distribution of z and the properties of the corruption noise. With no noise at all, the optimal denoising function would be a straight line. In general, the denoising function pushes the values towards higher probabilities as shown by the green arrows.

Figure 2 shows the structure of the Ladder network. Every layer contributes to the cost function a term $C_d^{(l)} = \|\mathbf{z}^{(l)} - \hat{\mathbf{z}}^{(l)}\|^2$ which trains the layers above (both encoder and decoder) to learn the denoising function $\hat{\mathbf{z}}^{(l)} = g^{(l)}(\tilde{\mathbf{z}}^{(l)})$ which maps the corrupted $\tilde{\mathbf{z}}^{(l)}$ onto the denoised estimate $\hat{\mathbf{z}}^{(l)}$. As the estimate $\hat{\mathbf{z}}^{(l)}$ incorporates all the prior knowledge about $\mathbf{z}$, the same cost function term also trains the encoder layers below to find cleaner features which better match the prior expectation.

Since the cost function needs both the clean $\mathbf{z}^{(l)}$ and corrupted $\tilde{\mathbf{z}}^{(l)}$, the encoder is run twice: a clean pass for $\mathbf{z}^{(l)}$ and a corrupted pass for $\tilde{\mathbf{z}}^{(l)}$. Another feature which differentiates the Ladder network from regular dAEs is that each layer has a skip connection between the encoder and decoder. This feature mimics the inference structure of latent variable models and makes it possible for the higher levels of the network to leave some of the details for lower levels to represent. Rasmus *et al.* (2015a) showed that such skip connections allow dAEs to focus on abstract invariant features on the higher levels, making the Ladder network a good fit with supervised learning that can select which information is relevant for the task at hand.

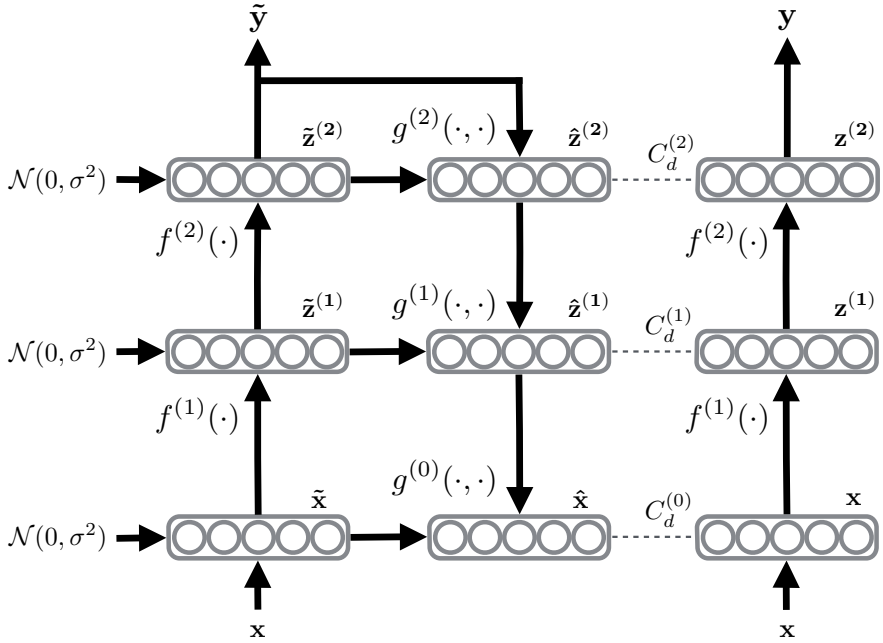


Figure 2: A conceptual illustration of the Ladder network when $L = 2$. The feedforward path $\mathbf{x} \to \mathbf{z}^{(1)} \to \mathbf{z}^{(2)} \to \mathbf{y}$) shares the mappings $f^{(l)}$ with the corrupted feedforward path, or encoder ($\mathbf{x} \to \tilde{\mathbf{z}}^{(1)} \to \tilde{\mathbf{z}}^{(2)} \to \tilde{\mathbf{y}}$). The decoder ($\tilde{\mathbf{z}}^{(l)} \to \hat{\mathbf{z}}^{(l)} \to \hat{\mathbf{x}}$) consists of denoising functions $g^{(l)}$ and has costs functions $C_d^{(l)}$ on each layer trying to minimize the difference between $\hat{\mathbf{z}}^{(l)}$ and $\mathbf{z}^{(l)}$. The output $\mathbf{y}$ of the encoder can also be trained using supervised learning.

Algorithm 1 Calculation of the output and cost function of the Ladder network

Require: $\mathbf{x}(n)$
  # Corrupted encoder and classifier
  $\tilde{\mathbf{h}}^{(0)} \leftarrow \tilde{\mathbf{z}}^{(0)} \leftarrow \mathbf{x}(n) + \texttt{noise}$
  for l = 1 **to** L **do**
    $\tilde{\mathbf{z}}_{\text{pre}}^{(l)} \leftarrow \mathbf{W}^{(l)} \tilde{\mathbf{h}}^{(l-1)}$
    $\tilde{\boldsymbol{\mu}}^{(l)} \leftarrow \texttt{batchmean}(\tilde{\mathbf{z}}_{\text{pre}}^{(l)})$
    $\tilde{\boldsymbol{\sigma}}^{(l)} \leftarrow \texttt{batchstd}(\tilde{\mathbf{z}}_{\text{pre}}^{(l)})$
    $\tilde{\mathbf{z}}^{(l)} \leftarrow \texttt{batchnorm}(\tilde{\mathbf{z}}_{\text{pre}}^{(l)}) + \texttt{noise}$
    $\tilde{\mathbf{h}}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  $P(\tilde{\mathbf{y}} \mid \mathbf{x}) \leftarrow \tilde{\mathbf{h}}^{(L)}$
  # Clean encoder (for denoising targets)
  $\mathbf{h}^{(0)} \leftarrow \mathbf{z}^{(0)} \leftarrow \mathbf{x}(n)$
  for l = 1 **to** L **do**
    $\mathbf{z}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)})$
    $\mathbf{h}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)}))$
  end for
  # Final classification:
  $P(\mathbf{y} \mid \mathbf{x}) \leftarrow \mathbf{h}^{(L)}$
  # Decoder and denoising
  for l = L **to** 0 **do**
    if l = L **then**
      $\mathbf{u}^{(L)} \leftarrow \texttt{batchnorm}(\tilde{\mathbf{h}}^{(L)})$
    else
      $\mathbf{u}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{V}^{(l)} \hat{\mathbf{z}}^{(l+1)})$
    end if
    $\forall i : \hat{z}_i^{(l)} \leftarrow g(\tilde{z}_i^{(l)}, u_i^{(l)})$ # Eq. (1)
    $\forall i : \hat{z}_{i,\text{BN}}^{(l)} \leftarrow \frac{\hat{z}_i^{(l)} - \tilde{\mu}_i^{(l)}}{\tilde{\sigma}_i^{(l)}}$
  end for
  # Cost function C for training:
  $\text{C} \leftarrow 0$
  if $t(n)$ **then**
    $\text{C} \leftarrow -\log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x})$
  end if
  $\text{C} \leftarrow \text{C} + \sum_{l=1}^{L} \lambda_l \left\| \mathbf{z}^{(l)} - \hat{\mathbf{z}}_{\text{BN}}^{(l)} \right\|^2$ # Eq. (2)

One way to picture the Ladder network is to consider it as a collection of nested denoising autoencoders which share parts of the denoising machinery between each other. From the viewpoint of the autoencoder at layer l, the representations on the higher layers can be treated as hidden neurons. In other words, there is no particular reason why $\hat{\mathbf{z}}^{(l+i)}$ produced by the decoder should resemble the corresponding representations $\mathbf{z}^{(l+i)}$ produced by the encoder. It is only the cost function $C_d^{(l+i)}$ that ties these together and forces the inference to proceed in a reverse order in the decoder. This sharing helps a deep denoising autoencoder to learn the denoising process as it splits the task into meaningful sub-tasks of denoising intermediate representations.

3 Implementation of the Model

The steps to implement the Ladder network (Section 3.1) are typically as follows: 1) take a feedforward model which serves supervised learning and as the encoder (Section 3.2), 2) add a decoder which can invert the mappings on each layer of the encoder and supports unsupervised learning (Section 3.3), and 3) train the whole Ladder network by minimizing a sum of all the cost function terms.

In this section, we will go through these steps in detail for a fully connected MLP network and briefly outline the modifications required for convolutional networks, both of which are used in our experiments (Section 4).

3.1 General Steps for Implementing the Ladder Network

Consider training a classifier[2], or a mapping from input $\mathbf{x}$ to output y with targets t, from a training set of pairs $\{\mathbf{x}(n), t(n) \mid 1 \leq n \leq N\}$. Semi-supervised learning (Chapelle *et al.*, 2006) studies how auxiliary unlabeled data $\{\mathbf{x}(n) \mid N+1 \leq n \leq M\}$ can help in training a classifier. It is often the case that labeled data is scarce whereas unlabeled data is plentiful, that is $N \ll M$.

The Ladder network can improve results even without auxiliary unlabeled data but the original motivation was to make it possible to take well-performing feedforward classifiers and augment them with an auxiliary decoder as follows:

1. Train any standard feedforward neural network. The network type is not limited to standard MLPs, but the approach can be applied, for example, to convolutional or recurrent networks. This will be the encoder part of the Ladder network.
2. For each layer, analyze the conditional distribution of representations given the layer above, $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$. The observed distributions could resemble for example Gaussian distributions where the mean and variance depend on the values $\mathbf{z}^{(l+1)}$, bimodal distributions where the relative probability masses of the modes depend on the values $\mathbf{z}^{(l+1)}$, and so on.
3. Define a function $\hat{\mathbf{z}}^{(l)} = g(\tilde{\mathbf{z}}^{(l)}, \hat{\mathbf{z}}^{(l+1)})$ which can approximate the optimal denoising function for the family of observed distributions. The function g is therefore expected to form a reconstruction $\hat{\mathbf{z}}^{(l)}$ that resembles the clean $\mathbf{z}^{(l)}$ given the corrupted $\tilde{\mathbf{z}}^{(l)}$ and the higher level reconstruction $\hat{\mathbf{z}}^{(l+1)}$.
4. Train the whole network in a full-labeled or semi-supervised setting using standard optimization techniques such as stochastic gradient descent.

3.2 Fully Connected MLP as Encoder

As a starting point we use a fully connected MLP network with rectified linear units. We follow Ioffe and Szegedy (2015) to apply batch normalization to each preactivation including the topmost layer in the L-layer network. This serves two purposes. First, it improves convergence due to reduced covariate shift as originally proposed by Ioffe and Szegedy (2015). Second, as explained in Section 2, DSS-type cost functions for all but the input layer require some type of normalization to prevent the denoising cost from encouraging the trivial solution where the encoder outputs just

[2]Here we only consider the case where the output $t(n)$ is a class label but it is trivial to apply the same approach to other regression tasks.

constant values as these are the easiest to denoise. Batch normalization conveniently serves this purpose, too.

Formally, batch normalization for layers $l = 1 \dots L$ is implemented as

$$\mathbf{z}^{(l)} = \mathrm{N_B}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)})$$
$$\mathbf{h}^{(l)} = \phi\big(\boldsymbol{\gamma}^{(l)}(\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)})\big),$$

where $\mathbf{h}^{(0)} = \mathbf{x}$, $\mathrm{N_B}$ is a component-wise batch normalization $\mathrm{N_B}(x_i) = (x_i - \hat{\mu}_{x_i})/\hat{\sigma}_{x_i}$, where $\hat{\mu}_{x_i}$ and $\hat{\sigma}_{x_i}$ are estimates calculated from the minibatch, $\boldsymbol{\gamma}^{(l)}$ and $\boldsymbol{\beta}^{(l)}$ are trainable parameters, and $\phi(\cdot)$ is the activation function such as the rectified linear unit (ReLU) for which $\phi(\cdot) = \max(0, \cdot)$. For outputs $\mathbf{y} = \mathbf{h}^{(L)}$ we always use the softmax activation. For some activation functions the scaling parameter $\boldsymbol{\beta}^{(l)}$ or the bias $\boldsymbol{\gamma}^{(l)}$ are redundant and we only apply them in non-redundant cases. For example, the rectified linear unit does not need scaling, linear activation function needs neither scaling nor bias, but softmax requires both.

As explained in Section 2 and shown in Figure 2, the Ladder network requires two forward passes, clean and corrupted, which produce clean $\mathbf{z}^{(l)}$ and $\mathbf{h}^{(l)}$ and corrupted $\tilde{\mathbf{z}}^{(l)}$ and $\tilde{\mathbf{h}}^{(l)}$, respectively. We implemented corruption by adding isotropic Gaussian noise $\mathbf{n}$ to inputs and after each batch normalization:

$$\begin{aligned}
\tilde{\mathbf{x}} &= \tilde{\mathbf{h}}^{(0)} = \mathbf{x} + \mathbf{n}^{(0)} \\
\tilde{\mathbf{z}}^{(l)}_{\mathrm{pre}} &= \mathbf{W}^{(l)}\tilde{\mathbf{h}}^{(l-1)} \\
\tilde{\mathbf{z}}^{(l)} &= \mathrm{N_B}(\tilde{\mathbf{z}}^{(l)}_{\mathrm{pre}}) + \mathbf{n}^{(l)} \\
\tilde{\mathbf{h}}^{(l)} &= \phi\big(\boldsymbol{\gamma}^{(l)}(\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)})\big).
\end{aligned}$$

Note that we collect the value $\tilde{\mathbf{z}}^{(l)}_{\mathrm{pre}}$ here because it will be needed in the decoder cost function in Section 3.3.

The supervised cost C_{c} is the average negative log probability of the noisy output $\tilde{\mathbf{y}}$ matching the target $t(n)$ given the inputs $\mathbf{x}(n)$

$$C_{\mathrm{c}} = -\frac{1}{N}\sum_{n=1}^{N} \log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x}(n)).$$

In other words, we also use the noise to regularize supervised learning.

We saw networks with this structure reach close to state-of-the-art results in purely supervised learning (e.g., see Table 1) which makes them good starting points for improvement via semi-supervised learning by adding an auxiliary unsupervised task.

3.3 Decoder for Unsupervised Learning

When designing a suitable decoder to support unsupervised learning, we followed the steps outlined in Section 3.1: we analyzed the histograms of the activations of hidden neurons in well-performing models trained by supervised learning and then designed denoising functions $\hat{\mathbf{z}}^{(l)} = g(\tilde{\mathbf{z}}^{(l)}, \hat{\mathbf{z}}^{(l+1)})$ which are able to approximate the optimal denoising function that provides the best estimate of the clean version $\mathbf{z}^{(l)}$. In this section, we present just the end result of the analysis but more details can be found in Appendix B.

The encoder provides multiple targets which the decoder could try to denoise. We chose the target to be the batch-normalized projections $\mathbf{z}^{(l)}$ before the activation function is applied. As mentioned earlier, batch-normalization of the target prevents the encoder from collapsing the representation to a constant. Such a solution minimizes denoising error but is obviously not the one we are looking for. On the other hand, it makes sense to target the encoder representation before the activation function is applied. This is because the activation function is typically the step where information is lost, for example, due to saturation or pooling. In the Ladder network, the decoder can recover this lost information through the lateral skip connections from the encoder to the decoder.

The network structure of the decoder does not need to resemble the decoder but in order to keep things simple, we chose a decoder structure whose weight matrices $\mathbf{V}^{(l+1)}$ have shapes similar to

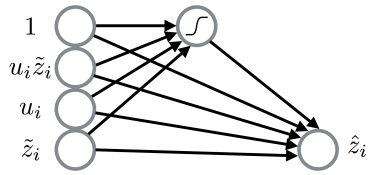


Figure 3: Each hidden neuron is denoised using a miniature MLP network with two inputs, u_i and $\tilde{z}_i$ and a single output $\hat{z}_i$. The network has a single sigmoidal unit, skip connections from input to output and input augmented by the product $u_i\hat{z}_i$. The total number of parameters is therefore 9 per each hidden neuron

the weight matrices $\mathbf{W}^{(l)}$ of the encoder, except being transposed, and which performs the denoising neuron-wise. This keeps the number of parameters of the decoder in check but still allows the network to represent any distributions: any dependencies between hidden neurons can still be represented through the higher levels of the network which effectively implement a higher-level denoising autoencoder.

In practice the neuron-wise denoising function g is implemented by first computing a vertical mapping from $\hat{\mathbf{z}}^{(l+1)}$ and then batch normalizing the resulting projections:

$$\mathbf{u}^{(l)} = \mathrm{N_B}(\mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)})\,,$$

where the matrix $\mathbf{V}^{(l)}$ has the same dimension as the transpose of $\mathbf{W}^{(l)}$ on the encoder side. The projection vector $\mathbf{u}^{(l)}$ has the same dimensionality as $\mathbf{z}^{(l)}$ which means that the final denoising can be applied neuro-wise with a miniature MLP network that takes two inputs, $\tilde{z}_i^{(l)}$ and $u_i^{(l)}$, and outputs the denoised estimate $\hat{z}_i^{(l)}$:

$$\hat{z}_i^{(l)} = g_i^{(l)}(\tilde{z}_i^{(l)}, u_i^{(l)})\,.$$

Note the slight abuse of notation here since $g_i^{(l)}$ is now a function of scalars $\tilde{z}_i^{(l)}$ and $u_i^{(l)}$ rather than the full vectors $\tilde{\mathbf{z}}^{(l)}$ and $\hat{\mathbf{z}}^{(l+1)}$.

Figure 3 illustrates the structure of the miniature MLPs which are parametrized as follows:

$$\hat{z}_i^{(l)} = g_i^{(l)}(\tilde{z}_i^{(l)}, u_i^{(l)}) = \mathbf{a}_i^{(l)}\boldsymbol{\xi}_i^{(l)} + b_i^{(l)}\mathtt{sigmoid}(\mathbf{c}_i^{(l)}\boldsymbol{\xi}_i^{(l)}) \tag{1}$$

where $\xi_i^{(l)} = [1, \tilde{z}_i^{(l)}, u_i^{(l)}, \tilde{z}_i^{(l)}u_i^{(l)}]^T$ is the augmented input, $\mathbf{a}_i^{(l)}$ and $\mathbf{c}_i^{(l)}$ are trainable 1×4 weight vectors, and $b_i^{(l)}$ is a trainable weight. In other words, each hidden neuron of the network has its own miniature MLP network with 9 parameters. While this makes the number of parameters in the decoder slightly higher than in the encoder, the difference is insignificant as most of the parameters are in the vertical projection mappings $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$ which have the same dimensions (apart from transposition).

For the lowest layer, $\hat{\mathbf{x}} = \hat{\mathbf{z}}^{(0)}$ and $\tilde{\mathbf{x}} = \tilde{\mathbf{z}}^{(0)}$ by definition, and for the highest layer we chose $\mathbf{u}^{(L)} = \tilde{\mathbf{y}}$. This allows the highest-layer denoising function to utilize prior information about the classes being mutually exclusive which seems to improve convergence in cases where there are very few labeled samples.

The proposed parametrization is capable of learning denoising of several different distributions including sub- and super-Gaussian and bimodal distributions. This means that the decoder supports sparse coding and independent component analysis.[3] The parametrization also allows the distribution to be modulated by $\mathbf{z}^{(l+1)}$ through $\mathbf{u}^{(l)}$, encouraging the decoder to find representations $\mathbf{z}^{(l)}$ that

[3]For more details on how denoising functions represent corresponding distributions see Valpola (2015, Section 4.1).

have high mutual information with $\mathbf{z}^{(l+1)}$. This is crucial as it allows supervised learning to have an indirect influence on the representations learned by the unsupervised decoder: any abstractions selected by supervised learning will bias the lower levels to find more representations which carry information about the same thing.

Rasmus *et al.* (2015a) showed that modulated connections in g are crucial for allowing the decoder to recover discarded details from the encoder and thus for allowing invariant representations to develop. The proposed parametrization can represent such modulation but also standard additive connections that are normally used in dAEs. We also tested alternative formulations for the denoising function, the results of which can be found in Appendix B.

The cost function for the unsupervised path is the mean squared reconstruction error per neuron, but there is a slight twist which we found to be important. Batch normalization has useful properties as noted in Section 3.2, but it also introduces noise which affects both the clean and corrupted encoder pass. This noise is highly correlated between $\mathbf{z}^{(l)}$ and $\tilde{\mathbf{z}}^{(l)}$ because the noise derives from the statistics of the samples that happen to be in the same minibatch. This highly correlated noise in $\mathbf{z}^{(l)}$ and $\tilde{\mathbf{z}}^{(l)}$ biases the denoising functions to be simple copies[4] $\hat{\mathbf{z}}^{(l)} \approx \tilde{\mathbf{z}}^{(l)}$.

The solution we found was to implicitly use the projections $\mathbf{z}_{\mathrm{pre}}^{(l)}$ as the target for denoising and scale the cost function in such a way that the term appearing in the error term is the batch normalized $\mathbf{z}^{(l)}$ instead. For the moment, let us see how that works for a scalar case:

$$\begin{aligned}
\frac{1}{\sigma^2}\left\|z_{\mathrm{pre}} - \hat{z}\right\|^2 &= \left\|\frac{z_{\mathrm{pre}} - \mu}{\sigma} - \frac{\hat{z} - \mu}{\sigma}\right\|^2 = \left\|z - \hat{z}_{\mathrm{BN}}\right\|^2 \\
z &= \mathrm{N_B}(z_{\mathrm{pre}}) = \frac{z_{\mathrm{pre}} - \mu}{\sigma} \\
\hat{z}_{\mathrm{BN}} &= \frac{\hat{z} - \mu}{\sigma},
\end{aligned}$$

where μ and σ are the batch mean and batch std of z_{pre}, respectively, that were used in batch normalizing z_{pre} into z. The unsupervised denoising cost function C_{d} is thus

$$C_{\mathrm{d}} = \sum_{l=1}^{L} \lambda_l C_{\mathrm{d}}^{(l)} = \sum_{l=0}^{L} \frac{\lambda_l}{N m_l} \sum_{n=1}^{N} \left\|\mathbf{z}^{(l)}(n) - \hat{\mathbf{z}}_{\mathrm{BN}}^{(l)}(n)\right\|^2, \tag{2}$$

where m_l is the layer's width, N the number of training samples, and the hyperparameter λ_l a layer-wise multiplier determining the importance of the denoising cost.

The model parameters $\mathbf{W}^{(l)}, \boldsymbol{\gamma}^{(l)}, \boldsymbol{\beta}^{(l)}, \mathbf{V}^{(l)}, \mathbf{a}_i^{(l)}, b_i^{(l)}, \mathbf{c}_i^{(l)}$ can be trained simply by using the back-propagation algorithm to optimize the total cost $C = C_{\mathrm{c}} + C_{\mathrm{d}}$. The feedforward pass of the full Ladder network is listed in Algorithm 1. Classification results are read from the $\mathbf{y}$ in the clean feedforward path.

3.4 Variations

Section 3.3 detailed how to build a decoder for the Ladder network to match the fully connected encoder described in 3.2. It is easy to extend the same approach to other encoders, for instance, convolutional neural networks (CNN). For the decoder of fully connected networks we used vertical mappings whose shape is a transpose of the encoder mapping. The same treatment works for the convolution operations: in the networks we have tested in this paper, the decoder has convolutions whose parametrization mirrors the encoder and effectively just reverses the flow of information. As the idea of convolution is to reduce the number of parameters by weight sharing, we applied this to the parameters of the denoising function g, too.

Many convolutional networks use pooling operations with stride, that is, they downsample the spatial feature maps. The decoder needs to compensate this with a corresponding upsampling. There are several alternative ways to implement this and in this paper we chose the following options: 1)

[4]The whole point of using *denoising* autoencoders rather than regular autoencoders is to prevent skip connections from short-circuiting the decoder and force the decoder to learn meaningful abstractions which help in denoising.

on the encoder side, pooling operations are treated as separate layers with their own batch normalization and linear activations function and 2) the downsampling of the pooling on the encoder side is compensated by upsampling with copying on the decoder side. This provides multiple targets for the decoder to match, helping the decoder to recover the information lost on the encoder side.

It is worth noting that a simple special case of the decoder is a model where $\lambda_l = 0$ when $l < L$. This corresponds to a denoising cost only on the top layer and means that most of the decoder can be omitted. This model, which we call the Γ-model due to the shape of the graph, is useful as it can easily be plugged into any feedforward network without decoder implementation. In addition, the Γ-model is the same for MLPs and convolutional neural networks. The encoder in the Γ-model still includes both the clean and the corrupted paths as in the full ladder.

4 Experiments

With the experiments with MNIST and CIFAR-10 dataset, we wanted to compare our method to other semi-supervised methods but also to show that we can attach the decoder both to a fully-connected MLP network and to a convolutional neural network, both of which were described in Section 3. We also wanted to compare the performance of the simpler Γ-model (Sec. 3.4) to the full Ladder network and experimented with only having a cost function on the input layer. With CIFAR-10, we only tested the Γ-model.

We also measured the performance of the supervised baseline models which only included the encoder and the supervised cost function. In all cases where we compared these directly with Ladder networks, we did our best to optimize the hyperparameters and regularization of the baseline supervised learning models so that any improvements could not be explained, for example, by the lack of suitable regularization which would then have been provided by the denoising costs.

With the convolutional networks, our focus was exclusively on semi-supervised learning. The supervised baselines for all labels only intend to show that the performance of the selected network architectures are in line with the ones reported in the literature. We make claims neither about the optimality nor the statistical significance of these baseline results.

We used the Adam optimization algorithm (Kingma and Ba, 2015) for weight updates. The learning rate was 0.002 for the first part of learning, followed by an annealing phase during which the learning rate was linearly decreased to zero. Minibatch size was 100. The source code for all the experiments is available at `https://github.com/arasmus/ladder` unless explicitly noted in the text.

4.1 MNIST dataset

For evaluating semi-supervised learning, we used the standard 10 000 test samples as a held-out test set and randomly split the standard 60 000 training samples into 10 000-sample validation set and used $M = 50\ 000$ samples as the training set. From the training set, we randomly chose $N = 100$, 1000, or all labels for the supervised cost.[5] All the samples were used for the decoder which does not need the labels. The validation set was used for evaluating the model structure and hyperparameters. We also balanced the classes to ensure that no particular class was over-represented. We repeated each training 10 times varying the random seed that was used for the splits.

After optimizing the hyperparameters, we performed the final test runs using all the $M = 60\ 000$ training samples with 10 different random initializations of the weight matrices and data splits. We trained all the models for 100 epochs followed by 50 epochs of annealing. With minibatch size of 100, this amounts to 75 000 weight updates for the validation runs and 90 000 for the final test runs.

[5] In all the experiments, we were careful not to optimize any parameters, hyperparameters, or model choices based on the results on the held-out test samples. As is customary, we used 10 000 labeled validation samples even for those settings where we only used 100 labeled samples for training. Obviously this is not something that could be done in a real case with just 100 labeled samples. However, MNIST classification is such an easy task even in the permutation invariant case that 100 labeled samples there correspond to a far greated number of labeled samples in many other datasets.

Test error % with # of used labels	100	1000	All
Semi-sup. Embedding (Weston *et al.*, 2012)	16.86	5.73	1.5
Transductive SVM (from Weston *et al.*, 2012)	16.81	5.38	1.40*
MTC (Rifai *et al.*, 2011b)	12.03	3.64	0.81
Pseudo-label (Lee, 2013)	10.49	3.46	
AtlasRBF (Pitelis *et al.*, 2014)	8.10 ($\pm$ 0.95)	3.68 ($\pm$ 0.12)	1.31
DGN (Kingma *et al.*, 2014)	3.33 ($\pm$ 0.14)	2.40 ($\pm$ 0.02)	0.96
DBM, Dropout (Srivastava *et al.*, 2014)			0.79
Adversarial (Goodfellow *et al.*, 2015)			0.78
Virtual Adversarial (Miyato *et al.*, 2015)	2.66	1.50	0.64 ($\pm$ 0.03)
Baseline: MLP, BN, Gaussian noise	21.74 ($\pm$ 1.77)	5.70 ($\pm$ 0.20)	0.80 ($\pm$ 0.03)
Γ-model (Ladder with only top-level cost)	4.34 ($\pm$ 2.31)	1.71 ($\pm$ 0.07)	0.79 ($\pm$ 0.05)
Ladder, only bottom-level cost	1.38 ($\pm$0.49)	1.07 ($\pm$ 0.06)	**0.61** ($\pm$ 0.05)
Ladder, full	**1.13** ($\pm$ 0.04)	**1.00** ($\pm$ 0.06)	

Table 1: A collection of previously reported MNIST test errors in the permutation invariant setting followed by the results with the Ladder network. * = SVM. Standard deviation in parenthesis.

4.1.1 Fully-connected MLP

A useful test for general learning algorithms is the permutation invariant MNIST classification task. Permutation invariance means that the results need to be invariant with respect to permutation of the elements of the input vector. In other words, one is not allowed to use prior information about the spatial arrangement of the input pixels. This excludes, among others, convolutional networks and geometric distortions of the input images.

We chose the layer sizes of the baseline model somewhat arbitrarily to be 784-1000-500-250-250-250-10. The network is deep enough to demonstrate the scalability of the method but not yet an overkill for MNIST.

The hyperparameters we tuned for each model are the noise level that is added to the inputs and to each layer, and denoising cost multipliers $\lambda^{(l)}$. We also ran the supervised baseline model with various noise levels. For models with just one cost multiplier, we optimized them with a search grid $\{\ldots, 0.1, 0.2, 0.5, 1, 2, 5, 10, \ldots\}$. Ladder networks with cost function on all layers have a much larger search space and we explored it much more sparsely. For instance, the optimal model we found for $N = 100$ labels had $\lambda^{(0)} = 1000$, $\lambda^{(1)} = 10$ and $\lambda^{(\geq 2)} = 0.1$. A good value for the std of Gaussian corruption noise $\mathbf{n}^{(l)}$ was mostly 0.3 but with $N = 1000$ labels, a better value was 0.2. According to validation data, denoising costs above the input layer were not helpful with $N = 50\,000$ labels so we only tested the bottom model with all $N = 60\,000$ labels. For the complete set of selected denoising cost multipliers and other hyperparameters, please refer to the code.

The results presented in Table 1 show that the proposed method outperforms all the previously reported results. The improvement is most significant in the most difficult 100-label case where denoising targets on all layers provides the greatest benefit over having a denoising target only on the input layer. This suggests that the improvement can be attributed to efficient unsupervised learning on all the layers of the Ladder network. Encouraged by the good results, we also tested with $N = 50$ labels and got a test error of 1.39 % ($\pm$ 0.55 %).

The simple Γ-model also performed surprisingly well, particularly for $N = 1000$ labels. The denoising cost at the highest layer turns out to encourage distributions with two sharp peaks. While this clearly allows the model to utilize information in unlabeled samples, effectively self-labeling them, it also seems to suffer from confirmation bias, particularly with less labels. While the median error with $N = 100$ labels is 2.61 %, the average is significantly worse due to some runs in which the model seems to get stuck with its inital misconception about the classes. The Ladder network with denoising targets on every layer converges much more reliably as can be seen from the low standard deviation of the results.

Test error without data augmentation % with # of used labels	100	all
EmbedCNN (Weston *et al.*, 2012)	7.75	
SWWAE (Zhao *et al.*, 2015)	9.17	0.71
Baseline: Conv-Small, supervised only	6.43 ($\pm$ 0.84)	0.36
Conv-FC	0.91 ($\pm$ 0.14)	
Conv-Small, Γ-model	**0.86** ($\pm$ 0.41)	

Table 2: CNN results for MNIST

4.1.2 Convolutional networks

We tested two convolutional networks for the general MNIST classification task but omitted data augmentation such as geometric distortions. We focused on the 100-label case since with more labels the results were already so good even in the more difficult permutation invariant task.

The first network was a straight-forward extension of the fully-connected network tested in the permutation invariant case. We turned the first fully connected layer into a convolution with 26-by-26 filters, resulting in a 3-by-3 spatial map of 1000 features. Each of the 9 spatial locations was processed independently by a network with the same structure as in the previous section, finally resulting in a 3-by-3 spatial map of 10 features. These were pooled with a global mean-pooling layer. Essentially we thus convolved the image with the complete fully-connected network. Depooling on the top-most layer and deconvolutions on the layers below were implemented as described in Section 3.4. Since the internal structure of each of the 9 almost independent processing paths was the same as in the permutation invariant task, we used the same hyperparameters that were optimal for the permutation invariant task. In Table 2, this model is referred to as Conv-FC.

With the second network, which was inspired by ConvPool-CNN-C from Springenberg *et al.* (2014), we only tested the Γ-model. The MNIST classification task can typically be solved with a smaller number of parameters than CIFAR-10 for which this topology was originally developed, so we modified the network by removing layers and reducing the amount of parameters in the remaining layers. In addition, we observed that adding a small fully connected layer having 10 neurons on top of the global mean pooling layer improved the results in the semi-supervised task. We did not tune other parameters than the noise level, which was chosen from $\{0.3, 0.45, 0.6\}$ using the validation set. The exact architecture of this network is detailed in Table 4 in Appendix A. It is referred to as Conv-Small since it is a smaller version of the network used for CIFAR-10 dataset.

The results in Table 2 confirm that even the single convolution on the bottom level improves the results over the fully connected network. More convolutions improve the Γ-model significantly although the high variance of the results suggests that the model still suffers from confirmation bias. The Ladder network with denoising targets on every level converges much more reliably. Taken together, these results suggest that combining the generalization ability of convolutional networks[6] and efficient unsupervised learning of the full Ladder network would have resulted in even better performance but this was left for future work.

4.2 Convolutional networks on CIFAR-10

CIFAR-10 dataset consists of small 32-by-32 RGB images from 10 classes. There are 50 000 labeled samples for training and 10 000 for testing. Like the MNIST dataset, it has been used for testing semi-supervised learning so we decided to test the simple Γ-model with a convolutional network that has reported to perform well in the standard supervised setting with all labels. We tested a few model architectures and selected ConvPool-CNN-C by Springenberg *et al.* (2014). We also evaluated the strided convolutional version by Springenberg *et al.* (2014), and while it performed well with all labels, we found that the max-pooling version overfitted less with fewer labels, and thus used it.

The main differences to ConvPool-CNN-C are the use of Gaussian noise instead of dropout and the convolutional per-channel batch normalization following Ioffe and Szegedy (2015). While dropout

[6] In general, fully convolutional networks excel in MNIST classification task. The performance of the fully supervised Conv-Small with all labels is in line with the literature and is provided as a rough reference only (only one run, no attempts to optimize, not available in the code package).

Test error % with # of used labels	4 000	All
All-Convolutional ConvPool-CNN-C (Springenberg *et al.*, 2014)		9.31
Spike-and-Slab Sparse Coding (Goodfellow *et al.*, 2012)	31.9	
Baseline: Conv-Large, supervised only	23.33 ($\pm$ 0.61)	9.27
Conv-Large, Γ-model	**20.09** ($\pm$ 0.46)	

Table 3: Test results for CNN on CIFAR-10 dataset without data augmentation

was useful with all labels, it did not seem to offer any advantage over additive Gaussian noise with less labels. For a more detailed description of the model, please refer to model Conv-Large in Table 4.

While testing the model performance with a limited number of labeled samples ($N = 4\ 000$), we found out that the model over-fitted quite severely: training error for most samples decreased so much that the network did not effectively learn anything from them as the network was already very confident about their classification. The network was equally confident about validation samples even when they were misclassified. We noticed that we could regularize the network by stripping away the scaling parameter $\boldsymbol{\beta}^{(L)}$ from the last layer. This means that the variance of the input to the softmax is restricted to unity. We also used this setting with the corresponding Γ-model although the denoising target already regularizes the network significantly and the improvement was not as pronounced.

The hyperparameters (noise level, denoising cost multipliers and number of epochs) for all models were optimized using $M = 40\ 000$ samples for training and the remaining $10\ 000$ samples for validation. After the best hyperparameters were selected, the final model was trained with these settings on all the $M = 50\ 000$ samples. All experiments were run with with 5 different random initializations of the weight matrices and data splits. We applied global contrast normalization and whitening following Goodfellow *et al.* (2013b), but no data augmentation was used.

The results are shown in Table 3. The supervised reference was obtained with a model closer to the original ConvPool-CNN-C in the sense that dropout rather than additive Gaussian noise was used for regularization.[7] We spent some time in tuning the regularization of our fully supervised baseline model for $N = 4\ 000$ labels and indeed, its results exceed the previous state of the art. This tuning was important to make sure that the improvement offered by the denoising target of the Γ-model is not a sign of poorly regularized baseline model. Although the improvement is not as dramatic as with MNIST experiments, it came with a very simple addition to standard supervised training.

5 Related Work

Early works in semi-supervised learning (McLachlan, 1975; Titterington *et al.*, 1985) proposed an approach where inputs $\mathbf{x}$ are first assigned to clusters, and each cluster has its class label. Unlabeled data would affect the shapes and sizes of the clusters, and thus alter the classification result. This approach can be reinterpreted as input vectors being corrupted copies $\tilde{\mathbf{x}}$ of the ideal input vectors $\mathbf{x}$ (the cluster centers), and the classification mapping being split into two parts: first denoising $\tilde{\mathbf{x}}$ into $\mathbf{x}$ (possibly probabilistically), and then labeling $\mathbf{x}$.

It is well known (see, e.g., Zhang and Oles, 2000) that when training a probabilistic model that directly estimates $P(y \mid \mathbf{x})$, unlabeled data cannot help. One way to study this is to assign probabilistic labels $q(y_t) = P(y_t \mid \mathbf{x}_t)$ to unlabeled inputs $\mathbf{x}_t$ and try to train $P(y \mid \mathbf{x})$ using those labels: It can be shown (see, e.g., Raiko *et al.*, 2015, Eq. (31)) that the gradient will vanish. There are different ways of circumventing that phenomenon by adjusting the assigned labels $q(y_t)$. These are all related to the Γ-model.

Label propagation methods (Szummer and Jaakkola, 2003) estimate $P(y \mid \mathbf{x})$, but adjust probabilistic labels $q(y_t)$ based on the assumption that nearest neighbors are likely to have the same label. The labels start to propagate through regions with high density $P(\mathbf{x})$. The Γ-model implicitly assumes

[7] Same caveats hold for this fully supervised reference result for all labels as with MNIST: only one run, no attempts to optimize, not available in the code package.

that the labels are uniform in the vicinity of a clean input since corrupted inputs need to produce the same label. This produces a similar effect: The labels start to propagate through regions with high density $P(\mathbf{x})$. Weston *et al.* (2012) explored deep versions of label propagation.

Co-training (Blum and Mitchell, 1998) assumes we have multiple views on $\mathbf{x}$, say $\mathbf{x} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$. When we train classifiers for the different views, we know that even for the unlabeled data, the true label is the same for each view. Each view produces its own probabilistic labeling $q^{(j)}(y_t) = P(y_t \mid \mathbf{x}_t^{(j)})$ and their combination $q(y_t)$ can be fed to train the individual classifiers. If we interpret having several corrupted copies of an input as different views on it, we see the relationship to the proposed method.

Lee (2013) adjusts the assigned labels $q(y_t)$ by rounding the probability of the most likely class to one and others to zero. The training starts by trusting only the true labels and then gradually increasing the weight of the so called *pseudo-labels*. Similar scheduling could be tested with our Γ-model as it seems to suffer from confirmation bias. It may well be that the denoising cost which is optimal in the beginning of the learning is smaller than the optimal at later stages of learning.

Dosovitskiy *et al.* (2014) pre-train a convolutional network with unlabeled data by treating each clean image as its own class. During training, the image is corrupted by transforming its location, scaling, rotation, contrast, and color. This helps to find features that are invariant to the used transformations. Discarding the last classification layer and replacing it with a new classifier trained on real labeled data leads to surprisingly good experimental results.

There is an interesting connection between our Γ-model and the contractive cost used by Rifai *et al.* (2011a): a linear denoising function $\hat{z}_i^{(L)} = a_i \tilde{z}_i^{(L)} + b_i$, where a_i and b_i are parameters, turns the denoising cost into a stochastic estimate of the contractive cost. In other words, our Γ-model seems to combine clustering and label propagation with regularization by contractive cost.

Recently Miyato *et al.* (2015) achieved impressive results with a regularization method that is similar to the idea of contractive cost. They required the output of the network to change as little as possible close to the input samples. As this requires no labels, they were able to use unlabeled samples for regularization. While their semi-supervised results were not as good as ours with a denoising target at the input layer, their results with full labels come very close. Their cost function is at the last layer which suggests that the approaches are complementary and could be combined, potentially further improving the results.

So far we have reviewed semi-supervised methods which have an unsupervised cost function at the output layer only and therefore are related to our Γ-model. We will now move to other semi-supervised methods that concentrate on modeling the joint distribution of the inputs and the labels.

The Multi-prediction deep Boltzmann machine (MP-DBM) (Goodfellow *et al.*, 2013a) is a way to train a DBM with backpropagation through variational inference. The targets of the inference include both supervised targets (classification) and unsupervised targets (reconstruction of missing inputs) that are used in training simultaneously. The connections through the inference network are somewhat analogous to our lateral connections. Specifically, there are inference paths from observed inputs to reconstructed inputs that do not go all the way up to the highest layers. Compared to our approach, MP-DBM requires an iterative inference with some initialization for the hidden activations, whereas in our case, the inference is a simple single-pass feedforward procedure.

The Deep AutoRegressive Network (Gregor *et al.*, 2014) is an unsupervised method for learning representations that also uses lateral connections in the hidden representations. The connectivity within the layer is rather different from ours, though: Each unit h_i receives input from the preceding units $h_1 \dots h_{i-1}$, whereas in our case each unit $\hat{z}_i$ receives input only from z_i. Their learning algorithm is based on approximating a gradient of a description length measure, whereas we use a gradient of a simple loss function.

Kingma *et al.* (2014) proposed deep generative models for semi-supervised learning, based on variational autoencoders. Their models can be trained either with the variational EM algorithm, stochastic gradient variational Bayes, or stochastic backpropagation. They also experimented on a stacked version (called M1+M2) where the bottom autoencoder M1 reconstructs the input data, and the top autoencoder M2 can concentrate on classification and on reconstructing only the hidden representation of M1. The stacked version performed the best, hinting that it might be important not to

carry all the information up to the highest layers. Compared with the Ladder network, an interesting point is that the variational autoencoder computes the posterior estimate of the latent variables with the encoder alone while the Ladder network uses the decoder, too, to compute an implicit posterior approximate (encoder provides the likelihood part which gets combined with the prior). It will be interesting to see whether the approaches can be combined. A Ladder-style decoder might provide the posterior and another decoder could then act as the generative model of variational autoencoders.

Zeiler *et al.* (2011) train deep convolutional autoencoders in a manner comparable to ours. They define max-pooling operations in the encoder to feed the max function upwards to the next layer, while the argmax function is fed laterally to the decoder. The network is trained one layer at a time using a cost function that includes a pixel-level reconstruction error, and a regularization term to promote sparsity. Zhao *et al.* (2015) use a similar structure and call it the stacked what-where autoencoder (SWWAE). Their network is trained simultaneously to minimize a combination of the supervised cost and reconstruction errors on each level, just like ours.

Recently Bengio (2014) proposed target propagation as an alternative to backpropagation. The idea is to base learning not on errors and gradients but on expectations. This is very similar to the idea of denoising source separation and therefore resembles the propagation of expectations in the decoder of the Ladder network. In the Ladder network, the additional lateral connections between the encoder and the decoder play an important role and it will remain to be seen whether the lateral connections are compatible with target propagation. Nevertheless, it is an interesting possibility that while the Ladder network includes two mechanisms for propagating information, backpropagation of gradients and forward propagation of expectations in the decoder, it may be possible to rely solely on the latter, avoiding problems related to propagation of gradients through many layers, such as exploding gradients.

6 Discussion

We showed how a simultaneous unsupervised learning task improves CNN and MLP networks reaching the state-of-the-art in various semi-supervised learning tasks. Particularly the performance obtained with very small numbers of labels is much better than previous published results which shows that the method is capable of making good use of unsupervised learning. However, the same model also achieves state-of-the-art results and a significant improvement over the baseline model with full labels in permutation invariant MNIST classification which suggests that the unsupervised task does not disturb supervised learning.

The proposed model is simple and easy to implement with many existing feedforward architectures, as the training is based on backpropagation from a simple cost function. It is quick to train and the convergence is fast, especially with batch normalization.

Not surprisingly, largest improvements in performance were observed in models which have a large number of parameters relative to the number of available labeled samples. With CIFAR-10, we started with a model which was originally developed for a fully supervised task. This has the benefit of building on existing experience but it may well be that the best results will be obtained with models which have far more parameters than fully supervised approaches could handle.

An obvious future line of research will therefore be to study what kind of encoders and decoders are best suited for the Ladder network. In this work, we made very little modifications to the encoders whose structure has been optimized for supervised learning and we designed the parametrization of the vertical mappings of the decoder to mirror the encoder: the flow of information is just reversed. There is nothing preventing the decoder to have a different structure than the encoder. Also, there were lateral connections from the encoder to the decoder on every layer and on every pooling operation. The miniature MLP used for every denoising function g gives the decoder enough capacity to invert the mappings but the same effect could have been accomplished by not requiring the decoder to match the activations of the encoder on every layer.

A particularly interesting future line of research will be the extension of the Ladder networks to the temporal domain. While there exist datasets with millions of labeled samples for still images, it is prohibitively costly to label thousands of hours of video streams. The Ladder network can be scaled up easily and therefore offers an attractive approach for semi-supervised learning in such large-scale problems.

Acknowledgements

We have received comments and help from a number of colleagues who would all deserve to be mentioned but we wish to thank especially Yann LeCun, Diederik Kingma, Aaron Courville and Ian Goodfellow for their helpful comments and suggestions. The software for the simulations for this paper was based on Theano (Bastien *et al.*, 2012; Bergstra *et al.*, 2010) and Blocks (van Merriënboer *et al.*, 2015). We also acknowledge the computational resources provided by the Aalto Science-IT project. The Academy of Finland has supported Tapani Raiko.

References

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y. (2014). How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv:1407.7906*.

Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 899–907.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy 2010)*. Oral Presentation.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proc. of the eleventh annual conference on Computational learning theory (COLT '98)*, pages 92–100.

Chapelle, O., Schölkopf, B., Zien, A., *et al.* (2006). *Semi-supervised learning*. MIT press.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 766–774.

Goodfellow, I., Bengio, Y., and Courville, A. C. (2012). Large-scale feature learning with spike-and-slab sparse coding. In *Proc. of ICML 2012*, pages 1439–1446.

Goodfellow, I., Mirza, M., Courville, A., and Bengio, Y. (2013a). Multi-prediction deep Boltzmann machines. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 548–556.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *the International Conference on Learning Representations (ICLR 2015)*. arXiv:1412.6572.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013b). Maxout networks. In *Proc. of ICML 2013*.

Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *Proc. of ICML 2014*, Beijing, China.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *the International Conference on Learning Representations (ICLR 2015)*, San Diego. arXiv:1412.6980.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 3581–3589.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML 2013*.

McLachlan, G. (1975). Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. *J. American Statistical Association*, **70**, 365–369.

Miyato, T., ichi Maeda, S., Koyama, M., Nakae, K., and Ishii, S. (2015). Distributional smoothing by virtual adversarial examples. *arXiv:1507.00677*.

Pitelis, N., Russell, C., and Agapito, L. (2014). Semi-supervised learning using an unsupervised atlas. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2014)*, pages 565–580. Springer.

Raiko, T., Berglund, M., Alain, G., and Dinh, L. (2015). Techniques for learning binary stochastic feedforward neural networks. In *ICLR 2015*, San Diego.

Ranzato, M. A. and Szummer, M. (2008). Semi-supervised learning of compact document representations with deep networks. In *Proc. of ICML 2008*, pages 792–799. ACM.

Rasmus, A., Raiko, T., and Valpola, H. (2015a). Denoising autoencoder with modulated lateral connections learns invariant representations of natural images. *arXiv:1412.7210*.

Rasmus, A., Valpola, H., and Raiko, T. (2015b). Lateral connections in denoising autoencoders support supervised learning. *arXiv:1504.08215*.

Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., and Glorot, X. (2011a). Higher order contractive auto-encoder. In *ECML PKDD 2011*.

Rifai, S., Dauphin, Y. N., Vincent, P., Bengio, Y., and Muller, X. (2011b). The manifold tangent classifier. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pages 2294–2302.

Särelä, J. and Valpola, H. (2005). Denoising source separation. *JMLR*, **6**, 233–272.

Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. *Neural networks*, **4**(1), 67–79.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net. *arxiv:1412.6806*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, **15**(1), 1929–1958.

Suddarth, S. C. and Kergosien, Y. (1990). Rule-injection hints as a means of improving network performance and learning time. In *Proceedings of the EURASIP Workshop 1990 on Neural Networks*, pages 120–129. Springer.

Szummer, M. and Jaakkola, T. (2003). Partially labeled classification with Markov random walks. *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, **14**, 945–952.

Titterington, D., Smith, A., and Makov, U. (1985). Statistical analysis of finite mixture distributions. In *Wiley Series in Probability and Mathematical Statistics*. Wiley.

Valpola, H. (2015). From neural PCA to deep unsupervised learning. In *Adv. in Independent Component Analysis and Learning Machines*, pages 143–171. Elsevier. arXiv:1411.7783.

van Merriënboer, B., Bahdanau, D., Dumoulin, V., Serdyuk, D., Warde-Farley, D., Chorowski, J., and Bengio, Y. (2015). Blocks and fuel: Frameworks for deep learning. *CoRR*, **abs/1506.00619**.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, **11**, 3371–3408.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R. (2012). Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer.

Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV 2011*, pages 2018–2025. IEEE.

Zhang, T. and Oles, F. (2000). The value of unlabeled data for classification problems. In *Proc. of ICML 2000*, pages 1191–1198.

Zhao, J., Mathieu, M., Goroshin, R., and Lecun, Y. (2015). Stacked what-where auto-encoders. arXiv:1506.02351.

A Specification of the convolutional models

Table 4: Description ConvPool-CNN-C by Springenberg *et al.* (2014) and our networks based on it.

Model		
ConvPool-CNN-C	Conv-Large (for CIFAR-10)	Conv-Small (for MNIST)
Input 32×32 or 28×28 RGB or monochrome image		
3×3 conv. 96 ReLU	3×3 conv. 96 BN LeakyReLU	5×5 conv. 32 ReLU
3×3 conv. 96 ReLU	3×3 conv. 96 BN LeakyReLU	
3×3 conv. 96 ReLU	3×3 conv. 96 BN LeakyReLU	
3×3 max-pooling stride 2	2×2 max-pooling stride 2 BN	2×2 max-pooling stride 2 BN
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	3×3 conv. 64 BN ReLU
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	3×3 conv. 64 BN ReLU
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	
3×3 max-pooling stride 2	2×2 max-pooling stride 2 BN	2×2 max-pooling stride 2 BN
3×3 conv. 192 ReLU	3×3 conv. 192 BN LeakyReLU	3×3 conv. 128 BN ReLU
1×1 conv. 192 ReLU	1×1 conv. 192 BN LeakyReLU	
1×1 conv. 10 ReLU	1×1 conv. 10 BN LeakyReLU	1×1 conv. 10 BN ReLU
global meanpool	global meanpool BN	global meanpool BN
		fully connected 10 BN
10-way softmax		

Here we describe two model structures, Conv-Small and Conv-Large, that were used for MNIST and CIFAR-10 datasets, respectively. They were both inspired by ConvPool-CNN-C by Springenberg *et al.* (2014). Table 4 details the model architectures and differences between the models in this work and ConvPool-CNN-C. It is noteworthy that this architecture does not use any fully connected layers, but replaces them with a global mean pooling layer just before the softmax function. The main differences between our models and ConvPool-CNN-C are the use of Gaussian noise instead of dropout and the convolutional per-channel batch normalization following Ioffe and Szegedy (2015). We also used 2x2 stride 2 max-pooling instead of 3x3 stride 2 max-pooling. LeakyReLU was used to speed up training, as mentioned by Springenberg *et al.* (2014). We utilized batch normalization in all layers, including pooling layers. Gaussian noise was also added to all layers, instead of applying dropout in only some of the layers as with ConvPool-CNN-C.

B Formulation of the Denoising Function

The denoising function g tries to map the clean $\mathbf{z}^{(l)}$ to the reconstructed $\hat{\mathbf{z}}^{(l)}$, where $\hat{\mathbf{z}}^{(l)} = g(\tilde{\mathbf{z}}^{(l)}, \hat{\mathbf{z}}^{(l+1)})$. The reconstruction is therefore based on the corrupted value, and the reconstruction of the layer above.

An optimal functional form of g depends on the conditional distribution $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$. For example, if the distribution $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ is Gaussian, the optimal function g, that is the function that achieves the lowest reconstruction error, is going to be linear with respect to $\tilde{\mathbf{z}}^{(l)}$ (Valpola, 2015, Section 4.1).

When analyzing the distribution $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ learned by a purely supervised network, it would therefore be desirable to parametrize g in such a way as to be able to optimally denoise the kinds of distributions the network has found for the hidden activations.

In our preliminary analyses of the distributions learned by the hidden layers, we found many different very non-Gaussian distributions that we wanted the g-function to be able to denoise. One example were bimodal distributions, that were often observed in the layer below the final classification layer. We could also observe that in many cases, the value of $\mathbf{z}^{(l+1)}$ had an impact on $p(\mathbf{z}^{(l)} \mid \mathbf{z}^{(l+1)})$ beyond shifting the mean of the distribution, which led us to propose a form where the vertical connections from $\hat{\mathbf{z}}^{(l+1)}$ could modulate the horizontal connections from $\tilde{\mathbf{z}}^{(l)}$ instead of only additively shifting the distribution defined by g. This corresponds to letting the variance and other higher-order cumulants of $\mathbf{z}^{(l)}$ depend on $\hat{\mathbf{z}}^{(l+1)}$.

Based on this analysis, we proposed the following parametrization for g:

$$\hat{z} = g(\tilde{z}, u) = \mathbf{a}\boldsymbol{\xi} + b\texttt{sigmoid}(\mathbf{c}\boldsymbol{\xi}) \tag{3}$$

where $\xi = [1, \tilde{z}, u, \tilde{z}u]^T$ is the augmented input, $\mathbf{a}$ and $\mathbf{c}$ are trainable weight vectors, b is a trainable scalar weight. We have left out the superscript (l) and subscript i in order not to clutter the equations[8]. This corresponds to a miniature MLP network as explained in Section 3.3.

In order to test whether the elements of the proposed function g were necessary, we systematically removed components from g or changed g altogether and compared to the results obtained with the original parametrization. We tuned the hyperparameters of each comparison model separately using a grid search over some of the relevant hyperparameters. However, the std of additive Gaussian corruption noise was set to 0.3. This means that with $N = 1000$ labels, the comparison does not include the best-performing model reported in Table 1.

As in the proposed function g, all comparison denoising functions mapped neuron-wise the corrupted hidden layer pre-activation $\tilde{\mathbf{z}}^{(l)}$ to the reconstructed hidden layer activation given one projection from the reconstruction of the layer above: $\hat{z}_i^{(l)} = g(\tilde{z}_i^{(l)}, u_i^{(l)})$.

Test error % with # of used labels	100	1000
Proposed function g: miniature MLP with $\tilde{z}u$	1.11 ($\pm$ 0.07)	1.11 ($\pm$ 0.06)
Comparison g_2: No augmented term $\tilde{z}u$	2.03 ($\pm$ 0.09)	1.70 ($\pm$ 0.08)
Comparison g_3: Linear g but with $\tilde{z}u$	1.49 ($\pm$ 0.10)	1.30 ($\pm$ 0.08)
Comparison g_4: Only the mean depends on u	2.90 ($\pm$ 1.19)	2.11 ($\pm$ 0.45)
Comparison g_5: Gaussian z	**1.06** ($\pm$ 0.07)	**1.03** ($\pm$ 0.06)

Table 5: Semi-supervised results from the MNIST dataset. The proposed function g is compared to alternative parametrizations

The comparison functions $g_{2\ldots5}$ are parametrized as follows:

Comparison g_2: No augmented term

$$g_2(\tilde{z}, u) = \mathbf{a}\boldsymbol{\xi}' + b\texttt{sigmoid}(\mathbf{c}\boldsymbol{\xi}') \tag{4}$$

where $\xi' = [1, \tilde{z}, u]^T$. g_2 therefore differs from g in that the input lacks the augmented term $\tilde{z}u$. In the original formulation, the augmented term was expected to increase the freedom of the denoising to modulate the distribution of z by u. However, we wanted to test the effect on the results.

Comparison g_3: Linear g

$$g_3(\tilde{z}, u) = \mathbf{a}\boldsymbol{\xi}. \tag{5}$$

g_3 differs from g in that it is linear and does not have a sigmoid term. As this formulation is linear, it only supports Gaussian distributions. Although the parametrization has the augmented term that lets u modulate the slope and shift of the distribution, the scope of possible denoising functions is still fairly limited.

Comparison g_4: u affects only the mean of $p(z \mid u)$

$$g_4(\tilde{z}, u) = a_1 u + a_2\texttt{sigmoid}(a_3 u + a_4) + a_5\tilde{z} + a_6\texttt{sigmoid}(a_7\tilde{z} + a_8) + a_9 \tag{6}$$

g_4 differs from g in that the inputs from u are not allowed to modulate the terms that depend on $\tilde{z}$, but that the effect is additive. This means that the parametrization only supports optimal denoising functions for a conditional distribution $p(z \mid u)$ where u only shifts the mean of the distribution of z but otherwise leaves the shape of the distribution intact.

Comparison g_5: Gaussian z As reviewed by Valpola (2015, Section 4.1), assuming z is Gaussian given u, the optimal denoising can be represented as

$$g_5(\tilde{z}, u) = (\tilde{z} - \mu(u))\, v(u) + \mu(u)\,. \tag{7}$$

We modeled both $\mu(u)$ and $v(u)$ with a miniature MLP network: $\mu(u) = a_1\texttt{sigmoid}(a_2 u + a_3) + a_4 u + a_5$ and $v(u) = a_6\texttt{sigmoid}(a_7 u + a_8) + a_9 u + a_{10}$. Given u, this parametrization is linear with respect to $\tilde{z}$, and both the slope and the bias depended nonlinearly on u.

[8] For the exact definition, see Eq. (1).

Results All models were tested in a similar setting as the semi-supervised fully connected MNIST task using $N = 1000$ labeled samples. We also reran the best comparison model on $N = 100$ labels. The results of the analyses are presented in Table 5.

As can be seen from the table, the alternative parametrizations of g are mostly inferior to the proposed parametrizations at least in the model structure we use. A notable exception is the denoising function g_5 which corresponds to a Gaussian model of z. Although the difference is small, it actually achieved the best performance with both $N = 100$ labels and $N = 1000$ labels. It therefore looks like using two miniature MLPs, one for the mean and the other for the variance of a Gaussian denoising model, offers a slight benefit over a single miniature MLP that was used in the experiments in Section 4.

Note that even if $p(z|u)$ is Gaussian given u, its marginal $p(z)$ is typically non-Gaussian. A conditionally Gaussian $p(z|u)$ which was implicitly used in g_5 forces the network to represent any non-Gaussian distributions with higher-level hidden neurons and these may well turn out to be useful features. For instance, if the marginal $p(z)$ is a mixture of Gaussian distributions, higher levels have a pressure to represent the mixture index because then $p(z|u)$ would be Gaussian and the denoising g_5 optimal as long as the higher layer represents the index information in such a way that g_5 can decode it from u.

In any case, these results support the finding by Rasmus *et al.* (2015a) that modulation of the lateral connection from $\tilde{z}$ to $\hat{z}$ by u is critical for encouraging the development of invariant representation in the higher layers of the model. Comparison function g_4 lacked this modulation and it performed clearly worse than any other denoising function listed in Table 5. Even the linear g_3 performed very well as long it had the term $\tilde{z}u$. Leaving the nonlinearity but removing $\tilde{z}u$ in g_2 hurt the performance much more.

In addition to the alternative parametrizations for the g-function, we ran experiments using a more standard autoencoder structure. In that structure, we attached an additional decoder to the standard MLP by using one hidden layer as the input to the decoder, and the reconstruction of the clean input as the target. The structure of the decoder was set to be the same as the encoder, that is the number and size of the layers from the input to the hidden layer where the decoder was attached was the same as the number and size of the layers in the decoder. The final activation function in the decoder was set to be the sigmoid nonlinearity. During training, the target was the weighted sum of the reconstruction cost and the classification cost.

We tested the autoencoder structure with 100 and 1000 labeled samples. We ran experiments for all possible decoder lengths, that is we tried attaching the decoder to all hidden layers. However, we did not manage to get significantly better performance than the standard supervised model without any decoder in any of the experiments.

FAST GRAPH REPRESENTATION LEARNING WITH PYTORCH GEOMETRIC

Matthias Fey & Jan E. Lenssen
Department of Computer Graphics
TU Dortmund University
44227 Dortmund, Germany
{matthias.fey,janeric.lenssen}@udo.edu

ABSTRACT

We introduce *PyTorch Geometric*, a library for deep learning on irregularly structured input data such as graphs, point clouds and manifolds, built upon PyTorch. In addition to general graph data structures and processing methods, it contains a variety of recently published methods from the domains of relational learning and 3D data processing. PyTorch Geometric achieves high data throughput by leveraging sparse GPU acceleration, by providing dedicated CUDA kernels and by introducing efficient mini-batch handling for input examples of different size. In this work, we present the library in detail and perform a comprehensive comparative study of the implemented methods in homogeneous evaluation scenarios.

1 INTRODUCTION

Graph Neural Networks (GNNs) recently emerged as a powerful approach for representation learning on graphs, point clouds and manifolds (Bronstein et al., 2017; Kipf & Welling, 2017). Similar to the concepts of convolutional and pooling layers on regular domains, GNNs are able to (hierarchically) extract localized embeddings by passing, transforming, and aggregating information (Bronstein et al., 2017; Gilmer et al., 2017; Battaglia et al., 2018; Ying et al., 2018).

However, implementing GNNs is challenging, as high GPU throughput needs to be achieved on highly sparse and irregular data of varying size. Here, we introduce *PyTorch Geometric* (PyG), a geometric deep learning extension library for PyTorch (Paszke et al., 2017) which achieves high performance by leveraging dedicated CUDA kernels. Following a simple message passing API, it bundles most of the recently proposed convolutional and pooling layers into a single and unified framework. All implemented methods support both CPU and GPU computations and follow an immutable data flow paradigm that enables dynamic changes in graph structures through time. PyG is released under the MIT license and is available on GitHub.[1] It is thoroughly documented and provides accompanying tutorials and examples as a first starting point.[2]

2 OVERVIEW

In PyG, we represent a graph $\mathcal{G} = (\boldsymbol{X}, (\boldsymbol{I}, \boldsymbol{E}))$ by a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ and a sparse adjacency tuple $(\boldsymbol{I}, \boldsymbol{E})$, where $\boldsymbol{I} \in \mathbb{N}^{2 \times E}$ encodes edge indices in coordinate (COO) format and $\boldsymbol{E} \in \mathbb{R}^{E \times D}$ (optionally) holds D-dimensional edge features. All user facing APIs, *e.g.*, data loading routines, multi-GPU support, data augmentation or model instantiations are heavily inspired by PyTorch to keep them as familiar as possible.

Neighborhood Aggregation. Generalizing the convolutional operator to irregular domains is typically expressed as a *neighborhood aggregation* or *message passing* scheme (Gilmer et al., 2017)

$$\vec{x}_i^{(k)} = \gamma^{(k)} \left(\vec{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\vec{x}_i^{(k-1)}, \vec{x}_j^{(k-1)}, \vec{e}_{i,j} \right) \right) \tag{1}$$

[1] GitHub repository: https://github.com/rusty1s/pytorch_geometric
[2] Documentation: https://rusty1s.github.io/pytorch_geometric

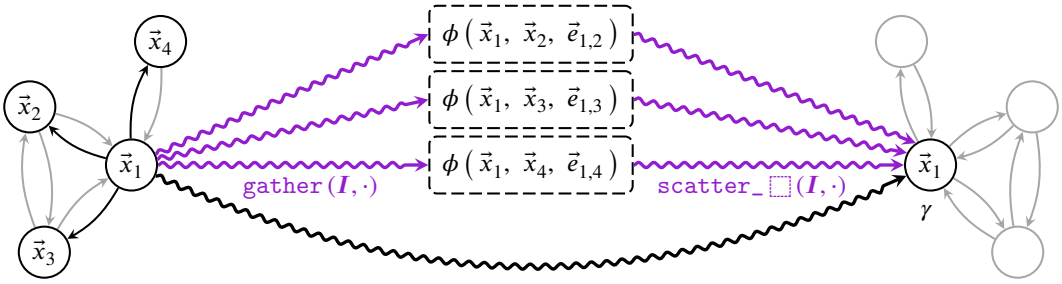


Figure 1: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices $\boldsymbol{I}$, hence alternating between node parallel space and edge parallel space.

where $\square$ denotes a differentiable, permutation invariant function, *e.g.*, sum, mean or max, and γ and ϕ denote differentiable functions, *e.g.*, MLPs. In practice, this can be achieved by gathering and scattering of node features and making use of broadcasting for element-wise computation of γ and ϕ, as visualized in Figure 1. Although working on irregularly structured input, this scheme can be heavily accelerated by the GPU.

We provide the user with a general `MessagePassing` interface to allow for rapid and clean prototyping of new research ideas. To use, users only need to define the methods $\phi^{(k)}$, *i.e.*, `message`, and $\gamma^{(k)}$, *i.e.*, `update`, as well as chosing an aggregation scheme $\square$. For implementing $\phi^{(k)}$, node features are automatically mapped to the respective source and target nodes.

Almost all recently proposed neighborhood aggregation functions can be lifted to this interface, including (but not limited to) the methods already integrated into PyG: For learning on arbitrary graphs we have already implemented GCN (Kipf & Welling, 2017) and its simplified version (SGC) from Wu et al. (2019), the spectral chebyshev and ARMA filter convolutions (Defferrard et al., 2016; Bianchi et al., 2019), GraphSAGE (Hamilton et al., 2017), the attention-based operators GAT (Veličković et al., 2018) and AGNN (Thekumparampil et al., 2018), the Graph Isomorphism Network (GIN) from Xu et al. (2019), and the Approximate Personalized Propagation of Neural Predictions (APPNP) operator (Klicpera et al., 2019). For learning on point clouds, manifolds and graphs with multi-dimensional edge features, we provide the relational GCN operator from Schlichtkrull et al. (2018), PointNet++ (Qi et al., 2017), PointCNN (Li et al., 2018), and the continuous kernel-based methods MPNN (Gilmer et al., 2017), MoNet (Monti et al., 2017), SplineCNN (Fey et al., 2018) and the edge convolution operator (EdgeCNN) from Wang et al. (2018b).

Global Pooling. PyG also supports graph-level outputs as opposed to node-level outputs by providing a variety of *readout* functions such as global add, mean or max pooling. We additionaly offer more sophisticated methods such as set-to-set (Vinyals et al., 2016), sort pooling (Zhang et al., 2018) or the global soft attention layer from Li et al. (2016).

Hierarchical Pooling. To further extract hierarchical information and to allow deeper GNN models, various pooling approaches can be applied in a spatial or data-dependent manner. We currently provide implementation examples for Graclus (Dhillon et al., 2007; Fagginger Auer & Bisseling, 2011) and voxel grid pooling (Simonovsky & Komodakis, 2017), the iterative farthest point sampling algorithm (Qi et al., 2017) followed by k-NN or query ball graph generation (Qi et al., 2017; Wang et al., 2018b), and differentiable pooling mechanisms such as DiffPool (Ying et al., 2018) and top$_k$ pooling (Gao & Ji, 2018; Cangea et al., 2018).

Mini-batch Handling. Our framework supports batches of multiple graph instances (of potentially different size) by automatically creating a single (sparse) block-diagonal adjacency matrix and concatenating feature matrices in the node dimension. Therefore, neighborhood aggregation methods can be applied without modification, since no messages are exchanged between disconnected graphs. In addition, an automatically generated assignment vector ensures that node-level information is not aggregated across graphs, *e.g.*, when executing global aggregation operators.

Processing of Datasets. We provide a consistent data format and an easy-to-use interface for the creation and processing of datasets, both for large datasets and for datasets that can be kept in memory

Table 1: Semi-supervised node classification with both fixed and random splits.

Method	Cora		CiteSeer		PubMed	
	Fixed	Random	Fixed	Random	Fixed	Random
Cheby	81.4 ± 0.7	77.8 ± 2.2	70.2 ± 1.0	67.7 ± 1.7	78.4 ± 0.4	75.8 ± 2.2
GCN	81.5 ± 0.6	79.4 ± 1.9	71.1 ± 0.7	68.1 ± 1.7	79.0 ± 0.6	77.4 ± 2.4
GAT	83.1 ± 0.4	81.0 ± 1.4	70.8 ± 0.5	69.2 ± 1.9	78.5 ± 0.3	78.3 ± 2.3
SGC	81.7 ± 0.1	80.2 ± 1.6	71.3 ± 0.2	68.7 ± 1.6	78.9 ± 0.1	76.5 ± 2.4
ARMA	82.8 ± 0.6	80.7 ± 1.4	$\mathbf{72.3 \pm 1.1}$	68.9 ± 1.6	78.8 ± 0.3	77.7 ± 2.6
APPNP	$\mathbf{83.3 \pm 0.5}$	$\mathbf{82.2 \pm 1.5}$	71.8 ± 0.5	$\mathbf{70.0 \pm 1.4}$	$\mathbf{80.1 \pm 0.2}$	$\mathbf{79.4 \pm 2.2}$

during training. In order to create new datasets, users just need to read/download their data and convert it to the PyG data format in the respective `process` method. In addition, datasets can be modified by the use of `transforms`, which take in separate graphs and transform them, *e.g.*, for data augmentation, for enhancing node features with synthetic structural graph properties, to automatically generate graphs from point clouds or to sample point clouds from meshes.

PyG already supports a lot of common benchmark datasets often found in literature which are automatically downloaded and processed on first instantiation. In detail, we provide over 60 graph kernel benchmark datasets[3] (Kersting et al., 2016), *e.g.*, PROTEINS or IMDB-BINARY, the citation graphs Cora, CiteSeer, PubMed and Cora-Full (Sen et al., 2008; Bojchevski & Günnemann, 2018), the Coauthor CS/Physics and Amazon Computers/Photo datasets from Shchur et al. (2018), the molecule datasets QM7b (Montavon et al., 2013) and QM9 (Ramakrishnan et al., 2014), and the protein-protein interaction graphs from Hamilton et al. (2017). In addition, we provide embedded datasets like MNIST superpixels (Monti et al., 2017), FAUST (Bogo et al., 2014), ModelNet10/40 (Wu et al., 2015), ShapeNet (Chang et al., 2015), COMA (Ranjan et al., 2018), and the PCPNet dataset from Guerrero et al. (2018).

3 Empirical Evaluation

We evaluate the correctness of the implemented methods by performing a comprehensive comparative study in homogeneous evaluation scenarios. Descriptions and statistics of all used datasets can be found in Appendix A. For all experiments, we tried to follow the hyperparameter setup of the respective papers as closely as possible. The individual experimental setups can be derived and all experiments can be replicated from the code provided at our GitHub repository.[4]

Semi-supervised Node Classification. We perform semi-supervised node classification (*cf.* Table 1) by reporting average accuracies of (a) 100 runs for the fixed train/val/test split from Kipf & Welling (2017), and (b) 100 runs of randomly initialized train/val/test splits as suggested by Shchur et al. (2018), where we additionally ensure uniform class distribution on the train split.

Nearly all experiments show a high reproducibility of the results reported in the respective papers. However, test performance is worse for all models when using random data splits. Among the experiments, the APPNP operator (Klicpera et al., 2019) generally performs best, with ARMA (Bianchi et al., 2019), SGC (Wu et al., 2019), GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) following closely behind.

Graph Classification. We report the average accuracy of 10-fold cross validation on a number of common benchmark datasets (*cf.* Table 2) where we randomly sample a training fold to serve as a validation set. We only make use of discrete node features. In case they are not given, we use one-hot encodings of node degrees as feature input. For all experiments, we use the global mean operator to obtain graph-level outputs. Inspired by the Jumping Knowledge framework (Xu et al., 2018), we compute graph-level outputs after each convolutional layer and combine them via concatenation. For evaluating the (global) pooling operators, we use the GraphSAGE operator as our baseline. We omit Jumping Knowledge when comparing global pooling operators, and hence report an additional

[3] Kernel datasets: `http://graphkernels.cs.tu-dortmund.de`

[4] `https://github.com/rusty1s/pytorch_geometric/tree/master/benchmark`

Table 2: Graph classification.

	Method	MUTAG	PROTEINS	COLLAB	IMDB-BINARY	REDDIT-BINARY
Flat	GCN	74.6 ± 7.7	73.1 ± 3.8	**80.6 ± 2.1**	72.6 ± 4.5	89.3 ± 3.3
	SAGE	74.9 ± 8.7	**73.8 ± 3.6**	79.7 ± 1.7	72.4 ± 3.6	89.1 ± 1.9
	GIN-0	**85.7 ± 7.7**	72.1 ± 5.1	79.3 ± 2.7	**72.8 ± 4.5**	89.6 ± 2.6
	GIN-ϵ	83.4 ± 7.5	72.6 ± 4.9	79.8 ± 2.4	72.1 ± 5.1	**90.3 ± 3.0**
Hier.	Graclus	77.1 ± 7.2	73.0 ± 4.1	79.6 ± 2.0	72.2 ± 4.2	88.8 ± 3.2
	top_k	76.3 ± 7.5	72.7 ± 4.1	**79.7 ± 2.2**	72.5 ± 4.6	87.6 ± 2.4
	DiffPool	**85.0 ± 10.3**	**75.1 ± 3.5**	78.9 ± 2.3	**72.6 ± 3.9**	**92.1 ± 2.6**
Global	SAGE w/o JK	73.7 ± 7.8	72.7 ± 3.6	79.6 ± 2.4	72.1 ± 4.4	87.9 ± 1.9
	GlobalAttention	74.6 ± 8.0	72.5 ± 4.5	**79.6 ± 2.2**	72.3 ± 3.8	87.4 ± 2.5
	Set2Set	73.7 ± 6.9	**73.6 ± 3.7**	79.6 ± 2.3	72.2 ± 4.2	**89.6 ± 2.4**
	SortPool	**77.3 ± 8.9**	72.4 ± 4.1	77.7 ± 3.1	**72.4 ± 3.8**	74.9 ± 6.7

Table 3: Point cloud classification.

Method	ModelNet10
MPNN	92.07
PointNet++	92.51
EdgeCNN	92.62
SplineCNN	92.65
PointCNN	**93.28**

Table 4: Training runtime comparison.

Dataset	Epochs	Method	DGL	PyG
Cora	200	GCN	4.2s	**0.7s**
		GAT	33.4s	**2.2s**
CiteSeer	200	GCN	3.9s	**0.8s**
		GAT	28.9s	**2.4s**
PubMed	200	GCN	12.7s	**2.0s**
		GAT	87.7s	**12.3s**
MUTAG	50	R-GCN	3.3s	**2.4s**

baseline based on global mean pooling. For each dataset, we tune (1) the number of hidden units $\in \{16, 32, 64, 128\}$ and (2) the number of layers $\in \{2, 3, 4, 5\}$ with respect to the validation set.

Except for DiffPool (Ying et al., 2018), (global) pooling operators do not perform as benefically as expected to their respective (flat) counterparts, especially when baselines are enhanced by Jumping Knowledge (Xu et al., 2018). However, the potential of more sophisticated approaches may not be well-reflected on these simple benchmark tasks (Cai & Wang, 2018). Among the flat GNN approaches, the GIN layer (Xu et al., 2019) generally achieves the best results.

Point Cloud Classification. We evaluate various point cloud methods on ModelNet10 (Wu et al., 2015) where we uniformly sample 1,024 points from mesh surfaces based on face area (*cf.* Table 3). As hierarchical pooling layers, we use the iterative farthest point sampling algorithm followed by a new graph generation based on a larger query ball (PointNet++ (Qi et al., 2017), MPNN (Gilmer et al., 2017) and SplineCNN (Fey et al., 2018)) or based on a fixed number of nearest neighbors (EdgeCNN (Wang et al., 2018b) and PointCNN (Li et al., 2018)). We have taken care to use approximately the same number of parameters for each model.

All approaches perform nearly identically with PointCNN (Li et al., 2018) taking a slight lead. We attribute this to the fact that all operators are based on similar principles and might have the same expressive power for the given task.

Runtime Experiments. We conduct several experiments on a number of dataset-model pairs to report the runtime of a whole training procedure obtained on a single NVIDIA GTX 1080 Ti (*cf.* Table 4). As it shows, PyG is very fast despite working on sparse data. Compared to the Deep Graph Library (DGL) 0.1.3 (Wang et al., 2018a), PyG trains models up to 15 times faster.

4 Roadmap and Conclusion

We presented the PyTorch Geometric framework for fast representation learning on graphs, point clouds and manifolds. We are actively working to further integrate existing methods and plan to quickly integrate future methods into our framework. All researchers and software engineers are invited to collaborate with us in extending its scope.

Acknowledgments

This work has been supported by the *German Research Association (DFG)* within the Collaborative Research Center SFB 876, *Providing Information by Resource-Constrained Analysis*, projects A6 and B2. We thank Moritz Ludolph for his contribution to PyTorch Geometric and Christopher Morris for proofreading and helpful advice.

References

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional ARMA filters. *CoRR*, abs/1901.01343, 2019.

F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*, 2014.

A. Bojchevski and S. Günnemann. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. In *Signal Processing Magazine*, 2017.

C. Cai and Y. Wang. A simple yet effective baseline for non-attribute graph classification. *CoRR*, abs/1811.03508, 2018.

C. Cangea, P. Veličković, N. Jovanović, T. N. Kipf, and P. Liò. Towards sparse hierarchical graph classifiers. In *NeurIPS-W*, 2018.

A. X. Chang, T. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *CoRR*, abs/1512.03012, 2015.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. In *TPAMI*, 2007.

B. O. Fagginger Auer and R. H. Bisseling. A GPU algorithm for greedy graph matching. In *Facing the Multicore - Challenge II - Aspects of New Paradigms and Technologies in Parallel Computing*, 2011.

M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *CVPR*, 2018.

H. Gao and S. Ji. Graph U-Net. https://openreview.net/forum?id=HJePRoAct7, 2018. Submitted to ICLR.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37, 2018.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels. http://graphkernels.cs.tu-dortmund.de, 2016.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *ICLR*, 2019.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution on $\mathcal{X}$-transformed points. In *NeurIPS*, 2018.

G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K. Müller, and O. A. von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics*, 2013.

F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2017.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 2014.

A. Ranjan, T. Bolkart, S. Sanyal, and M. J. Black. Generating 3D faces using convolutional mesh autoencoders. In *ECCV*, 2018.

M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

G. Sen, G. Namata, M. Bilgic, and L. Getoor. Collective classification in network data. *AI Magazine*, 29, 2008.

O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. In *NeurIPS-W*, 2018.

M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.

K. K. Thekumparampil, C. Wang, S. Oh, and L. Li. Attention-based graph neural network for semi-supervised learning. *CoRR*, abs/1803.03735, 2018.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

M. Wang, L. Yu, A. Gan, D. Zheng, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, J. Zhao, H. Lin, C. Ma, D. Deng, Q. Guo, H. Zhang, J. Li, A. J. Smola, and Z. Zhang. Deep graph library. http://dgl.ai, 2018a.

Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018b.

F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. *CoRR*, abs/1902.07153, 2019.

Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015.

K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

R. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

A DATASETS

Table 5: Statistics of the datasets used in the experiments.

Dataset	Graphs	Nodes	Edges	Features	Classes	Label rate
Cora	1	2,708	5,278	1,433	7	0.052
CiteSeer	1	3,327	4,552	3,703	6	0.036
PubMed	1	19,717	44,324	500	3	0.003
MUTAG	188	17.93	19.79	7	2	0.800
PROTEINS	1,113	39.06	72.82	3	2	0.800
COLLAB	5,000	74.49	2,457.22	—	3	0.800
IMDB-BINARY	1,000	19.77	96.53	—	2	0.800
REDDIT-BINARY	2,00	429.63	497.754	—	2	0.800
ModelNet10	4,899	1,024	∼19,440	—	10	0.815

We give detailed descriptions and statistics (*cf.* Table 5) of the datasets used in our experiments:

Citation Networks. In the citation network datasets Cora, Citeseer and Pubmed nodes represent documents and edges represent citation links. The networks contain bag-of-words feature vectors for each document. We treat the citation links as (undirected) edges. For training, we use 20 labels per class.

Social Network Datasets. COLLAB is derived from three public scientific collaboration datasets. Each graph corresponds to an ego-network of different researchers from each field with the task to classify each graph to the field the corresponding researcher belongs to. IMDB-BINARY is a movie collaboration dataset where each graph corresponds to an ego-network of actors/actresses. An edge is drawn between two actors/actresses if they appear in the same movie. The task is to classify the genre of the graph. REDDIT-BINARY is a online discussion dataset where each graph corresponds to a thread. An edge is drawn between two users if one of them responded to another's comment. The task is to classify each graph to the community/subreddit it belongs to.

Bioinformatic Datasets. MUTAG is a dataset consisting of mutagenetic aromatic and heteroaromatic nitro compounds. PROTEINS holds a set of proteins represented by graphs. Nodes represent secondary structure elements (SSEs) which are connected whenever there are neighbors either in the amino acid sequence or in 3D space.

3D Object Datasets. ModelNet10 is an orientation-aligned dataset of CAD models. Each model corresponds to exactly one out of 10 object categories. Categories were chosen based on a list of the most common object categories in the world.

High-Fidelity Image Generation With Fewer Labels

Mario Lucic [*1] **Michael Tschannen** [*2] **Marvin Ritter** [*1] **Xiaohua Zhai** [1] **Olivier Bachem** [1] **Sylvain Gelly** [1]

Abstract

Deep generative models are becoming a cornerstone of modern machine learning. Recent work on conditional generative adversarial networks has shown that learning complex, high-dimensional distributions over natural images is within reach. While the latest models are able to generate high-fidelity, diverse natural images at high resolution, they rely on a vast quantity of labeled data. In this work we demonstrate how one can benefit from recent work on self- and semi-supervised learning to outperform state-of-the-art (SOTA) on both unsupervised ImageNet synthesis, as well as in the conditional setting. In particular, the proposed approach is able to match the sample quality (as measured by FID) of the current state-of-the art conditional model BigGAN on ImageNet *using only 10% of the labels* and outperform it using 20% of the labels.

1. Introduction

Deep generative models have received a great deal of attention due to their power to learn complex high-dimensional distributions, such as distributions over natural images (Zhang et al., 2018; Brock et al., 2019), videos (Kalchbrenner et al., 2017), and audio (Van Den Oord et al., 2016). Recent progress was driven by scalable training of large-scale models (Brock et al., 2019; Menick & Kalchbrenner, 2019), architectural modifications (Zhang et al., 2018; Chen et al., 2019a; Karras et al., 2018), and normalization techniques (Miyato et al., 2018).

High-fidelity natural image generation (typically trained on ImageNet) hinges upon having access to vast quantities of labeled data. This is unsurprising as labels induce rich side information into the training process, effectively dividing the extremely challenging image generation task into semantically meaningful sub-tasks.

[*]Equal contribution [1]Google Brain, Zurich, Switzerland [2]ETH Zurich, Zurich, Switzerland. Correspondence to: Mario Lucic <lucic@google.com>, Michael Tschannen <mi.tschannen@gmail.com>, Marvin Ritter <marvinritter@google.com>.

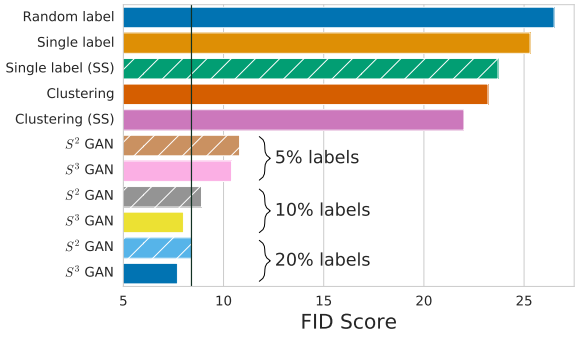


Figure 1. FID of the baselines and the proposed method. The vertical line indicates the baseline (BigGAN) which uses all the labeled data. The proposed method (S^3GAN) is able to match the state-of-the-art while using only 10% of the labeled data and outperform it with 20%.

However, this dependence on vast quantities of labeled data is at odds with the fact that most data is unlabeled, and labeling itself is often costly and error-prone. Despite the recent progress on unsupervised image generation, the gap between conditional and unsupervised models in terms of sample quality is significant.

In this work, we take a significant step towards closing the gap between conditional and unsupervised generation of high-fidelity images using generative adversarial networks (GANs). We leverage two simple yet powerful concepts:

(i) Self-supervised learning: A semantic feature extractor for the training data can be learned via self-supervision, and the resulting feature representation can then be employed to guide the GAN training process.

(ii) Semi-supervised learning: Labels for the entire training set can be inferred from a small subset of labeled training images and the inferred labels can be used as conditional information for GAN training.

Our contributions In this work, we

1. propose and study various approaches to reduce or fully omit ground-truth label information for natural image generation tasks,
2. achieve a new SOTA in unsupervised generation on ImageNet, match the SOTA on 128×128 IMAGENET using only 10% of the labels, and set a new SOTA using only 20% of the labels (measured by FID), and
3. open-source all the code used for the experiments at `github.com/google/compare_gan`.

2. Background and related work

High-fidelity GANs on IMAGENET Besides BIGGAN (Brock et al., 2019) only a few prior methods have managed to scale GANs to ImageNet, most of them relying on class-conditional generation using labels. One of the earliest attempts are GANs with auxiliary classifier (ACGANs) (Odena et al., 2017) which feed one-hot encoded label-information with the latent code to the generator and equip the discriminator with an auxiliary head predicting the image class in addition to whether the input is real or fake. More recent approaches rely on a label projection layer in the discriminator essentially resulting in per-class real/fake classification (Miyato & Koyama, 2018) and self-attention in the generator (Zhang et al., 2018). Both methods use modulated batch normalization (De Vries et al., 2017) to provide label information to the generator. On the unsupervised side, Chen et al. (2019b) showed that auxiliary rotation loss added to the discriminator has a stabilizing effect on the training. Finally, appropriate gradient regularization enables scaling MMD-GANs to ImageNet without using labels (Arbel et al., 2018).

Semi-supervised GANs Several recent works leveraged GANs for semi-supervised learning of classifiers. Both Salimans et al. (2016) and Odena (2016) train a discriminator that classifies its input into $K + 1$ classes: K image classes for real images, and one class for generated images. Similarly, Springenberg (2016) extends the standard GAN objective to K classes. This approach was also considered by Li et al. (2017) where separate discriminator and classifier models are applied. Other approaches incorporate inference models to predict missing labels (Deng et al., 2017) or harness the joint distribution (of labels and data) matching for semi-supervised learning (Gan et al., 2017). We emphasize that this line of work focuses on training a classifier from a few labels, rather than using few labels to improve the quality of the generated model. Up to our knowledge, improvements in sample quality through partial label information are reported in Li et al. (2017); Deng et al. (2017); Sricharan et al. (2017), all of which consider only low-resolution data sets from a restricted domain.

Self-supervised learning Self-supervised learning methods employ a label-free auxiliary task to learn a semantic feature representation of the data. This approach was successfully applied to different data modalities, such as images (Doersch et al., 2015; Caron et al., 2018), video (Agrawal et al., 2015; Lee et al., 2017), and robotics (Jang et al., 2018; Pinto & Gupta, 2016). The current state-of-the-art method on IMAGENET is due to Gidaris et al. (2018) who proposed predicting the rotation angle of rotated training images as an auxiliary task. This simple self-supervision approach yields representations which are useful for downstream image classification tasks. Other forms of self-supervision include predicting relative locations of disjoint image patches of a given image (Doersch et al., 2015; Mundhenk et al., 2018) or estimating the permutation of randomly swapped image patches on a regular grid (Noroozi & Favaro, 2016). A study on self-supervised learning with modern neural architectures is provided in Kolesnikov et al. (2019).

Figure 2. Top row: 128×128 samples from the fully supervised current state-of-the-art model BIGGAN. Bottom row: Samples form the proposed S^3GAN which matches BIGGAN in terms of FID and IS using only 10% of the ground-truth labels.

3. Reducing the appetite for labeled data

In a nutshell, instead of providing hand-annotated ground truth labels for real images to the discriminator, we will provide inferred ones. To obtain these labels we will make use of recent advancements in self- and semi-supervised learning. Before introducing these methods in detail, we first discuss how label information is used in state-of-the-art GANs. The following exposition assumes familiarity with the basics of the GAN framework (Goodfellow et al., 2014).

Incorporating the labels To provide the label information to the discriminator we employ a linear projection layer as proposed by Miyato & Koyama (2018). To make the exposition self-contained, we will briefly recall the main ideas. In a "vanilla" (unconditional) GAN, the discriminator D learns to predict whether the image at its input x is real or generated by the generator G. We decompose the discriminator into a learned discriminator representation, $\tilde{D}$, which is fed into a linear classifier, $c_{\text{r/f}}$, i.e., the discriminator is given by $c_{\text{r/f}}(\tilde{D}(x))$. In the *projection discriminator*, one learns an embedding for each class of the same dimension as the representation $\tilde{D}(x)$. Then, for a given image, label input x, y the decision on whether the sample is real or generated is based on two components: (a) on whether the representation $\tilde{D}(x)$ itself is consistent with the real data, and (b) on whether the representation $\tilde{D}(x)$ is consistent with the real data *from class* y. More formally, the discriminator takes the form $D(x, y) = c_{\text{r/f}}(\tilde{D}(x)) + P(\tilde{D}(x), y)$, where $P(\tilde{x}, y) = \tilde{x}^\top W y$ is a linear projection layer applied to a feature vector $\tilde{x}$ and the one-hot encoded label y as an

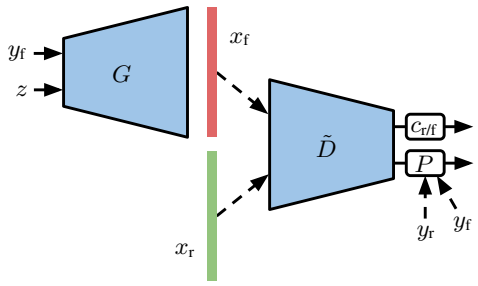


Figure 3. Conditional GAN with projection discriminator. The discriminator tries to predict from the representation $\tilde{D}$ whether a real image x_r (with label y_r) or a generated image x_f (with label y_f) is at its input, by combining an unconditional classifier $c_{r/f}$ and a class-conditional classifier implemented through the projection layer P. This form of conditioning is used in BIGGAN. Outward-pointing arrows feed into losses.

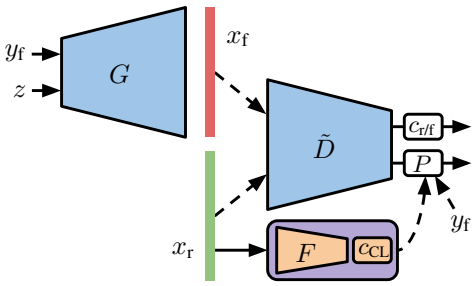


Figure 4. CLUSTERING: Unsupervised approach based on clustering the representations obtained by solving a self-supervised task. F corresponds to the feature extractor learned via self-supervision and c_{CL} is the cluster assignment function. After learning F and c_{CL} on the real training images in the pre-training step, we proceed with conditional GAN training by inferring the labels as $\hat{y}_{\text{CL}} = c_{\text{CL}}(F(x))$.

input. As for the generator, the label information y is incorporated through class-conditional BatchNorm (Dumoulin et al., 2017; De Vries et al., 2017). The conditional GAN with projection discriminator is illustrated in Figure 3. We proceed with describing the pre-trained and co-training approaches to infer labels for GAN training in Sections 3.1 and 3.2, respectively.

3.1. Pre-trained approaches

Unsupervised clustering-based method We first learn a representation of the real training data using a state-of-the-art self-supervised approach (Gidaris et al., 2018; Kolesnikov et al., 2019), perform clustering on this representation, and use the cluster assignments as a replacement for labels. Following Gidaris et al. (2018) we learn the feature extractor F (typically a convolutional neural network) by minimizing the following *self-supervision loss*

$$\mathcal{L}_{\text{R}} = -\frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}} \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log p(c_{\text{R}}(F(x^r)) = r)], \quad (1)$$

where $\mathcal{R}$ is the set of the 4 rotation degrees $\{0°, 90°, 180°, 270°\}$, x^r is the image x rotated by r, and c_{R} is a linear classifier predicting the rotation degree r. After learning the feature extractor F, we apply mini batch k-Means clustering (Sculley, 2010) on the representations of the training images. Finally, given the cluster assignment function $\hat{y}_{\text{CL}} = c_{\text{CL}}(F(x))$ we train the GAN using the hinge loss, alternatively minimizing the discriminator loss $\mathcal{L}_D$ and generator loss $\mathcal{L}_G$, namely

$$\begin{aligned}\mathcal{L}_D &= -\mathbb{E}_{x\sim p_{\text{data}}(x)}[\min(0, -1 + D(x, c_{\text{CL}}(F(x))))] \\ &\quad - \mathbb{E}_{(z,y)\sim\hat{p}(z,y)}[\min(0, -1 - D(G(z,y), y))] \\ \mathcal{L}_G &= -\mathbb{E}_{(z,y)\sim\hat{p}(z,y)}[D(G(z,y), y)],\end{aligned}$$

where $\hat{p}(z,y) = p(z)\hat{p}(y)$ is the prior distribution with $p(z) = \mathcal{N}(0, I)$ and $\hat{p}(y)$ the empirical distribution of the cluster labels $c_{\text{CL}}(F(x))$ over the training set. We call this approach CLUSTERING and illustrate it in Figure 4.

Semi-supervised method While semi-supervised learning is an active area of research and a large variety of algorithms has been proposed, we follow Beyer et al. (2019) and simply extend the self-supervised approach described in the previous paragraph with a semi-supervised loss. This ensures that the two approaches are comparable in terms of model capacity and computational cost. Assuming we are provided with labels for a subset of the training data, we attempt to learn a good feature representation via self-supervision and simultaneously train a good linear classifier on the so-obtained representation (using the provided labels).[1] More formally, we minimize the loss

$$\begin{aligned}\mathcal{L}_{\text{S}^2\text{L}} = -\frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}} \Big\{&\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log p(c_{\text{R}}(F(x^r)) = r)] \\ &+ \gamma\mathbb{E}_{(x,y)\sim p_{\text{data}}(x,y)}[\log p(c_{\text{S}^2\text{L}}(F(x^r)) = y)]\Big\},\end{aligned} \quad (2)$$

where c_{R} and $c_{\text{S}^2\text{L}}$ are linear classifiers predicting the rotation angle r and the label y, respectively, and $\gamma > 0$ balances the loss terms. The first term in (2) corresponds to the self-supervision loss from (1) and the second term to a (semi-supervised) cross-entropy loss. During training, the latter expectation is replaced by the empirical average over the subset of labeled training examples, whereas the former is set to the empirical average over the entire training set (this convention is followed throughout the paper). After we obtain F and $c_{\text{S}^2\text{L}}$ we proceed with GAN training where we

[1] Note that an even simpler approach would be to first learn the representation via self-supervision and *subsequently* the linear classifier, but we observed that learning the representation and classifier simultaneously leads to better results.

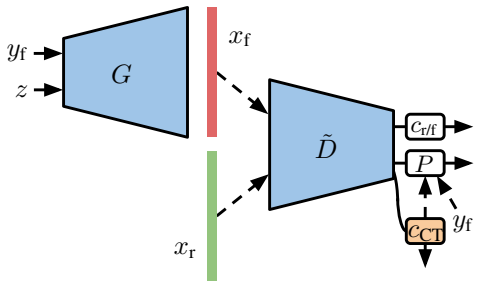


Figure 5. S^{2}GAN-CO: During GAN training we learn an auxiliary classifier c_{CT} on the discriminator representation $\tilde{D}$, based on the labeled real examples, to predict labels for the unlabeled ones. This avoids training a feature extractor F and classifier c_{S^2L} prior to GAN training as in S^{2}GAN.

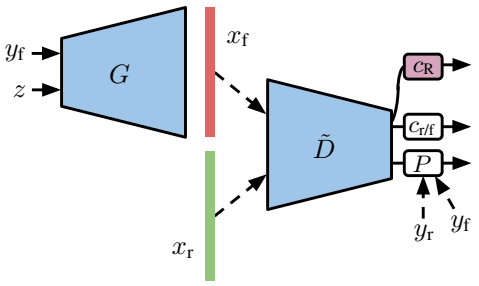


Figure 6. Self-supervision by rotation-prediction during GAN training. Additionally to predicting whether the images at its input are real or generated, the discriminator is trained to predict rotations of both rotated real and fake images via an auxiliary linear classifier c_R. This approach was successfully applied by Chen et al. (2019b) to stabilize GAN training. Here we combine it with our pre-trained and co-training approaches, replacing the ground truth labels y_r with predicted ones.

label the real images as $\hat{y}_{S^2L} = c_{S^2L}(F(x))$. In particular, we alternatively minimize the same generator and discriminator losses as for CLUSTERING except that we use c_{S^2L} and F obtained by minimizing (2):

$$\begin{aligned}\mathcal{L}_D = &-\mathbb{E}_{x\sim p_{\text{data}}(x)}[\min(0, -1 + D(x, c_{S^2L}(F(x))))] \\ &- \mathbb{E}_{(z,y)\sim p(z,y)}[\min(0, -1 - D(G(z,y), y))] \\ \mathcal{L}_G = &-\mathbb{E}_{(z,y)\sim p(z,y)}[D(G(z,y), y)],\end{aligned}$$

where $p(z, y) = p(z)p(y)$ with $p(z) = \mathcal{N}(0, I)$ and $p(y)$ uniform categorical. We use the abbreviation S^{2}GAN for this method.

3.2. Co-training approach

The main drawback of the transfer-based methods is that one needs to train a feature extractor F via self supervision and learn an inference mechanism for the labels (linear classifier or clustering). In what follows we detail co-training approaches that avoid this two-step procedure and learn to infer label information during GAN training.

Unsupervised method We consider two approaches. In the first one, we completely remove the labels by simply labeling all real and generated examples with the same label[2] and removing the projection layer from the discriminator, i.e., we set $D(x) = c_{r/f}(\tilde{D}(x))$. We use the abbreviation SINGLE LABEL for this method. For the second approach we assign random labels to (unlabeled) real images. While the labels for the real images do not provide any useful signal to the discriminator, the sampled labels could potentially help the generator by providing additional randomness with different statistics than z, as well as additional trainable parameters due to the embedding matrices in class-conditional BatchNorm. Furthermore, the labels for the fake data could facilitate the discrimination as they provide side information about the fake images to the discriminator. We term this method RANDOM LABEL.

Semi-supervised method When labels are available for a subset of the real data, we train an auxiliary linear classifier c_{CT} directly on the feature representation $\tilde{D}$ of the discriminator, *during GAN training*, and use it to predict labels for the unlabeled real images. In this case the discriminator loss takes the form

$$\begin{aligned}\mathcal{L}_D = &-\mathbb{E}_{(x,y)\sim p_{\text{data}}(x,y)}[\min(0, -1 + D(x,y))] \\ &- \lambda\mathbb{E}_{(x,y)\sim p_{\text{data}}(x,y)}[\log p(c_{CT}(\tilde{D}(x)) = y)] \\ &- \mathbb{E}_{x\sim p_{\text{data}}(x)}[\min(0, -1 + D(x, c_{CT}(\tilde{D}(x))))] \\ &- \mathbb{E}_{(z,y)\sim p(z,y)}[\min(0, -1 - D(G(z,y), y))], \quad (3)\end{aligned}$$

where the first term corresponds to standard conditional training on ($k\%$) labeled real images, the second term is the cross-entropy loss (with weight $\lambda > 0$) for the auxiliary classifier c_{CT} on the labeled real images, the third term is an unsupervised discriminator loss where the labels for the unlabeled real images are predicted by c_{CT}, and the last term is the standard conditional discriminator loss on the generated data. We use the abbreviation S^{2}GAN-CO for this method. See Figure 5 for an illustration.

3.3. Self-supervision during GAN training

So far we leveraged self-supervision to either craft good feature representations, or to learn a semi-supervised model (cf. Section 3.1). However, given that the discriminator itself is just a classifier, one may benefit from augmenting this classifier with an auxiliary task—namely self-supervision through rotation prediction. This approach was already explored in (Chen et al., 2019b), where it was observed to stabilize GAN training. Here we want to assess its impact

[2]Note that this is not necessarily equivalent to replacing class-conditional BatchNorm with standard (unconditional) BatchNorm as the variant of conditional BatchNorm used in this paper also uses chunks of the latent code as input; besides the label information.

when combined with the methods introduced in Sections 3.1 and 3.2. To this end, similarly to the training of F in (1) and (2), we train an additional linear classifier c_{R} on the discriminator feature representation $\tilde{D}$ to predict rotations $r \in \mathcal{R}$ of the rotated real images x^r and rotated fake images $G(z, y)^r$. The corresponding loss terms added to the discriminator and generator losses are

$$-\frac{\beta}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \mathbb{E}_{x \sim p_{\mathrm{data}}(x)}[\log p(c_{\mathrm{R}}(\tilde{D}(x^r) = r)] \quad (4)$$

and

$$-\frac{\alpha}{|\mathcal{R}|} \mathbb{E}_{(z,y) \sim p(z,y)}[\log p(c_{\mathrm{R}}(\tilde{D}(G(z, y)^r) = r)], \quad (5)$$

respectively, where $\alpha, \beta > 0$ are weights to balance the loss terms. This approach is illustrated in Figure 6.

4. Experimental setup

Architecture and hyperparameters GANs are notoriously unstable to train and their performance strongly depends on the capacity of the neural architecture, optimization hyperparameters, and appropriate regularization (Lucic et al., 2018; Kurach et al., 2018). We implemented the conditional BigGAN architecture (Brock et al., 2019) which achieves state-of-the-art results on ImageNet.[3] We use exactly the same optimization hyper-parameters as Brock et al. (2019). Specifically, we employ the Adam Optimizer with the learning rates $5 \cdot 10^{-5}$ for the generator and $2 \cdot 10^{-4}$ for the discriminator ($\beta_1 = 0$ $\beta_2 = 0.999$). We train for 250k generator steps with 2 discriminator iterations before each generator step. The batch size was fixed to 2048, and we use a latent code z with 120 dimensions. We employ spectral normalization in both generator and discriminator. In contrast to BigGAN, we do not apply orthogonal regularization as this was observed to only marginally improve sample quality (cf. Table 1 in Brock et al. (2019)) and we do not use the truncation trick.

Datasets We focus primarily on IMAGENET, the largest and most diverse image data set commonly used to evaluate GANs. IMAGENET contains 1.3M training images and 50k test images, each corresponding to one of 1k object classes. We resize the images to $128 \times 128 \times 3$ as done in Miyato & Koyama (2018) and Zhang et al. (2018). Partially labeled data sets for the semi-supervised approaches are obtained by randomly selecting $k\%$ of the samples from each class.

[3] We dissected the model checkpoints released by Brock et al. (2019) to obtain exact counts of trainable parameters and their dimensions, and match them to *byte* level (cf. Tables 10 and 11). We want to emphasize that at this point this methodology is *bleeding-edge* and successful state-of-the-art methods require careful architecture-level tuning. To foster reproducibility we meticulously detail this architecture at tensor-level detail in Appendix B and open-source our code at https://github.com/google/compare_gan.

Evaluation metrics We use the Fréchet Inception Distance (FID) (Heusel et al., 2017) and Inception Score (Salimans et al., 2016) to evaluate the quality of the generated samples. To compute the FID, the real data and generated samples are first embedded in a specific layer of a pre-trained Inception network. Then, a multivariate Gaussian is fit to the data and the distance computed as $\mathrm{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \mathrm{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}})$, where μ and Σ denote the empirical mean and covariance, and subscripts x and g denote the real and generated data respectively. FID was shown to be sensitive to both the addition of spurious modes and to mode dropping (Sajjadi et al., 2018; Lucic et al., 2018). Inception Score posits that conditional label distribution of samples containing meaningful objects should have low entropy, and the variability of the samples should be high leading to the following formulation: $\mathrm{IS} = \exp(\mathbb{E}_{x \sim Q}[d_{KL}(p(y \mid x), p(y))])$. Although it has some flaws (Barratt & Sharma, 2018), we report it to enable comparison with existing methods. Following (Brock et al., 2019), the FID is computed using the 50k IMAGENET testing images and 50k randomly sampled fake images, and the IS is computed from 50k randomly sampled fake images. All metrics are computed for 5 different randomly sampled sets of fake images and we report the mean.

Methods We conduct an extensive comparison of methods detailed in Table 1, namely: Unmodified BIGGAN, the unsupervised methods SINGLE LABEL, RANDOM LABEL, CLUSTERING, and the semi-supervised methods S^2GAN and S^2GAN-CO. In all S^2GAN-CO experiments we use soft labels, i.e., the soft-max output of c_{CT} instead of one-hot encoded hard estimates, as we observed in preliminary experiments that this stabilizes training. For S^2GAN we use hard labels by default, but investigate the effect of soft labels in separate experiments. For all semi-supervised methods we have access only to $k\%$ of the ground truth labels where $k \in \{5, 10, 20\}$. As an additional baseline, we retain $k\%$ labeled real images and discard all unlabeled real images, then using the remaining labeled images to train BIGGAN (the resulting model is designated by BIGGAN-$k\%$). Finally, we explore the effect of self-supervision during GAN training on the unsupervised and semi-supervised methods.

We train every model three times with a different random seed and report the median FID and the median IS. With the exception of the SINGLE LABEL and BIGGAN-$k\%$, the standard deviation of the mean across three runs is very low. We therefore defer tables with the mean FID and IS values and standard deviations to Appendix D. All models are trained on 128 cores of a Google TPU v3 Pod with BatchNorm statistics synchronized across cores.

Unsupervised approaches For CLUSTERING we simply used the best available self-supervised rotation model from (Kolesnikov et al., 2019). The number

Table 1. A short summary of the analyzed methods. The detailed descriptions of pre-training and co-trained approaches can be found in Sections 3.1 and 3.2, respectively. Self-supervision during GAN training is described in Section 3.3.

METHOD	DESCRIPTION
BIGGAN	Conditional (Brock et al., 2019)
SINGLE LABEL	Co-training: Single label
RANDOM LABEL	Co-training: Random labels
CLUSTERING	Pre-trained: Clustering
BIGGAN-$k\%$	Drop all but $k\%$ labeled data
S^2GAN-CO	Co-training: Semi-supervised
S^2GAN	Pre-trained: Semi-supervised
S^3GAN	S^2GAN with self-supervision
S^3GAN-CO	S^2GAN-CO with self-supervision

of clusters for CLUSTERING is selected from the set $\{50, 100, 200, 500, 1000\}$. The other unsupervised approaches do not have hyper-parameters.

Pre-trained and co-training approaches We employ the wide ResNet-50 v2 architecture with widening factor 16 (Zagoruyko & Komodakis, 2016) for the feature extractor F in the pre-trained approaches described in Section 3.1. We optimize the loss in (2) using SGD for 65 epochs. The batch size is set to 2048, composed of B unlabeled examples and $2048 - B$ labeled examples. Following the recommendations from Goyal et al. (2017) for training with large batch size, we (i) set the learning rate to $0.1\frac{B}{256}$, and (ii) use linear learning rate warm-up during the initial 5 epochs. The learning rate is decayed twice with a factor of 10 at epoch 45 and epoch 55. The parameter γ in (2) is set to 0.5 and the number of unlabeled examples per batch B is 1536. The parameters γ and B are tuned on 0.1% labeled examples held out from the training set, the search space is $\{0.1, 0.5, 1.0\} \times \{1024, 1536, 1792\}$. The accuracy of the so-obtained classifier $c_{\text{S}^2\text{L}}(F(x))$ on the IMAGENET validation set is reported in Table 3. The parameter λ in the loss used for S^2GAN-CO in (3) is selected form the set $\{0.1, 0.2, 0.4\}$.

Self-supervision during GAN training For all approaches we use the recommend parameter $\alpha = 0.2$ from (Chen et al., 2019b) in (5) and do a small sweep for β in (4). For the values tried ($\{0.25, 0.5, 1.0, 2\}$) we do not see a huge effect and use $\beta = 0.5$ for S^3GAN. For S^3GAN-CO we did not repeat the sweep used $\beta = 1.0$.

5. Results and discussion

Recall that the main goal of this work is to match (or outperform) the fully supervised BIGGAN in an unsupervised fashion, or with a small subset of labeled data. In the following, we discuss the advantages and drawbacks of the analyzed approaches with respect to this goal.

As a baseline, our reimplementation of BIGGAN obtains an FID of 8.4 and IS of 75.0, and hence reproduces the result reported by Brock et al. (2019) in terms of FID. We observed some differences in training dynamics, which we discuss in detail in Section 5.4.

5.1. Unsupervised approaches

The results for unsupervised approaches are summarized in Figure 7 and Table 2. The fully unsupervised RANDOM LABEL and SINGLE LABEL models both achieve a similar FID of ~ 25 and IS of ~ 20. This is a quite considerable gap compared to BIGGAN and indicates that additional supervision is necessary. We note that one of the three SINGLE LABEL models collapsed whereas all three RANDOM LABEL models trained stably for 250k generator iterations.

Pre-training a semantic representation using self-supervision and clustering the training data on this representation as done by CLUSTERING reduces the FID by about 10% and increases IS by about 10%. These results were obtained for 50 clusters, all other options led to worse results. While this performance is still considerably worse than that of BIGGAN this result is the current state-of-the-art in unsupervised image generation (Chen et al. (2019b) report an FID of 33 for unsupervised generation).

Example images from the clustering are shown in Figures 14, 15, and 16 in the supplementary material. The clustering is clearly meaningful and groups similar objects within the same cluster. Furthermore, the objects generated by CLUSTERING conditionally on a given cluster index reflect the distribution of the training data belonging the corresponding cluster. On the other hand, we can clearly observe multiple classes being present in the same cluster. This is to be expected when under-clustering to 50 clusters. Interestingly, clustering to many more clusters (say 500) yields results similar to SINGLE LABEL.

Table 2. Median FID and IS for the unsupervised approaches (see Table 14 in the appendix for mean and standard deviation).

	FID	IS
RANDOM LABEL	26.5	20.2
SINGLE LABEL	25.3	20.4
SINGLE LABEL (SS)	23.7	22.2
CLUSTERING	23.2	22.7
CLUSTERING (SS)	22.0	23.5

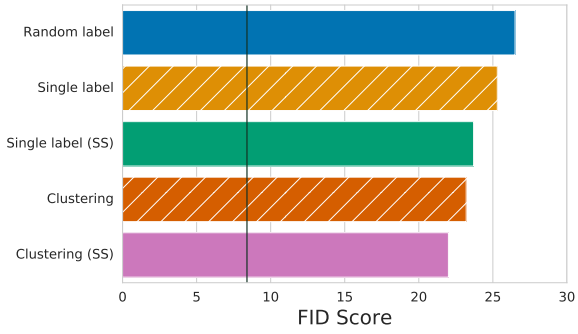


Figure 7. Median FID obtained by our unsupervised approaches. The vertical line indicates the the median FID of our BIGGAN implementation which uses labels for all training images. While the gap between unsupervised and fully supervised approaches remains significant, using a pre-trained self-supervised representation (CLUSTERING) improves the sample quality compared to SINGLE LABEL and RANDOM LABEL, leading to a new state-of-the art in unsupervised generation on IMAGENET.

5.2. Semi-supervised approaches

Pre-trained The S^{2}GAN model where we use the classifier pre-trained with both a self-supervised and semi-supervised loss (cf. Section 3.1) suffers a very minor increase in FID for 10% and 5% labeled real training data, and matches BIGGAN both in terms of FID and IS when 20% of the labels are used (cf. Table 3). We stress that this is despite the fact that the classifier used to infer the labels has a top-1 accuracy of only 50%, 63%, and 71% for 5%, 10%, and 20% labeled data, respectively (cf. Table 3), compared to 100% of the original labels. The results are shown in Table 4 and Figure 8, and random samples as well as interpolations can be found in Figures 9–17 in the supplementary material.

Co-trained The results for our co-trained model S^{2}GAN-CO which trains a linear classifier in semi-supervised fashion on top of the discriminator representation during GAN training (cf. Section 3.2) are shown in Table 4. It can be seen that S^{2}GAN-CO outperforms all fully unsupervised approaches for all considered label percentages. While the gap between S^{2}GAN-CO with 5% labels and CLUSTERING in terms of FID is small, S^{2}GAN-CO has a considerably larger IS. When using 20% labeled training examples S^{2}GAN-CO obtains an FID of 13.9 and an IS of 49.2, which is remarkably close to BIGGAN and S^{2}GAN given the simplicity of the S^{2}GAN-CO approach. As the the percentage of labels decreases, the gap between S^{2}GAN and S^{2}GAN-CO increases.

Interestingly, S^{2}GAN-CO does not seem to train less stably than S^{2}GAN approaches even though it is forced to learn the classifier during GAN training. This is particularly remarkable as the BIGGAN-k% approaches, where we only retain the labeled data for training and discard all unlabeled data, *are very unstable and collapse after 60k to 120k iterations*, for all three random seeds and for both 10% and 20% labeled data.

Table 3. Top-1 and top-5 error rate (%) on the IMAGENET validation set of $c_{S^2L}(F(x))$ using both self- and semi-supervised losses as described in Section 3.1. While the models are clearly not state-of-the-art compared to the fully supervised IMAGENET classification task, the quality of labels is sufficient to match and in some cases improve the state-of-the-art GAN natural image synthesis.

	LABELS		
METRIC	5%	10%	20%
TOP-1 ERROR	50.08	36.74	29.21
TOP-5 ERROR	26.94	16.04	10.33

Table 4. Pre-trained vs co-training approaches, and the effect of self-supervision during GAN training (see Table 12 in the appendix for mean and standard deviation). While co-training approaches outperform fully unsupervised approaches, they are clearly outperformed by the pre-trained approaches. Self-supervision during GAN training helps in all cases.

	FID			IS		
	5%	10%	20%	5%	10%	20%
S^{2}GAN	10.8	8.9	8.4	57.6	73.4	77.4
S^{2}GAN-CO	21.8	17.7	13.9	30.0	37.2	49.2
S^{3}GAN	10.4	8.0	7.7	59.6	78.7	83.1
S^{3}GAN-CO	20.2	16.6	12.7	31.0	38.5	53.1

5.3. Self-supervision during GAN training

So far we have seen that the pre-trained semi-supervised approach, namely S^{2}GAN, is able to achieve state-of-the-art performance for 20% labeled data. Here we investigate whether self-supervision during GAN training as described in Section 3.3 can lead to further improvements. Table 4 and Figure 8 show the experimental results for S^{3}GAN, namely S^{2}GAN coupled with self-supervision in the discriminator.

Self-supervision leads to a reduction in FID and increase in IS across all considered settings. In particular *we can match the state-of-the-art* BIGGAN *with only* 10% *of the labels and outperform it using* 20% *labels, both in terms of FID and IS.*

For S^{3}GAN the improvements due to self-supervision during GAN training in FID are considerable, around 10% in most of the cases. Tuning the parameter β of the discriminator self-supervision loss in (4) did not dramatically increase

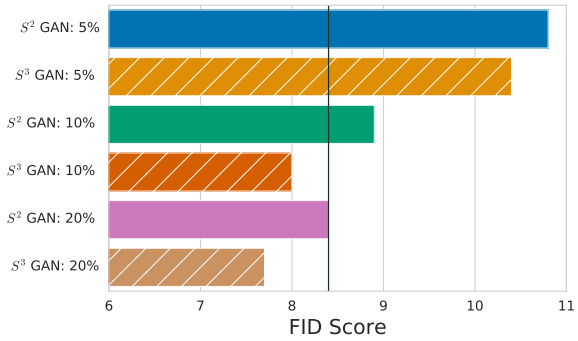


Figure 8. The vertical line indicates the median FID of our BIGGAN implementation which uses all labeled data. The proposed S^{3}GAN approach is able to match the performance of the state-of-the-art BIGGAN model using 10% of the ground-truth labels and outperforms it using 20%.

the benefits of self-supervision during GAN training, at least for the range of values considered. As shown in Tables 2 and 4, self-supervision during GAN training (with default parameters α, β) also leads to improvements by 5 to 10% for both S^{2}GAN-CO and SINGLE LABEL. In summary, self-supervision during GAN training with default parameters leads to a stable improvement across all approaches.

5.4. Other insights

Effect of soft labels A design choice available to practitioners is whether to use hard labels (i.e., the argmax over the logits), or soft labels (softmax over the logits) for S^{2}GAN and S^{3}GAN (recall that we use soft labels by default for S^{2}GAN-CO and S^{3}GAN-CO). Our initial expectation was that soft labels should help when very little labeled data is available, as soft labels carry more information which can potentially be exploited by the projection discriminator. Surprisingly, the results presented in Table 5 show clearly that the opposite is true. Our current hypothesis is that this is due to the way labels are incorporated in the projection discriminator, but we do not have empirical evidence yet.

Optimization dynamics Brock et al. (2019) report the FID and IS of the model *just before the collapse*, which can be seen as a form of early stopping. In contrast, we manage to stably train the proposed models for 250k generator iterations. In particular, we also observe stable training for our vanilla BIGGAN implementation. The evolution of the FID and IS as a function of the training steps is shown in Figure 21 in the appendix. At this point we can only speculate about the origin of this difference.

Higher resolution and going below 5% labels Training these models at higher resolution becomes computationally harder and it necessitates tuning the learning rate. We trained several S^{3}GAN models at 256×256 resolution and show the resulting samples in Figures 12–13 and interpolations in Figures 19–20. We also conducted S^{3}GAN experiments in which only 2.5% of the labels are used and observed FID of 13.6 and IS of 46.3. This indicates that given a small number of samples one can significantly outperform the unsupervised approaches (c.f. Figure 7).

Table 5. Training with hard (predicted) labels leads to better models than training with soft (predicted) labels (see Table 13 in the appendix for mean and standard deviation).

	FID			IS		
	5%	10%	20%	5%	10%	20%
S^{2}GAN	10.8	8.9	8.4	57.6	73.4	77.4
+SOFT	15.4	12.9	10.4	40.3	49.8	62.1

6. Conclusion and future Work

In this work we investigated several avenues to reduce the appetite for labeled data in state-of-the-art generative adversarial networks. We showed that recent advances in self- and semi-supervised learning can be used to achieve a new state of the art, both for unsupervised and supervised natural image synthesis.

We believe that this is a great first step towards the ultimate goal of few-shot high-fidelity image synthesis. There are several important directions for future work: (i) investigating the applicability of these techniques for even larger and more diverse data sets, and (ii) investigating the impact of other self- and semi-supervised approaches on the model quality. (iii) investigating the impact of self-supervision in other deep generative models. Finally, we would like to emphasize that further progress might be hindered by the engineering challenges related to training large-scale generative adversarial networks. To help alleviate this issue and to foster reproducibility, we have open-sourced all the code used for the experiments.

Acknowledgments

We would like to thank Ting Chen and Neil Houlsby for fruitful discussions on self-supervision and its application to GANs. We would like to thank Lucas Beyer, Alexander Kolesnikov, and Avital Oliver for helpful discussions on self-supervised semi-supervised learning. We would like to thank Karol Kurach and Marcin Michalski their major contributions the Compare GAN library. We would also like to thank the BigGAN team (Andy Brock, Jeff Donahue, and Karen Simonyan) for their insights into training GANs on TPUs. Finally, we are grateful for the support of members of the Google Brain team in Zurich.

References

Agrawal, P., Carreira, J., and Malik, J. Learning to see by moving. In *International Conference on Computer Vision*, 2015.

Arbel, M., Sutherland, D., Bińkowski, M. a., and Gretton, A. On gradient regularizers for mmd gans. In *Advances in Neural Information Processing Systems*. 2018.

Barratt, S. and Sharma, R. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

Beyer, L., Kolesnikov, A., Oliver, A., Xiaohua, Z., and Gelly, S. Self-supervised Semi-supervised Learning. In *Manuscript in preparation*, 2019.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

Caron, M., Bojanowski, P., Joulin, A., and Douze, M. Deep clustering for unsupervised learning of visual features. *European Conference on Computer Vision*, 2018.

Chen, T., Lucic, M., Houlsby, N., and Gelly, S. On self modulation for generative adversarial networks. In *International Conference on Learning Representations*, 2019a.

Chen, T., Zhai, X., Ritter, M., Lucic, M., and Houlsby, N. Self-Supervised GANs via Auxiliary Rotation Loss. In *Computer Vision and Pattern Recognition*, 2019b.

De Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., and Courville, A. C. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems*, 2017.

Deng, Z., Zhang, H., Liang, X., Yang, L., Xu, S., Zhu, J., and Xing, E. P. Structured Generative Adversarial Networks. In *Advances in Neural Information Processing Systems*, 2017.

Doersch, C., Gupta, A., and Efros, A. A. Unsupervised visual representation learning by context prediction. In *International Conference on Computer Vision*, 2015.

Dumoulin, V., Shlens, J., and Kudlur, M. A learned representation for artistic style. In *International Conference on Learning Representations*, 2017.

Gan, Z., Chen, L., Wang, W., Pu, Y., Zhang, Y., Liu, H., Li, C., and Carin, L. Triangle generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2017.

Gidaris, S., Singh, P., and Komodakis, N. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G., and Hochreiter, S. GANs trained by a two time-scale update rule converge to a Nash equilibrium. In *Advances in Neural Information Processing Systems*, 2017.

Jang, E., Devin, C., Vanhoucke, V., and Levine, S. Grasp2Vec: Learning Object Representations from Self-Supervised Grasping. In *Conference on Robot Learning*, 2018.

Kalchbrenner, N., van den Oord, A., Simonyan, K., Danihelka, I., Vinyals, O., Graves, A., and Kavukcuoglu, K. Video pixel networks. In *International Conference on Machine Learning*, 2017.

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.

Kolesnikov, A., Zhai, X., and Beyer, L. Revisiting Self-supervised Visual Representation Learning. In *Computer Vision and Pattern Recognition*, 2019.

Kurach, K., Lucic, M., Zhai, X., Michalski, M., and Gelly, S. The GAN Landscape: Losses, architectures, regularization, and normalization. *arXiv preprint arXiv:1807.04720*, 2018.

Lee, H.-Y., Huang, J.-B., Singh, M., and Yang, M.-H. Unsupervised representation learning by sorting sequences. In *International Conference on Computer Vision*, 2017.

Li, C., Xu, T., Zhu, J., and Zhang, B. Triple generative adversarial nets. In *Advances in Neural Information Processing Systems*. 2017.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. Are GANs Created Equal? A Large-scale Study. In *Advances in Neural Information Processing Systems*, 2018.

Menick, J. and Kalchbrenner, N. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. In *International Conference on Learning Representations*, 2019.

Miyato, T. and Koyama, M. cgans with projection discriminator. In *International Conference on Learning Representations*, 2018.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *International Conference on Learning Representations*, 2018.

Mundhenk, T. N., Ho, D., and Chen, B. Y. Improvements to context based self-supervised learning. In *Computer Vision and Pattern Recognition*, 2018.

Noroozi, M. and Favaro, P. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, 2016.

Odena, A. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

Odena, A., Olah, C., and Shlens, J. Conditional Image Synthesis with Auxiliary Classifier GANs. In *International Conference on Machine Learning*, 2017.

Pinto, L. and Gupta, A. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *IEEE International Conference on Robotics and Automation*, 2016.

Sajjadi, M. S., Bachem, O., Lucic, M., Bousquet, O., and Gelly, S. Assessing generative models via precision and recall. In *Advances in Neural Information Processing Systems*, 2018.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, 2016.

Sculley, D. Web-scale k-means clustering. In *International Conference on World Wide Web*. ACM, 2010.

Springenberg, J. T. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *International Conference on Learning Representations*, 2016.

Sricharan, K., Bala, R., Shreve, M., Ding, H., Saketh, K., and Sun, J. Semi-supervised conditional GANs. *arXiv preprint arXiv:1708.05789*, 2017.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. 2016.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *British Machine Vision Conference*, 2016.

Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. Self-Attention Generative Adversarial Networks. *arXiv preprint arXiv:1805.08318*, 2018.

A. Additional samples and interpolations

Figure 9. Samples obtained from S^{3}GAN (20% labels, 128×128) when interpolating in the latent space (left to right).

Figure 10. Samples obtained from S^{3}GAN (20% labels, 128×128) when interpolating in the latent space (left to right).

Figure 11. Samples obtained from S^{3}GAN (20% labels, 128×128) when interpolating in the latent space (left to right).

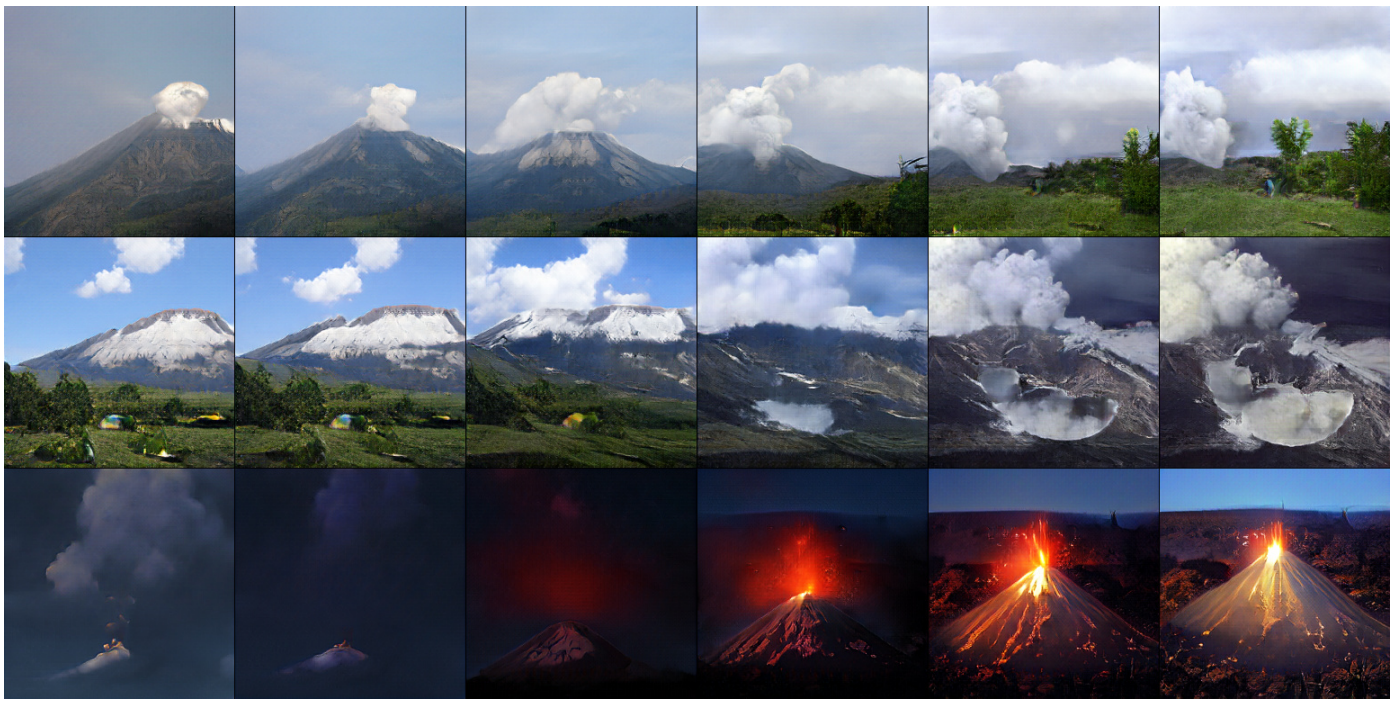
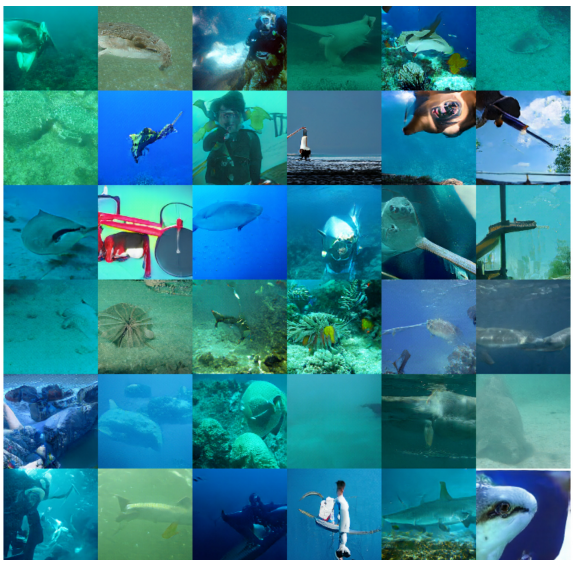


Figure 12. Samples obtained from S^{3}GAN (10% labels, 256×256) when interpolating in the latent space (left to right).

Figure 13. Samples obtained from S^3GAN (10% labels, 256×256) when interpolating in the latent space (left to right).

Real images.



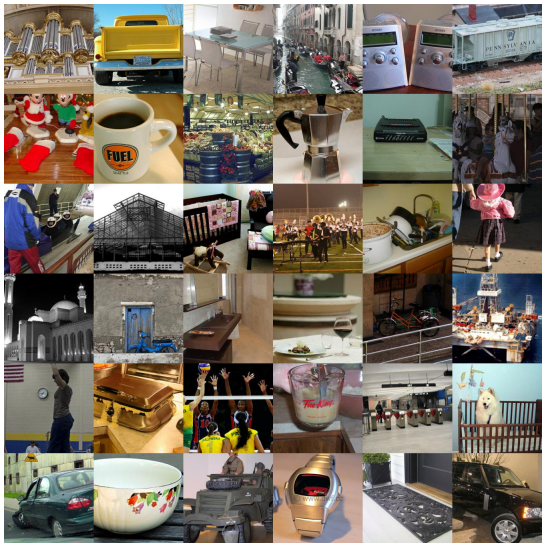
Generated images.

Figure 14. Real and generated images (128×128) for one of the 50 clusters produced by CLUSTERING. Both real and generated images show mostly underwater scenes.

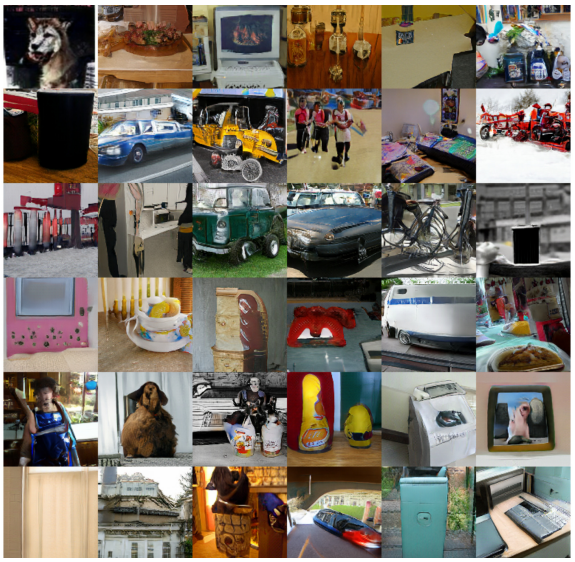
Real images.

Generated images.

Figure 15. Real and generated images (128 × 128) for one of the 50 clusters produced by CLUSTERING. Both real and generated images show mostly outdoor scenes featuring different animals.



Real images.



Generated images.

Figure 16. Real and generated images (128 × 128) for one of the 50 clusters produced by CLUSTERING. In contrast to the examples shown in Figures 14 and 15 the clusters show diverse indoor and outdoor scenes.

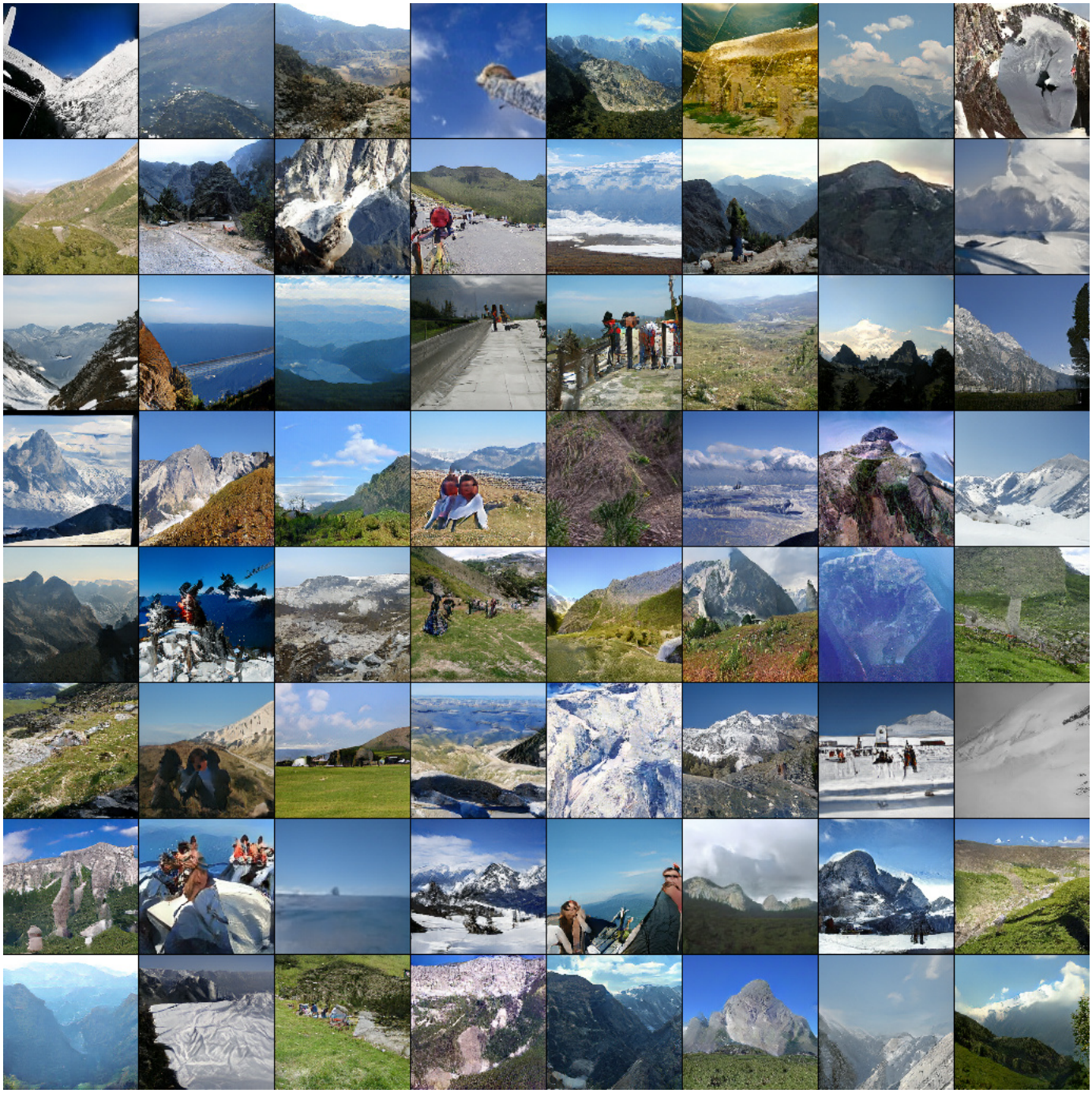


Figure 17. Samples generated by S^3GAN (20% labels, 128×128) for a single class. The model captures the great diversity within the class. Human faces and more dynamic scenes present challenges.

Figure 18. Generated samples by S^3GAN (20% labels, 128×128) for different classes. The model correctly learns the different classes and we did not observe class leakage.

Figure 19. Generated samples by S^{3}GAN (10% labels, 256×256) for a single class. The model captures the diversity within the class.

Figure 20. Generated samples by S^{3}GAN (10% labels, 256×256) for a single class. The model captures the diversity within the class.

B. Architectural details

The ResNet architecture implemented following Brock et al. (2019) is described in Tables 6 and 7. We use the abbreviations RS for resample and BN for batch normalization. In the resample column, we indicate downscale(D)/upscale(U)/none(-) setting. In Table 7, y stands for the labels and $\boldsymbol{h}$ is the output from the layer before (i.e., the pre-logit layer). Tables 8 and 9 show ResBlock details. The addition layer merges the shortcut path and the convolution path by adding them. h and w are the input height and width of the ResBlock, c_i and c_o are the input channels and output channels for a ResBlock. For the last ResBlock in the discriminator without resampling, we simply drop the shortcut layer from ResBlock. We list all the trainable variables and their shape in Tables 10 and 11.

Table 6. ResNet generator architecture. "ch" represents the channel width multiplier and is set to 96.

LAYER	RS	OUTPUT
$\boldsymbol{z} \sim \mathcal{N}(0, 1)$	-	120
Dense	-	$4 \times 4 \times 16 \cdot ch$
ResBlock	U	$8 \times 8 \times 16 \cdot ch$
ResBlock	U	$16 \times 16 \times 8 \cdot ch$
ResBlock	U	$32 \times 32 \times 4 \cdot ch$
ResBlock	U	$64 \times 64 \times 2 \cdot ch$
Non-local block	-	$64 \times 64 \times 2 \cdot ch$
ResBlock	U	$128 \times 128 \times 1 \cdot ch$
BN, ReLU	-	$128 \times 128 \times 3$
Conv $[3, 3, 1]$	-	$128 \times 128 \times 3$
Tanh	-	$128 \times 128 \times 3$

Table 7. ResNet discriminator architecture. "ch" represents the channel width multiplier and is set to 96.

LAYER	RS	OUTPUT
Input image	-	$128 \times 128 \times 3$
ResBlock	D	$64 \times 64 \times 1 \cdot ch$
Non-local block	-	$64 \times 64 \times 1 \cdot ch$
ResBlock	D	$32 \times 32 \times 2 \cdot ch$
ResBlock	D	$16 \times 16 \times 4 \cdot ch$
ResBlock	D	$8 \times 8 \times 8 \cdot ch$
ResBlock	D	$4 \times 4 \times 16 \cdot ch$
ResBlock	-	$4 \times 4 \times 16 \cdot ch$
ReLU	-	$4 \times 4 \times 16 \cdot ch$
Global sum pooling	-	$1 \times 1 \times 16 \cdot ch$
Sum(embed(y)$\cdot\boldsymbol{h}$)+(dense$\rightarrow 1$)	-	1

Table 8. ResBlock discriminator.

LAYER	KERNEL	RS	OUTPUT
Shortcut	$[1, 1, 1]$	D	$h/2 \times w/2 \times c_o$
BN, ReLU	-	-	$h \times w \times c_i$
Conv	$[3, 3, 1]$	-	$h \times w \times c_o$
BN, ReLU	-	-	$h \times w \times c_o$
Conv	$[3, 3, 1]$	D	$h/2 \times w/2 \times c_o$
Addition	-	-	$h/2 \times w/2 \times c_o$

Table 9. ResBlock generator.

LAYER	KERNEL	RS	OUTPUT
Shortcut	$[1, 1, 1]$	U	$2h \times 2w \times c_o$
BN, ReLU	-	-	$h \times w \times c_i$
Conv	$[3, 3, 1]$	U	$2h \times 2w \times c_o$
BN, ReLU	-	-	$2h \times 2w \times c_o$
Conv	$[3, 3, 1]$	-	$2h \times 2w \times c_o$
Addition	-	-	$2h \times 2w \times c_o$

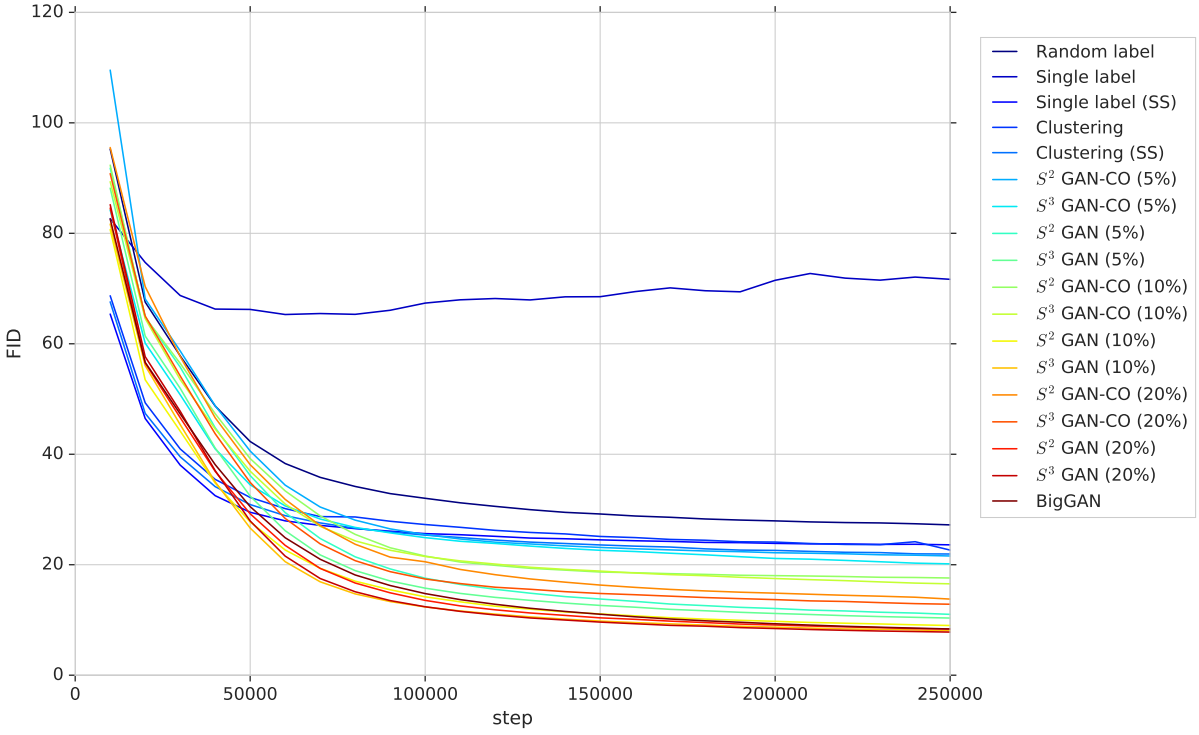
NAME	SHAPE	SIZE
discriminator/B1/same_conv1/kernel:0	(3, 3, 3, 96)	2,592
discriminator/B1/same_conv1/bias:0	(96,)	96
discriminator/B1/down_conv2/kernel:0	(3, 3, 96, 96)	82,944
discriminator/B1/down_conv2/bias:0	(96,)	96
discriminator/B1/down_conv_shortcut/kernel:0	(1, 1, 3, 96)	288
discriminator/B1/down_conv_shortcut/bias:0	(96,)	96
discriminator/non_local_block/conv2d_theta/kernel:0	(1, 1, 96, 12)	1,152
discriminator/non_local_block/conv2d_phi/kernel:0	(1, 1, 96, 12)	1,152
discriminator/non_local_block/conv2d_g/kernel:0	(1, 1, 96, 48)	4,608
discriminator/non_local_block/sigma:0	()	1
discriminator/non_local_block/conv2d_attn_g/kernel:0	(1, 1, 48, 96)	4,608
discriminator/B2/same_conv1/kernel:0	(3, 3, 96, 192)	165,888
discriminator/B2/same_conv1/bias:0	(192,)	192
discriminator/B2/down_conv2/kernel:0	(3, 3, 192, 192)	331,776
discriminator/B2/down_conv2/bias:0	(192,)	192
discriminator/B2/down_conv_shortcut/kernel:0	(1, 1, 96, 192)	18,432
discriminator/B2/down_conv_shortcut/bias:0	(192,)	192
discriminator/B3/same_conv1/kernel:0	(3, 3, 192, 384)	663,552
discriminator/B3/same_conv1/bias:0	(384,)	384
discriminator/B3/down_conv2/kernel:0	(3, 3, 384, 384)	1,327,104
discriminator/B3/down_conv2/bias:0	(384,)	384
discriminator/B3/down_conv_shortcut/kernel:0	(1, 1, 192, 384)	73,728
discriminator/B3/down_conv_shortcut/bias:0	(384,)	384
discriminator/B4/same_conv1/kernel:0	(3, 3, 384, 768)	2,654,208
discriminator/B4/same_conv1/bias:0	(768,)	768
discriminator/B4/down_conv2/kernel:0	(3, 3, 768, 768)	5,308,416
discriminator/B4/down_conv2/bias:0	(768,)	768
discriminator/B4/down_conv_shortcut/kernel:0	(1, 1, 384, 768)	294,912
discriminator/B4/down_conv_shortcut/bias:0	(768,)	768
discriminator/B5/same_conv1/kernel:0	(3, 3, 768, 1536)	10,616,832
discriminator/B5/same_conv1/bias:0	(1536,)	1,536
discriminator/B5/down_conv2/kernel:0	(3, 3, 1536, 1536)	21,233,664
discriminator/B5/down_conv2/bias:0	(1536,)	1,536
discriminator/B5/down_conv_shortcut/kernel:0	(1, 1, 768, 1536)	1,179,648
discriminator/B5/down_conv_shortcut/bias:0	(1536,)	1,536
discriminator/B6/same_conv1/kernel:0	(3, 3, 1536, 1536)	21,233,664
discriminator/B6/same_conv1/bias:0	(1536,)	1,536
discriminator/B6/same_conv2/kernel:0	(3, 3, 1536, 1536)	21,233,664
discriminator/B6/same_conv2/bias:0	(1536,)	1,536
discriminator/final_fc/kernel:0	(1536, 1)	1,536
discriminator/final_fc/bias:0	(1,)	1
discriminator_projection/kernel:0	(1000, 1536)	1,536,000

Table 10. Tensor-level description of the discriminator containing a total of 87,982,370 parameters.

NAME	SHAPE	SIZE
generator/embed_y/kernel:0	(1000, 128)	128,000
generator/fc_noise/kernel:0	(20, 24576)	491,520
generator/fc_noise/bias:0	(24576,)	24,576
generator/B1/bn1/condition/gamma/kernel:0	(148, 1536)	227,328
generator/B1/bn1/condition/beta/kernel:0	(148, 1536)	227,328
generator/B1/up_conv1/kernel:0	(3, 3, 1536, 1536)	21,233,664
generator/B1/up_conv1/bias:0	(1536,)	1,536
generator/B1/bn2/condition/gamma/kernel:0	(148, 1536)	227,328
generator/B1/bn2/condition/beta/kernel:0	(148, 1536)	227,328
generator/B1/same_conv2/kernel:0	(3, 3, 1536, 1536)	21,233,664
generator/B1/same_conv2/bias:0	(1536,)	1,536
generator/B1/up_conv_shortcut/kernel:0	(1, 1, 1536, 1536)	2,359,296
generator/B1/up_conv_shortcut/bias:0	(1536,)	1,536
generator/B2/bn1/condition/gamma/kernel:0	(148, 1536)	227,328
generator/B2/bn1/condition/beta/kernel:0	(148, 1536)	227,328
generator/B2/up_conv1/kernel:0	(3, 3, 1536, 768)	10,616,832
generator/B2/up_conv1/bias:0	(768,)	768
generator/B2/bn2/condition/gamma/kernel:0	(148, 768)	113,664
generator/B2/bn2/condition/beta/kernel:0	(148, 768)	113,664
generator/B2/same_conv2/kernel:0	(3, 3, 768, 768)	5,308,416
generator/B2/same_conv2/bias:0	(768,)	768
generator/B2/up_conv_shortcut/kernel:0	(1, 1, 1536, 768)	1,179,648
generator/B2/up_conv_shortcut/bias:0	(768,)	768
generator/B3/bn1/condition/gamma/kernel:0	(148, 768)	113,664
generator/B3/bn1/condition/beta/kernel:0	(148, 768)	113,664
generator/B3/up_conv1/kernel:0	(3, 3, 768, 384)	2,654,208
generator/B3/up_conv1/bias:0	(384,)	384
generator/B3/bn2/condition/gamma/kernel:0	(148, 384)	56,832
generator/B3/bn2/condition/beta/kernel:0	(148, 384)	56,832
generator/B3/same_conv2/kernel:0	(3, 3, 384, 384)	1,327,104
generator/B3/same_conv2/bias:0	(384,)	384
generator/B3/up_conv_shortcut/kernel:0	(1, 1, 768, 384)	294,912
generator/B3/up_conv_shortcut/bias:0	(384,)	384
generator/B4/bn1/condition/gamma/kernel:0	(148, 384)	56,832
generator/B4/bn1/condition/beta/kernel:0	(148, 384)	56,832
generator/B4/up_conv1/kernel:0	(3, 3, 384, 192)	663,552
generator/B4/up_conv1/bias:0	(192,)	192
generator/B4/bn2/condition/gamma/kernel:0	(148, 192)	28,416
generator/B4/bn2/condition/beta/kernel:0	(148, 192)	28,416
generator/B4/same_conv2/kernel:0	(3, 3, 192, 192)	331,776
generator/B4/same_conv2/bias:0	(192,)	192
generator/B4/up_conv_shortcut/kernel:0	(1, 1, 384, 192)	73,728
generator/B4/up_conv_shortcut/bias:0	(192,)	192
generator/non_local_block/conv2d_theta/kernel:0	(1, 1, 192, 24)	4,608
generator/non_local_block/conv2d_phi/kernel:0	(1, 1, 192, 24)	4,608
generator/non_local_block/conv2d_g/kernel:0	(1, 1, 192, 96)	18,432
generator/non_local_block/sigma:0	()	1
generator/non_local_block/conv2d_attn_g/kernel:0	(1, 1, 96, 192)	18,432
generator/B5/bn1/condition/gamma/kernel:0	(148, 192)	28,416
generator/B5/bn1/condition/beta/kernel:0	(148, 192)	28,416
generator/B5/up_conv1/kernel:0	(3, 3, 192, 96)	165,888
generator/B5/up_conv1/bias:0	(96,)	96
generator/B5/bn2/condition/gamma/kernel:0	(148, 96)	14,208
generator/B5/bn2/condition/beta/kernel:0	(148, 96)	14,208
generator/B5/same_conv2/kernel:0	(3, 3, 96, 96)	82,944
generator/B5/same_conv2/bias:0	(96,)	96
generator/B5/up_conv_shortcut/kernel:0	(1, 1, 192, 96)	18,432
generator/B5/up_conv_shortcut/bias:0	(96,)	96
generator/final_norm/gamma:0	(96,)	96
generator/final_norm/beta:0	(96,)	96
generator/final_conv/kernel:0	(3, 3, 96, 3)	2,592
generator/final_conv/bias:0	(3,)	3

Table 11. Tensor-level description of the generator containing a total of 70,433,988 parameters.

C. FID and IS training curves



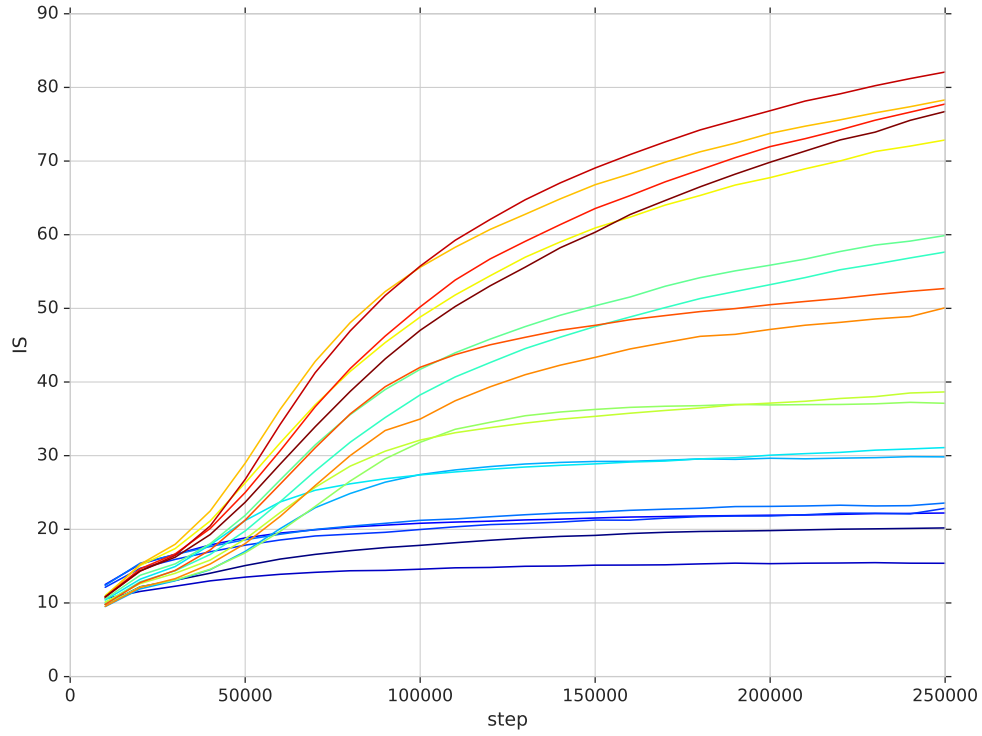


Figure 21. Mean FID and IS (3 runs) on ImageNet (128×128) for the models considered in this paper, as a function of the number of generator steps. All models train stably, except SINGLE LABEL (where one run collapsed).

D. FID and IS: Mean and standard deviations

Table 12. Pre-trained vs co-training approaches, and the effect of self-supervision during GAN training. While co-training approaches outperform fully unsupervised approaches, they are clearly outperformed by the pre-trained approaches. Self-supervision during GAN training helps in all cases.

	FID			IS		
	5%	10%	20%	5%	10%	20%
S^2GAN	11.0±0.31	9.0±0.30	8.4±0.02	57.6±0.86	72.9±1.41	77.7±1.24
S^2GAN-CO	21.6±0.64	17.6±0.27	13.8±0.48	29.8±0.21	37.1±0.54	50.1±1.45
S^3GAN	10.3±0.16	8.1±0.14	7.8±0.20	59.9±0.74	78.3±1.08	82.1±1.89
S^3GAN-CO	20.2±0.14	16.5±0.12	12.8±0.51	31.1±0.18	38.7±0.36	52.7±1.08

Table 13. Training with hard (predicted) labels leads to better models than training with soft (predicted) labels.

	FID			IS		
	5%	10%	20%	5%	10%	20%
S^2GAN	11.0±0.31	9.0±0.30	8.4±0.02	57.6±0.86	72.9±1.41	77.7±1.24
S^2GAN SOFT	15.6±0.58	13.3±1.71	11.3±1.42	40.1±0.97	49.3±4.67	58.5±5.84

Table 14. Mean FID and IS for the unsupervised approaches.

	FID	IS
CLUSTERING	22.7±0.80	22.8±0.42
CLUSTERING(SS)	21.9±0.08	23.6±0.19
RANDOM LABEL	27.2±1.46	20.2±0.33
SINGLE LABEL	71.7±66.32	15.4±7.57
SINGLE LABEL(SS)	23.6±0.14	22.2±0.10