Machines that Learn with Limited or No Supervision: A Survey on Deep Learning Based Techniques

Soumava Roy, Samitha Herath, Richard Nock, and Fatih Porikli

Abstract

The ability to learn with limited supervision has been the foundation for many machine learning wonders. Here, we draw connections to deep learning based solutions, which have revolutionized research in visual understanding, from their conventional hand-crafted counterparts. Our discussion aims to provide answers to two fundamental questions, namely, what can we learn with limited supervision and how can we learn it. We revisit representation learning, distribution learning, and knowledge transfer techniques, and lay out a taxonomy of emerging unsupervised methods while highlighting some of their applications.

I. Introduction

In 1997, DeepBlue defeated Garry Kasparov, the reigning world champion, in a chess match. Nineteen years later in 2016, AlphaGo won against Lee Sedol, a 9 dan professional player who ranked the second in international titles, in a more challenging game of Go. Both matches attracted massive news coverage and are considered milestones in developing intelligent machines. However, there is a fundamental difference between these two machines. Loosely speaking, DeepBlue was an advanced search engine equipped with specialized logics and rules to assess the positions it faced. On the contrary, AlphaGo made use of the games previously played by humans (and also machines) to learn the principles of the game by itself, avoiding the design of the rules by hand. This big leap in the conception of intelligent machines is achieved by advances in what is known as deep learning.

Deep learning refers to a class of algorithms that can identify a complex and nonlinear function by a hierarchical composition of simple operations, which are determined by a large number of training examples. Majority of the deep learning techniques are tailored towards what computer scientists consider as supervised learning. That is, the training examples presented to a deep neural network form pairs where for a given input $\boldsymbol{x}$ the network is expected to produce the response $f(\boldsymbol{x})$. Though deep neural networks trained under the supervised scheme have triumphed in numerous tasks (*e.g.*, object detection, image tagging) [28], learning with limited or no supervision is naturally more appealing and valuable.

We investigate what can be learned with limited or no supervision, and how such a goal can be achieved. In a nutshell, representation learning, distribution learning, and knowledge transfer cover the range of our quest and our discussion focuses on deep learning methods that can cater these three tasks. We first examine Auto-Encoders (AEs) for learning representations. We discuss class of techniques for Generative Adversarial Networks (GANs) and investigate their capacity to learn complex distributions. We articulate the concepts behind their probabilistic extensions, e.g. Variational Auto-Encoders (VAEs). We then explore knowledge transfer under limited supervision settings and review Zero-Shot Learning and Teacher-Student Networks in this context.

II. Prologue

Throughout the paper, we use bold uppercase letters to denote matrices (*e.g.*, $\boldsymbol{X}$) and bold lowercase to column vectors (*e.g.*, $\boldsymbol{x}$). In the text, $\mathbf{I}_n$ is the $n \times n$ identity matrix. We use tuple (x, y) to denote a data instance: $\boldsymbol{x} \in \mathbb{R}^n$ with its label $y \in \mathbb{R}$.

A network is represented as a collection of nonlinear functions $\{f_1^{\theta_1}, ...f_k^{\theta_k}..., f_n^{\theta_n}\}$ where $f_k^{\theta_k} : \mathbb{R}^{d_k} \rightarrow \mathbb{R}^{d_{k+1}}$ represents the function of the k^{th} layer. Here, θ_k are the parameters of the k^{th} function and its input's dimension is d_k. The tuple $\Theta = (\theta_1, ..., \theta_k..., \theta_n)$ collectively represent the

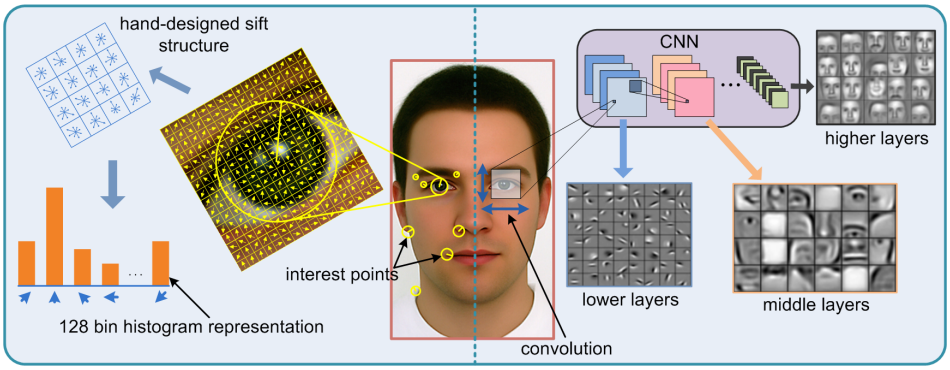


Fig. 1: A comparison between the state-of-the-art SIFT descriptors and deep learning based representations. On the left, we show how SIFT descriptors are extracted. Such descriptors are entirely handcrafted. In contrast, deep learning representations are learned in a data-driven fashion. As shown, convolutional filters of the higher level layers represent more spatially extended and semantically meaningful information of the object class.

parameters of the n-layer network. For a feed-forward network, the parameters Θ^* are learned by minimizing

$$\Theta^* = \arg\min_{\Theta} \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, f_n^{\theta_n} \circ f_{n-1}^{\theta_{n-1}} \circ \ldots \circ f_2^{\theta_2} \circ \ f_1^{\theta_1}(\boldsymbol{x}^{(i)})\big). \tag{1}$$

Above, $\mathcal{L}$ is the loss function measuring the discrepancy between the output of the network and the expected output, and $y^{(i)} \in \mathbb{R}$ for a training sample $x^{(i)} \in \mathbb{R}^{d_1}$. To apply gradient descent to the k^{th} layer parameters, the derivative $\nabla_{\theta_k}\mathcal{L}$ should be known, which is obtained by the chain-rule. Back-propagation of error is a memory efficient numerical algorithm where stochastic gradient descent algorithms such as momentum update are used for parameter updates.

III. Representation Learning

Performance of a learning algorithm is highly dependent on the conformity of its input data, in other words, a learned model works at its expected level of accuracy if only it is supplied with "good inputs". Preferably, a good representation should remain invariant to the variations in the input (*e.g.*, a good face model is robust to rotations). Also, a good representation should be discriminative (*e.g.*, a good face model is one that can discriminate different individuals with ease). Traditional learning algorithms customarily assume that another oracle is responsible for reconstructing the "good inputs". As such, great effort has been devoted to develop handcrafted schemes that can extract good inputs from visual data. A prime example is the popular Scale Invariant Feature Transform (SIFT) [35] for representing image regions. Conceptually, the SIFT assumes that edges in the image are imperative (and scale invariant) clues thus it encodes them through an involved procedure.

Handcrafted representations come with various drawbacks, their non-adaptive nature being a critical one. For example, the SIFT descriptor is completely blind to its inputs, regardless of they are medical images or landscape scenes. This inherently degrades the quality and accuracy of the learned model. Besides, thinking again about the SIFT descriptor, why should we believe that working with edges is what a machine needs for understanding images? Perhaps we should let the machine to decide what kind of information it requires for this task. Even more, developing automatic feature learning strategies has analogies to the processing that happens inside the human brain. See Fig. 1 for a an illustrative comparison of handcrafted and data-driven representations.

The goal of learning of "good inputs" has led to the emergence of data-driven representations that are nowadays mostly obtained by deep learning where its features are learned automatically from the

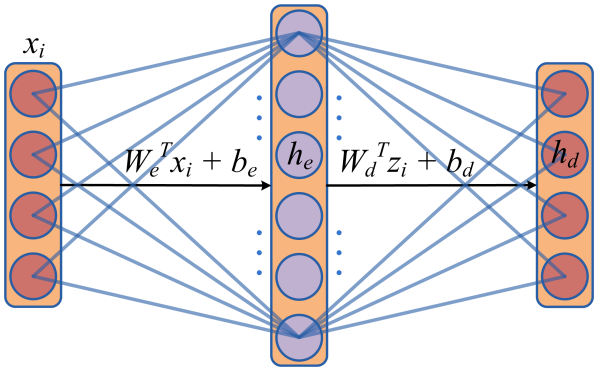


Fig. 2: An autoencoder with a single hidden layer. We represent the intermediate encoder output with $\boldsymbol{z}_i$. The nonlinear activations h_e and h_d are applied at each element of the transformed vectors.

data in a bottom-up fashion. For unsupervised representation learning, autoencoders have become the de facto architectures. They are successfully applied to obtain lower-dimensional embeddings and demonstrated higher accuracy results in numerous classification and regression tasks across various fields ranging from image understanding, speech recognition, natural language processing, sentiment analysis, DNA mutation analysis in genetics and reconstructing brain circuits. Furthermore, they are often employed to learn salient features for computer vision tasks such as image segmentation.

A. Auto-Encoders

Given a set of examples $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m\},\ \boldsymbol{x}_i \in \mathbb{R}^n$, the problem of representation learning can be cast as determining the function $f_\theta \in \mathcal{F}$ such that $f_\theta(\boldsymbol{x})$ preserves some properties over $\boldsymbol{X}$.

An autoencoder [5], [49] is a nonlinear network that realizes the notion of representation learning described above. It comprises of two parts, namely, an *encoder* and a *decoder*. The encoder is parameterized by a weight matrix $\boldsymbol{W}_e \in \mathbb{R}^{n\times p}$, a bias term $\boldsymbol{b}_e \in \mathbb{R}^p$ and a nonlinear (*e.g.* Sigmoid) function $h_e : \mathbb{R} \to \mathbb{R}$. It realizes the mapping $f_e : \mathbb{R}^n \to \mathbb{R}^p$ as $f_e(\boldsymbol{x}) = h_e(\boldsymbol{W}_e^T\boldsymbol{x} + \boldsymbol{b}_e)$. Coupled with an encoder, a secondary nonlinear mapping $f_d : \mathbb{R}^p \to \mathbb{R}^n$ as $f_d(\boldsymbol{x}) = h_d(\boldsymbol{W}_d^T\boldsymbol{x} + \boldsymbol{b}_d)$ is formed by the decoder. As such, a single layer autoencoder is parameterized by the tuple $\theta = (\boldsymbol{W}_e, \boldsymbol{W}_d, \boldsymbol{b}_e, \boldsymbol{b}_d, h_e, h_d)$. A simple way to learn the parameters of an autoencoder is to minimize the reconstruction loss $\sum_i \|\boldsymbol{x}_i - f_d\big(f_e(\boldsymbol{x}_i)\big)\|^2 = \sum_i \|\boldsymbol{x}_i - h_d\big(\boldsymbol{W}_d^T f_e(\boldsymbol{x}_i) + \boldsymbol{b}_d\big)\|^2$. Evidently, by stacking autoencoders together, one can build a richer set of nonlinear mappings (see Fig. 2 for a conceptual diagram).

It is helpful to draw some connections between autoencoders and the Principal Component Analysis (PCA) algorithm for representation learning. The minimization objective for the PCA algorithm is $\sum_i \|\boldsymbol{x}_i - \boldsymbol{W}_{pca}\boldsymbol{W}_{pca}^T\boldsymbol{x}_i\|^2$ with an extra orthogonality constraint in the form $\boldsymbol{W}_{pca}^T\boldsymbol{W}_{pca} = \mathbf{I}_p$ with $p < n$ [11]. Assuming $\boldsymbol{b}_e = \mathbf{0}_n$, $\boldsymbol{b}_d = \mathbf{0}_p$, $h_e(\boldsymbol{x}) = \boldsymbol{x}$, $h_d(\boldsymbol{x}) = \boldsymbol{x}$, the loss of an autoencoder becomes $\sum_i \|\boldsymbol{x}_i - \boldsymbol{W}_d\boldsymbol{W}_e^T\boldsymbol{x}_i\|^2$. This reveals that the PCA algorithm is indeed a specific realization of an autoencoder. To its advantage, the flexibility of an autoencoder (two mappings $\boldsymbol{W}_e$ and $\boldsymbol{W}_d$ instead of one in the PCA) can lead to more competent solutions.

Below, we discuss widely used structures of autoencoders.

Under-Complete Auto-Encoders (uC-AEs): Frequently used to derive salient features, a uC-AE is the network obtained by choosing $p < n$. As shown above, for linear units, an under-complete autoencoder generalizes the PCA algorithm. With nonlinear units, it is expected to learn a nonlinear yet more powerful generalization of PCA. Unfortunately, uC-AEs with a high capacity encoder and decoder may only act as a look-up table thereby simply copying the input to the output instead of learning salient features.

Denoising Auto-Encoders (DAEs): Here, a distorted version of the dataset is given as the input to the encoder, and the decoder is encouraged to produce the original and clean version of the dataset.

Distortion is achieved by adding noise to the input (or with a transformation such as changing the orientation of the image, flipping for its other variants). Ideally, DAE is expected to undo the effect of the noise corruption by learning a robust representation that can be used to clean the noisy input. To this end, DAE often relies on the concept of higher-level representations by stacking autoencoders [45]. Training of the entire network is carried by layer-wise greedy pretraining until the required dimensionality reduction is achieved [3]. Greedy layer-wise pretraining where the input to a to-be-trained-layer is the output of concatenation of pretrained representation layers may lead to discovery of discriminative features at every layer that are important for the reconstruction of the input. DAEs can also be understood with the concepts of manifold learning where the corrupted examples sit farther away from the manifold in comparison to the uncorrupted and clean samples. The DAE, by seeing both noisy and clean version of the data, attempts to learn a mapping that goes from low probability noisy points to high probability uncorrupted points; thereby learning a nonlinear manifold, which captures the variations in the data.

Sparse Auto-Encoders (SAEs): Any possible configuration of an autoencoder can be trained given the capacity of the encoder and the decoder is high enough to model the complexity of the data and the distribution of the latent variables. However, with an increase in capacity (*e.g.* the number of encoder and decoder pairs), the autoencoder simply acts as a look-up table. Therefore, a check in the model's capacity is done by introducing regularization in the layers [17]. This regularization encourages the autoencoder to learn distinct properties that are needed to reproduce the input at the output. With this, a regularized autoencoder can still be over-complete i.e. $p \geq n$, but learn useful features. One such regularization technique is to make the representation sparse. Sparse, over-complete representations increase the separability of the data and also provide simple yet coherent interpretation of the input data in terms of a small number of distinct parts that reveal the hidden structural properties of the data [40], reminiscent of the receptive fields of V1 neurons [27]. A sparse autoencoder is trained with an additional sparsity constraint on the hidden layer representations. This ensures the features to be unique to the dataset they are trained on. The lower level features can be combined to obtain more distinct higher level abstract features. A simple single layer sparse encoder can be represented as $L(x, f_d(f_e(x))) + \Omega(z = f_e(x))$ where Ω is the sparsity function. It is observed that training of such sparse autoencoders on natural images produces Gabor-like filters across multiple orientations, frequencies, and locations [19].

Convolutional Auto-Encoders (CAEs): Autoencoders tend to ignore the spatial information in the given visual data that usually include local and repetitive patterns. Inspired by Convolutional Neural Networks (CNNs), CAEs [48] are introduced as hierarchical unsupervised feature extractors that can scale to large-size input data and share architecture-wise similarities to CNNs. Conceptually, a CAE discovers the discriminative features that are fundamental to the reconstruction of its input. The encoder is a convolutional network that learns a local representation of the input. The decoder is a deconvolutional network that permits the unsupervised reconstruction of the input from the learned representation of the encoder. Aside from filtering and rectification (nonlinear functions), the decoder benefits from an upsampling operation to undo the effect of pooling operations in the encoder. Training follows the AE scheme while maintaining the spatial structure of the input. A block diagram of a typical CAE network as in [36] is presented in Fig. 3.

B. Training Auto-Encoders

The common practice in training a deep network is to randomly initialize the parameters of the network followed by updating them using the backprop algorithm. The unique structure of autoencoders enables us to resort to better training procedures. In particular, parameters of a stacked autoencoders are learned by a greedy layer-wise strategy. In doing so, the parameters of the k-th layer of a stacked autoencoder is used to train the next layer, *i.e.* layer k+1. As an example, consider a stacked autoencoder in the form $\mathrm{E}\#1 \rightarrow \mathrm{E}\#2 \rightarrow \mathrm{D}\#2 \rightarrow \mathrm{D}\#1$.

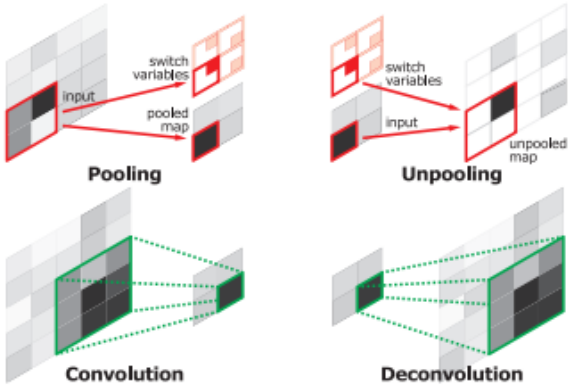


Fig. 3: Block diagram of a CAE as reported in [36]. The pooling-unpooling and convolution-deconvolution operations are shown in red and green, respectively. Unpooling is performed using switch variables (that record the spatial locations of maximum activations during pooling). Deconvolution is performed with a transposed convolution, which associates a single input to multiple outputs.

In the first go, greedy layer-wise pretraining of E#1 $\rightarrow$ D#1 is performed based on an error function. Once this training is over, E#1 provides the lowest level features that are needed to reconstruct the input at D#1. Now, greedy pretraining of the second layer E#1 $\rightarrow$ E#2 $\rightarrow$ D#2 is performed to reconstruct the output of E#1 at D#2. However, no changes are made in the learned features of E#1 during the training of the second layer. In the final step, all layers are stacked according to the full architecture E#1 $\rightarrow$ E#2 $\rightarrow$ D#2 $\rightarrow$ D#1 and the network is finetuned, which makes sure that the combination of layer-wise pretrained features can reconstruct the input with ease.

The aforementioned form of training autoencoders has been shown to improve the generalization ability of the network in comparison to the random initialization. This can be explained by noting that the layer-wise training initializes the parameters of the network in the vicinity of better local minima as the parameters are restricted to a relatively small range [3]. An alternative approach is a closed-form learning scheme proposed in [8]. Here, each layer is trained on a denoising objective with ordinary least squares. However, to have a closed form solution, the nonlinearity is omitted during the training time. Later, the learned layers are stacked with the nonlinearities for possible finetuning.

IV. Distribution Learning

To learn distributions, neural network based generative models have resurfaced due to their intriguing properties. Such generative models can be used towards a better interpretation of high-dimensional data and they enable machine learning algorithms to deal with multi-modal outputs [34]. In addition, they can provide predictions on missing inputs, which is useful in semi-supervised learning.

A. Variational Auto-Encoders

In order to overcome the restrictions of the Boltzmann Machines and their variants, neural networks are considered as a natural choice since the output of their hidden layers can be considered as latent variables. Thus neural networks are trained with as stochastic function approximators that are modeled to learn the joint or conditional probability distribution over the input $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$, and thereby act as generative models. The decoder is represented as f($\boldsymbol{z}$; θ) such that f: $\mathbb{R}^p \rightarrow \mathbb{R}^n$ that learns the probability distribution $p(\boldsymbol{x}|\boldsymbol{z})$. Along with the prior distribution $p(\boldsymbol{z})$ on the latent variables $\boldsymbol{z}$, the objective is to find a better configuration of the latent variables that

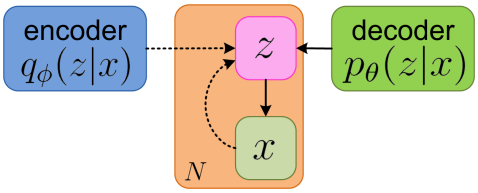


Fig. 4: Block diagram of a VAE. Above, ϕ and θ represent the parameters of the encoder and decoder networks, respectively. Solid lines indicate the generator network and the dashed-lines show the encoder network. The generator network learns $p_\theta(\boldsymbol{z}|\boldsymbol{x}) = p_\theta(\boldsymbol{x}|\boldsymbol{z})\ p_\theta(\boldsymbol{z})$, and the encoder approximates $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ using $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Both ϕ and θ are learned simultaneously according to (3)

.

produce samples representing the input. This is achieved by maximizing the posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ so that (2) is maximized as

$$p(\boldsymbol{x} \in X) = \int p(\boldsymbol{x}|\boldsymbol{z};\theta)p(\boldsymbol{z} \in Z)dz \tag{2}$$

where $p(\boldsymbol{x}|\boldsymbol{z};\theta)$ is used instead of $f(\boldsymbol{z};\theta)$, so as to make the dependence of $\boldsymbol{x}$ on $\boldsymbol{z}$ explicit under the law of total probability. The distribution of the decoder is Gaussian i.e. $p(\boldsymbol{x}|\boldsymbol{z};\theta) = N(\boldsymbol{x}|f(\boldsymbol{z};\theta), \sigma^2 I)$. However, the choice of the latent variables remains a difficult task as there exist a lot of interdependencies among them. These dependencies among the latent dimensions are usually avoided by assuming that $\boldsymbol{z}$ are sampled from a unit-covariance Gaussian distribution. The main reason behind this assumption is that any d-dimensional distribution can be generated by taking a set of normally distributed variables and a complicated function (the decoder represents this function). However, $p(\boldsymbol{z}|\boldsymbol{x})$ is difficult to calculate as its evaluation depends on all the configuration of latent variables, thereby requiring exponential calculation time. Therefore, this posterior distribution is approximated using a new probability distribution $q(\boldsymbol{z}|\boldsymbol{x})$ where the approximation step is achieved with an encoder f: $\mathbb{R}^n \rightarrow \mathbb{R}^p$ that is trained as a function approximator. This can be ensured by taking use of metrics that measure the differences between two probability distributions. One such metric is Kullback-Leibler divergence that measures the difference between $q(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z}|\boldsymbol{x})$ as follows

$$\underset{\boldsymbol{x}\sim D}{E}[log\, p(\boldsymbol{x}) - KL(q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})]\,] = \underset{\boldsymbol{x}\sim D}{E}[\underset{\boldsymbol{z}\sim Q}{E}[log\, p(\boldsymbol{x}|\boldsymbol{z})] - KL[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]] \tag{3}$$

where D represents the training dataset.

The above equation is the central theme of the variational autoencoder. On the left hand side of the equation, we want to maximize $logP(\boldsymbol{x})$ and minimize difference between the approximate and true posterior of latent values given the input. Assuming a high capacity of $Q(\boldsymbol{z}|\boldsymbol{x})$, then the difference between $Q(\boldsymbol{z}|\boldsymbol{x})$ and $P(\boldsymbol{z}|\boldsymbol{x})$ will be zero and we will end up optimizing $log(P(\boldsymbol{x}))$. We usually choose $Q(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{z}|\mu(\boldsymbol{x},\phi), \Sigma(\boldsymbol{x},\phi))$ such that μ and Σ are learned using the encoder network with parameters ϕ. The right hand of (3) is overly dependent on large number of samples, but this is mitigated by using stochastic gradient with one training sample at a time. However, sampling is non-continuous and has no gradient thereby back-propagating the error derivative through the sampling layer becomes a difficulty. This is overcome by the "re-parameterization" trick [12]. Thus given $\mu(\boldsymbol{x})$ and $\Sigma(\boldsymbol{x})$, we can sample first from $\epsilon \sim N(0,1)$ and compute z = $\mu(\boldsymbol{x}) + \Sigma^{\frac{1}{2}}(\boldsymbol{x}) * \epsilon$. This is similar to sampling from N($\mu(\boldsymbol{x}), \Sigma(\boldsymbol{x})$). With this re-parameterization trick, back propagation of the error gradient through the sampling layers is avoided and the overall architecture can be finetuned as SGD. Block diagram of a VAE is given in Fig. 4.

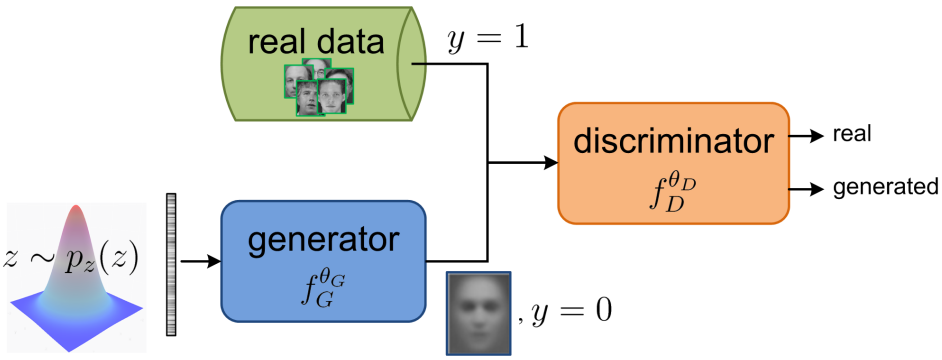


Fig. 5: GANs can be used to generate face images. The generator network accepts a random vector $\boldsymbol{z}$ and maps it to an image. The discriminator networks aims to differentiate between original and generated images, and the generator aims to fool the discriminator. Training process continues until the discriminator cannot differentiate between the original and generated images anymore. After training, one can generate face-like images from random vectors.

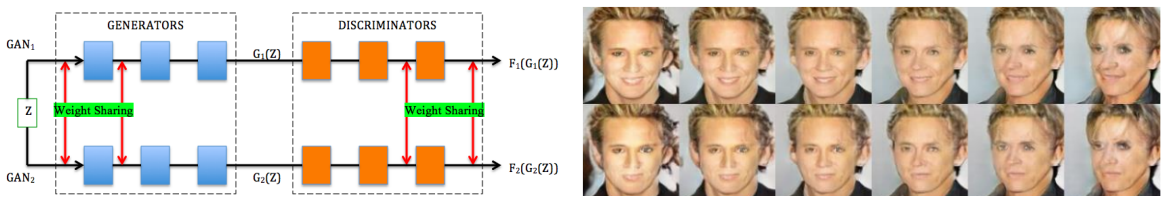
B. Generative Adversarial Networks (GANs)

General Adversarial Networks consist of two networks, namely, a generator network G and a discriminator network D. The generator and the discriminator networks are locked in a battle to outperform the other's functionality. That is, these two networks compete in a zero-sum game through iterations of min-max optimization where the generative network attempts to map random noise vectors to the data samples while the discriminator tries to distinguish whether its input is a genuine sample or one generated by G. We can draw an analogy with the generator network being a team of counterfeiters who produce fake currency and the discriminator network being the police who try to detect fake currency. This competition continues until the fake currency produced by the counterfeiters is indistinguishable from the genuine ones.

The discriminator is learned in a supervised manner and its objective is to differentiate between two classes (genuine, fake). The backpropagated gradient of the discriminator is used to finetune the weights of the generator, such that, the generator gets better at confusing the discriminator. The function optimized, $L(\theta_G, \theta_D)$, depends on the parameter of the density modeled by the generator, and of the classifier learned by the discriminator. Its abstract formulation can be summarized by general function $L(\theta_G, \theta_D) = E_{\boldsymbol{x}\sim D}[f_{\theta_D}(\boldsymbol{x})] - E_{\boldsymbol{z}\sim p(\boldsymbol{z})}[h_{\theta_D}(g_{\theta_G}(\boldsymbol{z}))]$, where D is the true distribution (estimated through a training sample), $p(\boldsymbol{z})$ is an uninformative distribution (uniform, Gaussian, etc.) and g_{θ_G} is the function that maps the support of $p(\boldsymbol{z})$ to that of the objects of interest (see Fig. 5).

For optimization of $L(\theta_G, \theta_D)$, one method directly considers distributions as they appear in L [18], [37]. Other approaches are more geometric, focusing for example on the statistics of the distributions [43], on the transportation distance between distributions [1] (eventually regularized [16]), or on a distance between their moments [31]. Intuitively, distribution and geometric approaches are similar, the latter ones making the optimization explicit on specific parameterizations of the distribution. It is interesting to notice that in many cases, $L(\theta_G, \theta_D)$ has the same form as the one described above [33]. In addition to the two-player game of GANs, a number related approaches have been devised for specific problems as well.

Coupled Generative Adversarial Networks (CoGANs): GANs are designed to learn the distribution of data pertaining to a single domain, such as indoor scenes or facial images. As such, GANs cannot aptly model the data distribution across multiple domains. This difficulty is addressed in CoGANs by associating multiple dedicated GANs to each and every domain [32]. Training GANs is done collectively with weight sharing of parameters that enforces the collection of GANs to



(a) (b)

Fig. 6: Architecture of a CoGAN. Shared Weights are marked as red. These sharing of weights makes it possible to generate images across two different domains as shown in 6b where smiling and neutral faces have been generated using a CoGAN.

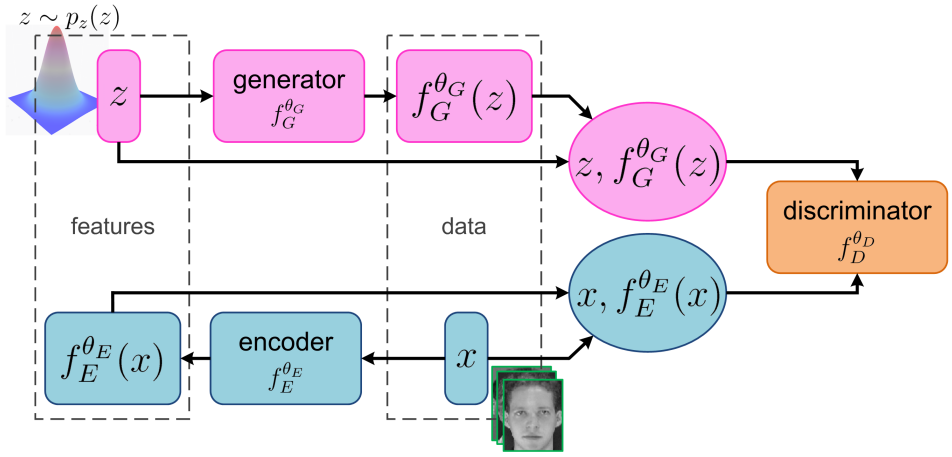


Fig. 7: Architecture of a BiGAN as reported in [13] consists of an additional encoder that maps from the input to the latent space.

converge to the joint distribution across multiple domains. A pictorial representation of a CoGAN with two GANs is shown in Fig. 6.

Bi-Directional GANs (BiGANs): In GANs, the generator network draws samples from a Gaussian distribution to reconstruct the output. A Bi-Directional GAN [13] targets the sampling process of a standard GAN and makes use of an encoder to learn a mapping from the input space to the latent space of the generative model (see Fig. 7 for an illustration). The hypothesize here is that the encoder of the BiGAN provides a useful feature representation for related semantic tasks.

Information Maximizing GANs: The primary aim of unsupervised learning is to attain *disentangled* representations, *i.e.*, a representation that captures the most salient features presented in the data. Though the unsupervised learning algorithms have successfully improved their ability to learn distinct representations, they are still oblivious to the downstream applications during their training time. Therefore, the burden lies on the unsupervised learning to learn the correct features that can be well generalized to be of use for different downstream applications. GANs doesn't impose any restrictions on the way the noisy latent variables are used. The noise will be used in a highly entangled manner thereby resulting in a generation of correlated latent variables. It would be helpful if certain latent variables are, in turn, expected to learn specific structures in the data. Thereby, in [9], the unstructured noise vector is decomposed into two components (1) $\vec{z}$, which represents the incompressible noise, and (2) $\vec{c}$, which represents the actual latent code that learns the salient representations of the data. The generator is now represented as $f_G(\vec{z}, \vec{c})$. The objective is achieved by maximizing the mutual information between $\vec{z}$ and $\vec{c}$ over the entire training

procedure, along with its original objective. Mutual information between two random variables A & B is defined as the reduction of uncertainty of A is given the value of B. By maximizing the mutual information, a constraint in placed that both $\vec{z}$ and $\vec{c}$ must be correlated to each other. This constraint leads to learning of better salient features compared to GANs, and it is equivalent to regularize function $L(\theta_G, \theta_D)$.

C. Applications of Autoencoders

VAEs were one of the first models that successfully generated handwritten digits and facial images, where low dimensional manifolds were learned to generate new sample images [24].

GANs and their variants have been successfully used for generating natural images [39], hallucinating faces [46], super-resolving faces [47], upsampling images [29], and generating photo-realistic images from rough sketches [50], among the others. The features learned by GANs have been shown to endow better discriminatory power when used for supervised tasks [39].

V. Knowledge Transfer

Techniques that benefit from available knowledge to work around a difficult learning problem (*e.g.*, learning tasks with insufficient data for training) are commonly referred to as knowledge transfer methods. In this part, we first discuss the problem of Zero- and One-Shot Learning (ZSL/OSL) from deep learning viewpoint. As the names imply, the number of training samples per class is zero for ZSL and merely one for OSL.

Finally, we examine Teacher-Student Networks (TSNs) [22] that distil the knowledge of a large model into a smaller one.

A. Zero- and One-Shot Learning

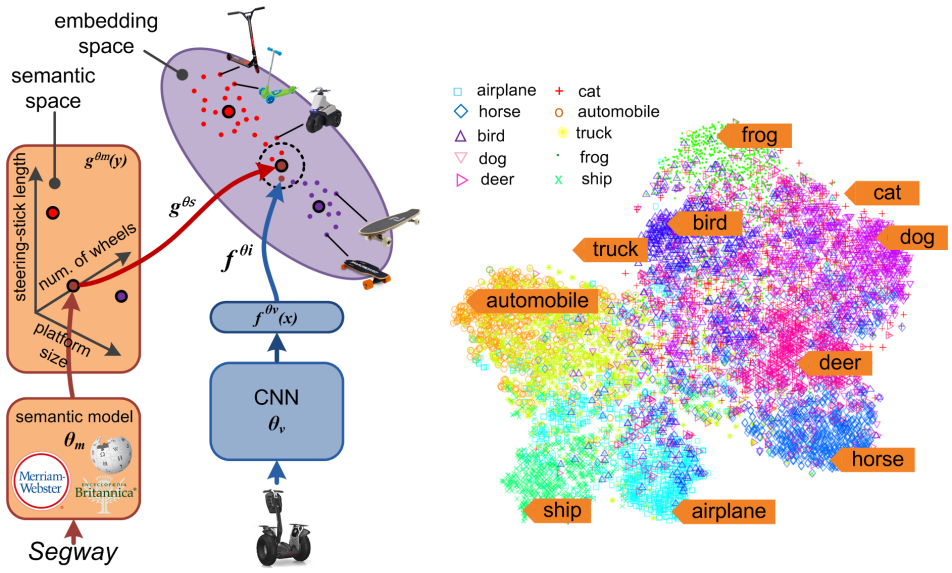
ZSL and OSL are classification problems branched from both open-set classification and transfer learning. The objective behind OSL is classification by only seeing one training example of a class. In ZSL, the model classifies instances of an unseen category through its semantic description.

The intuition behind ZSL and OSL is to employ knowledge from previously seen classes as supporting knowledge towards achieving the goal. Let us elaborate through an example. As human beings, it is easy for us to recognize a Segway even if we have seen it once before. This is because our mind can decompose it into a collection of known parts such as wheels a steering stick. Many variations of these components are already encoded in our memory. Hence, we have little difficulties compiling different models of Segways. Even in the case of ZSL, given a description of a Segway, we can guess how components should be connected. For this, we use our existing knowledge, perhaps on scooters, bicycles, skateboards, etc. This concept and a typical ZSL network are illustrated in Fig. 8.

ZSL and OSL learning algorithms use the knowledge from many seen classes, referred to as support set, to establish or quickly grasp concepts of unseen classes. In ZSL, the support knowledge is used to identify a common space called the embedding space where salient regions have meanings. In OSL, two dominant themes in the literature can be categorized as use of the supporting knowledge to generalize from one-shot instances and use of the support set to learn abstract concepts. An example of the former technique is to learn transformations to augment the one-shot classes [21]. Learning an embedding is the main idea behind constructing abstract concepts as done in [25].

B. Zero-Shot Learning

ZSL is a form of open set classification where examples of all classes are not available during training. This limitation prohibits us from training a classifier, a common approach in deep learning theory. An alternative and widely used technique in ZSL is to learn an embedding space such that classes are better separated, *i.e.* distances between pairs with different class labels are large while distances between pairs from the same class are small. The assumption is that the learned space is discriminative enough for the unseen classes.



(a) (b)

Fig. 8: (a) A typical ZSL solution with a convolutional neural network. The semantic space is a simple space with three attributes. Note that, for the unseen class (*e.g.*, Segway) no training images are provided. A test instance is classified based on the closest semantic vector in the embedding space. (b) The embedding space learned by [44]. The word mappings are indicated by arrows. For two unseen classes (cats and trucks), the images are embedded close to their similar objects (dogs and automobiles).

ZSL methods follow a similar idea to the Siamese networks, albeit with a subtle difference. In conventional Siamese networks, an embedding is sought to maximize the separability power on pairs of type-alike queries (*e.g.*, images). The embedding space in ZSL, not only should be discriminative for the type-alike queries (*e.g.*, images of birds should be easily recognized from airplanes) but it should also establish proper connections between samples and their associated semantics (*e.g.*, images of birds should be matched to the semantics of animals).

To put this into perspective, let the training images from seen classes $\mathcal{Y} = \{1, \cdots, N_s\}$ be $\mathcal{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k, \cdots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_k \in \mathbb{R}^n$. The subset $\mathcal{X}_y = \{\boldsymbol{x^{(y)}}_1, \cdots, \boldsymbol{x^{(y)}}_{n_y}\} \subset \mathcal{X}$ contains all the training instances belonging to a class, $y \in \mathcal{Y}$. Let $h : \mathcal{Y} \rightarrow \mathbb{R}^s$ be a mapping from $\mathcal{Y}$ to a semantic space $\mathbb{R}^s$. One way of constructing $\mathbb{R}^s$ is through a language model where classes like birds are given attributes (*e.g.*, beak, wings). Our goal is to establish relations between images and semantics through an embedding. This can be done by extending the conventional Siamese network to learn two mappings $f^{\theta_i} : \mathbb{R}^n \rightarrow \mathbb{R}^e$ and $g^{\theta_s} : \mathbb{R}^s \rightarrow \mathbb{R}^e$ by minimizing

$$L(\theta_i, \theta_s) = \sum_{y \in \mathcal{Y}} \sum_{j=1}^{n_y} \mathcal{L}\big(g^{\theta_s}(h(y)), f^{\theta_i}(\boldsymbol{x}_j^{(y)})\big). \tag{4}$$

The loss $\mathcal{L} : \mathbb{R}^e \times \mathbb{R}^e \rightarrow \mathbb{R}^+$ is small if a sample is correctly matched to its semantic and large otherwise. Euclidean distance is obviously a simple, yet effective choice here (see Fig. 8 for an illustration).

Most of the ZSL solutions rely on semantic spaces learned from text documents (*e.g.*, Wikipedia articles). This is partly because deep networks for natural language processing are available and effective [10]. Furthermore, text sources provide rich semantics. For example, an article about a bird species usually contains descriptions about the color, size, and shape of the beak of the bird. Such concepts, if encoded well within a network, can be used effectively to describe unseen birds.

Examples of such constructions include [44], [30] and [15] where in the latter the embedding space successfully works with 20K unseen classes. We note that Siamese networks have been successfully used for OSL as well [25].

C. One-Shot Learning

Studies in cognitive learning (*e.g.*, [7]) suggest that we use our past knowledge to learn new concepts from single encounters or even speeding up the learning process. It is also believed that humans use their prior knowledge to generalize the level of understanding attained from a single encounter. Learning a model for a class is one way to generalize the knowledge about it. Along with this line, [26] proposes to use the support data to learn a library of pen strokes for the task of handwritten character recognition. The library is then used to find the best stroke model explaining a given one shot instance. The learned model can be thought of as a generalized representation of the one-shot class and hence is used to estimate the likelihood of a query instance. Recent works, *e.g.* [21], suggest a different philosophy and use generative models to augment the one-shot instances for image classification.

D. Teacher-Student Networks

A Teacher-Student Networks (TSNs) [22] is a unique form of knowledge transfer, aiming to boil down a large deep network, called as the teacher network, into a smaller network with fewer parameters, called as the student network. Student network models trained using the TSN framework have been shown to achieve better performances than similar models trained using the conventional supervised frameworks [2], [22]. Interestingly, some studies even show that the student network model can outperform the larger teacher model [42].

The training objective in TSN learning is to mimic the responses of the teacher models by the student network. This is done by training the student to follow the outputs (*e.g.*, probability vectors or outputs of the last hidden layer [2]) of the teacher. Formally, for a given set of training examples, $\mathcal{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_k \in \mathbb{R}^n$, the student network model $f^{\theta_s} : \mathbb{R}^n \rightarrow \mathbb{R}^c$ follows the teacher's responses $f^{\theta_t} : \mathbb{R}^n \rightarrow \mathbb{R}^c$. The parameters of the student network model θ_s^* are learned by minimizing a loss in the form

$$\theta_s^* = \arg\min_{\theta_s} \sum_{k=1}^{N} \mathcal{L}\big(f^{\theta_t}(\boldsymbol{x}_k), f^{\theta_s}(\boldsymbol{x}_k)\big). \tag{5}$$

Here, $\mathcal{L} : \mathbb{R}^c \times \mathbb{R}^c \rightarrow \mathbb{R}^+$ measures how dissimilar is the response of the student network model $f^{\theta_s}(\cdot)$ to that of the teacher network model, *i.e.*, $f^{\theta_t}(\cdot)$. That is, $\mathcal{L}\big(f^{\theta_t}(\boldsymbol{x}_k), f^{\theta_s}(\boldsymbol{x}_k)\big)$ is large if $f^{\theta_s}(\boldsymbol{x}_k)$ and $f^{\theta_t}(\boldsymbol{x}_k)$ are dissimilar (and small otherwise).

The teacher in TSN learning can either be a single model [2], [42] or a collection of models [6], [22]. An ensemble is a collection of models trained with diversities (*e.g.*, different initializations, different training sets) [20]. A straightforward extension of the learning scheme in Eq. (5) to an ensemble is to replace the teacher network output, $f^{\theta_t}(\boldsymbol{x}_k)$ with the average response of the ensemble members [22].

To train a deeper student network, [42] introduces a hint-based learning concept. In hint-based learning, first few layers of the student network are pre-trained using the outputs of an intermediate layer of the teacher model (see Fig. 9 for a visual demonstration). This approach is motivated by a concept known as curriculum learning where training is gradually intensified [4]. Conventional curriculum learning algorithms use heuristics and problem specific measures to organize the training data according to its complexity. In the TSN framework, the classification confidence of the teachier network for a training sample is a reliable measure of complexity. This leads to a natural way of embedding curriculum learning into TSNs. The work in [42] benefits from this to gradually increase the impact of complex training samples to the end of the training epochs.

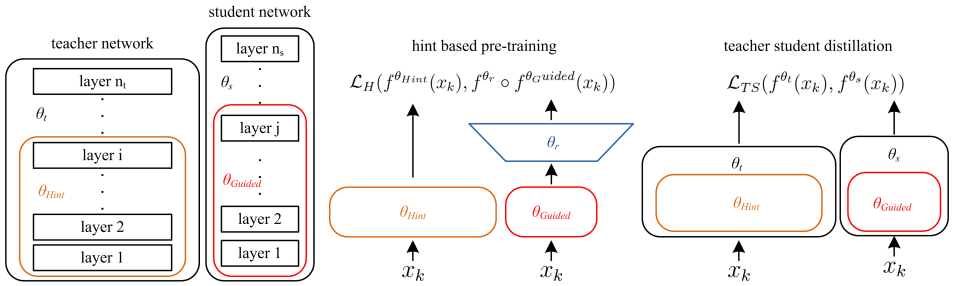


Fig. 9: Hint-based curriculum learning of [42]. As in a typical TSN, the teacher, θ_t is the trained model. The hint model use the first i layers of the teacher network, named the hint network to pretrain the first j layers of the student network. Once the guided network is trained, an end-to-end training based on the TSN scheme is performed. The regressor model θ_r is used to remove the structural dissimilarities (*e.g.*, mismatches in the dimensionality) between the hint and the guided layers.

E. Applications of Knowledge Transfer

ZSL has been recently used to generate images as well. Reed *et al*. show that the relationship between semantics and visual features can be used to generate images given a description [41]. This is achieved by making use of the adversarial training on images and their semantic descriptions. Furthermore, Google translate now uses zero-shot translation [23].

An interesting application of TSN learning is to protect sensitive data as deep networks tend to memorize certain training instances. Recent studies show that this can become a security concern, especially when training data contains sensitive information such as face images or medical records [14]. To this end, the work in [38] suggests training a student network using non-sensitive data to mimic the teacher network. Upon training, the student network can perform at a reasonable level of accuracy while sensitive data is not used for training.

To achieve real-time action recognition, a TSN framework is proposed where the teacher network receives the optical flow information as inputs while the student network sees the motion vectors at its input. The TSN then tries to tune the parameters of the student network to estimate the outputs of the teacher network operating with the optical flow information. This enables one to feed a network with motion vectors and attain the performance of a network working with optical flow information in real time.

VI. Epilogue

Machine learning has seeped into our day to day life in many forms. Sophisticated search engines, accurate translators, and access control systems are a few of them. Many of these are marvels of supervised learning. On top of these, limited and no supervision learning methods have potential to open up a new generation of applications that require deeper exploration of the data, which is difficult to be guided by a human. Such applications could extend from diagnosing diseases to autonomous robot swarms to explore the space beyond our reach.

Acknowledgement

We extend our sincerest gratitude to Dr. Mehrtash Harandi for his comments and sharing his knowledge on unsupervised learning methods. This work was supported under the Australian Research Councils Discovery Projects funding scheme (project DP150104645) and by DATA61-CSIRO.

REFERENCES

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017. IV-B

[2] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 2654–2662, 2014. V-D, V-D

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Proc. Advances in Neural Information Processing Systems (NIPS)*, 19:153, 2007. III-A, III-B

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proc. Int. Conference on Machine Learning (ICML)*, pages 41–48, 2009. V-D

[5] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294, 1988. III-A

[6] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006. V-D

[7] Susan Carey and Elsa Bartlett. Acquiring a single new word. 1978. V-C

[8] Minmin Chen, Zhixiang Xu, Fei Sha, and Kilian Q Weinberger. Marginalized denoising autoencoders for domain adaptation. In *Proc. Int. Conference on Machine Learning (ICML)*, pages 767–774, 2012. III-B

[9] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016. IV-B

[10] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. Int. Conference on Machine Learning (ICML)*, pages 160–167, 2008. V-B

[11] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 16:2859–2900, 2015. III-A

[12] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016. IV-A

[13] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016. 7, IV-B

[14] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015. V-E

[15] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 2121–2129, 2013. V-B

[16] Aude Genevay, Gabriel Peyré, and Marco Cuturi. Sinkhorn-autodiff: Tractable Wasserstein learning of generative models. *CoRR*, abs/1706.00292, 2017. IV-B

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. III-A

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014. IV-B

[19] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proc. Int. Conference on Machine Learning (ICML)*, pages 399–406, 2010. III-A

[20] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(10), 1990. V-D

[21] Bharath Hariharan and Ross Girshick. Low-shot visual object recognition. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. V-A, V-C

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. V, V-D, V-D

[23] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*, 2016. V-E

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. IV-C

[25] Gregory Koch. *Siamese neural networks for one-shot image recognition*. PhD thesis, University of Toronto, 2015. V-A, V-B

[26] Brenden M Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. 2011. V-C

[27] Yann LeCun. Learning invariant feature hierarchies. In *Proc. European Conference on Computer Vision (ECCV)*, pages 496–505. Springer, 2012. III-A

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. I

[29] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016. IV-C

[30] Jimmy Lei Ba, Kevin Swersky, Sanja Fidler, et al. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4247–4255, 2015. V-B

[31] Yujia Li, Kevin Swersky, and Richard Zemel. Generative moment matching networks. pages 1718–1727, 2015. IV-B

[32] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 469–477, 2016. IV-B

[33] Shuang Liu, Olivier Bousquet, and Kamalika Chaudhuri. Approximation and convergence properties of generative adversarial learning. *CoRR*, abs/1705.08991, 2017. IV-B

[34] William Lotter, Gabriel Kreiman, and David Cox. Unsupervised learning of visual structure using predictive generative networks. *arXiv preprint arXiv:1511.06380*, 2015. IV

[35] David G Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004. III

[36] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015. III-A, 3

[37] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: training generative neural samplers using variational divergence minimization. pages 271–279, 2016. IV-B

[38] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Machine learning with privacy by knowledge aggregation and transfer. 2017. V-E

[39] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. IV-C

[40] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 1137–1144, 2006. III-A

[41] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *Proc. Int. Conference on Machine Learning (ICML)*, pages 1060–1069, 2016. V-E

[42] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. V-D, V-D, 9

[43] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. pages 2226–2234, 2016. IV-B

[44] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 935–943, 2013. 8, V-B

[45] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010. III-A

[46] Xin Yu and Fatih Porikli. Face hallucination with tiny unaligned images by transformative discriminative neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*, 2017. IV-C

[47] Xin Yu and Fatih Porikli. Hallucinating very low-resolution unaligned and noisy face images by transformative discriminative autoencoders. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. IV-C

[48] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2528–2535, 2010. III-A

[49] Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *NIPS*, 1994. III-A

[50] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pages 597–613. Springer, 2016. IV-C