Progressive Ladder Networks for Semi-Supervised Transfer Learning

Darshan Thaker
The University of Texas at Austin
dbthaker@cs.utexas.edu

Abstract

Semi-supervised learning has achieved remarkable success in the past few years at harnessing the power of unlabeled data and tackling domains where few labeled data examples exist. We test the hypothesis that deep semi-supervised architectures learn general representations. We combine two well-known techniques for semi-supervised and transfer learning, ladder networks and progressive neural networks, to create the progressive ladder network, a framework for transferring from semi-supervised representations to supervised representations. We use this framework for domain adaptation in the digit recognition domain, from the MNIST handwritten digits dataset to the Street View House Numbers (SVHN) dataset. Our experiments show that semi-supervised representations are more transferable than supervised learning representations. We find that using our progressive ladder network architecture on simple fully connected neural networks trained on SVHN can yield an increase in test accuracy of about 40%*, from around* 35% *to* 77.7%*.*

1. Introduction

Semi-supervised learning deals with the problem of learning from datasets with few labeled examples and many unlabeled examples. The field lies at the intersection of supervised and unsupervised learning, motivated by the fact that unlabeled examples can complement the supervised learning process by exploiting some hidden latent representation of the unlabeled examples [15]. The usefulness of this form of learning is evident, given the lack of copious labeled examples in many real-world datasets.

In a related vein, *transfer learning* aims to leverage knowledge gained from one source task in another related, but separate target task. In cases where labeled data for the target task is rare, transfer learning has clear value. However, even in cases where this is not true, transfer learning can act as a meaningful mechanism of "bootstrapping" models trained on the target task with knowledge from a related source task. This bootstrapping might allow the model to generalize better to conditions which the model has not seen before in the training data. In this report, we deal with a subfield of transfer learning, *domain adaptation*, and apply it for deep neural networks trained on computer vision tasks. Formally, when performing domain adaptation, a model trained on some input distribution $\mathcal{X}$ and label space $\mathcal{Y}$ is tasked with predicting examples with a different input distribution $\mathcal{X}'$, but living in the sample label space $\mathcal{Y}$. Most simple approaches to deep transfer learning in the vision domain concern *pre-training* [9]. In this method, when training a neural network for a target task, weights of a neural network are initialized to the weights of a network trained on a related source task. For most computer vision related tasks, neural networks are initialized to a convolutional neural network trained on ImageNet ([5], [13], [2]). However, pre-training is inherently destructive to the weights of a network trained on a source task. This phenomenon is known as catastrophic forgetting, and is characterized by the knowledge of some task A degrading as knowledge is transferred and finetuned for another task B. A recent approach proposed by Rusu et al. attempted to tackle this problem through *progressive networks*, a model for transferring knowledge across tasks [12].

Combining these two fields yields *semi-supervised transfer learning*, a way to learn transferrable representations from both labeled and unlabeled data. Related work has primarily looked at ways to solve the problem of transferring models trained on task A, where labeled data is abundant, to a related task B, where labeled data is scarce or non-existent altogether. For example, Haeusser et al. propose a technique for associative domain adaptation, which yields impressive results transferring to tasks with no labeled data [1]. We consider the flipped task of performing domain adaptation from a domain with not much labeled data to a domain with large amounts of labeled data. This is a middle-ground between transferring representations learned from large amounts of labeled data in both domains and can be more useful than training from scratch on a new domain.

Contributions. The premise of our work is the hypothe-

sis that representations learned from semi-supervised learning approaches are not only generalizable when evaluated on the same dataset, but also are transferrable. To that end, we propose a combination of a ladder network (for semi-supervised learning) and a progressive network (for transfer learning), coined a *progressive ladder network*. We experiment with three variants of the ladder network: a fully connected ladder network, a convolutional ladder network and a progressive AlexNet combined with a ladder network [5], with emphasis on the fully connected ladder network. We find that for the domain of digit recognition, progressive ladder networks perform better than neural networks trained from scratch and also slightly outperform vanilla progressive networks trained using supervised mechanisms on two digit recognition domains. When trained for the same number of epochs, the progressive ladder network reaches a higher accuracy faster than the baseline, suggesting that the transfer serves as a meaningful initialization to the network. In addition, to analyze this phenomenon, we employ a perturbation mechanism on the layers of the progressive ladder network.

2. Background

2.1. Ladder Networks

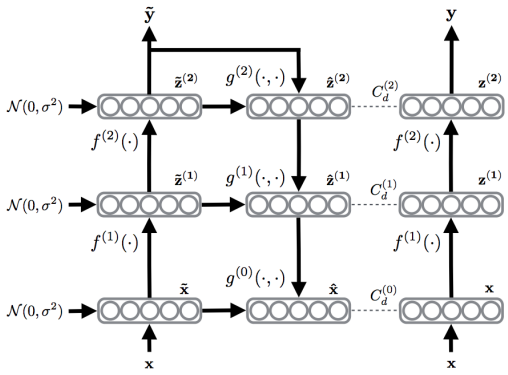


Figure 1: Ladder Network Architecture (Figure from [11])

Rasmus et al. propose an architecture for semi-supervised learning, coined the *ladder network* [11]. The network is built off an encoder and decoder model, with a supervised cost and unsupervised cost, as shown in Figure 1. The main difference between an autoencoder and a ladder network is the introduction of skip connections between each layer of the encoder and the corresponding layer in the decoder. Valpola et al. show that this network mimics a hierarchical latent variable model, increasing its representational power [16]. On the MNIST dataset [6] using only 100 labeled examples, ladder networks achieve an impressive error rate of 0.89%, as compared to a traditional convolutional neural network, which achieves a 6.43% error rate with the same number of labeled examples. We describe each part of the ladder network in detail in the following subsections, following by the presentation by Rasmus et al.

2.1.1 Encoder

The ladder network has two feedforward paths that share the same weights. One is a 'clean' encoder and the other is a 'corrupted' encoder. In the vanilla ladder network, the encoder is a fully connected multi-layer perceptron, with batch normalization after each layer [3]. For the corrupted encoder, isotropic Gaussian noise is added to inputs to each layer as well as the post batch normalization representations. This noise is drawn from the parametric normal distribution $\mathcal{N}(0, \sigma^2)$. Both encoders take in the same input x and share the same weight matrices, but the corrupted encoder outputs a noisy $\tilde{y}$, and the clean encoder outputs y. The output of the corrupt encoder is used to optimize a supervised learning cost for labeled examples L (the noise acts as a natural regularizer).

$$L = \{(x(0), t(0), \ldots, (x(N), t(N))\} \tag{1}$$

$$C_c = -\frac{1}{N}\sum_{n=1}^{N} \log P(\tilde{y} = t(n)|x(n)) \tag{2}$$

2.1.2 Decoder

The decoder is formulated as a "reverse" encoder, where the size of the weights of each layer of the decoder is the size of the transpose of the weight matrix of the corresponding layer of the encoder. Because the corrupted encoder yields a noisy representation $\tilde{y}$, the decoder must use a denoising function at each layer. Rasmus et al. utilize a Gaussian denoising function of the form in Equation 3. Slightly relaxing the assumption that each latent variable follows a Gaussian distribution, Rasmus et al. instead assume that the distribution of the latent variables on a layer of the encoder is Gaussian conditional on the values of the layer above, which is shown to be empirically than other simple denoising functions. Let $\hat{z}_i^{(l)}$ be the latent variable representation of the ith neuron in layer l of the decoder, following notation in Figure 1. Given a representation $\hat{z}^{(l+1)}$, Rasmus et al. propose to perform batch normalization and then apply a Gaussian denoising function, parametrized by a location and scale nonlinearity (similar to the noise variable introduced in variational autoencoders [4]). Let $b_i^{(l)}$ be the batch-normalized $\hat{z}_i^{(l)}$. Then, the Gaussian denoising function is of the form

$$\hat{z}_i^{(l)} = \left(\hat{z}_i^{(l)} - \mu_i(b_i^{(l)})\right) v_i(b_i^{(l)}) + \mu_i(b_i^{(l)})$$
$$\mu_i(b_i^{(l)}) = a_{1,i}^{(l)}\text{sigmoid}(a_{2,i}^{(l)} b_i^{(l)} + a_{3,i}^{(l)}) + a_{4,i}^{(l)} b_i^{(l)} + a_{5,i}^{(l)}$$
$$v_i(b_i^{(l)}) = a_{6,i}^{(l)}\text{sigmoid}(a_{7,i}^{(l)} b_i^{(l)} + a_{8i}^{(l)}) + a_{9,i}^{(l)} b_i^{(l)} + a_{10,i}^{(l)}$$

with learnable $a_{j,i}$ parameters. The decoder is used to perform unsupervised training using a per-layer mean-squared cost between the latent variable representation $z^{(l)}$ and the corresponding representation obtained by layer l of the clean encoder, as illustrated in Figure 1. There is a hyperparameter for each layer of the decoder, and this hyperparameter controls the contribution of each layer to the overall unsupervised cost. The intuition for this cost is that the layers of the clean encoder are forced to be similar to a noise-robust latent variable representation learned from unlabeled data.

Training the overall ladder network equates to minimizing the sum of the supervised and unsupervised cost. At evaluation time, only the clean encoder is used to make predictions for new examples.

2.2. Progressive Networks

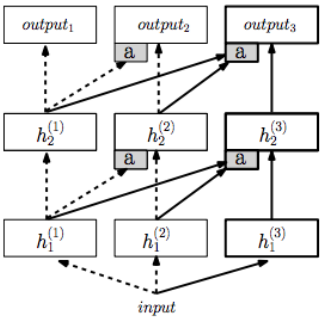


Figure 2: Progressive Network Architecture (Figure from [12])

Rusu et al. propose an effective framework for transfer learning, known as the *progressive network* architecture [12]. This architecture enables transfer from models trained on k tasks to a new related task. For this report, the only relevant case is when $k = 1$, so we are transferring from one domain to another domain. Suppose we have trained a multi-layer neural network on task A, denoted as a column, and wish to transfer the knowledge to another column (multilayer deep neural network) training on task B. Then, the idea of a progressive neural network is to tackle the problem of catastrophic forgetting by "freezing" the weights of the task A column and adding lateral connections from every layer l in the task A column to layer $l + 1$ in the task B column. To take into account different input scales for every column, the lateral connections are fed through an adapter, where a learnable scalar coefficient is multiplied by the weights. A more complex, but similar, case for transferring knowledge from 2 tasks to a 3rd task is shown in Figure 2. Rusu et al. show that this simple framework works very well for transfer in deep reinforcement learning domains.

3. Progressive Ladder Networks

Pezeshki et al. performed various ablation studies for the ladder network to isolate the important parts of the architecture [10]. They find that the most important parts of the architecture were the introduction of lateral connections in the autoencoder and the introduction of noise in the corrupted encoder. The latter was found to greatly improve generalization. Expanding upon this, we test transferability of the ladder network by combining the ladder network with the progressive neural network, coined as the *progressive ladder network*. Combining these ideas is fairly straightforward, since it involves training a ladder network on a source dataset using semi-supervised approaches, and then using the clean encoder as a column for a progressive neural network. The other column is a neural network with the same architecture as the clean encoder, but trained fully supervised on a target dataset. This allows us to test transferability of semi-supervised representations when used for fully supervised training of related tasks. Specifically, for our vanilla ladder network, we use a multi-layer perceptron with 7 fully connected layers, with $784, 1000, 500, 250, 250, 250, 10$ number of neurons in each layer respectively. The hyperparameters for choosing the contribution of each layer of the decoder to the unsupervised cost (as explained in Section 2.1.2), is chosen to be $[1000, 10, 0.1, 0.1, 0.1, 0.1, 0.1]$.

3.1. Variants

5 × 5 conv. 32 ReLU
2 × 2 max-pooling stride 2 BN
3 × 3 conv. 64 BN ReLU
3 × 3 conv. 64 BN ReLU
2 × 2 max-pooling stride 2 BN
3 × 3 conv. 128 BN ReLU
1 × 1 conv. 10 BN ReLU
global meanpool BN
fully connected 10 BN

Figure 3: ConvPool-CNN-C Variant for Conv Ladder Network (Figure from [11])

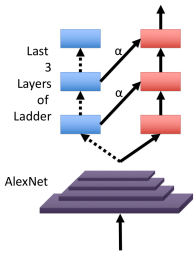


Figure 4: Progressive AlexNet

Rasmus et al. experiment with variants of the ladder network, the major variant being a convolutional ladder network. The network used is the same used as Rasmus et al., inspired by ConvPool-CNN-C from Springenberg et al. [14]. The specific network is shown in Figure 3. For this network, they experimented with a simple model, called the Γ-model, where the unsupervised cost is only considered for the top layer of the corrupted encoder, so there is no implementation of the decoder. We tried using a convolutional ladder network for transfer as well, trained via the Γ-model with no decoder implementation.

Additionally, we experimented with a progressive neural network from the ladder network to a fully supervised model appended to a pre-trained network such as AlexNet [5]. This is shown in Figure 4.

4. Experiments

4.1. Datasets

We experiment with domain adaptation for the digit recognition domain. Specifically, we explore transferability from the MNIST handwritten digits dataset [6] to the Street View House Numbers (SVHN) dataset [8]. We pick these datasets for two reasons:

- The simplicity of the handwritten digit domain complements the current state of semi-supervised learning research, since these approaches do not work very well on advanced datasets where latent variable representations are difficult to extract in the first place. Additionally, this allows use of experimentation with simple fully connected multi-layer perceptrons for the ladder network as opposed to large and deep advanced networks, reducing training time significantly.

- The domain transfer problem is nontrivial, since the SVHN dataset contains many more artifacts than the sanitized MNIST dataset. For example, in the SVHN dataset, some images contain more than one number, but the neural network is tasked with predicting the middle number in the image. The background clutter is also much larger in the SVHN dataset, so a successful transfer approach would learn to use high-level features of a network trained for MNIST to recognize digits and then learn the remaining modalities of the dataset on its own.

For semi-supervised learning on MNIST, we use only 100 labeled examples for all our experiments.

4.2. Results

To fairly evaluate the transferability of semi-supervised representations, we compare 3 approaches for transfer against a baseline:

- *Supervised (Baseline)* - This is a network trained fully supervised on the SVHN dataset. For the fully connected ladder network, this baseline is the same as the network described in Section 3, and for the convolutional ladder network, this network is the same as shown in Figure 3.

- *Pre-trained Supervised* - When adapting domains from MNIST to SVHN, we first train a semi-supervised ladder network for MNIST. To get a baseline for naive transfer learning approaches, we then take this network as a pre-trained initialization and then finetune the full network on SVHN.

- *Supervised* $\rightarrow$ *Supervised* - Another approach for transfer is using progressive neural networks for transfer between two neural networks, both trained using fully supervised neural networks. Specifically, we train a neural network using all MNIST training data examples and then combine this network with a progressive architecture to train a neural network for the SVHN dataset.

- *Semi-Supervised* $\rightarrow$ *Supervised* - This is the progressive ladder network approach, performing transfer learning from a semi-supervised learning approach to a supervised learning approach.

For fair comparisons between *Semi-Supervised* $\rightarrow$ *Supervised* and *Supervised* $\rightarrow$ *Supervised*, we train both source networks on MNIST (fully supervised and semi-supervised) to the same test accuracy. Thus, whichever transfer approach works better suggests that the corresponding source MNIST network is generalizing better to the digit recognition domain space. These networks were all trained for 100 epochs, which is a fairly small learning time. This is because our aim is not to show the maximum accuracy a classifier can reach, but rather the importance of transfer in the initialization of a neural network.

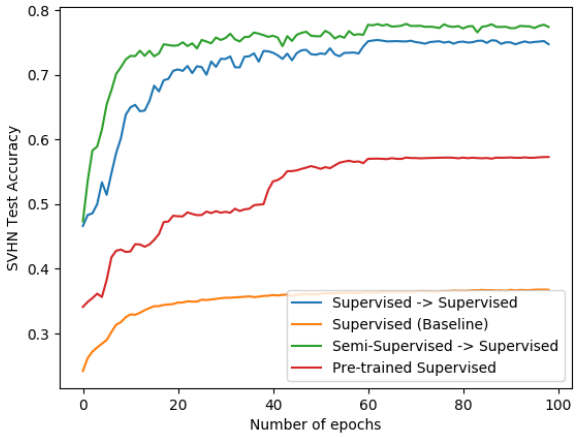


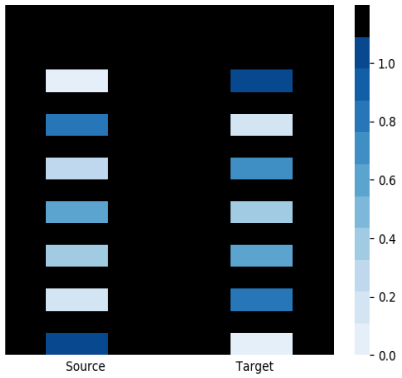
Figure 5: Fully Connected MLP Transfer Analysis

Fully connected MLPs. We evaluate the fully connected MLP as an encoder. This network is fairly simple, since it is comprised of only fully connected layers. The results of comparing the 4 approaches described in the previous section are shown in Figure 5. The semi-supervised ladder network was trained to 94.77% accuracy on MNIST dataset, and the supervised baseline network was trained to 95.8% accuracy.

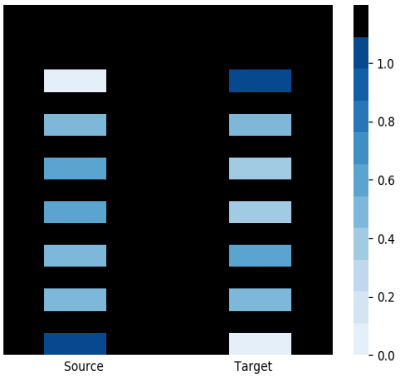
From Figure 5, it is clear that the progressive ladder network performs better than the other approaches, when trained for the same number of epochs. It performs better than the alternatives in two main ways. First, it increases test accuracy on SVHN by approximately 40% when comparing the baseline to the progressive ladder network, which is a considerable gain in accuracy. Secondly, the progressive ladder network reaches test accuracy on SVHN of around 0.74 in just 10 epochs of training, whereas the *Supervised* $\rightarrow$ *Supervised* approach reaches that accuracy in 60 epochs, and the other two baselines do not reach that high of a test accuracy in 100 epochs of training. The progressive ladder network seems valuable, since it improves accuracy significantly of a simple network that is comprised of only 7 fully connected layers. Current state of the art approaches for the SVHN dataset reach close to 98.31% test accuracy (Ex. [7]), but all these top approaches use very deep convolutional neural networks. It would be interesting to try integrating these approaches with progressive ladder networks to see if convergence to high test accuracies on SVHN is faster, but this was out of scope for this project.

Convolutional Networks. Next, we experimented with the architecture shown in Figure 3 for the progressive ladder network. Unfortunately, this did not work as well for MNIST semi-supervised training as the implementation by Rasmus et al did. Our implementation reached only 82% accuracy on the MNIST dataset with 100 labeled examples. Consequently, as expected, using this architecture for the progressive ladder network did not work well, yielding a test accuracy of around 50%. We have several hypotheses for why this did not work well initially. The first, explaining why the convolutional ladder network did not work well for MNIST semi-supervised training, is the extensive hyperparameter search needed for the ladder network. Specifically, the hyperparameter controlling the contribution of the unsupervised cost at each layer of the decoder (described in Section 2.1.2) was crucial in achieving a high test accuracy. This makes sense, since this cost controls how much each layer of the clean encoder will try to match the denoised representation at the corresponding layer of the decoder. Tuning this hyperparameter further may have led to better results on the MNIST task, and correspondingly better results when evaluated using the progressive ladder network. Additionally, we experimented only with the Γ variant of the ladder network (as described in Section 3.1). This means that only the top layer of the encoder is forced to have representations similar to the decoder representations, which makes it very similar to the *Supervised* $\rightarrow$ *Supervised* baseline for transfer, especially since the last layer of the ladder network is not used for transfer in the progressive architecture. This explains the poor results when the convolutional Γ ladder network was evaluated on SVHN using a progressive ladder network architecture.

Progressive AlexNet. When we evaluated the architecture for AlexNet and using a mini-progressive network



(a) Progressive Ladder Network



(b) Baseline *Supervised* → *Supervised*

Figure 6: Perturbation Analyses showing APS for each layer

Ablations	Best Test Accuracy
MLP Encoder	77.7%
Without scalar coefficient	7%

Figure 7: Progressive Network Ablation Study for Learnable Scalar Coefficients

for the fully-connected layers, we did not achieve great performance on the SVHN dataset, achieving only 30.43% test accuracy after training for 100 epochs. This is probably because the output of the last convolutional layer of AlexNet does not match the typical input to the fully connected layer trained on MNIST. The fully connected layer trained on MNIST was a higher-level layer in the fully connected ladder network, whose inputs from the previous layer in the source network do not match in scale to the inputs of the AlexNet convolutional layer. This input mismatch means the network is likely performing worse with the progressive neural network than training from scratch.

Analysis and Discussion. As a general theme, especially for the fully connected MLP network, we found that the initialization of the neural network influenced performance drastically. Given a "bad" initialization, results could be near random on the SVHN dataset, which exposes the flaw of using fully connected MLPs for columns of the progressive neural network. Additionally, we performed an ablation study of the learnable scalar coefficient for the lateral connections in the progressive neural network, and found these to be very important for learning. If this parameter was not added, the results were worse than random. This shows that despite lying in the same label space, the MNIST and SVHN dataset are different enough domains, where simply adding lateral connections gives a terrible initialization to the model trained on the SVHN dataset. These results are shown in Figure 7.

To gain some more insights into the difference between the progressive ladder network and the *Supervised* → *Supervised* transfer approach, we employ a perturbation scheme, as proposed by Rusu et al. [12]. Specifically, we wish to find the contribution of each layer of both the source column and target column to overall performance on the SVHN task. To evaluate this, we inject Gaussian noise layers after each layer in both source and target columns. During test evaluation, if we wish to find the contribution of layer l to the output, we inject various levels of Gaussian noise (post-RELU) after layer l. We define the noise precision of a layer l at column k, or $\Lambda_i^{(k)}$, to be the reciprocal of the variance of Gaussian noise (σ^2) added such that test accuracy is halved. Then, we define the Average Perturbation Sensitivity (APS) of a layer i in column k to be:

$$APS(i,k) = \frac{\Lambda_i^{(k)}}{\sum_k \Lambda_i^{(k)}} \tag{3}$$

The average perturbation sensitivity is a normalized value that roughly indicates the contribution of a specific layer to the output, because if a layer can be perturbed only slightly to decrease performance significantly, this layer likely contains features that are very important for classification accuracy. The results of evaluating this on the progressive ladder network and the *Supervised* → *Supervised*

transfer approach are shown in Figure 6. From this figure, it is clear that there are obvious wins from using the progressive architecture, one of which being that the first layer of the source column is reused significantly when training on the target task, rendering the first layer of the target column highly unimportant. This is understandable, since the first layer typically encodes low-level features of digits that are easily transferable across different digit recognition domains like MNIST to SVHN. The results from Figure 5 would indicate that the progressive ladder network is "more" transferable than the *Supervised* $\rightarrow$ *Supervised* approach. However, some of the APS results contradict this claim, since for the *Supervised* $\rightarrow$ *Supervised*, the middle layers (layer 4 and 5) are deemed more important than the corresponding layers in the progressive ladder network. Layer 6, on the other hand, is significantly more important in the progressive ladder network than in the *Supervised* $\rightarrow$ *Supervised* approach. Visualizing these features in a fully connected layer, especially layers deep in the network, is tricky, which is why we did not experiment with these visualizations, but this may provide some insight in the future into this confusing phenomenon.

5. Conclusion

We introduce the *progressive ladder network*, a combination of a progressive network and a ladder network, as a framework for transfer learning from semi-supervised representations to supervised representations. We test this framework for domain adaptation in the simple domain of transfer from the MNIST dataset to the SVHN dataset, and find that the framework yields considerable gains in accuracy and number of epochs taken to reach high accuracy levels. The next step for testing this framework is adapting it to more complex domains and testing out more complex variants of the ladder network involving convolutional networks, since this might provide better results on the SVHN domain than the fully-connected ladder network employed for our experiments in this report.

References

[1] P. Haeusser, T. Frerix, A. Mordvintsev, and D. Cremers. Associative domain adaptation. In *International Conference on Computer Vision (ICCV)*, 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[4] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial Intelligence and Statistics*, pages 464–472, 2016.

[8] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[9] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[10] M. Pezeshki, L. Fan, P. Brakel, A. Courville, and Y. Bengio. Deconstructing the ladder network architecture. In *International Conference on Machine Learning*, pages 2368–2376, 2016.

[11] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[12] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[15] S. C. Suddarth and Y. Kergosien. Rule-injection hints as a means of improving network performance and learning time. In *Neural Networks*, pages 120–129. Springer, 1990.

[16] H. Valpola. From neural pca to deep unsupervised learning. *Advances in Independent Component Analysis and Learning Machines*, pages 143–171, 2015.