Transfer Learning with Feedback Networks

Maxwell Spero
Stanford University
maxspero@stanford.edu

Abstract

Transfer learning is a tool often used to take knowledge learned from one domain and applying it to another. Pre-trained feedforward convolutional neural networks, trained on a large dataset like ImageNet, are often used for other visual recognition tasks by training a new model on feature representations extracted from the image using these pre-trained networks. Feedback networks are an alternative tool that find significantly different representations by running a representation of an image through a feedback loop and re-classifying at every iteration. We apply a range of transfer learning techniques from feedforward networks to feedback networks and show that important properties from feedback networks, such as early prediction hold across transfer learning to a new dataset.

1. Introduction

Feedback networks have several distinct advantages over traditional feedforward networks. One main advantage is early prediction. When training a feedback network, we make an attempt to predict after every feedback iteration. Because of this, our network makes its best attempt at a good feature representation by the end of an iteration, and every iteration strives for the same feature representation. This gives us an interesting property of feedback networks: early prediction. Because we can predict after every iteration, by convergence every the output of every iteration should be at least interpretable. While the first iteration may not perform as well as the the final iteration, it provides some good information in a fraction of the inference time.

By applying transfer learning techniques to learn from the feature representation at the end of an iteration, I show that we keep this advantage of early prediction in the new network, even without any more fine-tuning to the feedback layers. I found that a model trained on only the final feature representation after the last iteration of the feedback network is still able to predict from outputs of earlier iterations with reasonable accuracy.

One disadvantage of not using feature representations from a feedforward network is that we lose the ability to choose how low- or high-level our features are by choosing which layer we predict from. With feed-forward networks, we can use earlier convolutional layers to see a feature representation of lower level features such as shape or style. Or, we can use later convolutional layers which tend to have neurons that encode more meaning about the object and its identity.

By using feedback output instead of a feedforward network, we let go of the ability to choose which convolutional layer to use, but in place of it gain a feature embedding that could possibly hold more meaning than any single convolutional layer.

Note: this project was a combined final project for CS331B and CS229. Approval for this joint project was obtained from both classes. The deep learning parts of this project are for CS331B and the non-deep learning parts are for CS229.

2. Related work

Some of the first research to show great results from transfer learning was by using features from CNNs trained on ImageNet [2, 12, 11]. These researchers used CNN features to train an SVM or logistic regression model. These showed incredible, even state-of-the-art results using simple methods even without fine-tuning. Yosinksi et al. [14] dove deeper into transfer learning, showing that transferred features gave boosts to network performance after fine-tuning. They also showed how the benefits decreased as base and target task diverged, but lower level convolution weights were more transferable, and they got more specific the deeper the network became. It is interesting to study transfer learning in regards to feedback networks, because the way weights are shared, all feedback layers show up in each $1/n$ of the feedback module, where n is the number of iterations. That means "earlier" layers also become "later" layers and vice versa – it is unclear without studying this how general the outputs of a feedback module might be.

In other related work, feedback has been around for a long time in the form of recurrence. LSTM, which stands

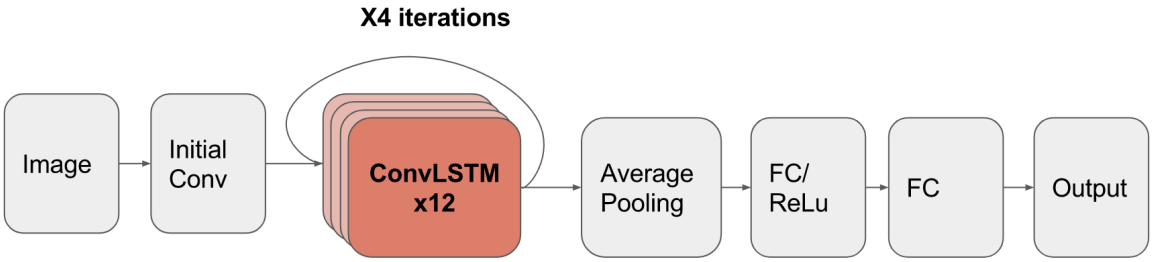

Figure 1. The architecture used for initial training of the feedback network

for Long short-term memory [4], is a form of a recurrent neural network from 1997 that provided great results back then. In fact, LSTMs are still often used today in text understanding. Convolutional LSTM networks [13] have been used to incorporate both the spatial and temporal dimension into predictions for weather. Recurrent models have also been used to add attention into networks [8] to save computation time and improve performance across different tasks.

Recurrence is even used over the entire body of a neural network, as seen in Feedback Networks [15]. In this implementation, all of the computation except an initial convolution and a final fully connected layer are recurrent, and can improve with each iteration. They show that early prediction works well with their network, and the feedback element enforces a curriculum that allows for better training. I will explore this specific architecture in more depth, as it is a very general architecture with the main focus being feedback.

3. Feedback networks

I implemented a few feedback network architectures in PyTorch [9]. The architecture used is very similiar to the one described in Feedback Networks [15]. It served as a good enough baseline, so I tested a few changes, used the ones that gave me the best performance, and then moved on to the transfer learning experiments. I spent about half of my time implementing the initial network and half of my time setting up the transfer learning experiments.

3.1. Convolutional LSTM

The core of these feedback network implementations is an operation called a convolutional LSTM (ConvLSTM) [13]. In a ConvLSTM, information propagates in two directions: spatially and temporally. Spatial order is denoted by superscript d and temporal order is denoted by subscript t. Each ConvLSTM layer at depth d and time t receives spatial input $\mathbf{X}_t^{d-1}$ from the previous depth, and temporal hidden state input $\mathbf{H}_{t-1}^d$ from the previous iteration. Like an LSTM, a ConvLSTM has an input gate, forget gate, cell gate, and output gate. To borrow from the Feedback Networks paper [15], the following is how a ConvLSTM is formulated.

$$i_t^d = \sigma(W_{d,xi}(\mathbf{X}_t^{d-1}) + W_{d,hi}(\mathbf{H}_{t-1}^d)) \quad (1)$$

$$f_t^d = \sigma(W_{d,xf}(\mathbf{X}_t^{d-1}) + W_{d,hf}(\mathbf{H}_{t-1}^d)) \quad (2)$$

$$\hat{C}_t^d := \tanh(W_{d,xc}(\mathbf{X}_t^{d-1}) + W_{d,hc}(\mathbf{H}_{t-1}^d)) \quad (3)$$

$$C_t^d = f_t^d \circ C_{t-1}^d + i_t^d \circ \hat{C}_t^d \quad (4)$$

$$o_t^d := \sigma(W_{d,xo}(\mathbf{X}_t^{d-1}) + W_{d,ho}(\mathbf{H}_{t-1}^d)) \quad (5)$$

$$\mathbf{H}_t^d, \mathbf{X}_t^d = o_t^d \circ tanh(C_t^d) \quad (6)$$

Where σ denotes the sigmoid function and $\circ$ denotes the Hadamard product. $\mathbf{X}_t^0$ is the input to the ConvLSTM layer and $\mathbf{H}_0^d$ is initialized to 0. The function W is a design choice. In the original Convolutional LSTM paper, Xingjian et al. used a single convolution followed by a Spatial BatchNorm [5] as W. In the Feedback Networks paper, Zamir et al. used multiple stacked convolutions. I used a single convolution as W for simplicity.

3.2. Architecture

The architecture I ended up using is shown in Figure 1. The initial convolution uses a 3x3 filter with stride 1 and 16 channels of output, followed by a Spatial BatchNorm. The feedback module used is a stack of 12 ConvLSTM layers.

I will describe a ConvLSTM layer by its input i and output o channels as $i \rightarrow o$, with stride o/i. All of the ConvLSTM layers had 3×3 convolution filters for input and hidden state. The ConvLSTM stack I used has three $16 \rightarrow 16$ layers, a $16 \rightarrow 32$ layer, two $32 \rightarrow 32$ layers, a $32 \rightarrow 64$ layer, and five $64 \rightarrow 64$ layers. All of these layers combined form one iteration, with weights shared across iterations. There are four iterations.

Connected to the output of every feedback iteration is a 4×4 average pooling layer, a $256 \rightarrow 256$ fully connected

layer, a ReLU activation function, and a $256 \rightarrow N$ fully connected output layer, where N is the number of classes we are training on.

3.3. Loss

Loss is computed at every iteration. Let L_i be the cross entropy loss of the output at iteration i and let L be the total loss.

$$L = \sum_{i}^{n} \gamma^i L_i \tag{7}$$

Where γ is the discount factor. A value of $\gamma = 1$ would give equal value to all iterations. I used $\gamma = 1.2$ which emphasized improvement over time and provided better final-iteration results.

4. Initial architecture experiments

I did some experiments with architecture, going through a few iterations before coming to the architecture described above. I tried an architecture with physical depth of 8 and three iterations. That's 24 ConvLSTM layers the input is passed through (eight unique ones), which is half as many as the 48 layers (12 unique) that were used for the final network. I found that greater depth, meaning more unique layers, improved the network more than additional iterations.

I originally implemented this network with only a single fully connected layer at the end, but I don't think that sorted out nonlinearities well enough after the feedback module, so I added a second layer. I also added dropout to the fully connected layer (not the output) to combat overfitting.

5. Datasets

I used three different datasets for training Feedback networks. CIFAR-10 [7] was a clear first choice. At 10 classes, and 6000 images per class, this is a dataset with plenty of training data to go around. In each image, the subject is centered and generally fills the frame. Each image is only 32×32, which is very small compared to the standard 224×224 image size in more state-of-the-art models today. However, 32×32 already fit the architecture described by Zamir et al. Additionally, this seemed like it would take less time to train on than using large images from a dataset like ImageNet, so I stuck with it.

Another dataset I used was CIFAR-100 [7]. It is a good comparison to CIFAR-10 because of its similarities to CIFAR-10. CIFAR-100 is basically the same as CIFAR-10, but it has 100 classes. It has the same amount of total training data, which means 600 images for each class. CIFAR-100 contains twenty superclasses of 5 classes each, and there ends up being almost no overlap between the classes in CIFAR-100 and CIFAR-10 (the one exception to that is the car class in CIFAR-10 is very similar to streetcar in CIFAR-100).

The third dataset I used is the PASCAL Visual Object Classes [3] dataset from 2012. Images were around 500px on their long end, but varied. This dataset can be used as a segmentation challenge, as there are segmentation maps available for each image, however I used it as a multi-label classification task to keep it simple. The largest challenge here was classifying multiple objects: if a person is on a horse, Person and Horse (and only these two) must both be classified at 1 to obtain loss of 0. This is also a challenging dataset because the subject to be labeled is often not centered , not cropped, and there is a lot more noise in terms of busy details around the subject. To fit the architecture used for CIFAR-10 and CIFAR-100, we resized these images to 32x32, which was likely too small and hurt performance by making the task even more challenging. However, all we aimed to show was that transferring knowledge from other datasets could improve performance on this one, so it wasn't too much of a concern that we lost some performance in resizing. Lastly, Pascal VOC contained only 1465 training examples, which is much less than the 60000 images in the other two datasets. This lack of training data was another reason why transfer learning has so much potential to be powerful.

For preprocessing, I normalized all images to 0.5 mean and 0.5 standard deviation before training.

6. Initial training

I trained networks on each dataset individually for 25 epochs with the Adam optimizer [6]. By 25 epochs, all of the networks had begun to overfit and validation accuracy was no longer going up so I stopped training. Validation accuracies for each dataset and at each iteration can be found in Table 1.

	CIFAR-10	CIFAR-100	Pascal VOC
Iteration 1	78.70%	33.25%	35.05%
Iteration 2	80.63%	37.22%	36.29%
Iteration 3	81.21%	39.18%	37.80%
Iteration 4	81.36%	38.85%	37.90%

Table 1. Validation accuracies after training on each dataset

Accuracy for Pascal VOC was about as low as CIFAR-100, despite having 1/5 as many labels. This was likely due to the fact that Pascal VOC was a much more difficult task, with uncentered and uncropped subjects and only a fraction of the training data. This was something we hoped to be able to improve on by using transfer learning to boost Pascal VOC accuracy with results from CIFAR-10 and CIFAR-100.

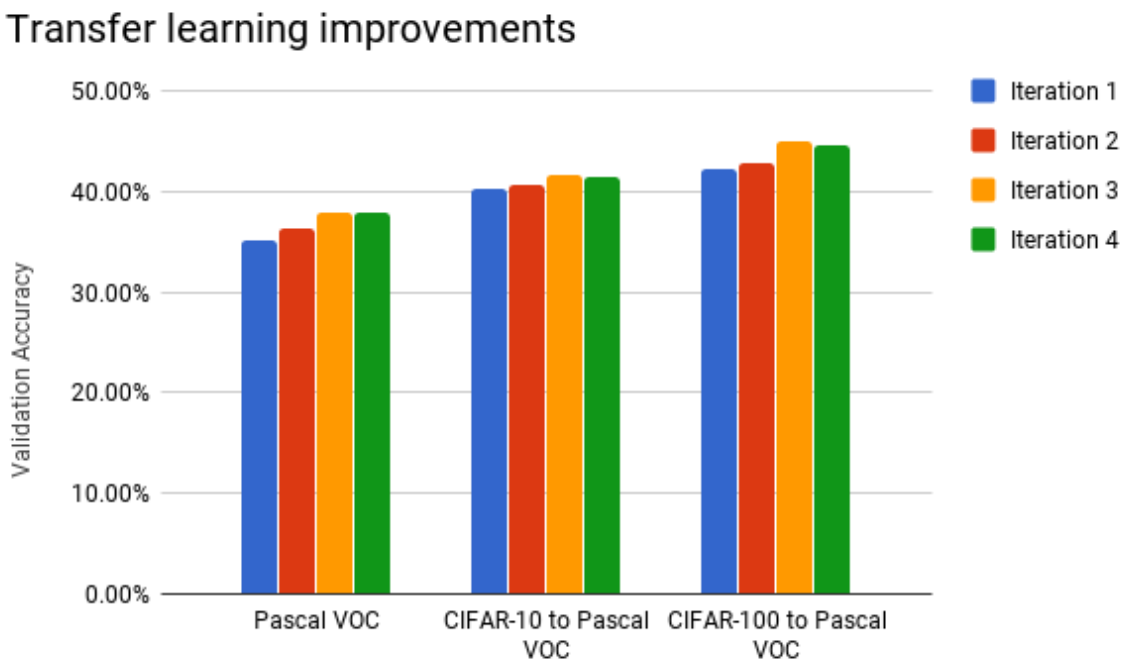

Figure 2. Improvements shown when re-training final two fully connected layers for Pascal VOC

7. Experiments

7.1. Fine tuning

The first experiment was fine-tuning the feedback network. I took the network that had converged in training on CIFAR-10, and reset the weights for the final to fully-connected layers to be randomly initialized (changing the number of outputs from 10 to 20). I then froze the weights for every layer in the network except the newly initialized layers. Finally, I trained until convergence (5 epochs) on Pascal VOC and recorded the results.

I repeated the experiment with CIFAR-100. The results are shown in Figure 2. Transfer learning from CIFAR-10 to Pascal VOC gave a 4-5% boost in performance across all iterations. Transfer learning from CIFAR-100 to Pascal VOC gave a 7-8% boost in performance across all iterations.

Despite having the same number of training examples, it seems CIFAR-100 generalized better than CIFAR-10 due to having seen more classes during training.

This result also shows that we can still get good results without fine-tuning the feedback layers, meaning that feedback layers still can generalize information about inputs as well as feedforward layers can. Additionally, this shows that "early prediction" property of feedback networks still holds after transfer learning.

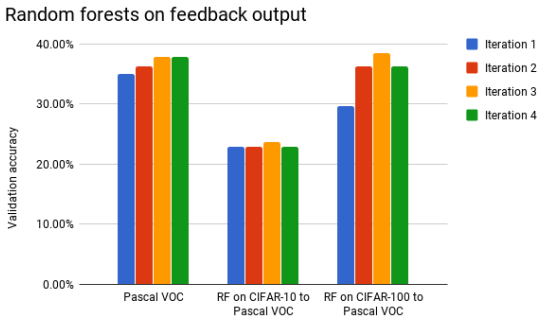

Figure 3. Attempts to use feature embeddings from networks to train a random forest for multilabel classification on Pascal VOC

7.2. Random forests

The second experiment was seeing how well random forests [1] performed on the feature embedding that is the output of the feedback module. I used random forests because they can provide binary predictions for the presence of every class, making them a good learner to tackle Pascal VOC. I used 128-tree random forests, implementation by the scikit-learn library [10]. Figure 3 shows the results of training random forests on the output of iteration 4.

Results did not beat the feedback network trained entirely on Pascal VOC, and I believe that is because random

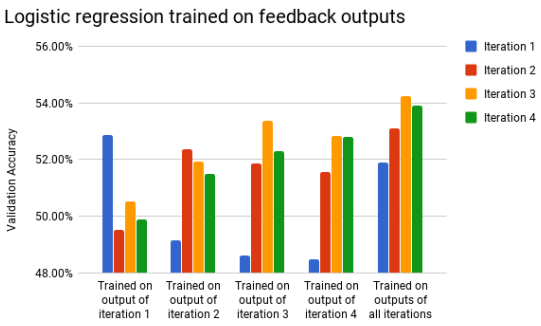

Figure 4. Reduced overfitting shown when trained on outputs of all feedback iterations

forests were not able to capture the complexity of the feature embeddings like transfer layers did in the section on fine tuning. This experiment serves to show that using a feature embedding from another network trained on a different dataset, with no fine tuning, was not enough by itself to provide good results on a harder problem like Pascal VOC.

7.3. Which iteration is best to train on?

I did one final experiment with transfer learning from the CIFAR-100 network to CIFAR-10, to stay in the single-label domain with similar images. As mentioned earlier, there is almost no overlap between classes in CIFAR-100 to CIFAR-10. By choosing multinomial logistic regression, a linear classifier, we use a much simpler classifier and do not expect to obtain the same accuracy as two fully connected layers could. Because it fits so quickly, it opened up the doors to some tests. We trained five logistic regression models, again using the scikit-learn library [10].

I fit one logistic regression model to the output of each iteration of feedback (four total) and one logistic regression model on the output of all of the iterations (each iteration output as a separate training example). I then tested each of these models on *all* outputs with a validation set.

Output is shown in Figure 4. Each model that was fit to only one iteration ended up performing best on that iteration, but still performed decently on the other iterations. However, by fitting to the output of all iterations, the logistic regression model was able to overfit less and actually perform better across the board. So to answer the question of which iteration is best to train on, the answer is all of them.

These results show that a linear classifier is enough to obtain decent results across transfer learning when the domains are as close as CIFAR-100 to CIFAR-10. Additionally, it shows that feature embeddings are stable and interpretable across all iterations. While there was varying performance depending on which iteration's output the model was trained on, accuracies were all in the same general range, which is a good indicator that the feedback network is stable enough that feature embeddings are linearly separable into predictions at all iterations.

8. Conclusion

One important thing to know is that my specific feedback network implementation did not perform as well as state-of-the-art feedforward networks, in initial training or transfer learning efficacy. However, I already knew and expected that going in. My goal with this project was instead to explore the properties of feedback networks through transfer learning and see which properties hold. We showed that transfer learning works on feedback networks, just like feed-forward networks, even without having the need to fine-tune the feedback layers. Additionally, feature embeddings from feedback are relatively stable between iterations, even after being transfered to a dataset and without fine-tuning. This allows even linear classifiers to interpret these feature embeddings with decent accuracy. Ultimately, we see that the interesting properties of feedback networks, such as early prediction and similar feature embeddings across iterations, do hold across transfer learning.

9. Future Work

I would like to try these results and more with a more effective feedback network. I don't think the architecture I used was on par with current state-of-the-art networks. The ConvLSTM layer is significantly slower to compute and slower to train than a standard convolution layer, because it has eight times as many parameters. Computing all of the LSTM gate operations is also significantly more expensive than a simple ReLU. Computing loss over four iterations also increases the number of computations made nearly fourfold as well. It was very slow to train this network on CIFAR-10 and that isn't even a very big dataset compared to ImageNet, with its millions of images.

I do believe it is likely there will soon be more efficient networks out there that incorporate feedback, and maybe one day they will outperform feedforward networks while maintaining these interesting properties that I studied in this paper. I think there is lots of work to be done in finding efficient ways to create deep networks that incorporate feedback, training them on ImageNet, and coming back to experiments like these with much better results.

References

[1] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 4

[2] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014. 1

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 3

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 1

[5] 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. 2

[6] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[7] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 3

[8] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. Recurrent models of visual attention. *CoRR*, abs/1406.6247, 2014. 2

[9] 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. 2017. 2

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 4, 5

[11] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014. 1

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[13] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015. 2

[14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014. 1

[15] A. R. Zamir, T.-L. Wu, L. Sun, W. B. Shen, B. E. Shi, J. Malik, and S. Savarese. Feedback networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1808–1817. IEEE, 2017. 2