Quantifying the transferability of features in deep neural networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

A high percentage of recently reported deep networks trained on natural images exhibit a curious aspect in common: they all learn features similar to gabor filters on the first layer. Such features appear to be *general*, as opposed to *specific* for a particular task. Because higher layers are difficult to visualize, not as much is known about whether the are general or specific. In this paper we experimentally quantify the generality of neurons on each layer of a deep convolutional neural network and through this expose a few surprising aspects. First, we find that transferability is negatively affected by two distinct issues — not only the expected specialization of higher layer neurons to their original task at the expense of the target task, but also optimization difficulties related to splitting networks in the middle of co-adapted neurons. In an example network trained on ImageNet, we demonstrate that either of these two issues may dominate, depending on whether features are transferred from the bottom, middle, or top of the network. We also show how the transferability gap grows as the distance between base task and target task increases, but how even transfer from distant tasks can be better than using random features. Finally, we show a surprising result that initializing a network with transferred features from almost any number of layers can produce a boost to generalization that lingers even after fine-tuning to the target dataset.

1 Introduction

Many trained neural networks reported recently in the literature exhibit a curious aspect in common: they all learn features that resemble Gabor filters in the first layer. The appearance of these Gabor filters is so common that to obtain anything else on a natural-image dataset causes suspicion of poorly chosen hyperparameters or a software bug. This phenomenon occurs not only for different datasets, but even with very different training objectives, from supervised image classification (Krizhevsky *et al.*, 2012) to unsupervised sparse representation learning (Le *et al.*, 2011).

Because finding Gabor features on the first layer seems to occur regardless of the exact cost function and natural image dataset used, we may call these these features *general*. On the other hand, we know that the last layer of a trained network will depend greatly on the chosen {network model, dataset} combination. For example, for networks trained toward a supervised classification objective, each output unit will be specific to a particular class, at least to the extent that training is successful. In this sense, let's say that the last layer features are *specific*. So far these terms are only intuitively defined, but we shall provide a candidate for a more rigorous definition below. If these Gabor features of the first features are general, and the features of the last layer are specific, there must be a transition from general to specific somewhere in the network. This observation raises a few questions:

- Can we quantify the degree to which a particular layer is general or specific?

- For a particular deep network, does the transition occur suddenly at a single layer, or is it spread out over several layers?
- Where does this transition take place: near the bottom, middle, or top of the network?

Once one has general features from some layer of a *base* network trained on a dataset, a common trick is to repurpose the features, or *transfer* them, to another task (Caruana, 1995; Bengio *et al.*, 2011; Bengio, 2011; Donahue *et al.*, 2013). The usual course is to create a new network containing as its first n layers the first n layers of the base network, with the rest of the layers in the new network randomly initialized and trained toward the second *target* task. When following this path, one can choose to backpropagate the errors from the new task into the base (copied) features to *fine-tune* them to the new task, or they can be left frozen, meaning that they do not change during training on the new task. In cases where the target dataset is smaller than the base dataset, fine-tuning the base features may result in overfitting, so the features are often left frozen. On the other hand, if the target dataset is large enough, the base features can be fine-tuned to the new task. Of course, if the target dataset is very large, there would be little need to transfer because the base filters could just be learned from scratch. We compare results from each of these two domains — fine-tuned features or frozen features — in a later section.

In this paper we make several contributions:

1. We propose one way of answering the first question above through the degree to which features from one dataset transfer to another (Section 2). We then train a large convnet on the ImageNet dataset and use our definition of generality to examine the general to specific transition (Section 4). Others have loosely addressed the question of generaltiy by showing how features from higher layers in a convnet can be transferred to new tasks Donahue *et al.* (2013) but here we systematically characterize the general to specific transition, which yields the next three results.
2. We experimentally show that the difficulty in using transferred features can be decomposed into two separate issues: optimization difficulties due to breaking co-adaptation between neurons on neighboring layers and the specificity of the features themselves. In different regions of the network, we show how different effects dominate (Section 4.1).
3. We show how the transferability gap grows as the distance between base task and target task increases, and we compare both types of transfer to using random filters, which we find not to work as well on the large ImageNet dataset as for smaller datasets, as reported by Jarrett *et al.* (2009) (Section 4.2).
4. Finally, we find an interesting effect where fine-tuning filters from a different dataset can lead to better solutions, even when the datasets are the same size and training time is identical (Section 4.1).

2 Generality vs. Specificity as transfer from one dataset to another

We have noted the curious tendency of Gabor features to show up in the first layer of neural networks trained on natural images for a large range of network architectures and cost functions. We say these features are general if they are useful for many different tasks or specific if they are useful only for a few. We can measure the specificity of a set of features by training it on one task A and then using it for another task B. Of course, this definition depends on the distance between A and B. In this paper we take tasks A and B to be classifying different non-overlapping subsets of a particular dataset, ImageNet, as released in the Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) (Deng *et al.*, 2009). It contains 1,281,167 labeled training images and 50,000 test images, and each image in the training and validation sets is labeled with one of 1000 classes.

To create tasks A and B, we randomly split the 1000 classes into two groups, A and B, each containing 500 classes and approximately half of the data, or about 645,000 examples each. We train one convolutional net on the half-dataset A and one on the half-dataset B. These networks, which we call baseA and baseB, are shown in the top two positions of Figure 1. We then choose an n from $\{1, 2, \ldots, 7\}$ and train and compare several new networks. To make the discussion more concrete, let's pick for now $n = 3$ (Figure 1 also shows the $n = 3$ case). First, we define and train the following two networks:

- A *selffer* network B3B where the first 3 layers are copied from baseB and frozen. The five higher layers are initialized randomly and trained toward dataset B. Here we're only using data from B and layers from a network baseB that was trained on B. This may seem a little strange, but we actually need it as an additional control for the next transfer network.
- A *transfer* network A3B where the first 3 layers are copied from baseA and frozen. The five higher layers are initialized randomly and trained toward dataset B. Intuitively, here we're copying the first 3 layers from the wrong dataset, A, and then trying to learn higher layer features on top of them to classify B. If this works well, we'll say that those first three layers are general, at least with respect to B, and if not, we'll say they are specific to A.

Then we create and train fine-tuned versions of the two above networks:

- A *selffer* network B3B$^+$ just like B3B, but where all layers learn.
- A *transfer* network A3B$^+$ just like A3B, but where all layers learn.

Figure 1 shows in the bottom two positions the networks created for $n = 3$, but we repeated this process for all n in $\{1, 2, \ldots, 7\}$[1] and in both directions, i.e. transferring from A to B and B to A.

The results for the random A/B splits are shown in Section 4.1. Before proceeding, we note an important point about the random splits chosen in this way. ImageNet contains 1000 classes, some of which occur in clusters of related classes. For example, there happen to be many types of certain animals included in the dataset, particularly dogs and cats, like these 13 classes from the biological family *Felidae*: {`tabby cat, tiger cat, Persian cat, Siamese cat, Egyptian cat, mountain lion, lynx, leopard, snow leopard, jaguar, lion, tiger, cheetah`}

Because each randomly chosen A/B half will probably contain 6 or 7 of these felids, each half will have detectors trained on all levels to classify some types of felids. When generalizing to the other half, we would expect that the new high-level felid detectors trained on top of fixed old low-level felid detectors would work well. For comparison, we've also created a special split of the dataset into two halves that are as far apart from each other as possible: one category containing only *natural* classes and another containing *man-made* objects. This split is not quite even — there are 449 classes in the natural group and 551 in the man-made group. In Section 4.2 we show (a) how features transfer more poorly (so we say the are more specific) across this larger generalization gap and (b) how the generality of subsequent layers falls of smoothly.

3 Details of Experimental Setup

Since Krizhevsky *et al.* (2012) won the ImageNet 2012 competition, there has naturally been much interest and work toward tweaking hyperparameters of large convolutional models. For example, Zeiler and Fergus (2013) found that it is better to decrease the first layer filters sizes from 11×11 to 7×7 and to use a smaller stride of 2 instead of 4. For this study we decided to use the original architecture from Krizhevsky *et al.* (2012) as provided in the reference implementation in Caffe, a library for training convolutional models on a GPU (Jia, 2013). We aim not to maximize absolute performance, but to study transfer results on what is probably the most popular single architecture, so that our results will be comparable, extensible, and useful to the largest number of other researchers. We expect none of the qualitative results presented here to depend at all on small architecture tweaks.

For the bulk of the experiments in this paper, the master learning rate was started at 0.01, annealed over the course of training by dropping it by a factor of 10 every 100,000 iterations, and learning was stopped after 450,000 iterations. Each iteration took about 1.7 seconds, meaning that the whole training procedure for a single network takes 9.5 days. Because occupying a whole GPU for this long was cumbersome, we also devised a set of hyperparameters to allow faster learning by boosting the learning rate by 25% to 0.0125, annealing by a factor of 10 after only 64,000 iterations, and stopping after 200,000 iterations. These selections were made after looking at the learning curves

[1]Note that $n = 8$ doesn't make sense in any case: B8B is just baseB, R8B is just a completely retrained version of baseB, and A8B would have no hope of working, because nothing would be trained and the 500 softmax neurons would not match between A and B.

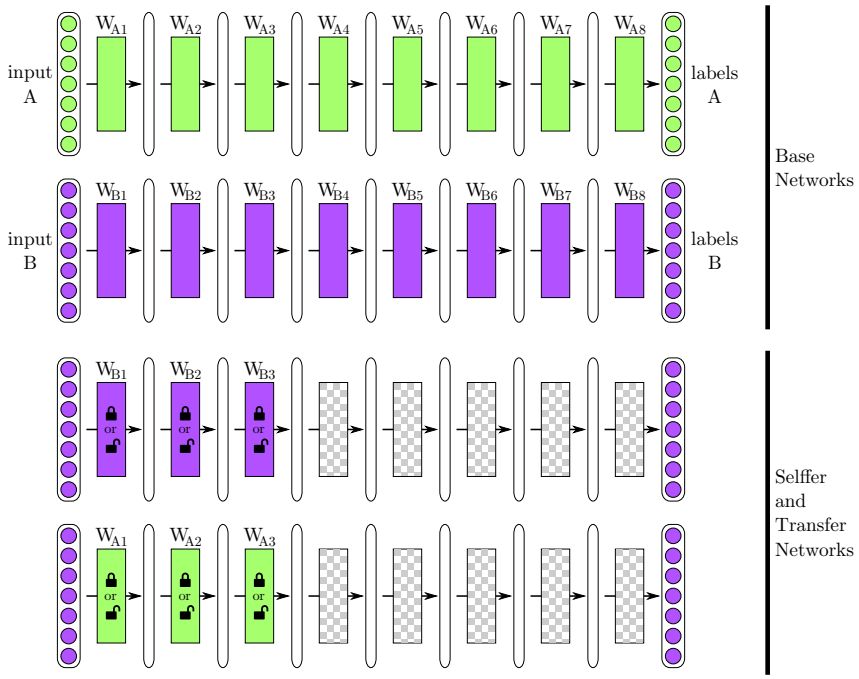

Figure 1: Overview of the experimental treatments and controls. *Top two rows:* The base convnets are trained using standard supervised backprop on only half of the dataset (A or B). The baseline control is repeated for each half of the dataset (A and B). The labeled rectangles (e.g. W_{A1}) represent the weight vector for that layer, with the color indicating which dataset the layer was originally trained toward. The vertical, ellipsoidal bars represent neural activations that occur when the network is activated. *Third row:* The "selfer network" control. In this control the first N weight layers of the network are copied from a base network (e.g. one trained on dataset B) and then the entire network is trained on that same dataset (in this example, dataset B). These layers are either locked during training (in the "locked selfer treatments") or allowed to train as usual (in the "fine-tuned selfer treatments"). The weights of layers after the first N layers are initialized randomly and then allowed to train as usual. This treatment exposes co-adaptation that can only be discovered when filters at different layers are trained together. *Fourth row:* The "transfer network" experimental treatment. Same as the selfer network treatment, except that the first N layers are copied from one half of the dataset (e.g. A) and then the entire network is trained on the *other* half of the dataset (e.g. dataset B). This treatment tests transfer learning.

for the base case and estimating at which points in each region of constant learning rate the learning had plateaued and thus annealing could take place. This faster training schedule was only used for the natural vs. man-made experiments in Section 4.2.

Our base model attains a final top-1 error on the validation set of 42.5%, about the same as the 42.9% reported by Donahue *et al.* (2013) and 1.8% worse than Krizhevsky *et al.* (2012), the latter difference probably due to the few minor training differences. We checked these values only to demonstrate that the network was converging reasonably. As our goal is not to improve the state of the art, but to investigate properties of transfer, we were content with this level of performance. Because code is often more clear than text, we've also made all code and parameter files necessary to reproduce these experiments available on `http://github.com/...removed_for_review...`.

4 Transfer Results and Discussion

We performed two sets of experiments. The main experiment using random A/B splits is discussed in Section 4.1, and the two comparison cases using random filters and the Natural/Man-made split is presented in Section 4.2.

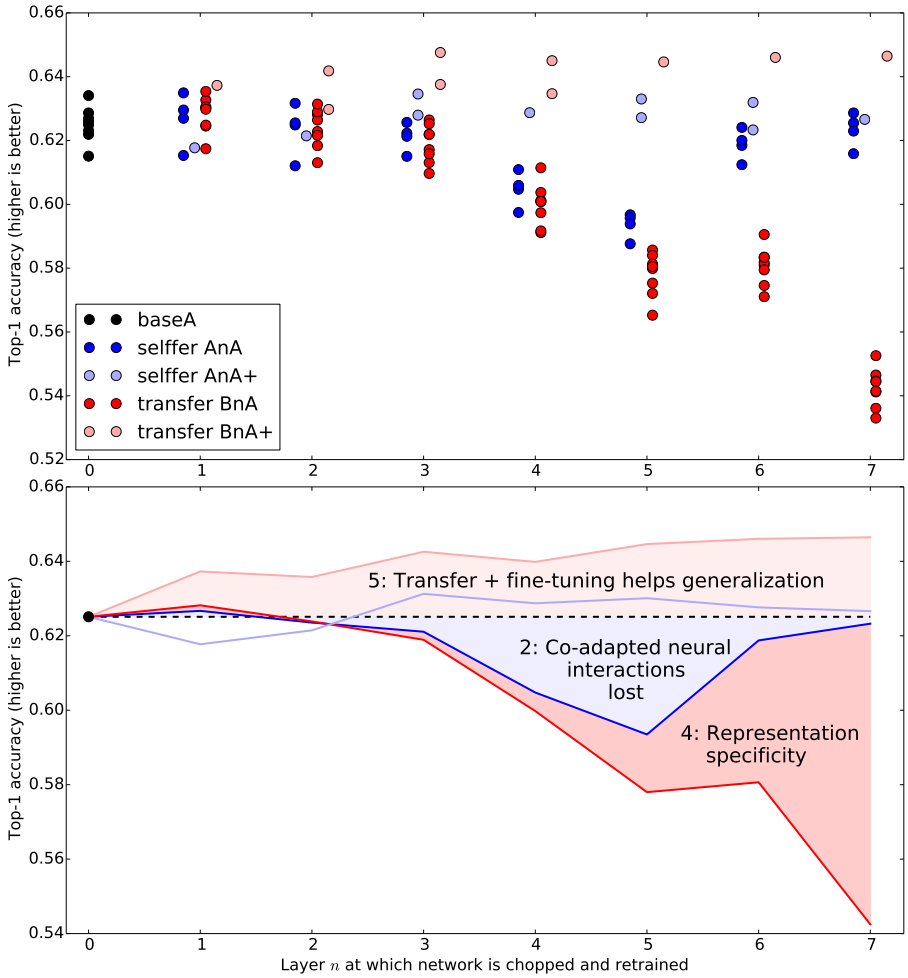

Figure 2: The distilled results from the main experiment in this paper, see the text for detail. *Top panel:* Raw data. *Bottom panel*: Lines connecting the means of each treatment. Shaded areas and numbers refer to which interpretation of Section 4.1 applies.

4.1 Random A/B splits

The results of all transfer learning experiments are shown in condensed form in Figure 2. Each dot in the figure represents the average accuracy over the validation set for an entire trained network. The black dots above $n = 0$ represent the accuracy of baseA or baseB. There are eight points, because we made four separate random A/B splits. Each dark blue dot represents an AnA network[2], that is, a network that was trained on dataset A, chopped off at layer n, and then whose upper layers were reinitialized and retrained toward dataset A while the lower layers remained frozen. The light blue points represent AnA$^+$ networks, the same as AnA but where the lower layers were fine-tuned as well. Each dark red dot represents a BnA network, one whose first n layers are copied from baseB and frozen, and then whose upper layers were randomly initialized and trained toward dataset A. The light red points represent BnA$^+$ networks where all layers were fine-tuned.

In the top subplot plot we've shifted each type of point slightly to the left or write simply for visual clarity; if not for this shift they would all be shown directly above $n = 1$, $n = 2$, etc. and would

[2]Actually these dots represent AnA nets or BnB nets, the statistics of which are equivalent, because in both cases the net is trained on the same random 500 classes. To simplify notation, we'll just calling these AnA networks. Similarly, we've aggregated the BnA and AnB networks and just call them BnA

overlap. In the bottom subplot we've average the results at each layer to show more clearly the overall trend and to highlight the three important effects.

It turns out we can conclude quite a lot from these results! In each of the following interpretations, we compare the performance to the base case (black dots).

1. The black baseA points show that the mean performance of a network simply trained to classify a random subset of 500 classes attains a top-1 accuracy of 0.625, or 37.5% top-1 error. This is different than the 42.5% top-1 error attained on the 1000 class network for a combination of two reasons: it is pushed higher because the network is trained only on half of the data, which could lead to more overfit, and it is pushed lower because there are only 500 classes, so there are only half as many ways to make mistakes. It is this latter effect that dominates, leading to lower error overall.
2. The dark blue AnA points show a curious behavior. We can see at layer one that the performance is the same (as the baseA points), which was expected. If we learn eight layers of features, save the first layer of learned Gabor features, and reinitialize the whole network and retrain it toward the same task, we'd expect to do just as well. This is also true for layer 2. Layers 3, 4, 5, and 6, particularly 4 and 5, behave somewhat differently. When we keep the first, say, 5 layers and retrain layers 6-8, performance suffers. This is evidence that the original network contained *co-adapted features* on successive layers, that is features that interact with each other in a complex way such that this co-adaptation *could not be relearned* by the upper layers. In other words, gradient descent was able to find a good solution the first time, but this was only possible because the layers were jointly trained. As we continue to the right, we see that layer 6 is nearly back to the base level of performance, as is layer 7. As we get closer and closer to the final 500-way softmax output layer (8), there is less to relearn, and apparently relearning these one or two layers is simple enough for gradient descent to find a good solution. Alternately, we may say that there is less co-adaptation of features between layers 6 and 7 or between 7 and 8.

   We are not sure whether or not this effect has been previously observed in the literature. Certainly one would suspect that optimization difficulties might occur, but we do not believe it has been previously shown that such difficulties are worse in the middle of a network than near the bottom or top.
3. The light blue AnA$^+$ points show that, as expected, when we allow the copied lower layer features to learn as well, we're able to find a solution just as good as the original solution. The whole network is trained, just initialized from a partial previous solution instead of completely randomly, which neither helps nor hurts performance.
4. The dark red BnA points show the effect we set out to measure in the first place: the transferability of features from one network to another at each layer. Layers one and two transfer almost perfectly from B to A, giving evidence that at least for our two tasks, not only are the Gabor features general, but the second layer features are as well. Layer three shows a slight drop, and layers 4-7 show a more significant drop in performance. Thanks to the ANA points, we can tell that this drop is from a combination of two separate effects: the drop from lost co-adaptation *and* the drop from features which are less and less general. On layers 3, 4, and 5, the first effect dominates, whereas on layers 6 and 7 the first effect diminishes and the specificity of representation dominates the drop in performance.

   Although examples of successful feature transfer have been reported elsewhere in the literature, to our knowledge these results have been limited to noticing that transfer from a given layer is much better than chance, e.g. noticing that the BnA points are much better than chance level performance (.002 for 500 classes). We believe this is the first time that the extent to which transfer is successful has been carefully quantified layer by layer and that the two separate effects have been decoupled, showing that each effect dominates in part of the regime.
5. The light red BnA$^+$ points show perhaps the most surprising effect: that transferred features followed by fine-tuning result in networks that generalize better than those trained directly on the target dataset. We suspected that this effect might be simply due to longer total training time (450k base + 450k fine-tuned for BnA$^+$ vs. 450k for baseA), but this cannot be the case, because the AnA$^+$ networks are also trained for the same longer length of time and do not exhibit this boost in performance.

Table 1: Performance accuracy boost of transfer + fine-tuning BnA$^+$ over controls. All differences are significant at the $p < .001$ level.

layers aggregated	mean boost over baseA	mean boost over selffer AnA$^+$
1-7	1.6%	1.4%
3-7	1.8%	1.4%
5-7	2.1%	1.7%

Thus, the most likely explanation is that even after 450k iterations of fine tuning (beginning with completely random top layers), the effects of having seen the alternate dataset still linger, boosting generalization performance. We found surprising the fact that this effect can linger through so much retraining. Further, this lingering boon to generalization seems not to depend very much on how much of the first network we keep to initialize the second network: keeping anywhere from one to seven layers produces improved performance, with slightly better performance as we keep more layers. The average boost across layers 1 to 7 is 1.6% over the base case, and the average if we keep at least five layers is 2.1%.[3] The degree of performance boost is shown in Table 1.

4.2 Random filters and Natural/Man-made split

In this section we compare the transfer performance of our random A/B splits to a control — completely random filters — and to our manually created 449 vs. 551 class natural/man-made split.

We make the first comparison because Jarrett *et al.* (2009) showed, — quite strikingly — that the combination of random convolutional features, rectification, pooling, and local normalization can work almost as well as learned features. They showed this on relatively small networks of two or three learned layers and on a smaller dataset; and it is natural to ask whether or not the nearly optimal performance of random filters carries over to a deeper network trained on a larger dataset.

We make the second comparison to see how features transfer between two tasks that are as far apart as possible using the ImageNet dataset. Many are using and will use features trained from ImageNet for completely new tasks, and this second control gives some idea of how well features from various layers will transfer.

Figure 3 shows the results of these two additional experiments. The upper-left subplot shows the accuracy obtained when using random filters for the first n layers, for various choices of n. Performance falls off fairly quickly in layers 1 and 2, which suggests that getting random features to work in general convnets may not be as straightforward as it was for the smaller network size and smaller dataset used by Jarrett *et al.* (2009). However, the comparison is a bit tricky, because whereas we use max pooling and local normalization on layers 1 and 2 just as they did, we use a different nonlinearity: $\mathrm{relu}(x)$ instead of $\mathrm{abs}(\tanh(x))$. The difference in performance could be due to a few factors: slightly different nonlinearity, the dataset itself, sizes of critical pieces of the network, etc. Their experiment only considered two layers of random features. In our case and with our chosen hyperparameters, we see that by the third layer of random features, accuracy has dropped nearly to the chance level. Of course, this represents only one datapoint, and it may well be possible to tweak layer sizes and random initialization magnitudes to enable much better performance.

The upper-right subplot of Figure 3 shows the accuracy of the baseA (black circle), baseB (black square), BnA (magenta circles), and AnB (magenta squares) networks. The baseA and BnA networks are both eventually trained toward A, which is the natural half of the split and has higher

[3] It is somewhat strange for us to aggregate the performance over several layers. Indeed, we'd prefer to average many runs at the same layer, but each point is rather computationally expensive to obtain so we have very few as of the time of this submission. However, we think some aggregation is informative, because the performance at each layer is based on completely different random draws of initialization weights. Thus, the fact that layers 5, 6, and 7 result in almost identical performance across random draws is at least somewhat indicative that multiple runs at a given layer would result in similar performance.

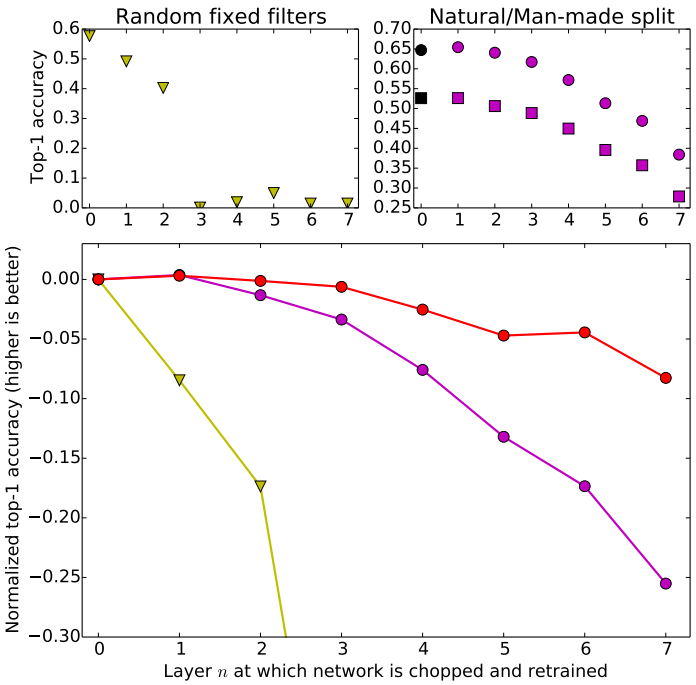

Figure 3: Performance degredation vs. layer; see text for details. (top left): random fixed features. (top right): a manual split of ImageNet into natural vs. man-made groups. (bottom): the top two plots normalized by their base performance and compared to the A/B split from Section 4.1 (in red).

average performance. This may be due to having only 449 classes instead of 551, or simply being an easier task, or both.

Finally, in the bottom plot of Figure 3 we normalize these several experiments by subtracting the performance of their individual base cases (which vary from each other for reasons mentioned above). We then plot the relative drop in performance as we use more random layers or pull more layers from a network trained on another dataset. This comparison makes two things apparent: first, that the transferability gap between two randomly selected A and B halves grows more slowly with increasing layer than does the transferability gap between two less related halves, and second, that for this network architecture, transferring even from a distant task is better than just using random filters. This latter result may hold because the networks trained in (Jarrett *et al.*, 2009) could have been well into the overfitting regime on a dataset as small as Caltech-101 (Fei-Fei *et al.*, 2004).

5 Conclusions

In this paper we've demonstrated a method for quantifying the ability to transfer features from a deep net layer by layer. We showed how transferability is negatively affected by two distinct issues — optimization difficulties related to splitting networks in the middle of co-adapted layers and the specialization of higher layer neurons to their original task at the expense of the target task. In a large convolutional network similar to those that represent the state of the art on the large ImageNet dataset, we observed that either of these two issues may dominate, depending on whether features are transferred from the bottom, middle, or top of this deep network. We've quantified how the transferability gap grows as the distance between tasks increases, but how even features transferred from distant tasks can be better than random features. Finally, we've observed that initializing a network with transferred features from almost any number of layers can produce a boost to generalization that lingers even after a lengthy training procedure.

References

Bengio, Y. (2011). Deep learning of representations for unsupervised and transfer learning. In *JMLR W&CP: Proc. Unsupervised and Transfer Learning*.

Bengio, Y., Bastien, F., Bergeron, A., Boulanger-Lewandowski, N., Breuel, T., Chherawala, Y., Cisse, M., Côté, M., Erhan, D., Eustache, J., Glorot, X., Muller, X., Pannetier Lebeuf, S., Pascanu, R., Rifai, S., Savard, F., and Sicard, G. (2011). Deep learners benefit more from out-of-distribution examples. In *JMLR W&CP: Proc. AISTATS'2011*.

Caruana, R. (1995). Learning many related tasks at the same time with backpropagation. pages 657–664, Cambridge, MA. MIT Press.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. Technical report, arXiv preprint arXiv:1310.1531.

Fei-Fei, L., Fergus, R., and Perona, P. (2004). Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. In *Conference on Computer Vision and Pattern Recognition Workshop (CVPR 2004)*, page 178.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pages 2146–2153. IEEE.

Jia, Y. (2013). Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Le, Q. V., Karpenko, A., Ngiam, J., and Ng, A. Y. (2011). ICA with reconstruction cost for efficient overcomplete feature learning. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1017–1025.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks. Technical Report Arxiv 1311.2901.