an interesting/influential/important paper from the world of CS
every weekday morning, as selected by Adrian Colyer

Understanding, generalisation, and transfer learning in
deep neural networks

FEBRUARY 27, 2017
This is the first in a series of posts looking at the ‘top 100 awesome deep learning papers (https://github.com/terryum/awesome-deep-learning-papers).’ Deviating from the normal one-paper-per-day format, I’ll take the papers mostly in their groupings as found in the list (with some subdivision, plus a few extras thrown in) – thus we’ll be looking at multiple papers each day. The papers in today’s selection all shed light on what it is that DNNs (mostly CNNs) are really learning when trained. Since one way of understanding what a DNN has truly learned is to see how well the trained networks (or subsets of them) can perform on new tasks, we’ll also learn a lot about generalization, and what we learn can help us to define better models that take advantage of transfer learning.

The six papers we’ll look at today are:


I’ve done my best to distill the knowledge in these papers too, but inevitably this post is going to be a little longer than my normal target length! You might need one-and-a-half cups of coffee for this one ;).

Visualising and understanding convolutional networks

*Convolutional neural networks (convnets) have demonstrated excellent performance at tasks such as hand-written digit classification and face detection… Despite this encouraging progress, there is still little insight into the internal operation and behavior of these complex models, or how they achieve such good performance. From a scientific standpoint, this is deeply unsatisfactory.*

If we’re going to understand what a convnet is doing, we need some way to map the feature activity in intermediate layers
back into the input pixel space (we’re working with ConvNets trained on the ImageNet dataset here). Zeiler and Fergus use a clever construction that they call a **deconvnet** that uses the same components as the convnet to be decoded, but in reverse order. Some of the convnet components need to be augmented slightly to capture additional information that helps in the reversing process. (It’s a little reminiscent of the data flow provenance work ([https://blog.acolyer.org/2017/02/01/explaining-outputs-in-modern-data-analytics/](https://blog.acolyer.org/2017/02/01/explaining-outputs-in-modern-data-analytics/)) that we looked at earlier this year.)

Here’s an example with the standard convnet on the **right-hand side**, and the deconvnet layers added on the **left-hand side**.

**Figure 1.** Top: A deconvnet layer (left) attached to a convnet layer (right). The deconvnet will reconstruct an approximate version of the convnet features from the layer beneath. Bottom: An illustration of the unpooling operation in the deconvnet, using **switches** which record the location of the local max in each pooling region (colored zones) during pooling in the convnet.

Now that we can project from any layer back onto pixels, we can get a peek into what they seem to be learning. This leads to now-familiar pictures such as this:
Note in the above how layer 2 responds to corners and edge/colour combinations, layer 3 seems to capture similar textures, layer 4 is more class-specific (e.g. dog faces), and layer 5 shows entire objects with significant pose variation. Using these visualisations and looking at how they change over time during training it is also possible to see lower layers converging within relatively few epochs, whereas upper layers take considerably longer to converge. Small transformations in the input image have a big effect on lower layers, but lesser impact in higher layers.

The understanding gleaned from inspecting these visualisations proved to be a helpful tool for improving the underlying models too. For example, a 2nd layer visualization showed aliasing artefacts caused by a large stride size, reducing the stride size gave an increase in classification performance.

Experiments with model structure showed that having a minimum depth to the network, rather than any one specific section in the overall model, is vital to model performance.
DeCAF: A deep convolutional activation feature for generic visual recognition

Many visual recognition challenges have data sets with comparatively few examples. In DeCAF, the authors explore whether a convolutional network trained on ImageNet (a large dataset) can be generalised to other tasks where less data is available:

> Our model can either be considered as a deep architecture for transfer learning based on a supervised pre-training phase, or simply as a new visual feature DeCAF defined by the convolutional network weights learned on a set of pre-defined object recognition tasks.

After training deep a convolutional model (using Krizhevsky et al.'s competition winning 2012 architecture), features are extracted from the resulting model and used as inputs to generic vision tasks. Success in those tasks would indicate that the convolutional network is learning generically useful features of images (in much the same way that word embeddings learn features of words (https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/)).

Let DeCAF_{N} be the activations of the nth hidden layer of the CNN. DeCAF_{7} is the final hidden layer just before propagating through the last fully connected layer to produce class predictions. All of the weights from the CNN up to the layer under test are frozen, and either a logistic regression or support vector machine is trained using the CNN features as input.

On the Caltech-101 dataset (https://en.m.wikipedia.org/wiki/Caltech_101) the DeCAF_{6} + SVM outperformed the previous best state of the art (a method with a combination of five traditional hand-engineered image features)!

The Office dataset contains product images from amazon.com, as well as images taken in office environments using webcams and DSLRs. DeCAF features were shown to be robust to resolution changes (webcam vs DSLR), providing not only better within category clustering, but also was able to cluster same category instances across domains. DeCAF + SVM again dramatically outperformed the baseline SURF features available with the Office dataset.

For sub-category recognition (e.g. distinguishing between lots of different bird types in the Caltech-UCSD birds dataset) DeCAF_{6} with simple logistic regression again obtained a significant increase over existing approaches: “To the best of our knowledge, this is the best accuracy reported so far in the literature.”

And finally, for scene recognition tasks, DeCAF + logistic regression on the SUN-397 large-scale scene recognition database also outperformed the current state-of-the-art.

Convolution neural networks trained on large image sets were therefore forcefully demonstrated to learn features with sufficient representational power and generalization ability to perform at state-of-the-art levels on a wide variety of image-based tasks. It’s the beginning of the end of hand-engineered features, and welcome to the era of deep-learned features.

> The ability of a visual recognition system to achieve high classification accuracy on tasks with sparse labeled data has proven to be an elusive goal in computer vision research, but our multi-task deep learning framework and fast open-source implementation are significant steps in this direction.

CNN features off-the-shelf: an astounding baseline for recognition

CNN features off the shelf further reinforces that we can learn general features useful for image-based tasks, and apply them very successfully in new domains. This time the baseline features are taken from a trained convolutional neural network model called Overfeat, which has been optimized for object image classification in ILSVRC. Then for a variety of tasks, instead of using state-of-art image processing pipelines, the authors simply take the features from the CNN

https://blog.acolyer.org/2017/02/27/understanding-generalisati...
representation, and bolt on an SVM. Sounds familiar?

![Diagram of CNN representation process](https://blog.acolyer.org/2017/02/27/understanding-generalisati...)

The tasks undertaken progress from quite close to the original classification task, to more and more demanding (i.e. distant tasks). At every step of the way, the CNN features prove their worth!

**Step 1: Object and scene recognition**

The Pascal VOC 2007 dataset has ~10,000 images of 20 classes of animals, and is considered more challenging than ILSVRC. Applying the Overfeat CNN features to this dataset resulted in a model outperforming all previous efforts “by a significant margin.” The following chart shows how classification performance improves depending on the level from the original CNN that is chosen as the input to the final SVM:

**Figure 1:** *top* CNN representation replaces pipelines of s.o.a methods and achieve better results. e.g. DPD [50].

**bottom** Augmented CNN representation with linear SVM consistently outperforms s.o.a. on multiple tasks. Specialized CNN refers to other works which specifically designed the CNN for their task.

The tasks undertaken progress from quite close to the original classification task, to more and more demanding (i.e. distant tasks). At every step of the way, the CNN features prove their worth!
For scene recognition, the MIT-67 indoor scene dataset has 15,620 images of 67 indoor scene classes. The CNN + SVM model significantly outperformed a majority of the baseline models and just edges a state-of-the-art award by 0.1% accuracy over the previous best AlexConvNet model (also a CNN).

Step 2: Fine-grained recognition

Here we’re back with birds (Caltech UCSD 200-2011 dataset) and also flowers (Oxford 102 flowers dataset). Can the more generic OverFeat features pick up potentially subtle differences between the very similar classes? On the birds dataset the model gets very close to the state of the art (also a CNN), and beats all other baselines. On the flowers dataset, the CNN+SVM model outperforms the previous state-of-the-art.

Step 3: Attribute detection

Have the OverFeat features encoded something about the semantic properties of people and objects? The H3D dataset defines 9 attributes for person images (for example, ‘has glasses,’ and ‘is male.’). The UIUC 64 dataset has attributes for objects (e.g., ‘is 2D boxy’, ‘has head’, ‘is furry’). The CNN-SVM achieves state of the art on UIUC 64, and beat several existing models on H3D.

Step 4: Instance retrieval

What about trying the CNN-SVM model on instance retrieval problems? This is a domain where the state-of-the-art using highly optimized engineered vectors and mid-level features. Against methods that do not incorporate 3D geometric constraints (which do better), the CNN features proved very competitive on building and holiday datasets.

What have we learned?

*It's all about the features! SIFT and HOG descriptors produced big performance gains a decade ago and now deep convolutional features are providing a similar breakthrough for recognition. Thus, applying the well-established computer vision procedures on CNN representations should potentially push the reported results even further. In any case, if you develop any new algorithm for a recognition task, it must be compared against the strong baseline of generic deep features + simple classifier.*
How transferable are features in deep neural networks?

The previous papers mostly focused on taking the higher layers from the pre-trained CNNs as input features. In ‘How transferable are features in deep neural networks’ the authors systematically explore the generality of the features learned at each layer – and as we’ve seen, to the extent that features at a given layer are general, we’ll be able to use them for transfer learning.

The usual transfer learning approach is to train a base network and then copy its first $n$ layers to the the first $n$ layers of a target network. The remaining layers of the target network are then randomly initialized and trained toward the target task. One can choose to back-propagate the errors from the new task into the base (copied) features to fine-tune them to the new task, or the transferred feature layers can be left frozen...

The experiment setup is really neat. Take an 8-layer CNN model, and split the 1000 ImageNet classes into two groups (so that each contains approximately half the data or 645,000 examples). Train one instance of the model on half A, and call it baseA. Train another instance of the model on half B, and call it baseB. Starting with baseA, we can define seven starter networks, A1 through A7, that copy the first 1 through 7 layers from baseA respectively (and of course we can do the same from baseB to give B1 through B7). Say we’re interested in exploring how well features learned at layer 3 transfer. We can construct the following four networks:

- B3B – the first 3 layers are copied from baseB and frozen. The remaining five higher layers are initialized randomly and we train on task B as a control. (The authors call this a ‘selfer’ network)
- A3B – the first 3 layers are copied from baseA and frozen. The remaining five layers are initialized randomly as before, and trained on task B. If A3B performs as well as B3B, we have evidence that the first three layers are general.
- B3B+ like B3B but the first three layers are subsequently fine-tuned during training.
- A3B+ like A3B but the first three layers are subsequently fine-tuned during training.
Repeat this process for all layers 1..7. Running these experiments leads to the following results:

Looking at the dark blue dots first (BnB), we see an interesting phenomenon. When freezing early layers and then retraining the later layers towards the same task, the resulting performance is very close to baseB. But layers 3, 4, 5, and 6 (especially 4 and 5) show significantly worse performance:

*This performance drop is evidence that the original network contained fragile co-adapted features on successive layers.*

As we get closer to the final layers, performance is restored as it seems there is less to learn… “To our knowledge it has not been previously observed in the literature that such optimization difficulties may be worse in the middle of a network than near the bottom or top.”

Note that the light blue dots (BnB+), where we allow fine-tuning, restore full performance.

The red dots show the transfer learning results. Starting with the frozen base version (dark red, AnB), we see strong transference in layers 1 and 2, and only a slight drop in layer 3, indicating that the learned features are general. Through layers 4-7 though we see a significant drop in performance.
Thanks to the BnB 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 that are less and less general.

Finally let’s look at the light red AnB+ points. These do better than the baseline! Surprised? Even when the dataset is large, transferring features seems to boost generalization performance. Keeping anywhere from one to seven layers seems to infer some benefit (average boost 1.6%) so the effect is seen everywhere. One way that I think about this is that the transferred layers had a chance to learn from different images that the selfer networks never see – thus they have a better chance of learning better generalizations.

The short summary – transfers can improve generalization performance. Two issues impact how well transfer occurs: fragile co-adaptation of middle layers, and specialisation of higher layers.

Learning and transferring mid-level image representations using convolutional neural networks

This paper uses the by-now familiar ‘train a CNN on ImageNet and extract features to transfer to other tasks approach, but also explores training techniques that can help to maximise the transfer benefits. Here’s the setup:

The target task for transfer learning is Pascal VOC object and action classification, “we wish to design a network that will output scores for target categories, or background if none of the categories are present in the image.” Transfer is achieved by removing the last fully-connected layer from the pre-trained network and adding an adaptation layer (fully connected layers FCa and FCb) and train them on the labelled data of the target task.

The source dataset (ImageNet) contains nice images of single centered objects. The target dataset (Pascal VOC) contains complex scenes with multiple target objects at various scales and background clutter:
The distributed of object orientations and sizes as well as, for example, their mutual occlusion patterns is very different between the two tasks. This issue has also been called “a dataset capture bias.” In addition, the target task may contain many other objects not present in the source task training data (“a negative bias”).

Here's the new twist: to address these biases, the authors use a sliding window and extract around 500 square patches from each image by sampling on eight different scales using a regularly spaced grid and 50% or more overlap between neighbouring patches:

Figure 3: Illustration of different dataset statistics between the source (ImageNet) and target (Pascal VOC) tasks. Pascal VOC data displays objects embedded in complex scenes, at various scales (right), and in complex mutual configurations (middle). Left: Image from ImageNet with label **maltese terrier**. Middle and right: Images from Pascal VOC with label **dog**.
To label the patches in the resulting training data, the authors measure the overlap between the bounding box of a patch P and the ground truth bounding boxes of annotated objects in the image. If there is sufficient overlap with a given class, the patch is labelled as a positive training example for the class.

You shouldn’t be surprised at this point to learn that the resulting network achieves state of the art performance on Pascal VOC 2007 object recognition, and gets very close to the state of the art on Pascal VOC 2012. The authors also demonstrate that the network learns about the size and location of target objects within the image. For the Pascal VOC 2012 action recognition task, state of the art results were achieved by allowing fine-tuning of the copied layers during training.

Our work is part of the recent evidence that convolutional neural networks provide means to learn rich mid-level image features transferrable to a variety of visual recognition tasks.

Distilling the knowledge in a neural network

Let’s finish with something a little bit different: what can insects teach us about neural network training and design?
Many insects have a larval form that is optimized for extracting energy and nutrients from the environment and a completely different adult form that is optimized for the very different requirements of traveling and reproduction. In large-scale machine learning, we typically use very similar models for the training stage and the deployment stage despite their very different requirements.

What if we use large cumbersome models during training (e.g., very deep networks, ensembles), so long as those models make it easier to extract structure from the training data, and then find a way to transfer or distill that the training model has learned into a a more compact form suitable for deployment? We want to cram as much of the knowledge as possible from the large model into the smaller one.

If the cumbersome model generalizes well because, for example, it is the average of a large ensemble of different models, a small model trained to generalize in the same way will typically do much better on test data than a small model that is trained in the normal way on the same training set as was used to train the ensemble.

How can we train the small model effectively though? By using the class probabilities produced by the cumbersome model as "soft targets" for training the small model. The large model has learned not just the target prediction class, but a probability distribution over all classes – and the relative probabilities of incorrect answers still contain a lot of valuable information. The essence of the idea is to train the small model to reproduce the probability distribution, not just the target output class.

Neural networks typically produce class probabilities by using a "softmax" output layer that converts the logit, \( z_i \), computed for each class into a probability, \( q_i \), by comparing \( z_i \) with the other logits.

\[
q_i = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}
\]

where \( T \) is a temperature normally set to 1. A higher \( T \) value produces a softer probability distribution over classes. The cumbersome model is trained using a high temperature in its softmax, and the same high temperature is used when training the distilled model. When that model is deployed though, it uses a temperature of 1.

A 'cumbersome' large neural net with two hidden layers of 1200 rectified linear units trained on 60,000 training cases using dropouts to simulate training an ensemble of models sharing weights, achieved only 67 test errors. A smaller model network with 800 units in each layer and no regularization saw 146 test errors. However, a distilled smaller network of the same size trained to match the soft targets from the large network achieved only 74 test errors.

In an Automatic Speech Recognition (ASR) test an ensemble of 10 models was distilled to a single model that performed almost as well. The results compare well to a very strong baseline model similar to that used by Android voice search.

<table>
<thead>
<tr>
<th>System</th>
<th>Test Frame Accuracy</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>58.9%</td>
<td>10.9%</td>
</tr>
<tr>
<td>10xEnsemble</td>
<td>61.1%</td>
<td>10.7%</td>
</tr>
<tr>
<td>Distilled Single model</td>
<td>60.8%</td>
<td>10.7%</td>
</tr>
</tbody>
</table>

Table 1: Frame classification accuracy and WER showing that the distilled single model performs about as well as the averaged predictions of 10 models that were used to create the soft targets.

A very nice use of the technique is in learning specialized models as part of an ensemble. Take a large data set (e.g. Google's JFT image dataset with 100M images) and a large number of labels (15,000 in JFT): it's likely there are several subsets of labels on which a general model gets confused. Using a clustering algorithm to find classes that are often predicted together, an ensemble is created with one generalist model, and many specialist models, one for each of the top \( k \) clusters. The specialist models are trained on data highly enriched in examples from the confusable subsets. The hope is that the resulting knowledge can be distilled back into a single large net, although the authors did not demonstrate that final step in the paper.
We have shown that distilling works very well for transferring knowledge from an ensemble or from a large highly regularized model into a smaller, distilled model. Furthermore, we have shown that the performance of a single really big net that has been trained for a long time can be significantly improved by learning a large number of specialist nets, each of which learns to discriminate between the classes in a highly confusable cluster.

Understanding through counter-examples

Another interesting way of understanding what DNNs have learned, is through the discovery of counter-examples that confuse them. The ‘top 100 awesome deep learning papers (https://github.com/terryum/awesome-deep-learning-papers)’ section on understanding, generalisation, and transfer learning (which we’ve been working through today) contains one paper along those lines. But this post is long enough already, and the subject is sufficiently interesting that I’d like to expand it with a few additional papers as well. So we’ll look at that tomorrow…

6 Comments

Atcold PERMALINK
April 5, 2017 5:35 am
About your conclusions: «The short summary – transfers can improve generalization performance. Two issues impact how well transfer occurs: fragile co-adaptation of middle layers, and specialisation of higher layers.», this is quite not correct. This article was even moiled at NIPS, when it has been presented orally. Turns out that the poor performance of AnB is the direct consequence of the optimisation problem that we try to address. More precisely, if you freeze the base, you’ll end up with far fewer degrees of freedom, hence, your optimisation will suffer. Unlock the base, and magically you’ll have many more parameters for you to play with. It performs better that BnB+ because of lower overfitting due to a larger input data set. So, it was a unfortunate article…

Atcold PERMALINK
April 5, 2017 6:01 am
The aim in «Distilling the knowledge in a neural network» is noble, but it doesn’t work for anything that is not a toy data set. More specifically, MNIST is the only one that showed some decent results…

P.S. It’s considered unacceptable to post arXiv links to PDFs instead of their abstract page 😞

adriancolyer PERMALINK*
April 5, 2017 7:21 am
It’s very helpful getting an “insider” perspective on some of these papers, thank you. And I’ve wondered what the accepted way is to link to arXiv papers is too! (I link to pdf’s for consistency across all of the different paper sources that TMP touches). I’m changing that convention for some of the ACM DL papers, so I guess I should swap now for arXiv too :).

Atcold PERMALINK
April 5, 2017 4:45 pm
Great! This is because, from mobile, someone may just want to view the abstract (the PDF gets downloaded immediately, otherwise) or choose a different version (with the PDF link you get the version that has been linked or the most recent one).

Atcold PERMALINK
April 5, 2017 4:48 pm
Anyhow, thank you for writing these! They help refresh some recent historical milestones in the field. I am even considering to use these for my class. These summaries are beautifully written!
Trackbacks

Unsupervised learning and GANs | the morning paper