Transfer Learning

Girish Shanmugam S
Department of IT
Uppsala University

Jude Felix
Department of IT
Uppsala University

Sharadh Tiwari
Department of IT
Uppsala University

Abstract

While learning, humans usually try to apply and transfer the knowledge gained from previous events which are similar to the current task. Machines could also be developed to learn like humans instead of making them learn from scratch every time using an approach called *transfer learning*. In our work, we try to inspect and explain why and how transfer learning works on images trained on deep neural networks. Understanding the roles and dynamics of each layer in a dense network could help decide whether the layers could actually be transferred and reused in a new similar setup. We conduct a set of experiments to analyze the transferability of features in a deep neural network. The experiments and analysis in this work could be useful in understanding why and how transfer learning works when to use it and how to use it to get the best performance out of it.

1 Introduction

Deep Neural Networks (DNN) are artificial neural networks which have several layers in between the input and output. The DNN architectures such as convolutional neural networks, deep belief networks, and recurrent neural networks are used in fields of computer vision, speech recognition, natural language processing, and bioinformatics where they have produced results comparable to human analysis and in some cases even perform better [15]. The DNNs are often considered to be black box [2]. One of the reasons is that their predictions are often non-traceable for humans. Visualizing the layers in a DNN could be a possible way to understand what exactly each layer represents. The visualization of intermediate layers of a DNN model led to a few interesting findings [3]. The layers that are deep in the network, visualized features specific to the training dataset and earlier layers in the network visualized more general features like edges, texture, or background. This observation could be really helpful as we could reuse these generic features while we train a new DNN instead of relearning from scratch. This is one of the core ideas behind the concept of *transfer learning* in machine learning [9].

Transfer learning is a method in machine learning that retrieves the knowledge gained from one problem and reuses it to solve another related problem. This is similar to human learning behavior. Transfer learning could be really useful in cases where there is not much to learn from the current task or data in hand. This could be due to the limited size of the training as the data could be rare, expensive to collect, and label. In such cases, learning from a related task could be transferred to the current task to get better performance. Transfer learning is being extensively applied in image classification and video classification tasks [12]. It is also applied in natural language processing tasks such as sentiment classification, text classification, spam email detection, and multiple language text classification [9].

The main objective of our work is to understand how and why transfer learning works. Few analyses have been done earlier to understand how to transfer learning works [7] [16]. Though the kind of experiments we perform are similar, we have used a different dataset and a different model architecture. Also, we have used Pytorch [11] for analysis whereas the earlier work is in Caffe [8]. The set of analysis carried out in our work are as follows:

1. Inspect the layers of a CNN model by visualizing the filters and feature maps.
2. Identify whether a particular layer is a generic layer or a specific layer
3. Examine whether the transition from generic to specific layer occurs smoothly over several layers or occurs immediately after a single layer.

The rest of the report is organized as follows. Section 2 explains the basic concepts behind deep neural networks. Our approach for understanding and visualizing the layers in Convolutional Neural Networks are explained in Section 3. An overview and a brief description of how and why transfer learning works is explained in Section 4. In Section 5, we describe the dataset used, set of models used in our experiments, and the technical specifications along with hyperparameters used in our experiments for reproducibility. In Section 6, we summarise and discuss our results. Finally, in Section 7, we conclude and provide details about possible directions for future work.

2 Deep Neural Networks on Images

A Convolutional Neural Network (CNN) is a class of deep neural networks that are suited for data with spatial relationships. CNNs work best with an image as an input because of their matrix or grid-like structure that stores the spatial relationships of the pixels that form an image. A CNN architecture is generally comprised of a Convolution Layer, Pooling Layer, and Fully-connected layer. CNNs differ from classical Multi-Layer Perceptrons because of their capability to preserve the relationship between the pixels of the images. In a Multi-Layer Perceptron, one would need to flatten the image before using it which results in an increased number of trainable parameters (with large image size) and loss in the spatial relationship which negatively affects the accuracy of the model significantly.

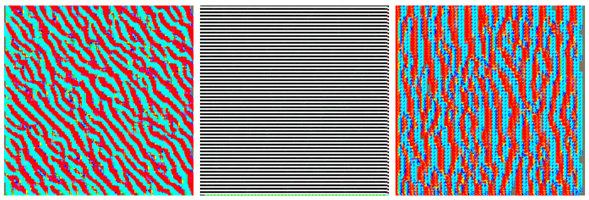
At the core of a CNN architecture there are Convolutional layers. The parameters of a Convolutional layer are composed of a set of learnable filters (kernel) of a fixed size, a skipping factor, and a connection table. Filters are a small matrix that stretches across the whole input and the skipping factor determines the number of pixels the filter/kernel skips in x and y-direction between subsequent convolutions [4]. The filter is slid or convolved across the input volume and a dot product is calculated between the values in the filter and the input values generating a feature map. The output can be controlled by using strides, depth, or padding depending on the kind of analysis being performed. The input to the convolutional layer is either a pixel value or a value from a feature map depending on the position of the layer in the network. A pooling layer between convolutional layers in a CNN architecture is used to gradually reduce the spatial size which reduces the number of parameters and the computation of the network and also controls the model from overfitting.

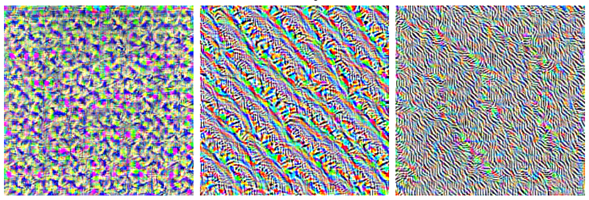
Pooling uses a *max* operation to reduce the number of feature maps. For example, a 2x2 max-pooling will take the highest value from a feature map (eg, grid of 3x3) by reducing the number of values. This is extremely useful for cases where a slight manipulation (rotating, cropping, etc) of an image will result in different pixel values but the image is still the same. Pooling helps in keeping only the significant features, leads to faster convergence and improves the generalization performance[4]. The fully connected (FC) layer has all of its neurons connected to all the activations in the previous layers. An FC layer is generally added at the end of the architecture and has a non-linear or softmax activation function for probability-based output class predictions.

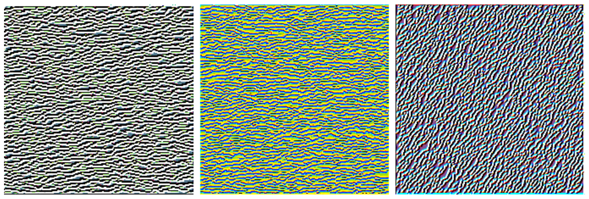
3 Intermediate layers in Convolutional Neural Networks

Understanding the roles and dynamics of each layer of CNN could help observe how CNNs solve the problems and how suitable they are for classification. Popular and efficient models for images such as VGG-16 and Alexnet which are trained on the ImageNet dataset were selected for investigation. The VGG-16 model is a 16 layer model with 13 convolutional filters and 3 fully connected layers [14]. The Alexnet model consists of 5 convolutional layers and 3 fully connected layers. The layers of these models were visualized using Pytorch as shown in Figure 1.

The visualizations of the layers revealed a few interesting results. The layers at the beginning of the model were extracting general features like Gabor filters, edges, textures, patterns as in Figures 1a and 1c and the deeper layers were more specific to the task as in Figures 1b and 1d. This shows that there is a transition between the layers from being general to becoming task-specific. Also, most of the efficient models for classification have generic features as initial layers [3]. Hence this could also be used as criteria to evaluate a newly trained model. These general layers from a well trained efficient model could be reused for similar problems instead of training a model from scratch with random weights.


(a) Filters from the first layer of Alexnet model.


(b) Filters from the last layer of Alexnet model.


(c) Filters from the first layer of VGG-16 model.

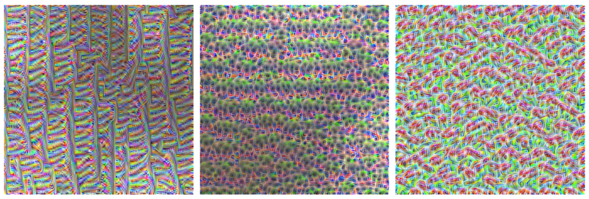

(d) Filters from the last layer of VGG-16 model.

Figure 1: Filters of different layers from Alexnet and VGG-16 model.

4 Transfer Learning

Humans usually apply the learning from a particular situation to another situation in a different context. It is much easier to adapt and cross utilize our knowledge if thew tasks are related to each other. A person with an extensive music background will be able to learn the piano more easily using previous knowledge in music. This idea of transferring the knowledge from one problem to another problem is called transfer learning. Similarly, we could perform transfer learning in machine learning as well and try to evolve machine learning as human learning as possible. We could reuse the initial layers of an already trained model that knows how to extract features. Only the latter layers need to be tuned to the specific task. The initial layers extract the features and the latter layers use the extracted features to classify. A general transfer learning approach is explained in the Algorithm 1. We first train a base network on a base dataset and task A, and then we re-purpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task B [7]. After transferring the features from base task, the copied features could either be *frozen* or *finetuned*.

Freezing: The transferred weights are left unaltered and only the rest of the layers are trained. The transferred weights are frozen if the network is dense and the target dataset is small as it might lead to overfitting.

Finetuning: The transferred weights are also trained along with the other weights. The weights are finetuned if the network is sparse and the target dataset is huge.

Algorithm 1: General Transfer Learning Approach

```
1 Train a base network;
2 Copy its first n layers to first n layers of target network;
3 Remaining layer of target network are randomly initialized and trained for specific task;
4 The copied features can be either:;
5 Frozen and left unchanged during training;
6 Fine tuned by back propagating the errors specific to the task;
```

The main advantage of using transfer learning is that it gives competitive performance even with a small training set. Even for large training sets, it gives the state of the art results compared to the models trained from scratch [9]. It reduces the training time as it converges faster due to pre-trained weights and not random initialization of weights. Overfitting could be avoided when a dense network is being trained on a small dataset when transfer learning is used [6]. For transfer learning to work few conditions need to be satisfied: (i) the generic features should be relevant to both base task and target task and not just specific to the base task (ii) Dataset used in source task is a good enough representation (iii) Source task has more data than the target task.

5 Experiments

Our experiments are almost similar to the kind of analysis carried out on the transferability of features in a deep neural network [7]. We perform a similar set of experiments and examine whether it works on a different dataset and architecture.

5.1 Dataset

Transfer Learning works only if the base task and target tasks are similar. There are also a few cases where transfer works better than random weights initialization even if the scenarios are dissimilar to each other [13]. We do not investigate this scenario in our work and consider two tasks that are similar to each other.

Dataset A: This should have considerably large data and good representation as this is going to be our base task from which the learning is going to be transferred. Hence, the ImageNet dataset [5] which has around 1000 classes and 1.2 million images is used as the dataset for Task A.

Dataset B: Cats Vs Dogs dataset from Kaggle was used as ImageNet dataset also has classes of cats and dog families. This would make the dataset similar to dataset A but with few images when compared to a huge ImageNet dataset. This dataset has 25,000 images of dogs and cats.

5.2 Model Description

The model we used for the experiments had the same architecture as the AlexNet model [10]. Since we wanted to perform experiments by transferring weights from each convolutional layer in the model, we wanted to choose a model with comparatively less number of convolutional layers but with a reasonable performance. AlexNet model has five convolutional layers and the entire architecture is shown in Figure 2 illustrated in AlexNet work [10].

Two models ***modelA*** and ***modelB*** based on AlexNet architecture were trained on datasets A and B respectively. Since it is sensible to transfer only the generic layers of a base model to the target model, it is essential to know whether a layer is generic or to know the degree to which a layer could be generic or specific. Also, it would be helpful if we could find what happens when we transfer from different parts of the model(top, middle, or bottom part of the network). Based on this, we decided to develop four models to be used in our experiments.

- ***nBToB_fr:*** *n* layers from *modelB* are copied while the rest of the layers are randomly initialised. The weights of the copied layers are frozen. The model is trained on dataset B.

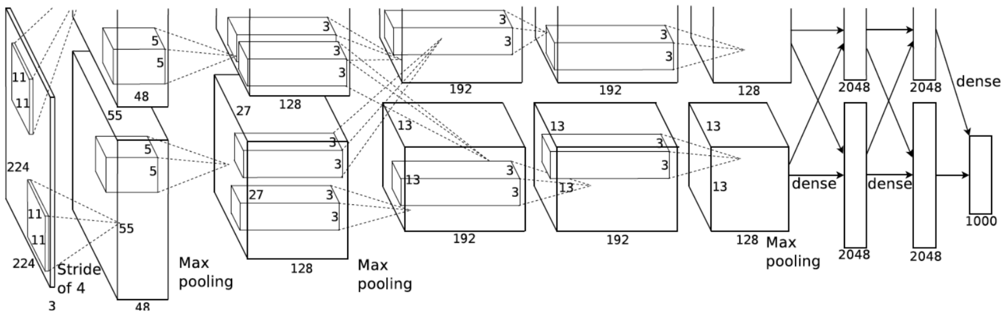


Figure 2: Model Architecture

- ***nBToB_ft:*** *n* layers from *modelB* are copied while the rest of the layers are randomly initialised. All the weights are trained on dataset B.
- ***nAToB_fr:*** *n* layers from *modelA* are copied while the rest of the layers are randomly initialised. The weights of the copied layers are frozen. The model is trained on dataset B.
- ***nAToB_ft:*** *n* layers from *modelA* are copied while the rest of the layers are randomly initialised. All the weights are trained on dataset B.

5.3 Training Setup

The list of hyperparameters used in our experiments is listed in Table 1 to facilitate the reproducibility of our work. We tried several hyperparameters and chose the combination that provided the best results. Our experiments were performed on GPU provided by Google Colab. Pytorch was used for implementation and our code to reproduce these experiments are available at `https://github.com/GirishShanmugam/transfer-learning`

Optimiser	Adam
Loss	CrossEntropyLoss
Batch Size	64
Learning Rate	0.0001
Epochs	10

Table 1: Model Specifications

6 Results and Discussion

AlexNet architecture has five convolutional layers. The value of *n* could be {1,...,5}. Every model has to be trained and evaluated for every single value of *n*. Hence, we performed 20 experiments in total and analyzed the results when a different number of specific layers are replaced. Due to time constraints, we performed each experiment twice and reported average precision values. Due to the stochasticity in initializing the weights of the models, ideally, the experiments should be conducted several times and the average should be taken into consideration for analysis. The results of our experiments are reported in Table 2 and in Figure 3. Some of our observations and interpretations from the results are summarised as follows:

1. *The features of layers 4 and 5 are related to each other in modelB*. This can be seen by observing the performance of model *nBToB_fr* where there is a drop in performance when we transfer and freeze the weights of the first 4 layers indicated by *nBToB_fr* (blue line) in Figure 3. By doing this, we break the relationship between layers 4 and 5, and the final layer of the new model is not able to relearn the relationship even when we try to finetune the weights trained on dataset B. Hence it is not ideal to transfer 4 layers from the model. The rest of the layers are less co-adapted with each other as we do not see any degradation in performance.

2. *Fine tuning overcomes co-adaptation.* We do not observe any performance drop when we transfer 4 layers and fine-tune all the weights indicated by *nBToB_ft* (red line) in Figure 3. This shows that finetuning has helped to solve this problem of co-adaptation, as all the layers are fine-tuned.

3. *First few layers have generic features.* Transferring and freezing the pre trained weights of first few layers trained on dataset A as indicated by *nAToB_fr* (green line) in Figure 3 gives better performance than the weights trained from scratch as indicated by *nBToB_fr* (blue line) and *nBToB_ft* (red lines) in Figure 3. This shows that the first few layers are generic and relevant to both the tasks.

4. *Transferring and fine-tuning gives better performance* The model *nAToB_ft* (brown line) indicated in the Figure 3 gives the best results irrespective of the number of layers transferred with an exception of three layers. This shows that we get better results when we transfer from a good source task and fine-tune the features in all layers based on the target task.

5. *Transferring as many layers as possible and fine-tuning gives the best performance.* We get a top accuracy of *96.16%* when we transfer 5 layers from source task and fine-tuning them all to adapt to the target task. The performance of the model *nAToB_ft* seems to be directly proportional to the number of layers transferred.

6. *Transfer learning needs a considerable amount of fine-tuning to surpass freezing.* On comparing the models with transfer learning *nAToB_fr* and *nAToB_ft*, the model with the frozen weights performs almost as good as the finetuned model and at times even better than the finetuned model (for layer 3). This is could because we only fine-tune the model for 10 epochs and if we might have run the experiments with more epochs [7], then the model *nAToB_ft* might have clearly performed better than *nAToB_fr* irrespective of the number of layers transferred. Also fine tuning with too much epochs might lead to over-fitting as mentioned in Section 4.

	1	2	3	4	5
nBToB_fr	88.26	89.86	90.02	88.08	88.94
nBToB_ft	90.38	90.54	90.46	91.68	92.12
nAToB_fr	91.2	94.44	95.84	95.42	95.44
nAToB_ft	92.76	94.5	95.36	95.68	96.16

Table 2: Accuracy of models for all values of *n*

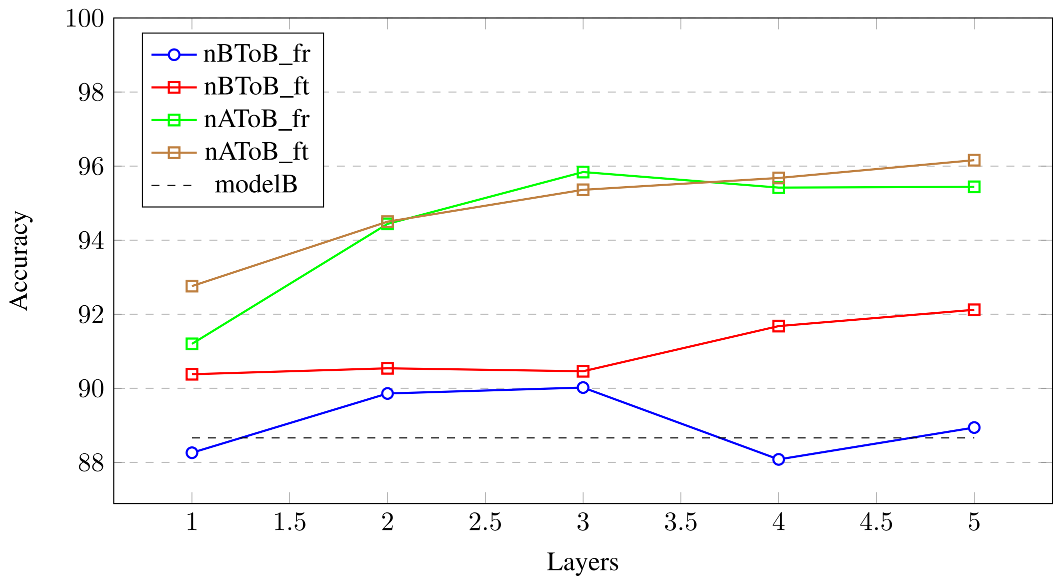


Figure 3: Analysis of transferability of features in similar datasets

7 Conclusion and Future Works

We inspected the layers of a CNN model by visualizing them to understand the roles and dynamics and observed that first layer features are generic and the final few layers are specific to the task. This forms the basis for understanding why and how transfer learning works. We carried out a set of experiments to analyze the transferability of features between neural networks for image datasets. The experiments help to understand how and what to transfer between neural networks and also show that transfer learning along with fine-tuning achieves better results in our case.

Exploring whether the target Kaggle dataset that we used overlaps with the source ImageNet dataset and examining whether it would have an impact on performance is an analysis that we plan to do in the future. We also plan to use a more deeper model (with more convolutional layers), a large target dataset, and to investigate how transfer learning performs for the case where the source and target tasks are dissimilar [1]. In addition to this, we also plan to study how transfer learning could be applied in a different context like textual data.

Acknowledgments

We would like to show our gratitude to Nadezhda Koriakina for sharing her knowledge with us during the discussions and her constant support during the course of this work.

References

[1] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From generic to specific deep representations for visual recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 36–45, 2015.

[2] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey. *ArXiv*, abs/1911.12116, 2019.

[3] François Chollet. *Deep Learning with Python*. Manning Publications, 2017.

[4] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[7] Yoshua Bengio Jason Yosinski, Jeff Clune and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[9] Taghi M. Khoshgoftaar Karl Weiss and DingDing Wang. A survey of transfer learning. *Journal of Big Data volume*, 3(9), 2016.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[12] L. Shao, F. Zhu, and X. Li. Transfer learning for visual categorization: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1019–1034, 2015.

[13] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan 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.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] Xiaohui Liu Nianyin Zeng Yurong Liu Weibo Liu, Zidong Wang and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. In *Neurocomputing*, volume 234, pages 11–26. Elsevier, 2017.

[16] Emil Widell and Jesper Edström. Transferability of features from multiple depths of a deep convolutional neural network. *Master's Theses in Mathematical Sciences*, 2018.