



Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 37
Fine tuning in CNNs

So far we have seen several variants of CNN architectures, which firstly try to increase the depth to improve performance, then try to make computations more efficient, while not sacrificing performance. And then finally variants in which it was okay to sacrifice performance, but you needed it to bring the CNN under a lower regime of computations.

We saw all of these variants. And in all of these variants, including the squeeze excitation network, the back propagation of the CNN by itself was not affected, which let us play with these architectures. In this lecture, we will talk about an important aspect of CNNs that allows us to design architectures for newer tasks or newer domains, given an architecture that works on a particular domain. For example, if we know AlexNet works on the Image net data set, how can we leverage that knowledge to develop a model for another domain for another task also.

(Refer Slide Time: 01:30)





Limitations of Working with CNNs

Practical concerns of working with CNNs:

- Optimization of parameters in deep models (characteristic of CNNs) very hard, requires careful parameter initializations and hyperparameter tuning
- Can suffer from overfitting, as data samples used for training are lesser compared to parameters being trained
- Require a long time and computational power to train

Model	Parameters	Model	Training time	Hardware
AlexNet	60m	AlexNet	5-6 days	two GTX 580 3GB
VGG	138m	VGG	2-3 weeks	four NVIDIA Titan Black
Inception v1	5m	Inception v1	1 week	not mentioned
Inception v3	23m			
Resnet 50	25m			



Vineeth N.B. (IIT-H)
3.5.5 Finetuning in CNNs
2 / 10


To discuss this, let us start with limitations of CNN themselves. CNNs have a few fundamental problems. Firstly, optimization of the weights or parameters in these deep models, CNNs are typically deep models, is generally hard, it requires proper weight initialization, it requires a lot of hyper parameter tuning, various kinds of hyper parameters that is one of the fundamental limitations. Secondly, it can suffer from over fitting because

data samples used for training are generally much lesser in number to the parameters being claimed. We already saw that Alex net has 16 million parameters, VGG has 138 million, Inception v1, which is the first version of Google net had 5 million parameters, an improved version of it called Inception v3 had 23 million parameters, Resnet 50 had 25 million parameters and so on.

So CNNs fundamentally required a long time, and a lot of computational power to train these networks. Alex net, as we said, took about a week to train on 2 Nvidia GTX 580 GPUs, VGG took two to three weeks on 4 Nvidia Titan black GPUs, and Inception v1 or Google net version 1 took about a week on hardware that was not shared.

So all this can become even more difficult when you have to design an architecture for a new domain, in which case, you have to design an architecture, run it for about a week or several days, see how it works, go back and change the hyper parameter, again, run it for one more week, see how it works. And this kind of an approach to designing new architectures of CNN architectures will not scale. So how do we manage doing this for newer tasks and newer domains?



(Refer Slide Time: 03:36)



Strategies Used

- Better weight initialization:
 - **Glorot/He initialization:** Empirically shown to give good results
 - **Hand-designed:** Using domain knowledge, come up with features like edges (with certain orientations), shapes etc.
 - **Locally trained using unsupervised learning approaches:** Use unsupervised greedy layerwise pretraining to get features one layer at a time starting from the initial layer. Rarely used nowadays due to increased computational power and dataset sizes
- Regularization methods:
 - L2-weight decay, L1-weight decay
 - Dropout, BatchNorm, Input/Gradient Noise
 - Data augmentation

Alleviates overfitting, does not train faster though!



Vineeth N B. (IIT-H) 5.5 Finetuning in CNNs 3 / 10

Some things you can try are perhaps try a good weight visualization. So that way, you become independent of the design of the architecture. We have seen Glorot or He initialization, that is one possibility. You can also initialize your CNNs with hand design filters, you could use an edge filter, or one of those filters to initialize the filters in convolution layers, and then ask the CNN to fine tune them. Or, we have already seen

unsupervised greedy layer wise pre-training as another approach that could provide an initialization which the neural network can then capitalize on.

As we said earlier, this approach of unsupervised greedy layer wise pre-training is not used these days due to the increased computational power and also data set sizes, it takes a long time to do the greedy layer wise pre-training as only an initialization method. The other thing that we talked about to improve training of neural networks is to use regularization methods, be it L2 weight decay. L1 weight decay, dropout, batch norm, adding input noise, adding gradient noise, data augmentation, so on and so forth. This may alleviate over fitting, but may not really lead to faster training of CNNs. So what do we do?

(Refer Slide Time: 05:01)

Interesting Property of CNNs

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier

- Features learned by CNN layers are hierarchical
- Initial layers learn simple/generic features like edges, colour blobs, etc - remain constant across various models trained on different datasets
- Later layers perceive more abstract/specialized features and are generally dataset-specific
- What can we do with this?

Feature visualization of convolutional net trained on ImageNet from [Zeller & Fergus 2013]

Credit: CS231N, Stanford Univ

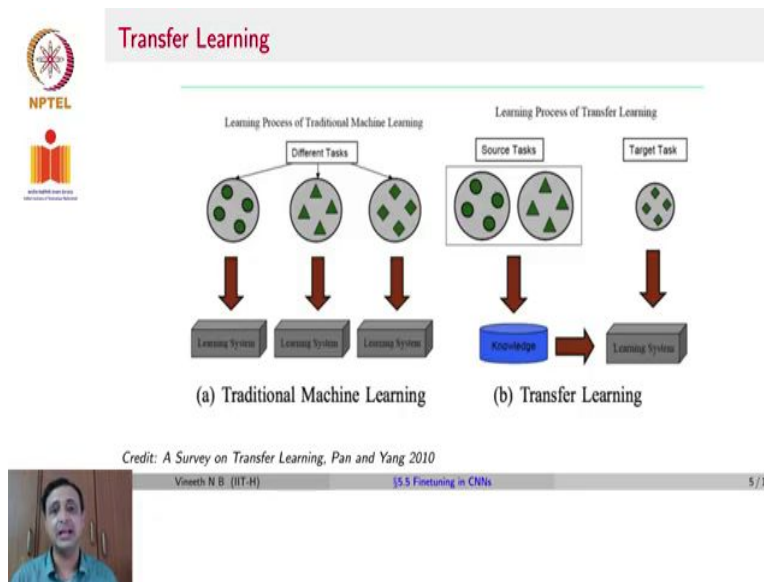
Vineeth N B. (IIT-H) 5.5 Finetuning in CNNs 4/10

An interesting property of CNN, which has been observed, which is something that we will focus on in the next week of lectures, is that if you visualize the filters that are learned by layers in the convolution network after training the CNN, one happens to observe that the earlier layers have filters that probably represent filters that could capture oriented edges of different kinds, color blobs, so on and so forth.

In other words, these are what we call low level features, simple low level features that we get out of images. Filters that are learned by intermediate layers in a CNN seem to capture some mid-level features which could be considered as combinations of low level features, a certain texture that is a combination of two edges, or a certain congregation of colors, and edges in colors, and so on and so forth.

And finally, the filters at later layers learn high level features, such as impressions of objects, for instance, which are higher level abstractions, when you compare to only edges in initial layers. The question we asked now is, what can we do with this? Can we use this in some way? Can we use this understanding of how CNN filters are learning to be able to design architectures better for newer tasks and newer domains?

(Refer Slide Time: 06:36)




The answer there is a setting in machine learning known as transfer learning. In transfer learning, before we go to transfer learning in traditional machine learning, if one had different tasks, you would have a data set for each of these tasks. And using a data set for each task, you would have a different model through a learning system that you learn.

In transfer learning, you have a set of source tasks that are given to you. And a target task on which you want to perform well, that is the new domain, a new task that you want to solve the problem on. So when we say source and target, for example, you could imagine now that you have trained Alex net, on the image net dataset and tomorrow, you have data coming from a different domain.

For example, let us say you want to build a recognition of objects for self-driving cars in India, these objects may be different from the objects in image net. So how do you now transfer the model that you learnt on image net to a model that you want to develop on in a new domain? In this case, you try to use the data in the target domain, as well as the model or the knowledge that you got from the source domain to be able to learn a model for the target domain. So this we call transfer learning to be able to transfer knowledge from an earlier


source task to a target task that we are currently interested in. How does this relate to the hierarchical learning of abstractions of features in CNNs, we will see that in a moment.

(Refer Slide Time: 08:23)



Transfer Learning

- Using knowledge learned over a different task(s) (having sufficient data) to aid the training of current task
- Since pretrained models with good results are readily available, they can reduce the time spent on training, hyperparameter tuning and thus need for high-end computing hardware
- Pretrained weights of CNN model can be used as:
 - Only parameters of classification layers are trained; rest of the network is frozen
 - Pretrained weights serve as initialization, and the entire network (or few layers at the end) are further finetuned to better model target task




Vineeth N B. (IIT-H) 5.5 Finetuning in CNNs 6 / 10

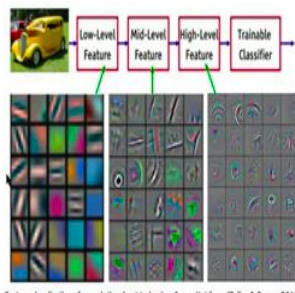
So we know now that using knowledge learned over a different task to aid the training of current tasks is transfer learning. We now also know that we do have pre trained models with good results available to us, for example, an Alex net on image net is pre trained, and that model is already given to us, can be leveraged this in some way to improve performance on a target task.

We can do this in a couple of ways. We can do this because we know that the Alex nets filters or any other models filters, any other CNN models filters for that matter, has some general features in the first few layers. And the features slowly start getting specialized as you go to deeper layers. Why do we say that?

(Refer Slide Time: 09:15)




Interesting Property of CNNs



- Features learned by CNN layers are hierarchical
- Initial layers learn simple/generic features like edges, colour blobs, etc - remain constant across various models trained on different datasets
- Later layers perceive more abstract/specialized features and are generally dataset-specific
- What can we do with this?


Credit: CS231N, Stanford Univ



Vineeth N.B. (IIT-H) 5.5 Finetuning in CNNs 4/10


If you go back to an earlier slide, you see the earlier layers filters seem to capture general information like edges, and color blobs, and so on. Whereas the later filters seem to capture some object specific information, which could be specialized for that task.

(Refer Slide Time: 09:40)



Transfer Learning

- Using knowledge learned over a different task(s) (having sufficient data) to aid the training of current task
- Since pretrained models with good results are readily available, they can reduce the time spent on training, hyperparameter tuning and thus need for high-end computing hardware
- Pretrained weights of CNN model can be used as:
 - Only parameters of classification layers are trained; rest of the network is frozen
 - Pretrained weights serve as initialization, and the entire network (or few layers at the end) are further finetuned to better model target task
- Choice depends on variables such as dataset size and similarity between target and source datasets



Vineeth N.B. (IIT-H) 5.5 Finetuning in CNNs 6/10

So what we can do now is, you could take a pre trained neural network such as an Alex net to a newer domain, and keep all of those weights of Alex net exactly the same. Do not update them for this new task, instead take only the last layer, which we call as the classification layer, and train only those parameters for the newer task.

We call this to be fine tuning, where you are initializing the weights of a network for a new task based on some pre trained weights of another model, and you only fine tune certain layers in the new network for the new task. You could now talk about the first setting here, which only trains the classification layer.

You could also use the pre trained weights as an initialization, and then fine tune the entire network or certain layers at the end of the CNN, not just the classification layer, you could take the last 3 layers or last 4 layers, anything of your choice to fine tune to better model the target task. Which one do you choose depends on your data set size, and how similar is your target task to the source task.

(Refer Slide Time: 11:05)

Which mode to select?

Dataset is small; target and source datasets are similar:

- Specialized features likely remain same for source and target datasets
- Parameters of classification layer are randomly initialized and trained, while rest of network remains frozen (to prevent overfitting)

The diagram shows an 'Input' box leading to a 'Feature Extraction Layers' block. This block contains two cyan boxes: 'Source task Generic Layers' and 'Source task Special Layers'. An arrow points from this block to a red box labeled 'Randomly initialized Classification Layers', which then leads to an 'Output' box.

At the bottom of the slide, there is a video feed of a speaker, the name 'Vineeth N B. (IIT-M)', the title '5.5 Finetuning in CNNs', and the page number '7 / 10'.

Let us see a few possibilities here and see what we can do. If your data set was fundamentally small, and your target and source data sets are fairly similar to each other. What you can do now is because this target and source datasets are similar to each other, you could say that the specialized features are likely to remain the same in both these datasets, for models trained on both these datasets.

So what you can do now is you take the source tasks, generic layers, you take the source tasks special layer, so this entire block that you see in the middle could be an entire pre trained network, such as an Alex net, or Resnet, that you trained on another data set, which we call as a source task. We leave that as it is and we only train you randomly initialize the classification layer alone and train only that one for the outputs in your new task.

This is because your data set is small, you feel you may not have enough data to retrain or fine tune all these feature abstraction layers, that as well focus only on training the classification layer properly. So in this case, we are initializing the new network with a pre trained weights and we are fine tuning only the classification layer.



(Refer Slide Time: 12:26)

Another setting is where the data set is small and also the target and the source data sets are dissimilar this time, which means the specialized features could still be common. The Edge detectors and those kinds of filters could be common for the source and target task. But the specialized features may not really be common between the two tasks.

What can we do here, we now consider the entire pre trained CNN because we know that the feature maps of the generic layers which are by General layers, we mean the early layers of the CNN, you can consider that from your pre train network, take those feature maps as output, and then train another classifier such as support vector machine, or could also be a neural network to give you the output for the target task.

Why are we doing this? Because the data set is small, and the target and source data sets are dissimilar, we cannot use the specialized features of your pre trained network on the source task. We now assume that the generic features that we get from the first few layers by features by representations we mean those feature maps for each input. Those generic representations, we assume will help us improve performance on a classifier, although the data set is small.

(Refer Slide Time: 13:57)



Which mode to select?

Dataset is large:


- We can use pretrained network as a good initialization which is finetuned on target dataset
- While finetuning, learning rate is kept low in order to not change pretrained parameters too much
- If dataset is very different, it can either be trained from scratch or techniques like transitive transfer learning¹ or its successors can be applied

Diagram:

```
graph LR; Input[Input] --> FE[Feature Extraction Layers]; subgraph FE; direction LR; G[Source task/ Randomly initialized Generic Layers]; S[Source task/ Randomly initialized Special Layers]; end; FE --> CL[Randomly initialized Classification Layers]; CL --> Output[Output];
```

¹Tan et al, Transitive Transfer Learning, KDD 2015

Vineeth N.B. (IIT-H) 5.5 Finetuning in CNNs 9 / 10



A third setting now is when your data set is fundamentally large. When your data set is large, you have some more luxury, you can now use the pre-trained network as a good initialization to fine tune the entire network now on the target data set. But you can do a couple of things also intelligently to improve performance.

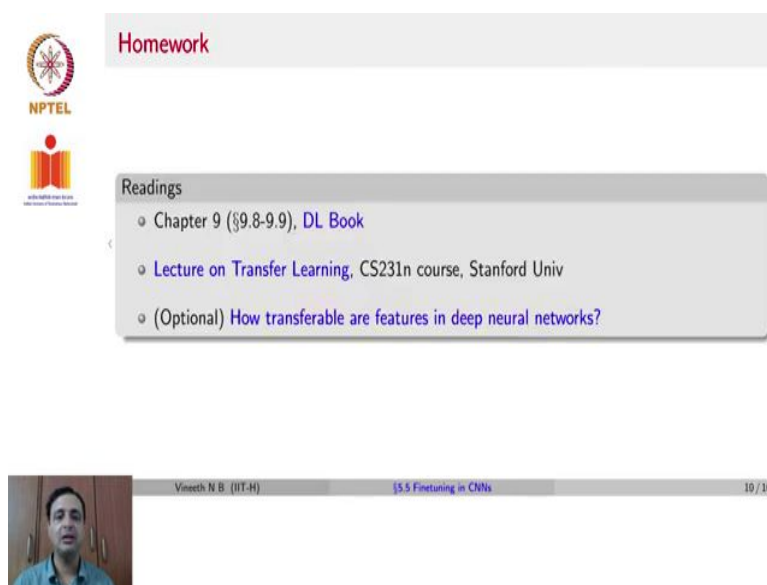
While fine tuning, you can keep the learning rate to be quite low. Remember, learning rate is the coefficient of your gradient in your gradient descent method, you can keep your learning rate to be quite low, so that your pre trained parameters are not changed significantly. You are saying that I still want my overall network weights to be similar to the weights that I learned from my source task. But I have the data, I am going to update it but let me keep the learning rate low so that I do not make too many updates to my weights from what they were from a source task.

On the other hand, if data set is very different, and the data set is large, there are explicit methods for this, such as what is known as transitive transfer learning or things like that, where you can either train from scratch, you do not need to fine tune at all, you can randomly initialize using Clorox initialization and train from scratch on the new target task because you have a large data set, you are saying that the target task and the source tasks are not similar to each other so you may as well train from scratch, or use methods such as transitive transfer learning, which is a known method in traditional machine learning and that can help you improve performance.

The main idea of this discussion is to tell you that the design of architectures for CNNs while we saw a few different designs, it is not trivial to design newer and newer architectures for a given problem. There are many hyper parameters, number of layers with resolution, number of filters and also learning hyper parameters to try to come up with the appropriate combination for a domain and a task by itself can be a very difficult task.

And using the idea of transfer learning and fine tuning could help you take an existing architecture that has solved a different problem and adapt it to a new problem. This is a very common procedure that people follow to take CNNs to newer domains and newer tasks.

(Refer Slide Time: 16:33)



The screenshot shows a presentation slide with the following content:

- Homework**
- Readings**
 - Chapter 9 (§9.8-9.9), DL Book
 - Lecture on Transfer Learning, CS231n course, Stanford Univ
 - (Optional) [How transferable are features in deep neural networks?](#)

At the bottom of the slide, there is a video thumbnail of the presenter, Vineeth N.B. (IIT-H), and a progress indicator showing 10/10 slides.

For more reading, you can read chapter 9 of the deep learning book and the lecture on transfer learning in CS 231n. And if you are interested, one of the earliest articles that try to study how transferable are features of CNNs, in the context that we just discussed.