A Comprehensive guide to Fine-tuning Deep Learning Models in Keras (Part I)

OCTOBER 3, 2016

In this post, I am going to give a comprehensive overview on the practice of fine-tuning, which is a common practice in Deep Learning.

Drawing from my own experience, I will list out the rationale behind fine-tuning, the techniques involved, and last and most important of all, detailed step-by-step guide of how to fine-tune Convolutional Neural Network models in Keras in Part II of this post.

First, why do we fine-tune Models?

When we are given a Deep Learning task, say, one that involves training a Convolutional Neural Network (Covnet) on a dataset of images, our first instinct would be to train the network from scratch. However, in practice, deep neural networks like Covnet has a huge number of parameters, often in the range of millions. Training a Covnet on a small dataset (one that is smaller than the number of parameters) greatly affects the Covnet's ability to generalize, often result in overfitting.

Therefore, more often in practice, one would fine-tune existing networks that are trained on a large dataset like the ImageNet (1.2M labeled images) by continue training it (i.e. running back-propagation) on the smaller dataset we have. Provided that our dataset is not drastically different in context to the original dataset (e.g. ImageNet), the pre-trained model will already have learned features that are relevant to our own classification problem.

When to fine-tune Models?

In general, if our dataset is not drastically different in context from the dataset which the pre-trained model is trained on, we should go for fine-tuning. Pre-trained network on a large and diverse dataset like the ImageNet captures universal features like curves and edges in its early layers, that are relevant and useful to most of the classification problems.

Of course, if our dataset represents some very specific domain, say for example, medical images or Chinese handwritten characters, and that no pre-trained networks on such domain can be found, we should then consider training the network from scratch.

One other concern is that if our dataset is small, fine-tuning the pre-trained network on a small dataset might lead to overfitting, especially if the last few layers of the network are fully connected layers, as in the case for VGG network. Speaking from my experience, if we have a few thousand raw samples, with the common data augmentation strategies implemented (translation, rotation, flipping, etc), fine-tuning will usually get us a better result.

If our dataset is really small, say less than a thousand samples, a better approach is to take the output of the intermediate layer prior to the fully connected layers as features (bottleneck features) and train a linear classifier (e.g. SVM) on top of it. SVM is particularly good at drawing decision boundaries on a small dataset.

Fine-tuning Techniques

Below are some general guidelines for fine-tuning implementation:

1. The common practice is to ***truncate the last layer*** (softmax layer) of the pre-trained network and replace it with our new softmax layer that are relevant to our own problem. For example, pre-trained network on ImageNet comes with a softmax layer with 1000 categories.

If our task is a classification on 10 categories, the new softmax layer of the network will be of 10 categories instead of 1000 categories. We then run back propagation on the network to fine-tune the pre-trained weights. Make sure cross validation is performed so that the network will be able to generalize well.

2. ***Use a smaller learning rate*** to train the network. Since we expect the pre-trained weights to be quite good already as compared to randomly initialized weights, we do not want to distort them too quickly and too much. A common practice is to make the initial learning rate 10 times smaller than the one used for scratch training.

3. It is also a common practice to ***freeze the weights of the first few layers*** of the pre-trained network. This is because the first few layers capture universal features like curves and edges that are also relevant to our new problem. We want to keep those weights intact. Instead, we will get the network to focus on learning dataset-specific features in the subsequent layers.

Where can we find Pre-trained Networks?

It depends on the deep learning framework. For popular frameworks like *Caffe*, *Keras*, *TensorFlow*, *Torch*, *MxNet*, etc, their respective contributors usually keep a list of the state-of-the-art Covnet models (VGG, Inception, ResNet, etc) with their implementations and pre-trained weights on a common dataset like the ImageNet or CIFAR.

The best way to find those pre-trained models is to google your specific model and framework. However, to facilitate your search process, I put together a list with the common pre-trained Covnet models on popular frameworks.

- **Caffe**
  - Model Zoo - A platform for third party contributors to share pre-trained caffe models

- **Keras**
  - Keras Application - Implementation of popular state-of-the-art Convnet models like VGG16/19, googleNetNet, Inception V3, and ResNet

- **TensorFlow**
  - VGG16
  - Inception V3

    - ResNet

- **Torch**
    - LoadCaffe - Maintains a list of popular models like AlexNet and VGG .Weights ported from Caffe

- **MxNet**
    - MxNet Model Gallery - Maintains pre-trained Inception-BN (V2) and Inception V3.

Fine-tuning in Keras

In Part II of this post, I will give a detailed step-by-step guide on how to go about implementing fine-tuning on popular models VGG, Inception V3, and ResNet in Keras.

If you have any questions or thoughts feel free to leave a comment below.

You can also follow me on Twitter at @flyyufelix.

Edwin • 4 months ago

"Training a Covnet on a small dataset (one that is smaller than the number of parameters) greatly affects the Covnet's ability to generalize, often result in overfitting."

You do know that VGG-16 had 138 million parameters while the training dataset only had 1.2 million images right......

Peeteer • 2 years ago

Hi Felix, I think that I understand how to fine tine model in order to classify my own 10-class dataset but I looking for a way to learn model additional classes instead.
Should I cut out last model layer and keep it somewhere, create new softmax layer then add it to the model then train new classes and merge there 2 softmax layers together?

^ | v • Reply • Share ›

Saif Moammed ➦ Peeteer • a year ago

Hi, I'm having the same problem. Didi you figure out how to add your new 10 classes to the original classifier?

^ | v • Reply • Share ›

Peeteer ➦ Saif Moammed • a year ago

I'm not sure but what I've understood from some sources the best way is to learn from scratch new classes, but it depends how much they are different from trained dataset

^ | v • Reply • Share ›

Sivaram Rolangi • 2 years ago

How to use the classification network for Segmentation

^ | v • Reply • Share ›

Felix Long Yin Yu ➦ Sivaram Rolangi • 2 years ago

These are 2 different problems. For Segmentation you can refer to architectures like U-Net and Mask RCNN

1 ^ | v • Reply • Share ›

AP • 2 years ago

Thanks for the post. BTW, it should be "convnet" (not "covnet").

^ | v • Reply • Share ›

Felix Long Yin Yu ➦ AP • 2 years ago

Ouuuh, thanks for reminding :)

^ | v • Reply • Share ›

Olix • 2 years ago

Thanks Felix for the guide and repo. One question: you don't seem to subtract images from mean before feeding them into the network for training while i thought it's a required step with pretrained models. Could you pls elaborate?

^ | v • Reply • Share ›

Felix Long Yin Yu ➦ Olix • 2 years ago

Good point. Mean subtraction is required to get these pre-trained models to work. You can refer to my README at https://github.com/flyyufel...

https://github.com/flyyufelix...

Reply • Share ›

Guilherme Felix Long Yin Yu • a year ago • edited

If you simply subtract from the mean like you did, aren't some pixels potentially gonna underflow?

Reply • Share ›

Xu Zhang • 2 years ago

What do you mean in this sentence? "Training a Covnet on a small dataset (one that is smaller than the
number of parameters) greatly affects the Covnet’s ability to generalize, often result in overfitting"
Do you mean when the number of samples in a dataset is smaller than the number of parameters, if we train our model from the scratch, the model will be overfitting for sure?

Reply • Share ›

Felix Long Yin Yu Xu Zhang • 2 years ago

Yes, if the number of training samples is smaller than the number of parameters, we might run the risk of overfitting to the