

## Fine-tune with Pretrained Models

Many of the exciting deep learning algorithms for computer vision require massive datasets for training. The most popular benchmark dataset, ImageNet (<http://www.image-net.org/>), for example, contains one million images from one thousand categories. But for any practical problem, we typically have access to comparatively small datasets. In these cases, if we were to train a neural network's weights from scratch, starting from random initialized parameters, we would overfit the training set badly.

One approach to get around this problem is to first pretrain a deep net on a large-scale dataset, like ImageNet. Then, given a new dataset, we can start with these pretrained weights when training on our new task. This process is commonly called *fine-tuning*. There are a number of variations of fine-tuning. Sometimes, the initial neural network is used only as a *feature extractor*. That means that we freeze every layer prior to the output layer and simply learn a new output layer. In another document ([https://github.com/dmlc/mxnet-notebooks/blob/master/python/how\\_to/predict.ipynb](https://github.com/dmlc/mxnet-notebooks/blob/master/python/how_to/predict.ipynb)), we explained how to do this kind of feature extraction. Another approach is to update all of the network's weights for the new task, and that's the approach we demonstrate in this document.

To fine-tune a network, we must first replace the last fully-connected layer with a new one that outputs the desired number of classes. We initialize its weights randomly. Then we continue training as normal. Sometimes it's common to use a smaller learning rate based on the intuition that we may already be close to a good result.

In this demonstration, we'll fine-tune a model pretrained on ImageNet to the smaller caltech-256 dataset. Following this example, you can fine-tune to other datasets, even for strikingly different applications such as face identification.

We will show that, even with simple hyper-parameters setting, we can match and even outperform state-of-the-art results on caltech-256.

Network	Accuracy
Resnet-50	77.4%
Resnet-152	86.4%

## Prepare data

We follow the standard protocol to sample 60 images from each class as the training set, and the rest for the validation set. We resize images into 256x256 size and pack them into the rec file. The scripts to prepare the data is as following.

In order to successfully run the following bash script on Windows please use <https://cygwin.com/install.html> .

```
wget http://www.vision.caltech.edu/Image_Datasets/Caltech256/256_ObjectCategories.tar
tar -xf 256_ObjectCategories.tar

mkdir -p caltech_256_train_60
for i in 256_ObjectCategories/*; do
  c=`basename $i`
  mkdir -p caltech_256_train_60/$c
  for j in `ls $i/*.jpg | shuf | head -n 60`; do
    mv $j caltech_256_train_60/$c/
  done
done

python ~/mxnet/tools/im2rec.py --list True --recursive True caltech-256-60-train caltech_256_train_60/
python ~/mxnet/tools/im2rec.py --list True --recursive True caltech-256-60-val 256_ObjectCategories/
python ~/mxnet/tools/im2rec.py --resize 256 --quality 90 --num-thread 16 caltech-256-60-val 256_ObjectCategories/
python ~/mxnet/tools/im2rec.py --resize 256 --quality 90 --num-thread 16 caltech-256-60-train caltech_256_train_60/
```

The following code downloads the pregenerated rec files. It may take a few minutes.

```
import os, urllib
def download(url):
    filename = url.split("/")[-1]
    if not os.path.exists(filename):
        urllib.urlretrieve(url, filename)
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-train.rec')
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-val.rec')
```

Next, we define the function which returns the data iterators:

```
import mxnet as mx

def get_iterators(batch_size, data_shape=(3, 224, 224)):
    train = mx.io.ImageRecordIter(
        path_imgrec      = './caltech-256-60-train.rec',
        data_name         = 'data',
        label_name        = 'softmax_label',
        batch_size        = batch_size,
        data_shape         = data_shape,
        shuffle            = True,
        rand_crop          = True,
        rand_mirror        = True)
    val = mx.io.ImageRecordIter(
        path_imgrec       = './caltech-256-60-val.rec',
        data_name         = 'data',
        label_name        = 'softmax_label',
        batch_size        = batch_size,
        data_shape         = data_shape,
        rand_crop          = False,
        rand_mirror        = False)
    return (train, val)
```

We then download a pretrained 50-layer ResNet model and load it into memory. Note that if `load_checkpoint` reports an error, we can remove the downloaded files and try `get_model` again.

```
def get_model(prefix, epoch):
    download(prefix+'-symbol.json')
    download(prefix+'-%04d.params' % (epoch,))

get_model('http://data.mxnet.io/models/imagenet/resnet/50-layers/resnet-50', 0)
sym, arg_params, aux_params = mx.model.load_checkpoint('resnet-50', 0)
```

## Train

We first define a function which replaces the last fully-connected layer for a given network.

```
def get_fine_tune_model(symbol, arg_params, num_classes, layer_name='flatten0'):
    """
    symbol: the pretrained network symbol
    arg_params: the argument parameters of the pretrained model
    num_classes: the number of classes for the fine-tune datasets
    layer_name: the layer name before the last fully-connected layer
    """
    all_layers = symbol.get_internals()
    net = all_layers[layer_name+'_output']
    net = mx.symbol.FullyConnected(data=net, num_hidden=num_classes, name='fc1')
    net = mx.symbol.SoftmaxOutput(data=net, name='softmax')
    new_args = dict({k:arg_params[k] for k in arg_params if 'fc1' not in k})
    return (net, new_args)
```

Now we create a module. We first call `init_params` to randomly initialize parameters, next use `set_params` to replace all parameters except for the last fully-connected layer with pretrained model.

```
import logging
head = '%(asctime)-15s %(message)s'
logging.basicConfig(level=logging.DEBUG, format=head)

def fit(symbol, arg_params, aux_params, train, val, batch_size, num_gpus):
    devs = [mx.gpu(i) for i in range(num_gpus)]
    mod = mx.mod.Module(symbol=symbol, context=devs)
    mod.fit(train, val,
        num_epoch=8,
        arg_params=arg_params,
        aux_params=aux_params,
        allow_missing=True,
        batch_end_callback = mx.callback.Speedometer(batch_size, 10),
        kvstore='device',
        optimizer='sgd',
        optimizer_params={'learning_rate':0.01},
        initializer=mx.init.Xavier(rnd_type='gaussian', factor_type="in", magnitude=2),
        eval_metric='acc')
    metric = mx.metric.Accuracy()
    return mod.score(val, metric)
```

Then we can start training. We use AWS EC2 g2.8xlarge, which has 8 GPUs.

```
num_classes = 256
batch_per_gpu = 16
num_gpus = 8

(new_sym, new_args) = get_fine_tune_model(sym, arg_params, num_classes)


batch_size = batch_per_gpu * num_gpus
(train, val) = get_iterators(batch_size)
mod_score = fit(new_sym, new_args, aux_params, train, val, batch_size, num_gpus)
assert mod_score > 0.77, "Low training accuracy."
```

You will see that, after only 8 epochs, we can get 78% validation accuracy. This matches the state-of-the-art results training on caltech-256 alone, e.g. VGG ([http://www.robots.ox.ac.uk/~vgg/research/deep\\_eval/](http://www.robots.ox.ac.uk/~vgg/research/deep_eval/)).

Next, we try to use another pretrained model. This model was trained on the complete Imagenet dataset, which is 10x larger than the Imagenet 1K classes version, and uses a 3x deeper Resnet architecture.

```
get_model('http://data.mxnet.io/models/imagenet-11k/resnet-152/resnet-152', 0)
sym, arg_params, aux_params = mx.model.load_checkpoint('resnet-152', 0)
(new_sym, new_args) = get_fine_tune_model(sym, arg_params, num_classes)
mod_score = fit(new_sym, new_args, aux_params, train, val, batch_size, num_gpus)
assert mod_score > 0.86, "Low training accuracy."
```

As can be seen, even for a single data epoch, it reaches 83% validation accuracy. After 8 epoches, the validation accuracy increases to 86.4%.

 [finetune.ipynb](#) (finetune.ipynb)