Felix Yu

About Me

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

OCTOBER 8, 2016

This is Part II of a 2 part series that cover fine-tuning deep learning models in Keras. Part I states the motivation and rationale behind fine-tuning and gives a brief introduction on the common practices and techniques. This post will give a detailed step-by-step guide on how to go about implementing fine-tuning on popular models *VGG*, *Inception*, and *ResNet* in Keras.

Why do we pick Keras?

Keras is a simple to use neural network library built on top of Theano or TensorFlow that allows developers to prototype ideas very quickly. Unless you are doing some cutting-edge research that involves customizing a completely novel neural architecture with different activation mechanism, Keras provides all the building blocks you need to build reasonably sophisticated neural networks.

It also comes with a great documentation and tons of online resources.

Note on Hardware

I would strongly suggest getting a GPU to do the heavy computation involved in Covnet training. The speed difference is very substantial. We are talking about a matter of hours with a GPU versus a matter of days with a CPU.

I would recommend GTX 980Ti or a slightly expensive GTX 1080 which cost around $600 bucks.

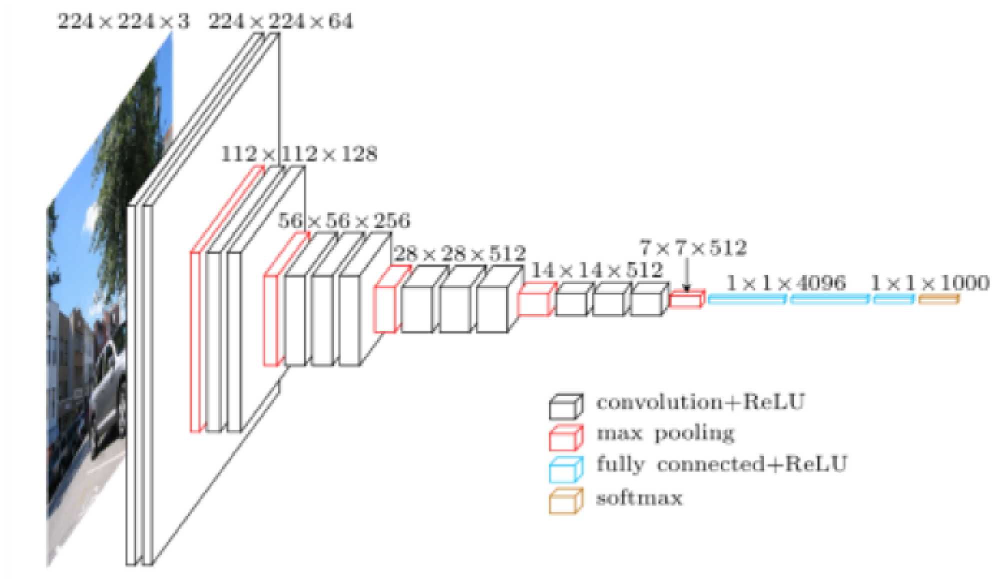
Fine-tuning in Keras

I have implemented starter scripts for fine-tuning convnets in Keras. The scripts are hosted in this github page. Implementations of VGG16, VGG19, GoogLeNet, Inception-V3, and ResNet50 are included. With that, you can customize the scripts for your own fine-tuning task.

Below is a detailed walkthrough of how to fine-tune **VGG16** and **Inception-V3** models using the scripts.

Fine-tune VGG16. VGG16 is a 16-layer Covnet used by the Visual Geometry Group (VGG) at Oxford University in the 2014 ILSVRC (ImageNet) competition. The model achieves a 7.5% top 5 error rate on the validation set, which is a result that earned them a second place finish in the competition.

Schematic Diagram of VGG16 Model:



The script for fine-tuning VGG16 can be found in vgg16.py. The first part of the `vgg_std16_model` function is the model schema for VGG16. After defining the fully connected layer, we load the ImageNet pre-trained weight to the model by the following line:

```
model.load_weights('cache/vgg16_weights.h5')
```

For fine-tuning purpose, we truncate the original softmax layer and replace it with our own by the following snippet:

```
model.layers.pop()
model.outputs = [model.layers[-1].output]
model.layers[-1].outbound_nodes = []
model.add(Dense(num_class, activation='softmax'))
```

Where the `num_class` variable in the last line represents the number of class labels for our classification task.

Sometimes, we want to freeze the weight for the first few layers so that they remain intact throughout the fine-tuning process. Say we want to freeze the weights for the first 10 layers. This can be done by the following lines:

```
for layer in model.layers[:10]:
    layer.trainable = False
```

We then fine-tune the model by minimizing the cross entropy loss function using stochastic gradient descent (sgd) algorithm. Notice that we use an initial learning rate of 0.001, which is smaller than the learning rate for training scratch model (usually 0.01).

```
sgd = SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=Tru
model.compile(optimizer=sgd, loss='categorical_crossentro
model = vgg_std16_model(img_rows, img_cols, channel, num_
```

Where img_rows, img_cols, and channel define the dimension of the input. For colored image with resolution 224x224, img_rows = img_cols = 224, channel = 3.

Next, we load our dataset, split it into training and testing sets, and start fine-tuning the model:

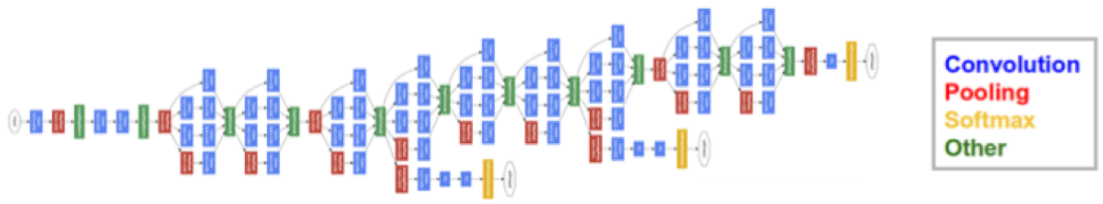
```
X_train, X_valid, Y_train, Y_valid = load_data()

model.fit(train_data, test_data,
          batch_size=batch_size,
          nb_epoch=nb_epoch,
          shuffle=True,
          verbose=1,
          validation_data=(X_valid, Y_valid),
          )
```

The fine-tuning process will take a while, depending on your hardware. After it is done, we use the model the make prediction on the validation set and return the score for the cross entropy loss:

```
predictions_valid = model.predict(X_valid, batch_size=bat
score = log_loss(Y_valid, predictions_valid)
```

Fine-tune Inception-V3. Inception-V3 achieved the second place in the 2015 ImageNet competition with a 5.6% top 5 error rate on the validation set. The model is characterized by the usage of the Inception Module, which is a concatenation of features maps generated by kernels of varying dimensions.

Schematic Diagram of the 27-layer Inception-V1 Model (Idea similar to that of V3):



The code for fine-tuning Inception-V3 can be found in inception_v3.py. The process is mostly similar to that of VGG16, with one subtle difference. Inception-V3 does not use Keras' Sequential Model due to branch merging (for the inception module), hence we cannot simply use `model.pop()` to truncate the top layer.

Instead, after we create the model and load it up with the ImageNet weight, we perform the equivalent of top layer truncation by defining another fully connected sofmax (`x_newfc`) on top of the last inception module (x). This is done using the following snippet:

```
# Last Inception Module
x = merge([branch1x1, branch3x3, branch3x3dbl, branch_poo
                  mode='concat', concat_axis=channel_axis
                  name='mixed' + str(9 + i))

# Fully Connected Softmax Layer
x_fc = AveragePooling2D((8, 8), strides=(8, 8), name='avg
x_fc = Flatten(name='flatten')(x_fc)
x_fc = Dense(1000, activation='softmax', name='prediction

# Create model
```

```
model = Model(img_input, x_fc)

# Load ImageNet pre-trained data
model.load_weights('cache/inception_v3_weights_th_dim_ord

# Truncate and replace softmax layer for transfer learnin
# Cannot use model.layers.pop() since model is not of Seq
# The method below works since pre-trained weights are st
x_newfc = AveragePooling2D((8, 8), strides=(8, 8), name='
x_newfc = Flatten(name='flatten')(x_newfc)

# Create another model with our customized softmax
model = Model(img_input, x_newfc)

# Learning rate is changed to 0.001
sgd = SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=Tru
model.compile(optimizer=sgd, loss='categorical_crossentro
model = inception_v3_model(img_rows, img_cols, channel, n
```

That's it for Inception-V3. Starting script for other models such as **VGG19**, **GoogleLeNet**, and **ResNet** can be found here.

Fine-tuned Networks in Action

If you are a Deep Learning or Computer Vision practitioner, most likely you have already tried fine-tuning pre-trained network for your own classification problem before.

To me, I came across this interesting Kaggle Competition which requires candidates to identify distracted drivers through analyzing in-car camera images. This is a good opportunity for me to try out fine-tuning in Keras.

Following the fine-tuning methods listed above, together with data preprocessing/augmentation and model ensembling our team managed to achieved top 4% in the competition. Detailed account of our method and lessons learned are captured in this post.

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

You can also follow me on Twitter at @flyyufelix.

ALSO ON **FLYYUFELIX.GITHUB.IO**

Lessons learned from Kaggle StateFarm …
4 years ago • 4 comments
I recently took part in the Kaggle State Farm Distracted Driver …

Convert Caffe weights to Keras for …
4 years ago • 6 comments
MotivationThose who have applied deep learning would know, being deep is both …

Train a Reinforce Learning agent t
3 years ago • 4 comme
OpenAI hosted a cor challenging participa create the best agent

18 Comments **flyyufelix.github.io** **Privacy Policy** 1 **Login**

Recommend 7 Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Yineng Xiong • 3 years ago

Hello
what if I want to use
from keras.applications import VGG19
to train a VGG network on CIFAR10 dataset
what should I do?

2 | 1 • Reply • Share ›

Brian • 3 years ago

Could you add a little explanation on what exactly these commands are doing?

1) model.layers.pop()
2) model.outputs = [model.layers[-1].output]
3) model.layers[-1].outbound_nodes = []
4) model.add(Dense(num_class, activation='softmax'))

Since model layers are in a list, #1 removes the last layer (the original softmax)

#2 -#3 I am unsure about

#4 adds the new softmax layer.

1 | • Reply • Share ›

Chester → Brian • 3 years ago

Since in #1 we have popped the last layer, which is the original output layer, we need to update the destination of the output layer. #2 is to reset the second last layer as the output layer. #3 is to say the second last layer of the original model should not have anymore connections to the popped layer, since it is now the output layer.

| • Reply • Share ›

Brian → Chester • 3 years ago

Is this detailed anywhere - I guess the inner workings of the

Sequential model? Or is it gained only by trying to go through the code? Cheers!

Reply • Share ›

Chester ➦ Brian • 3 years ago

It's documented here
https://keras.io/layers/abo...
If you look at the documentation on model.layer.set_weights, it allows you to set the weights as you initialize the layer, so that you don't have to do the above if you choose to implement transfer learning in a non-sequential fashion (strip layers in the middle, etc...)

Reply • Share ›

Harpreet Singh ➦ Chester • 3 years ago

You don't need to do this. Just use model.pop() and it takes care of the output shape stuff.
https://github.com/keras-te...

1 • Reply • Share ›

essay writing australia • 3 years ago

Might be a nice thing that you have created those kind of guides that would ensure someone's idea to be nourished. Thus, it would be a great step of their career to improve their knowledge when it comes to programming.

1 • Reply • Share ›

Felix Long Yin Yu ➦ essay writing australia • 3 years ago

Thanks for your kind words. Hope it helps!

Reply • Share ›

Oxydron • 4 years ago

Thank you very much for this post! I was looking for this sort of guide and finally got it.
Best regards and nice work.

1 • Reply • Share ›

Felix Long Yin Yu ➦ Oxydron • 3 years ago

Thanks Ben. Hope it helps!

1 • Reply • Share ›

sanch • 2 years ago

The code is clearly not working and where is the model . ? "cache/vgg16_weights.h5" is not there. Also, it keeps on giving dim_ordering error on the pooling layers

1 ^ | ˅ 1 • Reply • Share ›

Aditya Ramesh • 3 years ago

Great post Felix.
When we are using pretrained weights for a model like VGG16 on Keras, would it be necessary to preprocess the input the same way the original model did?

Do I need to use keras.applications.VGG16.preprocess_input as a preprocessing function for my training set?

1 ^ | ˅ 1 • Reply • Share ›

abhay kumar • 2 years ago

You have forgot to mention the customised class softmax layers with num_classes in inception example

^ | ˅ • Reply • Share ›

George Delisle • 2 years ago

Hello,
I was tasked with creating a sort of image classifier based on signal shape, taken from oscilloscope screenshots. When running pictures of the signals through the VGG16 model right now, I get a 99.9% prediction for "oscilloscope" which I believe is a good start. Do you know of any guidance or any beginning steps as to how I can get the model to differentiate between certain characteristics in the signal, such as color or shape? Many thanks for your help!

George

^ | ˅ • Reply • Share ›

Rahul Kulhalli • 3 years ago

Hey Felix!
First of all, great post!
Just out of curiosity, is it fine to use Adam for the bottleneck training phase (with a low learning rate, as you mentioned?)

^ | ˅ • Reply • Share ›

Priya Gupta • 3 years ago

In your code