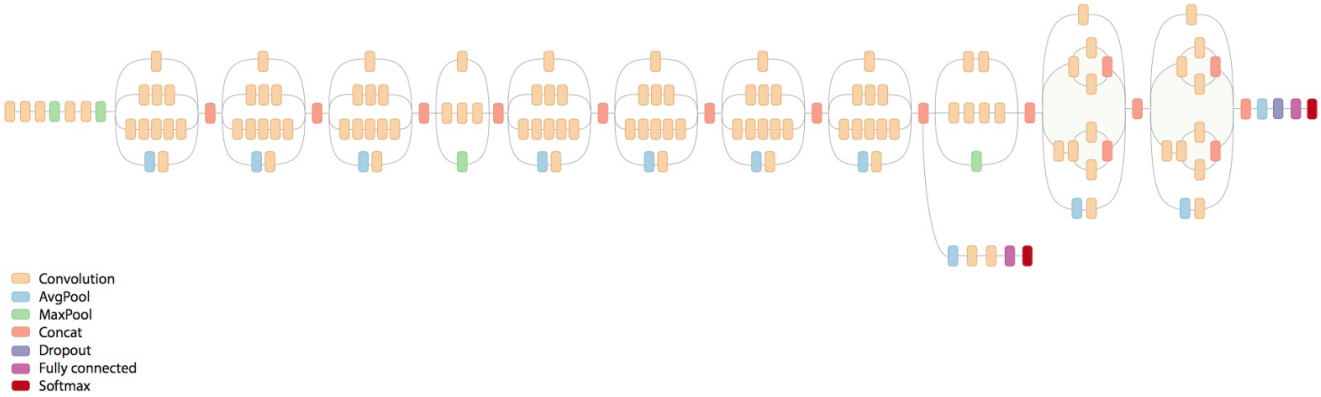
Prakash Jay Follow
Data Scientist
Apr 15 · 4 min read

# Transfer Learning using Keras




Inception-V3 Google Research

## What is Transfer Learning?

Transfer learning, is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

## Why Transfer Learning?

- • In practice a very few people train a Convolution network from scratch (random initialisation) because it is rare to get enough dataset. So, using pre-trained network weights as initialisations or a fixed feature extractor helps in solving most of the problems in hand.

- Very Deep Networks are expensive to train. The most complex models take weeks to train using hundreds of machines equipped

with expensive GPUs.

Determining the topology/flavour/training method/hyper parameters for deep learning is a black art with not much theory to guide you.

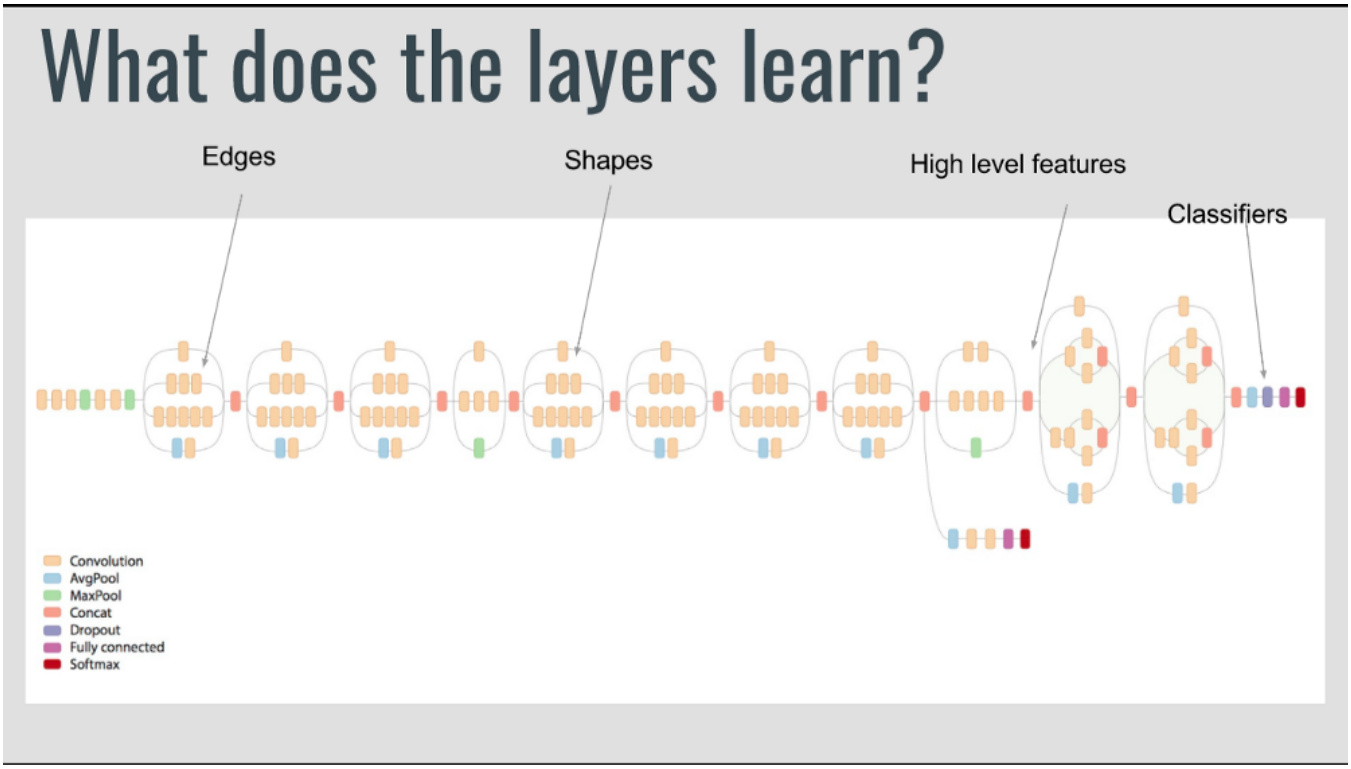
## My Experiences:

*"DON'T TRY TO BE AN HERO" ~Andrej Karapathy*

Most of the Computer Vision Problems I faced doesn't have very large datasets(5000 images—40,000 images). Even with extreme data augmentation strategies it is difficult to achieve decent accuracy. Training these networks with millions of parameters generally tend to overfit the model. So Transfer learning comes to our rescue.

## How Transfer Learning helps ?

When you look at what these Deep Learning networks learn, they try to detect edges in the earlier layers, Shapes in the middle layer and some high level data specific features in the later layers. These trained networks are generally helpful in solving other computer vision problems. Lets have a look at how to do transfer learning using Keras and various cases in Transfer learning.




Inception V3 Google Research

## 3. Simple implementation using Keras:

```
from keras import applications
from keras.preprocessing.image import ImageDataGenera
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, Glo
from keras import backend as k
from keras.callbacks import ModelCheckpoint, Learning

img_width, img_height = 256, 256
train_data_dir = "data/train"
validation_data_dir = "data/val"
nb_train_samples = 4125
nb_validation_samples = 466
batch_size = 16
epochs = 50

model = applications.VGG19(weights = "imagenet", incl

"""
Layer (type)                 Output Shape
=================================================
input_1 (InputLayer)         (None, 256, 256, 3)
_________________________________________________
block1_conv1 (Conv2D)        (None, 256, 256, 64)
_________________________________________________
block1_conv2 (Conv2D)        (None, 256, 256, 64)
_________________________________________________
block1_pool (MaxPooling2D)   (None, 128, 128, 64)
_________________________________________________
block2_conv1 (Conv2D)        (None, 128, 128, 128)
_________________________________________________
block2_conv2 (Conv2D)        (None, 128, 128, 128)
_________________________________________________
block2_pool (MaxPooling2D)   (None, 64, 64, 128)
_________________________________________________
block3_conv1 (Conv2D)        (None, 64, 64, 256)
_________________________________________________
block3_conv2 (Conv2D)        (None, 64, 64, 256)
_________________________________________________
block3_conv3 (Conv2D)        (None, 64, 64, 256)
_________________________________________________
block3_conv4 (Conv2D)        (None, 64, 64, 256)
```

Keeping in mind that convnet features are more generic in early layers and more original-dataset-specific in later layers, here are some common rules of thumb for navigating the 4 major scenarios:

## 1. New dataset is small and similar to original dataset:

There is a problem of over-fitting, if we try to train the entire network. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.

So lets freeze all the VGG19 layers and train only the classifier

```
for layer in model.layers:
   layer.trainable = False

#Now we will be training only the classifiers (FC layers)
```

## 2. New dataset is large and similar to the original dataset

Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune through the full network.

```
for layer in model.layers:
   layer.trainable = True

#The default is already set to True. I have mentioned it
here to make things clear.
```

In case if you want to freeze the first few layers as these layers will be detecting edges and blobs, you can freeze them by using the following code.

```
for layer in model.layers[:5]:
```

```
    layer.trainable = False.

# Here I am freezing the first 5 layers
```

### 3. New dataset is small but very different from the original dataset

Since the dataset is very small, We may want to extract the features from the earlier layer and train a classifier on top of that. This requires a little bit of knowledge on h5py.

```
from keras import applications
from keras.preprocessing.image import ImageDataGenera
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, Glo
from keras import backend as k
from keras.callbacks import ModelCheckpoint, Learning

img_width, img_height = 256, 256

### Build the network
img_input = Input(shape=(256, 256, 3))
x = Conv2D(64, (3, 3), activation='relu', padding='sa
x = Conv2D(64, (3, 3), activation='relu', padding='sa
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='s
x = Conv2D(128, (3, 3), activation='relu', padding='s
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2

model = Model(input = img_input, output = x)

model.summary()
"""
_________________________________________________________
Layer (type)                 Output Shape
=========================================================
input_1 (InputLayer)         (None, 256, 256, 3)
_________________________________________________________
block1_conv1 (Conv2D)        (None, 256, 256, 64)
_________________________________________________________
block1_conv2 (Conv2D)        (None, 256, 256, 64)
_________________________________________________________
block1_pool (MaxPooling2D)   (None, 128, 128, 64)
_________________________________________________________
block2_conv1 (Conv2D)        (None, 128, 128, 128)
_________________________________________________________
block2_conv2 (Conv2D)        (None, 128, 128, 128)
_________________________________________________________
block2_pool (MaxPooling2D)   (None, 64, 64, 128)
=========================================================
```

The above code should help. It will extract the “block2_pool” features. In general this is not helpful as this layer has (64*64*128) features and training a classifier on top of it might not help us exactly. We can add a few FC layers and train a neural network on top of it. That should be straight forward.

Add few FC layers and output layer.

Set the weights for earlier layers and freeze them.

Train the network.

### 4. New dataset is large and very different from the original dataset.

This is straight forward. since you have large dataset, you can design your own network or use the existing ones.

Train the network using random initialisations or use the pre-trained network weights as initialisers. The second one is generally preferred.

If you are using a different network or making small modification here and there for the existing network, Be careful with the naming conventions.

## References

cs231n.github.io/transfer-learning/

keras.io

https://github.com/fchollet/keras

Let me know your feedback and interesting things you were able to do with transfer learning.

My Github profile: https://github.com/Prakashvanapalli