



STEVENS INSTITUTE OF TECHNOLOGY

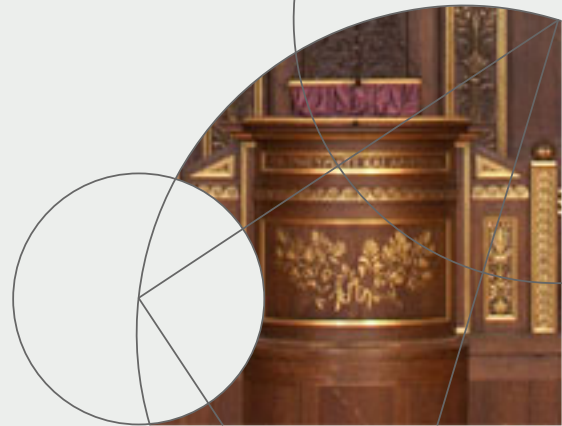


Data Acquisition and Processing II: Deep Learning @ Transfer Learning

Rensheng Wang,

rwang1@stevens.edu

Dept. of Electrical and Computer Engineering
Stevens Institute of Technology



Transfer Learning

❑ What is transfer learning?

It 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.



Transfer Learning

- ☐ My Experiences:

”DON'T TRY TO BE AN HERO”

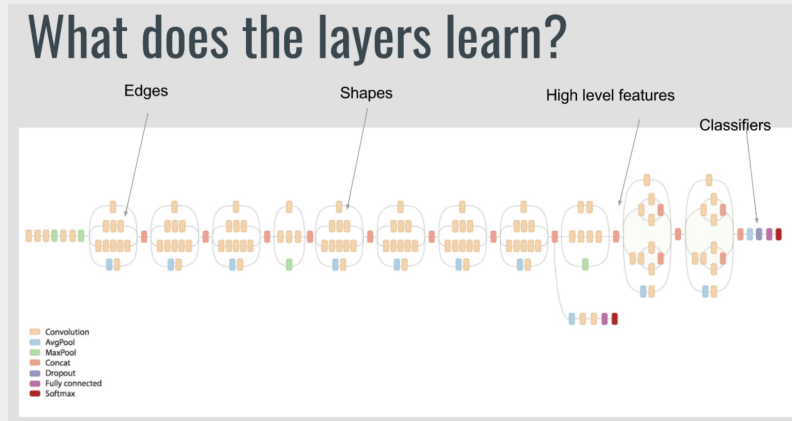
~ Andrej Karapathy (Director of AI and Autopilot Vision at Tesla)

- ☐ Most of the Computer Vision Problems we faced do not 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 (e.g., Inception V3 Google Research).
- These trained networks are generally helpful in solving other computer vision problems. Let us have a look at how to do transfer learning using Keras and various cases in Transfer learning.



Transfer Learning : Simple Implementation Using Keras

■ Sample Code:

```
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense,
GlobalAveragePooling2D
from keras import backend as k
from keras.callbacks import ModelCheckpoint, LearningRateScheduler,
TensorBoard, EarlyStopping
```



Transfer Learning

■ Sample Code:

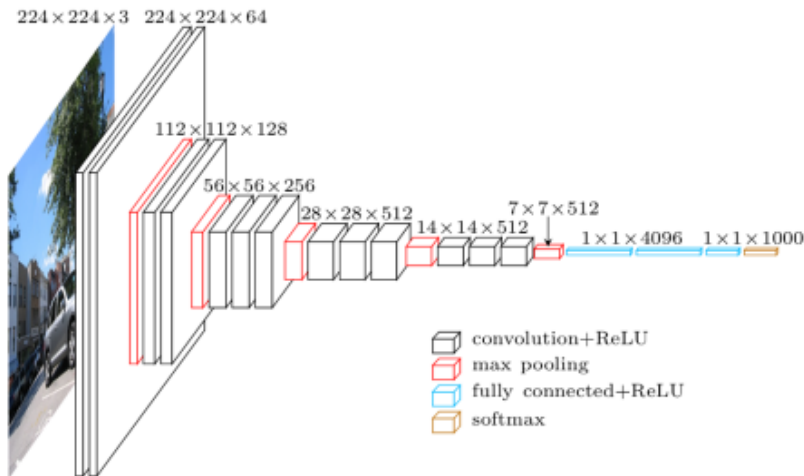
```
img_width, img_height = 256, 256
train_data_dir = "data/train"
validation_data_dir = "data/val"
nb_train_samples = 4125
nb_validation_samples = 466
batch_size = 16
epochs = 50
```



Argument: `include_top`

Sample Code:

```
model = applications.VGG19(weights = "imagenet", \
    include_top=False, input_shape = (img_width, img_height, 3))
```



Argument: include_top

Sample Code:

```
model = applications.VGG19(weights = "imagenet", \
    include_top=False, input_shape = (img_width, img_height, 3))
```

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	(None, 256, 256, 3)	0
<hr/>		
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1792
<hr/>		
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36928
<hr/>		
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0
<hr/>		
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73856
<hr/>		
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147584
<hr/>		
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0
<hr/>		
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295168
<hr/>		
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590080
<hr/>		
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590080
<hr/>		
block3_conv4 (Conv2D)	(None, 64, 64, 256)	590080
<hr/>		
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0
<hr/>		
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1180160

block4_conv2 (Conv2D)	(None, 32, 32, 512)	2359808
<hr/>		
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2359808
<hr/>		
block4_conv4 (Conv2D)	(None, 32, 32, 512)	2359808
<hr/>		
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0
<hr/>		
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2359808
<hr/>		
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
<hr/>		
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
<hr/>		
block5_conv4 (Conv2D)	(None, 16, 16, 512)	2359808
<hr/>		
block5_pool (MaxPooling2D)	(None, 8, 8, 512)	0

```
=====
Total params: 20,024,384.0
Trainable params: 20,024,384.0
Non-trainable params: 0.0
=====
```



Transfer Learning

Sample Code:

```
# Freeze the layers which you don't want to train.  
# Here we freeze the first 5 layers.  
for layer in model.layers[:5]:  
    layer.trainable = False  
  
#Adding custom Layers  
x = model.output  
x = Flatten()(x)  
x = Dense(1024, activation="relu")(x)  
x = Dropout(0.5)(x)  
x = Dense(1024, activation="relu")(x)  
predictions = Dense(16, activation="softmax")(x)  
  
# creating the final model  
model_final = Model(input = model.input, output = predictions)
```



Transfer Learning

Sample Code:

```
# compile the model
model_final.compile(loss = "categorical_crossentropy", \
                    optimizer = optimizers.SGD(lr=0.0001, momentum=0.9), \
                    metrics=["accuracy"])

# Initiate the train and test generators with data Augmentation
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    horizontal_flip = True,
    fill_mode = "nearest",
    zoom_range = 0.3,
    width_shift_range = 0.3,
    height_shift_range=0.3,
    rotation_range=30)
```



Transfer Learning

Sample Code:

```
# Initiate the train and test generators with data Augmentation
test_datagen = ImageDataGenerator(
    rescale = 1./255,
    horizontal_flip = True,
    fill_mode = "nearest",
    zoom_range = 0.3,
    width_shift_range = 0.3,
    height_shift_range=0.3,
    rotation_range=30)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size = (img_height, img_width),
    batch_size = batch_size,
    class_mode = "categorical")
```



Transfer Learning

Sample Code:

```
validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size = (img_height, img_width),
    class_mode = "categorical")

# Save the model according to the conditions
checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc', \
    verbose=1, save_best_only=True, save_weights_only=False, \
    mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, \
    patience=10, verbose=1, mode='auto')
```



Transfer Learning

❏ Sample Code:

```
# Train the model
model_final.fit_generator(
    train_generator,
    samples_per_epoch = nb_train_samples,
    epochs = epochs,
    validation_data = validation_generator,
    nb_val_samples = nb_validation_samples,
    callbacks = [checkpoint, early])
```



Transfer Learning

□ 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
- ② New dataset is large and similar to the original dataset
- ③ New dataset is large and very different from the original dataset
- ④ New dataset is small but very different from the original dataset



Case 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)
```



Case 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
```



Case 3: 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 initializations or use the pre-trained network weights as initializers. 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.



Case 4: 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 ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras.layers import Input, Conv2D, MaxPooling2D
from keras import backend as k
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, \
TensorBoard, EarlyStopping

img_width, img_height = 256, 256
```



Case 4: New dataset is small but very different from the original dataset


□ Sample code:

```
### Build the network
img_input = Input(shape=(256, 256, 3))
x = Conv2D(64, (3, 3), activation='relu', padding='same', \
          name='block1_conv1')(img_input)
x = Conv2D(64, (3, 3), activation='relu', padding='same', \
          name='block1_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same', \
          name='block2_conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', \
          name='block2_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
```



Case 4: New dataset is small but very different from the original dataset

 Sample code:

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

model.summary()
```

```

"""
Layer (type)                 Output Shape              Param #
=====
input_1 (InputLayer)        (None, 256, 256, 3)      0

block1_conv1 (Conv2D)        (None, 256, 256, 64)     1792
block1_conv2 (Conv2D)        (None, 256, 256, 64)     36928
block1_pool (MaxPooling2D)   (None, 128, 128, 64)     0
block2_conv1 (Conv2D)        (None, 128, 128, 128)    73856
block2_conv2 (Conv2D)        (None, 128, 128, 128)    147584
block2_pool (MaxPooling2D)   (None, 64, 64, 128)     0
=====
Total params: 260,160.0
Trainable params: 260,160.0
Non-trainable params: 0.0
"""

```



Case 4: New dataset is small but very different from the original dataset

□ Sample code:

```
layer_dict = dict([(layer.name, layer) for layer in model.layers])  
[layer.name for layer in model.layers]  
"""  
['input_1',  
 'block1_conv1',  
 'block1_conv2',  
 'block1_pool',  
 'block2_conv1',  
 'block2_conv2',  
 'block2_pool']  
"""
```



Case 4: New dataset is small but very different from the original dataset

□ Sample code:

```
import h5py
weights_path = 'vgg19_weights.h5'
# ('https://github.com/fchollet/deep-learning-models/releases/\
#      download/v0.1/vgg19_weights_tf_dim_ordering_tf_kernels.h5')
model.load_weights(weights_path, by_name=True)

layer_names = [layer.name for layer in model.layers]

layer_count=0;
for layer in model.layers:
    weights = layer.get_weights()
    layer_count=layer_count+1
    print("[INFO] Model Layer Configuration with respect to each layer weights as 1")
print("[INFO] The total number layers is : " + str(layer_count))
```

