06_transfer_learning

September 29, 2021

1 How to further train a pre-trained model

We will demonstrate how to freeze some or all of the layers of a pre-trained model and continue training using a new fully-connected set of layers and data with a different format.

1.1 Imports & Settings

[116]:
```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from pathlib import Path

from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16
from keras.layers import Dense, Flatten, Dropout
from keras.models import Sequential, Model
from keras.callbacks import ModelCheckpoint, TensorBoard
import matplotlib.pyplot as plt
%matplotlib inline
```

1.2 Load Dog Dataset

Before running the code cell below, download the dataset of dog images here.

[68]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

[69]:
```
cifar10_labels = {0: 'airplane',
                  1: 'automobile',
                  2: 'bird',
                  3: 'cat',
                  4: 'deer',
                  5: 'dog',
                  6: 'frog',
                  7: 'horse',
                  8: 'ship',
                  9: 'truck'}
```

[70]:
```
num_classes = len(cifar10_labels)
```

[71]:
```
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

[72]:
```
# X_train, X_valid = X_train[5000:], X_train[:5000]
# y_train, y_valid = y_train[5000:], y_train[:5000]
```

1.3 Obtain the VGG-16 Bottleneck Features

We use the VGG16 weights, pre-trained on ImageNet with the much smaller 32 x 32 CIFAR10 data. Note that we indicate the new input size upon import and set all layers to not trainable:

[118]:
```
vgg16 = VGG16(include_top=False, input_shape =X_train.shape[1:])
vgg16.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
```

```
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

1.4 Freeze model layers

1.4.1 Selectively freeze layers

[120]:
```python
for layer in vgg16.layers:
    layer.trainable = False
```

[98]:
```python
vgg16.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
```

```
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

1.4.2 Add new layers to model

We use Keras' functional API to define the vgg16 output as input into a new set of fully-connected layers like so:

[99]:
```
#Adding custom Layers
x = vgg16.output
x = Flatten()(x)
x = Dense(512, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(256, activation="relu")(x)
predictions = Dense(10, activation="softmax")(x)
```

We define a new model in terms of inputs and output, and proceed from there on as before:

[100]:
```
transfer_model = Model(inputs = vgg16.input,
                       outputs = predictions)
```

[101]:
```
transfer_model.compile(loss = 'categorical_crossentropy',
                       optimizer = 'Adam',
                       metrics=["accuracy"])
```

[102]:
```
validation_split = .1
```

We use a more elaborate ImageDataGenerator that also defines a validation_split:

[103]:
```
datagen = ImageDataGenerator(
        rescale=1. / 255,
        horizontal_flip=True,
        fill_mode='nearest',
        zoom_range=0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        rotation_range=30,
        validation_split=validation_split)
```

[104]:
```
batch_size =32
epochs = 10
```

We define both train- and validation generators for the fit method:

[105]:
```
train_generator = datagen.flow(X_train,
                               y_train,
                               subset='training')
val_generator = datagen.flow(X_train,
                             y_train,
                             subset='validation')
```

[108]:
```
vgg16_path = 'models/cifar10.transfer.vgg16.weights.best.hdf5'
checkpointer = ModelCheckpoint(filepath=vgg16_path,
                               verbose=1,
                               save_best_only=True)
```

And now we proceed to train the model:

[109]:
```
transfer_model.fit_generator(train_generator,
                             steps_per_epoch=X_train.shape[0] // batch_size,
                             epochs=epochs,
                             validation_data=val_generator,
                             validation_steps=(X_train.shape[0] * .2) //
 ↪batch_size,
                             callbacks=[checkpointer],
                             verbose=1)
```

```
Epoch 1/10
1562/1562 [==============================] - 16s 10ms/step - loss: 1.5325 - acc:
0.4553 - val_loss: 1.3096 - val_acc: 0.5438

Epoch 00001: val_loss improved from inf to 1.30961, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 2/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.3717 - acc:
0.5138 - val_loss: 1.2726 - val_acc: 0.5532
```

```
Epoch 00002: val_loss improved from 1.30961 to 1.27260, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 3/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.3253 - acc:
0.5339 - val_loss: 1.2515 - val_acc: 0.5591

Epoch 00003: val_loss improved from 1.27260 to 1.25149, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 4/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.3060 - acc:
0.5410 - val_loss: 1.2249 - val_acc: 0.5715

Epoch 00004: val_loss improved from 1.25149 to 1.22492, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 5/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.2715 - acc:
0.5509 - val_loss: 1.2011 - val_acc: 0.5766

Epoch 00005: val_loss improved from 1.22492 to 1.20108, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 6/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.2526 - acc:
0.5595 - val_loss: 1.1950 - val_acc: 0.5868

Epoch 00006: val_loss improved from 1.20108 to 1.19496, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 7/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.2387 - acc:
0.5635 - val_loss: 1.1926 - val_acc: 0.5783

Epoch 00007: val_loss improved from 1.19496 to 1.19262, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 8/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.2249 - acc:
0.5678 - val_loss: 1.1779 - val_acc: 0.5948

Epoch 00008: val_loss improved from 1.19262 to 1.17794, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 9/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.2177 - acc:
0.5700 - val_loss: 1.1687 - val_acc: 0.5927

Epoch 00009: val_loss improved from 1.17794 to 1.16873, saving model to
models/cifar10.transfer.vgg16.weights.best.hdf5
Epoch 10/10
1562/1562 [==============================] - 15s 10ms/step - loss: 1.2042 - acc:
0.5739 - val_loss: 1.1826 - val_acc: 0.5862
```

```
Epoch 00010: val_loss did not improve from 1.16873
```

[111]:
```
# load the weights that yielded the best validation accuracy
transfer_model.load_weights(vgg16_path)
```

[112]:
```
transfer_model.evaluate(X_test, y_test)[1]
```

```
10000/10000 [==============================] - 2s 236us/step
```

[112]:
```
0.3587
```

1.4.3 Test Set Classification Accuracy

10 epochs lead to a mediocre test accuracy of 35.87% because the assumption that image features translate to so much smaller images is somewhat questionable but it serves to illustrate the workflow.

[114]:
```
# get index of predicted dog breed for each image in test set
vgg16_predictions = np.argmax(transfer_model.predict(X_test), axis=1)
```

[115]:
```
test_accuracy = np.sum(vgg16_predictions==np.argmax(y_test, axis=1))/
 ↪len(vgg16_predictions)
print('\nTest accuracy: %.4f%%' % test_accuracy)
```

```

Test accuracy: 0.3587%
```