Skills Network Labs

File Edit View Run Kernel Git Tabs Settings Help

DL0321EN-3-1-Pretrained-

IBM Developer SKILLS NETWORK

Pre-Trained Models

Objective

In this lab, you will learn how to leverage pre-trained models to build image classifiers instead of building a model from scratch.

Table of Contents

1. Import Libraries and Packages 2. Download Data 3. Define Global Constants 4. Construct ImageDataGenerator Instances 5. Compile and Fit Model

Import Libraries and Packages

Let's start the lab by importing the libraries that we will be using in this lab.

First, we will import the ImageDataGenerator module since we will be leveraging it to train our model in batches.

```
from keras.preprocessing.image import ImageDataGenerator
```

In this lab, we will be using the Keras library to build an image classifier, so let's download the Keras library.

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

Finally, we will be leveraging the ResNet50 model to build our classifier, so let's download it as well.

```
from keras.applications import ResNet50
from keras.applications.resnet50 import preprocess_input
```

Download Data

For your convenience, I have placed the data on a server which you can retrieve easily using the **wget** command. So let's run the following line of code to get the data. Given the large size of the image dataset, it might take some time depending on your internet speed.

```
## get the data
!wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/concrete_data_week3.zip
```

And now if you check the left directory pane, you should see the zipped file *concrete_data_week3.zip* appear. So, let's go ahead and unzip the file to access the images. Given the large number of images in the dataset, this might take a couple of minutes, so please be patient, and wait until the code finishes running.

```
!unzip concrete_data_week3.zip
```

Now, you should see the folder *concrete_data_week3* appear in the left pane. If you open this folder by double-clicking on it, you will find that it contains two folders: *train* and *valid*. And if you explore these folders, you will find that each contains two subfolders: *positive* and *negative*. These are the same folders that we saw in the labs in the previous modules of this course, where *negative* is the negative class and it represents the concrete images with no cracks and *positive* is the positive class and it represents the concrete images with cracks.

Important Note: There are thousands and thousands of images in each folder, so please don't attempt to double click on the *negative* and *positive* folders. This may consume all of your memory and you may end up with a **50*** error. So please **DO NOT DO IT**.

Define Global Constants

Here, we will define constants that we will be using throughout the rest of the lab.

1. We are obviously dealing with two classes, so *num_classes* is 2.
2. The ResNet50 model was built and trained using images of size (224 x 224). Therefore, we will have to resize our images from (227 x 227) to (224 x 224).
3. We will training and validating the model using batches of 100 images.

```
num_classes = 2

image_resize = 224

batch_size_training = 100
batch_size_validation = 100
```

Construct ImageDataGenerator Instances

In order to instantiate an ImageDataGenerator instance, we will set the **preprocessing_function** argument to *preprocess_input* which we imported from **keras.applications.resnet50** in order to preprocess our images the same way the images used to train ResNet50 model were processed.

```
data_generator = ImageDataGenerator(
    preprocessing_function=preprocess_input,
)
```

Next, we will use the *flow_from_directory* method to get the training images as follows:

```
train_generator = data_generator.flow_from_directory(
    'concrete_data_week3/train',
    target_size=(image_resize, image_resize),
    batch_size=batch_size_training,
    class_mode='categorical')
```

Your Turn: Use the *flow_from_directory* method to get the validation images and assign the result to **validation_generator**.

```
## Type your answer here
```

```
Double-click **here** for the solution.

<!-- The correct answer is:
validation_generator = data_generator.flow_from_directory(
    'concrete_data_week3/valid',
    target_size=(image_resize, image_resize),
    batch_size=batch_size_validation,
    class_mode='categorical')
-->
```

Build, Compile and Fit Model

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. Learn more here.

In this section, we will start building our model. We will use the Sequential model class from Keras.

```
model = Sequential()
```

Next, we will add the ResNet50 pre-trained model to out model. However, note that we don't want to include the top layer or the output layer of the pre-trained model. We actually want to define our own output layer and train it so that it is optimized for our image dataset. In order to leave out the output layer of the pre-trained model, we will use the argument *include_top* and set it to **False**.

```
model.add(ResNet50(
    include_top=False,
    pooling='avg',
    weights='imagenet',
    ))
```

Then, we will define our output layer as a **Dense** layer, that consists of two nodes and uses the **Softmax** function as the activation function.

```
model.add(Dense(num_classes, activation='softmax'))
```

You can access the model's layers using the *layers* attribute of our model object.

```
model.layers
```

You can see that our model is composed of two sets of layers. The first set is the layers pertaining to ResNet50 and the second set is a single layer, which is our Dense layer that we defined above.

You can access the ResNet50 layers by running the following:

```
model.layers[0].layers
```

Since the ResNet50 model has already been trained, then we want to tell our model not to bother with training the ResNet part, but to train only our dense output layer. To do that, we run the following.

```
model.layers[0].trainable = False
```

And now using the *summary* attribute of the model, we can see how many parameters we will need to optimize in order to train the output layer.

```
model.summary()
```

Next we compile our model using the **adam** optimizer.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Before we are able to start the training process, with an ImageDataGenerator, we will need to define how many steps compose an epoch. Typically, that is the number of images divided by the batch size. Therefore, we define our steps per epoch as follows:

```
steps_per_epoch_training = len(train_generator)
steps_per_epoch_validation = len(validation_generator)
num_epochs = 2
```

Finally, we are ready to start training our model. Unlike a conventional deep learning training were data is not streamed from a directory, with an ImageDataGenerator where data is augmented in batches, we use the **fit_generator** method.

[ ]:
```
fit_history = model.fit_generator(
    train_generator,
    steps_per_epoch=steps_per_epoch_training,
    epochs=num_epochs,
    validation_data=validation_generator,
    validation_steps=steps_per_epoch_validation,
    verbose=1,
)
```

Now that the model is trained, you are ready to start using it to classify images.

Since training can take a long time when building deep learning models, it is always a good idea to save your model once the training is complete if you believe you will be using the model again later. You will be using this model in the next module, so go ahead and save your model.

[ ]:
```
model.save('classifier_resnet_model.h5')
```

Now, you should see the model file *classifier_resnet_model.h5* apprear in the left directory pane.

Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational.

This notebook is part of a course on **Coursera** called *AI Capstone Project with Deep Learning*. If you accessed this notebook outside the course, you can take this course online by clicking here.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-18	2.0	Shubham	Migrated Lab to Markdown and added to course repo in GitLab

---