




Transfer learning VGG16

 Created By	
 Stakeholders	
 Status	
 Type	
 Created	@May 5, 2022 3:45 PM
 Last Edited Time	@May 10, 2022 3:40 PM
 Last Edited By	

Classification of COVID-19 with cough COVID-19 no cough sounds

Problem: Classification of cough sounds into two classes:

First- cough sounds from COVID-19 positive users who have cough as a symptom, and

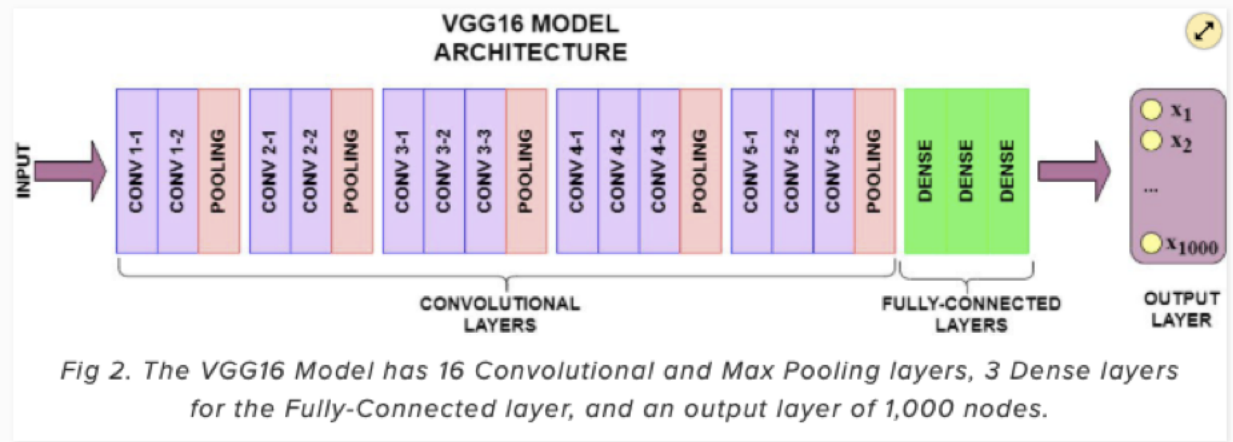
Second- cough sounds from COVID-19 positive users who do not have cough as a symptom.

Challenge: Both classes are close to each other as all of the users in both classes have tested positive for COVID-19, the only difference is the cough is a symptom in one class, and there is no cough as a symptom for another class.

Solution:

We have a limited data set - hence, we chose to implement a transfer learning approach using VGG16. VGG stands for visual geometry group, it is also called as OxfordNet.

VGG16 is a pre-trained CNN model having 16 convolutional layers and trained on millions of images from 1000 classes. The architecture of CNN is very simple as explained below:



Key points of VGG16 architecture:

1. It has 16 convolutional layers.
2. It always uses 3 x 3 kernel for convolution.
3. Max pool is 2 x 2
4. Trained on ImageNet dataset
5. The number of nodes in the output layer must be equal to the number of classes.
6. Another version is VGG19: it has 19 convolutional layers.
7. It takes input sizes - 224, 224, 3 for images.

Practical Approach/ Code in Python:

The steps I followed to implement VGG16 on my dataset are explained below:

1. Preparation of dataset directories:

Create a folder named "dataset" (you may use any other name as well).

Inside this folder create 3 folders: training_dataset, test_dataset, and Validation_dataset.

Inside these 3 folders, we must have two folders: cough, and no-cough (for each folder), divide the whole dataset into training_set (70%), test_set (20%), and validation_set (10%).

2. Import necessary libraries:

I have imported many libraries before the coding part, and a few libraries as we proceed in the coding steps.

```
import numpy as np
import keras
from keras.applications import vgg16, vgg19
from keras.preprocessing import image
from keras.applications.imagenet_utils import preprocess_input, decode_predictions
import scipy.misc
```

I have used the Keras library for the implementation of VGG16.

The pre-trained networks “VGG16, VGG19” is imported from the “applications” module of “Keras”.

In the next line, I have imported the “image” module from the “preprocessing” module of “Keras”.

Next, I have imported script.misc (scipy is a science python library used for signal processing).

3. Import the model and load the pre-defined weights. In VGG16 the weights are pre-defined as ‘imagenet’.

And check the summary of the layers of the VGG16 network.

```
model = vgg16.VGG16(include_top=True, weights='imagenet')
model.summary()
```

The VGG16 model is pre-trained on the classes, so if we use “include_top = True”, we need to retrain it on all the classes. Which will add computational cost. Hence, it’s better to set “include_top = False”.

4. Generate the spectrograms of the dataset

Because the dataset we have is the audio (.wav) files of cough sounds, we need to transfer our audio signal into an intermediate representation, in this case, it is its corresponding time-frequency representation i.e. spectrogram.

```
import librosa
import glob
import os.path
import matplotlib.pyplot as plt
from librosa import display

t=[]
sampling_rate=[]
path = '/Users/garimasharma/Downloads/data-to-share-covid-19-sounds/KDD_paper_data/covidandroidnocough/cough/'
for filename in glob.glob(os.path.join(path, '*.wav')):
    y, sr = librosa.load(filename)
    #y = y / tf.int16.max
    sampling_rate.append(sr)
    t.append(y)
    y_freq_domain = librosa.stft(y,n_fft=512, hop_length=256)
    y_fft_abs = np.abs(y_freq_domain)
    # display a spectrogram
    fig, ax = plt.subplots()
    img = librosa.display.specshow(librosa.amplitude_to_db(y_fft_abs,ref=np.max),y_axis='log', x_axis='time', ax=ax)
    file = str(filename) + '.png'
    plt.savefig(str(file))
```

In this set of code, we have imported the following libraries:

- Librosa: for audio signal processing
- Glob: for batch reading of files from a directory
- Matplotlib.pyplot: peplot (python plots) for data visualization

5. Divide the spectrogram images into training_set, test_set, and Validation set as explained in step 1.

6. After creating the dataset in various folders, now we will generate paths to these directories. Then see the number of spectrogram images in all folders

```
train_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/training_set'
test_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/test_set'
validation_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/validation_set'
```

```

train_covid_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/training_set/train_covid'
test_covid_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/test_set/test_covid'
validation_covid_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/validation_set/validation_covid'
train_nocovid_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/training_set/train_nocovid'
test_nocovid_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/test_set/test_nocovid'
validation_nocovid_dir = '/Users/garimasharma/Downloads/covid_transferlearning/dataset/validation_set/validation_nocovid'

```

```

num_covid_train = len(os.listdir(train_covid_dir))
num_covid_test = len(os.listdir(test_covid_dir))
num_covid_val = len(os.listdir(validation_covid_dir))

num_nocovid_train = len(os.listdir(train_nocovid_dir))
num_nocovid_test = len(os.listdir(test_nocovid_dir))
num_nocovid_val = len(os.listdir(validation_nocovid_dir))

print(f"The number of train covid: {num_covid_train}")
print(f"The number of test sampling in covid: {num_covid_test}")
print(f"The number of val samples in covid: {num_covid_val}")
print(f"The number of train sampling in no covid cough: {num_nocovid_train}")
print(f"The number of test noncovid: {num_nocovid_test}")
print(f"The number of val sampls in noncovid: {num_nocovid_val}")

```

The output of this step looks like this:

```

In [26]: runcell(4, '/Users/garimasharma/Downloads/Python basic codes/transfer_learning_vgg16.py')
The number of train covid: 138
The number of test sampling in covid: 26
The number of val samples in covid: 20
The number of train sampling in no covid cough: 31
The number of test noncovid: 11
The number of val sampls in noncovid: 10

```

7. Define the size of the input image and batch_size. The input_size is always 224, 224 for VGG16.

```

image_shape = 224
batch_size = 10

```

8. Reshape the images as per VGG16 specifications, for this we have used the module "ImageDataGenerator" from the image preprocessing of Keras.

```

from keras.preprocessing.image import ImageDataGenerator

train_image_generation = ImageDataGenerator(rescale=(1./255) )
train_data_generation = train_image_generation.flow_from_directory(train_dir,
                                                                    target_size=(image_shape, image_shape),
                                                                    batch_size=batch_size,
                                                                    class_mode='binary')

test_image_generation = ImageDataGenerator(rescale=(1./255))
test_data_generation = test_image_generation.flow_from_directory(test_dir,
                                                                target_size=(image_shape, image_shape),
                                                                batch_size=batch_size,
                                                                class_mode='binary')

val_image_generation = ImageDataGenerator(rescale=(1./255))
val_data_generation = val_image_generation.flow_from_directory(validation_dir,
                                                             target_size=(image_shape, image_shape),
                                                             batch_size=batch_size,
                                                             class_mode='binary')

```

9. Now we need to freeze the training layers of VGG16 because it is already trained.

```
for layer in model.layers:
    print(layer.name)
    layer.trainable = False
```

10. As now all the layers of VGG16 are frozen, we need to modify the last layers as per our data set structure. The last layer of the VGG16 is 'block5_pool'. I have added an extra pooling layer, dense layer, dropout layer, and an output layer to the existing architecture of VGG16. The output layer is a dense layer with 2 nodes (as it is a binary classification problem), and sigmoid activation. If it would have been a multi-class problem, then I would have used softmax activation.

```
last_layer = model.get_layer('block5_pool')
last_output = last_layer.output
x = keras.layers.GlobalMaxPooling2D()(last_output)
x = keras.layers.Dense(512, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
x = keras.layers.Dense(2, activation='sigmoid')(x)
```

11. Now we need to merge the original VGG layers with our customized layers.

```
import tensorflow as tf
model = tf.keras.Model(model.input, x)
```

12. Before training our model on the dataset, we will compile it.

```
model.compile(optimizer='adam', loss=tf.keras.losses.sparse_categorical_crossentropy, metrics=['acc'])
```

I have used 'Adam' as an optimizer, sparse categorical cross-entropy as loss function, and accuracy rate as the metric. If it would have been a multi-class problem, then I would have chosen categorical cross-entropy as the loss function.

13. Now, train our model for 50 epochs.

```
total_train = num_covid_train + num_noncovid_train
total_validation = num_covid_val + num_noncovid_val

vgg_classifier = model.fit(train_data_generation,
    steps_per_epoch=(total_train//batch_size),
    epochs = 50,
    validation_data=val_data_generation,
    validation_steps=(total_validation//batch_size),
    batch_size = batch_size, verbose = 1)
```

The output of this step looks like this:

```
In [39]: runcell(12, '/Users/garimasharma/Downloads/Python basic codes/transfer_learning_vgg16.py')
Epoch 1/50
16/16 [=====] - 48s 3s/step - loss: 0.4733 - acc: 0.8165 - val_loss: 0.6968 -
val_acc: 0.6667
Epoch 2/50
16/16 [=====] - 53s 3s/step - loss: 0.4571 - acc: 0.8000 - val_loss: 0.7119 -
val_acc: 0.6667
Epoch 3/50
16/16 [=====] - 51s 3s/step - loss: 0.4681 - acc: 0.8101 - val_loss: 0.8907 -
val_acc: 0.6667
```

14. Next step is to test our model on the test set. It is done as:

```
result = model.evaluate(test_data_generation, batch_size=batch_size)
print("test_loss, test accuracy", result)
```

15. the last step is to plot the graph between epochs and the accuracy rate for training and validation sets. If there is a significant gap between training and validation accuracies, this means that the model is highly overfitted (the main reason for overfitting is an unbalanced dataset). If our validation accuracy rate is closer to the training accuracy rate, that means our model is balanced and performing well.

```
acc=vgg_classifier.history['acc'] ##getting accuracy of each epochs
epochs_=range(0,50)
plt.plot(epochs_,acc,label='training accuracy')
plt.xlabel('no of epochs')
plt.ylabel('accuracy')

acc_val=vgg_classifier.history['val_acc'] ##getting validation accuracy of each epochs
plt.scatter(epochs_,acc_val,label="validation accuracy")
plt.title("no of epochs vs accuracy")
plt.legend()
```