LUNG DISEASE DETECTION Using VGG16 State-of-the-Art Algorithm

Description

This project as the title suggests focuses on the detection of any kind of lung disease or no disease when certain image(s) is/are given as inputs.

Here we have used the State-of-the-Art convolutional neural network model VGG16 using transfer learning. Transfer Learning is used as we directly use the weights and the kernels of the pretrained VGG16 model. Only thing what changes here is the final output layer which is replaced according to our number of output classes. (3 in our case)

VGG16 Model and its Advantages of use

The VGG16 Model is state-of-the-art model which is pretrained on the ImageNet dataset and has almost 138 million parameters.

The model has 16 layers of which 13 layers are the convolutional layers (5 of them have attached max pooling layers) and the remaining 3 are Fully Connected Dense layers. The last Fully Connected Dense layer is customized according to our output classes (3 in this case) because we have to use only the weights and the kernel of the architecture.

There are 13 convolutional layers out of them 5 layers have an attached Max Pooling layer.

The hyperparameters of these layers are consistent. So, each layer will have the same value of hyperparameters.

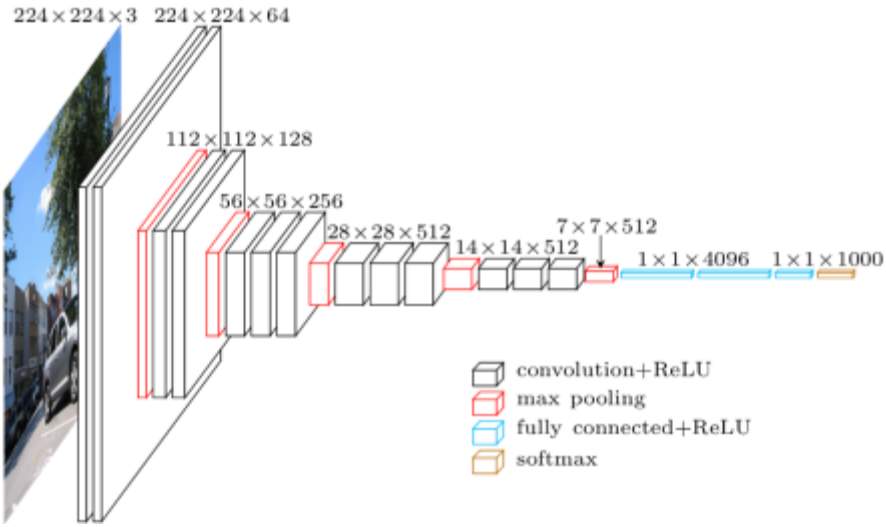
Filter size will be 3X3 while that of max pooling layer is 2X2, padding will be ‘same’ i.e., zero padding followed but not for max pooling layer, strides for convolutional layer is 1 while that of the max pooling layer is 2.

Moreover, activation function used for convolutional layers is ReLU

Number of kernels for each layer will increase by exponent of 2 starting with 64 for first two layers then 128 for the next two, then 256 for the next two and last 7 have 512 units.

Coming to the fully connected dense layers we have 4096 units in each layer and activation function is ReLU.

The final output layer instead of having 1000 units will only have 3 units as our classes are present with SoftMax activation function.



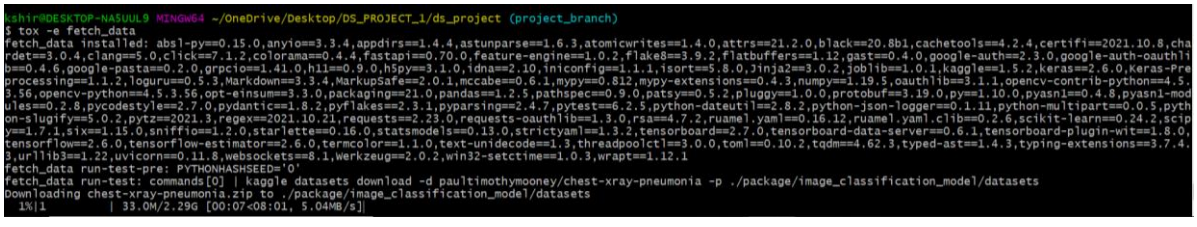
Advantages of VGG16 model can be:

- This model is small in size (only 16 layers) but efficient in terms of accuracy for predictions with respect to its size.
- Its biggest advantage is that the layers present in the model have consistent parameters (i.e., for each layer of the same type we have same value of the parameters).
  So, it focuses more on the layers rather than on the other hyper parameters of these layers like filter-size, number of units, strides, etc.
- This helps is improving the consistency and efficiency of the model for predictions.
- E.g., is that for convolutional layers the stride of 1 or filter size of 3X3 is fixed for all conv. Layers.
- Present in KERAS applications and so required no separate downloading.

Using the repository (Use the commands given in the Readme file)

1] For fetching of the dataset from Kaggle:

Active Kaggle account is required. The dataset used in this repository is Chest X-Ray Images (Pneumonia) available on the following
link: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia



Fetching the data from Kaggle

Training Dataset we have following number of images for each type (this is not an extreme imbalanced dataset)

```
pneumonia_bacteria    2538
normal                1349
pneumonia_virus       1345
```

After getting the zip file unzip it and get ready to use this dataset. The images in the dataset will look something like this:

The normal chest X-ray (left panel) depicts clear lungs without any areas of abnormal opacification in the image. Bacterial pneumonia (middle) typically exhibits a focal lobar consolidation, in this case in the right upper lobe (white arrows), whereas viral pneumonia (right) manifests with a more diffuse ‘‘interstitial’’ pattern in both lungs.

So, the live data for uploading needs to be similar to the form of above images.

2] For training the MODEL:

First of the images that we have fetched from Kaggle we need to tabulate them in the form of a table consisting of image path and the following class it belongs to namely (No Disease i.e., Normal, Pneumonia Bacteria, and Pneumonia Virus)

```
In [5]: train_df
```

Out[5]:

	image_name	label
0	train1/normal_train\normal_train (1).jpeg	normal
1	train1/normal_train\normal_train (10).jpeg	normal
2	train1/normal_train\normal_train (100).jpeg	normal
3	train1/normal_train\normal_train (1000).jpeg	normal
4	train1/normal_train\normal_train (1001).jpeg	normal
...	...	...
5227	val1/pneumonia_val\person1949_bacteria_4880.jpeg	pneumonia_bacteria
5228	val1/pneumonia_val\person1950_bacteria_4881.jpeg	pneumonia_bacteria
5229	val1/pneumonia_val\person1951_bacteria_4882.jpeg	pneumonia_bacteria
5230	val1/pneumonia_val\person1952_bacteria_4883.jpeg	pneumonia_bacteria
5231	val1/pneumonia_val\person1954_bacteria_4886.jpeg	pneumonia_bacteria

5232 rows × 2 columns

In the research environment we get this is the type of table. Similarly, we will get in our package training.

Training the model is essentially training it on this table after converting the image names into vectors with each of size (150,150,3) and rescaling it by 255.

Create a pipeline with a customized transformer to handle big data and convert it into vector array of in this case (number of training images,150,150,3) and the VGG16 model. Set the epochs and batch size base upon computational strength of the system.

The other part of the pipeline contains customized transformer in order to convert the images to their respective quantized vectors format. Since the dataset is an example of big data what we essential do is take a dummy array of all zeros of the required size (in this case it is (number of training images,150,150,3)). Then we take one image at a time convert it into required size of (150,150,3) and rescale it by 255. Finally, we replace this array in the dummy array at the given indices. This is how we convert the entire dataset into the given vectorized format.

Then fit the pipeline on this training data and save it for further use. Remember the model and the customized transformer need to be saved separately and assembled in the pipeline when used for predictions. This is because '.pkl' extension is not compatible with '.h5' KERAS models. So, we have to separate the model and the rest of the pipeline.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 3)                 24579
=================================================================
Total params: 14,739,267
Trainable params: 24,579
Non-trainable params: 14,714,688
```

Model summary

```
51/66 [=====================>.......] - ETA: 1:39 - loss: 0.2850 - accuracy: 0.9023
52/66 [======================>......] - ETA: 1:33 - loss: 0.2840 - accuracy: 0.9023
53/66 [======================>......] - ETA: 1:26 - loss: 0.2829 - accuracy: 0.9024
54/66 [=======================>......] - ETA: 1:20 - loss: 0.2821 - accuracy: 0.9031
55/66 [========================>.....] - ETA: 1:13 - loss: 0.2835 - accuracy: 0.9023
56/66 [========================>.....] - ETA: 1:07 - loss: 0.2831 - accuracy: 0.9023
57/66 [========================>.....] - ETA: 1:00 - loss: 0.2843 - accuracy: 0.9019
58/66 [=========================>....] - ETA: 53s - loss: 0.2816 - accuracy: 0.9030
59/66 [=========================>....] - ETA: 47s - loss: 0.2795 - accuracy: 0.9036
60/66 [==========================>...] - ETA: 40s - loss: 0.2801 - accuracy: 0.9034
61/66 [==========================>...] - ETA: 33s - loss: 0.2791 - accuracy: 0.9042
62/66 [===========================>..] - ETA: 26s - loss: 0.2793 - accuracy: 0.9040
63/66 [===========================>..] - ETA: 20s - loss: 0.2786 - accuracy: 0.9045
64/66 [============================>.] - ETA: 13s - loss: 0.2788 - accuracy: 0.9043
65/66 [============================>.] - ETA: 6s - loss: 0.2805 - accuracy: 0.9034
66/66 [==============================] - ETA: 0s - loss: 0.2812 - accuracy: 0.9025
66/66 [==============================] - 566s 9s/step - loss: 0.2812 - accuracy: 0.9025 - val_loss: 1.5202 - val_accuracy: 0.2808
Pipeline(steps=[('cd', CreateDataset(target_size=150)),
                ('estimator',
                 <keras.wrappers.scikit_learn.KerasClassifier object at 0x000001500253E9D0>)])
<keras.wrappers.scikit_learn.KerasClassifier object at 0x000001500253E9D0>
<keras.wrappers.scikit_learn.KerasClassifier object at 0x000001500253E9D0>
CreateDataset(target_size=150)
[1, 2, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 2, 0, 0, 1,
0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0
, 2, 2, 2, 2, 2, 0, 1, 2, 1, 2, 0, 2, 0, 0, 1, 1, 2, 2, 2, 2, 0, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
 2, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 0, 2, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 2,
0, 1, 0, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1,
 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1, 1,
2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1]
```

Training of model with testing it on test images.

3] For running the API:

First, we build the API using fastapi module leveraging the python async.io web framework. FastApi module is compatible with pydantic module which helps in input data validation and type checking automatically using schemas. Then we run this API with the uvicorn ASGI web server on http://localhost:8001/imageclassificationform

Also, we perform logging using loguru module to display the status messages. Make sure to place the live data in the live_data sub-folder in the api folder

https://drive.google.com/file/d/1-vfWtJmDneP24zPrlfkjW1U5ARf9lQ7D/view?usp=sharing

Refer to the above video.

FOR RUNNING THE REPOSITORY REFER TO THE COMMANDS IN README FILE.