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. So, this is a very basic project which actually shows how deep learning especially convolution neural networks can be used in the medical and healthcare industry.

Here we have used the State-of-the-Art convolutional neural network model VGG16 using transfer learning.

Advantages of using Vgg16

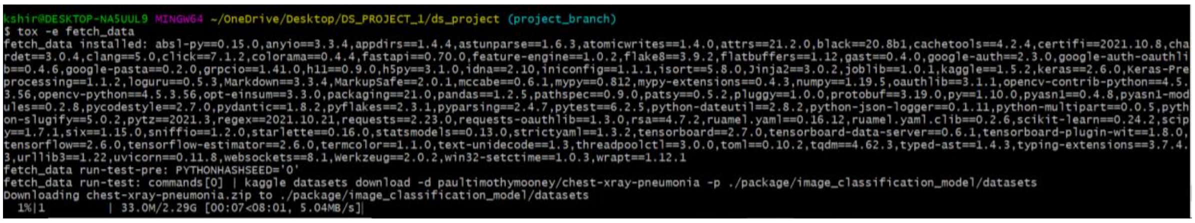
- This model is small in size (only 16 layers) but efficient in terms of accuracy for predictions with respect to its size.
- Its biggest advantage is that the layers present in the model have consistent parameters (i.e., for each layer of the same type we have same value of the parameters).
  So, it 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.

More on the layered architecture of VGG16 in the low-level document.

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

1] For fetching of the dataset from Kaggle:

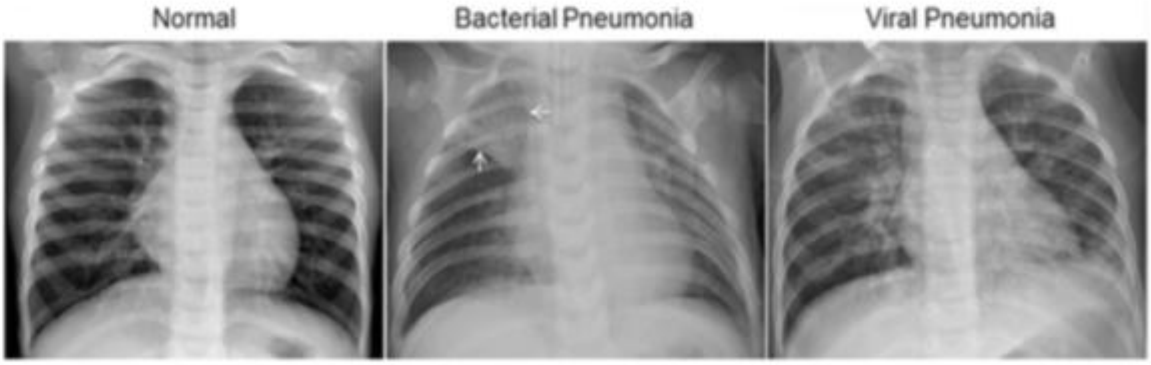
Active Kaggle account is required. The dataset used in this repository is Chest X-Ray Images (Pneumonia) available on the following link: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia



```
kshir@DESKTOP-NA5UUL9 MINGW64 ~/OneDrive/Desktop/DS_PROJECT_1/ds_project (project_branch)
$ tox -e fetch_data
fetch_data installed: absl-py==0.15.0,anyio==3.3.4,appdirs==1.4.4,astunparse==1.6.3,atomicwrites==1.4.0,attrs==21.2.0,black==20.8b1,cachetools==4.2.4,certifi==2021.10.8,cha
rdet==3.0.4,clang==5.0,click==7.1.2,colorama==0.4.4,fastapi==0.70.0,feature-engine==1.0.2,flake8==3.9.2,flatbuffers==1.12,gast==0.4.0,google-auth==2.3.0,google-auth-oauthli
b==0.4.6,google-pasta==0.2.0,grpcio==1.41.0,h11==0.9.0,h5py==3.1.0,idna==2.10,iniconfig==1.1.1,isort==5.8.0,Jinja2==3.0.2,joblib==1.0.1,kaggle==1.5.2,keras==2.6.0,Keras-Pre
processing==1.1.2,loguru==0.5.3,Markdown==3.3.4,MarkupSafe==2.0.1,mccabe==0.6.1,mypy==0.812,mypy-extensions==0.4.3,numpy==1.19.5,oauthlib==3.1.1,opencv-contrib-python==4.5.
3.56,opencv-python==4.5.3.56,opt-einsum==3.3.0,packaging==21.0,pandas==1.2.5,pathspec==0.9.0,patsy==0.5.2,pluggy==1.0.0,protobuf==3.19.0,py==1.10.0,pyasn1==0.4.8,pyasn1-mod
ules==0.2.8,pycodestyle==2.7.0,pydantic==1.8.2,pyflakes==2.3.1,pyparsing==2.4.7,pytest==6.2.5,python-dateutil==2.8.2,python-json-logger==0.1.11,python-multipart==0.0.5,pyth
on-slugify==5.0.2,pytz==2021.3,regex==2021.10.21,requests==2.23.0,requests-oauthlib==1.3.0,rsa==4.7.2,ruamel.yaml==0.16.12,ruamel.yaml.clib==0.2.6,scikit-learn==0.24.2,scip
y==1.7.1,six==1.15.0,sniffio==1.2.0,starlette==0.16.0,statsmodels==0.13.0,strictyaml==1.3.2,tensorboard==2.7.0,tensorboard-data-server==0.6.1,tensorboard-plugin-wit==1.8.0,
tensorflow==2.6.0,tensorflow-estimator==2.6.0,termcolor==1.1.0,text-unidecode==1.3,threadpoolctl==3.0.0,toml==0.10.2,tqdm==4.62.3,typed-ast==1.4.3,typing-extensions==3.7.4.
3,urllib3==1.22,uvicorn==0.11.8,websockets==8.1,Werkzeug==2.0.2,win32-setctime==1.0.3,wrapt==1.12.1
fetch_data run-test-pre: PYTHONHASHSEED='0'
fetch_data run-test: commands[0] | kaggle datasets download -d paultimothymooney/chest-xray-pneumonia -p ./package/image_classification_model/datasets
Downloading chest-xray-pneumonia.zip to ./package/image_classification_model/datasets
  1%|1          | 33.0M/2.29G [00:07<08:01, 5.04MB/s]
```

Fetching the data from Kaggle

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

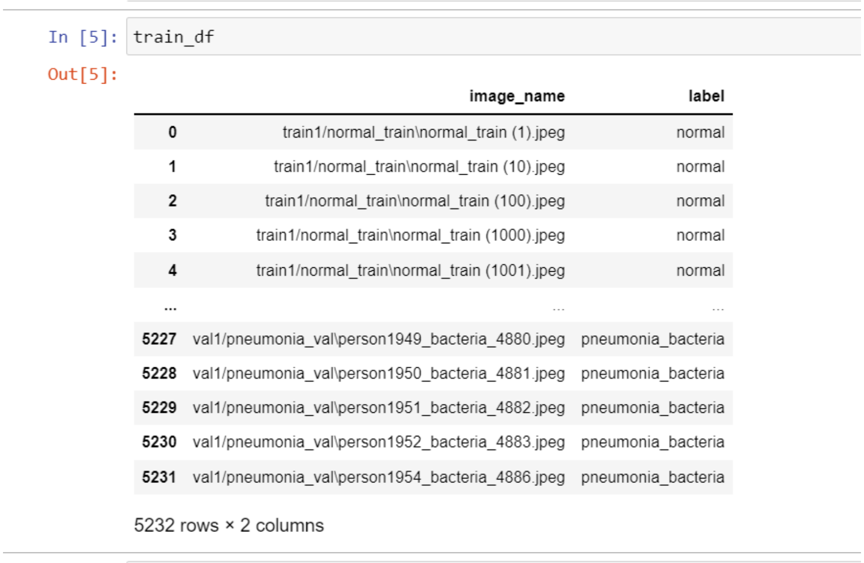


The normal chest X-ray (left panel), Bacterial pneumonia (middle), whereas viral pneumonia (right) manifests in both lungs.

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)



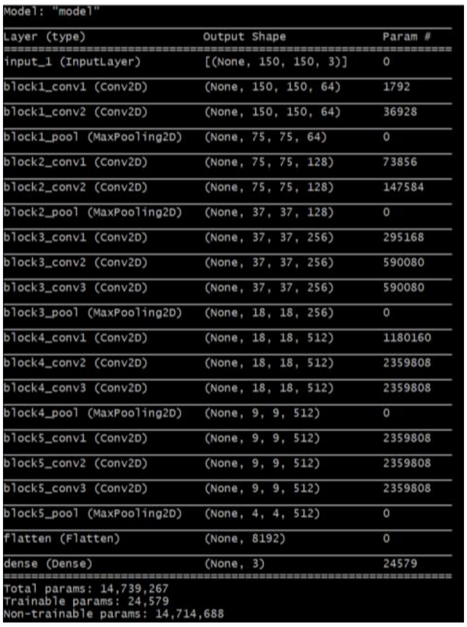
```
In [5]: train_df
Out[5]:
```

	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

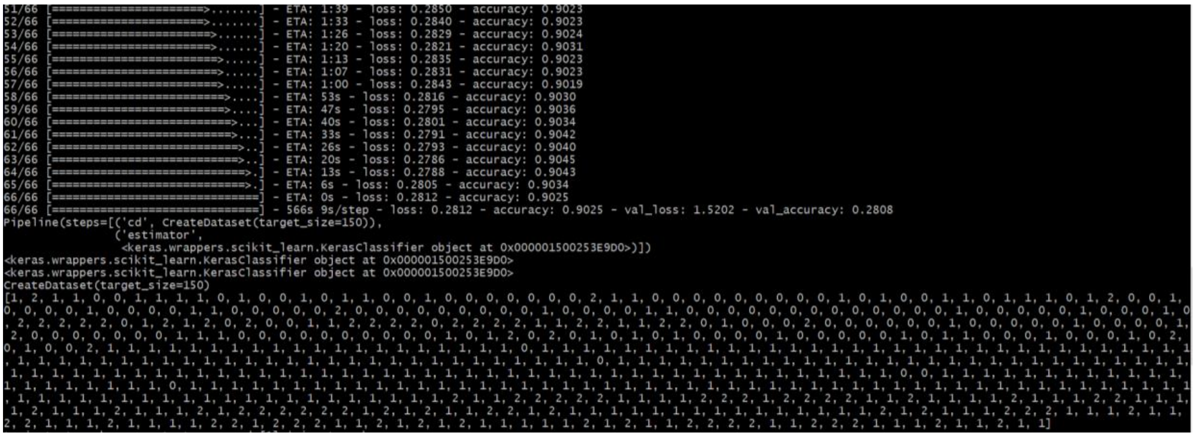
In the research environment we get this is the type of table. Similarly, we will get in our package training. Training the model is essentially training it on this table after converting the image names into vectors.

Create a pipeline with a customized transformer to handle big data and convert it into vector array and the VGG16 model. Then fit the pipeline on this training data and save it for further use. Set the epochs and batch size base upon computational strength of the system.



```
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, 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, 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.

Following are the results: Epochs=2   Batch Size= 64

On the first epoch I got a training accuracy of 0.8495 and validation accuracy of 0.1509

On the second epoch I got training accuracy of 0.9025 and validation accuracy of 0.2808

This shows that if the epochs and batch size are increased, we can achieve better results.

3] For running the API:

First, we build the API using fastapi module leveraging the python async.io web framework. Then we run this API with the uvicorn ASGI web server.

Also, we perform logging using loguru module.

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.