Lungs CT-based Classification for COVID19-vs-Non COVID19

Falguni Vivek Vasava
Registration Number : 2006617

***Abstract*— In this study, I have performed binary classification on Lungs CT-based dataset and improve the performance of the results using VGG16 Algorithm.**

***Keywords*—VGG16, CNN, COVID19 Dataset, Classification, CT-based dataset, Transfer Learning, Fine Tuning, Feature Extraction.**

I. LITERATURE REVIEW

A very versatile and intelligent AI system that use deep learning to accomplish both generative and descriptive tasks is known as a Convolutional Neural Network (CNN). CNNs are frequently used in conjunction with machine vision, which consists of video and image recognition, natural language processing, and recommender systems, and other applications (NLP) [1].

Transfer Learning

The concept of knowledge transfer (also known as learning to learn) is one of the core areas of machine learning. Transfer learning endeavours to improve prediction functions in the target task through the use of source domain and task data. As the figure-1 demonstrates, in conventional machine learning, tasks are isolated from one another, and each one has to be relearned. However, one way that existing knowledge acquired from previous tasks can be transferred to the learning process of a new task is by means of transfer learning. Various cases such as web document categorization, indoor Wi-Fi localization challenge, and sentiment classification all benefit from transfer learning. Our investigation will focus on the research field of visual transfer learning [2].

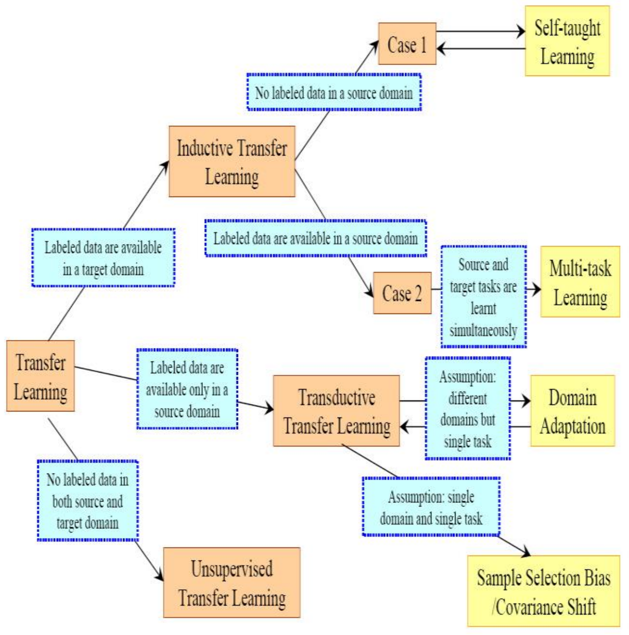

Case 1
Self-taught Learning
No labeled data in a source domain
Inductive Transfer Learning
Labeled data are available in a target domain
Labeled data are available in a source domain
Source and target tasks are learnt simultaneously
Case 2
Multi-task Learning
Transfer Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
No labeled data in both source and target domain
Assumption: single domain and single task
Unsupervised Transfer Learning
Sample Selection Bias /Covariance Shift

Figure-1: Categories of Transfer Learning [2]

There are three basic issues in transfer learning: what to transfer, how to transmit, and when to transfer. There are three kinds of transfer learning settings: unsupervised transfer learning where there are no labelled data; inductive transfer learning where the tasks are definitely different, even if there are similarities between the source and target domains; and inductive transfer learning in which tasks are identical but marginal probability distributions or feature spaces are different.

The 4 transfer learning categories are also different in the sort of knowledge they transfer: instance-transfer, parameter-transfer, feature-representation-transfer and relational-knowledge transfer.

VGG16 Algorithm

According to the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition," K. Simonyan and A. Zisserman of Oxford University designed the VGG16 convolutional neural network. The ImageNet database, which has more than 14 million images in its 1000+ classifications, reports that the model's accuracy in the top-five tests is 92.7 percent. ILSVRC-2014 received a number of well-known models, one of which was this model. By substituting huge kernel-sized filters (In the first and second convolutional layers, the weights are 11 and 5, respectively) with several 33 kernel-sized filters one after another, it achieves an improvement over AlexNet in terms of performance. NVIDIA Titan Black GPUs were used to train VGG16, which took several weeks to train [3].

VGG16 Architecture

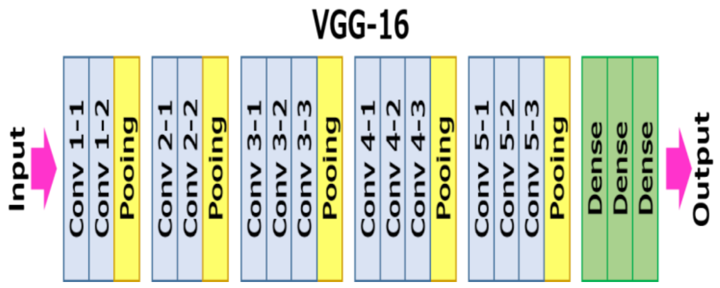


Figure – 2: VGG16 Architecture [3]

Applications of VGG16 Algorithm

- ➢ Many deep learning image categorization challenges make use of this technique.
- ➢ SqueezeNet.
- ➢ GoogLeNet.
- ➢ An Image Recognition Algorithm with Extremely High Accuracy

Advantages of VGG16 Algorithm

- For benchmarking purposes on a certain task, it is an excellent design to use.
- As a result of the abundance of publicly available pre-trained networks for VGG available on the internet, VGG is widely utilized for a wide range of applications.
- It has so many weight characteristics that the models are extremely heavy, weighing in at 550 MB or more in weight size.

Disadvantages of VGG16 Algorithm

- Model with a higher weight.
- More time for training.
- The gradient problem with vanishing gradients.
- However, and this may come as a surprise to you, deeper networks can have higher test error and less generalisation if they are constructed in a straightforward manner, such as VGG16 [4].

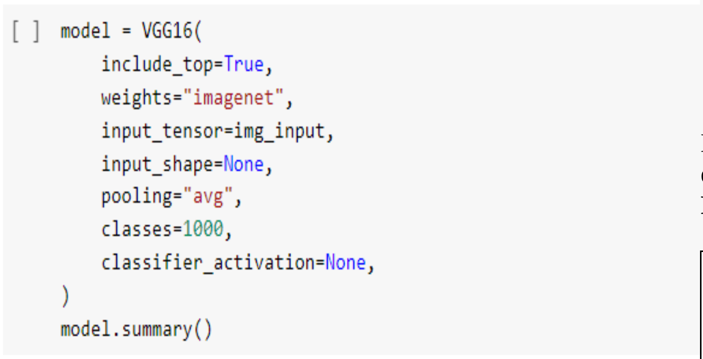
II. EXPERIMENT AND RESULTS

A. *Feature Extraction Using VGG16*

Dataset

The dataset was taken from Github repository provided by our university. It contains 349 CT Images of COVID-CT dataset from 216 patients and 463 non-COVID-CT dataset [5].

Model Parameters and Values

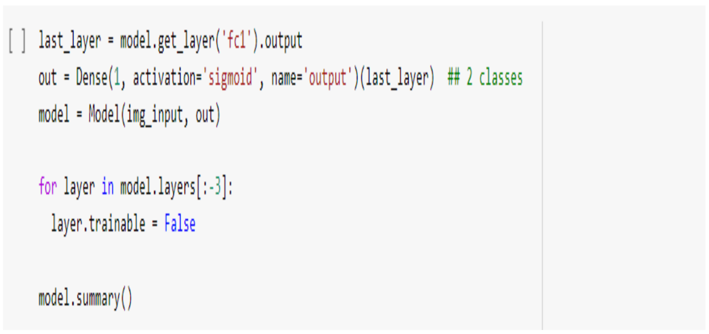


```
[ ]  model = VGG16(
         include_top=True,
         weights="imagenet",
         input_tensor=img_input,
         input_shape=None,
         pooling="avg",
         classes=1000,
         classifier_activation=None,
     )
     model.summary()
```

As seen in the above code of lines, the parameters and its values set are as follows [5]:

include_top = True, weights = “imagenet”, input_tensor = img_input (which is (224,224,3)), input_shape = None, pooling=”avg”, classes=1000, classifier_activation=None
I have tried using pooling = “max” but could not achieve satisfying results. Also classifier_activation was kept as “softmax” but could not achieve the results as desired. Above parameters finally gave me the result as was expected.

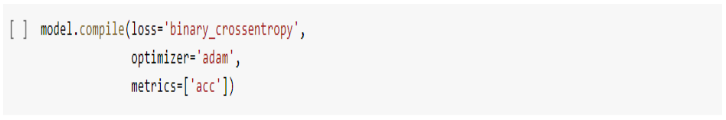
Model Layers



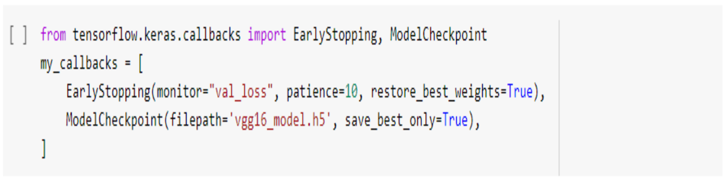
```
[ ] last_layer = model.get_layer('fc1').output
    out = Dense(1, activation='sigmoid', name='output')(last_layer)  ## 2 classes
    model = Model(img_input, out)


    for layer in model.layers[:-3]:
      layer.trainable = False


    model.summary()
```

As seen in the code model.get_layer(‘fc1’) is taken. I have tried with other options as fc2 and block_pool5 but was not getting desired results.

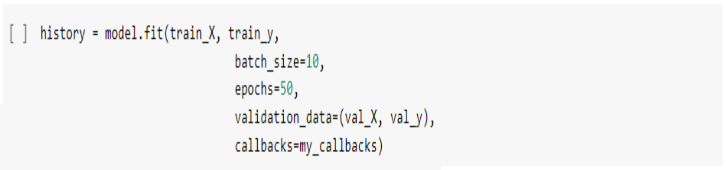
Model Compile and Fit Function



```
[ ] model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['acc'])
```



```
[ ] from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
    my_callbacks = [
        EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True),
        ModelCheckpoint(filepath='vgg16_model.h5', save_best_only=True),
    ]
```



```
[ ] history = model.fit(train_X, train_y,
                             batch_size=10,
                             epochs=50,
                             validation_data=(val_X, val_y),
                             callbacks=my_callbacks)
```

In the compile function, optimizer used was adam. Moreover in fit function, batch_size was kept as 10 and epochs to 50 to get the improved results [6].

Results and Discussion

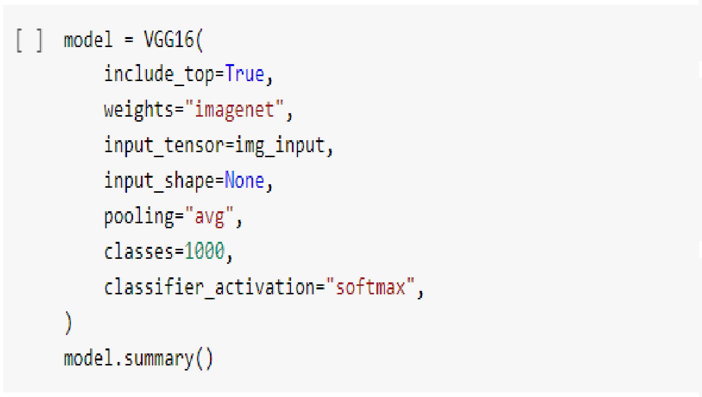
Metrics	Feature Extraction Transfer Learning (Given)	Feature Extraction Transfer Learning (Improved)
Train Accuracy	0.88	1.000
Validation Accuracy	0.74	0.860
Test Accuracy	0.84	0.875
Precision	0.94	0.833
Recall	0.72	0.937
F1 Score	0.82	0.882
AUC	0.84	0.875

Table – 1: Improved Feature Extraction Results

As seen in the table-1, almost all the metric values are improved by changing the parameters in the model and fit function and applying transfer learning concepts and running the model several times.

B. Fine Tuning Using VGG16

Model Parameters and Values

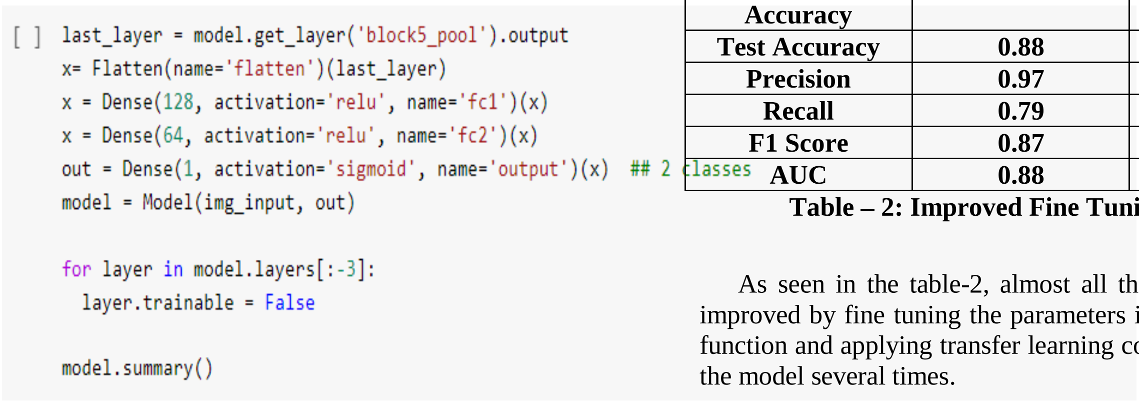


```
[ ]  model = VGG16(
         include_top=True,
         weights="imagenet",
         input_tensor=img_input,
         input_shape=None,
         pooling="avg",
         classes=1000,
         classifier_activation="softmax",
     )
     model.summary()
```

As seen in the above code of lines, the parameters and its values set are as follows:

include_top = True, weights = “imagenet”, input_tensor = img_input (which is (224,224,3)), input_shape = None, pooling=”avg”, classes=1000, classifier_activation=None
I have tried using pooling = “max” and “None” but could not achieve satisfying results. Also classifier_activation was kept as None and then I changed to “softmax” to achieve the results as desired. Above parameters finally gave me the result as was expected [7].

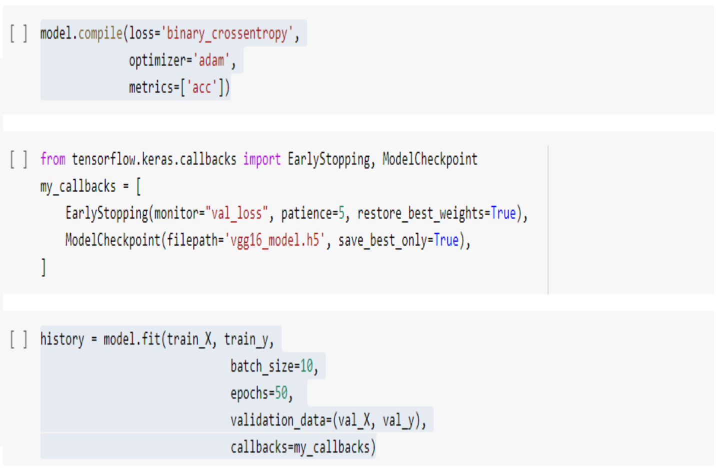
Model Layers



```
[ ]  last_layer = model.get_layer('block5_pool').output
     x= Flatten(name='flatten')(last_layer)
     x = Dense(128, activation='relu', name='fc1')(x)
     x = Dense(64, activation='relu', name='fc2')(x)
     out = Dense(1, activation='sigmoid', name='output')(x)  ## 2 classes
     model = Model(img_input, out)

     for layer in model.layers[:-3]:
       layer.trainable = False

     model.summary()
```

As seen in the code model.get_layer(‘block5_pool’) is taken. Also the activation function is taken as “sigmoid”. I also tried using “softmax”.

Model Compile and Fit



```
[ ]  model.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['acc'])
```

```
[ ]  from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
     my_callbacks = [
         EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True),
         ModelCheckpoint(filepath='vgg16_model.h5', save_best_only=True),
     ]
```

```
[ ]  history = model.fit(train_X, train_y,
                         batch_size=10,
                         epochs=50,
                         validation_data=(val_X, val_y),
                         callbacks=my_callbacks)
```

In the compile function, loss parameter value was kept as “binary_crossentropy” as there are only 2 probable classes in our case (COVID and Non COVID) and optimizer used was adam. Moreover in fit function, batch_size was kept as 10 and epochs to 50 to get the improved results [8].

Results and Discussion

Metrics	**Fine Tuning Transfer Learning (Given)**	**Fine Tuning Transfer Learning (Improved)**
Train Accuracy	**0.99**	**0.995**
Validation Accuracy	**0.91**	**0.920**
Test Accuracy	**0.88**	**0.916**
Precision	**0.97**	**0.934**
Recall	**0.79**	**0.895**
F1 Score	**0.87**	**0.914**
AUC	**0.88**	**0.916**

Table – 2: Improved Fine Tuning Results

As seen in the table-2, almost all the metric values are improved by fine tuning the parameters in the model and fit function and applying transfer learning concepts and running the model several times.

ACKNOWLEDGMENT

I would like to thanks my professor for guiding me through this assignment. Also I would like to thanks Sagi Haider for providing us with starting point for executing this algorithm.

REFERENCES

[1] https://searchenterpriseai.techtarget.com/definition/convolutional-neural-network

[2] Mohsen Kaboli. A Review of Transfer Learning Algorithms. [Research Report] Technische Universität München. 2017. hal-01575126

[3] https://neurohive.io/en/popular-networks/vgg16/.

[4] https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

[5] J. Zhao, Y. Zhang, X. He, and P. Xie, “COVID-CT-dataset: a CT scan dataset about COVID-19,” arXiv preprint arXiv:2003.13865, 2020.

[6] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, “The history began from alexnet: A comprehensive survey on deep learning approaches,” arXiv preprint arXiv:1803.01164, 2018.

[7] X. He, X. Yang, S. Zhang, J. Zhao, Y. Zhang, E. Xing, and P. Xie, “Sample-efficient deep learning for COVID-19 diagnosis based on CT scans,” medRxiv, 2020.

[8] https://github.com/sagihaider/TransferLearning_COVID19

[9] https://thedatafrog.com/en/articles/image-recognition-transfer-learning/