9/7/2018

Machine Learning Capstone Project: Image classification

I. Definition

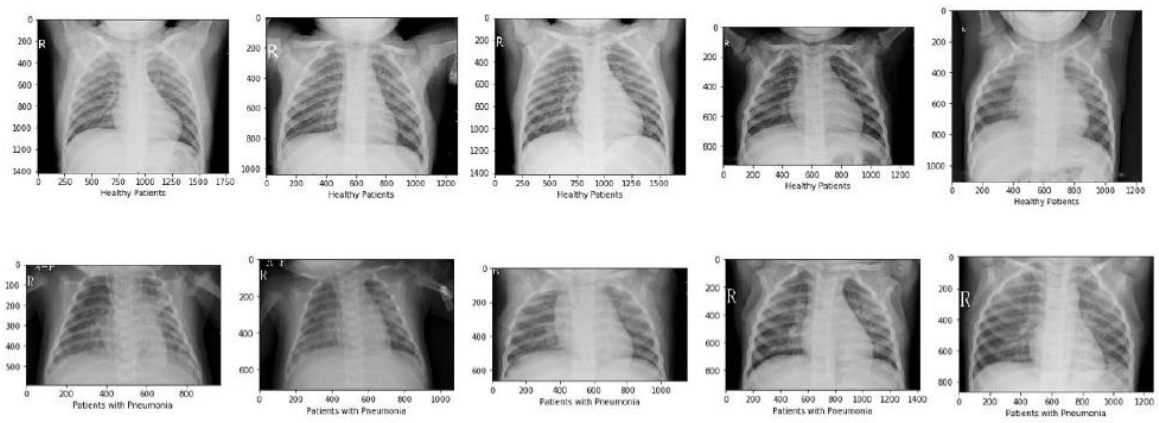
Project Overview

"The Flu is killing up to 4,000 Americans a Week" The Flu also known as influenza is a contagious viral infection that can be spread through coughing or sneezing. Pneumonia is a condition characterized by the inflammation of the lungs. The flu is a common cause of pneumonia, and in 2015 they were the eighth leading cause of death within the United States.

There were 40,414 deaths in the U.S. during the third week of 2018, the most recent data available, and 4,064 were from pneumonia or influenza, according to the CDC data. The number for that week is expected to rise more reports are sent to the agency. Although, additional tests are often used to diagnose this condition X-rays are still one of the most widely used, cost effective methods. Images are taken of patients are then visual analyzed to make a diagnosis or to determine if further evaluation is required.

Pneumonia accounts for over 15% of all deaths of children under 5 years old internationally and in 2015, 920,000 children under the age of 5 died from the disease. While common, accurately diagnosing pneumonia is a tall order. It requires review of a chest radiograph (CXR) by highly trained specialists and confirmation through clinical history, vital signs and laboratory exams. This competition seeks to improve the efficiency and reach of diagnostic services.

It is fascinating to me that when comparing the X-rays of healthy patients with those who have pneumonia that it difficult even notice differences between the examples.



Problem Statement

The goal is to use the labeled image dataset provided to train a Neural Network and the test the accuracy with which the model accurately diagnosis the condition.

Metrics

The goal of this project is to implement transfer learning to train a Convolutional Neural Network using by selecting the architecture that provides the best accuracy based on the given test set.

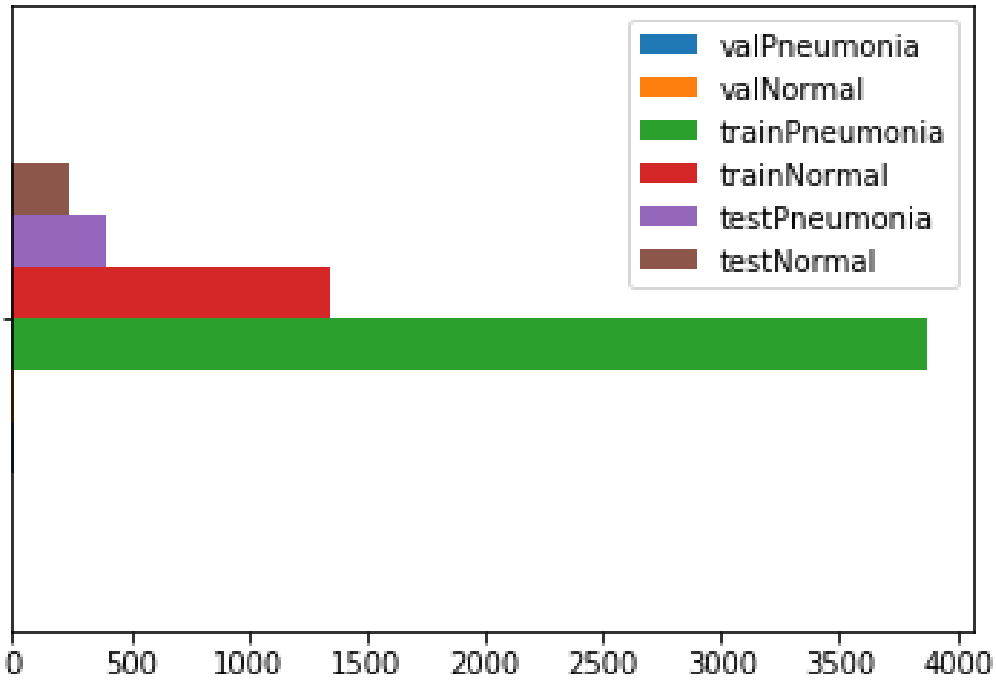
II. Analysis

Data Exploration

The dataset titled, “Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification” is provided by Daniel Kermany via Kaggle. It is organized into 3 folders (train, test, validation) and contains subfolders for each image category (Pneumonia/Normal). There are 5,860 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal). Of that total 1,585 files are of patients in good health (Normal), and 4,275 are of patients with various forms of pneumonia all labelled, Pneumonia. The images are different sizes but will be resized into uniform dimensions in order to be processed.

Luckily, the data was all in the Jpeg format as the ImageDataGenerator methods are able to process that format. Additionally the directory structure of the data worked with the kers’s

flow_from_directory method which requires that each class be saved in a separately labelled folder within each of the training, validation and test folders. The flow_from_directory method made it easier to resize the images directly. One minor complication existed in that the data was imbalanced.



However, although not shown due to scale, there were 8 images in each of the normal and validation folders for a combined amount of 16 validation images. The dataset, cited below, contained a relatively small validation set with which to select the ideal model, in the practice this may have adversely affected the fitting/training process. The data was used as given. It was viewed as a constraint of the problem, and attempts were made to optimize other parameters accordingly. In this project, there were two classes Normal and Pneumonia.

By convention, normal cases were labeled 0, while Pneumonia cases were labeled 1. Two different coding approaches were used to label the data. In the non-transfer learning approach the features and the labels were processed simultaneously but one by one using a created function. In the transfer approach, where some of the ImageDataGenerator based methods were used the feature data was processed in batches, in order, and the label data was created in order as well to be used in non-generator based methods.

```
np.argmax(Ytest, axis =1)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1])
```

Source: https://data.mendeley.com/datasets/rscbjbr9sj/2

Algorithms and Techniques

The basic approach of applying a convolutional neural network to classify images are as follows to load, normalize and resize image set. The next steps were to define a model architecture, i.e. the numbers, types of layers, number of nodes, optimizers, optimal learning rates in addition to other important hyperparameters. This is an iterative process that will requires careful thought and experimentation to achieve the best possible results.

Benchmark

The following results were recorded describing the effectiveness of pneumonia using X-ray:

> The initial chest radiographs of 31 patients with laboratory-proved pneumonia were evaluated by a panel of 6 radiologists who had no prior knowledge of the clinical data. No statistical reliability was found for distinguishing bacterial from nonbacterial pneumonia. Radiographic diagnoses were 67% accurate for the 16 cases of bacterial pneumonia, and 65% accurate for the 9 viral cases.

An accuracy of 60% or better would be result in an improvement compared to the study referenced.

The value of the model will be evaluated using the accuracy metric. The accuracy is defined as the number of times the model correctly predicts the same value as the actual value in the data set divided by the number of attempts. In this case, accuracy will be used to make a comparison to the benchmark results set forth in the paper above. However, this use case also suggests that in practice the measure recall may be an even more effective metric. Recall is defined as the number of cases that the model correctly predicts, true positives divided by the sum of true positives and false negatives. False negatives in this example would be people with pneumonia who are diagnosed as healthy. Conversely, true negatives would be health patients who were told that they have pneumonia. While the latter case may be scary proposition, and less than ideal, the former case is deadly. A sick patient misdiagnosed as healthy is a more important classification error and therefore a model that maximizes recall should probably be used in practice. The recall score will be calculated to evaluate the performance of the model.

III. Methodology

Data Preprocessing

The approach to identifying images was first to identify some of the necessary libraries and functions that would be required. A path library was required to locate the destination of the source data images. Initially the os library seemed like a good source, until the newer pathlib was discovered. The Path class inherits from pathlib and can be used to automatically forward slashes and backward slashes regardless of the operating system. The initial approach was to complete a basic CNN, without transfer learning, however even a basic model ran slowly on a CPU. One of the complications, of using an amazon ec2 instance with a Linux image was to format the image path that could be processed from different workspaces.

Another challenge associated with using amazon EC2 instances was trying to securely connect to the instance. This required getting up to speed with how to use Putty an ssh client to securely connect an amazon ec2 instance to jupyter notebook. Once connected, all of the data had to be uploaded the the virtual machine, despite these extra steps the benefit of quicker performance and the ability to experiment and identify bugs more quickly made it well worth it. Also, once connected, an EC2 instance made it feasible to implemented transfer learning using much larger models. As mentioned, earlier initially the data images were read off a CPU workspace, once they had been uploaded the code previously used had to be modified. Specifically, jupyter notebook automatically saves checkpoint files in locations that affected the way images were read in. After making modifications to the code, this issue was avoided altogether with a different approach using the flow_from_directory method.

The approach to identifying images was first to identify some of the necessary libraries and functions that would be required. A path library was required to locate the destination of the source data images. Initially the os library seemed like a good source, until the newer pathlib was discovered. The Path class inherits from pathlib and can be used to automatically forward slashes and backward slashes regardless of the operating system. The initial approach was to complete a basic CNN, without transfer learning, however even a basic model ran slowly on a CPU. One of the complications, of using an amazon ec2 instance with a Linux image was to format the image path that could be processed from different workspaces.

Another challenge associated with using amazon EC2 instances was trying to securely connect to the instance. This required getting up to speed with how to use Putty an ssh client to securely connect an amazon ec2 instance to jupyter notebook. Once connected, all of the data had to be uploaded the the virtual machine, despite these extra steps the benefit of quicker performance and the ability to experiment and identify bugs more quickly made it well worth it. Also, once connected, an EC2 instance made it feasible to implemented transfer learning using much larger models. As mentioned, earlier initially the data images were read off a CPU workspace, once they had been uploaded the code previously used had to be modified. Specifically, jupyter notebook automatically saves checkpoint files in locations that affected the way images were read in. After making modifications to the code, this issue was avoided altogether with a different approach using the flow_from_directory method.

Prior to discovering the flow_from_directory method the data was pre-processed the following way. A list was created with all the directory folders that contained images. Then each file path was individually labeled with a zero or one before all the data was stored in a dictionary called data folder.

From that point, key -value pair was provided to a function called createXy. The createXy function took in a set of data and returned features and the associated labels. This function could be used to format the data of for the train, val, and test sets.

While this process provided a good logical framework to evaluate the pre-processing it was largely abandoned. It was replaced by creating a "generator" object from the flow_from_directory method, using the predict_generator method to predict features. From this point as long as the tensors were of appropriate dimensions the data was ready to processed through the fully connected portion of a neural network.

Implementation

The implementation of the transfer learning model and the non-transfer model is relatively similar. The major difference being that the transfer model was built about the VGG16 architecture which contains a little over twenty layers with almost 15 million trainable parameters. The dense, fully connected portion of the architecture contains six layers including the input layer from the VGG architecture with more than 12 million parameters. Together the final transfer learning model contained more than 25 layers and about 27 million parameters. Comparatively, the non-transfer learning model contained only 12 layers and about 6 million parameters. The transfer learning model featured dropout layers to reduce the likelihood of overfitting.

Refinement

Initially, the larger transfer learned model achieved an accuracy of only 10% with no improvement after as many as 10 epochs. The smaller model achieved competitive accuracy on the data set after fewer epochs with improvement. In order to improve the accuracy of the VGG based model the first parameter that was adjusted was to resize the images to 224 x 224 pixels the image size that the VGG model was believed to be trained using. However, the most significant improvements were due to changes made to the optimizer. The final optimizer chosen was the Adam due to performance. Additionally, better performance was associated with slower learning rates and using a decay rate to further slow the learning rate as epochs passed.

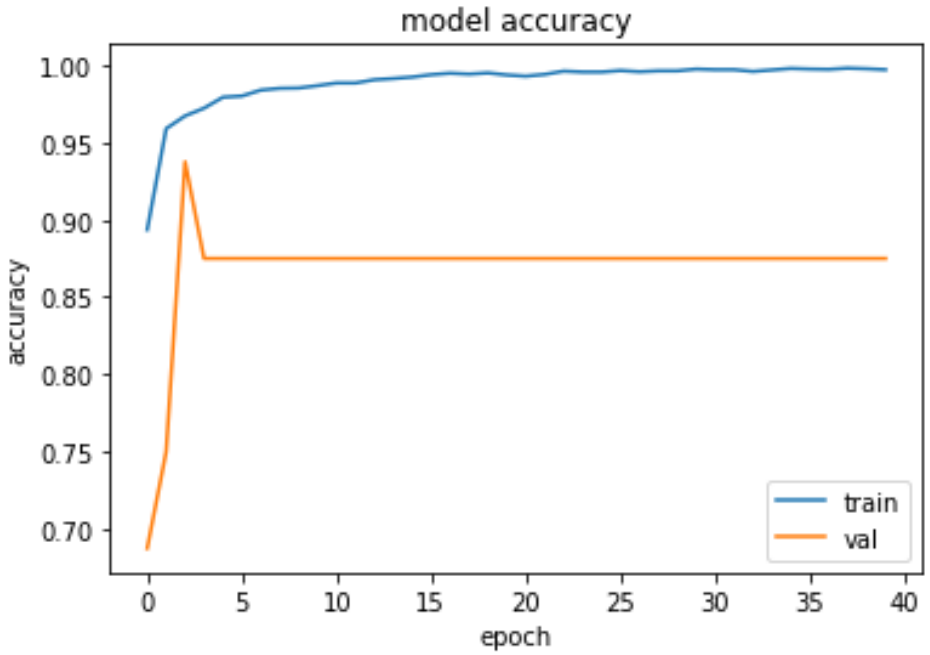
Worse performance was experienced by adding layer to the fully connected model and/or buy increasing the number of nodes of the existing model. Lastly, dropout layers of about 50% were found to be the most effective.

After 20 epochs or so, the training accuracy was close to 100%. As mentioned earlier, the validation set was much smaller, and left so as that was the structure of the cited data set. However, the validation accuracy range was so reduced that while a training set accuracy approached 100% over time the validation accuracy was sporadic after reaching 80% or better.

IV. Results

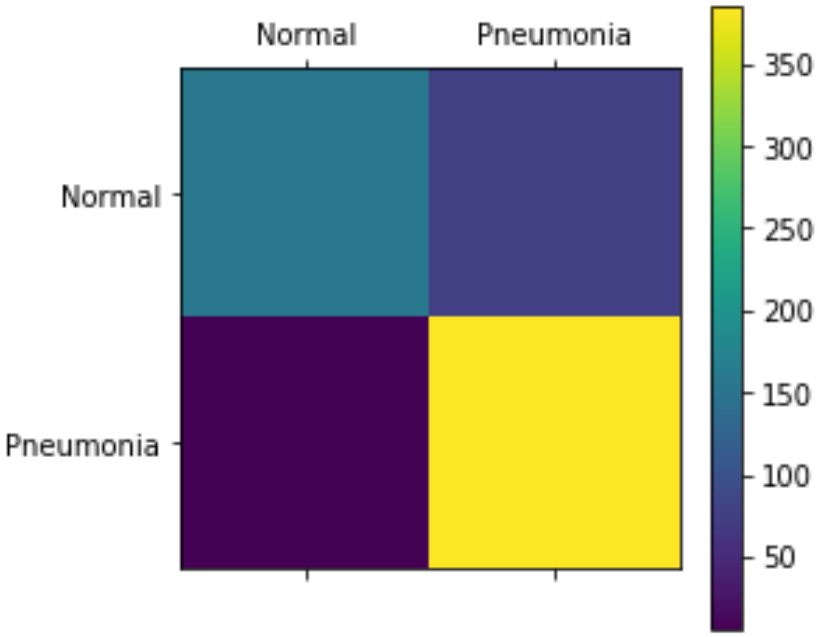
Model Evaluation and Validation

Both models, the transfer learned model and the smaller model achieved accuracies and recalls better than the benchmark. In transfer model run submitted the VGG based model the model achieved accuracy of 86% on the test set with and an 83% recall. As expected the model learned the dataset over time and began to perfectly fit the data set.



However, due to the relative size of the models I was surprised to see for this application both model performed similarly. For a practical implementation, and when considering additional expense of computing cost for the larger model might lead the smaller model to be a better choice. Further, upon inspection of the test examples both models performed very well on predicting the cases where

pneumonia was present and poorly on normal cases. In other words, the both predicted many healthy patients had pneumonia more than they predicted that sick patients were healthy.



Justification

As mentioned earlier an accuracy of 60% or better is considered an improvement to trained professional using the images to make a diagnosis. In this application, however even a small improvement one missed diagnosis discovered with this method could result in a saved life. The models selected reliably performed better than the benchmark after multiple trials/

V. Conclusion

Reflection

This was a challenging yet rewarding solution. I learned a great deal, and experienced many unexpected issues. I learned the most about connecting to and working on virtual machines. I also learned about many of the more robust methods in the Kera library that can be effectively used on much larger datasets. I was surprised that in many instances the brute force approach to optimization

did not produce the best results. In many instances adding layers or node decreased performance. Meanwhile, smaller changes or even reducing the model size produced better results. Although the model performance seemed reliably top out around 80% it seems that a larger validation set would have a wider range for optimization. There also seems to be much more to the generator methods, however the cost of iterating on GPU can become expensive.

Improvement

Given the constraints, I think the greatest improvements could be experienced with the use of data Augmentation. With more resources or time building upon the approach presented here those methods could show improved performance. Another, technique that could warrant further exploration is weighting the sampling of the training set to improve performance. However, due to the fact, of the nearly 100% training set accuracy achieved it is likely that the greatest constraint to this data set was the relatively small size of the validation set.