Machine Learning Capstone Project

Introduction

As the covid-19 ravages throughout the world, scientists are racing to understand the characteristics of this new virus. The pandemic is gradually being controlled by the high-income countries and now hitting the small to middle income nations, which according to the world bank could push 71 million people into extreme poverty in 2020 under the baseline scenario and 100 million under the downside scenario. Since tomography exams are not widely available, x-rays help doctors understand the extent of the damage of the lungs from their patients. A Computer aided diagnosis (CAD) that suggests the doctor the diagnostic of their patient's lungs could accelerate and improve the speed of attention at hospitals.

Recent Work

In academia, deep learning is starting to be applied more widely as a diagnostic aid (see works in Medical Image Computing and Computer Assisted Intervention — MICCAI 2019). For example, in this study (Serj, Bahram, Gabriela, & Valls, 2018) they test the architecture of a deep convoluted neural network (dCNN) for detecting lung cancer in computer tomography (CT) images. The output of that model is binary: has cancer or not.

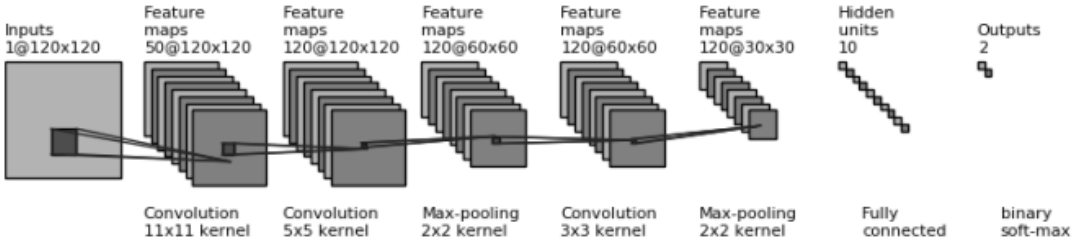

Figure 1: Architecture for the study of lung cancer detection

Computer assisted diagnostics not only have potential in western medicine, but in traditional Chinese medicine as well. Image recognition is relevant in analyzing the tongue (Qingli, Yiting, Hongying, Zhen, & Zhi, 2010) and heart pulse (Huiyan & Yiyu, 2005). In Chinese medicine information about those two parts can show a wide variety of insights to determine a person's condition.

Therefore, the author believes that in the future, computer vision and signal processing AI will be a key player in medicine in the whole world as more data is available, models become better, and computer architectures become cheaper (GPUs).

Problem Statement

The problem is a classification for four classes. The proposed model classifies between normal, and three types of pneumonia: bacteria, virus (not covid-19), and covid-19.

The percentage of precision for each class is tracked. That is, how many of the images inside a class were classified correctly.

Analysis

Seven types of x-rays are contained in the dataset. Stress-Smoking, streptococcus and SARS are discarded because there is not enough data. Also, pneumonia from bacteria has 2.772 images, so the model is unbalanced with respect to the other classes. The information can be downloaded from https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset .

Table 1: Data summary

	Label	Label_1_Virus_category	Label_2_Virus_category	Image_Count
0	Normal			1576
1	Pnemonia	Stress-Smoking	ARDS	2
2	Pnemonia	Virus		1493
3	Pnemonia	Virus	COVID-19	58
4	Pnemonia	Virus	SARS	4
5	Pnemonia	bacteria		2772
6	Pnemonia	bacteria	Streptococcus	5

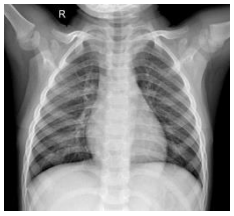

Figure 2

Figure 4

Figure 5

Figure 3

Figures 1 to 4 show the four different classes. Damage to the lung is seen as a white cloud, where it should be black.

Algorithms and techniques

This project uses a convoluted neural network model to analyze the images. CNNs have proven good at identifying image features. Studies are often implemented as existing CNNs architectures or modifying some layers of the existing models.

The benchmark model to compare to is the flower classification from the transfer-learning class from Udacity's deep learning with Pytorch course. This is a VGG16 model, a CNN whose last layer is trained to classify flowers.

The accuracy for each of the seven types of flowers is the following:

```
Test Accuracy of daisy: 81% (75/92)
Test Accuracy of dandelion: 89% (118/132)
Test Accuracy of roses: 70% (64/91)
Test Accuracy of sunflowers: 64% (65/101)
Test Accuracy of tulips: 65% (81/124)

Test Accuracy (Overall): 74% (403/540)
```

When it is used for the lung images, at least the same overall test accuracy is expected as a benchmark. If that accuracy seems low, as a remainder, sometimes it is difficult even for a human to distinguish some types of those flowers (tulips) from each other. An overall accuracy of 74% is much better than guessing, which would be an overall accuracy of approximately 20%.

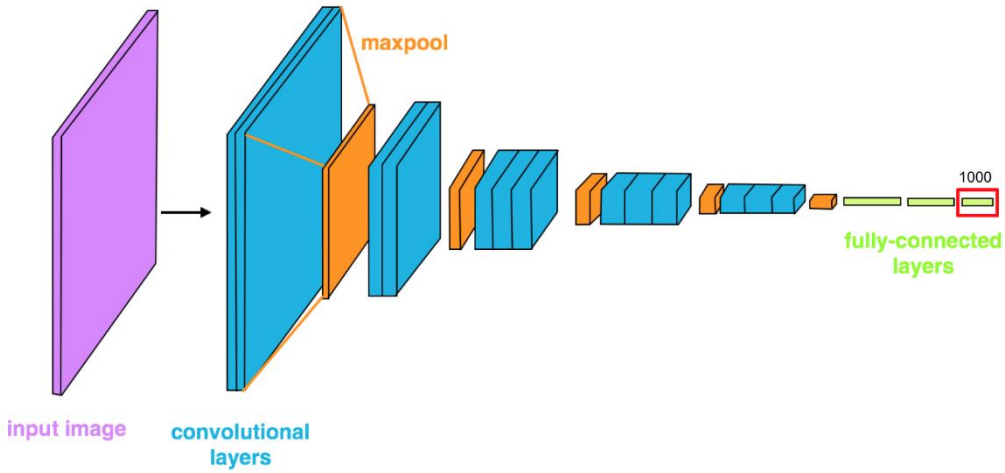

Figure 6: VGG 16 architecture

The VGG architecture consists in 13 convolutional layers that may duplicate or maintain the previous layer's depth, with a RELU activation between them. Each layer has a filter of 3x3, stride of 1, and padding equal to 1. To decrease the number of parameters and take the important information from each layer, five maxpooling layers are added in between that squeezes the image (though, some spatial information is lost in this process). Lastly in the "features" layers, comes an average pooling. The output is 7x7x512, which is fed to a multilayer perceptron (MLP) with three hidden layers, that have RELU and dropout in between each of them. This final MLP is what is aimed to train.

Methodology

Data preprocessing

The normal, virus and covid-19 classes area augmented to about the same quantity of images as the pneumonia-bacteria class. Since the covid-19 x-rays are just 58 pictures in the downloaded folder, 8 are separated for the test dataset. Albumentations is the data augmentation tool used for this specific task. With the help of a simulator of the package's effects, the following transforms are used:

- Horizontal flip.
- Random gamma.
- Blur.
- Random brightness contrast.
- Rotate: a few degrees so that orientation does not become a feature.
- Gauss noise.
- HueSaturationValue

On top of the albumentations, additional editions are carried out by torchvision. Here it is important to transform to tensor and to normalize the pixel values so that the model functions correctly:

- Resize: VGG16 only accepts 224x224 images (in 3 color channels).
- CenterCrop
- RandomRotation: to be consistent with the albumentation rotations too.
- ToTensor
- Normalize

Implementation

In the beginning, an MLP network was tried to analyze the images. It had 10 hidden layers and the input layer used a 1024x1024 pixels image. This produced an enormous one-dimensional tensor. The hidden layers did not squeeze the dimensions quick enough so when the code to form the architecture was run, the local PC froze, and the OS restarted on its own. On the second trial, a smaller image was used, and the second hidden layer immediately lowered the dimensions significantly after the input layer. It took some hours to finish training with the PC at 100% CPU, and when cuda was attempted to use, the training crashed due a lack of GPU memory. The results were not very good (<40% mean accuracy for each class), so this architecture was abandoned.

After reconsidering the strategy, it was decided to work with Amazon Web Services (AWS) and the proven architecture of CNNs. AWS cloud service shows a couple of advantages over working on the local PC: access to on-demand machines. Surprisingly, there is no need to use a GPU-capable machine for a notebook instance, for the heavy-duty machines may be accessed only on the moment of training, if needed. Finding how the Pytorch class exported things other than the model data was not easy, but eventually achieved it by accessing an environment variable called “SM_OUTPUT_DATA_DIR” from SageMaker.

The instance used for training is the *ml.p2.xlarge*, which has four vCPU, one GPU (K80) and 61 Gb of disk memory, and 12 Gb of GPU memory. Later, this proved enough because the model took a little less than two and a half hours for 10 epochs. Nonetheless, to accelerate training even more, an instance eight times bigger (ml.p2.8xlarge) was requested to AWS support. Sadly, this eight-fold more expensive machine did not prove eight times faster, so it was stopped after three epochs. All subsequent training exclusively used ml.p2.xlarge.

To implement the award-winning VGG16 model, the Pytorch API was chosen. It is simple to use – just import from torchvision models and create its instance object. AWS has a massive storage service called Simple Storage Service (S3). From it, a Pytorch estimator takes advantage of reading the huge amounts of data that can be uploaded here. Therefore, it was decided to upload the training data to S3. However, the connection was to unstable to upload everything, and it could not pause and start uploading again where it had stopped. Then it was attempted to upload to the notebook instance. Uploading more than a file to the notebook instance is very inconvenient, so a zip was tried. The same problem of stability was found in this method. On the other hand, by looking at Udacity’s forums, it was found a 3rd party software called Cyberduck that synchronizes with clouds systems allowed to upload the data with better stability, and handle the files directly in S3, among other features. To do this a user and secret key must be set up in AWS.

Training

For training the model, a single train.py file was created. There was no need to implement a model.py file because the train file imports and downloads the VGG model directly. It would have been required if the architecture itself had been coded.

In general, with neural networks, the best generalization is wanted, subject to the lowest memorization. For this, the training estimator separated 15% of the training data to cross validate. This is very important to avoid overfitting (memorization instead of learning). Whenever the training loss decreases, the model can predict better over the training data, but not necessarily over the validation data. In this way, as the training executes, if during the current epoch, both the training and validation losses decrease, then the updated model is saved. The validation data therefore is separate from the training and testing data. If the testing data were used to validate, then it would be implicit in the training and would be "contaminated" (the model chooses when to save based on test data, which in essence is training with the test data—and this is wrong).

Debugging the training code was a rather slow process because the training script takes some time to run (and, when debugging, fail).

Saving the model was a priority to avoid time consumption. Hence, instead of directly deploying the estimator object, a Pytorchmodel object was created that accessed the model's artifacts saved to S3 by the Pytorch estimator with the lowest possible validation loss, so that the model need not be retrained each time the notebook instance was run.

Deployment:

The instance of the Pytorchmodel uses a script called Predict.py which handles the input picture and outputs the model. It takes the *ml.c4.xlarge* machine approximately seven minutes to deploy. The functions input_fn and output_fn proved to be problematic since they were reused from another project. However, after reading the AWS documentation, realized that those functions had a default version supplied automatically by the Pytorchmodel class, and there was no need to customize their code for this project.

This was slow to debug this too, but Cloudview helped very much in finding the mistakes. Another problem was to process the correct tensor format as the model output.

Results

1st model: feature parameters frozen

The first model uses a lr = 0.01 and freezes all the feature layers except the last, fully connected layer (MLP).

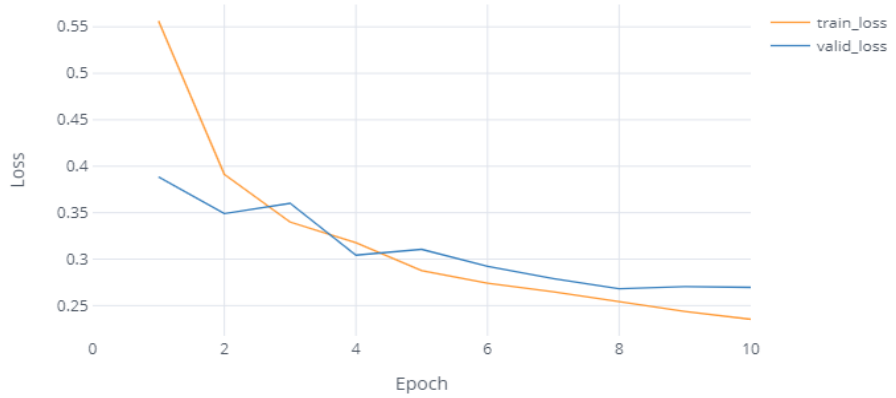

Figure 7: Losses for the frozen features model

One can observe that after epoch eight, the validation loss seems to be stable. Hence, the model artifacts were saved at epoch eight because it is a minimum. The model took approximately 2,5 h to train.

And the accuracy is the following:

```
Test Accuracy of Normal: 52% (122/234)
Test Accuracy of Pnemonia-Virus: 73% (109/148)
Test Accuracy of Pnemonia-Virus-COVID-19: 75% (156/208)
Test Accuracy of Pnemonia-bacteria: 95% (231/242)

Test Accuracy (Overall): 74% (618/832)
```

This model predicts the overall classes as good as the benchmark model. The accuracy of the normal lungs seems to be low (though much more than 25% guess accuracy). However, the model's main objective is not to discriminate between normal and sick, because someone with a cough and chest pain that visits the hospital is unlikely to have unaffected lungs. Surprisingly, the model classifies a pneumonia caused by any virus as well as those caused by covid-19.

2nd model: feature parameters free to change.

The second model uses the same learning rate and unfreezes the "feature" layers when training. This took a longer time because the optimizer had to change many more of the "freed" parameters.

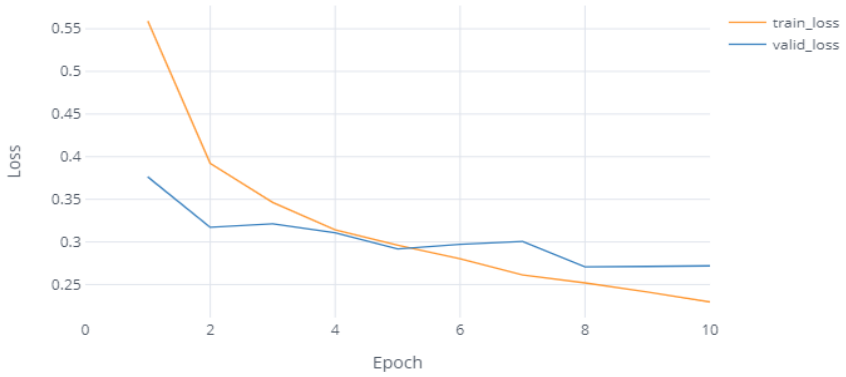

Figure 8: Losses for model with feature parameters "unfrozen"

After epoch eight, the valid loss also seems to stabilize. The model was also last saved at that epoch.

The overall accuracy diminished 1%:

```
Test Accuracy of Normal: 50% (119/234)
Test Accuracy of Pnemonia-Virus: 77% (115/148)
Test Accuracy of Pnemonia-Virus-COVID-19: 71% (149/208)
Test Accuracy of Pnemonia-bacteria: 93% (227/242)

Test Accuracy (Overall): 73% (610/832)
```

It can be concluded that freeing the feature parameters did not improve the solution and no farther testing is done by freeing the “feature” parameters.

3rd model: learning rate with 0.1 instead of 0.01

A faster learning rate means that the model converges faster, but also means that it may overshoot the solution’s minima. This accelerated learning rate produced the following:

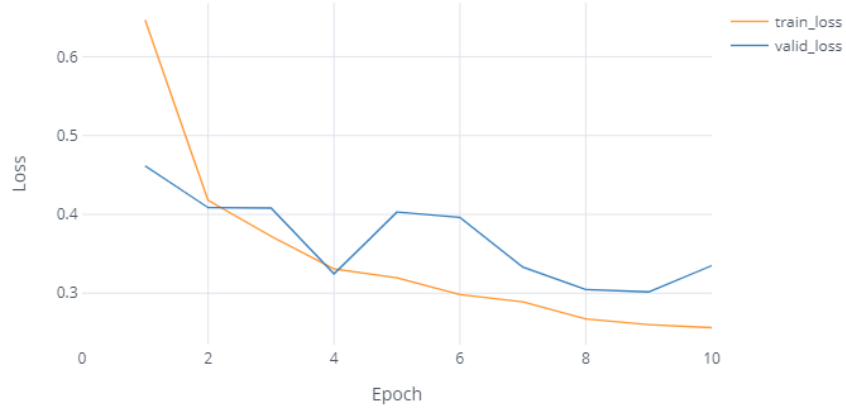

Figure 9: Model with Learning Rate of 0.1

Note that a minimum valid loss was achieved at epoch nine. The model’s train loss also falls more sharply than in the other models, as expected.

Surprisingly, the model performed best so far, with 76% of overall accuracy.

```
Test Accuracy of Normal: 52% (124/234)
Test Accuracy of Pnemonia-Virus: 88% (131/148)
Test Accuracy of Pnemonia-Virus-COVID-19: 78% (163/208)
Test Accuracy of Pnemonia-bacteria: 91% (222/242)

Test Accuracy (Overall): 76% (640/832)
```

For the virus was 88% and for covid-19 was 78%. Again, the bad performance in detecting a normally functioning lung (52%) decreases the overall accuracy.

4th model: learning rate with 0.001 instead of 0.01

On the other side, a slower learning rate implies that the model converges more slowly but is very unlikely to overshoot a minimum. This accelerated learning rate produced the following:

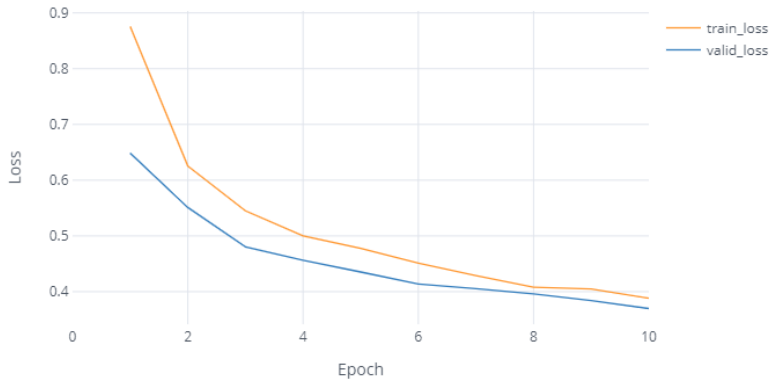

Figure 10: Model with Learning Rate of 0.001

Up until the last epoch, the model kept improving its validation loss.

However, the overall accuracy was 1% worse than the benchmark, and 3% less than the top performing model

```
Test Accuracy of Normal: 54% (128/234)
Test Accuracy of Pnemonia-Virus: 83% (123/148)
Test Accuracy of Pnemonia-Virus-COVID-19: 73% (153/208)
Test Accuracy of Pnemonia-bacteria: 86% (210/242)

Test Accuracy (Overall): 73% (614/832)
```

Conclusion

The best accuracy was 2% better than the benchmark model, produced by the model trained only in the fully connected layer and the fastest learning rate of 0.1. The author is not familiar with the precision that a radiologist would achieve in this dataset, but it can certainly suggest and help the medic what the diagnostic is.

Possible future enhancements

There are a couple of improvements that could be made to the model. First, supplying it with more data – especially covid-19, since it had only 58 images to work with. This in turn will

improve the ability to generalize. Second, instead of modifying the VGG16 architecture, RESNET could be tried, as it is another cutting-edge CNN.

References

Huiyan, W., & Yiyu, C. (2005). A quantitative system for pulse diagnosis in Traditional Chinese Medicine. *Proceedings of the 2005 IEEE.* Shanghai.

Qingli, L., Yiting, W., Hongying, L., Zhen, S., & Zhi, L. (2010). Tongue fissure extraction and classification. *Applied Optics*.

Serj, M. F., Bahram, L., Gabriela, H., & Valls, D. P. (2018, April 22). *A Deep Convolutional Neural Network for Lung Cancer Diagnostic*. Retrieved from Arxiv: https://arxiv.org/pdf/1804.08170.pdf

Udacity. (2020). *Deep learning nanodegree*. Retrieved from https://github.com/udacity/deep-learning-v2-pytorch