Constituting an End-to-End Deep Neural Network to precisely distinguish distinct types of Pneumonia using Chest X-Ray Images.

Swastik Nath.
Udacity Machine Learning Engineer Final Capstone Proposal.

DOMAIN BACKGROUND:

Pneumonia, one of the prime lungs disorder which typically strikes the facade of the lung's alveoli and proceeds to exhibit manifestations like breathing difficulty, fever, muscle ache, chest pain etc. and without precise diagnosis this might end up being lethal to humans. Pneumonia disperses in two methods, bacterial contamination and viral contagion. Every year, pneumonia concerns approximately 450 million people globally (7% of the population) and results in about 4 million deaths. With the influx of antibiotics and vaccines in the 20th century, survival has been considerably improved. Notwithstanding, in developing countries, and also among the very old, the very young and the chronically ill, pneumonia prevails as a leading cause of mortality. Pneumonia often shrinks the period of agony amidst those already approaching to death and has consequently been denominated "the old man's friend". Pneumonia generally commences as an upper respiratory tract infection that leads into the lower respiratory tract. It is a variety of pneumonitis (lung inflammation). The normal verdure of the upper airway gives shield by colliding with pathogens for nutrients. In the deeper airways, reflexes of the glottis, activities of correlative proteins and immunoglobulins are essential for safekeeping. Micro aspiration of contaminated excretions can taint the coarser airways and produce pneumonia. The advancement of pneumonia is decided by the virulence of the organism; the measure of bion required to start an infection; and the body's immune response towards the infection. Bacteria hold the most prevalent explanation of community-acquired pneumonia (CAP), with *Streptococcus pneumoniae* isolated in nearly 50% of cases. Other commonly isolated bacteria include *Haemophilus influenzae* in 20%, *Chlamydophila pneumoniae* in 13%, and *Mycoplasma pneumoniae* in 3% of cases; *Staphylococcus aureus*; *Moraxella catarrhalis*; *Legionella pneumophila*; and Gram-negative bacilli. Several drug-resistant variants of the preceding contagions are growing further traditional, including *Drug-resistant Streptococcus pneumoniae* (DRSP) and *Methicillin-resistant Staphylococcus aureus (MRSA)*. Ordinarily related viral contagions involve rhinoviruses, coronaviruses, influenza virus, respiratory syncytial virus (RSV), adenovirus, and parainfluenza.

In this world, there are still several fields where authority in accurately diagnosing these disorders is infrequent, however, if we can utilise the potential of deep learning and harness that horsepower to constitute a machine learning model that can accurately diagnose these chest complications just simply from the X-Ray images, it will unquestionably be helpful to the people, even it will be helpful to the experts rendering their perspicacity on specific events, where they believe it will be customary to consult an expert. As this model has been equipped with the dataset comprising real-world images identified by experts. According to the investigation done by Daniel S. Karmany, Michael Goldbaum et. al1 they introduced the architecture of CNN model already pre-trained on the ImageNet Dataset which carries almost 10 million images, where the model's feature extractor's parameters. The latest anxieties regarding viral epidemics are one of the most demanding research sectors and imbuing AI with that research would be great. In this clinical decision support diagnostic tool based on deep learning frameworks, we need to use a method called Transfer Learning which can train a neural network with just fractions of the data requirement.

I was subjectively motivated to work on this project because, the recent still ongoing devastation made by a viral epidemic, which substantiates same indications as Pneumonia, this is also timeliness to develop a corrective image classification system which can adequately address the diagnostics just by looking at the Chest X-Ray images, in fractions of a second. Just by changing a few parameters the model can be trained on different images and will efficiently enable people to diagnose their diseases while also helping physicians in comprehending the X-Ray images with more confidence.

PROBLEM STATEMENT:

To be particular in expressions of machine learning vocabularies, this predicament of classifying medical images based on specialist identified X-Ray images, is a type of Supervised Image-based Multiclass Classification problem, which is simply, every single image instance is a type of a single class. In this intricacy in hand, there is an entirety of 3 classes the images can be arranged to, the first one is Bacterial Pneumonia, the second one is Viral Pneumonia and the third one is the Pneumonia Negative. The photographs are X-Ray images upon which we need to train our model and render judgment. The X-Ray images being BW the images might have to be converted to RGB, notably when inferring as the ImageNet model is trained upon 3 channel RGB images. The images also need to be resized.

In contemporary advancements encompassing deep learning architectures, Convolutional Neural Networks (CNN) are established to be considered sound in terms of extracting and interpreting features of images uniquely. As the research paper by Daniel S. Karmany, Michael Goldbaum, et. al1 recommended that using Deep Convolutional Network which couples Convolutional Layers, RELU, Dropout Layers in different sequences and trained upon the dataset Imagenet comprising about 10 million images of different objects and freezes the feature extractor layers while modifying the classification dataset according to the number of associating classes. We then use a Cross-Entropy loss function to measure the deviation of the predictions of each class and use Stochastic Gradient Descent which behind the screen uses the Softmax Activation function, which transforms meanings of the output layers as the probabilities of the individual classes, which makes much more sense in the real world.

DATASETS AND INPUTS:

The dataset Version 2.0 and Version 3.0 for this model is made accessible by Daniel Karmany, Kang Zhang, Micheal Goldbaum et.al1 from Mendeley Data by Elsevier2, the dataset is associated to the University of California, San Diego and Guangzhou Women and Children's Medical Centre. The data contains thousands of expert validated Chest X-Ray images and are split into Train and Test set of independent images, and each of these sets there is two more distributions named Normal and Pneumonia. In the distribution Normal, there are thousands of Pneumonia Negative X-ray images which is named with the patient's id and type of the image class such as normal. In the Pneumonia distribution, we get other two types of images such as bacterial or viral contamination with their names as their representing labels appended with patient's id. The dataset is completely anonymized, so there is no possible way to trace back to the patient from the images. To feed these data to a machine learning model we need to redistribute the images into properly configured directories.

The whole dataset both Version 2.0 and Version 3.0 is available in the Mendeley Data website as zip files where version 2.0 is around 1.15 GB in size while the Version 3.0 is around 7.9 GB in size.

So, in our AWS SageMaker Notebook instance, we download the file by copying the https link and issue a wget command which goes ahead and download the file into the AWS SageMaker Notebook Instance workspace. Next, we create a new directory in the notebook instance to hold the properly distributed directories. So, at first, we create two more directories inside the new directory to hold the train and test set. Now, at this stage, we use python libraries and control loops and distribute the images from Normal and Pneumonia directory into 3 separate directories which will work as the label of these images for both the training and test set. So, basically what we did was we changed the distribution of the images and put the images in their specific label directory like we put Bacteria infected Pneumonia X-Ray images into their directory. As we have chosen PyTorch as our preferred deep learning framework, the framework requires that the images are distributed into the same number of directories as there are target classes and treats the name of the directory as the label of the image. Next up, we shuffle up the dataset images and divide them into batches to feed our PyTorch based Machine Learning Model. In the process, we also resize them to properly fit the model's requirements and convert them into multidimensional tensors. We use torchvision package to do so. PyTorch models perform the calculation on multidimensional tensors. We do not get into Image Augmentation process here as the dataset we have doesn't vary much and the complex architectures of Deep CNNs, augmentation result into overfitting to the training data. We repeat the process for the test dataset too, and also devoid of any kind of image augmentation process. As we intend to deploy or model via the AWS SageMaker platform, to use the training and the test dataset with a SageMaker PyTorch Estimator, we need to upload the dataset already processed and properly distributed to an Amazon Simple Storage Service (S3) object in the same datacentre region as the model would be deployed to.

Solution Statement:

To resolve this problem what we need to do is to use a Deep Convolutional Neural Network, with each of layer's parameters set to the values as it learned throughout its training on the ImageNet Dataset. The research paper by Karmany et. al suggested that we can freeze the feature extractor layers of the neural network architecture and update only the weights and biases of the last few layers, but in my case, I found that, doing so makes the model quite inaccurate in both training and test set. So, to remediate that, what I have done is initialized the model with the learned weights and biases for each of the layer but made the model update its full parameter range following the current gradient. It made the model to converge faster as it was able to get around 82% test set accuracy with only 4 epochs of training.

As the choice of deep neural network architecture, we use the VGG19 Batch Normalization and Resnet 50, both with pretrained weights and biases from the ImageNet classification challenge training. We change the underlying architecture of the model to some small extent, basically in the case of VGG19 Batch Normalization, I added a RELU activation layer and a Dropout Layer with probability of dropout being set to 50% of the total neurons and lastly I added a Linear Output Layer with the out_features set to the number of target classes in this case that is 3.

In the case of Resnet 50, I just made changes to the last layer and changed the last output linear layer's out_features to the number of target classes in this case which is again 3 and I at the first initialized the model's parameters with the parameters learnt from the training on the Imagenet Dataset.

As the loss function in both of the cases of VGG19 and Resnet 50 we have used the Cross Entropy, because it is one of the most efficient function there is to properly calculate the deviation of the model's prediction from it's label especially in cases like multi class or multi label classification problems. We also use Stochastic Gradient Descent as the optimizer here which goes ahead and updates the values of the parameters of each layer based upon the loss of the current epoch and current training datapoint. Another cause of using the SGD as the optimizer is because it uses the SoftMax function behind the screen which generates probabilities for the target class labels and from those probabilities the highest one is the class that's the model predicted. The probabilities are the measure how confident the model is in classifying that specific datapoint.

In case of training and testing of the model we feed the model shuffled batches of X-Ray images

To make inference over the images it first needs to be converted to Python Image Library (PIL) Image object and then it needs to be resized and after which it is transformed into a multidimensional tensor, which participate in the calculation at the CNN architecture. In case of inference, as the images are in BW format, we stack up three BW layers and creates a 3-channel image from the single channel BW Image and feed that image to the model endpoint to get the inference result.

Comparison with the Benchmark Model:

We have used only the Version 2.0 Dataset to participate in the architectural design decision procedure before we head over to training with the Amazon SageMaker and deploy the model to save resources and space.

According to the paper by Karmany, Goldbaum et. al[1], I downloaded the VGG19 Batch Normalization model with pretrained model parameters on the ImageNet Dataset, and made the gradient calculation for the features (Feature Extractor) layer to false leading to freezing of the layer parameters during the training process and trained it for up to 10 epochs. The accuracy of the model of the over the test dataset was quite impressive up to 78% with 490 correct predictions out of 624 total test datapoints.

Next up I download the Restnet50 model and fine-tuned the last fully connected layer's output features to the number of target classes but in this case I did not freeze the model's parameters, but instead I trained the model across its full parameter range with all the gradients being updated throughout the training by the SGD optimizer with the help of the loss function Cross Entropy Loss. I trained the model for only 2 epochs to avoid overfitting, as the Resnet50 being very complex and with relatively unvarying dataset, it is prone to overfitting. But with only 2 epochs of training it achieved 79% of accuracy on the test dataset with 498 correct predictions out of 624 total test datapoints.

Being less accurate on the test dataset I didn't deployed the models on the AWS SageMaker with their parameters set to the condition of not updating at the time of training but with all of their parameters set to requires_grad=True. But we drastically reduce the number of epochs of training down to 4 to not overfit on the training data.

The benchmark model might not be performing as good as the Resnet model because the model was pretrained on RGB images with so much variation of images, but the X-Ray images doesn't variate quite a much as the Imagenet Dataset varies. So, the model cannot update parameters of the feature extractors based upon the loss values of predictions, and this might drive the model to a slightly worse performance than the Resnet model with full parameter range gradient calculation across the training set.

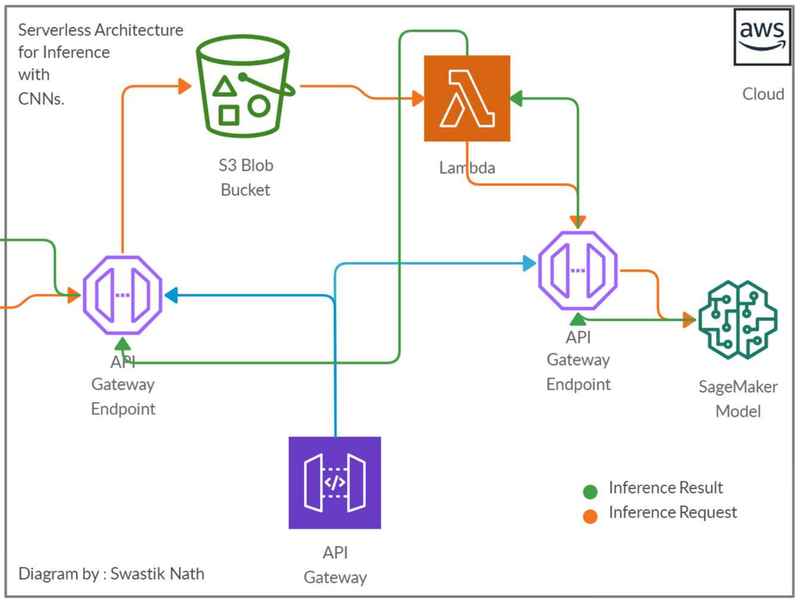
EVALUATION METRICS:
To accurately evaluate how our models are realising we used the metrics namely training set batch loss, training set accuracy, test set batch loss and test set accuracy. In our architectural decision scenario, the benchmark VGG19 model with the features extractor layer's gradient calculation set to false, we get to see that the average training loss is around 48.7% for the last epoch and average test loss in around 22%, with the test accuracy set to 78%.
In the case of SageMaker Deployment of the VGG19 model where we have not freeze the model parameters and have used the version 2.0 and the version 3.0 dataset, we get around 19.6% of test loss and 80.3% test accuracy. So, there we can see a bit of accuracy improvement over the benchmark model.

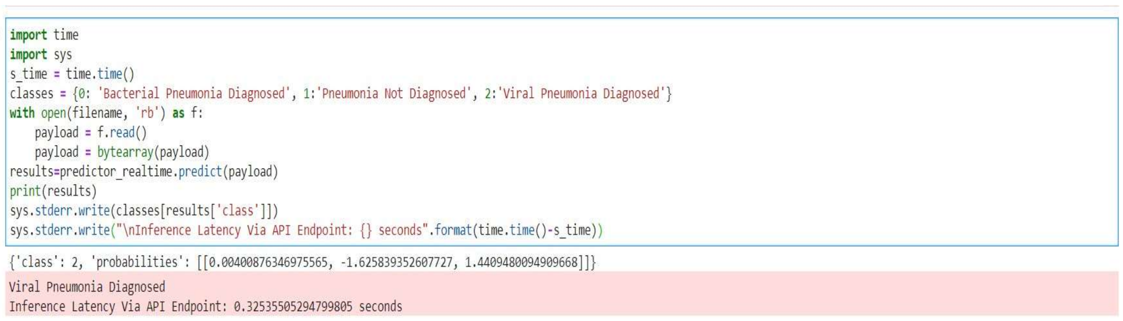
But in the case of our Resnet 50 model, we get to achieve around 53.6% but the test loss in around 21% with the test accuracy set to 79%. In the case of the SageMaker deployment of the model where we have not set the parameter gradient calculation to false leading to parameter freezing and have used both the Version 2.0 and Version 3.0 dataset alongside, we got around 82% test set accuracy and 19% of test set loss.

PROJECT DESIGN:

We desired to create an end-to-end model so, we trained and deployed the model using the AWS SageMaker. At the very first of all we combined the Version 2.0 and Version 3.0 dataset together, transformed each image into a PIL Image, resized each image so that the internal matrix multiplications never run into error, and then transformed them into PyTorch Tensors.

We then used the PyTorch's ImageFolder function to load, label and transform the image data, we then used the DataLoader function of PyTorch to load the images into batches after shuffling them, and next we upload the data to an Amazon S3 Bucket within the region of the notebook instance. We then go on to create SageMaker PyTorch Estimators and pass in the training script to follow the PyTorch Model training instructions, basically it contains all the details about the training procedures of the model after which we direct the S3 bucket location of both the training and test dataset as we uploaded them to the S3 bucket before and call the .fit method of the PyTorch Estimatior object. After the training is over, we compile a SageMaker PyTorchModel object to send inference request to it in real time and we pass in an inference script which includes the instructions to handle the input stream of bytearray as images and process them and then invokes the deployment endpoint. We have created the inference script in such a way that it provides the inferred class and the confidence of the model in the prediction.
The Lambda function can be invoked when an object is uploaded to the S3 and that object is then sent to the lambda function and it performs the pre-processing over the byte array object and then performs different transformations and send the result to a CloudWatch Log.

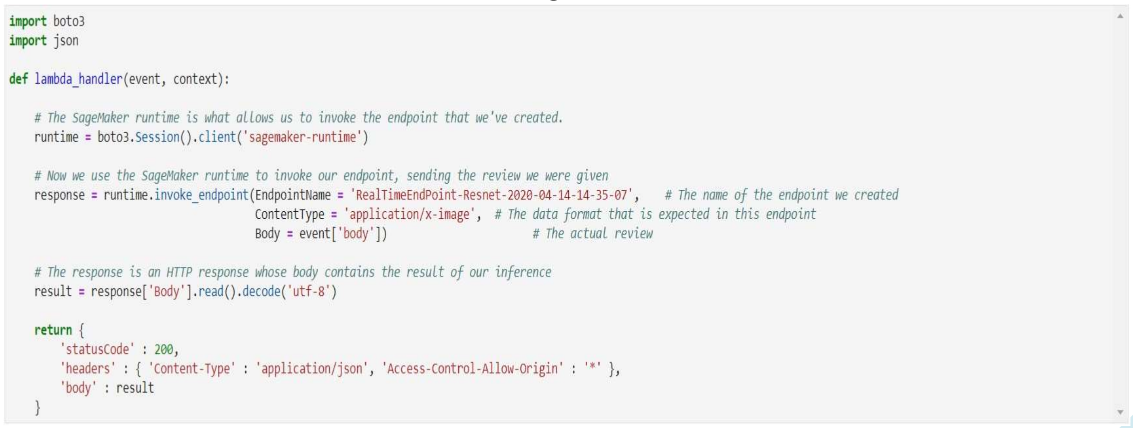




```
import time
import sys
s_time = time.time()
classes = {0: 'Bacterial Pneumonia Diagnosed', 1:'Pneumonia Not Diagnosed', 2:'Viral Pneumonia Diagnosed'}
with open(filename, 'rb') as f:
    payload = f.read()
    payload = bytearray(payload)
results=predictor_realtime.predict(payload)
print(results)
sys.stderr.write(classes[results['class']])
sys.stderr.write("\nInference Latency Via API Endpoint: {} seconds".format(time.time()-s_time))
```

```
{'class': 2, 'probabilities': [[0.00400876346975565, -1.625839352607727, 1.4409480094909668]]}
Viral Pneumonia Diagnosed
Inference Latency Via API Endpoint: 0.32535505294799805 seconds
```

We have also devised a Lambda Function which enables the endpoint to receive asynchronous inference request via the API gateway.



```
import boto3
import json

def lambda_handler(event, context):

    # The SageMaker runtime is what allows us to invoke the endpoint that we've created.
    runtime = boto3.Session().client('sagemaker-runtime')

    # Now we use the SageMaker runtime to invoke our endpoint, sending the review we were given
    response = runtime.invoke_endpoint(EndpointName = 'RealTimeEndPoint-Resnet-2020-04-14-14-35-07',    # The name of the endpoint we created
                                       ContentType = 'application/x-image',  # The data format that is expected in this endpoint
                                       Body = event['body'])                 # The actual review

    # The response is an HTTP response whose body contains the result of our inference
    result = response['Body'].read().decode('utf-8')

    return {
        'statusCode' : 200,
        'headers' : { 'Content-Type' : 'application/json', 'Access-Control-Allow-Origin' : '*' },
        'body' : result
    }
```

CITATIONS:

1. Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), "Large Dataset of Labeled Optical Coherence Tomogra Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), "Large Dataset of Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images", Mendeley Data, v3phy (OCT) and Chest X-Ray Images", Mendeley Data, v3

2. Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification", Mendeley Data, v2

3. Kermany, Daniel; Goldbaum, Michael; et. al "Identifying Medical Diagnosis and Treatable Diseases by Image-Based Deep Learning", Cell, Volume 172, Issue 5, P1122-1131.E9, February 22, 2018, CellPress : DOI: https://doi.org/10.1016/j.cell.2018.02.010

4. Yadav, S.S., Jadhav, S.M. Deep convolutional neural network based medical image classification for disease diagnosis. J Big Data 6, 113 (2019). https://doi.org/10.1186/s40537-019-0276-2