Chest X-Ray images (Pneumonia)

Presenter: Pegah Khazaei
February 2021
Professor: Dr. Hossein Hajiabolhassan
Teaching Assistant: Mr. Khodayari
Course: Deep Learning

Shahid Beheshti University
Department of Mathematics and Computer Science

Introduction

According to the World Health Organization (WHO), pneumonia kills about 2 million children under 5 years old every year and is consistently estimated as the single leading cause of childhood mortality, killing more children than HIV/AIDS, malaria, and measles combined. The WHO reports that nearly all cases (95%) of new-onset childhood clinical pneumonia occur in developing countries, particularly in Southeast Asia and Africa. Bacterial and viral pathogens are the two leading causes of pneumonia but require very different forms of management. Bacterial pneumonia requires urgent referral for immediate antibiotic treatment, while viral pneumonia is treated with supportive care. Therefore, accurate and timely diagnosis is imperative. One key element of diagnosis is radiographic data, since chest X-rays are routinely obtained as standard of care and can help differentiate between different types of pneumonia. Pneumonia is diagnosed in many ways, one common way of confirmation is through chest X-rays. Chest X-rays are the best tests, and most accurate, to determine if one has pneumonia. While it is crucial, detecting pneumonia can sometimes be a difficult task. Pneumonia often vaguely shows up in X-rays and can also get mixed in with other diseases present in that area.

Description of the dataset

The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal). Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care. For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

The dataset consists of only very few samples and that too unbalanced. The aim of this kernel is to develop a robust deep learning model from scratch on this limited amount of data. We all know that deep learning models are data hungry but if you know how things work, you can build good models even with a limited amount of data. Machine learning has a phenomenal range of applications, including in health and diagnostics. This tutorial will explain the complete pipeline from loading data to predicting results, and it will explain how to build an X-ray image classification model from scratch to predict whether an X-ray scan shows presence of pneumonia. This is especially useful during these current times as COVID-19 is known to cause pneumonia.

To prepare the images for our network, we have to resize them to 224 x 224 and normalize each color channel by subtracting a mean value and dividing by a standard deviation. We will also augment our training data in this stage. These operations are done using image transforms, which prepare our data for a neural network.

Data Augmentation

To get more data, we just need to make minor alterations to our existing dataset. Minor changes such as flips or translations or rotations. Our neural network would think these are distinct images anyway. A convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called invariance. More specifically, a CNN can be invariant to translation, viewpoint, size or illumination (Or a combination of the above). This means for training, we randomly resize and crop the images and also flip them horizontally. A different random transformation is applied each epoch (while training), so the network effectively sees many different versions of the same image. All of the data is also converted to Torch Tensors before normalization. The validation and testing data is not augmented but is only resized and normalized. The normalization values are standardized for Imagenet.

Data Iterators

To avoid loading all of the data into memory at once, we use training DataLoaders. First, we create a dataset object from the image folders, and then we pass these to a DataLoader.

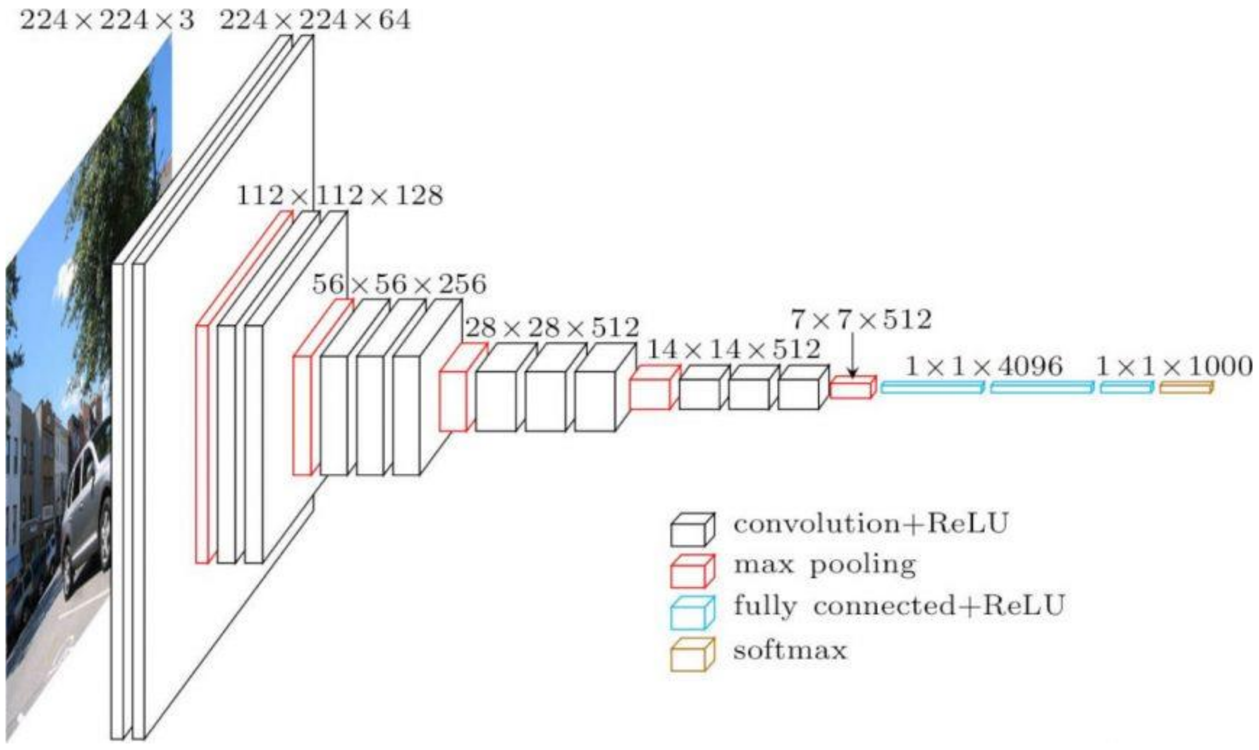
Using Pre-Trained Models for Image Classifications

PyTorch has many pretrained models we can use. All of these models have been trained on Imagenet which consists of millions of images across 1000 categories. What we want to do with pretrained models is freeze the early layers, and replace the classification module with our own.

Approach

- Load in pre-trained weights from a network trained on a large dataset
- Freeze all the weights in the lower (convolutional) layers
- Layers to freeze can be adjusted depending on similarity of task to large training dataset
- Replace the classifier (fully connected) part of the network with a custom classifier
- Number of outputs must be set equal to the number of classes
- Train only the custom classifier (fully connected) layers for the task
- Optimizer model classifier for smaller dataset

Vgg16



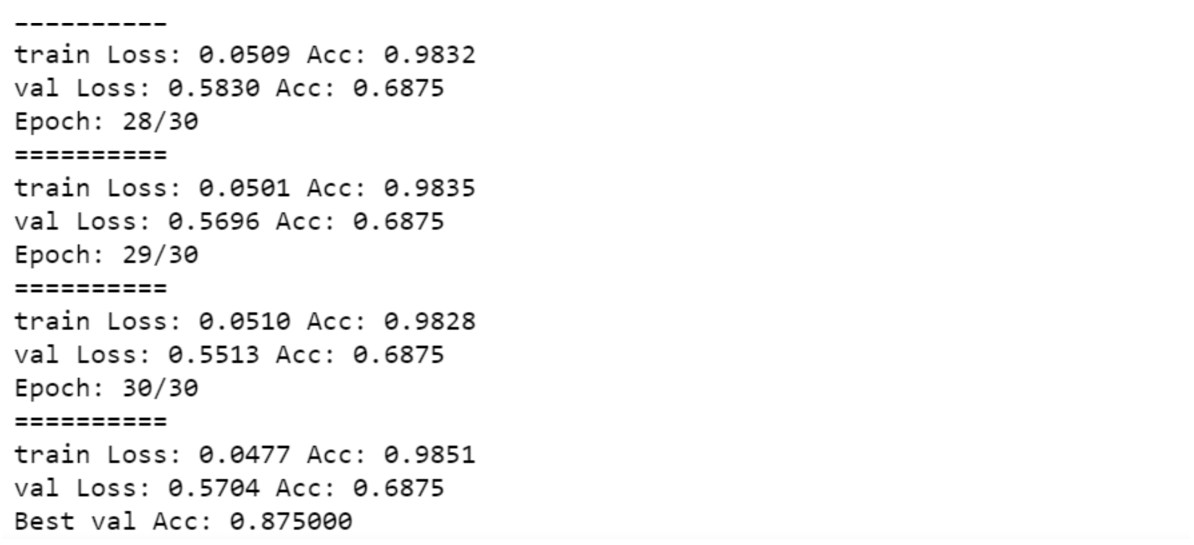
224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Training the model

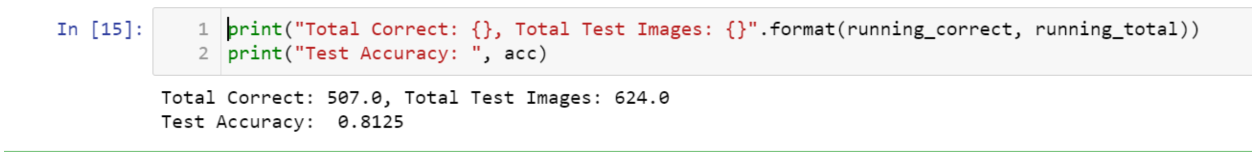
For training, we iterate through the train DataLoader, each time passing one batch through the model. One complete pass through the training data is known as an epoch, and we train for a set number of epochs or until early stopping kicks in (more below). After each batch, we calculate the loss (with criterion(output, targets)) and then calculate the gradients of the loss with respect to the model parameters with loss.backward(). This uses autodifferentiation and backpropagation to calculate the gradients.

After calculating the gradients, we call optimizer.step() to update the model parameters with the gradients. This is done on every training batch so we are implementing stochastic gradient descent (or rather a version of it with momentum known as Adam). For each batch, we also compute the accuracy for monitoring and after the training loop has completed, we start the validation loop. This will be used to carry out early stopping.

Now we will test our model and evaluate Model over all classes.



```
----------
train Loss: 0.0509 Acc: 0.9832
val Loss: 0.5830 Acc: 0.6875
Epoch: 28/30
==========
train Loss: 0.0501 Acc: 0.9835
val Loss: 0.5696 Acc: 0.6875
Epoch: 29/30
==========
train Loss: 0.0510 Acc: 0.9828
val Loss: 0.5513 Acc: 0.6875
Epoch: 30/30
==========
train Loss: 0.0477 Acc: 0.9851
val Loss: 0.5704 Acc: 0.6875
Best val Acc: 0.875000
```



```
In [15]: print("Total Correct: {}, Total Test Images: {}".format(running_correct, running_total))
         print("Test Accuracy: ", acc)
```

```
Total Correct: 507.0, Total Test Images: 624.0
Test Accuracy:  0.8125
```