Stanford MURA Competition

Day 1 Overview

In an attempt to team up with others, I had the pleasure to work with another fast.ai student, Rachana. Explaining things to each other had certainly helped to make both of us understand the competition better. Below are what I have learned about the dataset after playing around it for a while and discussing it with Rachana.

Challenges

- The basic training unit is not an individual X-Ray image but an individual X-Ray study that could consist of multiple images, each representing a different view on the same body part. An image does not have a label while a study does.
- The images inside the datasets have varying shapes, dimensions, and padding sizes.

Countermeasures

- Label each individual image according to the label of the study they belong to, then train the model using an individual image as basic training unit.
  - Pro: Easy to start. `ImageDataBunchworks` out of the box. At inference time, we can just take in all the images of the study, then aggregate them to produce a final result.
  - Con: It could be the case that even in positive studies, not all the images look positive, i.e., abnormal. The body part might only abnormal from one perspective but perfectly normal in all others. So, it is technically impossible for the model to really tell that the individual image is abnormal. As a result, many of the training data will be mere noises.
- Merge all the images in a study into one single image.
  - Pro: Easy to start. `ImageDataBunchworks` out of the box. Inference time is trivial as well because now the basic training unit is correctly an individual study.
  - Challenges: What would be a good way to merge the images together?
- Build a custom architecture that actually takes in multiple images as input.
  - Challenge
    - Architectural design
    - Studies have varying numbers of images

General Strategies

- You can train an end-to-end deep learning system that simply takes in an X-Ray image from an arbitrary body part and tells if it is abnormal.
- Since in inference time, the body-part of the image is given, we can train individual CNN

models for each body part.

Fleeting Jubilation

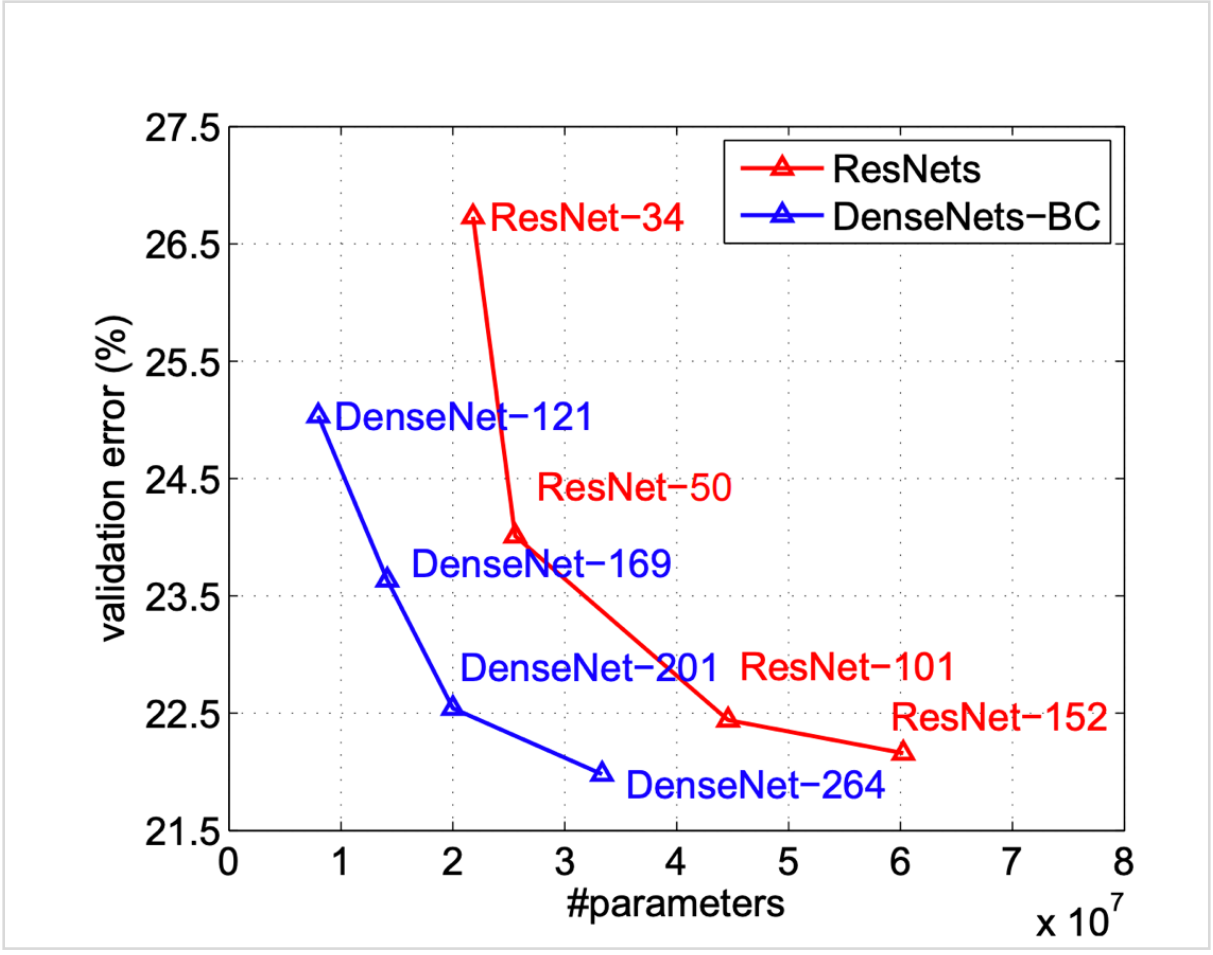
There is actually a side story I would like to share with you. At the beginning, I did not realize that the score in the leaderboard used Cohen's Kappa, a much stricter metrics than plain accuracy, which I thought the board used. So, when I ran the out-of-box fast.ai training loop on the dataset and got a near 0.8 score, which would place me in the top 10 had it really been the right metrics, I was exhilarated. Of course, the sensation did not last long as I soon discovered that I used the wrong metrics. Still unaware how much stricter Cohen's Kappa score is relative to accuracy, I was still hoping that maybe my model would still do alright. When the score came out, it shattered all my hopes. I got a miserably failing Cohen's Kappa, below the 0.60 line and placed right at the bottom of the leaderboard. Well, the shock did last for quite a while, but eventually I got back on my feet; after all, what is the point of the competition had it really been that easy?

Day 2,3,4

Initially, I stuck to my original overall assault strategy, that is, labeling individual images according to their study labels, and train the model on the entire dataset as a binary classification problem. I made much effort attempting to fine-tune the training process, hoping that it could take me a bit further. Below are a few strategies I have adopted:

- Experimenting with different learning rates
  - This one has mostly been trial and errors with little to no systematic strategy. Even when I did get a good result, I cannot be sure if it is because of the superiority of the chosen learning rate, or a matter of pure chance.
  - To reduce the noise randomness here, however, I started training more epochs while saving the best model along the way. A few patterns have been clear:
    - If there is too much perturbation in the learning rate, that is, it is spiking up and down all the time, it is probably the right time to pick a smaller learning rate.
    - It is perfectly normal for the model's performance to deteriorate before it significantly improves again.
      - The question still remains whether it is a sign to pick a smaller learning rate.
      - It is also a question of how big a deterioration is too big for the model to ever come back; to put it simply, how do we know if we have blown up the model.
  - Babysitting the model under training and hand-tune the learning rate along the way is a huge expenditure of time. Due to its case-by-case nature and the non-deterministic nature of deep learning training, it is hard to statistically evaluate the merit of all the decisions that I have made and generalize them to future circumstances.

    - Thus, it makes more sense to adopt a more general learning rate strategy that could be followed from the start of the training all the way to the end without any human intervention.
    - Here is a good example of what I mean general training strategy, from Kaiming He's ResNet paper: "The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to 60 × 10^4 iterations".
- Using different CNN architectures
    - Instead of good old ResNet-50, I tried ResNet-101. It did not help much. It makes sense though, as I could already fit the training set extremely well using just ResNet-50, reducing the training loss to 0.003, quite near 0. The problem is mostly overfitting obvious in the huge gap between the training loss and the validation loss, the latter of which I had difficulties even to reduce below 0.5.
    - I also tried a shallower architecture, here the ResNet-34. Surprisingly, it still managed to fit the training data very well. This really makes me think whether ResNet-34 is really deep enough a network to do the job at hand, even though most of the competition participants are using deeper network with more than 100 layers.
        - It needs to be noted here, however, that number of layers could mean very different things in terms of architecture complexities and number of parameters when it comes to different architectures.



    - For example, DenseNet-201 still has less parameters than ResNet-34.

        - If we are using the number of parameters as a measure of the architecture complexity, then in this sense DesNet-201 is actually less complicated than ResNet-34.
- Experimenting with regularization hyper-parmeters
    - Similar to the case with learning rate, the experiments done here are not systematic.
    - Since the biggest problem I came across during the training process is overfitting, here I focused my effort on regularization hyper-parameters, specifically, weight decay rate and dropout rate.
    - The efficacy of the regularization is measured by the gap between the training loss and the validation loss. When I jack up the weight decay rate and the dropout rate, the gap does seem to get smaller. However, as the training continues, the validation loss inevitably blows up again.
    - What starts to make me doubt the implementation of the fast.ai library is that even when I jack up the weight decay rate to a ridiculous 10, there does not seem to be a big difference and the training loss continue to drop while the validation loss keeps skyrocketing. That just does not make any sense.
    - It is probably a good idea to use only the most simple ideas and implement a working model just using better tested framework like PyTorch and TensorFlow.
        - For one thing, fast.ai as it is right now is a bit buggy to be relied upon for correctness.
        - The other issue is that fast.ai library uses a very complicated training strategy with all kinds of papers' recommendations and best practices jammed together; many of the pieces I have little to no clue of and do not have the faintest idea how they interact with each other. Take a simple example, I understand how L2 weight decay works with learning rate, so I would not expect the same strength of regularization when I jack up the learning rate and keep the weight decay rate the same. So when I playing around with learning rate and weight decay, I know how they interact. But that is not true for all the fancy pieces within the fast.ai library. As a result, there is very little I can assume and when something went wrong, it is basically impossible to know where the problem could be.
        - Train the model stating with a very simple strategy whose parts you are well familiar with. Then when something did not work out, you could always add more recipes into it until it works. Less is more. If two approaches obtain similar results, the simpler ones are far more superior.