Project 1: `nescient`

David Freifeld

March 21, 2022

1 About

`nescient` is a convolutional neural network designed to detect the presence of lung lesions in chest radiographs (x-rays). As a secondary goal, it aims to do so while maintaining patient privacy.

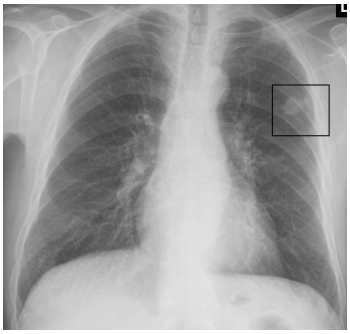

Figure 1: Lung lesions, in the ideal case, appear as circular splotches on a radiograph.

2 Data

Labeled chest radiograph images utilized for the training of `nescient` were sourced from Irvin & Rajpurkar et al.'s CheXpert dataset. More specifically, the variant of CheXpert `CheXpert-v1.0-small` was used for training, which consists of 223,414 grayscale JPEG images of a size of approximately 390×320 pixels. Each image contains an associated row in a CSV where labels representing the presence of twelve pathologies in the image are located alongside additional metadata such as the gender/age of the subject and the perspective of the radiograph image. The presence of each pathology is labeled as either positive (`1.0`), uncertain (`0.0` or `NaN`), or negative (`-1.0`).

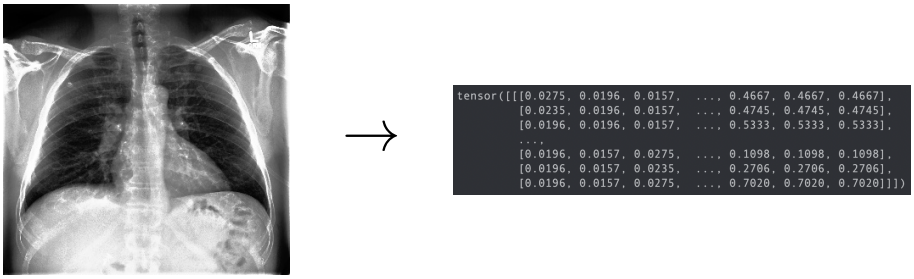

Figure 2: Sample of the data and its tensor form after processing.

2.1 Preprocessing

`nescient` solely attempts to classify the presence of a lung lesion in a chest radiograph, and thus filters the dataset to maintain an even balance of positive and negative examples of lung lesions. Additionally, to narrow the scope of the problem, `nescient` only operates on frontal chest radiographs.

As the number of positive examples is the limiting factor in the CheXpert dataset, the preprocessing script for `nescient` composes a final CSV containing labels for 14,080 images: all 7,040 positive samples, all 1,108 of the negative samples, and 5,932 uncertain samples. Confirmed negative images were unfortunately not abundant in the CheXpert dataset and so images with an uncertain presence of lung lesions were assumed to have none.

2.2 Runtime Transformations

At runtime, before each image is processed, `nescient` combines padding and transpose operations to convert the image to a standard resolution of 320×390 pixels. Images are then finally normalized by dividing the intensity of each pixel value by 255.

The lung lesion label corresponding to the image is also normalized at runtime such that negative labels are `0.0` instead of `-1.0` or `NaN`.

2.3 Train/Validation Sets

Before the model begins training, `nescient` randomly splits the data such that 10% of it is withheld for use as a validation set. Furthermore, the train set is accessed randomly in a different manner each epoch.

3 Architecture

The final architecture of the model was a convolutional neural network, the structure of which can be succinctly described with its implementation in PyTorch:

```
self.model = nn.Sequential(
    nn.Conv2d(1, 8, (10, 10), stride=3),
    nn.ReLU(),
    nn.MaxPool2d(8),
    nn.Flatten(1),
    nn.Linear(1560, 64),
    nn.ReLU(),
    nn.Linear(64, 32),
    nn.ReLU(),
    nn.Linear(32, 1),
    nn.Sigmoid(),
)
```

A convolutional network was most applicable to the problem of detecting lung lesions due to its improved ability to learn patterns in two dimensional data. Immediately flattening the image and inputting into a dense neural network was a possibility, but the sheer size of the input features would either overwhelm the network or force the development of a very bulky model that would likely perform poorly. The usage of a convolution operation between the image and a trainable "filter" is therefore preferable both due to its ability to more directly learn visual patterns and produce outputs of more reasonable dimensionalities. The number of filters was intentionally kept somewhat low to prevent overfitting. The filter size was based of the general size of lung lesions, and the stride was used as a method of reducing output size.

However, a simple translation of the image by a few pixels is enough to change the output of convolutional layer's filters, and so a max pooling layer is used to capture the maximum filter output in a 8x8 region. This way, even if the image has the correct pattern but is shifted over by a few pixels (ordinarily leading to a different input to the next layer), the maximum in the region will remain the same and thus so will the input to the next layer. The max pooling provides some degree of translational invariance, and additionally further reduces the dimensionality of the input to the dense layers.

After flattening the output, it is fed into three dense layers of some (but not too much) width in order to provide the model with enough complexity to properly classify the input as being a lung lesion or not.

The rectified linear unit (ReLU) activation is used throughout the network to provide non-linearity — excluding the last layer which uses a sigmoid to force outputs to be between 0 and 1.

Beyond the concrete structure of the network, the model has three hyperparameters: a learning rate of $5 \cdot 10^{-4}$, weight decay of $1 \cdot 10^{-4}$, and batch size of 8. These values were genrally decided through trial and error during the process of developing the model.

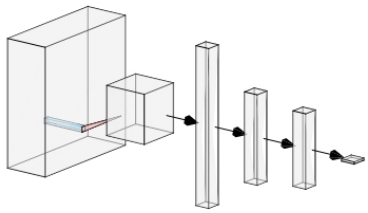

Figure 3: Simple visualization of the final architecture.

4 Results

4.1 Training Sample

```
Epoch 0/300: train loss 0.689, train acc 53.456%, val loss 0.675, val acc 57.599
Epoch 1/300: train loss 0.68, train acc 57.26%, val loss 0.669, val acc 61.435
Epoch 2/300: train loss 0.673, train acc 58.759%, val loss 0.664, val acc 60.511
Epoch 3/300: train loss 0.669, train acc 59.257%, val loss 0.657, val acc 61.79
Epoch 4/300: train loss 0.665, train acc 59.541%, val loss 0.658, val acc 62.926
Epoch 5/300: train loss 0.663, train acc 59.904%, val loss 0.661, val acc 61.009
Epoch 6/300: train loss 0.66, train acc 60.764%, val loss 0.656, val acc 62.5
Epoch 7/300: train loss 0.657, train acc 60.977%, val loss 0.657, val acc 63.139
Epoch 8/300: train loss 0.655, train acc 61.679%, val loss 0.664, val acc 60.653
Epoch 9/300: train loss 0.652, train acc 61.529%, val loss 0.656, val acc 62.429
Epoch 10/300: train loss 0.648, train acc 61.987%, val loss 0.671, val acc 61.222
Epoch 11/300: train loss 0.644, train acc 62.516%, val loss 0.663, val acc 61.861
Epoch 12/300: train loss 0.64, train acc 62.61%, val loss 0.66, val acc 62.571
Epoch 13/300: train loss 0.635, train acc 63.518%, val loss 0.676, val acc 59.801
Epoch 14/300: train loss 0.629, train acc 64.276%, val loss 0.676, val acc 59.801
Epoch 15/300: train loss 0.623, train acc 65.033%, val loss 0.682, val acc 59.943
Epoch 16/300: train loss 0.616, train acc 65.578%, val loss 0.675, val acc 62.003
Epoch 17/300: train loss 0.61, train acc 66.209%, val loss 0.69, val acc 60.795
```

```
Epoch 18/300: train loss 0.601, train acc 67.306%, val loss 0.711, val acc 60.227
Epoch 19/300: train loss 0.591, train acc 67.858%, val loss 0.688, val acc 60.014
Epoch 20/300: train loss 0.58, train acc 68.426%, val loss 0.733, val acc 58.736
...
Epoch 282/300: train loss 0.035, train acc 98.824%, val loss 6.053, val acc 53.125
Epoch 283/300: train loss 0.035, train acc 98.943%, val loss 5.508, val acc 53.125
Epoch 284/300: train loss 0.04, train acc 98.801%, val loss 5.964, val acc 53.977
Epoch 285/300: train loss 0.039, train acc 98.769%, val loss 5.899, val acc 54.688
Epoch 286/300: train loss 0.038, train acc 98.69%, val loss 5.208, val acc 52.273
Epoch 287/300: train loss 0.034, train acc 98.99%, val loss 5.639, val acc 53.267
Epoch 288/300: train loss 0.034, train acc 98.824%, val loss 5.614, val acc 53.409
Epoch 289/300: train loss 0.045, train acc 98.556%, val loss 5.743, val acc 53.338
Epoch 290/300: train loss 0.039, train acc 98.785%, val loss 6.317, val acc 53.338
Epoch 291/300: train loss 0.031, train acc 98.966%, val loss 5.901, val acc 53.338
Epoch 292/300: train loss 0.05, train acc 98.461%, val loss 6.274, val acc 54.688
Epoch 293/300: train loss 0.027, train acc 99.187%, val loss 5.818, val acc 51.918
Epoch 294/300: train loss 0.038, train acc 98.706%, val loss 6.773, val acc 53.693
Epoch 295/300: train loss 0.033, train acc 98.872%, val loss 5.367, val acc 55.54
Epoch 296/300: train loss 0.048, train acc 98.603%, val loss 5.206, val acc 53.551
Epoch 297/300: train loss 0.031, train acc 98.935%, val loss 5.422, val acc 53.338
Epoch 298/300: train loss 0.044, train acc 98.737%, val loss 5.995, val acc 55.398
Epoch 299/300: train loss 0.036, train acc 98.943%, val loss 5.284, val acc 53.054
```

4.2 Metrics

Recall is a performance metric that measures the rate of positive samples predicted correctly: that is to say it is $\frac{\text{True positives}}{\text{True positives+False negatives}}$. Recall is particularly suited to the task of detecting lung lesions because false negatives are potentially much more consequential than false positives: lung lesions can be cancerous and not detecting a cancerous mass could have fatal consequences.

At the same time, a very high rate of false positives will overload medical staff and so it is useful to have some level of *precision*, which is how often a model is correct when it outputs positive (or $\frac{\text{True positives}}{\text{True positives+False positives}}$).

`nescient` has a recall of 98.9% and a precision of 98.8% on its training set. On its validation set it has a peak recall of 60.5% and a precision of 72.7% , and while this is largely because the model does not generalize (having a peak validation accuracy of 63%), this potentially indicates the model is problematically biased towards false negatives. Experimenting with the decreasing the threshold for positive classification is one possible method of minimizing false negatives (at the cost of more false positives).

5 Process

5.1 Attempts

5.1.1 Wider Scope

Originally, `nescient` attempted to detect the presence of each of the pathologies labeled for a given image, which proved to add considerable complexity to the problem. CheXpert is not a usual multiclass classification task, as multiple classes can be true/false at the same time. Picking a loss function was therefore not

immediately clear.

The original combination of softmax and cross-entropy loss performed very poorly. The problem with this combination was both that softmax was a nonsensical choice for a problem where two classes could be positive at the same time (as softmax forces outputs to sum to one) and that cross-entropy is best suited for outputs that sum to one. The final layer was then switched to a normal linear layer with a sigmoid activation and MSE (mean squared error) was used for the loss function.

Despite these adjustments, prototype models still performed poorly. As a method of reducing complexity given the amount of time for the project, `nescient` instead switched to classifying the presence of a specific pathology: lung lesions (and as a result now uses binary cross entropy).

5.1.2 Exploding Gradients

Initial architectures were vastly too complex for the task of detecting lung lesions, containing multiple convolutional layers and many dense layers. Additionally, the image was not normalized at this point in time, so the combination of large inputs with a overly large network led to the model having massive activations. This in turn meant that during the backpropagation step, the weight updates were also extremely large — these "exploding gradients" quickly drove the model to solely output zero. Some immediate attempts at counteracting this were made before the more permanent solution was found.

- *Gradient clipping* modifies the gradient before backpropagation by imposing a maximum value and clipping all values in the gradient above it to that maximum.

- *GELU* is a variant of the ReLU activation that can be described as having a smooth dip in the low negatives. This dip allows for some recoverability if a large weight updates causes once positive activations to go negative, whereas in ReLU they would be zeroed out and the information lost.

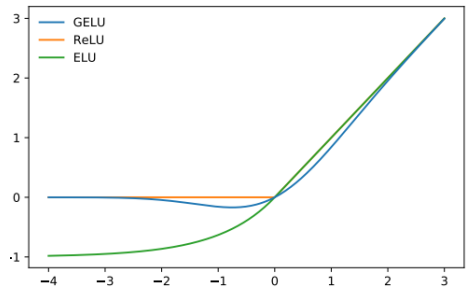

Figure 4: GELU activation compared to RELU and ELU.

- *Input normalization* was a matter of simply dividing each pixel in the image by 255 to have the input on a smaller scale from 0 to 1 to prevent large gradients

Finally, improvement came from drastic simplification of the model architecture from multiple convolutional and dense layers to a single convolutional layer and two dense layers.

5.1.3 Generalization

Once the model utilized a simpler architecture, it was capable of overfitting to the dataset, yet performed very poorly on the validation set. Two general techniques for improving the validation performance of the

model were attempted:

- *Dropout* randomly "drops out" nodes in a layer given a probability, causing them to be ignored by the network. This prevents the network from overrelying on specific features (which could be a sign of overfitting) and forces the activations of the network to become more sparse.
- *L2 regularization* is a method of regularization that essentially adds the sum of the weights squared to the loss (times a constant known as the *weight decay*). This prevents weights from getting too large, and by extent, prevents the network from overrelying on anything.

Beyond these factors, there was a wide amount of experimentation with model architecture based off of various hypotheses as to why the model was not generalizing:

- Perhaps the model did not have the complexity to properly classify the patterns of lung lesions, and was just slowly fitting what it had to the training set. The complexity and number of the linear was then experimented with as a result, yet did not achieve significantly higher validation accuracy.
- Perhaps the patterns of lung lesions were varied enough that the convolutional layer's filters were unable to completely reflect them? Increasing the number of filters was then attempted to no avail.
- The size of the max pooling was somewhat intense in order to reduce the number of input features for the first dense layer — perhaps this was preventing the model from learning?. Yet, reducing max pooling also had little beneficial effect.
- Perhaps lung lesions are on average larger in scale than anticipated? Increasing kernel size of the convolutional layers was attempted, yet also did not improve performance.
- Perhaps the model was still too complex, and dense layers should be substantially narrower? Simplifying the model through less filters and narrower dense layers still did not yield any better results.

The above list is but a short sample in the number of things attempted in the somewhat ardous attempt to improve the generalization of `nescient`.

5.2 Next Steps

If this project continued for longer, the most immediate next step would be to continue to attempt to improve the generalizability of the model. A more thorough analysis of the data to ensure its quality could better inform the trial-and-error process. Furthermore, data augmentation could increase the size of the dataset, which would likely also help the model generalize. Were a good validation accuracy to be achieved, fine-tuning the model to improve recall in the validation set would likely follow.

Beyond improving performance of the model, an original goal of the project was to have a secure model for processing medical data. As a result, further developing utilities for secure training of the model is another possible route for a continuation of the project.

Finally, the I/O and training times of the model is another aspect that could be improved. At times much of the runtime of `nescient` is dominated by file I/O, and during training the GPU could likely be better utilized.

5.3 Insights

The primary insight into the problem originally was the simplicity of the ideal case: lung lesions should theoretically just be circles. Unfortunately, this was soon followed by another insight: most of the data was not the ideal case and the presence of lung lesions was not very clear to the human eye. The problem gradually appeared to be simultaneously simple and complex as it involved picking out a theoretically simple pattern in a complex and noisy scene.

5.4 Feedback

I discussed many of my problems concerning generalization and exploding gradients with others.

- I discussed the lack of performance of my model with Matt originally, who suggested I try simplifying the convolutional portion of my network.
- Jack Liu pointed out some of the original problems with my loss and output layer when I was still attempting to detect each of the pathologies, and also suggested I cut down on model complexity.
- Zachary Sayyah helped me brainstorm variations of the model architecture to attempt to improve generalizability.

These pieces of feedback helped me greatly in pushing me towards simplifying the model architecture and exploring many of the options for improving the generalizability of the model.

6 Use/Ethics

This model is intended to be used for the automated detection of lung lesions. Medical facilities could incorporate `nescient` into their systems to catch lung lesions more often.

Privacy is the main ethical concern in many machine learning models operating upon medical data. To address this, `nescient` uses the `CrypTen` library for encrypting trained models and performing inference on encrypted inputs. This is possible due to *homomorphic* encryption schemes which allow for mathematical operations to be done upon encrypted values. While the privacy featues of `nescient` are underdeveloped due to issues with model performance stalling much of development, `CrypTen` does allow for some degree of encrypted training, and thus the model can be further tuned while maintaining privacy.