Udaikaran Singh and Wesley Kwan

Gary Cottrell

CSE 190

11/11/2018

**PA3: Deep Convolutional Network for Thorax Disease Detection**

**Udaikaran Singh Wesley Kwan**

Department of Computer Science Department of Computer Science

University of California, San Diego. University of California, San Diego

La Jolla, CA 92093. La Jolla, CA 92093

**Abstract**

**(…)**

**Introduction**

The problem this report aims to address to use x-ray imaging to detect and diagnose different diseases. The problem entails recognizing the different patterns that diseases tend to show.

In order to achieve this, we used a convolutional neural network, because of their known ability to perform well on images. Also, the 1024 x 1024 input images would only allow for a fully connected neural network that has too many parameters to train well on. Also, the internal representations of such networks are too complicated for humans to understand. Convolutional neural networks, on the other hand, train well on large image inputs and the filters learned by the network can be used to by humans to better understand disease patterns.

We test 3 models initially, along with testing some other changes to aid the model learning from the such, as reducing the input image size, normalizing the input images, and changing the objective function. The baseline architecture has 3 stacked convolutional layers, a maxpool layer, and then a 2 layer fully connected neural network. Architecture 1 tests the idea of reducing the numbers of filters within the convolutional layer and adding a fully connected layer. The idea of this is to see whether more parameters in the fully connected layer would aid the model. On the other hand, in architecture 2, we added more convolutional layers, and a maxpool to reduce dimensionality within the convolutional layers. The idea behind this architecture is that depth is more important for learning the images, and the maxpool will serve to reduce too many parameters to be learned.

We initialed the weights in our network with Xavier weight initialization. This method is made to prevent the activation at each neuron in our neural network from exploding or shrinking. The Xavier weight initialization creates the weights based on a gaussian distribution with a mean of 0 and variance of (1/N) where N is the average between number of neurons that weight connects. The motivation behind this is that we want to the neural network to not be greatly impacted by small changes. By having a gaussian distribution with mean 0, we force the network to have an activation at each neuron that is close to 0. Therefore, as the network learns from data, it does not have many exploding or shrinking activations. This theoretically helps to smooth the learning curve.

The experiments we performed were to find the relative performance of the different neural architectures to gain a sense of how impactful the depth of the convolutional layers and the size of the fully connected layers is. Also, our experiments aim to see how image pre-processing would impact the effectiveness of the neural network. Lastly, we wanted to see the impact that a weighted loss function would have on the performance of the neural network.

One of the main issues faced when learning from this dataset is that there is a large imbalance between the occurrences of positive and negative outputs. Most people are not diagnosed with a disease, and if they are, they usually have less than half of the diseases. Therefore, based on the loss function, the neural network would learn this imbalance, and most likely predict that a person does not a disease. Even though this does allow the network to have a high accuracy, not much is learned. To remedy this, we weighted the loss function such that occurrences of false negatives, meaning predictions of no disease when a person does have a disease would be punished harshly within the loss function.

To measure to results of our neural networks, we tested then on a test set of unseen x-ray images and tracked the statistics: accuracy, precision, recall, and balanced classification rate. The goal of this is to see if we can decrease the occurrences of false negatives (not predicting a disease, when one exists), while not causing a large rise in false positives (predicting a disease exits, when one does not). Also, we implemented a pseudo-confusion matrix that plots to see which diseases were mistaken for each other. Even though this does not follow the strict mathematical definition of a confusion table, it serves it’s purpose of allowing us to see what the neural network confuses.

**Related Works**

**(…)**

**Methods**

(i) Baseline Architecture

Baseline Architecture

|  |  |
| --- | --- |
| Layer (from input to output) | Description of Layer: |
| Input Layer | The input image is 1024 x 1024 x 1.  The image is in greyscale. |
| Convolutional Layer 1 | In-channel = 1  Out-channel = 12  Kernel size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 2 | In-channel = 12  Out-channel = 10  Kernel size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 3 | In-channel = 10  Out-channel = 8  Kernel size = 6  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Max-Pool Layer | Kernel size = 3  Stride = 3 |
| Fully Connected Layer 1 | In-features = 121032  Out-features = 128  Activation Function = relU  Batch Normalization Applied |
| Fully Connected Layer 2 (output) | In-features = 128  Out-features = 14  Activation Function = Sigmoid |

The loss criterion used is a binomial cross entropy function. The weight parameters were initialized using Xavier weight initialization. The gradient descent optimization used was the adam optimizer. No regularization was added to the model. We dealt with the class imbalance issue by implementing a weights loss function that punishes false negatives. The motivation of this is that this motivates the model to learn rare cases. We implemented cross validation by leaving out 10% of the training set and testing against it to determine whether the model is overtraining.

(ii) Experimental Architecture

Experimental Architecture 1

|  |  |
| --- | --- |
| Layer (from input to output) | Description of Layer: |
| Input Layer | The input image is 1024 x 1024 x 1.  The image is in greyscale. |
| Convolutional Layer 1 | In-channel = 1  Out-channel = 4  Kernel size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 2 | In-channel = 4  Out-channel = 8  Kernel size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 3 | In-channel = 8  Out-channel = 12  Kernel size = 6  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Max-Pool Layer | Kernel Size = 4  Stride = 4 |
| Fully Connected Layer 1 | In-features = 178608  Out-features = 512  Activation Function = relU  Batch Normalization Applied |
| Fully Connected Layer 2 | In-feature = 512  Out-feature = 128  Activation Function = relU  Batch Normalization Applied |
| Fully Connected Layer 3 (output) | In-features = 128  Out-features = 14  Activation Function = Sigmoid |

Experimental Architecture 2

|  |  |
| --- | --- |
| Layer (from input to output) | Description of Layer: |
| Input Layer | The input image is 1024 x 1024 x 1.  The image is in greyscale. |
| Convolutional Layer 1 | In-channel = 1  Out-channel = 16  Kernel Size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 2 | In-channel = 16  Out-channel = 14  Kernel Size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 3 | In-channel = 14  Out-channel = 12  Kernel Size = 8  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Max-Pool Layer | Kernel Size = 3  Stride = 3 |
| Convolutional Layer 4 | In-channel = 12  Out-channel = 10  Kernel Size = 6  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Convolutional Layer 5 | In-channel = 10  Out-channel = 8  Kernel Size = 6  Zero-Padding = 0  Stride = 1  Activation Function = relU  Batch Normalization Applied |
| Max-Pool Layer | Kernel Size = 3  Stride = 3 |
| Fully Connected Layer 1 | In-feature = 20808  Out-feature = 128  Activation Function = relU  Batch Normalization Applied |
| Fully Connected Layer 2 (output) | In-feature = 128  Out-feature = 14  Activation Function = sigmoid |

For both of these architectures, we used the adam gradient descent optimizer, no regularization function, and Xavier weight initialization. The class imbalance was addressed best testing a weighted loss function that punishes false negatives heavily. The idea is that the model will learn to guess negative for each disease, so by increasing the punishment for this, this motivates the better learning of rare classes. We implemented cross validation by leaving out 10% of the training set and testing against it to determine whether the model is overtraining.

**Results**

[describe implementation of results]

Experiment 1: Baseline Architecture

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

Experiment 2: Architecture 1 (w/o addressing class imbalance)

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

Experiment 3: Architecture 2 (w/o addressing class imbalance)

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

Experiment 4: Architecture 2 with Normalized Inputs

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

Experiment 5: Architecture 2 with Scaled Down Image (by ½)

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

Experiment 6: Architecture 2 with weighted objective function

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

Experiment 7: Architecture 2 with weighted objective function & Normalized Inputs

(i) Loss Curves

(ii) Accuracy Curves

(iii) Visualization of filter maps

(iv) Model Results

**Discussions**

**(…)**

**References**

**(…)**

**Authors’ Contributions**

**(…)**