Convolutional neural networks (CNNs) on Pneumonia Detection through Chest X-rays

Joy Zeng

2020-12-04

Overview

Introduction

- **Background**:
  - Pneumonia is an infection of the lungs
  - Diseases such as COVID-19, SARS onset pneumonia in both lungs
  - Chest X-rays is often used in detecting lung infection
- **Data**:
  - Chest X-ray that are labeled with either healthy or pneumonia
  - https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset

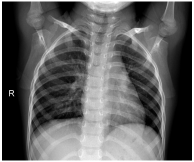

Figure: An Example Image in the dataset

  - **training set**: 5286 images (3944 Pneumonia)
    test set: 624 images (390 Pneumonia)
- **Goal**: classify the X-rays into Healthy vs Pneumonia

Data Preprocessing

1. **Resized** all of the images to the size of 300×300
   - Since model training requires all the images are of the same size
2. **Normalized** the images before modeling (substract both train/test sets by empirical mean and divided by standard deviation **computed over the training data**)
   - Classification loss is sensitive to the changes in weight matrix
3. Split training data into $\begin{cases} 50\% \ \textbf{training set} \\ 50\% \ \textbf{validation set} \end{cases}$

Models (Supervised learning framework)

Supervised learning framework:

- Given a dataset: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, the goal is to train a predictive function $f_{\boldsymbol{\theta}}(\cdot)$ so that $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ will be able to predict labels y_i (healthy or Pneumonia).
- Define **Loss** (error measurement) as: $L(\boldsymbol{\theta}) = \frac{1}{N}\sum_i L_i\big(f(\boldsymbol{x}_i; \boldsymbol{\theta}), y_i\big)$ where $L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$ (**Cross-entropy loss**/Softmax classifier)
- **Optimization**:
  We want to find a set of parameters that minimize the loss function: $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{n}\sum_{i=1}^{N} L\big(y_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\big)$
  - Most basic algorithm: **stochastic gradient descent**
  - SGD performs poorly on high-dimentional non-convex optimization. so we used the improved version, **Adam**, [Kingma and Ba (2014)]
- **Regularization** to prevent overfitting (more details later)
- f here is CNN models, and $\boldsymbol{x}$'s are matrix representation of pixels

Models (Backprop)

- The key of SGD algorithm is **computing the gradients** of the loss with respect to the model parameters.
  $L(\boldsymbol{\theta}) = \frac{1}{N}\sum_i L_i\big(f(\boldsymbol{x}_i; \boldsymbol{\theta})\big)$
  $\nabla_\theta L(\theta) = \frac{1}{N}\sum_i \nabla_\theta L_i\big(f(\boldsymbol{x}_i; \theta), y_i\big)$
- Hard to compute gradients in deep networks
- Use **Backpropagation** to compute the gradients
  - recursively applies the chain rule during the backward pass to compute the gradients

Architecture of CNN:
A series of convolutional layer + pooling layer, added by fully connected layers at the end.

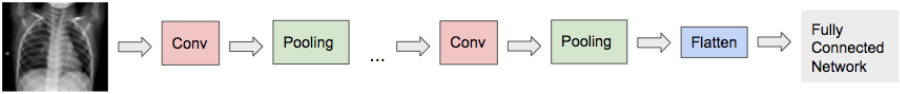

Figure: The General Architecture of CNN

1. Use **ReLU** as our activation function for non-saturating nonlinearity
2. Weight Initialization: use **He Init** specifically derived for ReLU, [He et al. (2015)]
3. Use **Batch normalization** to improve generalization on the test set (Regularization), [Loffe and Szegedy (2015)]
   - Normalizes each layer with zero-mean and unit variance
   - Robust to bad initialization
4. Dropout (Regularization)

Models (More details on CNN)

We explored the followings:

1. (fully connected layer) x 1 as the baseline model
2. (2D convolution layer + dropout + max/average pooling) x 2 + fully connected layer
3. (2D convolution layer + dropout + batch normalization + max/average pooling) x 2 + (fully connected layer) x 2

In particular:

- The final classification performance was evaluated based on accuracy, recall, and precision on the test set.

Visualizing the ConvNet via Saliency maps

To understand the networks:

- interested in knowing which pixels in the input image matter for the classification decision
- interested in knowing if there's any bias in the decision
- **Saliency Maps**, [Simonyan et al. (2013)]
    - computes the gradients of the predicted class scores with respect to the pixels of the input images
    - Gradients: for each pixel in the input images changes by a small amount, how much will the classification changes

Implementation

- Implemented using **TensorFlow** in Google Colab and trained via **GPU**
- Advantage of GPU: significantly reduced the training time
  - 10% of the data for 10 epochs via CPU: 30 hours
  - All data for 30 epochs via GPU: 20 minutes

More training details:

- During training, we used a **fixed number of epochs** as our stopping criterion
- Randomly shuffled the data after each epoch to create some randomness.
- The optimal set of parameters for each architecture was the set that has the minimum validation loss

Experiments and Results (Exploratory Analysis)

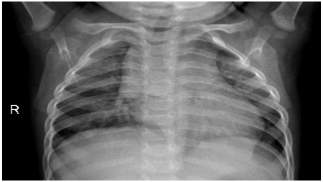

Figure: X-ray of a Normal Chest

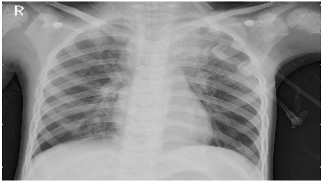

Figure: X-ray of a Pneumonia Chest

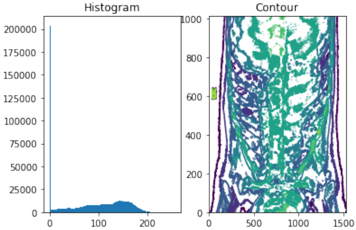

Figure: Histogram and Contour

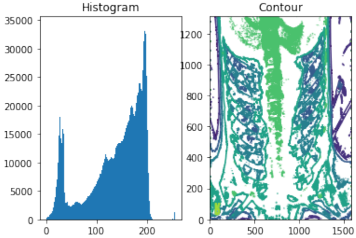

Figure: Histogram and Contour

Experiments and Results (Hyperparameters)

Table: Summary of Hyperparameters for each Neural Network

	Model 1	Model 2	Model 3
Batch size	64	64	64
# of epochs	30	30	30
Learning rate	10^{-3}	10^{-3}	10^{-3}
Filter size	NA	3×3	3×3
Stride (default)	NA	1	1
Dropout rate	NA	0.25	0.25

Experiments and Results (Final architecture)

The final architecture that was chosen

- **Model 1**:
  - FLAT $\rightarrow$ [90000]
  - FC (Class scores)

Experiments and Results (Final architecture)

- **Model 2**:
  - CONV1: 64 3×3 filters at stride 1, pad 0 $\rightarrow [298, 298, 64]$
  - CONV2: 32 3×3 filters at stride 1, pad 0 $\rightarrow [296, 296, 32]$
  - MAX POOL: 2×2 filter at stride 1 $\rightarrow [148, 148, 32]$
  - FLAT $\rightarrow [700928]$
  - FC (Class scores)

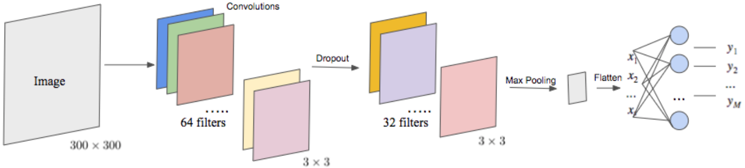

Figure: The Architecture for Model 2 (Simple CNN)

Experiments and Results (Final architecture)

- **Model 3**:
  - CONV1: 64 3×3 filters at stride 1, pad 0 $\rightarrow [298, 298, 64]$
  - NORM1 $\rightarrow [298, 298, 64]$
  - AVE POOL $\rightarrow [149, 149, 64]$
  - CONV2: 64 3×3 filters at stride 1, pad 0 $[147, 147, 64]$
  - MAX POOL: 2×2 filter at stride 1 $[73, 73, 64]$
  - FLAT $\rightarrow [341056]$
  - FC $\rightarrow [128]$
  - FC (Class scores)

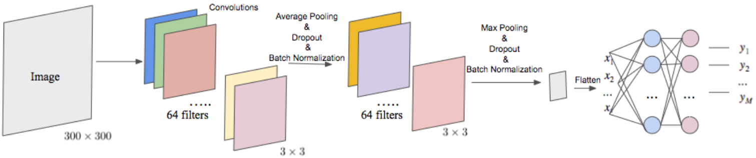

Figure: The Architecture for Model 3 (Complex CNN)

Experiments and Results (Results)

Table: Experiment Results for each Neural Network

	Model 1	Model 2	Model 3
Total # of parameters	180,002	1,420,962	43,693,634
Size	2MB	16MB	500MB
Training Time	$\approx$ 39 sec	$\approx$ 10min	$\approx$ 13min
Accuracy	71.47%	81.25%	81.09%
Precision	0.6906	0.7874	0.7688
Recall	0.9846	0.9590	0.9974

- Model 2 achieved the highest accuracy 81.25% among those 3 models
- Accuracy decreased slightly when increasing the complexity of the CNN models (from Model 2 to Model 3)
- recall (of Pneumonia) > 0.95 while precision < 0.8
  (We care more on recall since we care among all the patients with Pneumonia, how many we detectd v.s predicted some healthy people as Pneumonia)

Experiments and Results (Results)

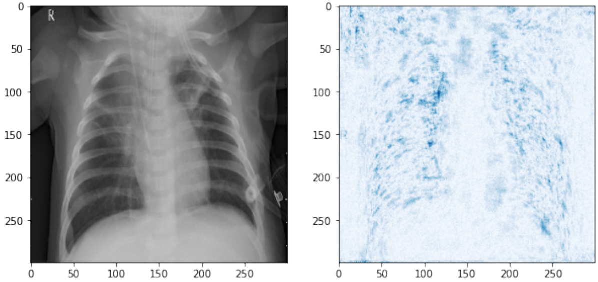

Figure: One of the Pneumonia X-ray and the Corresponding Saliency Map

- The white areas in Figure 9 represents the large values of absolute gradients
- seems like the network reacted mostly to the areas of opacity in the chest and classified the X-ray as Pneumonia (as expected)

Conclusion

- Overall, Model 2 has the highest performance among those 3 models and is sensitive in detecting patients with Pneumonia.
- With high accuracy and high recall, model 2 can be served as initial screening to speed up the diagnosis process and let doctors to further investigate the identified Pneumonia cases
- Saliency maps didn't indicate decision making bias

References

He, K., Zhang, X., Ren, S., & Sun, J. (2015).
Delving deep into rectifiers: Surpassinghuman-level performance on imagenet classification.

Ioffe, S., & Szegedy, C. (2015)
Batch normalization: Accelerating deep network training by reducing internal covariate shift

Diederik P Kingma and Jimmy Ba. (2014)
Adam: A method for stochastic optimization

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman (2013)
Deep inside convolutional networks: Visualising image classification models and saliency maps

The End