
Pneumonia Diagnosis with Convolutional Neural Networks

Group 41

Jim Jiayi Xu, Dylan Vizcarra, Apoorva Srivastava, Denana Vehab

Department of Electrical and Computer Engineering

University of California San Diego

La Jolla, 92093

{jjx002, dvizcarr, alsrivas, dvehab}@eng.ucsd.edu

Abstract

We use convolutional neural networks to classify chest x ray images as those with and without pneumonia. We present preliminary results using a custom architecture that is heavily influenced by VGG16 and demonstrate its effectiveness on a validation set. Then, achieve better performance by implementing a new architecture using a VGG16 convolutional base (trained on imagenet) and a custom fully connected classifier. Finally, we run our final architecture on a test set and document our results in this paper.

1 Introduction

We have created machine learning systems that diagnose pneumonia using chest x-rays. The system reads patient chest x-rays in jpg format as inputs. Then, a convolutional neural network classifies whether or not the patient has pneumonia. The output of our system is a binary label indicating the predicted classification. We would showcase the overall success of our system to show that healthcare professionals can incorporate machine learning in patient diagnoses to decrease workload and increase the rate of accurate diagnoses.

2 Related Work

We perform pneumonia diagnosis by classifying x-ray images. Classifying images using convolutional neural networks is a very common endeavor. In fact, this is a Kaggle challenge from a year ago [3], so there are many reference works to compare with our own. In [1] Wang's work provides descriptive information for a similar dataset of x-ray images with multiple labels. This paper demonstrates the effectiveness through weakly-supervised multi-label image classification and reports the results. Another influential work is [2] by Pranav Rajpurkar in which they develop a 121-layer CNN called CheXNet. Although, their goal is to output a probability pathology, it's helpful in our development to gain a high-level understanding of the different approaches to solving this issue. By comparing to these related works, we were able to anticipate what issues may arise and how they can be overcome.

3 Datasets and Features

We use the Chest X-Ray Images (Pneumonia) dataset from Kaggle [3]. The training dataset contains 5718 images split between 1842 non-pneumonia images and 3876 pneumonia images. The validation dataset contains 8 images of each class for a total of 16 images. The testing dataset contains 234 non-pneumonia images and 390 pneumonia images.

3.1 Preprocessing

We decided that the validation data too small because it contained only 16 images. We experienced a lot of difficulty tuning hyperparameters on this data because it is statistically insufficient. We expanded it to 336 images split between 136 non-pneumonia images and 200 pneumonia images. Because the training data is heavily biased toward pneumonia images, we attempted to encode this prior distribution into our validation set (and assume that our unseen testing set will have a similar distribution). Therefore, we tried our best to emulate this prior and had more pneumonia images than non-pneumonia images in the validation set. Next we enhanced the contrast of all the images to highlight the edges and other low level features of the image in hopes that they will be more visible to the convolutional neural network. In this process, we convert each image from the RGB color space to HSV and perform histogram equalization on the V channel, and then we convert back to RGB.

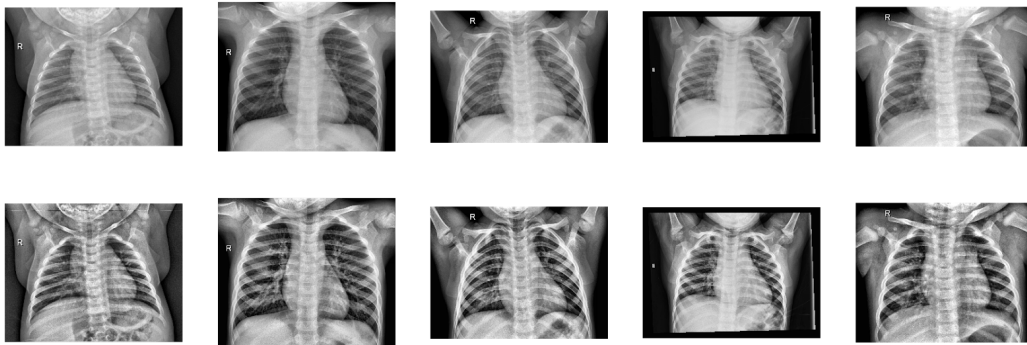


Figure 1: Non-Pneumonia Images Before (top) and After (bottom) Preprocessing

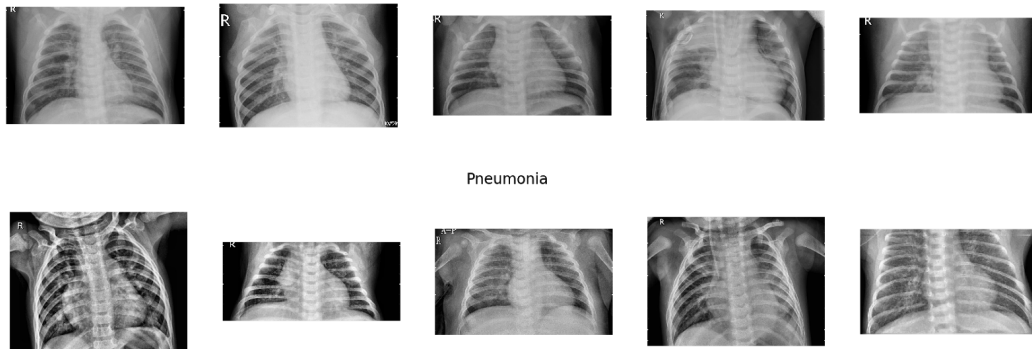


Figure 2: Pneumonia Images Before (top) and After (bottom) Preprocessing

Finally, we apply a stochastic data augmentation process during runtime. This process includes random horizontal flips, brightness scalings between 0.8 and 1.2, and rotations within 10 degrees. Our data generator randomly performs these augmentations as it assembles images and packs them into mini-batches for the convolutional neural network.

4 Methods

We use convolutional neural networks as our primary machine learning model for binary classification. Convolutions can extract features from an image while preserving spatial relationships between its pixels. A convolutional neural network lends itself well to image processing applications because it performs a series of convolutions on input images.

4.1 Convolutions

In image processing, convolution is a mathematical operation that takes two inputs, an image and a convolutional kernel, and outputs a new image. For each pixel in the input image, it computes the weighted sum of pixels within a local region around it and assigns it to a matching location in the output image. These weights are specified by the convolution kernel. Conceptually, the kernel slides to every position in the image and computes a new pixel as a weighted sum of the pixels that it slides over.

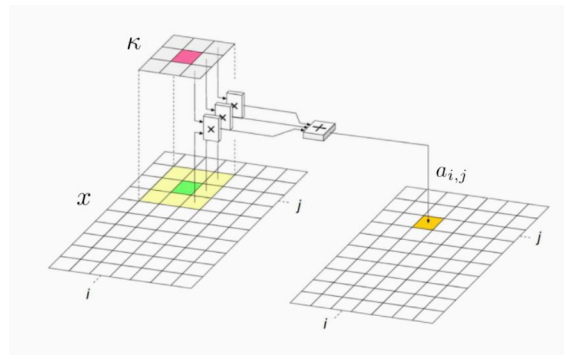


Figure 3: Image Convolution

A more computationally efficient interpretation of a convolution is a weighted sum of shifted versions of the image; the amount of horizontal and vertical shift are specified by the indices (i,j) of the convolution kernel (with $0,0$) at the center, and the weights for a particular shift are found by indexing into the convolution kernel.

The general equations for the two interpretations of a convolution for input x , kernel k , and output y are as follows:

$$(1) \quad y(i,j) = \sum_{s=-a}^a \sum_{t=-b}^b k(s,t) x(i-s, j-t), \text{ where } (i,j) \text{ denotes a specific pixel}$$

$$(2) \quad y = \sum_{s=-a}^a \sum_{t=-b}^b k(s,t) x(i-s, j-t), \text{ where } (i-s, j-t) \text{ denotes a shift of } (-s, -t) .$$

Convolutions can be used to extract features from images. For example, listed below in Figure 4 are various convolutions used to extract edges from an input image:





Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Figure 4: Examples of Convolutions as Feature Extractors

4.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a specific type of neural network architecture that incorporates convolutional layers and is influenced by the visual nervous system. This makes CNNs very suitable for image processing applications. In a convolutional layer, each neuron is connected to only input neurons within a small focal range, specified by a kernel size. Additionally, all neurons within the same layer (and depth within the layer) share the same weights. This layer implements the image convolution specified in the previous section. A CNN performs a series of successive convolutions of an image before a fully connected neural network classifies the final convolutional output. The convolutional layers behave as feature extractors for the fully connected neural network, eliminating the need for manual feature engineering. This greatly regularizes neural networks for image processing because it uses far fewer parameters than a fully connected neural network. Additional pooling layers exist to downsample the convolved outputs and increase the effective focal range of successive layers. Activation and loss functions found in traditional fully connected neural networks are also used in convolutional neural networks. We used ReLU activations for every layer except the output, which we used softmax for, and we categorical cross entropy for 2 classes[1]. These networks can still be trained with gradient descent.

$$(3) \text{ReLU}(x) = \max(x, 0)$$

$$(4) \text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

$$(5) \text{Cross entropy loss } L = - \sum_{c=1}^M y_c \log(p_c), \text{ where } y \text{ is the true label, } p \text{ is the predicted probability of the true class, and } c \text{ denotes the class}$$

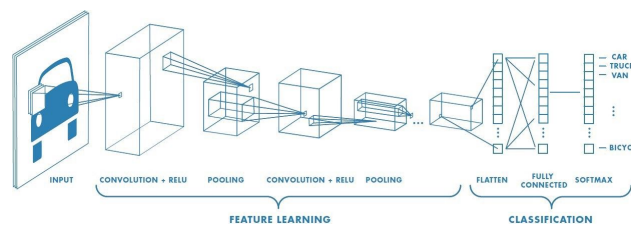


Figure 5: Example Convolutional Neural Network Architecture

5 Experiments

5.1 Training a CNN from scratch

Instead of using a pre-trained model, we wanted to see how much performance we could achieve by tuning hyperparameters on our own CNN. We initially developed a custom CNN using the popular VGG16 as an inspiration. We started with an exact duplicate of VGG16, and changed the number of output nodes from 10 to 2 to fit it with our application (corresponding to the two classes: pneumonia, not pneumonia). As expected with our small dataset, coupled with the enormous VGG16 architecture, we received extremely overfit results. We tried our best to regularize our model by reducing the number of parameters, adding batch-normalization layers in the convolutional base, and increasing dropout in the fully connected classifier. At this point, we decided to add the data augmentation described in section 3. Finally, in Keras, we implemented early stopping, such that the system will track validation performance and terminate training when validation error starts increasing too much. After we finished tuning hyper parameters, we achieved the results described below in Figure 6:

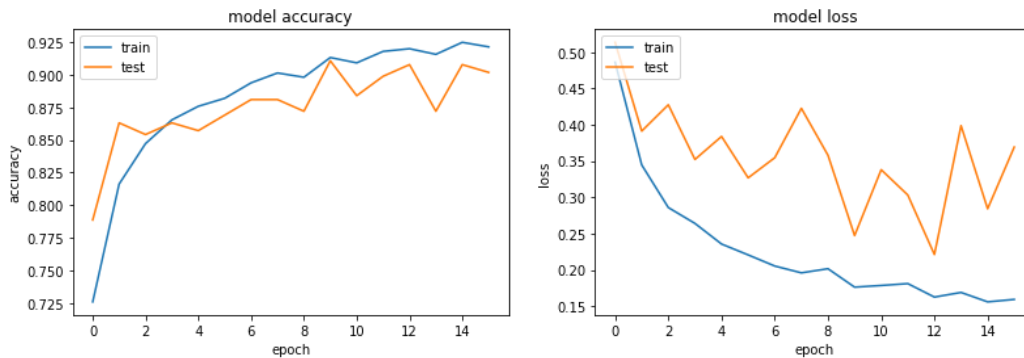


Figure 6: Performance of our CNN vs Epoch Time

We tracked accuracy and loss for both the training and validation sets. Although the validation performance has slight perturbation, it tracks the training performance well.

5.2 Training using VGG16

We wanted to compare our custom VGG16-inspired CNN with a famous successful CNN model, so we decided to use the real VGG16. We loaded the convolutional base of VGG16 trained on Imagenet to use as a feature extractor, and we added on top our fully connected classifier with 2 output nodes, corresponding to our two classes. During the training process, we froze the weights of the VGG16 convolutional base and trained only the classifier; otherwise the randomly initiated classifier weights will result in large error and loss, and greatly change the pretrained weights. We also implemented early stopping in Keras. With this new model, we achieved the results described below in Figure 7:

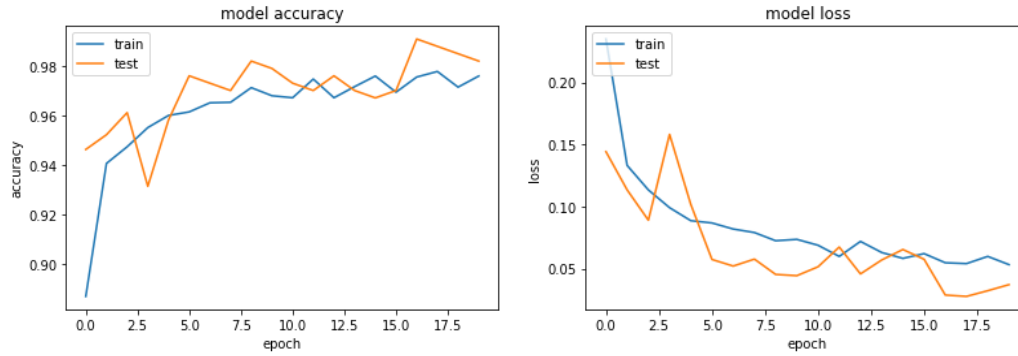


Figure 7: Performance of our Pretrained VGG16 vs Epoch Time

As expected this model performs much better than our custom model despite having more parameters because VGG16 was trained on a much larger dataset. The initial convolutional layers capture universally useful features such as edge and simple shape detectors. The final convolutional layers capture higher level features and can be adjusted to better much specific applications. We considered fine tuning this model by unfreezing the last group of convolutional layers in to jointly train with the classifier. However, since we achieved 99% validation accuracy [1] with this model, we judged this to be unnecessary.

5.3 Training using VGG19

We then wanted to use another pre-trained convolutional neural network. We decided to use VGG19. Similarly to the previous section, we loaded the convolutional base of this neural network trained on Imagenet, and added a fully connected classifier. We also implemented early stopping in Keras. With this new model, we achieved the results described in Figure 8:

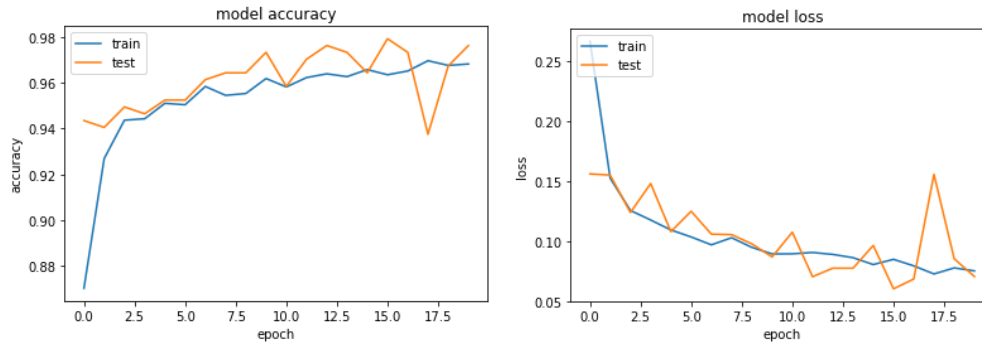


Figure 8: Performance of our Pretrained VGG19 vs Epoch Time

5.4 Test Results

After achieving satisfactory results on the validation set using the pretrained VGG feature extractors, we were confident enough to finally test performance on the testing set. Because the VGG16 model achieved the highest validation performance, we designated that as our main model. On the previously unexplored testing set, we achieved the following performance described in Figure 9:

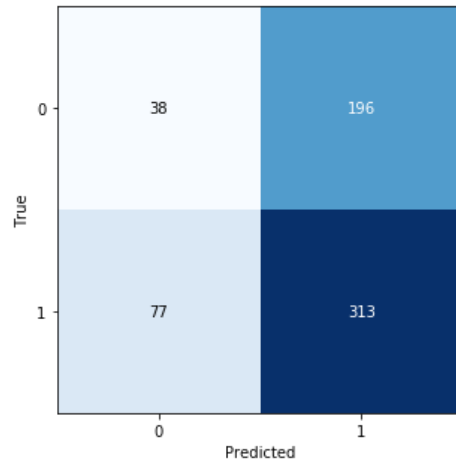


Figure 9: Confusion Matrix of Test Set Classifications

Accuracy: 0.806

Precision: 0.615

Recall: 0.803

5.5 Discussion

The final test dataset accuracy is noticeably lower than the validation accuracy. We tuned our model and hyperparameters on the validation dataset, and are surprised that performance is so different when using the test dataset. Following standard procedure when presenting machine learning performance, we did not tune our machine learning systems on the testing dataset. Although our model did not perform as well on the testing dataset as we have hoped, we still present this result as an honest representation of our machine learning model.

6 Conclusion and Future Work

Overall, our machine learning pneumonia classifier has 80% accuracy, 60% precision, and 80% recall. Although performance leaves room for improvement, we proudly present our machine learning system as a proof of concept that CNNs can indeed diagnose pneumonia with reasonable success. For future work, we recommend adding more data samples to the relatively small dataset, and to try other successful CNN architectures beside VGG16. Additionally, one can try to fine tune the pretrained VGG16 by unfreezing the last convolutional layers and jointly training with the classifier. We achieved high validation performance and believed this to be a reason not to attempt this, but again, our test performance was not perfect. Another interesting challenge is to localize the pneumonia within the image using and adjusted CNN architectures, such as mask RCNNs. This can greatly aid in successful diagnoses because health-care professionals can view directly the affected areas of the lungs instead of scanning the entire x-ray image.

7 Contributions

To tune hyperparameters more efficiently, we ran tests for each architecture in groups of 2. This way, we can run two tests at a time in parallel, effectively doubling our testing speed. Jim and Dylan tuned the custom CNN while Denana and Apoorva tuned VGG16. Jim's and Denana's custom and VGG16 models performed better and are featured in this report. Since Dylan and Apoorva achieved worse results, they cooperatively tuned a VGG19 based network. Ultimately, the VGG16 network performed best on the validation set, so we designated that as our overall testing model. For reference, we include a high level diagram of a VGG type network.

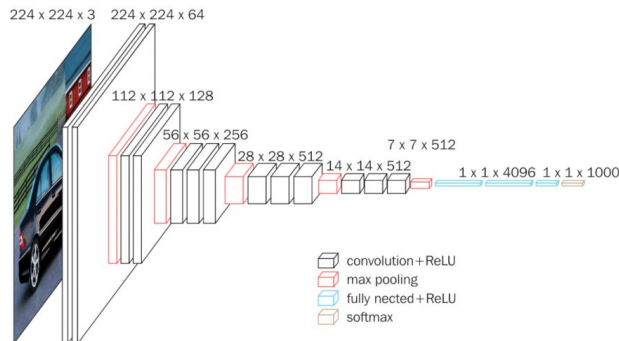


Figure 10: VGG Convolutional Neural Network

Jim Xu: performed tests on the custom CNN architecture, implemented histogram equalization, created Keras starter code for the team

Dylan Vizcarra: performed tests on custom CNN and VGG19 classifiers to tune hyperparameters

Denana: performed tests on the VGG16 based classifier to tune hyperparameters.

Apoorva: performed tests on the VGG16 and VGG19 based classifiers to tune hyperparameters

References

- [1] Wang, Peng, Le Lu, Lu, Bagheri, Ronald M. Summers (2017) Department of Radiology and Imaging Sciences, Clinical Center, National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD 20892
- [2] Pranav Rajpurkar (2017) CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning
- [3] "RSNA Pneumonia Detection Challenge," Kaggle. [Online]. Available: <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data>. [Accessed: 04-Jun-2019].