Convolutional Neural Networks for COVID-19 Diagnosis

David Zhang
University of California, Berkeley

Abstract

Utilizing a pre-cleaned/extracted dataset found on Kaggle, I build a convolutional neural network (CNN) DavidNet using `TensorFlow` to classify labeled chest x-ray images as normal, pneumonia, or COVID-19. I also implement CheXNet, a DenseNet121 architecture with weights trained on a similar but much larger dataset of chest x-rays for comparison. Comparisons are made on the basis of several metrics including accuracy, recall, precision, and F1 scores; as well as on a qualitative inspection of the misclassified examples. DavidNet outperforms CheXNet on this dataset, although the data and required modifications to CheXNet likely make it an unfair basis of comparison. There is still considerable room for improvement on both models, since DavidNet is quite simple, and I only train a small proportion of the modified CheXNet's weights.

1 Data Overview and CNNs

I give a brief overview of the data and convolutional neural networks.

1.1 Data Overview

The data used to train and validate the neural networks used was found on Kaggle[1]. The dataset consists of 317 frontal chest x-rays with dimensions $(224, 224, 3)$, with 251 images for training and 66 for testing. Within the training set, there are 111 COVID-19 x-rays, and 70 pneumonia and normal x-rays. In the test data, there are 26 COVID-19 x-rays, and 20 pneumonia and normal x-rays. There is some class imbalance. The pixel values are normalized, i.e. all the pixel values are divided by 255.

In addition to the normal training images, I perform image augmentation across the training images, effectively doubling the training data. The augmentation consists of some zooming, horizontal flipping, rotation, and shearing that is randomly applied.

1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of neural networks designed for tasks related to computer vision, such as image recognition and object detection. Their development is most commonly attributed to Yann LeCun [2]. CNNs essentially learn hierarchical representations of visual data, such as the patterns of a tiger or the eyes of a human, by automatically extracting features at different levels of abstraction. The key components of a CNN are as follows:

1. **Convolutional Layers**: These layers apply convolution operations to input data. Convolution involves sliding a matrix (filter) over the input to perform element-wise multiplication and summation, producing feature maps that help capture the hierarchical representations.

[1] COVID-19 x-rays are sourced from the University of Montreal here. Normal x-rays were sourced from here. More details can be found on this discussion thread.

[2] LeCun et al. (1998), seminal paper on CNNs.

2. **Pooling Layers**: Pooling layers reduce the spatial dimensions of the input data, reducing the computational load and preserving the most important information. An example is max pooling, where the feature map matrix is reduced to the maximum value within equally divided subsections of the matrix.
3. **Activation Functions**: Non-linear activation functions, such as ReLU (Rectified Linear Unit), introduce non-linearity to the model, allowing it to learn complex relationships in the data. These are applied after every convolution step.

Following the convolution/pooling layers, the final feature map is flattened, and the entries can be used as features in a typical fully connected neural network for classification, or any other machine learning model.

2 Architectures

2.1 DavidNet (Custom CNN)

I build DavidNet using `TensorFlow`. DavidNet consists of 3 convolutional layers and 3 pooling layers[3]. The first two convolutional layers use a $(3, 3)$ filter size with 32 filters and stride and dilation of 1 with ReLU activation. Each of these convolutional layers are followed by a max pooling layer with a pool size of $(2, 2)$. The third convolutional layer uses a $(3, 3)$ filter size with 64 filters, with stride and dilation of 1 and ReLU activation. This is followed by another max pooling layer with pool size $(2, 2)$.

After convolution and pooling, the output is a tensor of size $(26, 26, 24)$. This tensor is flattened, and the convolution/pooling layers are followed by 2 dense layers with ReLU activation and 128 and 64 units respectively. The final layer is a dense layer with softmax activation and 3 units, which outputs class probabilities using 64 features. The predicted class is the one which corresponds to the maximum probability.

2.2 DenseNet121/CheXNet

The DenseNet121 architecture was first introduced by Huang et al. (2017). Compared to DavidNet, it is an extremely deep CNN that consists of 121 total layers.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 1: Table from Huang et al. (2017) describing DenseNet architectures.

[3]More details on architecture can be found in the attached code.

Some of the key components of DenseNet are as follows:

1. **Dense Blocks**: Layers are densely connected, enhancing feature flow.
2. **Transition Layers**: Compresses feature maps to control parameter growth. Consists of batch normalization, 1x1 convolutional layers, and average pooling.
3. **Global Average Pooling**: Replaces fully connected layers by taking the average of feature maps.

CheXNet is a specific trained DenseNet121 model introduced by Rajpurkar et al. (2017). CheXNet was initialized using weights that are pretrained from ImageNet. Then, CheXNet was trained again on the ChestX-ray 14 dataset, which consists of 112,120 chest x-rays with 14 different disease classifications. 98,637 images were used for training. CheXNet achieved state-of-the-art results on the dataset, outperforming the average radiologist on the F1 score.

The weights for CheXNet are available on GitHub, and I load them onto a DenseNet121 instance in `TensorFlow`, without the top layers. For the top layers, I add a global pooling layer and a flatten layer, followed by two dense layers of 64 and 3 units respectively. The first dense layer uses ReLU activation, while the final dense layer uses softmax and outputs class probabilities.

2.3 Training

For CheXNet, the weights are frozen for all of the layers maintained from the original model. Therefore, I only train the added top layers. This amounts to 65,795 trainable parameters, and 7,037,504 untrainable CheXNet parameters. DavidNet has 5,575,011 trainable parameters.

Each of the CNNs are trained with a batch size of 64 for a maximum of 25 epochs (50 for image augmented data), with class weights calculated using `compute_class_weight` from `sklearn` to counteract the class imbalance. The ADAM optimizer is used to minimize cross entropy loss with an initial learning rate of 0.01, which decreases by 30% if the loss does not decrease in 2 epochs, with a minimum learning rate of 10^{-6}. Early stopping is implemented in every training case: training stops if loss does not decrease within 3 epochs. When training stops, the resulting CNN will be the one with the least validation loss during training.

3 Results

I evaluate DavidNet and CheXNet with and without training on augmented images[4]. Even when setting `TensorFlow`. `NumPy`, and `Python` random seeds, there is uncontrollable randomness in the training process, so the results reported and the ones in the code many slightly differ. I report the initial training and validation results.

3.1 DavidNet results

Initial Results The first training without image augmentation stopped at epoch 17/25. The final model takes weights from epoch 10, with a training accuracy of 0.9960 and a validation accuracy of 0.9545. When evaluating the model on the entire test set, it achieves an overall accuracy of 0.95. Some interesting things to note: DavidNet perfectly classifies COVID-19 x-rays, and has comparatively poor recall (true positive rate) on pneumonia. Overall, the F1 score is quite high across classes (> 0.9), where the F1 score is defined as

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}.$$

The F1 score can be seen as a balance between precision and recall, and it being quite high across classes indicates decent model performance. On inspection of the misclassified images, many of them are misclassified as normal, although they look quite normal. One x-ray is misclassified normal when it does not look normal, and one x-ray that appears normal is misclassified as pneumonia.

[4]Detailed plots and tables, such as training/validation metrics, confusion matrices, and reports of classification metrics can be found in the attached code in order to maintain the page limit.

Results after Image Augmentation Training Training with image augmentation immediately follows the initial training. The training ends after 6/50 epochs, where weights are taken from epoch 3 with a training accuracy of 0.7450 on the augmented images and a validation accuracy of 0.9242, a notable decrease from the initial results. When evaluating DavidNet on the entire test set after training, the overall accuracy decreases to 0.92, suggesting that DavidNet overfits during the initial training. The recall for pneumonia significantly improves from 0.89 to 1, but loses some precision. F1 decreases slightly across all classes. DavidNet still performs quite well for COVID-19, with precision 1 and recall of 0.96. When inspecting the misclassified images, there are more misclassified images, with several misclassified as pneumonia, although they appear quite normal.

3.2 CheXNet Results

Initial Results The first training of CheXNet stops at 25 epochs, ending with a training accuracy of 0.8566 and a validation accuracy of 0.7727. Weights are taken from the last epoch. When evaluating the model on the entire test set, the model achieves an overall accuracy of 0.78. Similarly to DavidNet, CheXNet performs best on COVID-19 x-rays with precision 0.95, recall 0.84, and F1 of 0.89. However, it performs quite poorly and similarly on normal and pneumonia x-rays, with an F1 of 0.69 and 0.74 respectively. When manually inspecting the misclassified images, CheXNet misclassifies two x-rays as normal which clearly look abnormal. Although there are some x-rays misclassified as normal when they appear normal, CheXNet also misclassifies many x-rays that appear normal as pneumonia. As alluded to earlier, CheXNet makes many more misclassifications in total, compared to DavidNet.

Results after Image Augmentation Training When training with augmented images, the training stops on the 22/50 epoch. Weights are taken from epoch 19 with a training accuracy of 0.8287 on the augmented data and validation accuracy of 0.7727. Compared to the initial results, the validation accuracy does not change. This could happen for a few reasons, but the classification metrics indicate that CheXNet learns nothing new from the augmented data. The overall accuracy remains 0.78, and the F1 remains relatively the same with F1 scores of 0.74, 0.67, and 0.90 for normal, pneumonia, and COVID respectively. Interestingly, overall performance for normal x-rays increases and decreases slightly for pneumonia x-rays, indicated by the F1 scores. On the bright side, this suggests that CheXNet did not initially overfit.

Although there could be a variety of reasons for this performance, this is likely because we are only retraining a small proportion of the weights with image augmentation. Since we freeze the weights from other layers, it may be that the added layers do not change enough to significantly impact the final predictions. Inspecting the misclassifications shows some differences, but the overall pattern is the same.

CheXNet Caveats While CheXNet performs worse than DavidNet, there are a few caveats. First, performance could likely be improved by training all of the weights instead of freezing them. Second, the plots of the training/validation loss still continue a downward trend even at the end of 25 epochs, indicating that CheXNet can be further improved with more epochs. However, due to time/computational restraints, I choose not to train it further. Additionally, it is highly likely that the architecture of the added layers can be further optimized, since its structure is designed primarily for simplicity for this project.

4 Conclusion

For the data used in this paper, DavidNet performs relatively well in differentiating between normal, pneumonia, and COVID-19 chest x-rays, even with a relatively simple architecture. While the modified CheXNet does not perform as well, there are many caveats to its lackluster performance with this data that do not suggest anything bad. Overall, this project shows that there is enormous potential and much more room for improvement for CNNs to do important and useful work within medicine, especially with regards to diagnosing and identifying COVID-19.

References

[1] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). *Densely Connected Convolutional Networks.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4700-4708).

[2] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Ng, A. (2017). *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning.* arXiv preprint arXiv:1711.05225.