Kaggle Project Writeup Shoya Yoshida (sy2905)

Method Explanation

For this project, I knew that Convolutional Neural Networks (CNNs) were the primary choice since they are known to excel in image classification tasks. So, as a proof of concept, I first set up a baseline model using Tensorflow and Keras to see how it would do. This was a very simple CNN with a couple layers of convolution with some batch normalization, followed by dense layers and drop out. I split the data such that 90% was for training, and the remaining 10% was for validation. I used the powerful ImageDataGenerator class from Keras to do all the preprocessing and image augmentation; I resized all images to 224 x 224 x 3, and the following are the augmentations I applied:

- Random zoom-ins and zoom-outs of 20%
- Random horizontal flips
- Random width shifts of the image up to 20%
- Random height shifts of the image up to 20%
- Random brightness changes of 30% (brighter and dimmer)
- Random rotations of up to 25 degrees.

These image augmentations were critical since some images were much brighter than the others, some chest images were a bit tilted, some were not the whole chest picture and were zoomed in, and pneumonia can affect either side of the chest, and etc. They overall let the model become much more robust and able to generalize better.

I also realized that we had much less training data for COVID; it had about five times less data than the other classes. In order to adjust for this imbalance, I used the *class_weight* parameter in the *model.fit()* function; I set it up so that the model weighs the COVID training images five times more than the other classes. This insured that the model learns the appropriate weights to classify COVID as well as the other models despite its small training size.

Despite all of this setup, I quickly found out that the performance on the CNN was not enough; it was only reaching about 75% validation accuracy. I tried bigger networks but some of them actually performed even worse. On top of this, other CNNs typically overfit the training data despite all of the image augmentation. To combat this, I used very high dropouts in the dense layers, some even up to 0.7.

Even with this, the performance wasn't very good. Thus, I moved on to my next approach, which was applying transfer learning from other very large pre-trained networks such as VGG16, which are also available through the Keras interface. Upon searching for pre-trained networks to finetune, I found ChexNet,[1] a 121-layer CNN that had been already trained on more than 100,000 chest X-ray images from NIH. I thought this was the perfect model to finetune for this task because it doesn't make too much sense to finetune other famous networks like ImageNet that have been trained on more mundane images like birds and cars. (Also, just for

[1] CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning https://arxiv.org/pdf/1711.05225.pdf

clarification, pneumonia was one of the 14 classes that this model was initially trained to classify; however, it didn't distinguish between viral and bacterial pneumonia, so I hope it is not against the competition rules to finetune this model.)

I then faced my biggest problem: no matter which model (CheXNet, VGG16, DenseNet, etc.) I tried to fine-tune, they didn't converge; the training loss didn't decrease, and accuracy stayed around 7% regardless of number of epochs. After being stuck on this problem for a day, I found out that it was because my learning rate for the Adam optimizer was too high. The default rate was 1e-3, and it needed to be at most 1e-4 in order for the finetuning to work. With this fix, the finetuning method started to work. I reached around 83% validation accuracy for CheXNet.

However, even with this transfer learning approach, I felt that the accuracy was still rather low. I had essentially concluded that around 83% accuracy was the best that a single model could do; my next effort was to ensemble various models. I could get the prediction probability for each testing sample for each different model I train, then average the prediction probabilities to get the "group-voted" answer.

So, I trained 10 different models. I used the ModelCheckpoint callback in Keras to save the models with the least validation loss during training so that I could pick out the best generalizing version for each model afterwards. Below are the models I trained and its validation accuracy. I tried networks like DenseNet after learning that they perform well from papers such as *A Novel Transfer Learning Based Approach for Pneumonia Detection in Chest X-ray Images*.[2]

Model Name	Best Epoch	Validation Accuracy (unweighted)	Extra Note
CheXNet Finetune v1	276	81%	1 extra Dense layer at the end
CheXNet Finetune v2	320	83%	2 extra Dense layers at the end
CheXNet Finetune v3	126	83%	No extra Dense layers at the end
Inception Finetune	245	75%	
VGG16 Finetune	84	84%	
DenseNet121 Finetune	89	89%	
DenseNet169 Finetune	54	83%	
DenseNet201 201	74	90%	
ResNet50	96	85%	
ResNet101	73	83%	

Surprisingly, in the end, simply finetuning the well-known large architectures like DeseNet actually outperformed the seemingly perfect CheXNet.

By ensembling all of these models together, I was able to reach 88.235% weighted accuracy on the public leaderboard.

[2] https://www.mdpi.com/2076-3417/10/2/559

Error Analysis

Below is the confusion matrix, recall, precision, and f1-score for the validation set based on the ensembled approach from above. There are 113 samples in this validation set.

Normalized Confusion Matrix:

True Label	bacterial	covid	normal	viral
bacterial	0.742857	0.0	0.0	0.257143
covid	0.000000	1.0	0.0	0.000000
normal	0.000000	0.0	1.0	0.000000
viral	0.257143	0.0	0.0	0.742857

Unnormalized Confusion Matrix:

True Label	bacterial	covid	normal	viral
bacterial	26	0	0	9
covid	0	8	0	0
normal	0	0	35	0
viral	9	0	0	26

Recall	Precision	F1-Score
0.87142	0.87142	0.87142

Interestingly enough, recall, precision, and the f1-score were all the same. Upon analyzing the confusion matrix, I observe that my model actually predicted all cases for COVID and normal correctly, while it switched up bacterial and viral 25% of the time.

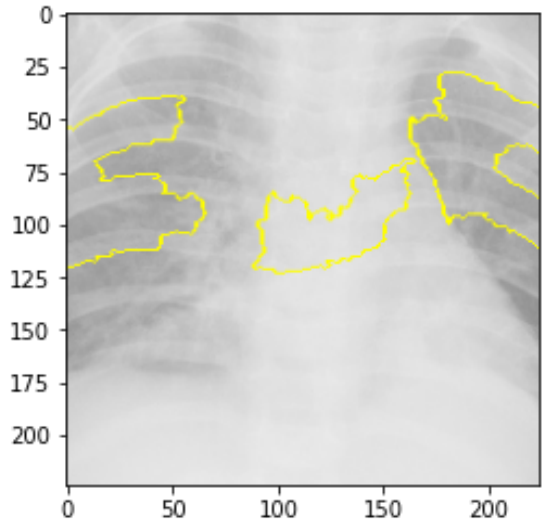
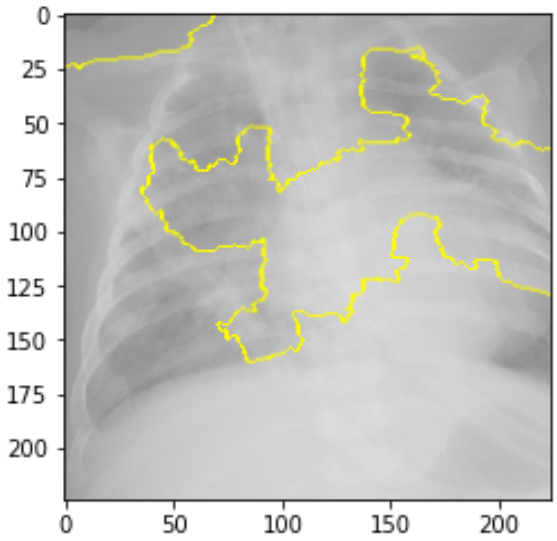
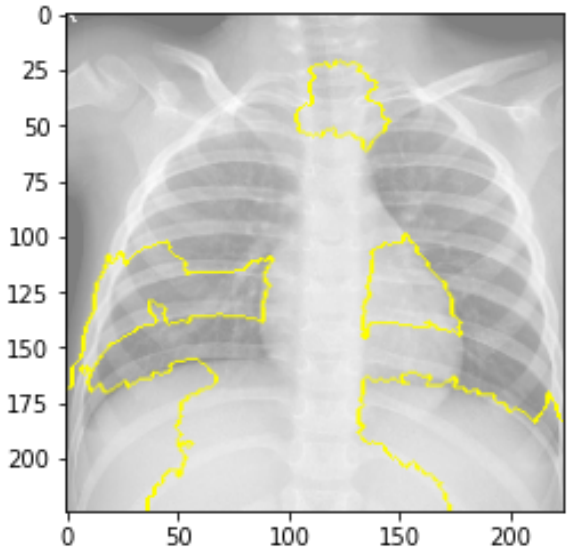
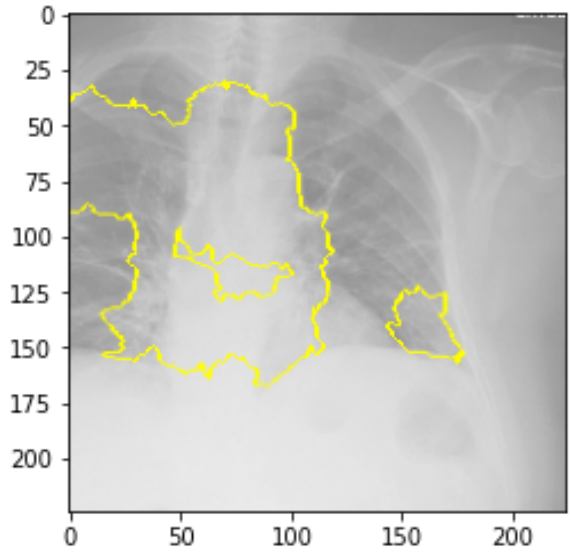
From a practical standpoint, it is obviously great that my model can detect COVID and normal almost always correctly, at least according to this confusion matrix. It should be noted that there are only eight samples of COVID in the validation, but it nevertheless tells us that this model must do overall very well for it regardless. However, it is concerning that the model does not distinguish very well between bacterial and viral pneumonia. Though they are both types of pneumonia, antibiotics can treat bacterial pneumonia while it can't treat viral pneumonia. Thus, a doctor that solely uses my model to diagnose may prescribe antibiotics to a patient with viral pneumonia, which would delay proper treatment and may eventually lead to some very bad

situations. Thus, this model definitely should not be used alone, especially when the patient is exhibiting symptoms of pneumonia.

Analyzing Interpretability

Deep Neural Networks are infamous for its lack of interpretability and is typically described as black box. However, recent advancements such as LIME[3] is able to give us clue into what the neural network is actually "seeing" in order to make judgements. Here, I analyze the interpretability of the finetuned DenseNet201, the model that had the best validation accuracy. Below are images from the validation set that has been annotated using LIME; the area that the neural network used the most in making the decision is written in with yellow lines.

[3] https://github.com/marcotcr/lime

Viral	Bacterial
	
Normal	COVID
	

I have no knowledge to completely judge if the area that the neural network is looking at is appropriate or not, but I can still observe that most of the important areas are indeed on the chest, so the neural network must be picking up subtle patterns displayed there. This annotation would provide a starting point for the clinician to carefully analyze in the X-ray image, so it should be useful. They should, of course, be careful though especially if the patient may have pneumonia for aforementioned reasons.

Improvements

First, some other minor bugs/issues other than the ones I already mentioned were dealing with some incompatibility issues between Tensorflow version 1 and 2 and running out of GPU's memory because some tensors were way too big to fit in the GPU on Google Colab. I solved the latter by decreasing the size of neural network and also decreasing the image size length from 256, which is what I initially tried, to 224. Perhaps running my own GPU instance would have been a good exercise.

If I were to do this competition again, I would simply take all of the pre-trained models obtainable from Keras for image classification, finetune all of them while making sure that the model doesn't overfit with a high dropout rate, then ensemble them together. I would love to see if I could hit 90% weighted accuracy on the test set. I would also like to systematically try which combinations of image augmentations would lead to the best results (I tried many, but it was done manually).