COMS 4771 Machine Learning HW 5

Name: Ruizhe Li UNI: rl3070

May 7, 2020

1 Model performance and comparison

As we are dealing with images, which are high dimensional inputs, dense neural network is not suitable as they will generate too many weights and does not work well on extracting graphical features. And the statistical machine learning tools such as SVM / logistic regressions would be very hard to capture the positional dependencies and sensitive to displacement. Thus, I started with CNN, which is famous for its feature extraction ability and efficiency in image classification area.

- Vanila CNN:

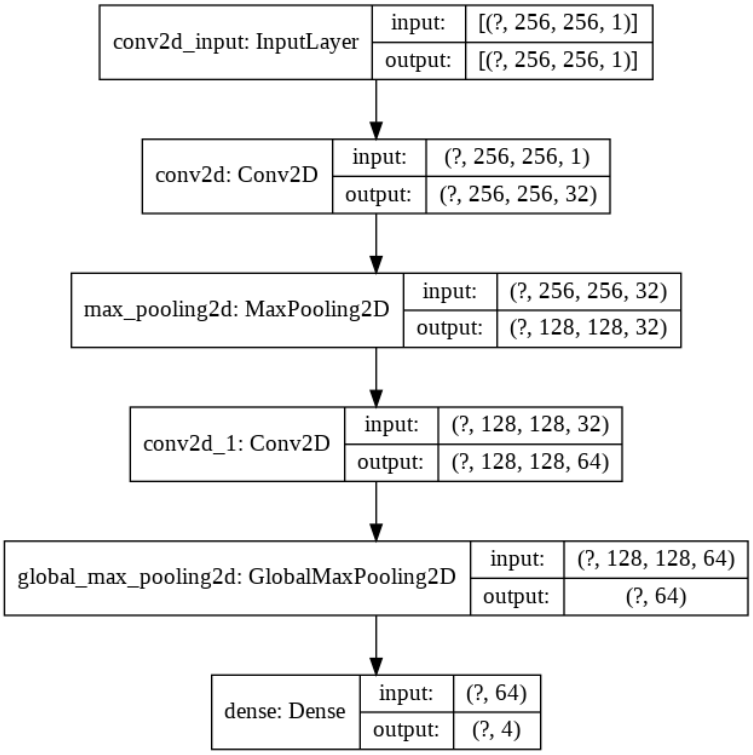

Figure 1: vanila model structure

First, we created a vanila CNN model, with two convolutional layers and one dense layer on the top. The top dense layer has 4 neurons with softmax activation. The model was trained for 50 epochs. And it started overfitting to the training data as the gap between training accuracy and testing accuracy were getting larger. The model can achieve around 68% balanced accuracy.

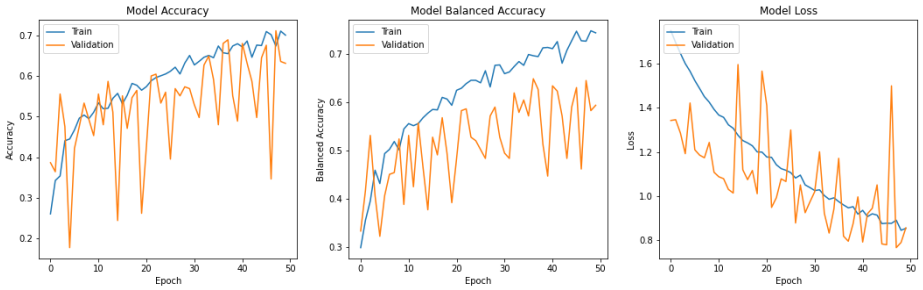

Figure 2: vanila model training history

- Vanila CNN with dropout and Batch Norm:
  Since the vanila model has overfitted the training set, I tried dropout layer and batch normalization layers. It's strange that the batch normalization layers were not helpful. The dropout layers had a little improvement and based on dropout layer my model can reach 4 layers and get a best balanced accuracy of 73%.

- Transfer learning and data augmentation:
  As our data size is small (only 1000+ images), I used transfer learning and data augmentation to supplement the dataset.
  The model is build on a VGG19 pretrained on imagenet. A pretrained model can help us better extract the graphical feature and the top dense layers are built to make classification based on the extracted graphical features.
  The model is trained in three steps:

  A. Train top dense layers only:
  Here we just use the base model as a feature extractor. And only train the dense layers on it. In this way we can achieve 70% balanced accuracy on validation dataset.

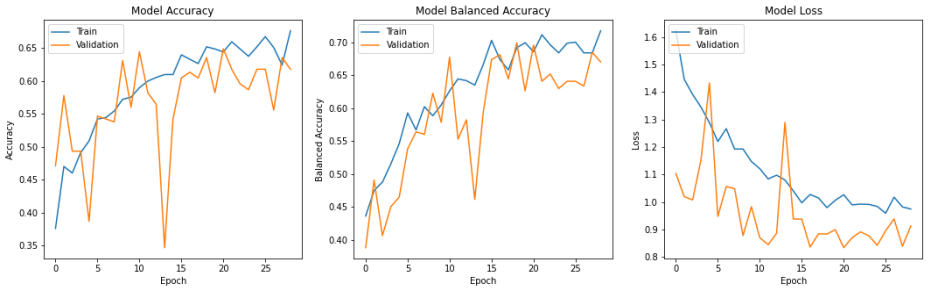

Figure 3: VGG19 train top layers only

  B. Fine tune the several top layers:
  As the model was not overfitted, I expected fine tuning would give the model more flexibility and yield a better result. Thus, the top half of the layers were unfrozen and trained with the dense layers. The learning rate scheduler was used to decrease the learning rate during training. After fine tuning, the model can achieve 80% balanced accuracy on the validation dataset.

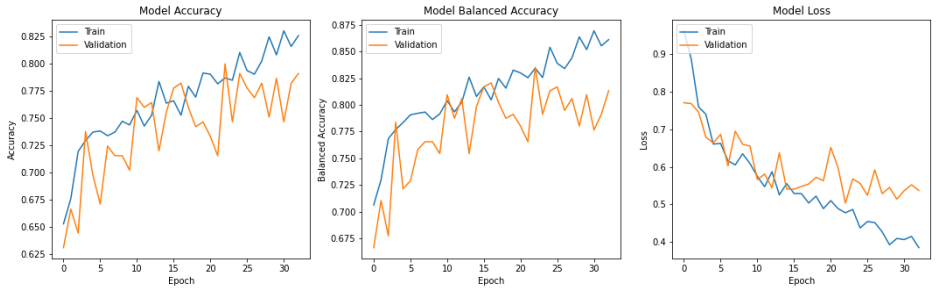

Figure 4: VGG19 fine tune half layers

C. Fine tune the whole model:
Looking at the history accuracy and loss curve of the previous session, it looks like the model has more potential. Thus, I unfroze the rest of layers, and hence all the variables in the model were trainable. However, it didn't give any improvement on accuracy.

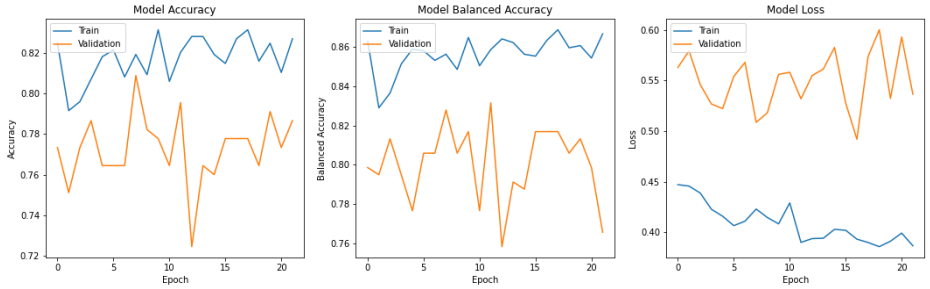

Figure 5: VGG19 fine tune whole model

As shown in the above figures, the fine tuned model gave me the best result. And I also applied Early Stopping and Model Checkpoint callbacks to help me restore the best model along the whole journey. The best model achieved 81% balanced accuracy on kaggle.

2 Data processing

- Input normalization: Each dimension in the input was normalized to [0,1] by a factor 1/255.
- Input size:
  - (None,256,256,1) was used in vanila model. It trained fast and gave the best result.
  - (None,512,512,1) was also used with vanila model. As the medical images are usually high dimensional (as I checked, the median height is around 900 pixels and median width is around 680 pixels). The model ran pretty slow with an increased input size and did not give me a better result. It was more likely to overfit. The reason could be the large resolution leads the model to focus more on local features, and the convolutional kernel cannot capture far dependencies in this setting. And the higher the resolution is, the more noise it contains. Thus, it's more likely to overfit.
  - (None,256,256,3) was used with VGG19 based model. The 3 channels are required by VGG19 model. And the ImageDataGenerator was used to convert gray-scale image to RGB channels.
- Data augmentation:

  - rescale=1./255,
  - shear range=0.2,
  - zoom range=0.2,
  - rotation range=20,
  - width shift range=0.2,
  - height shift range=0.2,
  - horizontal flip=True,

  We build the model with or without data augmentation. The model without data augmentation was likely to overfit on training data. And the data augmentation can improve results by 5% balanced accuracy on each model.

- Batch size - 32.
  The model runs with batched images to improve efficiency. And the data was shuffled before each epoch.

3 Error Analysis

Classification report

	normal	viral	bacterial	covid	macro avg	weighted avg	accuracy
precision	0.829	0.71	0.578	0.98	0.774	0.763	0.751
recall	0.935	0.605	0.686	0.833	0.765	0.751	0.751
f1-score	0.879	0.653	0.627	0.901	0.765	0.752	0.751
support	62	81	70	60	273	273	273

In the above table, we can see that the COVID precision is near 1.0, which means there is few false positives, all the detected people are real patients. However, the recall is not 1.0, which means it's possible that the model treats some patients as normal people.
In the real clinical setting, we can say that nearly every COVID patients detected by the model is a real patient. But for those who were considered as normal would still have a chance of being infected but not detected by model.

Confusion matrix

	normal	viral	bacterial	covid
normal	58	2	2	0
viral	4	49	28	0
bacterial	3	18	48	1
covid	5	0	5	50

From the confusion matrix, we can see the model is pretty good at detecting COVID. However there are still 5 COVID patients being considered as normal people and 5 being classified as bacterial. And it's hard for the model to distinguish between the viral and bacterial patients.
If the model is used in the real clinical settings, we need to double check the normal cases, because they would be patients without noticing by the model. And we should pay more attention to the viral and bacterial patients as they have a higher chance to be mis-classified.

4 Interpretability and clinical insights

Figure 6: raw figure of a normal lung

Figure 7: raw figure of a COVID lung

These are the raw figures of normal and COVID lung images. Although I am not a clinician, it seems that the normal lung image is much clearer than COVID lung image. There are something blurry in the COVID patients' lung.

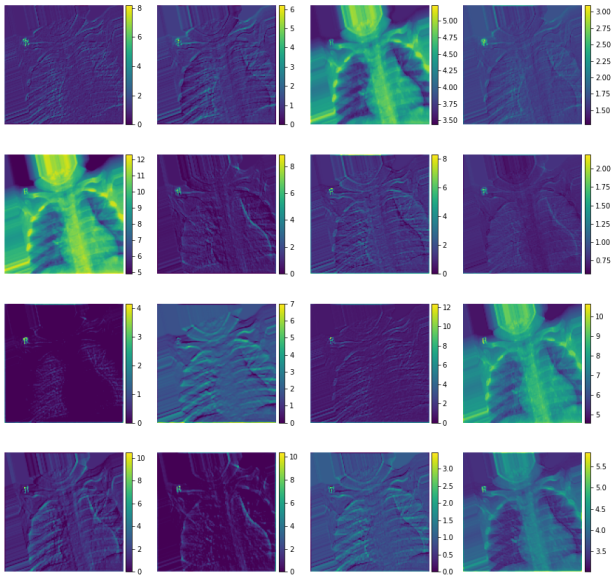

Figure 8: output of the first CNN block - normal

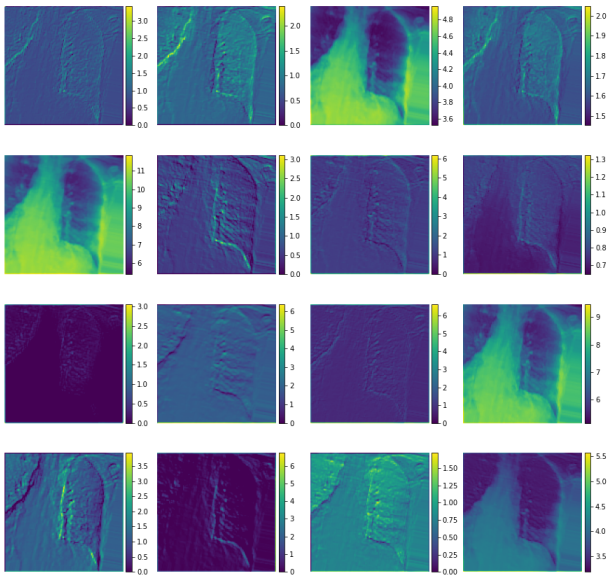

Figure 9: output of the first CNN block - COVID

Figures above are some outputs of the first CNN block in our VGG19 based transfer learning model (as there are so many channels, only 16 channels in the outputs are shown here). The first CNN block is very close to the input image. It recognizes the low level features of the image. For example, the second axis is recognizing bone texture and the third axis and fifth axis are recognizing lungs.
Compared with normal lung image outputs, the bone texture in COVID output is not clear, and the blurry lung tissues are recognized by some kernels. I think that could also be how clinicians diagnose patients from the picture.

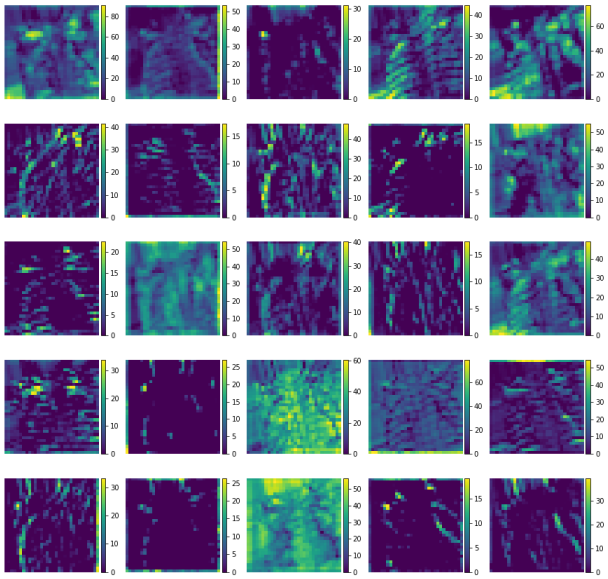

Figure 10: output of the third CNN block - normal

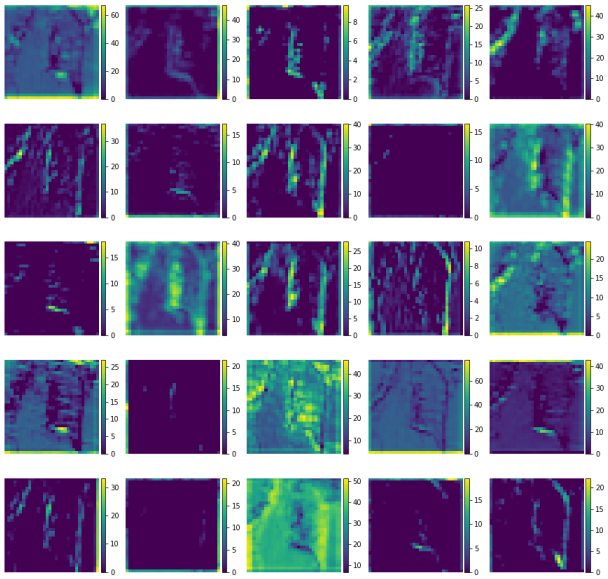

Figure 11: output of the third CNN block - COVID

The deeper the network goes, the harder to see clear pattern with human eyes. We can see the outputs of the third CNN block are still recognizable.

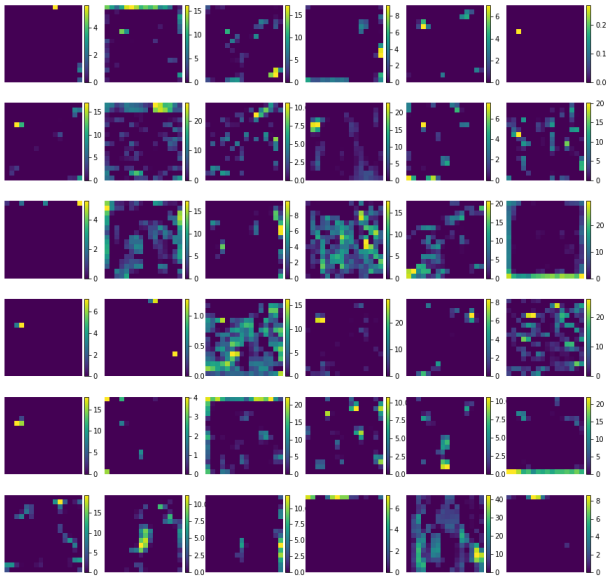

Figure 12: output of the fourth CNN block - normal

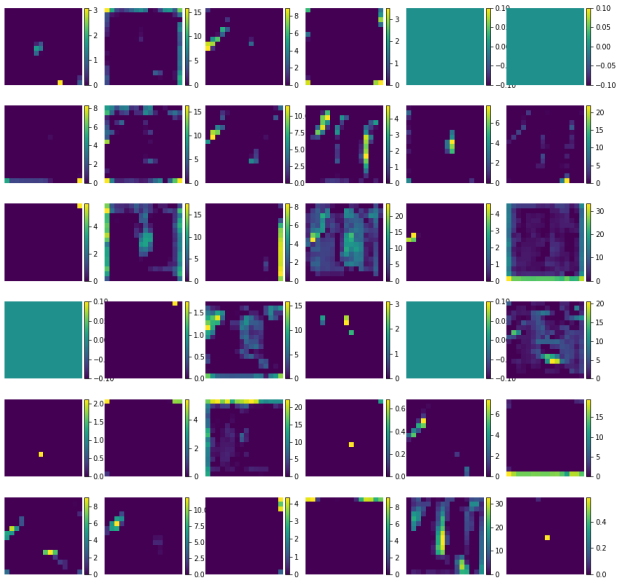

Figure 13: output of the fourth CNN block - COVID

The outputs of fourth CNN block are not recognizable at all. We can see different bright spots on the output map. Those should be the activated feature in the feature map extracted by CNN. The meaning of the spots are not interpretable by human, only the model knows the meaning of it. Notice that for images in different classes, different kernels were activated..
In conclusion, DNN is famous for its non-interpretability. However, we can use the output map of intermediate layers to find out what it is doing in each block. And the bottom layers extract the low level features like texture, the top layers mainly contains feature map embedded with encrypted information. The low level or intermediate level feature map can give clinicians insights to detect the virus.
More intermediate output figures are provided in the jupyter notebook

5 Bugs and future improvements

There are several bugs and issues slowed me down:

- Data loading and processing:
  Question: My vanila model and transfer learning model use different data loaders. As a result, I have to write different functions for interfaces but they mainly did the same job.
  Solution: Unify the data loader and their interface, modularize my code for reusability.

- Model tuning:
  Question: It is very hard to tune a good model. And sometimes I got a good model but later it cannot achieve the same performance again. And also, sometimes I changed the hyper-parameters of the model, it suddenly becomes much worse than before, but I forgot the original hyper-parameters.
  Solution: Version my code, always log the hyper-parameter configuration and settings. Set random seed to a constant. And perhaps, I should try auto tuning next time.

Conclusion: This is an interesting project. I am supper happy to be engaged in the COVID prediction. I hope to get more chance to involve myself in the research like this, where I can use what I learned in the class to contribute the the society!