Covid-19 X-ray Classifier

Task 1 description

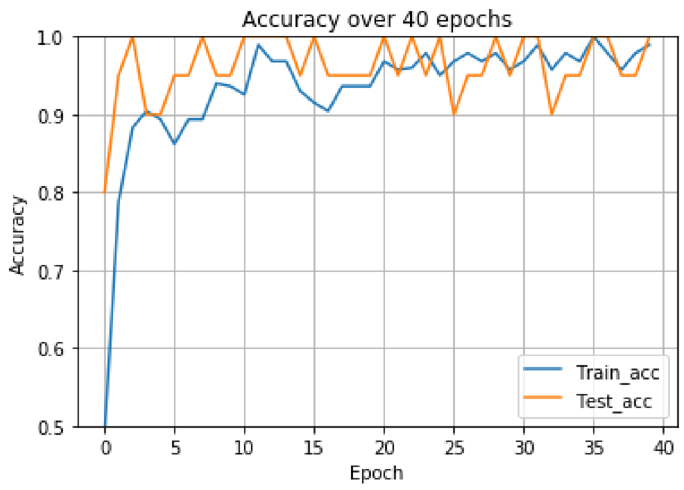
In this task, I am training a deep neural network model to classify Normal vs. COVID-19 chest X-rays. I first apply data augmentation and feature normalization on the data set and then for training, I apply transfer learning with pretrained model VGG16 with ImageNet weights. I used VGG16's convolutional base, but with my own classifier, which is shown below:

$$\mathbf{VGG16} \rightarrow \mathbf{Flatten} \rightarrow \mathbf{Dropout_1} \rightarrow \mathbf{Dense_1} \rightarrow \mathbf{Dropout_2} \rightarrow \mathbf{Dense_2}$$

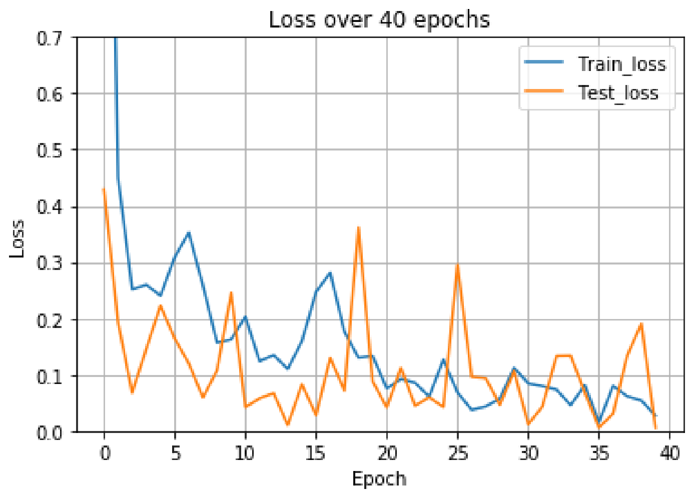
VGG16's convolutional base has 19 layers(including the input layer) with a total of 14,714,688 parameters, which will mostly be frozen during training. However, the last two layers of VGG16's convolutional base, which has 2,359,808 parameters, will be fine-tuned and therefore trainable. The convolutional base has output shape of $(7, 7, 512)$, which is then flattened by the flatten layer to an 1-D $7 \times 7 \times 512 = 25088$ output shape. A dropout layer is then applied with 0.5 dropout rate for regularization before the first dense layer. The first dense layer uses ReLu activation function, has 256 output neurons, and has a total of $\big(256 \times (25088 + 1)\big) = 6422784$ parameters. Another dropout layer is applied with 0.5 dropout rate for regularization before the last dense layer. The last dense layer uses sigmoid activation function, has 1 output neuron for binary classification, and has a total $\big(1 \times (256 + 1)\big) = 257$ parameters. The model has a total of $14{,}714{,}688 + 6422784 + 257 = 21{,}137{,}729$ parameters and $21{,}137{,}729 - (14{,}714{,}688 - 23{,}59{,}808) = 8{,}782{,}849$ trainable parameters. For the loss function, I use binary cross entropy, which works well for the binary classification my model is trying to perform. For the optimizer, I use Adam, which is an adaptive learning rate optimization algorithm, with 0.0005 learning rate. I have tried Adam, SGD, and RMSprop for my optimizer and Adam performs the best. For regularization, I use the two dropout layers(rate:0.5) and also a learning rate schedule to gradually decrease the learning rate as the training progresses. For the learning rate schedule, I use Inverse Time Decay with decay rate of 1. The training is done with batch size of 10 and epochs of 40.

Task 1 Results

Accuracy vs. Epoch plot:



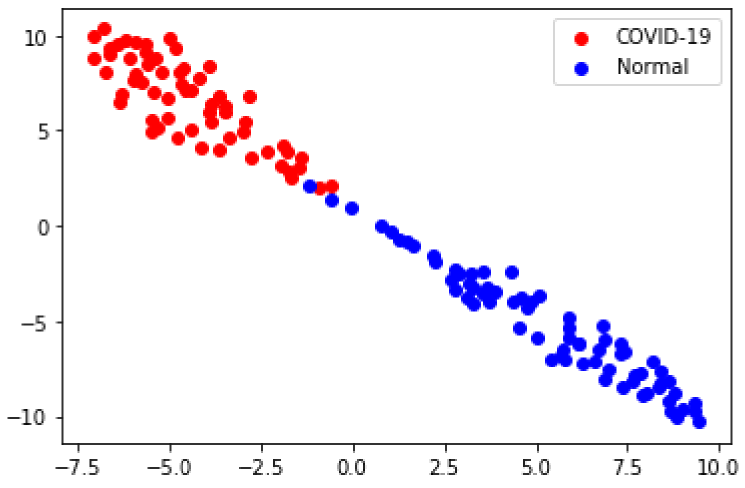
Loss vs. Epoch plot:



Overall performance:

Test loss: 0.0104
Test accuracy: 1.0000

TSNE plot:



The accuracy increases drastically in the beginning most likely due to the use of pretrained model. After epoch 5, the test and train accuracy have around the same performance and gradually improving. However, as training accuracy improves, validation accuracy tends to fluctuate and is most likely caused by a small validation set and overfitting. I applied a few regularization techniques and it significantly reduces the fluctuation. Same applies for the loss vs. epoch plot. When trying out different hyperparameters and regularization techniques, I was trying to find the sweet spot where the training and validation accuracy/loss are increasing/decrease at the same rate so they don’t have a huge gap in the plot. Overall, the model performs pretty well with 100% test accuracy and all test image have above 93% confidence correct classification. However, I think the test data set should be larger to more accurately measure the model’s performance.

The TSNE plot reduces high dimension data set from the first Dense layer into two dimension for better visualization. As we can see from the TSNE plot, the two classes is well separated, which means that the model extracts good features.

Task 2 Description

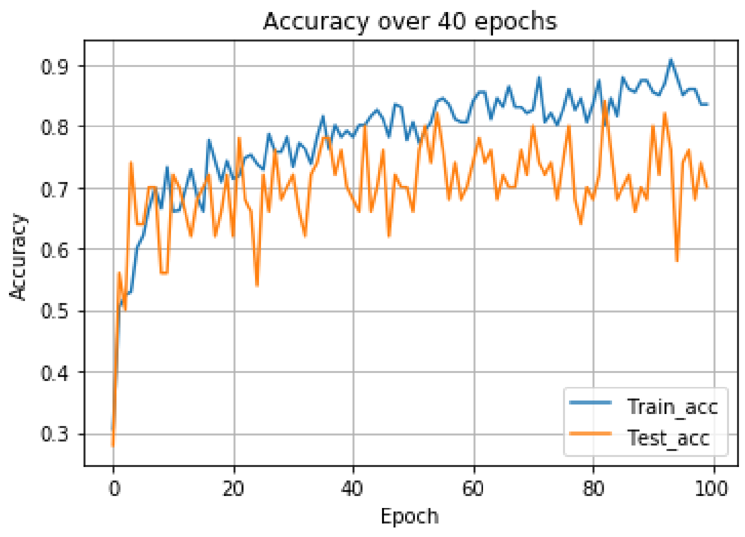
In this task, I am training a deep neural network to classify an chest X-ray image into one of the following 4 classes: Normal, COVID-19, Pneumonia-Bacteria, and Pneumonia-Viral. The model is really similar to task 1, but with different hyperparameters and classifier. I first applied data augmentation and feature normalization on the data set as usual and also used VGG16 as my pretrained model. I keep VGG16's convolutional base, but I replaced its classifier with my custom classifier shown below.

$$\mathbf{VGG16} \rightarrow \mathbf{Flatten} \rightarrow \mathbf{Dropout_1} \rightarrow \mathbf{Dense_1} \rightarrow \mathbf{Dropout_2} \rightarrow \mathbf{Dense_2}$$

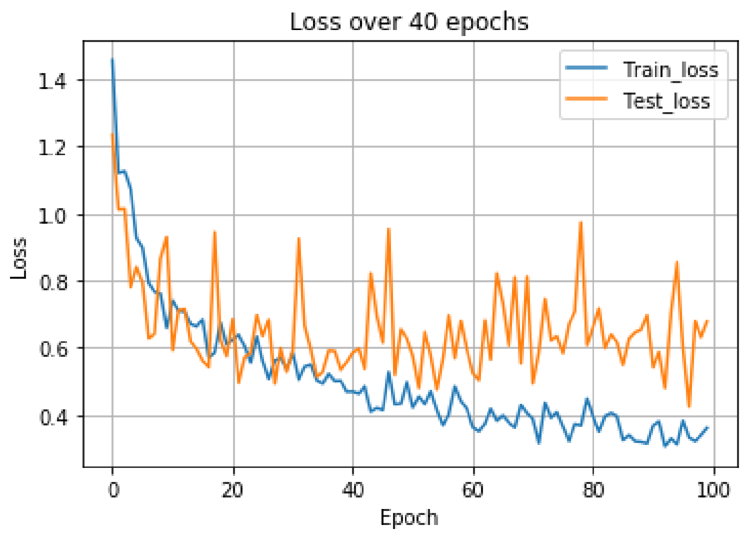
The convolutional base here is exactly the same as Task 1 since I also use VGG16's convolutional base, which has 19 layers(including the input layer) with a total of 14,714,688 parameters. In the convolutional base, I freeze the first 17 layers during training and fine-tuned the last two layers, which has 2,359,808 parameters. The convolutional base has output shape of $(7, 7, 512)$, which is then flattened by the flatten layer to an 1-D $7 \times 7 \times 512 = 25088$ output shape. A dropout layer is then applied with 0.2 dropout rate for regularization before the first dense layer. The first dense layer uses ReLu activation function, has 256 output neurons, and has a total of $\big(256 \times (25088 + 1)\big) = 6422784$ parameters. Another dropout layer is applied with 0.2 dropout rate for regularization before the last dense layer. The last dense layer uses SoftMax activation function, has 4 output neuron for 4 different output classes, and has a total of $\big(4 \times (256 + 1)\big) = 1028$ parameters. The model has a total of $14{,}714{,}688 + 6422784 + 1028 = 21{,}138{,}500$ parameters and $21{,}138{,}500 - (14{,}714{,}688 - 23{,}59{,}808) = 8{,}783{,}620$ trainable parameters. For the loss function, I use categorical cross entropy, which works well for multi-class classification. For the optimizer, I use Adam, which is an adaptive learning rate optimization algorithm, with 0.0001 learning rate. For this task, I also have tried Adam, SGD, and RMSprop for my optimizer and Adam performs the best. For regularization, I use the two dropout layers(rate:0.2). The training is done with batch size of 10 and epochs of 100.

<u>Task 2 Results</u>

Accuracy vs. Epochs plot:



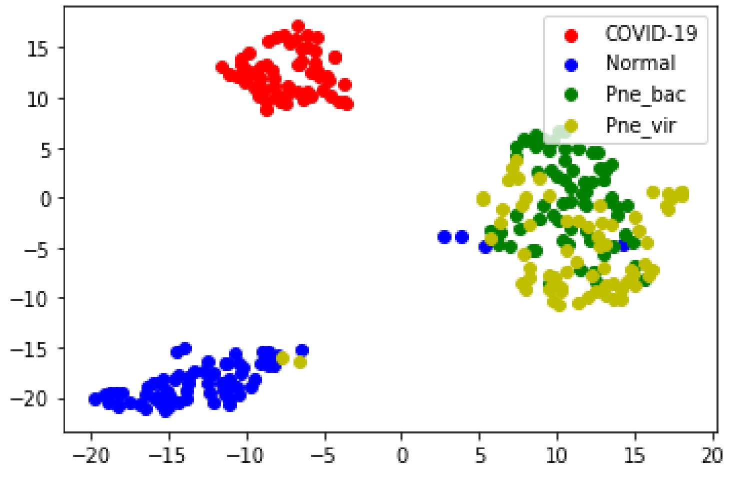
Loss vs. Epochs plot:



Overall performance:

Test loss: 0.6274
Test accuracy: 0.7778

TSNE plot:

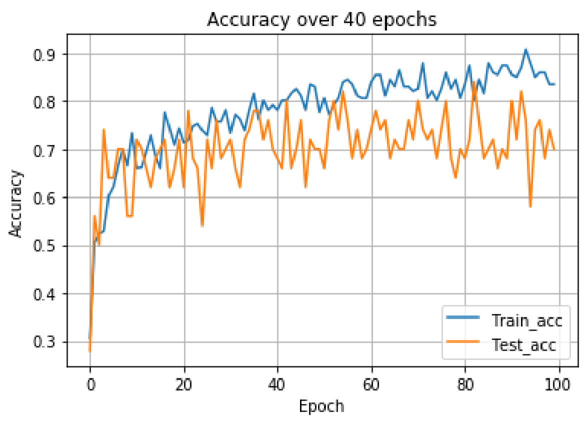


The accuracy and loss start to diverge after around 30 epochs. The training accuracy/loss increases/decreases in a steady rate. However, the validation accuracy/loss doesn't really improve much after 30 epochs and it also fluctuates a lot as we can see how spiky it is in the plots. This is most likely caused by a small validation data set and overfitting. I applied two dropout layers to regularize the model and it did help the model's performances slightly. I tried out several regularization techniques for the model such as early stopping, learning rate schedule similar to task 1, higher dropout rates, and etc., but they either doesn't help with the performance or it worsens it. My model has around 77.7% test accuracy, which is not ideal, but reasonable due to a small data set. The TSNE shows us that COVID-19 and Normal are well classified since they appear in separate compact clusters. On the other hand, Pneumonia-Bacteria and Pneumonia-Viral are well separated from COVID-19 and Normal, but they are not well separated from each other. Pneumonia-Bacteria and Pneumonia-Viral appear in the same cluster maybe because they are similar infections and we need more data to see the difference, but we can see from the plot that their clusters each has a slightly different mean of distribution.

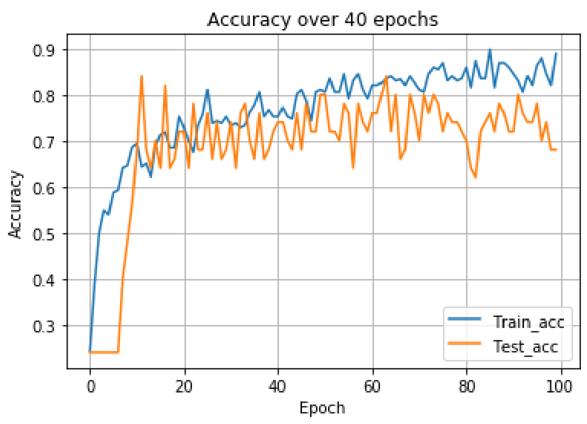
In order to improve accuracy, I first use external chest x-ray data from Kaggle and increase each class by 20 more images. However, I don't think 20 more data is enough since I didn't notice any improvement in accuracy. I couldn't increase the data set by more because it takes too long to train. I also tried out different pre-trained models such as VGG19, ResNet50, InceptionV3, and DenseNet121, but VGG16 still performs better than the rest. I will compare architectures VGG16 and ResNet50 on why and how they affect model performances.

VGG16 vs. ResNet50

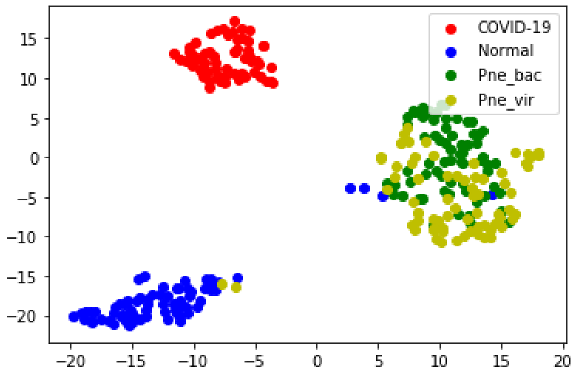
VGG16 (test accuracy: 77.78%):



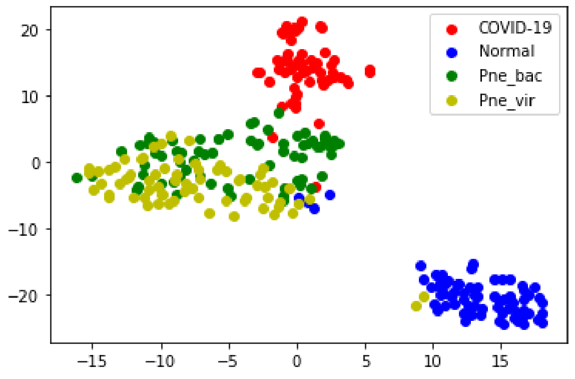
ResNet50 (test accuracy: 69.44%):



VGG16 TSNE:



ResNet50 TSNE:



VGG16 has a total of 21 layers including 13 convolutional layers, 5 max pooling layers and 3 dense layers. Since I have a really small data set, I use VGG16's convolutional base with pre-trained ImageNet weights with my custom classifier on top. During training, I freeze all the layers in VGG16 beside the last convolutional layer in order to do a little fine tuning. This model gives 77.78% accuracy. I then tried to fine-tune even more layer, but it did not improve the performances. When I tried to do more fine-tuning by unfreezing more layers, it makes the model more complex as there are more parameters to train. This leads to overfitting since I have a small data set. VGG16 is a deep network and therefore really slow to train if train from scratch and it needs a huge data set. However, it is a simply architecture and a good pre-train model to utilize for image classification.

After I tried VGG16, I wanted to improve my model performances even more, but simply using a deeper model such as VGG19 did not work. I then tried a really different architecture design, ResNet50, to see if it would help. ResNet50 is a deep residual CNN with 50 layers and its main advantage is that it lets us train even deeper networks while reducing vanishing gradient problem by using skip connections. Skip connections simply means that a layer can output to another layer by skipping some layers in between, which provides shortcuts for gradients to go through. When using ResNet50 in my model, I have to first tune the hyperparameters since VGG16 and ResNet50 are really different. I use 0.0003 learning rate and I replace ResNet50's classifier with my custom classifier, which is shown below:

$$\mathbf{GlobalAveragePooling2D} \rightarrow \mathbf{Dense_1} \rightarrow \mathbf{Dropou_1} \rightarrow \mathbf{Dense_2} \rightarrow \mathbf{Dropout_2} \rightarrow \mathbf{Dense_3}$$

The first two dense layers each has 512 output neurons with ReLu activation function, both dropout layer have dropout rate of 0.5, and the output dense layer has 4 output neurons for each class with SoftMax activation function. The optimizer and loss function are still Adam optimizer and categorical cross entropy. However, when I first train this, the accuracy stays at 26% and does not improve at all. I then realized that this low accuracy has to do with freezing the Batch Normalization layers in RestNet50 in Keras. The mean and variance in the BN layers came from the original ImageNet data set since we are using a pretrained model, however, our dataset has different mean and variance which cause this discrepancy. I then loop through the BN layers in RestNet50 to unfreeze and update the mean and variance. This allows the model to train and the accuracy improves to 69.44%, which makes more sense.

In task 2, VGG16 has a higher test accuracy than ResNet50 and we can also see from the TSNE plot that VGG16 extracts better feature than ResNet50. ResNet50's TSNE plot shows that COVID-19 isn't as well separated from the two pneumonia infections as VGG16's model did. However, I think if we have a lot more data, ResNet50 can outperform VGG16, since ResNet50 is deeper and it mitigates vanishing gradient problem that VGG has. We only have a small data set, so we have to keep our trainable model simpler in order to prevent overfitting. If we have enough data, we can fine tune more layers and increase the accuracy.

Bibliography

"Module: Tf.keras :   TensorFlow Core v2.1.0." *TensorFlow*, www.tensorflow.org/api_docs/python/tf/keras.

Vryniotis, Vasilis. "The Batch Normalization Layer of Keras Is Broken." *Machine Learning Blog & Software Development News*, 17 Apr. 2018, blog.datumbox.com/the-batch-normalization-layer-of-keras-is-broken/.

"Keras: The Python Deep Learning Library." *Home - Keras Documentation*, keras.io/.