Daniel Delijani

CS 542

Professor Bargal

Final Report

Model Writeup – Lung Image Classification

Task 1 – COVID, Normal

Part 1: Architecture Description

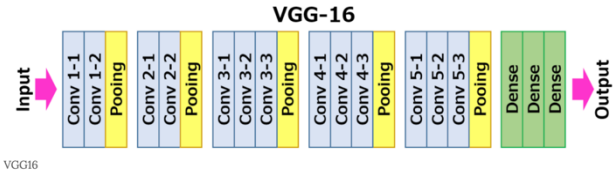

Figure 1

The model begins with using VGG16 as a base model. This model consists of 19 layers, as shown in figure 1. After careful experimentation, I found that providing the greatest validation accuracy when the last 5 layers were unfrozen, while the first 14 layers' weights remain the same as the weights the model generates after being tested on ImageNet. This ultimately results in a base model that contains 7,635,264 non-trainable parameters and 7,079,424 trainable parameters, giving the base model a total of 14,714,688 parameters. After this, I flatten the output and I decided to implement a fully connected layer with 256 neurons with relu as an activation function. I found 256 to be the perfect amount of neurons for the final hidden layer, and relu to be the activation function that led to the greatest validation accuracy. Then, to prevent overfitting, I added a dropout layer with a dropout rate of .25. This I found to be perfect in allowing the validation accuracy to increase right alongside the train accuracy. Additionally, after this layer I create the output layer as a fully connected layer with one output neuron (for binary classification), and sigmoid activation to output probabilities. This gave the model as a whole 21,137,729 total parameters, 13,502,465 of them

trainable and 7,635,264 of them non-trainable. I ended up using a very small learning rate (0.00001), as the model tended to reach a very high accuracy very quickly, so there was no need to use a high learning rate as this would allow the weights to precisely reach their respective local minima. For the loss function, I utilized binary cross entropy as it is both logically fitting and experimentally effective. For my optimizer, I used RMSProp. I noticed that it converged on 1.0 accuracy and 1.0 validation accuracy rather quickly; thus, to ensure that the model kept improving to a certain extent, I utilized RMSProp to keep an adaptive learning rate that would ensure that the gradient would not vanish, keeping each gradient descent update effective.

Part 2: Plot and comment on the accuracy and the loss

As you can tell from Figure 2, both the test accuracy and the test loss for the deep network were incredibly effective very quickly. The train and validation accuracy for the model essentially was 1.0 for almost the entire time, while the train and validation loss was around 0 the entire time as well. This, as mentioned in the previous section ultimately contributed to various decisions I made throughout the development of the model architecture.

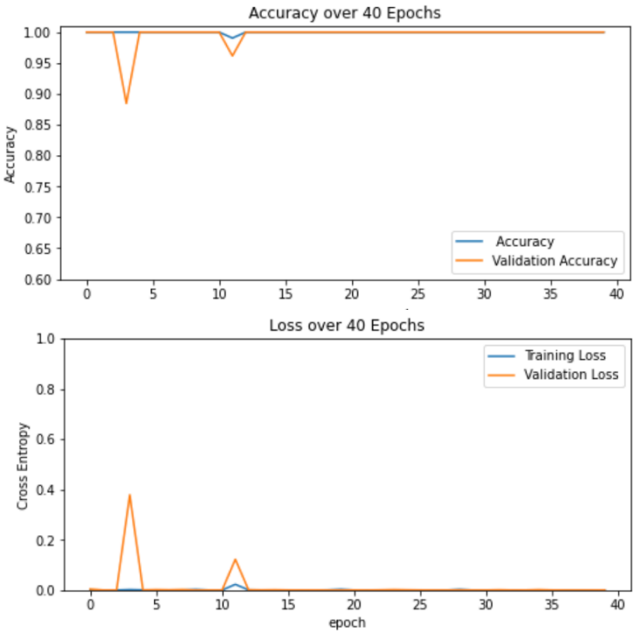

Figure 2

Part 3: Plot and comment on the t-SNE visualizations

I decided to take outputs of the flatten layer, which is the fourth to last layer and contains 25088 neurons. I felt that this layer with its many features would be able to well-distinguish the two classes, especially because there were 5 layers previous to it that have weights that have been optimized for this dataset. Ultimately, we get a graph that clearly distinguishes the two classes with the exception of one point as you can see in figure 3, so it ultimately did a fantastic job of developing meaningful and unique features for the images.

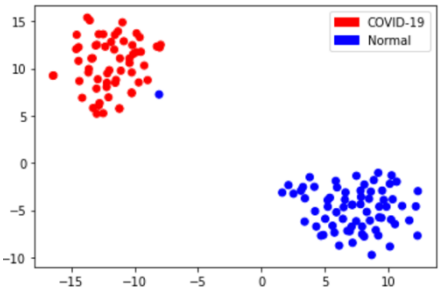

Figure 3

Part 4: Bonus

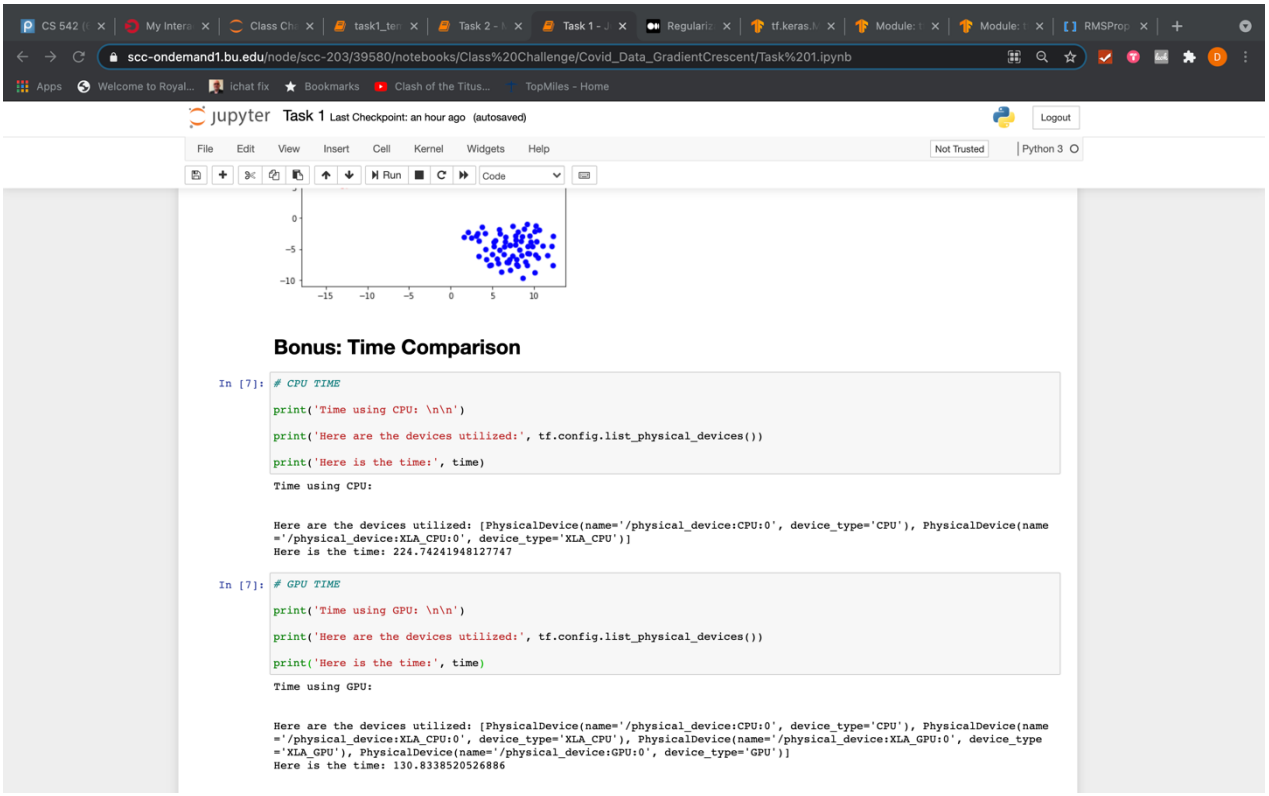

Figure 4

SCC cluster. I used the SCC cluster to launch Jupyter Notebook as you can see from the URL, and at the top of the notebook I included the line

os.environ['CUDA_VISIBLE_DEVICES'] = '-1' and fit the model alongside time.time() to receive the training time for the CPU. Then, I restarted the kernel and did the same for os.environ['CUDA_VISIBLE_DEVICES'] = '0'. Ultimately, the results are as follows:

CPU training time: 224.74241948127747

GPU training time: 130.8338520526886 seconds

A detailed description of what hardware tensorflow had access to for each respective training cycle can be seen in the output of the code in figure 4.

Task 2 – COVID, Normal, Bacterial Pneumonia, Viral Pneumonia

Part 1: Architecture Description + Comparison

For my first and primary model for task 2, the main problem I encountered throughout its development was overfitting. I had many models utilizing VGG16 that would obtain 1.0 train accuracy and test accuracy of about .65, so many of my architectural decisions are centered around this. Firstly, I use pretrained model VGG16 as a base model, with weights initialized to those when optimized on ImageNet (See Figure 1 for details on the model). This is because trying to use other base models simply did not produce good results, as I will show in my experiment with Task 2, Model 2. My first decision that helped prevent overfitting was to freeze 17 of the 19 layers of the base model. This gave the model far less parameters, which ultimately allowed it not overfit. The base model thus had 2,359,808 trainable parameters with the last two layers unfrozen, with 12,354,880 untrainable parameters throughout the first 17 layers for a total of 14,714,688 parameters. Next, I implement a flatten layer. Then, I create a dropout layer with a value of .4 to prevent overfitting. Then, I have a fully connected layer with 256 output neurons, with activation relu, and regularization function l2 norm with l = .3 to

further help prevent overfitting. Then I use yet another dropout layer with a rate of .4. Finally, for my output layer I have a fully connected layer with 4 output neurons (4 classes), with activation softmax to output probabilities, and regularization function l2 norm with l = .3 once again to help prevent overfitting. This ultimately gave the model as a whole 21,138,500 parameters, 8,783,620 of them trainable and 12,354,880 non-trainable. For the learning rate I went with a relatively small learning rate of .00001 as it had no problem reaching a local minima generally in a short amount of epochs, so this would allow it to narrow in on the perfect minima. I used RMSProp once again for this task as the variable learning rate ultimately allowed the model to fit the data very well. Next, for the loss function I utilize the categorical cross entropy loss function, which had the best results as I expected. Another decision I made was to prevent overfitting in this model was to utilize early stopping by only running 40 epochs. I found this to be the perfect amount where the data was not being overfit, and validation accuracy was highest.

For the second model, I used Resnet 50 as a base model as I read very positive affirmations of it online. I decided to freeze 171 of the 175 layers, leaving the last 4 layers open to training. This gave the base model 4096 trainable parameters and 23,583,616 non-trainable parameters for a total of 23,587,712 parameters. Next in the model is a flatten layer. Then I create a fully connected layer with 256 neurons (which in the past I have realized is very effective) with an activation of relu to help the model learn more efficiently. Then, for the output layer I implement a fully connected layer with 4 output neurons (because 4 categories) with an activation of softmax so it outputs probabilities. For the learning rate, I use a somewhat high rate at .0001 as the model

had trouble learning quickly with lower learning rates. For optimizer function, I use RMSProp again as it was consistently the most effective function. Additionally, I use categorical cross entropy as it worked the best and is a logical use case as well.

In terms of performance, model 1 performed far better than model 2. As you can see in Figure 5 and Figure 6, the accuracy for model 1 is far better than that for model 2. Ultimately, the test accuracy on the unseen data for model 1 is .75, whereas for model 2 it is .27. I attribute this difference to be the result of the base model. I think that for this classification task, VGG16 as a base model proves to be far more effective, and that though it may be possible to tweak parameters to achieve a decent model with Resnet 50, it is pointless as the same modifications applied to a model with VGG16 as its base model would prove far more effective. Additionally, to comment on model one specifically, all the measures I took to avoid overfitting proved successful as the validation accuracy is right on par with the train accuracy throughout the model fitting.

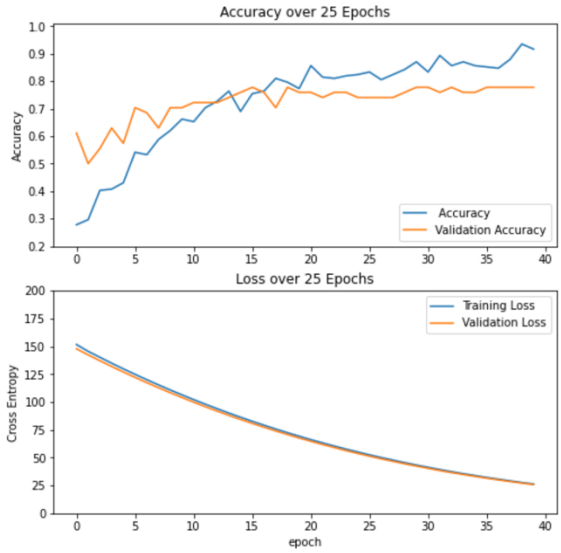

Figure 6 – Model 1 Metrics

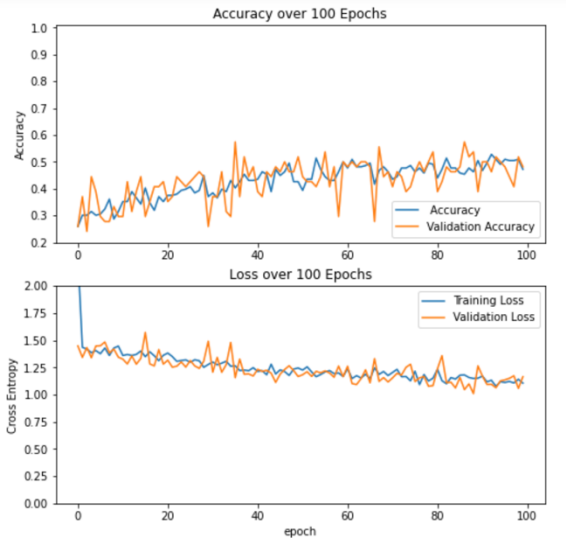

Figure 5 – Model 2 Metrics

Part 3: Plot and comment on the t-SNE visualizations

For both models, I found the first dense layer to have the best features that best separated the data into distinct clusters. For model 1, there are essentially 3 separate

clusters with one cluster that is vaguely separable into two sub-clusters as you can see in figure 7. It did a fantastic job of distinguishing between normal and COVID-19, but the features between bacterial Pnemonia and Viral Pnemonia are very similar, which is largely to be expected as they are similar diseases. For model 2, it seems to have only really separated normal from the rest of the classifications as a result of the Resnet 50 base model that simply was not suited for this task.

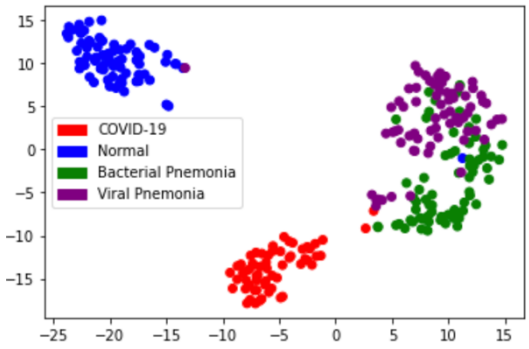

Figure 7 – Model 1 TSNE

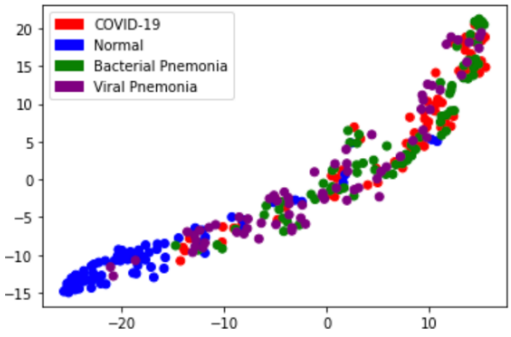

Figure 8 – Model 2 TSNE

This concludes the process of developing the models, from architecture to analysis and visualizations. Overall, though it was much work, it was very rewarding and very engaging! Great way to utilize many of the skills we have been learning in class. Thank you for your time!