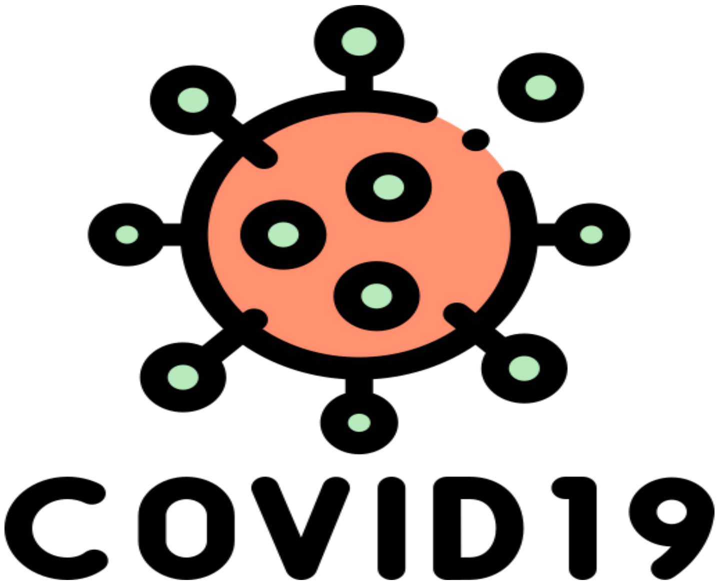
CS542 Class Challenge: Image Classification of COVID-19 X-rays



Soowhan Park (spark96@bu.edu)

April 29, 2021

Submitted for CS542 - Spring 2021

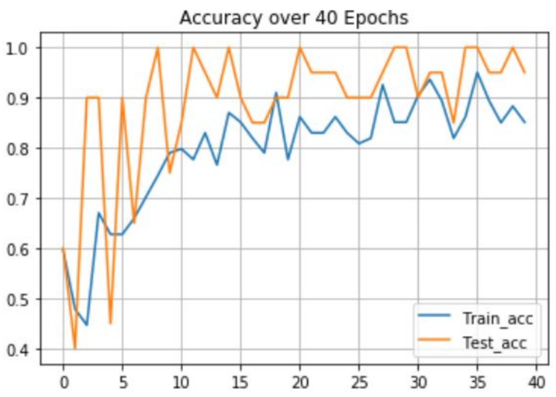
Task 1

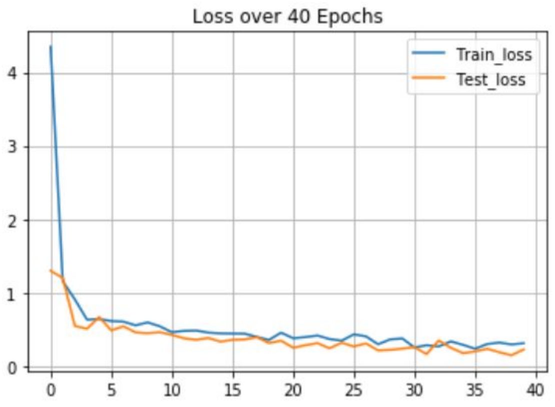
1. Architectures

Since we are given a small amount of data, it is helpful to use pre-trained models such as VGG16, ResNet50, VGG19, Inception and MobileNet. I used a VGG16 for Task 1, because it is one of the most commonly used pre-trained models to get started. VGG16's architecture contains pre-trained weights from imagenet, 3x3 filters, 224*224*3 image input size, and stride of 1. I've experimented with the test accuracy for both VGG16 and VGG19 and the results were very similar, so I chose VGG16 because it has a less complicated architecture with fewer layers compared to VGG19. Also, the test accuracy with VGG16 was 94.4%, which is very good enough, so I didn't feel it was necessary to experiment with other models further. I set the include_top to false to enable the transfer learning. After applying transfer learning, I flattened the layer, added a middle layer with a fully connected dimension of 256 with Relu activation function. I used dropout with a rate of 50% as a regularization to avoid overfitting. For the output layer, I used a single neuron since it is a binary output and applied sigmoid activation function, which is a softmax version of binary output.

Additionally, I experimented with 40 epochs, learning Rate with 0.0005, and a batch size of 5. I used the Adam optimization algorithm which is an extension of stochastic gradient descent. For loss function, I used Binary_Crossentropy function for the binary classification of either COVID-19 or normal. To avoid overfitting, I added a dropout layer with 50%.

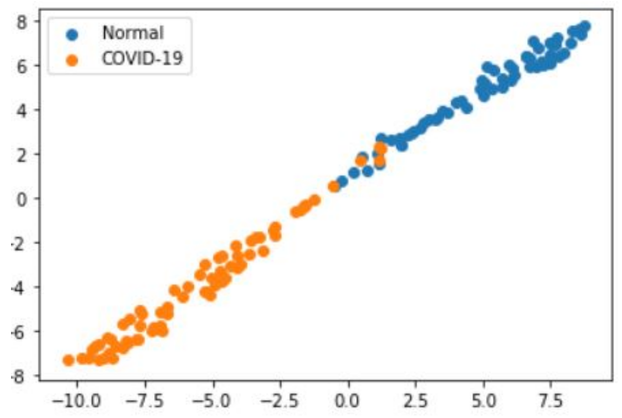
2. Accuracy & Loss





Both training and validation accuracies have a trend of increasing, which explains that the model is working well. Also, the validation accuracy is better than training accuracy, mainly due to the regularization used during the training because the weights are randomly dropped during the training which leads to lower accuracy. This also explains that the dropout was applied only during the training. For the loss, both training and validation have a trend of decreasing. Since the validation loss is not increasing at the later epoch, we can assume that the model is not overfitting. Overall, when I tested the model with unseen test samples, I was able to reach 94% accuracy and 0.3084 loss, so I can conclude that the model is doing well.

3. t-SNE



For the visualization, t-SNE is used to produce the 2d plot and reduce the dimensionality of the data into a two dimensional feature space. The red dots represent the X-ray images of COVID-19 patients, while the blue dots represent the X-ray images of normal patients. It is very clear to see the two distinct clusters between two classes, though there are few overlapping points. This result proves the high accuracy produced by the model and shows that there are ambiguities on the few sample images.

Task2

1. Architectures

a. Model 1

For Model1, I followed the model summary from the assignment template with some modifications. After using VGG16 as a pre-trained model, I flattened the layer and added fully connected layers with 256 neurons with ReLU activation function. After testing with a single fully connected layer and didn't get a satisfying testing accuracy, I added another fully connected layer with the same number of neurons and same activation function to learn more features. I applied dropout with 50% rates after each fully connected layer as a regularization method. Since we are classifying 4 classes, I put 4 neurons with softmax activation function for the output layer. I set the logtis to false to make sure that the output of the neural network to be normalized with softmax function. Also, I was able to increase the accuracy of the model from 61% to 66% after using a model checkpoint to save the optimal results and weights before fitting into the test set.

I experimented with 100 epochs, learning rate with 0.0001, and a batch size of 5. I used the Adam optimization algorithm which is an extension of stochastic gradient descent. For loss function, I used Categorical CrossEntropy function instead of Binary Cross Entropy for the multi-class classification. For regularization, I used two drop outs with 25% rates.

b. Model 2

For Model 2, I've experimented with different pretrained models, namely, ResNet50, VGG19, and MobileNet. I chose MobileNet among the others because I was looking for a light-weight neural network that gives good accuracy. In the beginning, the validation accuracy and test accuracy were not as high as training accuracy, but I thought I could minimize the gaps between them after modifying hyper parameters and adding regularization methods. I added another fully connected layer with 256 neurons with ReLU function to have a similar model complexity as Model1, so that I can directly compare the VGG16 and MobileNet. Ultimately, I was able to minimize the fluctuations between the epochs as well as the gap between training accuracy and validation accuracy. As a result, the test accuracy increased from 55% to 75%.

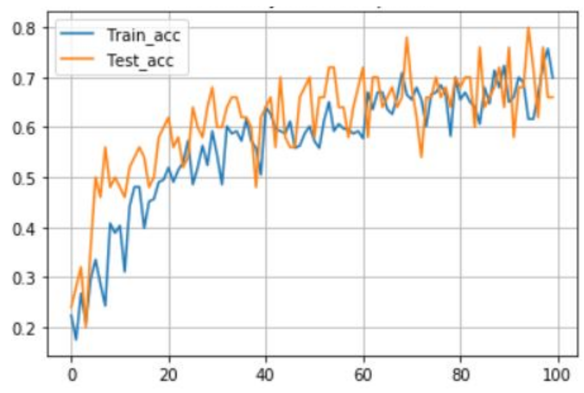
I experimented with 100 epochs, learning rate with 0.0001, and a batch size of 10. I used the Adam optimization algorithm which is an extension of stochastic gradient descent. For loss function, I used Categorical CrossEntropy function instead of Binary Cross Entropy for the multi-class classification. For regularization, I used two dropouts, one with 50% and another with 25% to prevent the loss increasing at the later epochs.

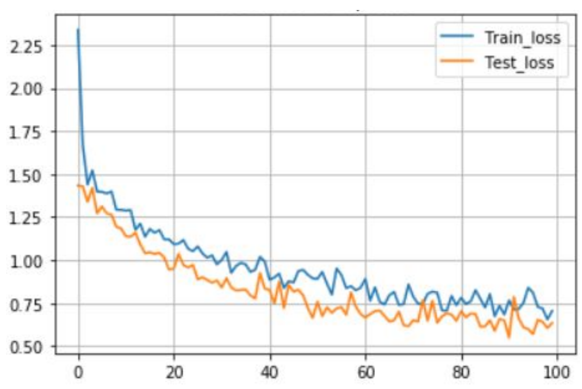
2. Model Comparison

Both Model 1 and Model 2 have similar architectures: two middle layers with 256 neurons and ReLU activation function, dropout layers after each middle layer, and output layer with 4 neurons with softmax activation functions. The main difference is that the second model used MobileNet as a pretrained model, whereas the first model used VGG16. My intuition of Task2 is to compare how the model complexity affects the test accuracy. I assumed there is a correlation between the light-weight neural network and small number of samples which contributes to overall test accuracy. Thus, I chose Mobilenet for the second model, because it is known for the light-weight neural network model. When I trained the first model without freezing the layers, the computation time was too long, so I set the trainable to false for efficiency. For the second model with Mobilenet, I didn't have to freeze the layer because the computation time was reasonable. Instead, I applied higher dropout rates for the second model. After looking at the test accuracy for both tasks, I was able to infer that freezing the layer led to losing some information because the weights are not being updated during training, which led to lower accuracy. It is interesting to see that the second model has a lower loss although it has higher dropout rates. It might be due to VGG 16 already having a more complex architecture than MobileNet. In conclusion, the second model using a Mobilenet performed better than the first model using VGG16 in terms of accuracy, loss and computation complexity.

3. Accuracy & Loss

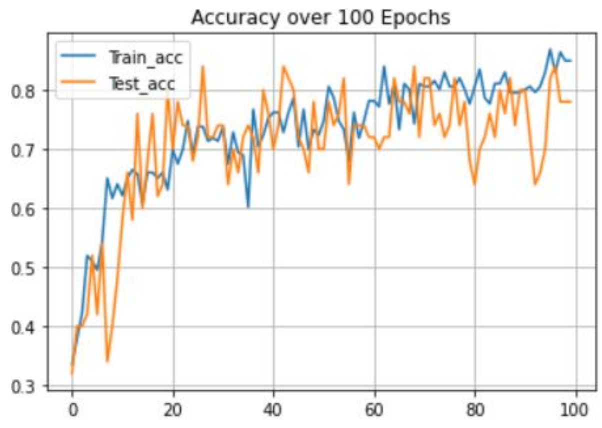
a. Model 1

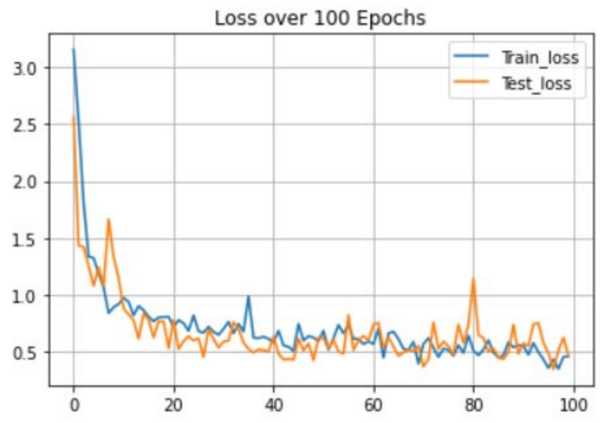




There is a trend of going upwards both training accuracy and validation accuracy. Although both training and validation accuracy is around 75%, the testing accuracy was 66%. This indicates that there are meaningful differences between training images and the testing images. For example, there might be many images of COVID-19 and normal, but then test images have more pneumonia images. However, the initial accuracy was around 61%, but I was able to increase it after using a model checkpoint to save the best results and weights during the training. Also, freezing the layer affected testing accuracy as the weights are not getting updated during the training. For loss, there is a trend going downward and there is no sign of overfitting since the loss is not increasing. Since there is no overfitting, the accuracy can be increased with more epochs and use early stopping to detect overfitting. In conclusion, the testing accuracy was 66% and testing loss was 0.79, so the model is not doing so well.

b. Model 2

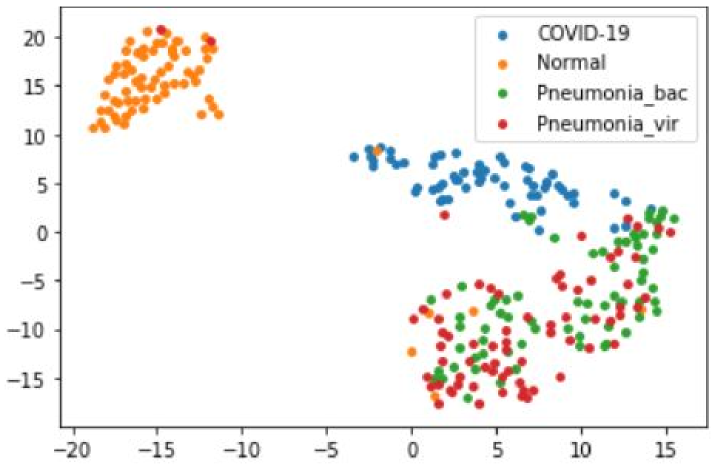




In the beginning, the training accuracy was over 90%, but the validation accuracy was around 80%. Based on the high training accuracy, I thought this model could give me a good result with some modifications to resolve the overfitting. One main reason for overfitting is because I did not freeze the layer, so all the weights were being updated during the training. Thus, I added two dropouts to reduce the gap between training accuracy and validation accuracy. As a result, I was able to increase the testing accuracy from 55% to 75%, though the training accuracy got lower. The general trend for training accuracy is increasing, although the validation starts to fluctuate after 50 epochs. This suggests that the model is still slightly overfitting, so the model can generalize better when using early stopping or adding other regularization techniques. For loss, the training has a clear trend of decreasing, while the validation decreases until 50 epochs and starts to fluctuate. However, it is hard to tell the loss is increasing, because there is no clear trend of increasing. In conclusion, the testing accuracy was 0.75 and testing loss was 0.69, so the model works decently.

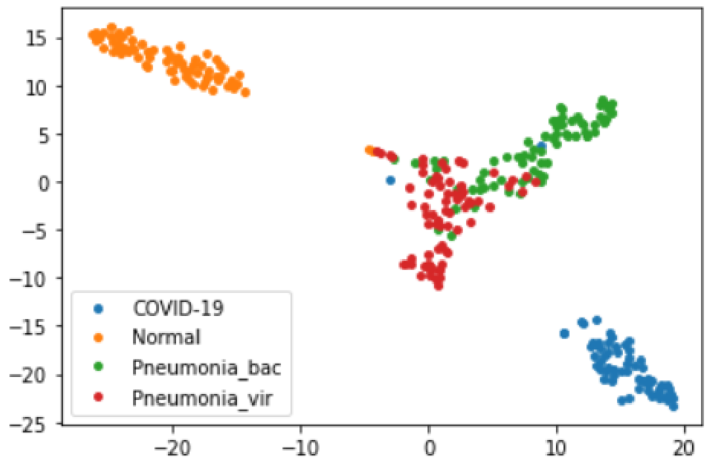
4. t-SNE

a. Model 1



For the visualization of classifying four classes, t-SNE is used to produce the 2d plot. The red dots represent the X-ray images of pneumonia-viral patients, green dots represent the X-ray images of pneumonia-bacterial patients, blue dots represent the X-ray images of COVID-19 patients, and normal dots represent the X-ray images of normal patients. There are two distinct clusters, COIVD-19 images and normal images, but there are lots of overlaps between pneumonia bacterial images and pneumonia-viral images. We can infer that it is hard to distinguish the difference between pneumonia-bacterial and pneumonia-viral, mainly due to the minor details between the images of two classes. This suggests that it might be helpful to use a binary classification for the pneumonia X-ray images. Overall, the visualization explains the low accuracy produced with Model 1.

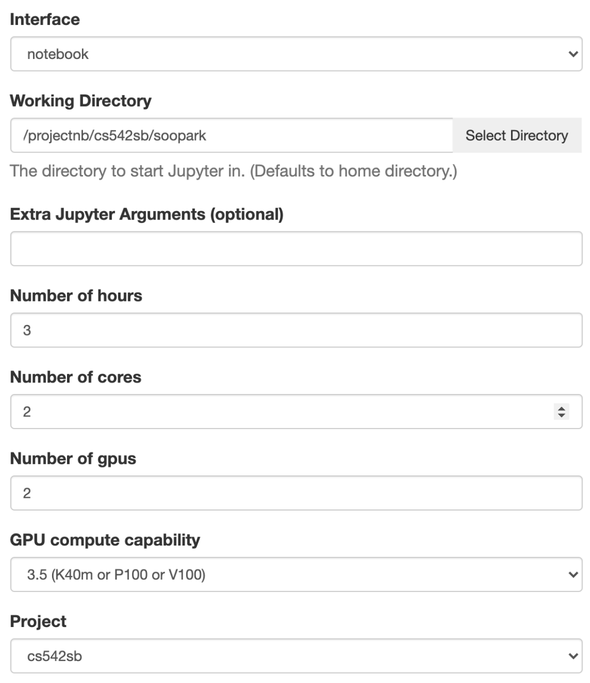
b. Model 2



Model 2 outperformed the Model 1 for creating clusters among the classes. Although there are some overlaps between pneumonia-bacterial and pneumonia-viral, it is clear to see the two clusters. Also, there are only a few overlaps for COVID-19 and Normal, which explain that the model is able to distinguish very well for those classes. This plot explains the 75% accuracy of the model, since there are some misclassifications and some good classifications.

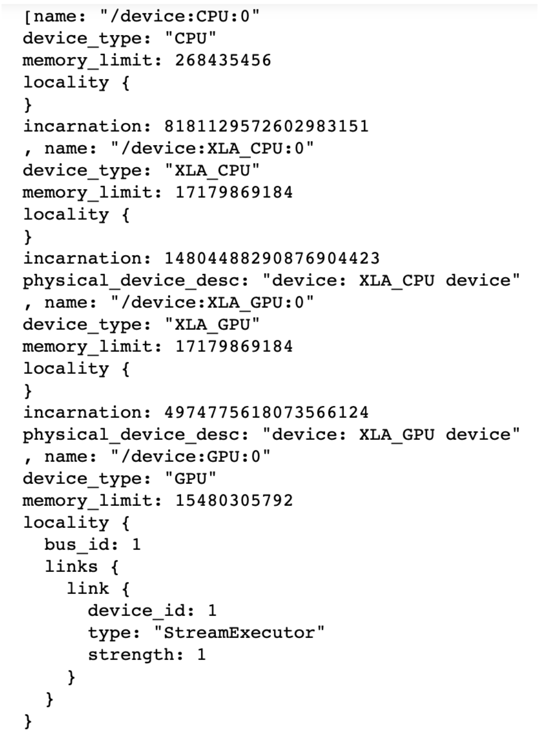
Bonus

As the video guided, I used interactive apps on SCC to run the jupyter notebook. I requested two CPUs and GPUs to avoid the task getting killed while running the code.



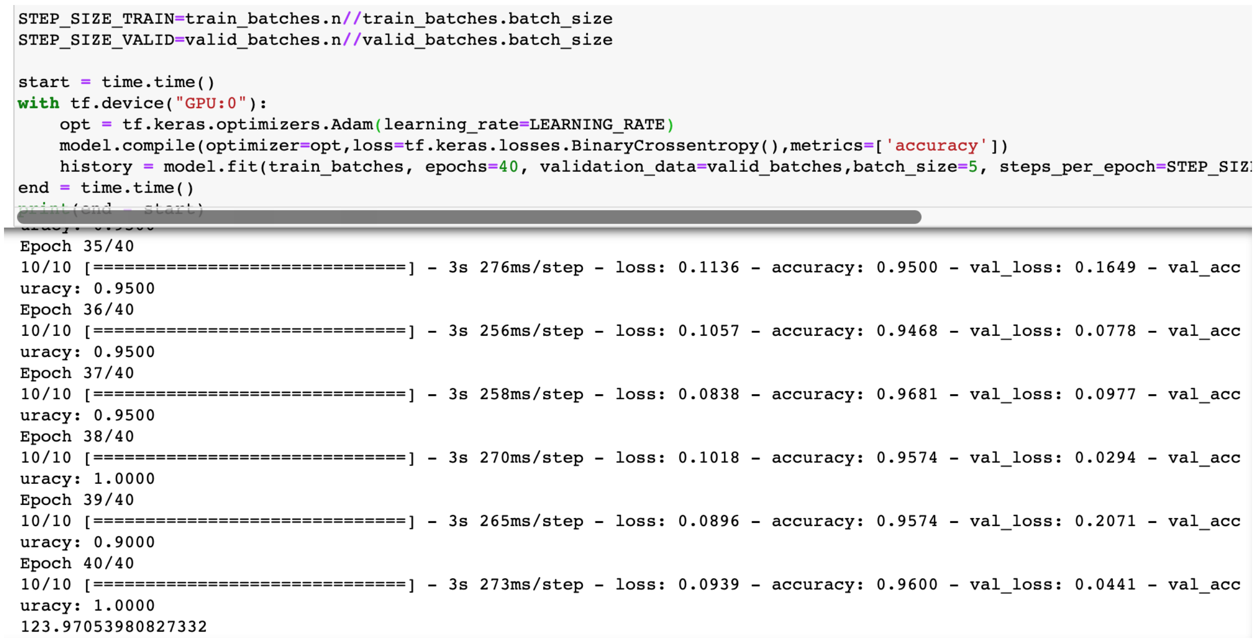
Interface	notebook
Working Directory	/projectnb/cs542sb/soopark
Extra Jupyter Arguments (optional)	
Number of hours	3
Number of cores	2
Number of gpus	2
GPU compute capability	3.5 (K40m or P100 or V100)
Project	cs542sb

The directory to start Jupyter in. (Defaults to home directory.)



```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 8181129572602983151
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 14804488290876904423
physical_device_desc: "device: XLA_CPU device"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 4974775618073566124
physical_device_desc: "device: XLA_GPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 15480305792
locality {
  bus_id: 1
  links {
    link {
      device_id: 1
      type: "StreamExecutor"
      strength: 1
    }
  }
}
```

<GPU>

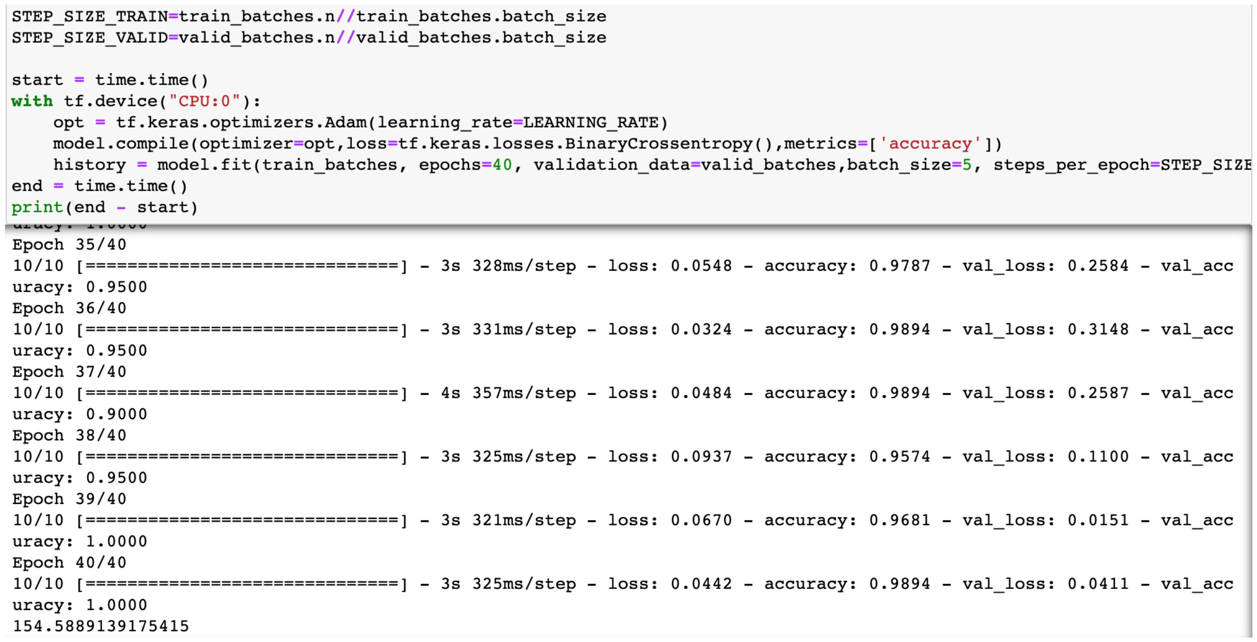


```
STEP_SIZE_TRAIN=train_batches.n//train_batches.batch_size
STEP_SIZE_VALID=valid_batches.n//valid_batches.batch_size

start = time.time()
with tf.device("GPU:0"):
    opt = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
    model.compile(optimizer=opt,loss=tf.keras.losses.BinaryCrossentropy(),metrics=['accuracy'])
    history = model.fit(train_batches, epochs=40, validation_data=valid_batches,batch_size=5, steps_per_epoch=STEP_SIZ
end = time.time()
print(end - start)
```

```
uracy: 0.9500
Epoch 35/40
10/10 [==============================] - 3s 276ms/step - loss: 0.1136 - accuracy: 0.9500 - val_loss: 0.1649 - val_acc
uracy: 0.9500
Epoch 36/40
10/10 [==============================] - 3s 256ms/step - loss: 0.1057 - accuracy: 0.9468 - val_loss: 0.0778 - val_acc
uracy: 0.9500
Epoch 37/40
10/10 [==============================] - 3s 258ms/step - loss: 0.0838 - accuracy: 0.9681 - val_loss: 0.0977 - val_acc
uracy: 0.9500
Epoch 38/40
10/10 [==============================] - 3s 270ms/step - loss: 0.1018 - accuracy: 0.9574 - val_loss: 0.0294 - val_acc
uracy: 1.0000
Epoch 39/40
10/10 [==============================] - 3s 265ms/step - loss: 0.0896 - accuracy: 0.9574 - val_loss: 0.2071 - val_acc
uracy: 0.9000
Epoch 40/40
10/10 [==============================] - 3s 273ms/step - loss: 0.0939 - accuracy: 0.9600 - val_loss: 0.0441 - val_acc
uracy: 1.0000
123.97053980827332
```

<CPU>



```
STEP_SIZE_TRAIN=train_batches.n//train_batches.batch_size
STEP_SIZE_VALID=valid_batches.n//valid_batches.batch_size

start = time.time()
with tf.device("CPU:0"):
    opt = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
    model.compile(optimizer=opt,loss=tf.keras.losses.BinaryCrossentropy(),metrics=['accuracy'])
    history = model.fit(train_batches, epochs=40, validation_data=valid_batches,batch_size=5, steps_per_epoch=STEP_SIZE
end = time.time()
print(end - start)
```

```
uracy: 1.0000
Epoch 35/40
10/10 [==============================] - 3s 328ms/step - loss: 0.0548 - accuracy: 0.9787 - val_loss: 0.2584 - val_acc
uracy: 0.9500
Epoch 36/40
10/10 [==============================] - 3s 331ms/step - loss: 0.0324 - accuracy: 0.9894 - val_loss: 0.3148 - val_acc
uracy: 0.9500
Epoch 37/40
10/10 [==============================] - 4s 357ms/step - loss: 0.0484 - accuracy: 0.9894 - val_loss: 0.2587 - val_acc
uracy: 0.9000
Epoch 38/40
10/10 [==============================] - 3s 325ms/step - loss: 0.0937 - accuracy: 0.9574 - val_loss: 0.1100 - val_acc
uracy: 0.9500
Epoch 39/40
10/10 [==============================] - 3s 321ms/step - loss: 0.0670 - accuracy: 0.9681 - val_loss: 0.0151 - val_acc
uracy: 1.0000
Epoch 40/40
10/10 [==============================] - 3s 325ms/step - loss: 0.0442 - accuracy: 0.9894 - val_loss: 0.0411 - val_acc
uracy: 1.0000
154.5889139175415
```

As the result above suggests, GPU is faster than CPU.