Assumption, coding the assignment, and issues encountered

From reading the instructions, I will assume that we need to follow the pytorch8 timm file to create and do our homework. From step 1, I will also be assuming that the dataset we need to use is the mednist dataset from the linked referenced file. I will be doing this homework assignment in google colab and all the files to the code can be accessed here.

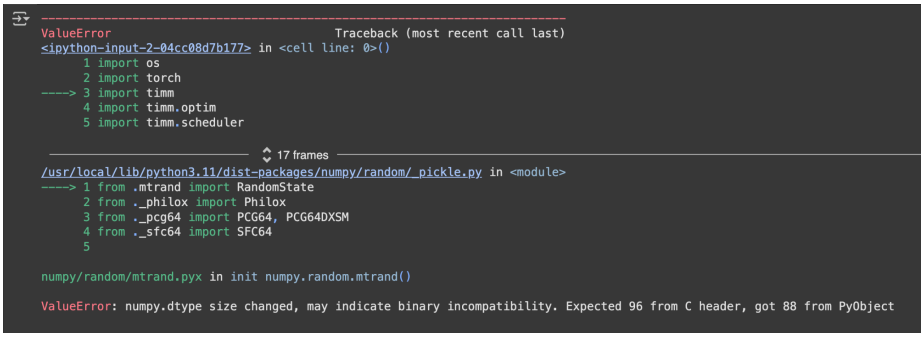
First, I started with setting up the dataset. I took the code to download the dataset from the mednist tutorial.

```
# download dataset
resource = "https://github.com/Project-MONAI/MONAI-extra-test-data/releases/download/0.8.1/MedNIST.tar.gz"
md5 = "0bc7306e7427e00ad1c5526a6677552d"

compressed_file = os.path.join(root_dir, "MedNIST.tar.gz")
data_dir = os.path.join(root_dir, "MedNIST")
if not os.path.exists(data_dir):
    download_and_extract(resource, compressed_file, root_dir, md5)
```

I noticed that at the bottom of the linked timm file, there is code to train a model. I assume that is the base code talked about in the assignment so I used it to complete the training code.

Before running the code, I ran into a bit of a problem in google colab. For whatever reason, when I try to import timm, it would not compile and throw an error. I tried almost everything I could to fix it. Delete and restart the session, uninstalling a combination of numpy, torch or torch vision and installing it again, etc. This was frustrating because each time I tried something, it would take a couple minutes to see the results since these libraries are sort of big. In the end I found the problem to be that I was installing timm using the pip command at the start. Since google collab already has timm installed, all I was doing was updating it and causing something to be incompatible with something else. So I just left timm out of the pip install and it was fine.



After this, I ran the code and it was able to train a model. Sadly, only after it completed training did I realize I had forgotten to set the seed for cuda, so I had to restart.

```
15  torch.manual_seed(42)
16  torch.cuda.manual_seed(42)
```

I have also looked at the rest of the assignment and found out we were supposed to do model export and inference optimization. I decided to add that into my code before doing any

more training. I went through the pytorch9 timm file to figure how to do that. I ended up using onnx to export my code and do an inference test using openvino as well as onnx to compare them. I will also be including the .onnx file for each model in the git repository.

```
# Export the model
torch.onnx.export(model,
                  x,
                  "restnet18-1.onnx",
                  export_params=True,
                  opset_version=10,
                  do_constant_folding=True,
                  input_names = ['input'],
                  output_names = ['output'],
                  dynamic_axes={'input' : {0 : 'batch_size'},
                                'output': {0 : 'batch_size'}})
```

I went through my whole code twice over to make sure I got everything correct before continuing training since the gpu resources on google colab is limited per day. This is when I encountered the timm import error again. It is still an incompatibility error but I know there shouldn't be anything wrong since I made sure to only pip install stuff that google colab doesn't already have. I found the fix however, just restarting the runtime instead of deleting the runtime for some reason made the error go away. This seems to be a common problem since I have seen a couple of classmates reach out about imports returning this error. I told them about restarting the session instead of deleting and it fixed it so it's all good.

For the models I will be training in the next section, I will be using resnet18 instead of resnet50d from the sample code because I think it will take a little bit less time to train. I also added the code to freeze all the layers except for the last FC layer for the same reason. I will also be only training each model with 5 epochs since I think it is the perfect balance of training time for the mode and time for me waiting for it to train.

```
for param in model.parameters():
    param.requires_grad = False


for param in model.fc.parameters():
    param.requires_grad = True
```

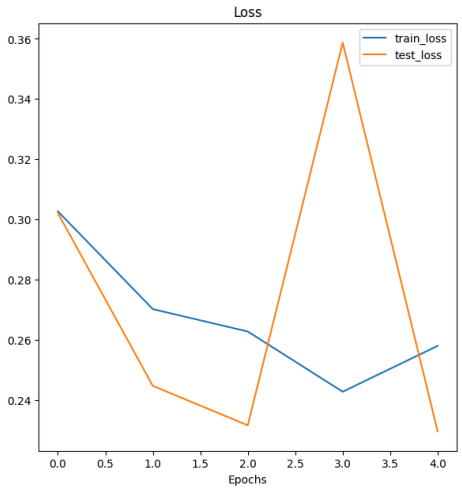
Also, for the inference test with onnx and openvino, I will use a batch of 16 from my training dataset unlike the sample code where only 1 image was used. CPU will also be used for both to make it fair.

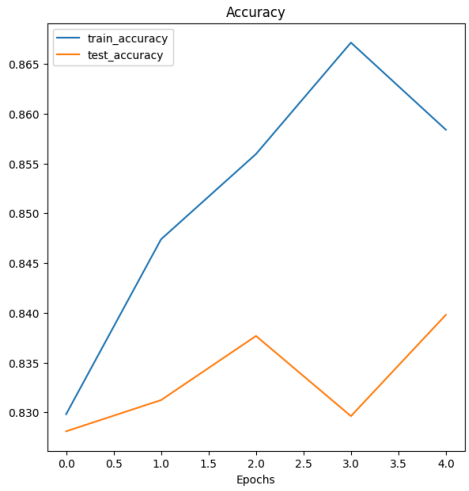
Trying different configurations, graphs, and analysis

The number next to each model indicates the order in which I trained the model. Each section will follow this format:
Description/what I changed in the configuration -> graphs of loss and accuracy, some stats from the model, and the inference time from onnx runtime and openvino -> some observation I made.

Resnet18-1

This is the model that I will be comparing other models that I trained to since it is the first one. Since it is the first one, I will keep everything simple. No data augmentation, batch of 16, AdamP as the optimizer, cross entropy loss as the loss function, and CosineLRScheduler as the scheduler. For detailed parameters in each variable, please refer to the code here.





	precision	recall	f1-score	support
AbdomenCT	0.9023	0.0609	0.1141	1971
BreastMRI	1.0000	0.9977	0.9989	1757
CXR	0.5218	1.0000	0.6857	2025
ChestCT	0.9969	0.9943	0.9956	1936
Hand	0.9946	0.9889	0.9917	2066
HeadCT	0.9985	1.0000	0.9993	2036
accuracy			0.8398	11791
macro avg	0.9023	0.8403	0.7975	11791
weighted avg	0.8998	0.8398	0.7955	11791

```
ONNX Runtime CPU Inference time = 782.53 ms
OpenVINO CPU Inference time = 1553.54 ms
```

For this first model, the training time took 688 seconds. Looking at the loss and accuracy graph, I see some concerning features. Starting at epoch 2, the loss for the test set spiked while the train set remained normal. This is also reflected in the accuracy graph where the train accuracy continues to climb while the test set accuracy dropped after the second epoch. It continues to climb as normal after and the difference between them is not that far off (~2% difference), however this might indicate something went wrong during training. I think the model might be a little bit overfit on the training set. As for the inference test, the onnx runtime is

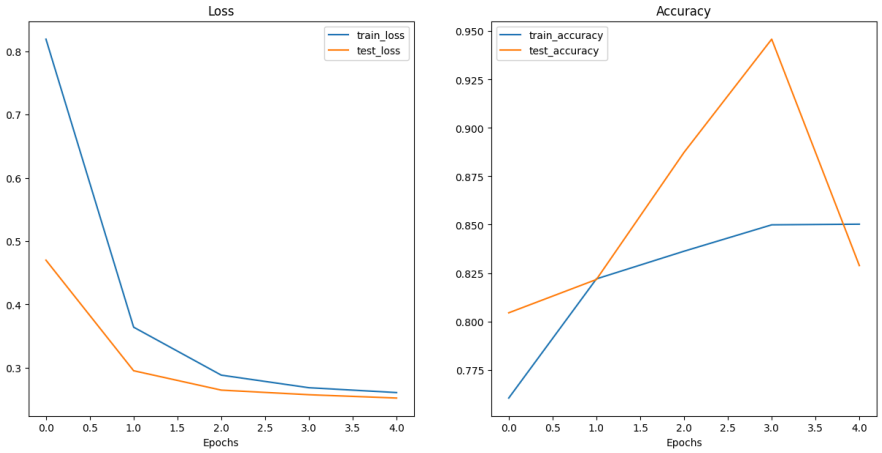
almost 2 times faster than openvino. This is surprising since I didn't think there was this much of a difference.

Resnet18-2

Since the first model showed a little of overfitting, I decided to try and prevent it. I think this is happening because my learning rate is too high, so I lowered the learning rate in the optimizer by one decimal and the warmup learning rate initialization in the scheduler by 2.

```
# optimizer = timm.optim.AdamP(model.parameters(), lr=0.01)
optimizer = timm.optim.AdamP(model.parameters(), lr=0.001)
scheduler = timm.scheduler.CosineLRScheduler(optimizer,
                                             t_initial=num_epoch_repeat,
                                             lr_min=1e-5,
                                             # warmup_lr_init=0.01,
                                             warmup_lr_init=0.0001,
                                             warmup_t=3,
                                             cycle_limit=num_epoch_repeat+1
                                            )
```

This should Make the loss a little smoother and hopefully reduce the overfitting problem.



	precision	recall	f1-score	support
AbdomenCT	0.5020	1.0000	0.6684	2002
BreastMRI	0.9994	0.9994	0.9994	1725
CXR	1.0000	0.0177	0.0349	1973
ChestCT	0.9901	0.9955	0.9928	2015
Hand	0.9954	0.9660	0.9805	2027
HeadCT	0.9990	0.9995	0.9993	2049
accuracy			0.8289	11791
macro avg	0.9143	0.8297	0.7792	11791
weighted avg	0.9127	0.8289	0.7774	11791

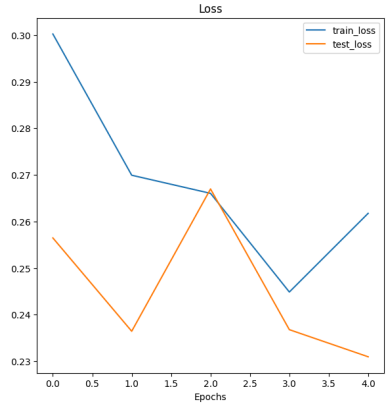
```
ONNX Runtime CPU Inference time = 752.68 ms
OpenVINO CPU Inference time = 1056.13 ms
```

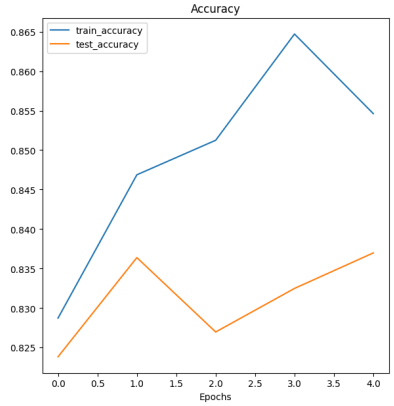
This model took 678 seconds to train. Looking at the loss and accuracy graph, it is looking a lot better compared to the first model. The loss with the train dataset and test set is curving downward which is expected in a normal model. There is still the little dip of accuracy in the test accuracy so that is a bit concerning. I think that indicates the model is still a tiny bit overfit but it should be fine for now. The inference time is similar to the first model where the onnx runtime is significantly faster than openvino. I don't know why this is since after some light research, my understanding is that opnvino is more optimized for cpu usage. Since I am running both using cpu, openvino should be faster but it's not. The only thing I can think of is that openvino is faster, but it has bigger overhead so you would only see the benefits if the model is really big.

Resnet18-3

For this next model, I want to see the effects of using a different optimizer so I change the optimizer from adamp to adamw.

```
# optimizer = timm.optim.AdamP(model.parameters(), lr=0.01)
optimizer = timm.optim.AdamW(model.parameters(), lr=0.01, weight_decay=0.01)
```





	precision	recall	f1-score	support
AbdomenCT	0.9076	0.0545	0.1029	1981
BreastMRI	0.9994	1.0000	0.9997	1784
CXR	0.5175	1.0000	0.6821	2022
ChestCT	0.9910	0.9955	0.9933	2001
Hand	0.9965	0.9800	0.9882	2005
HeadCT	1.0000	1.0000	1.0000	1998
accuracy			0.8370	11791
macro avg	0.9020	0.8383	0.7943	11791
weighted avg	0.8995	0.8370	0.7916	11791

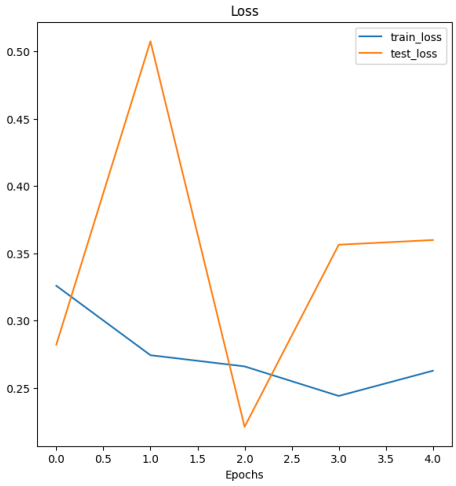
```
ONNX Runtime CPU Inference time = 506.79 ms
OpenVINO CPU Inference time = 594.81 ms
```

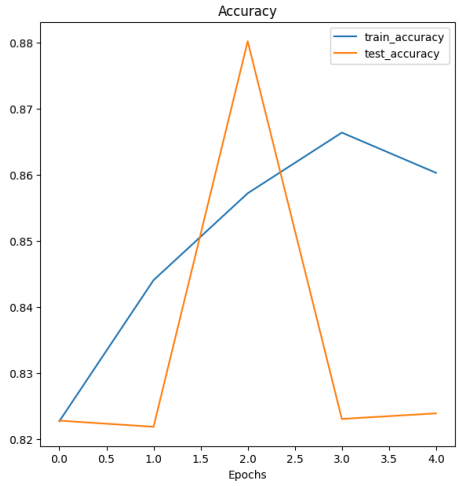
This model took 674 seconds to train. From looking at the loss and accuracy graph, it looks a lot like that of the graphs in the first model. I left the learning rate parameter for the optimizer at 0.01 so the only difference between this model and the first one is the optimizer used. From that I can see that the loss of the test dataset for both adamp and adamw still has the spike but the spike in adamw is not as drastic. The accuracy remains basically the same though. The biggest difference is in the inference time. The difference between openvino and onnx runtime is not that big. I don't know why this is the case. The optimizer shouldn't affect inference time. I think it might either be a fluke or something to do with me doing the test on google colab.

Resnet18-4

Continuing with the last model, I tried to train a model using another optimizer. This time I will try adabelief.

```
# optimizer = timm.optim.AdamP(model.parameters(), lr=0.01)
# optimizer = timm.optim.AdamW(model.parameters(), lr=0.01, weight_decay=0.01)
optimizer = timm.optim.AdaBelief(model.parameters(), lr=0.01, betas=(0.9, 0.999), weight_decay=1e-4)
```





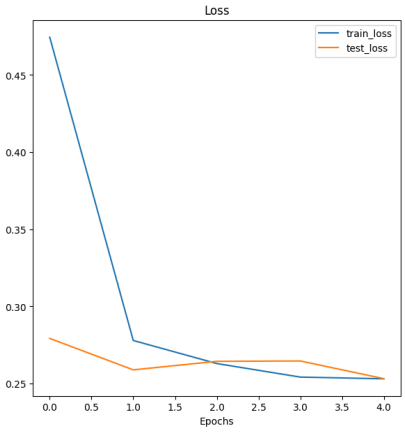
	precision	recall	f1-score	support
AbdomenCT	0.0000	0.0000	0.0000	2022
BreastMRI	0.9983	0.9994	0.9989	1771
CXR	0.4904	1.0000	0.6581	1966
ChestCT	0.9941	0.9960	0.9951	2021
Hand	0.9969	0.9785	0.9877	2004
HeadCT	1.0000	0.9990	0.9995	2007
accuracy			0.8239	11791
macro avg	0.7466	0.8288	0.7732	11791
weighted avg	0.7418	0.8239	0.7683	11791

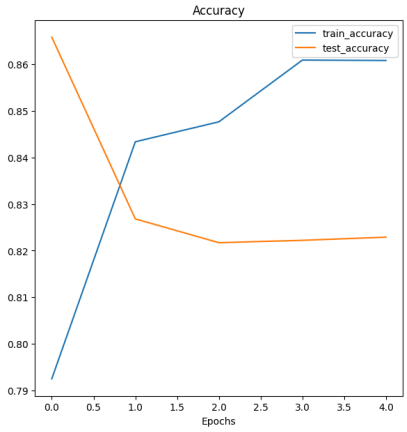
```
ONNX Runtime CPU Inference time = 551.21 ms
OpenVINO CPU Inference time = 595.89 ms
```

This model took 672 seconds to train. From looking at the graphs, I can see that something has gone horribly wrong since the loss for the test dataset spiked and dipped out of control. The accuracy for it also does not look right since it went up about 5% between epoch 1 and 3. After that it went down to the original accuracy and basically stayed the same throughout the next epoch. I think this is because of the learning rate again like that in model 1. I left the learning rate parameter in the optimizer at 0.01 so I can compare it to the first model but this might be way too big for a learning rate. Because of this, this model might also be overfit since the training loss and accuracy looks normal. For the inference time, it is similar to the last model where the difference between onnx runtime and openvino is not that big. I am starting to think this has something to do with google colab since I don't think changing optimizer would affect the inference time.

Resnet18-5

Since I think the last model is overfitted, I will do what I did for model 2 where I reduce the learning rate in the optimizer and the warmup learning rate init in the scheduler by 1 decimal point.





	precision	recall	f1-score	support
AbdomenCT	0.3000	0.0029	0.0058	2047
BreastMRI	1.0000	0.9983	0.9992	1805
CXR	0.4902	1.0000	0.6579	1978
ChestCT	0.9970	0.9950	0.9960	2020
Hand	0.9959	0.9829	0.9894	1989
HeadCT	0.9985	1.0000	0.9992	1952
accuracy			0.8229	11791
macro avg	0.7969	0.8299	0.7746	11791
weighted avg	0.7915	0.8229	0.7673	11791

```
ONNX Runtime CPU Inference time = 783.72 ms
OpenVINO CPU Inference time = 506.35 ms
```

This model took 677 seconds to train. From the graph, I can see that lowering the learning rate helped with the sporadic jump with test loss and accuracy. The loss lowered with each epoch like a normal epoch. The accuracy is a little bit weird but I think this is fine since the model might still be a little bit overfit on the training data. As for the inference time, I am surprised that openvino performed better than onnx runtime. I ran this multiple times too and it has the same result every time.

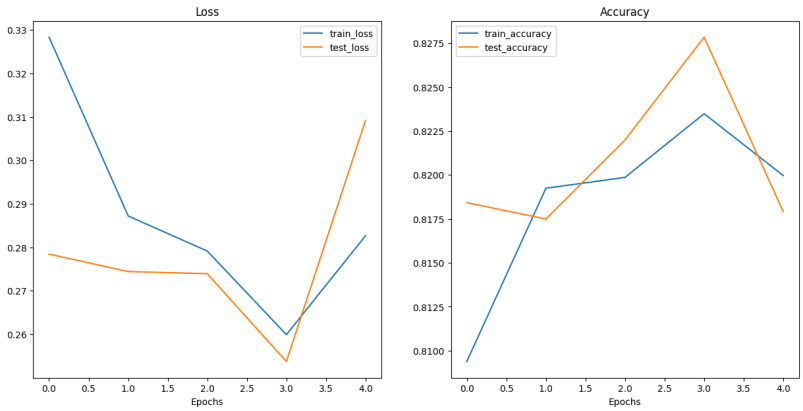
Resnet18-6

For this next model, I got curious on what would happen if I don't freeze all the layers of the model before training.

```
# for param in model.parameters():
#     param.requires_grad = False


# for param in model.fc.parameters():
#     param.requires_grad = True
```

I commented out the code to freeze the model so every layer in the base model will be trained with the new data.



	precision	recall	f1-score	support
AbdomenCT	0.0000	0.0000	0.0000	1992
BreastMRI	0.9967	1.0000	0.9983	1809
CXR	0.4943	1.0000	0.6616	2046
ChestCT	0.9899	0.9611	0.9753	2032
Hand	0.9941	0.9606	0.9771	1928
HeadCT	0.9915	1.0000	0.9957	1984
accuracy			0.8179	11791
macro avg	0.7444	0.8203	0.7680	11791
weighted avg	0.7387	0.8179	0.7634	11791

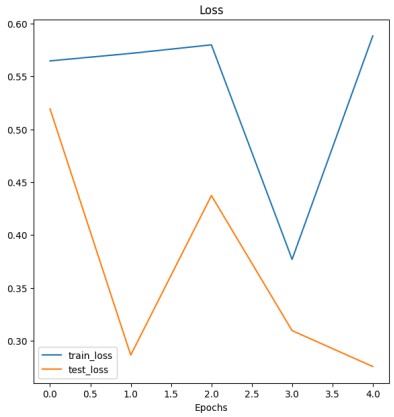
```
ONNX Runtime CPU Inference time = 726.08 ms
OpenVINO CPU Inference time = 825.00 ms
```

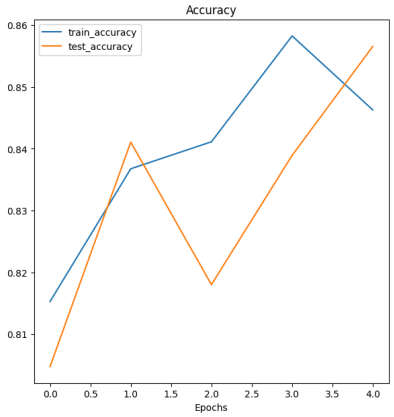
This model took 1129 seconds to train. As expected this looks longer (double the time) to train compared to just training the last FC layer. I think this big time increase is because of the back propagation which probably makes up a big part of the training time. For the graphs, it actually looks pretty normal. I am surprised by this because I kept every parameter the same as model 1 and as we know model one suffered from overfitting. Other than the loss and accuracy starting to sip after the third epoch, everything seems fine. The inference time looks as expected with the onnx runtime being faster than openvino.

Resnet18-7

For this next model, I want to see what would happen if I change the global pooling. I think resnet18 uses average global pooling so I changed it to use avgmax instead.

```python
# model = timm.create_model('resnet18', pretrained=True, num_classes=6, exportable=True)
model = timm.create_model('resnet18', pretrained=True, num_classes=6, exportable=True, global_pool='avgmax')
```





	precision	recall	f1-score	support
AbdomenCT	0.9417	0.1937	0.3213	2003
BreastMRI	0.9994	0.9910	0.9952	1781
CXR	0.5549	0.9995	0.7137	2021
ChestCT	0.9904	0.9959	0.9932	1968
Hand	0.9954	0.9752	0.9852	2019
HeadCT	0.9911	0.9995	0.9953	1999
accuracy			0.8566	11791
macro avg	0.9122	0.8592	0.8340	11791
weighted avg	0.9098	0.8566	0.8304	11791

```
ONNX Runtime CPU Inference time = 496.92 ms
OpenVINO CPU Inference time = 616.61 ms
```

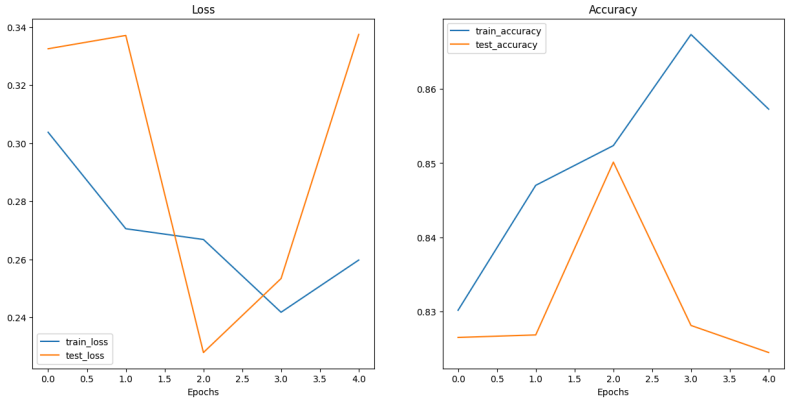
This model took 712 seconds to train. From looking at the graph, I think this looks pretty good compared to model 1. The loss graph is a bit jumpy but it looks like it's going down overall even with the train loss having a big jump up at the end. I think this is just the limitation of my number of epochs. I think it will be better if I leave it training for longer. The accuracy for this model is actually pretty good as we can see that the accuracy for both train and test go up overall throughout training. The inference time for this model looks normal with onnx runtime being faster than openvino.

Resnet18-8

Since the last model went well, I wanted to try something else. I want to see what happens if I apply some data augmentation.

```
# train_transform = create_transform(224, is_training=True)
train_transform = create_transform(224, is_training=True, auto_augment='rand-m9-mstd0.5')
```

I took the data augment parameter used from the sample code.



	precision	recall	f1-score	support
AbdomenCT	0.0000	0.0000	0.0000	1996
BreastMRI	0.9994	0.9944	0.9969	1775
CXR	0.4970	1.0000	0.6640	1995
ChestCT	0.9990	0.9833	0.9911	2032
Hand	0.9861	0.9852	0.9857	2023
HeadCT	0.9924	1.0000	0.9962	1970
accuracy			0.8244	11791
macro avg	0.7457	0.8271	0.7723	11791
weighted avg	0.7417	0.8244	0.7688	11791

```
ONNX Runtime CPU Inference time = 608.91 ms
OpenVINO CPU Inference time = 925.24 ms
```

This model took 680 seconds to train. From looking at the graphs, I can see that the data augment did not benefit the model. The train loss looks normal but the test loss did a massive dip then spike after epoch 1. I think the data augmentation might change the data too much so

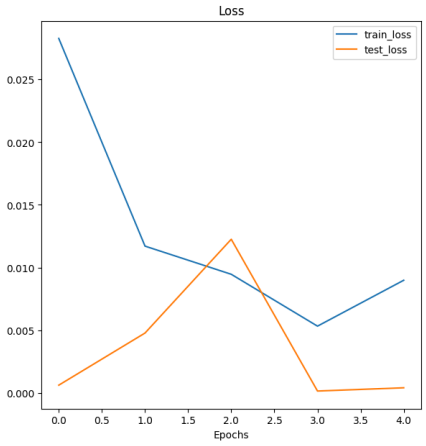
the model struggles with the training. This also might affect the model's generalization as the accuracy graph is a little bit worse than that of model 1. The inference time is as expected with onnx runtime being faster.

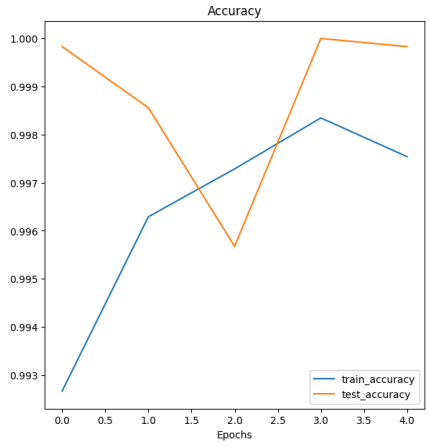
Resnet18-9

After the deadline for the assignment got extended, I have some more time to look at the models that I have trained. I realized that all of my models have not broken 90% accuracy. This is weird since images are clear, direct single class objects that a model could easily train on and achieve high accuracy. This is when I decided to look into the number of channels that my data have going into the model, and the number of image channels the model was taking in as input. I found out that since I used the timm transform like the sample code, it was automatically converting my image to 3 channels to match the model input. Since resnet18 takes 3 channels input, the data was transformed into 3 channels. So I decided to change both to be 1 channel only. I had to use torch vision for this since I don't think I can change this with timm.

```
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

```
model = timm.create_model('resnet18', pretrained=True, num_classes=6, exportable=True, in_chans=1)
```





	precision	recall	f1-score	support
AbdomenCT	1.0000	1.0000	1.0000	1972
BreastMRI	1.0000	0.9994	0.9997	1793
CXR	1.0000	1.0000	1.0000	1976
ChestCT	1.0000	1.0000	1.0000	2029
Hand	1.0000	0.9995	0.9997	1973
HeadCT	0.9990	1.0000	0.9995	2048
accuracy			0.9998	11791
macro avg	0.9998	0.9998	0.9998	11791
weighted avg	0.9998	0.9998	0.9998	11791

```
ONNX Runtime CPU Inference time = 539.61 ms
OpenVINO CPU Inference time = 570.27 ms
```

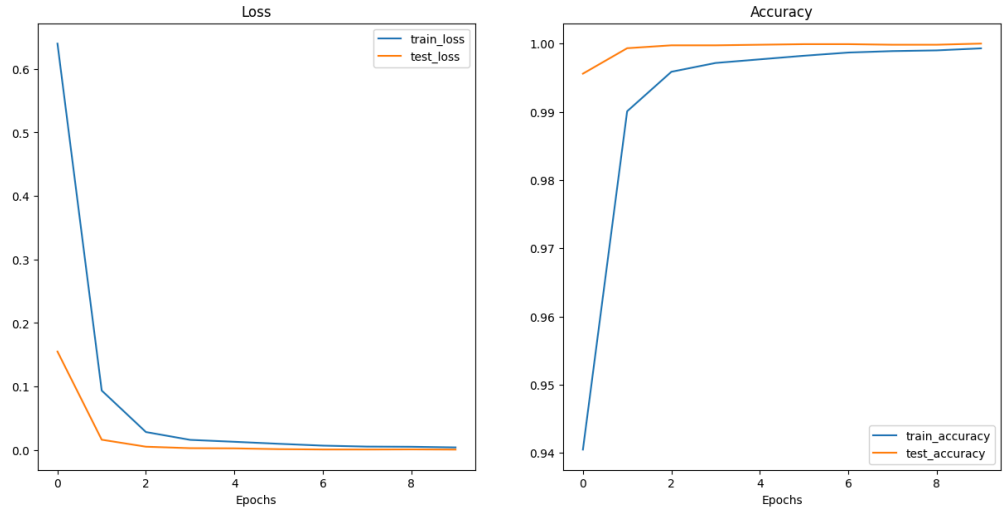
This model took 539 seconds to train which is faster than all the previous models that I trained. Looks like the number of channels can affect training time. The graphs for loss and accuracy still looks like there is a problem but I don't think it is a big deal considering the accuracy is already almost 100% and the oscillation in the accuracy and loss is in the 3rd decimal place. Just as I thought, transforming the data to 3 channels to fit the original resnet18 channel input was causing problems for my training. I will do one last model to try to smooth out the loss and accuracy graph.

Resnet18-10

For this last model, I want to do some of the adjustments that worked well before. I want to see if the graph will smoothing itself out eventually since the scheduler might take a while to stabilize the weights. From model 1, I know the learning rate is not good so alongside increasing the epoch by double, I will also lower the learning rate and warmup learning rate init.

```
batch_size=16
# num_epochs=5
num_epochs=10
```

```
optimizer = timm.optim.AdamP(model.parameters(), lr=0.001)
# optimizer = timm.optim.AdamP(model.parameters(), lr=0.01)
# optimizer = timm.optim.AdamW(model.parameters(), lr=0.001, weight_decay=0.01)
# optimizer = timm.optim.AdaBelief(model.parameters(), lr=0.001, betas=(0.9, 0.999), weight_decay=1e-4)
scheduler = timm.scheduler.CosineLRScheduler(optimizer,
                                             t_initial=num_epoch_repeat,
                                             lr_min=1e-5,
                                             warmup_lr_init=0.0001,
                                             warmup_t=3,
                                             cycle_limit=num_epoch_repeat+1
                                            )
```



	precision	recall	f1-score	support
AbdomenCT	1.0000	1.0000	1.0000	1972
BreastMRI	1.0000	1.0000	1.0000	1793
CXR	1.0000	1.0000	1.0000	1976
ChestCT	1.0000	1.0000	1.0000	2029
Hand	1.0000	1.0000	1.0000	1973
HeadCT	1.0000	1.0000	1.0000	2048
accuracy			1.0000	11791
macro avg	1.0000	1.0000	1.0000	11791
weighted avg	1.0000	1.0000	1.0000	11791

```
ONNX Runtime CPU Inference time = 515.04 ms
OpenVINO CPU Inference time = 563.07 ms
```

This model 1064 seconds to train. The graph looks almost perfect. It's not smooth but I think that's because of the scheduler or how simple data is and training it. The inference time looks normal as with previous models with onnx runtime being a little bit faster than openvino.