Prepare training, validation and test data lists

Randomly select 10% of the dataset as validation and 10% as test.

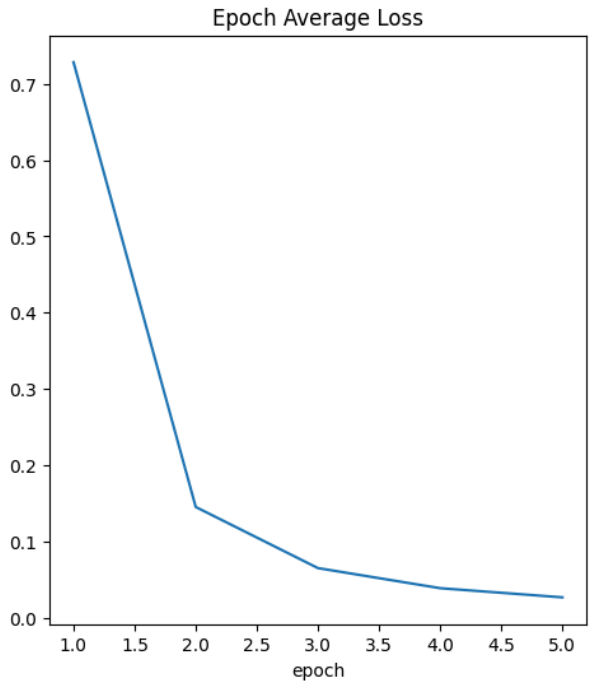
```
val_frac = 0.1
test_frac = 0.1
length = len(image_files_list)
indices = np.arange(length)
np.random.shuffle(indices)

test_split = int(test_frac * length)
val_split = int(val_frac * length) + test_split
test_indices = indices[:test_split]
val_indices = indices[test_split:val_split]
train_indices = indices[val_split:]

train_x = [image_files_list[i] for i in train_indices]
train_y = [image_class[i] for i in train_indices]
val_x = [image_files_list[i] for i in val_indices]
val_y = [image_class[i] for i in val_indices]
test_x = [image_files_list[i] for i in test_indices]
test_y = [image_class[i] for i in test_indices]

print(f"Training count: {len(train_x)}, Validation count: " f"{len(val_x)}, Test count: {len(test_x)}")
```

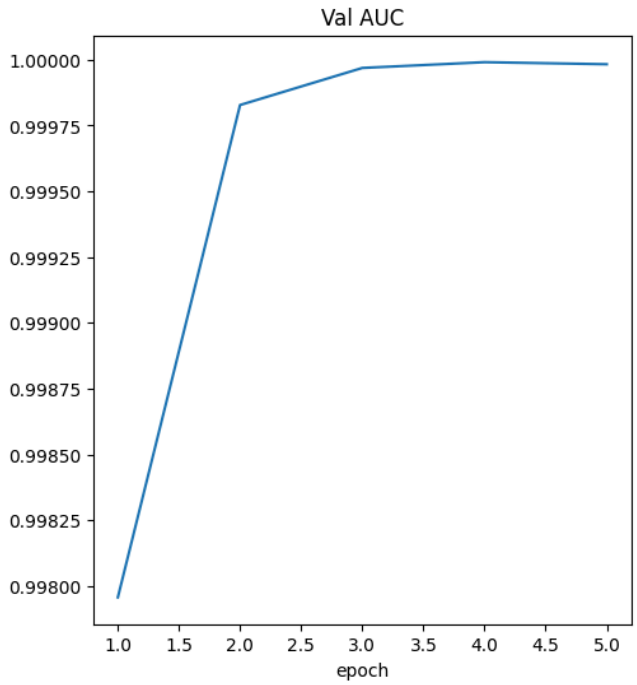
```
Training count: 47164, Validation count: 5895, Test count: 5895
```

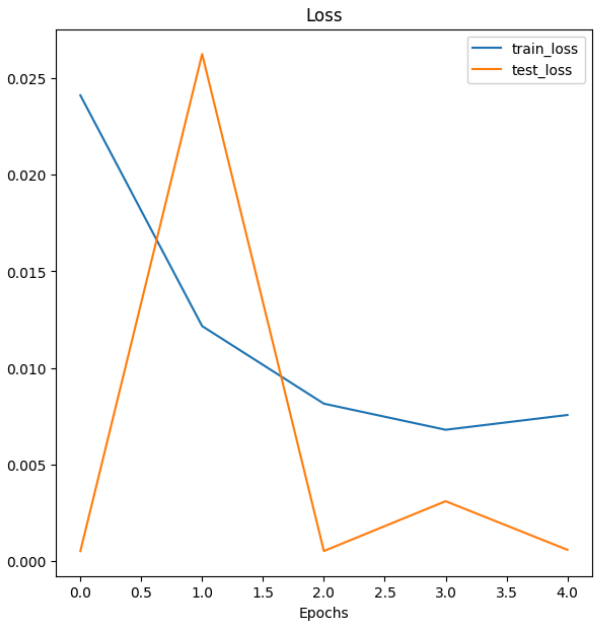
```
# Create directories for train, validation, and test datasets
train_dir = os.path.join(root_dir, "train")
test_dir = os.path.join(root_dir, "test")

# Remove the folders if they already exist
if os.path.exists(train_dir):
    shutil.rmtree(train_dir)
if os.path.exists(test_dir):
    shutil.rmtree(test_dir)

for folder in [train_dir, test_dir]:
    for class_name in class_names:
        os.makedirs(os.path.join(folder, class_name), exist_ok=True)

# Function to copy images to respective directories
def copy_images(indices, target_dir):
    for idx in indices:
        src = image_files_list[idx]
        class_name = class_names[image_class[idx]]
        dst = os.path.join(target_dir, class_name, os.path.basename(src))
        shutil.copy(src, dst)

# Copy images to train, validation, and test directories
copy_images(train_indices, train_dir)
copy_images(val_indices, test_dir)

print(f"Images organized into folders: {train_dir}, {test_dir}")
```

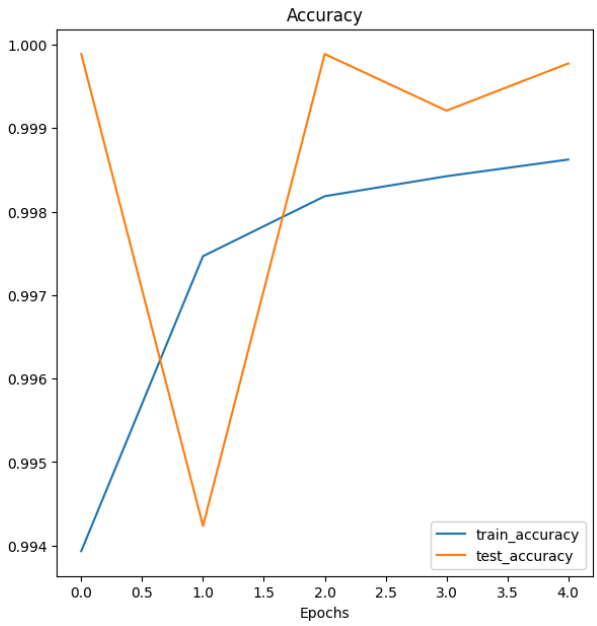
```
Training count: 50111, Test count: 8843
Images organized into folders: data/train, data/test
```

First, when I re-trained using TIMM, I adjusted the logic of dataset partitioning from the original mono ai-based random cut to using TIMM's indexed partitioning method for the three intervals idx[:test_split], idx[test_split:val_split], idx[val_split:] Split on. Since TIMM extracts data from folders, this is how I chose to organize the data. A directory of TRAIN and TEST was created by the code, and based on the previous randomized segmentation, the raw image files were copied one by one into the corresponding directory according to the category.

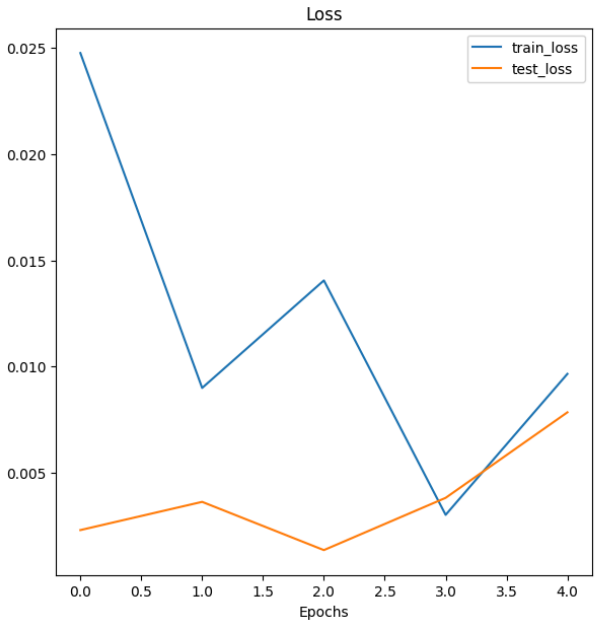


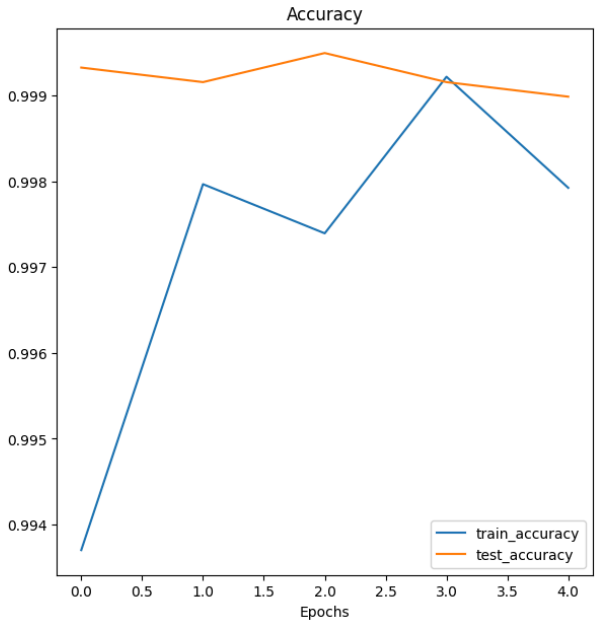


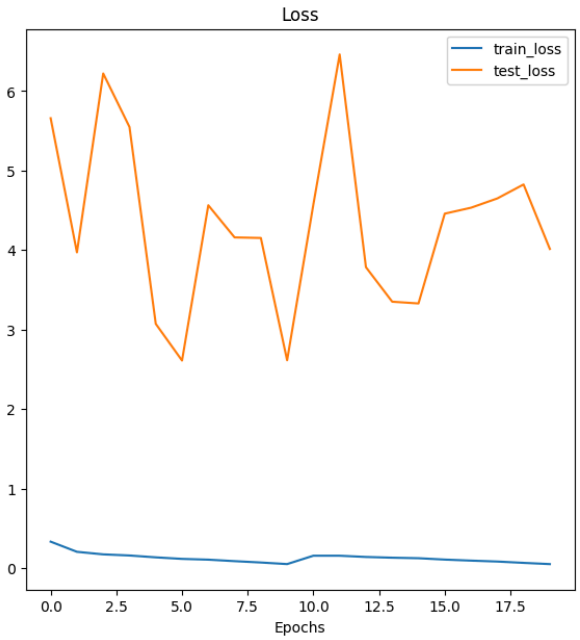


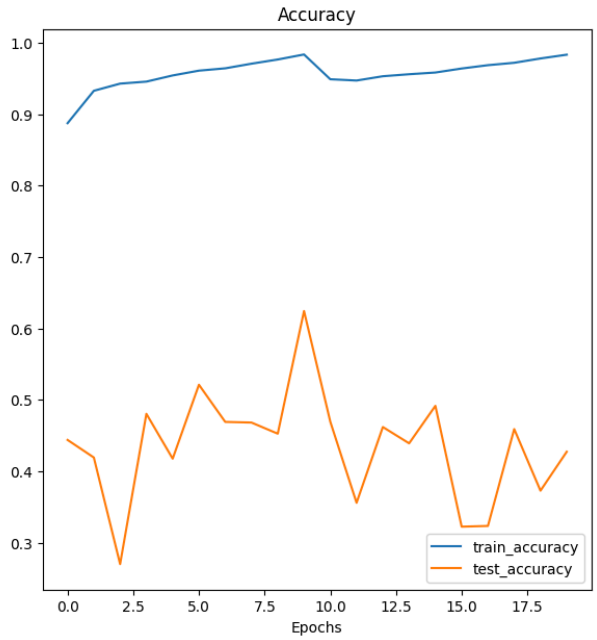


After the basic TIMM model obtained from sample code (CMPE_pytorch8_2024Fall_timm.ipynb) was optimized and tuned and trained, it can be learned from the graphs that the model's loss was greatly reduced, however, the accuracy didn't undergo much difference.

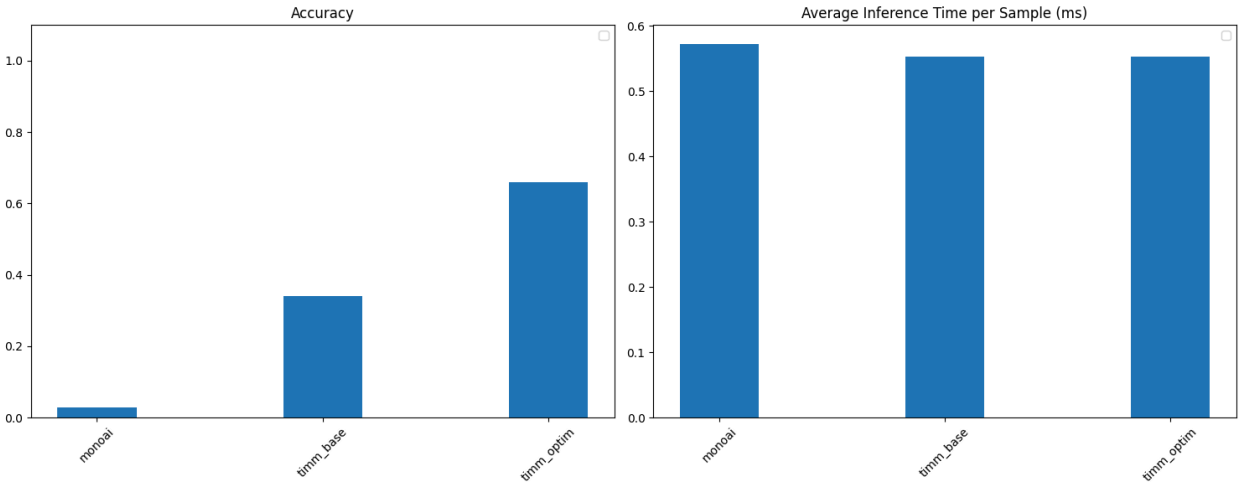








Based on the optimization methods provided in the example code, we added scheduler to the TIMM model as well as retrained it using the AdamP optimizer. This caused a huge increase in training time. However, the final results show that both the loss and accuracy have been improved to some extent.



Out[15]:

	Model	Accuracy	Inference Time (ms)
0	monoai	0.0278	0.57
1	timm_base	0.3411	0.55
2	timm_optim	0.6601	0.55

Next we transformed the output of the ptn models from the three training methods into the generalized onnx format. And we randomly selected some photos from the previous dataset and tested the accuracy of the three models at the same time. It can be seen that there is a significant improvement in the accuracy of the subsequent model compared to the initial model. The average interaction length was also maintained consistent.