Brain Tumor MRI Segmentation

March 5, 2022

Brain Tumor MRI Segmentation

Author: Kuangzheng Zhang and Jingmin Zhao

Youtube Link: https://youtu.be/f9T50cs_Wk0

Background

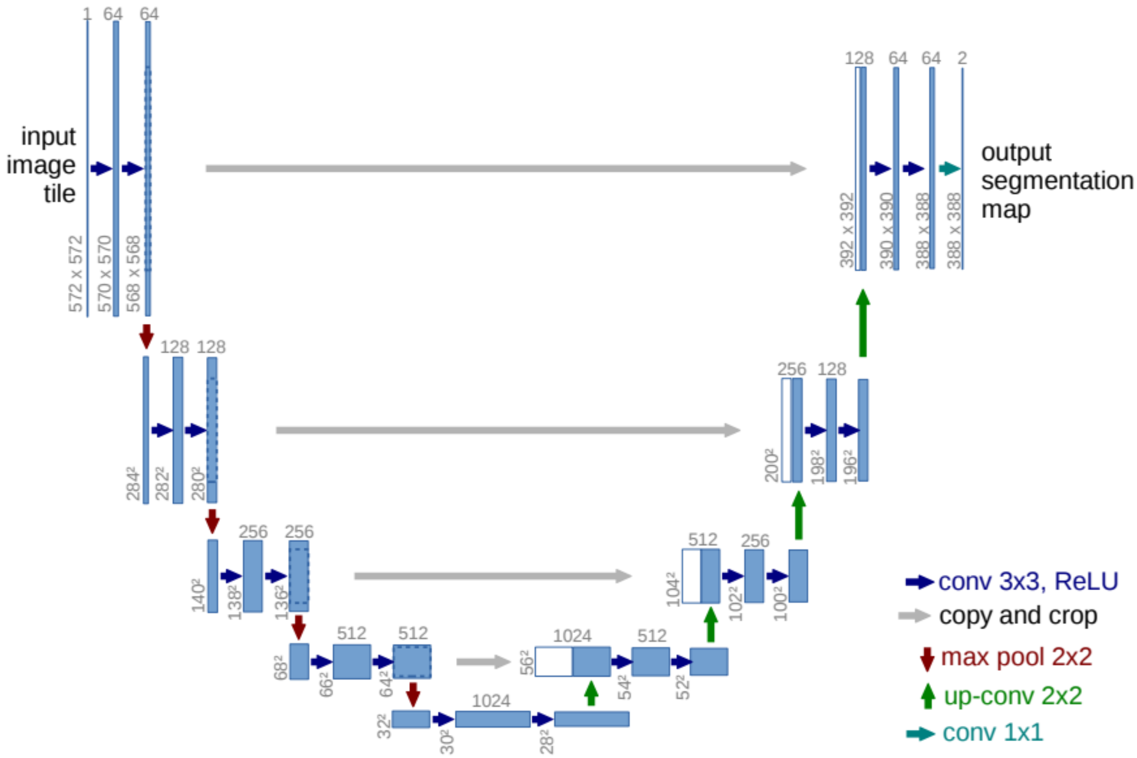
Brain Tumor Detection and Segmentation (BTD, BRATS) is a trending research topic in human brain image analysis, which relies on the DNN-based segmentation and classification frameworks. The image segmentation, damage locating, and the following quantitative assessment on these clinical image data can provide early cancer detection, reliable stage evaluation, customized treatment planning and subsequent monitoring of patients.

In this project, we decided to utilize the DNN framework to conduct accurate acceptable analysis on the multidimensional images, avoiding the need of manual annotation by human experts which could be time-consuming, risky and tedious. More efficient automated extraction and analyzing frameworks which speed up the analyzing of practical medical images have been proposed with novel neural network architecture tested on published datasets. Among these works, U-Net developed by the University of Freiburg is a fully convolutional nerual network designed aiming at precise biomedical image segmentation with fewer training data.

Model

The widely leveraged U-Net was created and proposed at "U-Net: Convolutional Networks for Biomedical Image Segmentation", based on heavy use of data augmentation which can increase the effiency of utilizing the available annotated dataset. As decribed in the published paper, the U-shaped architecture of the U-Net has a contracting path and a symmetric expanding path, in total of 23 convolutional layers. The former contracting path is a typical convolutional network architecture consists of convolutions followed by ReLU and max_pooling operations, while the expansive path adds an uppsampling of the feature map followed by a 2 * 2 convolution which halves the number of feature channels at each step and concatenations with features from the contracting path. The cropping step from the contracting path enables reducing the spatial information while increasing the feature information.

There are many biomedical image analysis and reconstruction applications using the U-Net and its variants to solve practical problems. The figure shown below is from original paper by Olaf Ronneberger, Philipp Fischer, and Thomas Brox.



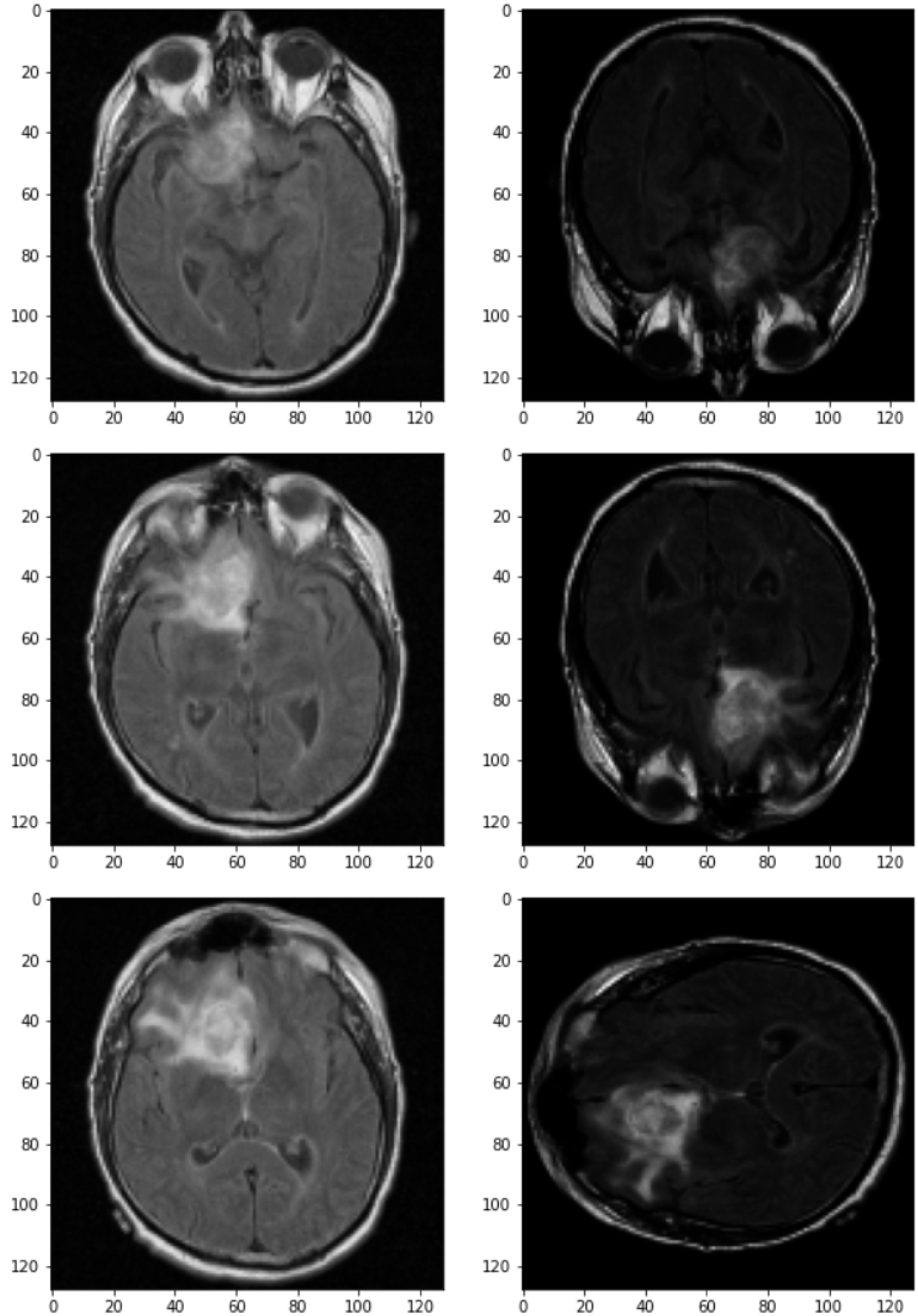
Dataset

In this project, the brain tumor dataset we used comes from Kaggle. It contains medical data for 110 patients. Each patient's directory contains several brain MRI images and their corresponding manual FLAIR abnormality segmentation masks. We remove some MRI that has no mask, which means no tumor detected, because they may affect the performance of our model during training process.

After the above preprocessing, the size of the dataset that we used is 1373, then we divide into train, validation and test dataset by the ratio of 6:2:2, which result in the train size of 823, validation size of 274 and test size of 276.

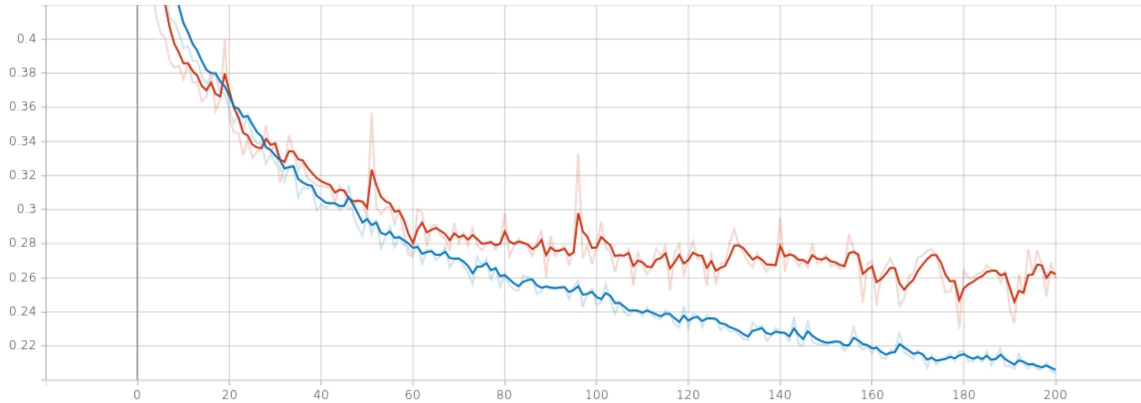
Although MRI can identify lesions accurately by clearly showing the location, scope and relationship with surrounding tissues and organs, there are many factors that affect imaging, thus creating many artifacts. The resolution is typically 256*256. Taken into account the challenges we have, we apply many transforms like resize, adjust contrast, random rotate and random flip to augment the existing dataset to highlight the target.

[ ]:
```
self.train_transform = transforms.Compose([
    transforms.LoadImaged(keys = ["img", "seg"]),
    transforms.AddChanneld(keys = ["img", "seg"]),
    transforms.ScaleIntensityd(keys=["img", "seg"]),
    transforms.Resized(spatial_size  = (128, 128), keys = ["img", "seg"]),
    transforms.AdjustContrastd(keys = "img", gamma = 2.5),
    transforms.RandRotate90d(keys=["img", "seg"], prob=0.5),
    transforms.RandFlipd(keys=["img", "seg"], prob=0.5)
])
```

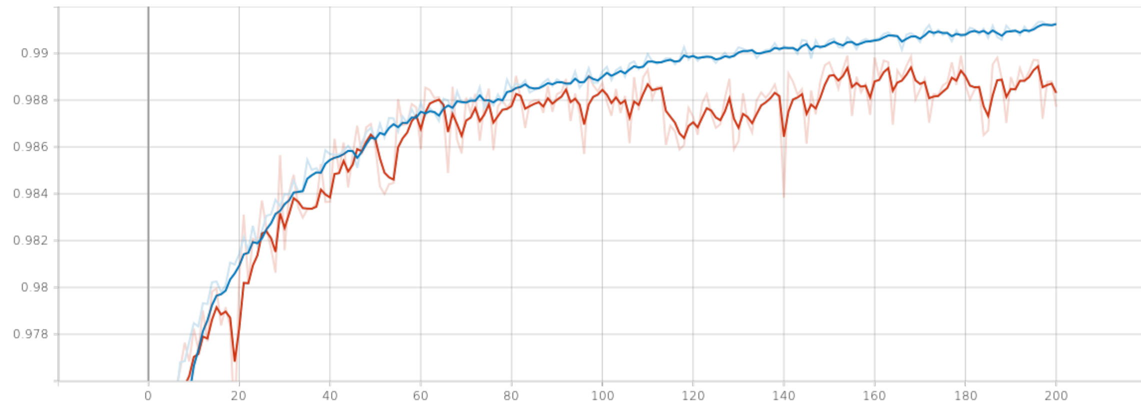

0 20 40 60 80 100 120

Result

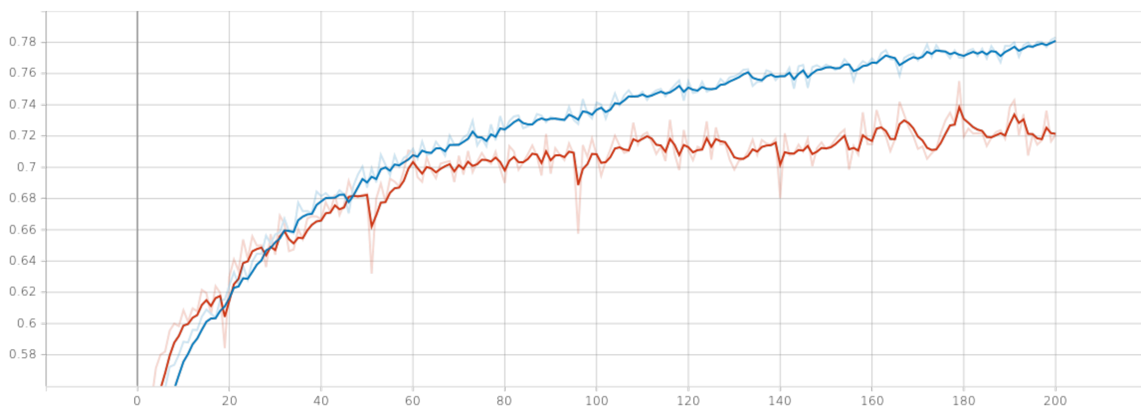
Loss



Accuracy



F1 Score



We train for 200 epochs with learning rate 0.001. The train loss converges to around 0.2 and validation loss to around 0.26. The train accuracy converges to around 99.1% and validation accuray to around 98.9%. The train F1 Score converges to around 0.78 and validation F1 Score to around 0.72.

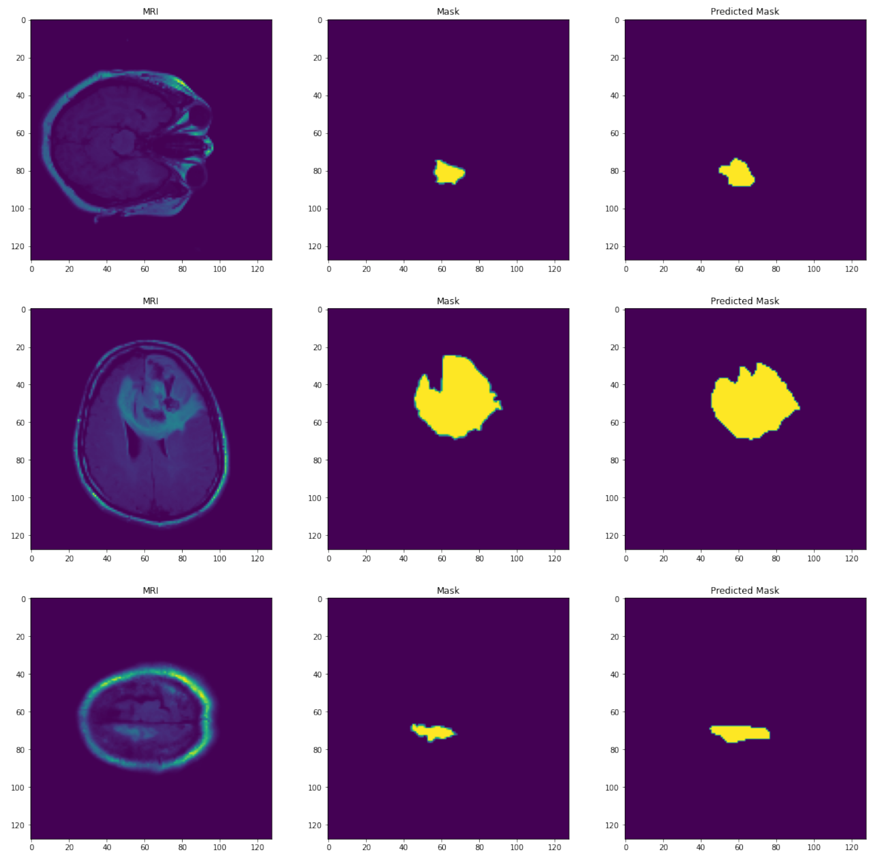
Then we test our trained model on the test dataset. Test loss achieves 0.227110, accuracy achieves 99.1% and F1 Score achieves 0.752088.

Below are three samples from the test dataset. The first column is the MRI image, the second column is the ground truth segmentation mask, the third column is the predict segmentation mask of our trained model. The prediction is very close to the ground truth, which means out model achieves great performance!

[ ]:
```
sample_num = 3
fig_num = 3
```

```python
fig, ax = plt.subplots(sample_num, fig_num, figsize = (20, 20))
for i, data in enumerate(trainer.test_dataloader):
    if i >= sample_num:
        break
    X, y = data["img"].to(trainer.device), data["seg"].to(trainer.device)
    trainer.model.eval()
    with torch.no_grad():
        pred = trainer.model(X)
        y, pred = y.int().view(-1).cpu().numpy(), (pred > .5).int().view(-1).cpu().numpy()
        pred = trainer.model(X)[0].detach().cpu().numpy()
        pred = pred.transpose((1, 2, 0)).squeeze()

    ax[i, 0].imshow(data["img"].squeeze())
    ax[i, 0].title.set_text('MRI')
    ax[i, 1].imshow(data["seg"].squeeze())
    ax[i, 1].title.set_text('Mask')
    ax[i, 2].imshow(pred > .5)
    ax[i, 2].title.set_text('Predicted Mask')
```



Appendix

Preparation

[ ]:
```python
import os
from google.colab import drive
drive.mount('/content/drive')
os.chdir("/content/drive/MyDrive/EE 435/Project2")
!ls
```

```
!pip install monai
!pip install imio
from IPython.display import clear_output
clear_output()
```

Import Libraries

```
[ ]: import os
import shutil
import itertools
import copy
import heapq
import skimage
import numpy as np
import pandas as pd
from glob import glob
import matplotlib.pyplot as plt
import datetime
from imio import load, save
from tqdm import tqdm
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,␣
↪classification_report, confusion_matrix

import monai
from monai import transforms
from monai.data import Dataset, DataLoader


import torch
from torch.optim import Adam
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import random_split

import tensorflow as tf
%load_ext tensorboard
```

Config

```
[ ]: class Config:
    def __init__(self, **kwargs):
        for key, value in kwargs.items():
            setattr(self, key, value)

config = Config(
    ORG_DATASET_DIR = './Dataset/lgg-mri-segmentation/kaggle_3m/',
    CONVERTED_DATASET_DIR = './Dataset/converted_data/',
    LOG_DIR = './Logs/tensorboard',
    TEMP_DATASET_PATH = './Temp/data.pkl',

    version = 'v4',
    split_ratio = '6:2:2',
    seed = 42,
    lr = 0.001,
    weight_decay = 0,
    epochs = 200,
    save_model = True,
    batch_size = 1,
```

```
    log_interval = 300,
    shuffle = True,
    cuda = True if torch.cuda.is_available() else False,
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
)
```

Define Trainer class

[ ]:
```
class Trainer:
    def __init__(self, config):
        self.ORG_DATASET_DIR = config.ORG_DATASET_DIR
        self.CONVERTED_DATASET_DIR = config.CONVERTED_DATASET_DIR
        self.LOG_DIR = config.LOG_DIR
        self.TEMP_DATASET_PATH = config.TEMP_DATASET_PATH

        self.version = config.version
        self.split_ratio = config.split_ratio
        self.seed = config.seed
        self.lr = config.lr
        self.weight_decay = config.weight_decay
        self.epochs = config.epochs
        self.save_model = config.save_model
        self.batch_size = config.batch_size
        self.log_interval = config.log_interval
        self.shuffle = config.shuffle
        self.cuda = config.cuda
        self.device = config.device

        self.globaliter = 0
        self.best_train_f1 = 0

        torch.manual_seed(self.seed)

        # Set up tensorboard
        train_log_dir = os.path.join(config.LOG_DIR, config.version, 'train')
        val_log_dir = os.path.join(config.LOG_DIR, config.version, 'val')
        test_log_dir = os.path.join(config.LOG_DIR, config.version, 'test')

        if os.path.exists(os.path.join(config.LOG_DIR, config.version)):
            shutil.rmtree(os.path.join(config.LOG_DIR, config.version))

        self.train_summary_writer = tf.summary.create_file_writer(train_log_dir)
        self.val_summary_writer = tf.summary.create_file_writer(val_log_dir)
        self.test_summary_writer = tf.summary.create_file_writer(test_log_dir)

        # Data Augmentation
        self.transform()

        # Prepare DataLoader
        self.train_dataloader, self.val_dataloader, self.test_dataloader = self.load_dataset()

        self.model = monai.networks.nets.UNet(
            spatial_dims = 2,
            channels = (16, 32, 64, 128, 256, 512),
            in_channels = 1,
            out_channels = 1,
            strides = (2, 2, 2, 2, 2),
            num_res_units = 3,
            dropout = 0.2
```

```
            ).to(config.device)
        self.loss_fn = monai.losses.DiceLoss(sigmoid = True)
        self.optimizer = Adam(self.model.parameters(), lr=self.lr, weight_decay=self.weight_decay)

    def load_dataset(self):
        # Load dataset
        self.img_dir = os.path.join(self.CONVERTED_DATASET_DIR, "img")
        self.seg_dir = os.path.join(self.CONVERTED_DATASET_DIR, "seg")

        imgs = sorted(glob(os.path.join(self.img_dir, '*')))
        segs = sorted(glob(os.path.join(self.seg_dir, '*')))
        assert len(imgs) == len(segs)
        self.size = len(imgs)
        self.split_ratio = list(map(int, self.split_ratio.split(':')))
        self.train_size = int(self.size * self.split_ratio[0] / sum(self.split_ratio))
        self.val_size = int(self.size * self.split_ratio[1] / sum(self.split_ratio))
        self.test_size = self.size - self.train_size - self.val_size
        print(f"Size: {self.size}  Train Size: {self.train_size}  Val Size: {self.val_size}  Test Size: {self.test_size}")

        train_files = [{"img": img, "seg": seg} for img, seg in zip(imgs[:self.train_size], segs[:self.train_size])]
        val_files = [{"img": img, "seg": seg} for img, seg in zip(imgs[self.train_size:-self.test_size], segs[self.train_size:-self.test_size])]
        test_files = [{"img": img, "seg": seg} for img, seg in zip(imgs[-self.test_size:], segs[-self.test_size:])]

        train_ds = Dataset(data = train_files, transform = self.train_transform)
        val_ds = Dataset(data = val_files, transform = self.val_transform)
        test_ds = Dataset(data = test_files, transform = self.test_transform)

        train_dataloader = DataLoader(train_ds, batch_size=1, shuffle=self.shuffle)
        val_dataloader = DataLoader(val_ds, batch_size=1, shuffle=self.shuffle)
        test_dataloader = DataLoader(test_ds, batch_size=1, shuffle=self.shuffle)
        return train_dataloader, val_dataloader, test_dataloader

    def transform(self):
        self.train_transform = transforms.Compose([
            transforms.LoadImaged(keys = ["img", "seg"]),
            transforms.AddChanneld(keys = ["img", "seg"]),
            transforms.ScaleIntensityd(keys=["img", "seg"]),
            transforms.Resized(spatial_size  = (128, 128), keys = ["img", "seg"]),
            transforms.AdjustContrastd(keys = "img", gamma = 2.5),
            transforms.RandRotate90d(keys=["img", "seg"], prob=0.5),
            transforms.RandFlipd(keys=["img", "seg"], prob=0.5)
        ])

        self.val_transform = transforms.Compose([
            transforms.LoadImaged(keys = ["img", "seg"]),
            transforms.AddChanneld(keys = ["img", "seg"]),
            transforms.ScaleIntensityd(keys=["img", "seg"]),
            transforms.Resized(spatial_size  = (128, 128), keys = ["img", "seg"]),
            transforms.AdjustContrastd(keys = "img", gamma = 2.5),
            transforms.RandRotate90d(keys=["img", "seg"], prob=0.5),
            transforms.RandFlipd(keys=["img", "seg"], prob=0.5)
        ])

        self.test_transform = self.val_transform
```

```
    def train(self, epoch, dataloader, model, loss_fn, optimizer):
        size = len(dataloader.dataset)
        num_batches = len(dataloader)
        model.train()
        train_loss, train_acc, train_f1 = 0, 0, 0
        self.train_preds = []
        self.train_labels = []
        for batch, data in enumerate(tqdm(dataloader)):
            X, y = data["img"].to(self.device), data["seg"].to(self.device)

            # Compute prediction error
            pred = model(X)
            loss = loss_fn(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Loss
            cur_train_loss = loss.item()
            train_loss += cur_train_loss

            # Accuracy
            y, pred = y.int().view(-1).cpu().numpy(), (pred > .5).int().view(-1).cpu().numpy()
            cur_train_acc = accuracy_score(y, pred)
            train_acc += cur_train_acc

            # F1
            cur_train_f1 = f1_score(y, pred)
            train_f1 += cur_train_f1

        train_loss /= num_batches
        train_acc /= num_batches
        train_f1 /= num_batches
        print(f"Train Epoch: {epoch}  Avg Train Loss: {train_loss:>7f}  Avg Train Acc:␣
↪{(100*train_acc):>0.1f}%  Avg Train F1: {train_f1:>7f}")
        if train_f1 > self.best_train_f1:
            self.best_train_f1 = train_f1
            torch.save(self.model.state_dict(), f"./Models/best-model-parameters-{self.version}.pt")
            print("Best Model Updated!")
        with self.train_summary_writer.as_default():
            tf.summary.scalar('loss', train_loss, step = epoch)
            tf.summary.scalar('acc', train_acc, step = epoch)
            tf.summary.scalar('f1', train_f1, step = epoch)

    def val(self, epoch, dataloader, model, loss_fn):
        size = len(dataloader.dataset)
        num_batches = len(dataloader)
        model.eval()
        val_loss, val_acc, val_f1 = 0, 0, 0
        with torch.no_grad():
            for batch, data in enumerate(tqdm(dataloader)):
                X, y = data["img"].to(self.device), data["seg"].to(self.device)
                pred = model(X)
                cur_val_loss = loss_fn(pred, y).item()
                val_loss += cur_val_loss
                y, pred = y.int().view(-1).cpu().numpy(), (pred > .5).int().view(-1).cpu().numpy()
                cur_val_acc = accuracy_score(y, pred)
```

```
                val_acc += cur_val_acc
                cur_val_f1 = f1_score(y, pred)
                val_f1 += cur_val_f1
        val_loss /= num_batches
        val_acc /= num_batches
        val_f1 /= num_batches
        print(f"Val Epoch: {epoch}  Avg Val Loss: {val_loss:>7f}  Avg Val Acc: {(100*val_acc):>0.
↪1f}%  Avg Val F1: {val_f1:>7f}")
        with self.val_summary_writer.as_default():
            tf.summary.scalar('loss', val_loss, step = epoch)
            tf.summary.scalar('acc', val_acc, step = epoch)
            tf.summary.scalar('f1', val_f1, step = epoch)

    def test(self, epoch, dataloader, model, loss_fn):
        size = len(dataloader.dataset)
        num_batches = len(dataloader)
        model.eval()
        test_loss, test_acc, test_f1 = 0, 0, 0
        with torch.no_grad():
            for batch, data in enumerate(tqdm(dataloader)):
                X, y = data["img"].to(self.device), data["seg"].to(self.device)
                pred = model(X)
                cur_test_loss = loss_fn(pred, y).item()
                test_loss += cur_test_loss
                y, pred = y.int().view(-1).cpu().numpy(), (pred > .5).int().view(-1).cpu().numpy()
                cur_test_acc = accuracy_score(y, pred)
                test_acc += cur_test_acc
                cur_test_f1 = f1_score(y, pred)
                test_f1 += cur_test_f1
        test_loss /= num_batches
        test_acc /= num_batches
        test_f1 /= num_batches
        print(f"Avg Test Loss: {test_loss:>7f}  Avg Test Acc: {(100*test_acc):>0.1f}%  Avg Test F1:␣
↪{test_f1:>7f}")
        with self.test_summary_writer.as_default():
            tf.summary.scalar('loss', test_loss, step = epoch)
            tf.summary.scalar('acc', test_acc, step = epoch)
            tf.summary.scalar('f1', test_f1, step = epoch)
```

Train and Validate

[ ]:
```
trainer = Trainer(config)
%tensorboard --logdir Logs/tensorboard/{trainer.version}
```

```
Output hidden; open in https://colab.research.google.com to view.
```

[ ]:
```
# Train and Validate
for epoch in range(1, trainer.epochs + 1):
    print(f"Epoch {epoch}\n-------------------------------")
    trainer.train(epoch, trainer.train_dataloader, trainer.model, trainer.loss_fn, trainer.optimizer)
    trainer.val(epoch, trainer.val_dataloader, trainer.model, trainer.loss_fn)

# Test
print("\nTest\n-------------------------------")
trainer.test(epoch, trainer.test_dataloader, trainer.model, trainer.loss_fn)
```

```
Epoch 1
-------------------------------

100%|     | 823/823 [05:31<00:00,  2.48it/s]
```

```
Train Epoch: 1  Avg Train Loss: 0.824555  Avg Train Acc: 86.0%  Avg Train F1:
0.269921
Best Model Updated!

100%|      | 274/274 [01:35<00:00,  2.86it/s]

Val Epoch: 1  Avg Val Loss: 0.663986  Avg Val Acc: 94.2%  Avg Val F1: 0.419618
Epoch 2
-------------------------------

100%|      | 823/823 [00:41<00:00, 19.78it/s]

Train Epoch: 2  Avg Train Loss: 0.600156  Avg Train Acc: 94.9%  Avg Train F1:
0.430969
Best Model Updated!

100%|      | 274/274 [00:07<00:00, 36.07it/s]

Val Epoch: 2  Avg Val Loss: 0.481279  Avg Val Acc: 96.3%  Avg Val F1: 0.516696
Epoch 3
-------------------------------

100%|      | 823/823 [00:41<00:00, 19.77it/s]

Train Epoch: 3  Avg Train Loss: 0.498888  Avg Train Acc: 96.7%  Avg Train F1:
0.503211
Best Model Updated!

100%|      | 274/274 [00:07<00:00, 36.10it/s]

Val Epoch: 3  Avg Val Loss: 0.445102  Avg Val Acc: 97.2%  Avg Val F1: 0.549908
```

……

Epoch 198

```
-------------------------------

100%|      | 823/823 [00:43<00:00, 19.12it/s]

Train Epoch: 198  Avg Train Loss: 0.210065  Avg Train Acc: 99.1%  Avg Train F1:
0.776910

100%|      | 274/274 [00:07<00:00, 35.33it/s]

Val Epoch: 198  Avg Val Loss: 0.248945  Avg Val Acc: 98.9%  Avg Val F1: 0.736306
Epoch 199
-------------------------------

100%|      | 823/823 [00:42<00:00, 19.34it/s]

Train Epoch: 199  Avg Train Loss: 0.205322  Avg Train Acc: 99.1%  Avg Train F1:
0.781502
Best Model Updated!

100%|      | 274/274 [00:07<00:00, 34.85it/s]

Val Epoch: 199  Avg Val Loss: 0.269047  Avg Val Acc: 98.9%  Avg Val F1: 0.716599
Epoch 200
-------------------------------

100%|      | 823/823 [00:42<00:00, 19.20it/s]

Train Epoch: 200  Avg Train Loss: 0.203892  Avg Train Acc: 99.1%  Avg Train F1:
0.783148
Best Model Updated!

100%|      | 274/274 [00:07<00:00, 34.64it/s]

Val Epoch: 200  Avg Val Loss: 0.259756  Avg Val Acc: 98.8%  Avg Val F1: 0.720901
```

```
Test
-------------------------------

100%|          | 276/276 [01:36<00:00,  2.86it/s]

Avg Test Loss: 0.227110  Avg Test Acc: 99.1%  Avg Test F1: 0.752088
```

Load Best Model to Test

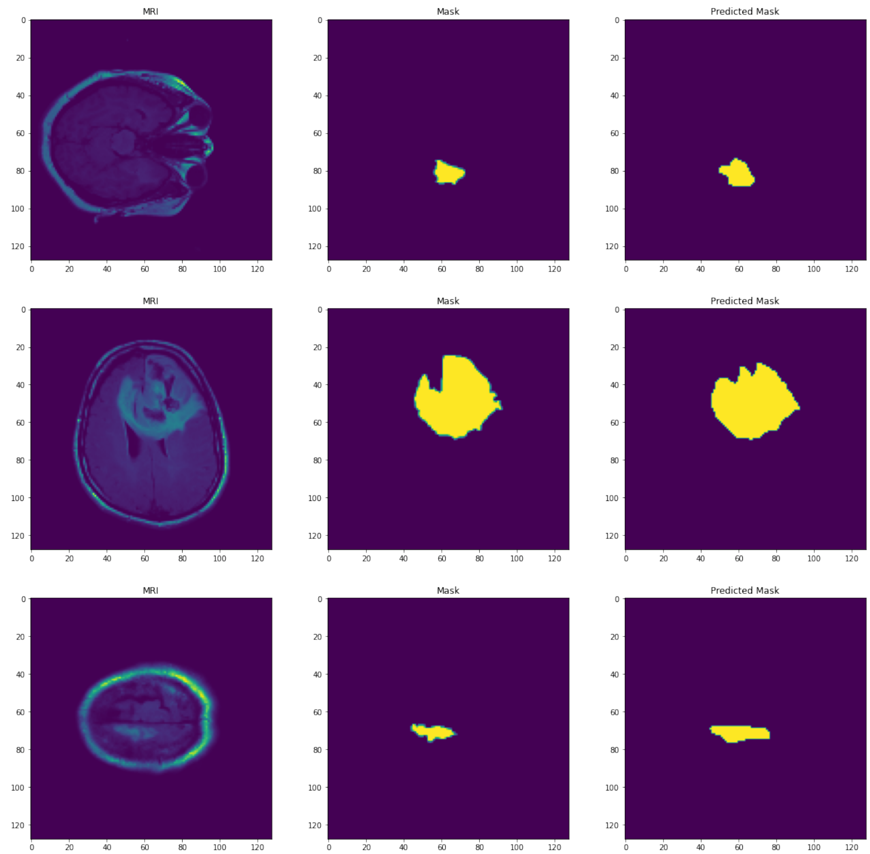
[ ]:
```
trainer = Trainer(config)
trainer.model.load_state_dict(torch.load(f"./Models/best-model-parameters-{trainer.version}.pt"))
```

```
Size: 1373  Train Size: 823  Val Size: 274  Test Size: 276
```

[ ]:
```
<All keys matched successfully>
```

[ ]:
```
sample_num = 3
fig_num = 3
fig, ax = plt.subplots(sample_num, fig_num, figsize = (20, 20))
for i, data in enumerate(trainer.test_dataloader):
    if i >= sample_num:
        break
    X, y = data["img"].to(trainer.device), data["seg"].to(trainer.device)
    trainer.model.eval()
    with torch.no_grad():
        pred = trainer.model(X)
        y, pred = y.int().view(-1).cpu().numpy(), (pred > .5).int().view(-1).cpu().numpy()
        pred = trainer.model(X)[0].detach().cpu().numpy()
        pred = pred.transpose((1, 2, 0)).squeeze()

    ax[i, 0].imshow(data["img"].squeeze())
    ax[i, 0].title.set_text('MRI')
    ax[i, 1].imshow(data["seg"].squeeze())
    ax[i, 1].title.set_text('Mask')
    ax[i, 2].imshow(pred > .5)
    ax[i, 2].title.set_text('Predicted Mask')
```


MRI
Mask
Predicted Mask
MRI
Mask
Predicted Mask
MRI
Mask
Predicted Mask