Amount Drive and Import Libraries

```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')


!pip install torch
!pip3 install torchvision
!pip3 install opencv-python

# import libraries
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
import sys

import torch
from PIL import Image

import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

Change Path Here

Path to the directory of train set and test set and also the path where to store trained model

```
train_input_dir = '/content/drive/My Drive/csc420/cat_data/Train/input/'
train_mask_dir = '/content/drive/My Drive/csc420/cat_data/Train/mask/'

test_input_dir = '/content/drive/My Drive/csc420/cat_data/Test/input/'
test_mask_dir = '/content/drive/My Drive/csc420/cat_data/Test/mask/'

default_saved_path = '/content/drive/My Drive/csc420/unet.model'
saved_unet_dice_path = '/content/drive/My Drive/csc420/unet_dice.model'
saved_unet_bce_path = '/content/drive/My Drive/csc420/unet_bce.model'
saved_unet_transfer_path = '/content/drive/My Drive/csc420/unet_transfer.model'
saved_unet_fine_tuning_path = '/content/drive/My Drive/csc420/unet_transfer_final.model'
```

Load Data for Training and Testing

Load training set and test set images and masks.

Customized Dataset

```
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import transforms

# Define customized dataset
class CatDataset(Dataset):
    def __init__(self, img_dir, mask_dir, transform=None):
        img_ids = [int(f.split('.')[-2]) for f in os.listdir(img_dir)]

        first_img = os.listdir(img_dir)[0]
        first_mask = os.listdir(mask_dir)[0]
        img_name = first_img.split('.')[0]
        img_extension = first_img.split('.')[-1]
        mask_name = first_mask.split('.')[0]
        mask_extension = first_mask.split('.')[-1]
        self.img_files = []
        self.mask_files = []
        self.transform = transform

        # add corresponding image and mask path into list for such id
        for id in img_ids:
            self.img_files.append(img_dir+img_name+'.'+str(id)+'.'+img_extension)
            self.mask_files.append(mask_dir+mask_name+'.'+str(id)+'.'+mask_extension)


    def __getitem__(self, index):
        img_path = self.img_files[index]
        mask_path = self.mask_files[index]

        img = Image.open(img_path)
        mask = Image.open(mask_path)
        if self.transform is not None:
            img = self.transform(img)
            mask = self.transform(mask)

        return img, mask

    def __len__(self):
        return len(self.img_files)
```

Load Data from Given Path

```
# load training input images and mask
img_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])
```

```
train_dataset = CatDataset(train_input_dir, train_mask_dir, transform=img_transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers=0

inputs, masks = next(iter(train_loader))
print(len(train_loader))
print(inputs.shape, masks.shape)

# load test input images and mask
test_dataset = CatDataset(test_input_dir, test_mask_dir, transform=img_transform)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=4, shuffle=True, num_workers=0)

inputs, masks = next(iter(test_loader))
print(len(test_loader))
print(inputs.shape, masks.shape)
```

```
15
torch.Size([4, 3, 128, 128]) torch.Size([4, 1, 128, 128])
6
torch.Size([4, 3, 128, 128]) torch.Size([4, 1, 128, 128])
```

Building U-Net

```
from torch import nn, optim
import torch.nn.functional as F

# class DoubleConv(nn.Module):
#     def __init__(self, in_channel, out_channel):
#         super(DoubleConv, self).__init__()
#         self.double_conv = nn.Sequential(
#             nn.Conv2d(in_channel, out_channel, kernel_size=3, padding=1),
#             nn.BatchNorm2d(out_channel),
#             nn.ReLU(),
#             nn.Conv2d(out_channel, out_channel, kernel_size=3, padding=1),
#             nn.BatchNorm2d(out_channel),
#             nn.ReLU())

#     def forward(self,x):
#         return self.double_conv(x)


# class UpSample(nn.Module):
#     def __init__(self, in_channel, out_channel):
#         super(UpSample, self).__init__()
#         self.up_sample = nn.Sequential(
#             nn.ConvTranspose2d(in_channel, out_channel, 2, stride=2, padding=0),
#             nn.ReLU())

#     def forward(self,x):
#         return self.up_sample(x)

def double_conv(in_channel, out_channel):
```

```
  return nn.Sequential(
            nn.Conv2d(in_channel, out_channel, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(),
            nn.Conv2d(out_channel, out_channel, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channel),
            nn.ReLU())

def up_sample(in_channel, out_channel):
  return nn.Sequential(
            nn.ConvTranspose2d(in_channel, out_channel, 2, stride=2, padding=0),
            nn.ReLU())


class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()
        # self.up_sample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.down_layer1 = double_conv(3, 64)
        self.down_layer2 = nn.Sequential(
            nn.MaxPool2d(2),
            double_conv(64, 128)
        )
        self.down_layer3 = nn.Sequential(
            nn.MaxPool2d(2),
            double_conv(128, 256)
        )
        self.down_layer4 = nn.Sequential(
            nn.MaxPool2d(2),
            double_conv(256, 512)
        )
        self.bottleneck = nn.Sequential(
            nn.MaxPool2d(2),
            double_conv(512, 1024)
        )
        self.up_sample1 = up_sample(1024, 512)
        self.up_sample2 = up_sample(512, 256)
        self.up_sample3 = up_sample(256, 128)
        self.up_sample4 = up_sample(128, 64)
        self.up_layer1 = double_conv(1024, 512)
        self.up_layer2 = double_conv(512, 256)
        self.up_layer3 = double_conv(256, 128)
        self.up_layer4 = nn.Sequential(
            double_conv(128, 64),
            nn.Conv2d(64, 1, kernel_size=1, padding=0)
        )

    def forward(self, img):
        # Contracting/downsampling path
        down_1 = self.down_layer1(img)
        down_2 = self.down_layer2(down_1)
        down_3 = self.down_layer3(down_2)
        down_4 = self.down_layer4(down_3)

        bottleneck = self.bottleneck(down_4)
```

```
        bottleneck = self.bottleneck(down_4)
        bottleneck_upsample = self.up_sample1(bottleneck)
        bottleneck_upsample = torch.cat((bottleneck_upsample, down_4), dim=1)

        # Expanding/upsampling path
        up_1 = self.up_layer1(bottleneck_upsample)
        up_1_upsample = self.up_sample2(up_1)
        up_1_upsample = torch.cat((up_1_upsample, down_3), dim=1)

        up_2 = self.up_layer2(up_1_upsample)
        up_2_upsample = self.up_sample3(up_2)
        up_2_upsample = torch.cat((up_2_upsample, down_2), dim=1)

        up_3 = self.up_layer3(up_2_upsample)
        up_3_upsample = self.up_sample4(up_3)
        up_3_upsample = torch.cat((up_3_upsample, down_1), dim=1)

        out = self.up_layer4(up_3_upsample)
        out = nn.Sigmoid()(out)
        return out

# build a u-net
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
model = UNet().to(device)

from torchsummary import summary
summary(model, (3, 128, 128))
```

```
cuda
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 128, 128]           1,792
       BatchNorm2d-2         [-1, 64, 128, 128]             128
              ReLU-3         [-1, 64, 128, 128]               0
            Conv2d-4         [-1, 64, 128, 128]          36,928
       BatchNorm2d-5         [-1, 64, 128, 128]             128
              ReLU-6         [-1, 64, 128, 128]               0
         MaxPool2d-7           [-1, 64, 64, 64]               0
            Conv2d-8          [-1, 128, 64, 64]          73,856
       BatchNorm2d-9          [-1, 128, 64, 64]             256
             ReLU-10          [-1, 128, 64, 64]               0
           Conv2d-11          [-1, 128, 64, 64]         147,584
      BatchNorm2d-12          [-1, 128, 64, 64]             256
             ReLU-13          [-1, 128, 64, 64]               0
        MaxPool2d-14          [-1, 128, 32, 32]               0
           Conv2d-15          [-1, 256, 32, 32]         295,168
      BatchNorm2d-16          [-1, 256, 32, 32]             512
             ReLU-17          [-1, 256, 32, 32]               0
           Conv2d-18          [-1, 256, 32, 32]         590,080
      BatchNorm2d-19          [-1, 256, 32, 32]             512
             ReLU-20          [-1, 256, 32, 32]               0
        MaxPool2d-21          [-1, 256, 16, 16]               0
           Conv2d-22          [-1, 512, 16, 16]       1,180,160
      BatchNorm2d-23          [-1, 512, 16, 16]           1,024
             ReLU-24          [-1, 512, 16, 16]               0
           Conv2d-25          [-1, 512, 16, 16]       2,359,808
      BatchNorm2d-26          [-1, 512, 16, 16]           1,024
             ReLU-27          [-1, 512, 16, 16]               0
        MaxPool2d-28            [-1, 512, 8, 8]               0
           Conv2d-29           [-1, 1024, 8, 8]       4,719,616
      BatchNorm2d-30           [-1, 1024, 8, 8]           2,048
             ReLU-31           [-1, 1024, 8, 8]               0
           Conv2d-32           [-1, 1024, 8, 8]       9,438,208
      BatchNorm2d-33           [-1, 1024, 8, 8]           2,048
             ReLU-34           [-1, 1024, 8, 8]               0
  ConvTranspose2d-35          [-1, 512, 16, 16]       2,097,664
             ReLU-36          [-1, 512, 16, 16]               0
           Conv2d-37          [-1, 512, 16, 16]       4,719,104
      BatchNorm2d-38          [-1, 512, 16, 16]           1,024
             ReLU-39          [-1, 512, 16, 16]               0
           Conv2d-40          [-1, 512, 16, 16]       2,359,808
      BatchNorm2d-41          [-1, 512, 16, 16]           1,024
             ReLU-42          [-1, 512, 16, 16]               0
  ConvTranspose2d-43          [-1, 256, 32, 32]         524,544
             ReLU-44          [-1, 256, 32, 32]               0
           Conv2d-45          [-1, 256, 32, 32]       1,179,904
      BatchNorm2d-46          [-1, 256, 32, 32]             512
             ReLU-47          [-1, 256, 32, 32]               0
           Conv2d-48          [-1, 256, 32, 32]         590,080
      BatchNorm2d-49          [-1, 256, 32, 32]             512
             ReLU-50          [-1, 256, 32, 32]               0
  ConvTranspose2d-51          [-1, 128, 64, 64]         131,200
             ReLU-52          [-1, 128, 64, 64]               0
           Conv2d-53          [-1, 128, 64, 64]         295,040
```

```
       BatchNorm2d-54          [-1, 128, 64, 64]             256
              ReLU-55          [-1, 128, 64, 64]               0
            Conv2d-56          [-1, 128, 64, 64]         147,584
       BatchNorm2d-57          [-1, 128, 64, 64]             256
              ReLU-58          [-1, 128, 64, 64]               0
   ConvTranspose2d-59         [-1, 64, 128, 128]          32,832
              ReLU-60         [-1, 64, 128, 128]               0
            Conv2d-61         [-1, 64, 128, 128]          73,792
       BatchNorm2d-62         [-1, 64, 128, 128]             128
              ReLU-63         [-1, 64, 128, 128]               0
            Conv2d-64         [-1, 64, 128, 128]          36,928
       BatchNorm2d-65         [-1, 64, 128, 128]             128
              ReLU-66         [-1, 64, 128, 128]               0
            Conv2d-67          [-1, 1, 128, 128]              65
================================================================
Total params: 31,043,521
Trainable params: 31,043,521
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 216.88
Params size (MB): 118.42
Estimated Total Size (MB): 335.48
----------------------------------------------------------------
```

Training U-Net

With self-implemented loss function:

- Binary Cross Entropy
- Sørensen–Dice-coefficient

Define Two Loss Functions and Train Function

```
# loss functions
def cross_entropy_loss(predicts, targets):
    # pixel-wise cross entropy
    norm_pred = predicts / predicts.max()
    # loss = -(ylog(p)+(1-y)log(1-p)) for a single predict with 2 class
    # computed loss as the average of all cross-entropies in the sample
    loss = -torch.mean((targets)*torch.log(predicts) + (1-targets)*torch.log(1-predicts)).sum()
    return loss


def dice_loss(predicts, targets):
    smooth = 1.
    predicts = predicts.reshape(-1)
    targets = targets.reshape(-1)
    intersection = torch.dot(predicts, targets)
```

```
    score = (2. * intersection + smooth) / (predicts.sum() + targets.sum() + smooth)
    return 1. - score


def train_unet_with_eval(model,
                         trainloader,
                         testloader,
                         train_criterion,
                         test_criterion,
                         device,
                         model_path='/content/drive/My Drive/csc420/unet.model',
                         epochs=5):
    print("Start Training ...")
    print("trianloader length:{}".format(len(trainloader)))
    print("testloader length:{}".format(len(testloader)))
    print("saved model path:{}".format(model_path))

    optimizer = optim.Adam(model.parameters())
    model.train()

    min_eval_loss = np.inf

    train_losses, test_losses = [], []
    for e in range(epochs):
#         print('Epoch {}/{}'.format(e+1, epochs))
#         print('=' * 20)

        running_loss = 0
        for i, data in enumerate(trainloader, 0):
#             print(i)
#             print("===Trainging phase===")
            optimizer.zero_grad()
            inputs, masks = data
#             print(inputs.shape, masks.shape)
            inputs = inputs.to(device)
            masks = masks.to(device)

            outputs = model(inputs)
#             print(outputs)
#             print("outputs size: {}".format(outputs.shape))
            loss = train_criterion(outputs, masks)
#             print("computed trainging loss: {}".format(loss.item()))

            # backward and optimize for training
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            train_losses.append(running_loss/len(trainloader))
        else:
            # Evaluate model after every epoch
#             print("===Predicting phase===")

            # Turn off gradients for validation, saves memory and computations
```

```
            with torch.no_grad():
                test_loss = 0
                accuracy = 0
                for i, data in enumerate(testloader, 0):
#                     print(i)
                    test_inputs, test_masks = data
                    test_inputs = test_inputs.to(device)
                    test_masks = test_masks.to(device)

                    # predict and caculate loss
                    eval_predicts = model(test_inputs)
                    eval_loss = test_criterion(eval_predicts, test_masks)
#                     print("computed loss: {}".format(eval_loss.item()))

                    test_loss += eval_loss.item()

                test_loss = test_loss/len(testloader)
                test_losses.append(test_loss)

                # save model with smallest valuation loss
                if test_loss < min_eval_loss:
                    print("Epoch{}: Save best model with test loss: {:.3f}.. ".format(e+1 , test_lo
                    min_eval_loss = test_loss
                    torch.save(model.state_dict(), model_path)

        print("Epoch: {}/{}.. ".format(e+1, epochs),
              "Training Loss: {:.3f}.. ".format(running_loss/len(trainloader)),
              "Test Loss: {:.3f}.. ".format(test_loss))
#         print("")

    print("List of train loss:{}".format(train_losses))
    print("List of test loss:{}".format(test_losses))
    # load best model weights
    print('Best valuation loss: {:4f}'.format(min_eval_loss))
```

Build U-Net Model and Call Train UNet Function with Dice Coefficient a

```
# build a u-net
model = UNet()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# train u-net with given input and masks and with dice loss as loss function
print("Train unet with dice coefficient as loss function...")
train_unet_with_eval(model,
                     train_loader,
                     test_loader,
                     dice_loss,
                     dice_loss,
                     device,
                     model_path=saved_unet_dice_path,
```

```
                    epochs=10)
```

```
Train unet with dice coefficient as loss function...
Start Training ...
trianloader length:15
testloader length:6
saved model path:/content/drive/My Drive/csc420/unet_dice.model
Epoch1: Save best model with test loss: 0.422..
Epoch: 1/10..  Training Loss: 0.507..  Test Loss: 0.422..
Epoch2: Save best model with test loss: 0.341..
Epoch: 2/10..  Training Loss: 0.385..  Test Loss: 0.341..
Epoch3: Save best model with test loss: 0.311..
Epoch: 3/10..  Training Loss: 0.337..  Test Loss: 0.311..
Epoch: 4/10..  Training Loss: 0.324..  Test Loss: 0.332..
Epoch5: Save best model with test loss: 0.298..
Epoch: 5/10..  Training Loss: 0.305..  Test Loss: 0.298..
Epoch: 6/10..  Training Loss: 0.284..  Test Loss: 0.325..
Epoch7: Save best model with test loss: 0.279..
Epoch: 7/10..  Training Loss: 0.279..  Test Loss: 0.279..
Epoch8: Save best model with test loss: 0.260..
Epoch: 8/10..  Training Loss: 0.296..  Test Loss: 0.260..
Epoch: 9/10..  Training Loss: 0.287..  Test Loss: 0.270..
Epoch: 10/10..  Training Loss: 0.276..  Test Loss: 0.263..
List of train loss:[0.03453582525253296, 0.07016656001408896, 0.1137714187304178
List of test loss:[0.4221632480621338, 0.3408726652463277, 0.3108118573824565, 0
Best valuation loss: 0.259752
```

Build U-Net Model and Call Train UNet Function with BCE as Loss

```
# build a new model to train with bce function
model = UNet()
model = model.to(device)

# train u-net with given input and masks and with BCE as loss function
print("Train unet with BCE coefficient as loss function...")
train_unet_with_eval(model,
                     train_loader,
                     test_loader,
                     cross_entropy_loss,
                     dice_loss,
                     device,
                     model_path=saved_unet_bce_path,
                     epochs=10)
```

```
Train unet with BCE coefficient as loss function...
Start Training ...
trianloader length:15
testloader length:6
saved model path:/content/drive/My Drive/csc420/unet_bce.model
Epoch1: Save best model with test loss: 0.533..
Epoch: 1/10..  Training Loss: 0.692..  Test Loss: 0.533..
Epoch2: Save best model with test loss: 0.467..
Epoch: 2/10..  Training Loss: 0.600..  Test Loss: 0.467..
Epoch3: Save best model with test loss: 0.417..
Epoch: 3/10..  Training Loss: 0.550..  Test Loss: 0.417..
Epoch: 4/10..  Training Loss: 0.521..  Test Loss: 0.422..
Epoch: 5/10..  Training Loss: 0.503..  Test Loss: 0.426..
Epoch6: Save best model with test loss: 0.388..
Epoch: 6/10..  Training Loss: 0.495..  Test Loss: 0.388..
Epoch: 7/10..  Training Loss: 0.516..  Test Loss: 0.398..
Epoch8: Save best model with test loss: 0.369..
Epoch: 8/10..  Training Loss: 0.483..  Test Loss: 0.369..
Epoch: 9/10..  Training Loss: 0.461..  Test Loss: 0.388..
Epoch: 10/10..  Training Loss: 0.466..  Test Loss: 0.383..
List of train loss:[0.045386401812235515, 0.10285961627960205, 0.144694224993387
List of test loss:[0.5331950187683105, 0.46684059500694275, 0.4172590672969818, 
Best valuation loss: 0.368942
```

Evaluation

Read any trained weight and evaluate on the trained model. Compute dice score and test accuracy.

```
# load test input images and mask with no shuffle
eval_dataset = CatDataset(test_input_dir, test_mask_dir, transform=img_transform)
eval_loader = torch.utils.data.DataLoader(eval_dataset, batch_size=1)

# use trained u-net to predict the test images
def evaluate_net(model, dataloader, loss_func, device):
    model.eval()

    accuracy = 0.
    test_loss = 0.
    all_predicts = 0
    with torch.no_grad():
        for i, data in enumerate(dataloader, 0):
            inputs, masks = data
            inputs = inputs.to(device)
            masks = masks.to(device)
            # predict and caculate loss
            predicts = model(inputs)
            loss = loss_func(predicts, masks)
            predicted_masks = (predicts>0.5).int()
            accuracy += float((predicted_masks == masks).sum()) / predicts.nelement()
            test_loss += loss.item()
```

```
            all_predicts = predicts if i is 0 else torch.cat([all_predicts, predicts], 0)

    accuracy = accuracy / len(dataloader)
    print("")
    print(all_predicts.shape)
    pil_predict_list = []

    for predict in torch.split(all_predicts, 1, dim=0):
#         print("predict[0]:{}".format(predict[0].shape))
#         transforms.ToPILImage()(predict[0].cpu())
        pil_predict_list.append(transforms.ToPILImage()(predict[0].cpu()))

    print(len(pil_predict_list))

    print("Dice Score: {:.3f}".format(1-test_loss/len(dataloader)))
    print("Test Accuracy: {}".format(accuracy))

    return pil_predict_list

print("Evaluation on UNet trained with dice loss: ")
trained_model = UNet().to(device)
trained_model.load_state_dict(torch.load(saved_unet_dice_path, map_location='cpu'))

predicts_dice = evaluate_net(trained_model, eval_loader, dice_loss, device)
```

```
Evaluation on UNet trained with dice loss:

torch.Size([21, 1, 128, 128])
21
Dice Score: 0.723
Test Accuracy: 0.6917579287574405
```

```
print("Evaluation on UNet trained with bce loss: ")
trained_model = UNet().to(device)
trained_model.load_state_dict(torch.load(saved_unet_bce_path, map_location='cpu'))

predicts_bce = evaluate_net(trained_model, eval_loader, dice_loss, device)
```

```
Evaluation on UNet trained with bce loss:

torch.Size([21, 1, 128, 128])
21
Dice Score: 0.600
Test Accuracy: 0.6913364955357143
```

Show test images with true masks and predict masks

```
def read_test_images(test_input_dir, test_mask_dir):
    # read images and masks again
    img_ids = [int(f.split('.')[-2]) for f in os.listdir(test_input_dir)]
    first_img = os.listdir(test_input_dir)[0]
    first_mask = os.listdir(test_mask_dir)[0]
```

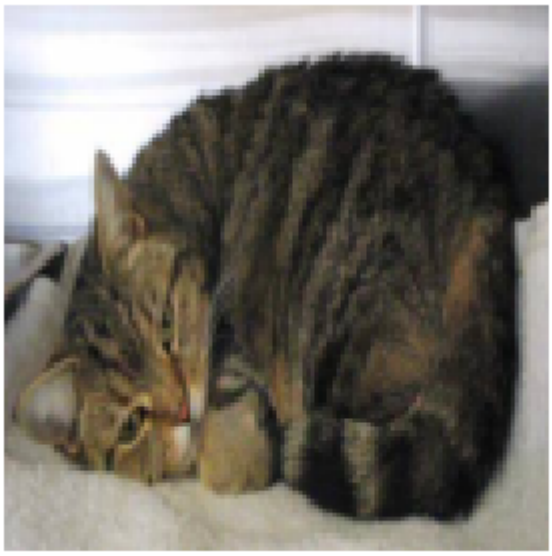
```
    img_name = first_img.split('.')[0]
    img_extension = first_img.split('.')[-1]
    mask_name = first_mask.split('.')[0]
    mask_extension = first_mask.split('.')[-1]
    imgs = []
    masks = []

    for id in img_ids:
        img = Image.open(test_input_dir+img_name+'.'+str(id)+'.'+img_extension)
        mask = Image.open(test_mask_dir+mask_name+'.'+str(id)+'.'+mask_extension)
        imgs.append(transforms.Resize((128, 128))(img))
        masks.append(transforms.Resize((128, 128))(mask))

    return imgs, masks

def process_predict_masks(predicts):
    predict_masks = []
    for predict in predicts:
        predict_masks.append((np.asarray(predict) > 255 * 0.5).astype(int) * 255)
    return predict_masks

def show_all_images(imgs, masks, predicts):
    # Get predicted masked from trained model and make them into a numpy array
    for i in range(len(predicts)):
      predicts[i] = ((np.asarray(predicts[i]))).astype(int)

    rows = 3
    cols = len(predicts)
    fig, axs = plt.subplots(rows, cols, figsize=(21,5))

    for i in range(rows):
      for j in range(cols):
        axs[i, j].axis('off')
        if i is 0:
          axs[i, j].imshow(imgs[j])
          axs[i, j].set_aspect('equal')
        elif i is 1:
          axs[i, j].imshow(masks[j], cmap="Greys")
          axs[i, j].set_aspect('equal')
        elif i is 2:
          axs[i, j].imshow(predicts[j], cmap="Greys")
          axs[i, j].set_aspect('equal')

    fig.subplots_adjust(wspace=0, hspace=0)
    plt.show()


def show_one_image(img, mask, predict):
    # Get predicted masked from trained model and make them into a numpy array
    predict = ((np.asarray(predict))).astype(int)
    fig, axs = plt.subplots(1, 3, figsize=(21,5))

    axs[0].axis('off')
    axs[0].imshow(img)
```

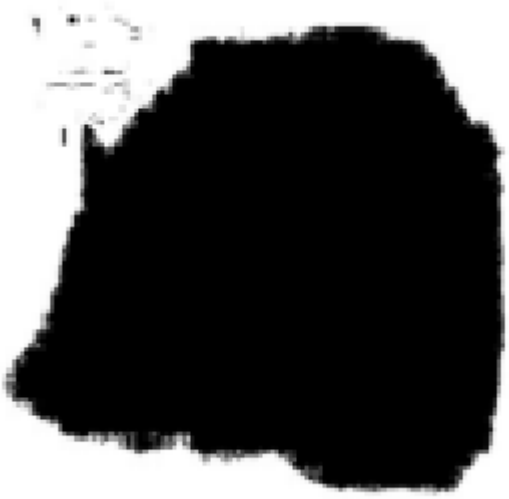
```
    axs[0].set_aspect('equal')

    axs[1].axis('off')
    axs[1].imshow(mask, cmap="Greys")
    axs[1].set_aspect('equal')

    axs[2].axis('off')
    axs[2].imshow(predict, cmap="Greys")
    axs[2].set_aspect('equal')

    fig.subplots_adjust(wspace=0, hspace=0)
    plt.show()

imgs, masks = read_test_images(test_input_dir, test_mask_dir)
predict_masks_dice = process_predict_masks(predicts_dice)
show_all_images(imgs, masks, predict_masks_dice)
```

```
imgs, masks = read_test_images(test_input_dir, test_mask_dir)
predict_masks_bce = process_predict_masks(predicts_bce)
show_all_images(imgs, masks, predict_masks_bce)
```

```
show_one_image(imgs[20], masks[20], predict_masks_dice[20])
```

```
show_one_image(imgs[20], masks[20], predict_masks_dice[20])
```

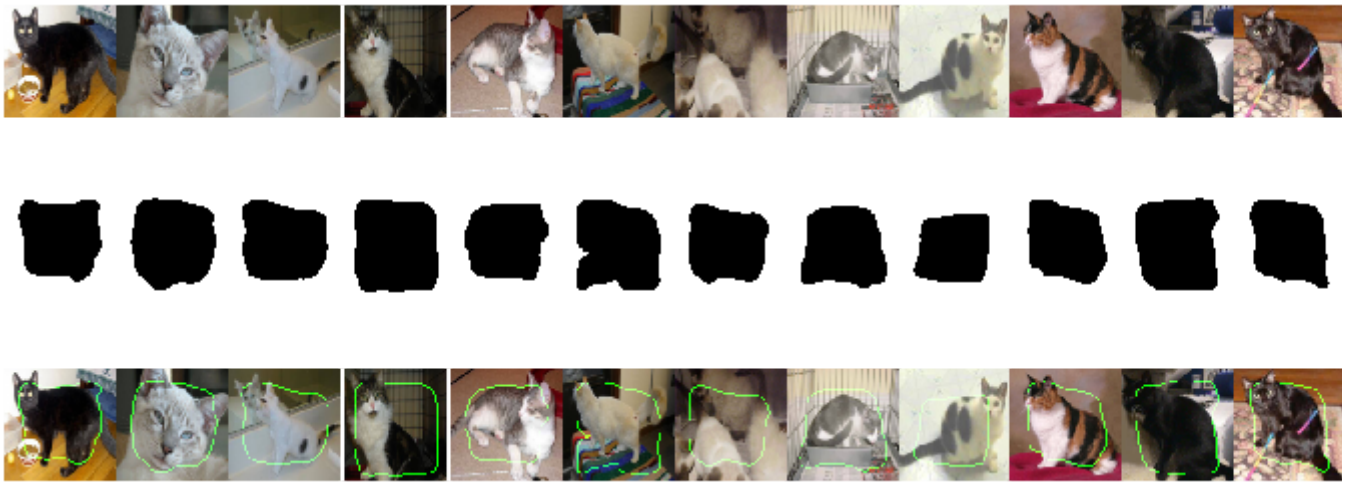




```
show_one_image(imgs[20], masks[20], predict_masks_bce[20])
```

1.4 Visualizing segmentation predictions

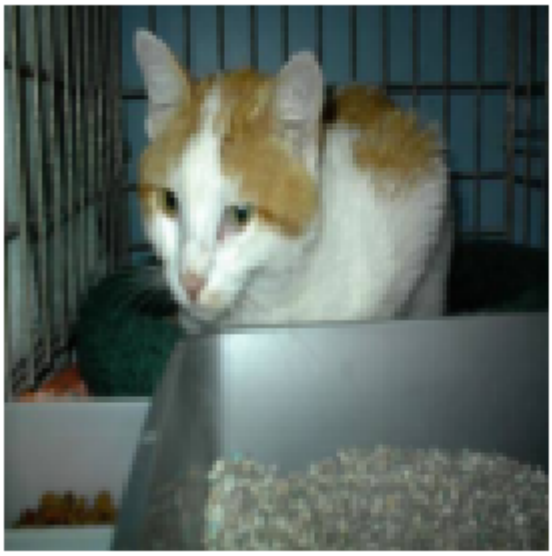
```
def show_segmentation(imgs, predict_masks):
    predict_segs = []
    for i in range(len(predict_masks)):
        predict = predict_masks[i].astype(float)
        laplacian = cv.Laplacian(predict, cv.CV_64F)

        predict_img = np.asarray(imgs[i]).copy()
        predict_img[laplacian != 0., 1] = 255
        predict_segs.append(predict_img)

    show_all_images(imgs, predict_masks, predict_segs)

def show_single_segmentation(img, predict_mask):
    predict = predict_mask.astype(float)
    laplacian = cv.Laplacian(predict, cv.CV_64F)

    predict_img = np.asarray(img).copy()
    predict_img[laplacian != 0., 1] = 255
    show_one_image(img, predict_mask, predict_img)

show_segmentation(imgs, predict_masks_dice)
```



```
show_segmentation(imgs, predict_masks_bce)
```

```
show_single_segmentation(imgs[3], predict_masks_dice[3])
```

```
show_single_segmentation(imgs[18], predict_masks_bce[18])
```



1.3 Transfer Learning

With The Oxford-IIIT Pet Dataset. Since the set has both images for cat and dogs, ignore all dog imag dataset has some invalid images. https://www.robots.ox.ac.uk/~vgg/data/pets/

Cat_breeds: ['Abyssinian', 'Bengal', 'Birman', 'Bombay', 'British_Shorthair', 'Egyptian_Mau', 'Maine_Coon' 'Siamese', 'Sphynx'].

```
# Code for testing

transfer_input_dir = '/content/drive/My Drive/csc420/pet_data/input/'
transfer_mask_dir = '/content/drive/My Drive/csc420/pet_data/mask/'

input_names = [f.split('.')[0] for f in os.listdir(transfer_input_dir)]
mask_names = [f.split('.')[0] for f in os.listdir(transfer_mask_dir)]


img_names = [name for name in input_names if name in mask_names]
# print(img_names[480:490])

img_files = []
mask_files = []

# add corresponding image and mask path into list
for name in img_names:
    img_files.append(transfer_input_dir+name+'.jpg')
    mask_files.append(transfer_mask_dir+name+'.png')


print(len(img_names))
```

```
index = 30
img_path = img_files[index]
mask_path = mask_files[index]

img = Image.open(img_path)
mask = Image.open(mask_path)

# Process mask images
mask_np = np.asarray(mask)
print(mask_np.shape)
new_mask = mask_np.copy()
new_mask -= 1
new_mask *= 255
mask = Image.fromarray(new_mask, 'L')

# img
```

```
1101
(375, 500)
```

Define Customized Data and Load Data

```
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import transforms

transfer_input_dir = '/content/drive/My Drive/csc420/pet_data/input/'
transfer_mask_dir = '/content/drive/My Drive/csc420/pet_data/mask/'
cat_breeds = ['Abyssinian', 'Bengal', 'Birman', 'Bombay', 'British_Shorthair',
              'Egyptian_Mau', 'Maine_Coon', 'Persian', 'Ragdoll', 'Russian_Blue',
              'Siamese', 'Sphynx']

# Define customized dataset
class OxfordCatDataset(Dataset):
    def __init__(self, img_dir, mask_dir, transform=None):
        input_names = [f.split('.')[0] for f in os.listdir(img_dir)]
        mask_names = [f.split('.')[0] for f in os.listdir(mask_dir)]

        # Find file names in both input and mask directory
        img_names = [name for name in input_names if name in mask_names]

        self.img_files = []
        self.mask_files = []
        self.transform = transform

        # add corresponding image and mask path into list
        for name in img_names:
            self.img_files.append(img_dir+name+'.jpg')
            self.mask_files.append(mask_dir+name+'.png')


    def __getitem__(self, index):
        img_path = self.img_files[index]
```

```
        mask_path = self.mask_files[index]

        img = Image.open(img_path)
        mask = Image.open(mask_path)

        # Process mask images
        mask_np = np.asarray(mask)
        new_mask = mask_np.copy()
        new_mask -= 1
        new_mask *= 255
        mask = Image.fromarray(new_mask, 'L')

        if self.transform is not None:
            img = self.transform(img)
            mask = self.transform(mask)

        return img, mask

    def __len__(self):
        return len(self.img_files)


# load training input images and mask
img_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])

transfer_dataset = OxfordCatDataset(transfer_input_dir, transfer_mask_dir, transform=img_transform)
transfer_loader = torch.utils.data.DataLoader(transfer_dataset, batch_size=10, shuffle=True, num_wo

inputs, masks = next(iter(transfer_loader))
print(len(transfer_loader))
print(inputs.shape, masks.shape)

# for i, data in enumerate(transfer_loader, 0):
#     print(i)
#     inputs, masks = data
#     print(inputs.shape, masks.shape)
```

```
111
torch.Size([10, 3, 128, 128]) torch.Size([10, 1, 128, 128])
```

Train Model

Pre-Training

```
# build a u-net
model = UNet()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
model = model.to(device)

# train u-net with given input and masks and with dice loss as loss function
print("Train unet with dice coefficient as loss function...")
train_unet_with_eval(model,
                     transfer_loader,
                     test_loader,
                     dice_loss,
                     dice_loss,
                     device,
                     model_path=saved_unet_transfer_path,
                     epochs=10)
```

```
Train unet with dice coefficient as loss function...
Start Training ...
trianloader length:111
testloader length:6
saved model path:/content/drive/My Drive/csc420/unet_transfer.model
Epoch1: Save best model with test loss: 0.614..
Epoch: 1/10..  Training Loss: 0.237..  Test Loss: 0.614..
Epoch2: Save best model with test loss: 0.608..
Epoch: 2/10..  Training Loss: 0.157..  Test Loss: 0.608..
Epoch: 3/10..  Training Loss: 0.143..  Test Loss: 0.633..
Epoch: 4/10..  Training Loss: 0.132..  Test Loss: 0.620..
Epoch: 5/10..  Training Loss: 0.125..  Test Loss: 0.701..
Epoch: 6/10..  Training Loss: 0.115..  Test Loss: 0.671..
Epoch: 7/10..  Training Loss: 0.117..  Test Loss: 0.691..
Epoch: 8/10..  Training Loss: 0.110..  Test Loss: 0.691..
Epoch: 9/10..  Training Loss: 0.102..  Test Loss: 0.718..
Epoch: 10/10..  Training Loss: 0.103..  Test Loss: 0.688..
List of train loss:[0.003157599015278859, 0.00636784450427906, 0.009613651413101
List of test loss:[0.6139963865280151, 0.6081120570500692, 0.6329765121142069, 0.
Best valuation loss: 0.608112
```

Fine- Tuning

```
# build a new model to train with bce function
pre_trained_model = UNet().to(device)
pre_trained_model.load_state_dict(torch.load(saved_unet_dice_path, map_location='cpu'))

count = 0
for child in pre_trained_model.children():
    count += 1
    if count < 6:
        for param in child.parameters():
            param.requires_grad = False

# summary(pre_trained_model, (3, 128, 128))

# train u-net with given input and masks and with BCE as loss function
print("Train unet with DICE coefficient as loss function...")
```

```
train_unet_with_eval(model,
                     train_loader,
                     test_loader,
                     dice_loss,
                     dice_loss,
                     device,
                     model_path=saved_unet_fine_tuning_path,
                     epochs=10)
```

```
Train unet with DICE coefficient as loss function...
Start Training ...
trianloader length:15
testloader length:6
saved model path:/content/drive/My Drive/csc420/unet_transfer_final.model
Epoch1: Save best model with test loss: 0.196..
Epoch: 1/10..  Training Loss: 0.136..  Test Loss: 0.196..
Epoch: 2/10..  Training Loss: 0.138..  Test Loss: 0.221..
Epoch3: Save best model with test loss: 0.195..
Epoch: 3/10..  Training Loss: 0.131..  Test Loss: 0.195..
Epoch: 4/10..  Training Loss: 0.128..  Test Loss: 0.201..
Epoch: 5/10..  Training Loss: 0.126..  Test Loss: 0.199..
Epoch: 6/10..  Training Loss: 0.121..  Test Loss: 0.229..
Epoch: 7/10..  Training Loss: 0.106..  Test Loss: 0.246..
Epoch: 8/10..  Training Loss: 0.111..  Test Loss: 0.210..
Epoch: 9/10..  Training Loss: 0.107..  Test Loss: 0.222..
Epoch: 10/10..  Training Loss: 0.105..  Test Loss: 0.203..
List of train loss:[0.006275371710459391, 0.011706717809041341, 0.017134873072306
List of test loss:[0.19558045268058777, 0.22125168641408285, 0.1954772969086965,
Best valuation loss: 0.195477
```

Evaluate Model

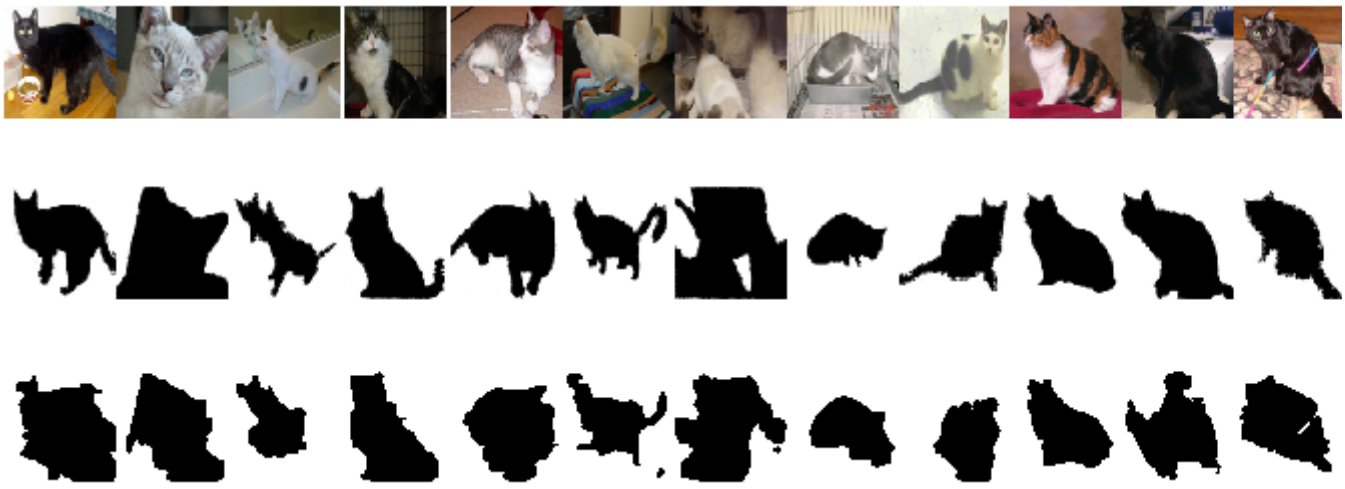
```
trained_model = UNet().to(device)
trained_model.load_state_dict(torch.load(saved_unet_transfer_path, map_location='cpu'))

predicts = evaluate_net(trained_model, eval_loader, dice_loss, device)
```

```
torch.Size([21, 1, 128, 128])
21
Dice Score: 0.363
Test Accuracy: 0.20617385137648808
```

```
# load test input images and mask with no shuffle
eval_dataset = CatDataset(test_input_dir, test_mask_dir, transform=img_transform)
eval_loader = torch.utils.data.DataLoader(eval_dataset, batch_size=1)

trained_model = UNet().to(device)
trained_model.load_state_dict(torch.load(saved_unet_fine_tuning_path, map_location='cpu'))

predicts = evaluate_net(trained_model, eval_loader, dice_loss, device)
```

```
imgs, masks = read_test_images(test_input_dir, test_mask_dir)
predict_masks = process_predict_masks(predicts)
show_all_images(imgs, masks, predict_masks)
```

```
torch.Size([21, 1, 128, 128])
21
Dice Score: 0.793
Test Accuracy: 0.7495378766741071
```



```
show_segmentation(imgs, predict_masks)
```

```
show_single_segmentation(imgs[20], predict_masks[20])
```