CV ASSIGNMENT 3

Q1: [COLAB LINK TO CODE FILE] [DRIVE LINK TO DATASETS AND SAVED MODELS]

Methodology

- Importing original MNIST dataset from openml

```
from sklearn.datasets import fetch_openml
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
```

```
# importing dataset from openml
mnist=fetch_openml('mnist_784')
X=mnist.data
y=mnist.target
X=X.astype(dtype='float32')
y=y.astype(dtype='long')
```

- Part a, find foreground masks using the TSS thresholding functions from MidSem Q1 and dump the new dataset using pickle.

```
def calc_tss(img):
    """
    Returns the TSS of the pixel values given in 'img'.
    """
    if img.size == 0:   # handling empty set case
        return 0
    mu = np.mean(img)   # find the mean
    return np.sum((img-mu)**2)  # return TSS
```

Dushyant Panchal, 2018033

```python
def otsu_threshold(img_gr):
    """
    Returns the OTSU Threshold for the pixel values
    in 'img' considering the sum of TSS's.
    """
    # accumulator variables
    min_tss_sum = float("inf")
    otsu_t = -1

    # looping over all possible thresholds
    for t in range(0,256):

        # set I0
        ind0 = img_gr<t
        I0 = img_gr[ind0]
        tss0 = calc_tss(I0)

        # set I1
        ind1 = img_gr>=t
        I1 = img_gr[ind1]
        tss1 = calc_tss(I1)

        # unweighted tss sum
        tss_sum = tss0 + tss1

        # see if current 't' is better
        if tss_sum < min_tss_sum:
            min_tss_sum = tss_sum
            otsu_t = t
    # print(min_tss_sum)
    # return the threshold calculated above
    return otsu_t
```

```
masks = []
for i in range(X.shape[0]):
  img = X[i]
  thresh = otsu_threshold(img)
  mask = 1*(img>=thresh)
  masks.append(mask)

masks = np.stack(masks)
```

```
import pickle

Q1a_data = {'images':X, 'fg-masks':masks, 'labels':mnist.target.astype('long')}
pickle.dump(Q1a_data, open("/content/drive/MyDrive/CV_A3/Q1/Q1a_new.pkl","wb"))
```

- Part b, use cv2 library functions to calculate the minimum bounding circle on above foreground masks and dump the new dataset using pickle.

```
circles = []

for i in range(masks.shape[0]):
  mask = masks[i].reshape(28,28).astype('uint8')*255

  # find all contours
  contours,_ = cv2.findContours(mask, 1, 2)

  # generate circles for each contour
  centers = []
  for cnt in contours:
    (x,y), r = cv2.minEnclosingCircle(cnt)
    centers.append( (int(x), int(y)) )

  # connect them all in case multiple contours found
  for j in range(1, len(centers)):
    mask = cv2.line(mask, centers[j-1], centers[j], 255, 2)

  # find the new contour on connected objects
  contours,_ = cv2.findContours(mask, 1, 2)

  # find the final enclosing circle now
  (x,y), r = cv2.minEnclosingCircle(contours[-1])
  circles.append([x,y,r])

circles = np.array(circles)
```

```
data['circles'] = circles
pickle.dump(data, open("/content/drive/MyDrive/CV_A3/Q1/Q1b_new.pkl","wb"))
```

- Part c, concatenate 4 consecutive images in a 2x2 fashion as well as their corresponding foreground masks while incrementing the window by 1 sample. Labelling: 0 denotes black background and 1-10 are mapped to 0-9 digit classes respectively.

```
X=data['images'].reshape(-1,28,28)
Y=data['fg-masks'].reshape(-1,28,28)
y=data['labels']
for i in range(Y.shape[0]):
  label = y[i]
  Y[i][Y[i]==1] = label+1
```

```
images = []
masks = []
for i in range(0,X.shape[0]-3):
  img = np.concatenate((np.concatenate((X[i],X[i+1]), axis=1),np.concatenate((X[i+2],X[i+3]), axis=1)), axis=0)
  mask = np.concatenate((np.concatenate((Y[i],Y[i+1]), axis=1),np.concatenate((Y[i+2],Y[i+3]), axis=1)), axis=0)
  images.append(img)
  masks.append(mask)

images = np.stack(images)
masks = np.stack(masks)
images.shape, masks.shape
```

```
((69997, 56, 56), (69997, 56, 56))
```

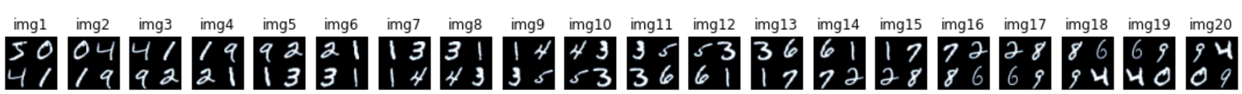
```
data = {'images':images, 'labels':masks}
pickle.dump(data, open("/content/drive/MyDrive/CV_A3/Q1/Q1c.pkl","wb"))
```

First 20 Samples from the datasets created:

- **Q1a**

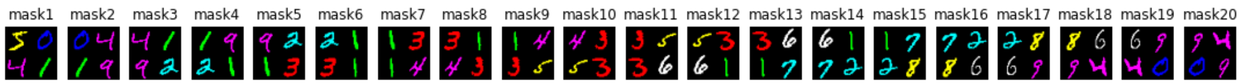
- **Q1b**

- **Q1c**



Dushyant Panchal, 2018033

CV ASSIGNMENT 3



References (for Q1):

- https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html
- https://stackoverflow.com/questions/47457918/how-can-i-get-the-minimum-enclosing-circle-with-opencv

Q2: **[COLAB LINK TO CODE FILE] [DRIVE LINK TO DATASETS AND SAVED MODELS]**

Code:

```python
from google.colab import drive
drive.mount('/content/drive')

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as T
from torch.utils.data import Dataset, DataLoader
import numpy as np

"""## HYPERPARAMETERS"""

num_epochs = 20
batch_size = 64

"""## DATASET LOADER"""

class CustomDataset(Dataset):
  def __init__(self, X, y):
    self.n_samples = X.shape[0]
    self.images = torch.from_numpy(X.reshape(-1,1,28,28))
    self.masks = torch.from_numpy(y.reshape(-1,28,28))

  def __getitem__(self, index):
    return self.images[index], self.masks[index]

  def __len__(self):
    return self.n_samples
```

```
"""## IMPORTING DATA AND PREPROCESSING"""

import pickle
from sklearn.model_selection import train_test_split

# importing pickle dumped data from Q1
data = pickle.load(open("/content/drive/MyDrive/CV_A3/Q1/Q1a.pkl","rb"))
data.keys()

# preprocessing
X = data['images'].astype('float32')/255
Y = data['fg-masks']
y = data['labels']

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
stratify=y, random_state=42)

# train and test dataset
train_dataset = CustomDataset(X_train, y_train)
test_dataset = CustomDataset(X_test, y_test)

# train and test data loader for batch training
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True, pin_memory=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=batch_size, shuffle=False, pin_memory=True)


"""## NEURAL NETWORK"""

class CNN(nn.Module):

  def __init__(self):
    super(CNN, self).__init__()

    self.conv1 = nn.Sequential(
        nn.Conv2d(1, 8, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
```

```
    )

    self.conv2 = nn.Sequential(
        nn.Conv2d(8, 16, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )

    self.conv3 = nn.Sequential(
        nn.Conv2d(16, 32, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )

    self.conv4 = nn.Sequential(
        nn.Conv2d(32, 64 kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )

    self.conv_t4 = nn.Sequential(
        nn.ConvTranspose2d(64, 32, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )

    self.conv_t3 = nn.Sequential(
        nn.ConvTranspose2d(32, 16, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )

    self.conv_t2 = nn.Sequential(
        nn.ConvTranspose2d(16, 8, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )

    self.conv_t1 = nn.Sequential(
        nn.ConvTranspose2d(8, 2, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )


def forward(self, x):
```

```
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)

    x = self.conv_t4(x)
    x = self.conv_t3(x)
    x = self.conv_t2(x)
    x = self.conv_t1(x)

    return x

"""### FUNCTION FOR MODEL EVALUATION"""

def evaluate_model(model, data_loader, criterion):

    model.eval()

    net_loss = 0

    with torch.no_grad():
        for imgs, masks in data_loader:
            imgs = imgs.to(device)
            masks = masks.to(device)

            pred_masks = model(imgs)

            loss = criterion(pred_masks, masks)
            net_loss += loss.item()

    net_loss /= len(data_loader)

    return net_loss

"""## TRAINING THE MODEL"""

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.Adam(model.parameters())

n_total_steps = len(train_loader)

loss_best_model = 100
PATH = "/content/drive/MyDrive/CV_A3/Q2/best_model1.pt"

for epoch in range(num_epochs):
    for i, (imgs, masks) in enumerate(train_loader):
        imgs = imgs.to(device)
        masks = masks.to(device)

        # Forward pass
        pred_masks = model(imgs)
        loss = criterion(pred_masks, masks)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    with torch.no_grad():
        model.eval()
        test_loss = evaluate_model(model, test_loader, criterion)
        train_loss = evaluate_model(model, train_loader, criterion)

        updated = ""

        if test_loss < loss_best_model:
            loss_best_model = test_loss
            torch.save({
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'train_loss': train_loss,
              'test_loss': test_loss,
            }, PATH)
            updated = "++"

        print (f'Epoch [{epoch+1}/{num_epochs}], Train-Loss:
{train_loss:.4f}, Test-Loss: {test_loss:.4f} {updated}')
```

```
        model.train()
```

Model Architecture:

- DownScaling through 2d convolutions (4 such layers)
- UpScaling through 2d t-convolutions (4 such layers)
- Thus producing the output of the same shape and size as the input.
- Correctness of the predicted labels is measured using pixel-wise **CrossEntropyLoss**.
- **Adam** Optimizer for training the model.

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv3): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv4): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv_t4): Sequential(
    (0): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv_t3): Sequential(
    (0): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv_t2): Sequential(
    (0): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv_t1): Sequential(
    (0): ConvTranspose2d(8, 2, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
)
```

Results

- Cross entropy loss

```
PATH = "/content/drive/MyDrive/CV_A3/Q2/best_model.pt"

checkpoint = torch.load(PATH)

model = CNN().to(device)
model.load_state_dict(checkpoint['model_state_dict'])

train_loss = checkpoint['train_loss']
test_loss = checkpoint['test_loss']

print (f'Train-Loss: {train_loss:.4f}, Test-Loss: {test_loss:.4f}')
```

```
Train-Loss: 0.0025, Test-Loss: 0.0026
```

- Jaccard score
  - Function for jaccard score given two images

```
def jaccard(img1, img2):
  union = np.logical_or(img1, img2)
  intersect = np.logical_and(img1, img2)
  jacc = np.sum(intersect)/np.sum(union)
  return jacc
```

  - Function for avg jaccard score given dataset and model instances

```
def avg_jaccard_score(dataset, model):
  n = len(dataset)
  imgs, masks = dataset[:]
  imgs = imgs.to(device)
  masks = masks.to(device)
  pred_masks = F.softmax(model(imgs), dim=1)

  jacc = 0

  for i in range(n):
    mask = masks[i].reshape(28,28).cpu().numpy()
    pred_mask = pred_masks[i][1].reshape(28,28).cpu().detach().numpy()>0.5
    jacc += jaccard(mask, pred_mask)

  return jacc/n
```

  - The Jaccard Score

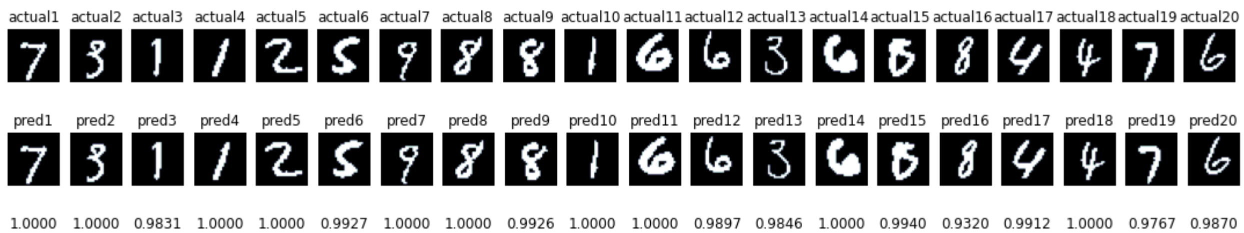
```
print("Avg Jaccard Score on Train Dataset:",avg_jaccard_score(train_loader, model))
```

```
Avg Jaccard Score on Train Dataset: 0.9916232886050755
```

```
print("Avg Jaccard Score on Test Dataset:",avg_jaccard_score(test_loader, model))
```

```
Avg Jaccard Score on Test Dataset: 0.9913862445447037
```

- Visualizing predictions: (row1 - actual, row2 - predicted, row3 - jaccard score)



Q3: [COLAB LINK TO CODE FILE] [DRIVE LINK TO DATASETS AND SAVED MODELS]

***Note**: since all the image samples do contain some or the other classes, we can omit the output unit for "is_present", since it will always give the output as '1' due to the whole dataset having 1 for its corresponding ground truth.*

Code:

```
from google.colab import drive
drive.mount('/content/drive')

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as T
from torch.utils.data import Dataset, DataLoader
import numpy as np

"""## HYPERPARAMETERS"""

num_epochs = 20
batch_size = 64
```

```
"""## DATASET LOADER"""

class CustomDataset(Dataset):
  def __init__(self, images, labels, circles):
    self.n_samples = images.shape[0]
    self.images = torch.from_numpy(images.reshape(-1,1,28,28))
    self.labels = torch.from_numpy(labels)
    self.circles = torch.from_numpy(circles)

  def __getitem__(self, index):
    return self.images[index], self.labels[index], self.circles[index]

  def __len__(self):
    return self.n_samples

"""## IMPORTING DATA AND PREPROCESSING"""

import pickle
from sklearn.model_selection import train_test_split

data = pickle.load(open("/content/drive/MyDrive/CV_A3/Q1/Q1b.pkl","rb"))
data.keys()

images = data['images'].astype('float32')/255
labels = data['labels']
circles = data['circles'].astype('float32')

images.shape, labels.shape, circles.shape

images_train, images_test, labels_train, labels_test, circles_train,
circles_test = train_test_split(images, labels, circles, test_size=0.2,
stratify=labels, random_state=42)

train_dataset = CustomDataset(images_train, labels_train, circles_train)
test_dataset = CustomDataset(images_test, labels_test, circles_test)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True, pin_memory=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=batch_size, shuffle=False, pin_memory=True)
```

```
"""## NEURAL NETWORK"""

class CNN(nn.Module):

  def __init__(self):
    super(CNN, self).__init__()

    self.conv_layers = nn.Sequential(
        nn.Conv2d(1, 8, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),

        nn.Conv2d(8, 16, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),

        nn.MaxPool2d(kernel_size=(2,2), stride=(2,2)),

        nn.Conv2d(16, 32, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),

        nn.Conv2d(32, 64, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),

        nn.MaxPool2d(kernel_size=(2,2), stride=(2,2)),
    )

    self.classifier = nn.Sequential(
        nn.Linear(64*4*4, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, 10),
    )

    self.localize = nn.Sequential(
        nn.Linear(64*4*4, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, 128),
        nn.ReLU(inplace=True),
        nn.Linear(128, 3),
    )
```

```
  def forward(self, x):
    features = self.conv_layers(x)
    features = features.view(-1, 64*4*4)
    return self.classifier(features), self.localize(features)

"""### FUNCTION FOR MODEL EVALUATION"""

def evaluate_model(model, data_loader, ce_loss, mse_loss):

    model.eval()

    clf_loss = 0
    lcl_loss = 0

    with torch.no_grad():
        for (imgs, labels, circles) in data_loader:
            imgs = imgs.to(device)
            labels = labels.to(device)
            circles = circles.to(device)

            pred_labels, pred_circles = model(imgs)
            classifier_loss = ce_loss(pred_labels, labels)
            localize_loss = mse_loss(pred_circles, circles)

            clf_loss += classifier_loss.item()
            lcl_loss += localize_loss.item()

    clf_loss /= len(data_loader)
    lcl_loss /= len(data_loader)

    return clf_loss, lcl_loss

"""## TRAINING THE MODEL"""

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = CNN().to(device)
ce_loss = nn.CrossEntropyLoss()
mse_loss = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters())
```

```
loss_best_model = [100, 100]

n_total_steps = len(train_loader)

PATH = "/content/drive/MyDrive/CV_A3/Q3/"

for epoch in range(num_epochs):
    for i, (imgs, labels, circles) in enumerate(train_loader):
        imgs = imgs.to(device)
        labels = labels.to(device)
        circles = circles.to(device)

        # Forward pass
        pred_labels, pred_circles = model(imgs)
        classifier_loss = ce_loss(pred_labels, labels)
        localize_loss = mse_loss(pred_circles, circles)
        loss = classifier_loss + localize_loss

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    with torch.no_grad():
        model.eval()
        test_loss = evaluate_model(model, test_loader, ce_loss, mse_loss)
        train_loss = evaluate_model(model, train_loader, ce_loss,
mse_loss)

        updated = ""

        if test_loss[0] < loss_best_model[0]:
            loss_best_model[0] = test_loss[0]
            torch.save({
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'train_loss': train_loss,
              'test_loss': test_loss,
            }, PATH+"ce_best_model.pt")
```

```
            updated += "ce++ "

        if test_loss[1] < loss_best_model[1]:
            loss_best_model[1] = test_loss[1]
            torch.save({
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'train_loss': train_loss,
              'test_loss': test_loss,
            }, PATH+"mse_best_model.pt")
            updated += "mse++"

        print (f'Epoch [{epoch+1}/{num_epochs}], Train-Loss: {train_loss},
Test-Loss: {test_loss} (ce_loss, mse_loss){updated}')
        model.train()
```

Model Architecture:

- Obtaining features using convolutions over the input image.
- These features are fed into a softmax layer of size 10 to predict the class labels.
- These features are also fed into an fc layer of size 3 to predict the bounding circle represented by (x,y) - the center coordinates and r - the radius of the circle.
- Correctness of predicted class labels is measured through **CrossEntropyLoss**.
- Correctness of the predicted enclosing circle is measured through **MeanSquaredError** over (x,y,r) - 3 tuple.
- **Adam** Optimizer for training the model.

```
CNN(
  (conv_layers): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=512, out_features=10, bias=True)
  )
  (localize): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=512, out_features=128, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=128, out_features=3, bias=True)
  )
)
```

Results

- Loss: (classifier_loss, localizer_loss) *(ce_best_model was chosen)*

```
PATH = "/content/drive/MyDrive/CV_A3/Q3/ce_best_model.pt"

checkpoint = torch.load(PATH)

model = CNN().to(device)
model.load_state_dict(checkpoint['model_state_dict'])

train_loss = checkpoint['train_loss']
test_loss = checkpoint['test_loss']

print (f'Train-Loss: {train_loss}, Test-Loss: {test_loss}')
```

```
Train-Loss: (0.006959874874604533, 0.0803002959404673), Test-Loss: (0.028359666604391345, 0.09041072970860081)
```

- Classification Accuracy Score

```
def get_accuracy(data_loader, model):

    n_correct = 0
    n_samples = 0

    with torch.no_grad():
        for (imgs, labels, circles) in data_loader:
            imgs = imgs.to(device)
            labels = labels.to(device)
            circles = circles.to(device)

            pred_labels, pred_circles = model(imgs)

            pred_labels = torch.argmax(F.softmax(pred_labels, dim=1), dim=1)
            n_correct += torch.sum(pred_labels==labels).item()
            n_samples += labels.shape[0]

    return 100*n_correct/n_samples
```

```
print(f"Accuracy Score on Train Dataset: {get_accuracy(train_loader, model):.2f}%")
```

```
Accuracy Score on Train Dataset: 99.82%
```

```
print(f"Accuracy Score on Test Dataset: {get_accuracy(test_loader, model):.2f}%")
```

```
Accuracy Score on Test Dataset: 99.15%
```

- Localizer Jaccard Score
    - Function for jaccard score given two images

```
def jaccard(img1, img2):
  union = np.logical_or(img1, img2)
  intersect = np.logical_and(img1, img2)
  jacc = np.sum(intersect)/np.sum(union)
  return jacc
```

    - Function for avg jaccard score given dataset and model instances

```python
def avg_jaccard_score(dataset, model):
  n = len(dataset)
  imgs, labels, circles = dataset[:]
  imgs = imgs.to(device)

  pred_labels, pred_circles = model(imgs)
  pred_labels = torch.argmax(F.softmax(pred_labels, dim=1), dim=1).cpu().detach().numpy()

  black = np.zeros((28,28))

  jacc = 0

  for i in range(n):
    if pred_labels[i] == labels[i]:
      x,y,r = pred_circles[i]
      pred_circle = cv2.circle(black.copy(), (int(x), int(y)), int(r), 255, -1)

      x,y,r = circles[i]
      act_circle = cv2.circle(black.copy(), (int(x), int(y)), int(r), 255, -1)

      jacc += jaccard(act_circle, pred_circle)

  return jacc/n
```

  - The Jaccard Score

```python
print("Avg Jaccard Score on Train Dataset:", avg_jaccard_score(train_loader, model))
```

```
Avg Jaccard Score on Train Dataset: 0.9285367280883048
```

```python
print("Avg Jaccard Score on Test Dataset:", avg_jaccard_score(test_loader, model))
```

```
Avg Jaccard Score on Test Dataset: 0.9200744465798885
```

- Visualizing Predictions: (row1 - predicted, row2 - actual)

Q4: [COLAB LINK TO CODE FILE] [DRIVE LINK TO DATASETS AND SAVED MODELS]

Code:

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as T
from torch.utils.data import Dataset, DataLoader
import numpy as np

"""## HYPERPARAMETERS"""

num_epochs = 20
batch_size = 64

"""## DATASET LOADER"""

class CustomDataset(Dataset):
  def __init__(self, X, y):
    self.n_samples = X.shape[0]
    self.images = torch.from_numpy(X.reshape(-1,1,56,56))
    self.labels = torch.from_numpy(y.reshape(-1,56,56))

  def __getitem__(self, index):
    return self.images[index], self.labels[index]

  def __len__(self):
    return self.n_samples

"""## IMPORTING DATA AND PREPROCESSING"""

import pickle
from sklearn.model_selection import train_test_split

data = pickle.load(open("/content/drive/MyDrive/CV_A3/Q1/Q1c.pkl","rb"))
data.keys()

X = data['images'].astype('float32')/255
y = data['labels']

X.shape, y.shape
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

train_dataset = CustomDataset(X_train, y_train)
test_dataset = CustomDataset(X_test, y_test)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True, pin_memory=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=batch_size, shuffle=False, pin_memory=True)

"""## NEURAL NETWORK"""

class CNN(nn.Module):

  def __init__(self):
    super(CNN, self).__init__()

    self.conv1 = nn.Sequential(
        nn.Conv2d(1, 16, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(16),
        nn.ReLU(inplace=True),
    )

    self.conv2 = nn.Sequential(
        nn.Conv2d(16, 32, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(32),
        nn.ReLU(inplace=True),
    )

    self.pool2 = nn.MaxPool2d(2, stride=2, return_indices=True)

    self.conv3 = nn.Sequential(
        nn.Conv2d(32, 64, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
    )

    self.conv4 = nn.Sequential(
        nn.Conv2d(64, 128, kernel_size=(3,3), stride=(1,1)),
```

```
        nn.BatchNorm2d(128),
        nn.ReLU(inplace=True),
    )

    self.pool4 = nn.MaxPool2d(2, stride=2, return_indices=True)

    self.conv5 = nn.Sequential(
        nn.Conv2d(128, 256, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(256),
        nn.ReLU(inplace=True),
    )

    self.conv_t5 = nn.Sequential(
        nn.ConvTranspose2d(256, 128, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(128),
        nn.ReLU(inplace=True),
    )

    self.unpool4 = nn.MaxUnpool2d(2, stride=2)

    self.conv_t4 = nn.Sequential(
        nn.ConvTranspose2d(128, 64, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
    )

    self.conv_t3 = nn.Sequential(
        nn.ConvTranspose2d(64, 32, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(32),
        nn.ReLU(inplace=True),
    )

    self.unpool2 = nn.MaxUnpool2d(2, stride=2)

    self.conv_t2 = nn.Sequential(
        nn.ConvTranspose2d(32, 16, kernel_size=(3,3), stride=(1,1)),
        nn.BatchNorm2d(16),
        nn.ReLU(inplace=True),
    )
```

```
    self.conv_t1 = nn.Sequential(
        nn.ConvTranspose2d(16, 11, kernel_size=(3,3), stride=(1,1)),
        nn.ReLU(inplace=True),
    )


  def forward(self, x):

    x = self.conv1(x)
    x = self.conv2(x)
    x, ind2 = self.pool2(x)

    x = self.conv3(x)
    x = self.conv4(x)
    x, ind4 = self.pool4(x)

    x = self.conv5(x)

    x = self.conv_t5(x)

    x = self.unpool4(x, ind4)
    x = self.conv_t4(x)
    x = self.conv_t3(x)

    x = self.unpool2(x, ind2)
    x = self.conv_t2(x)
    x = self.conv_t1(x)

    return x

"""### FUNCTION FOR MODEL EVALUATION"""

def evaluate_model(model, data_loader, criterion):

    model.eval()

    net_loss = 0

    with torch.no_grad():
        for imgs, masks in data_loader:
```

```
            imgs = imgs.to(device)
            masks = masks.to(device)

            pred_masks = model(imgs)

            loss = criterion(pred_masks, masks)
            net_loss += loss.item()

    net_loss /= len(data_loader)

    return net_loss

"""## TRAINING THE MODEL"""

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

n_total_steps = len(train_loader)

loss_best_model = 100
PATH = "/content/drive/MyDrive/CV_A3/Q4/best_model2.pt"

for epoch in range(num_epochs):
    for i, (imgs, masks) in enumerate(train_loader):
        imgs = imgs.to(device)
        masks = masks.to(device)

        # Forward pass
        pred_masks = model(imgs)
        loss = criterion(pred_masks, masks)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    with torch.no_grad():
```

```
        model.eval()
        test_loss = evaluate_model(model, test_loader, criterion)
        train_loss = evaluate_model(model, train_loader, criterion)

        updated = ""

        if test_loss < loss_best_model:
            loss_best_model = test_loss
            torch.save({
              'model_state_dict': model.state_dict(),
              'optimizer_state_dict': optimizer.state_dict(),
              'train_loss': train_loss,
              'test_loss': test_loss,
            }, PATH)
            updated = "++"

        print (f'Epoch [{epoch+1}/{num_epochs}], Train-Loss:
{train_loss:.4f}, Test-Loss: {test_loss:.4f} {updated}')
        model.train()
```

Model Architecture:

- DownScaling through 2d convolutions, 1 maxpool2d layer after conv2.
- UpScaling through 2d t-convolutions, 1 maxunpool2d layer before t-conv2.
- Thus producing the output of the same shape and size as the input.
- Correctness of the predicted labels is measured using pixel-wise **CrossEntropyLoss**.
- **Adam** Optimizer for training the model.

CV ASSIGNMENT 3

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv4): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv5): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv_t5): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (unpool4): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
  (conv_t4): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv_t3): Sequential(
    (0): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (unpool2): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
  (conv_t2): Sequential(
    (0): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv_t1): Sequential(
    (0): ConvTranspose2d(16, 11, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
)
```

Results

- Cross entropy loss

```
PATH = "/content/drive/MyDrive/CV_A3/Q4/best_model1.pt"

checkpoint = torch.load(PATH)

model = CNN().to(device)
model.load_state_dict(checkpoint['model_state_dict'])

train_loss = checkpoint['train_loss']
test_loss = checkpoint['test_loss']

print (f'Train-Loss: {train_loss:.4f}, Test-Loss: {test_loss:.4f}')
```

```
Train-Loss: 0.0057, Test-Loss: 0.0065
```

- Jaccard score
  - Function for jaccard score given two images

```
def jaccard(img1, img2):
  union = np.logical_or(img1, img2)
  intersect = np.logical_and(img1, img2)
  jacc = np.sum(intersect)/np.sum(union)
  return jacc
```

  - Function for avg jaccard score given dataset and model instances

CV ASSIGNMENT 3

```python
def avg_jaccard_score(data_loader, model):

  jacc = np.zeros(11)
  count = np.zeros(11)

  for (imgs, masks) in data_loader:
    imgs = imgs.to(device)
    masks = masks.to(device).cpu().numpy()
    pred_masks = F.softmax(model(imgs), dim=1)
    pred_masks = torch.argmax(pred_masks, dim=1).cpu().numpy()

    for j in range(imgs.shape[0]):
      masks_j = masks[j]
      pred_masks_j = pred_masks[j]
      for i in np.unique(masks_j):
        mask = (masks_j==i)
        pred_mask = (pred_masks_j==i)
        jacc[i] += np.sum(jaccard(mask, pred_mask))
        count[i] += 1

  return jacc/count
```

- The Jaccard Score (Macro average score over 11 classes, 0→bg, 1-10→0-9):

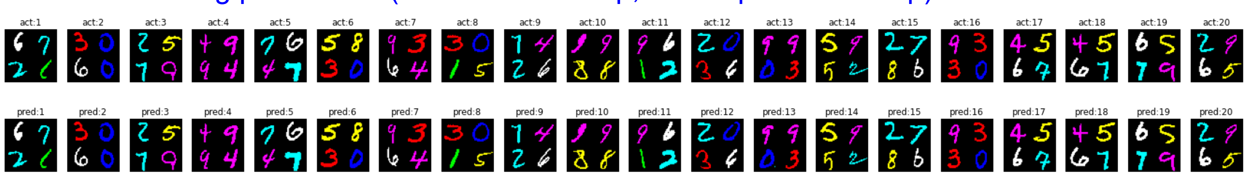
```python
print("Avg Jaccard Score on Train Dataset:", np.mean(avg_jaccard_score(train_loader, model)))
```

```
Avg Jaccard Score on Train Dataset: 0.984178278403969
```

```python
print("Avg Jaccard Score on Test Dataset:", np.mean(avg_jaccard_score(test_loader, model)))
```

```
Avg Jaccard Score on Test Dataset: 0.9816192472225979
```

- Visualizing predictions (row1 - actual map, row2 - predicted map)



References (for Q2,3,4):

- https://pytorch.org/docs/stable/generated/torch.nn.MaxUnpool2d.html
- https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html
- Lecture Slides
- PyTorch Docs

Dushyant Panchal, 2018033