DUE DATE: June 6th 11:59PM

Building a Deep Learning Model to Perform Image Classification with the American Sign Language Dataset

Now, it's your turn. You'll need to use what we have learned before to build a model to perform image classification with American Sign Language dataset.

Requirement:

- The data preparation and the training functions are given.
- You'll need to use a model of your choice and to perform the classification, aka predict the correct categories. Modify the `model` function to specify your model.
- Then use the training functions to train your model with `epoch=10`.
- You cannot ask for help from anyone outside your group. You can only ask questions to the instructor and only in the project QA session.

You can modify the cell below to include additional libraries you want to use.

```
import torch.nn as nn
import pandas as pd
import torch
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader

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

```
True
```

1. Data Preparation

Do not change anything in this part.

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
train_df = pd.read_csv("/content/drive/MyDrive/Spring 2025 STA138 Final Project/sign_mnist_train.csv")
valid_df = pd.read_csv("/content/drive/MyDrive/Spring 2025 STA138 Final Project/sign_mnist_valid.csv")
```

If you have erros in running the above cell, drag the folder `Spring 2025 STA138 Final Project` to `MyDrive`.

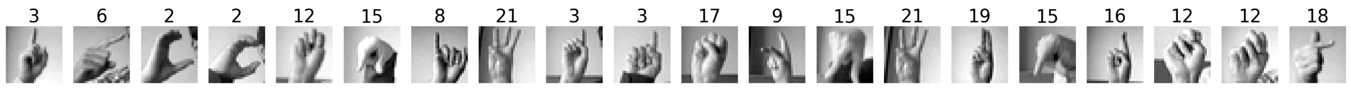
The following plots give you an idea of the feature images and the labels.

```
y_train = train_df.pop('label')
y_valid = valid_df.pop('label')
x_train = train_df.values
x_valid = valid_df.values
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(40,40))

num_images = 20
for i in range(num_images):
    row = x_train[i]
    label = y_train[i]

    image = row.reshape(28,28)
    plt.subplot(1, num_images, i+1)
    plt.title(label, fontdict={'fontsize': 30})
    plt.axis('off')
    plt.imshow(image, cmap='gray')
```



```
IMG_HEIGHT = 28
IMG_WIDTH = 28
IMG_CHS = 1

class MyDataset(Dataset):
    def __init__(self, base_df):
        x_df = base_df.copy()  # Some operations below are in-place
        y_df = x_df.pop('label')
        x_df = x_df.values / 255  # Normalize values from 0 to 1
        x_df = x_df.reshape(-1, IMG_CHS, IMG_WIDTH, IMG_HEIGHT)
        self.xs = torch.tensor(x_df).float().to(device)
        self.ys = torch.tensor(y_df).to(device)

    def __getitem__(self, idx):
        x = self.xs[idx]
        y = self.ys[idx]
        return x, y

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

```
BATCH_SIZE = 32

train_df = pd.read_csv("/content/drive/MyDrive/Spring 2025 STA138 Final Project/sign_mnist_train.csv")
valid_df = pd.read_csv("/content/drive/MyDrive/Spring 2025 STA138 Final Project/sign_mnist_valid.csv")


train_data = MyDataset(train_df)
train_loader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True)
train_N = len(train_loader.dataset)

valid_data = MyDataset(valid_df)
valid_loader = DataLoader(valid_data, batch_size=BATCH_SIZE)
valid_N = len(valid_loader.dataset)
```

```
batch = next(iter(train_loader))
batch
batch[0].shape
```

```
torch.Size([32, 1, 28, 28])
```

2. Creating the Model

SPECIFY your model below.

- A basic convolutionary neural network model is provided.
- You should change the model architecture (you can change the structure of the convolutionary neural network or use a feedforward neural network) or change the model parameters to get a better performance.

```
IMG_CHS = 1 # this is the number of channels for the image
n_classes = 24

model = nn.Sequential(
    # First convolutional block
    nn.Conv2d(IMG_CHS, 32, kernel_size=3, padding=1),  # Output: 32x28x28
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d(2),  # Output: 32x14x14

    # Second convolutional block
    nn.Conv2d(32, 64, kernel_size=3, padding=1),  # Output: 64x14x14
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(2),  # Output: 64x7x7

    # Third convolutional block
    nn.Conv2d(64, 128, kernel_size=3, padding=1),  # Output: 128x7x7
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(2),  # Output: 128x3x3

    # Classifier
    nn.Flatten(),
    nn.Linear(128*3*3, 256),
    nn.Dropout(0.5),
    nn.ReLU(),
    nn.Linear(256, n_classes)
)
model.to(device)
```

```
Sequential(
  (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Flatten(start_dim=1, end_dim=-1)
  (13): Linear(in_features=1152, out_features=256, bias=True)
  (14): Dropout(p=0.5, inplace=False)
  (15): ReLU()
  (16): Linear(in_features=256, out_features=24, bias=True)
)
```

3. Model Training

Do not change anything below.

```
loss_function = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters())

train_N = len(train_loader.dataset)
valid_N = len(valid_loader.dataset)

def get_batch_accuracy(output, y, N):
    pred = output.argmax(dim=1, keepdim=True)
    correct = pred.eq(y.view_as(pred)).sum().item()
    return correct / N

def train():
    loss = 0
    accuracy = 0

    model.train()
    for x, y in train_loader:
        x, y = x.to(device), y.to(device)
        output = model(x)
        optimizer.zero_grad()
        batch_loss = loss_function(output, y)
        batch_loss.backward()
        optimizer.step()

        loss += batch_loss.item()
        accuracy += get_batch_accuracy(output, y, train_N)
    print('Train - Loss: {:.4f} Accuracy: {:.4f}'.format(loss, accuracy))


def validate():
    loss = 0
    accuracy = 0

    model.eval()
    with torch.no_grad():
        for x, y in valid_loader:
            x, y = x.to(device), y.to(device)
            output = model(x)

            loss += loss_function(output, y).item()
            accuracy += get_batch_accuracy(output, y, valid_N)
    print('Valid - Loss: {:.4f} Accuracy: {:.4f}'.format(loss, accuracy))
```

Run the following cell the check the performance of your model.

```
epochs = 10

for epoch in range(epochs):
    print('Epoch: {}'.format(epoch))
    train()
    validate()
```

```
Epoch: 0
Train - Loss: 346.5461 Accuracy: 0.8797
Valid - Loss: 25.9504 Accuracy: 0.9626
Epoch: 1
Train - Loss: 27.5646 Accuracy: 0.9910
Valid - Loss: 31.7842 Accuracy: 0.9501
Epoch: 2
Train - Loss: 17.6828 Accuracy: 0.9942
Valid - Loss: 41.8435 Accuracy: 0.9491
Epoch: 3
Train - Loss: 17.2053 Accuracy: 0.9938
Valid - Loss: 25.6079 Accuracy: 0.9696
Epoch: 4
Train - Loss: 12.4237 Accuracy: 0.9960
Valid - Loss: 80.1071 Accuracy: 0.9052
Epoch: 5
Train - Loss: 10.4091 Accuracy: 0.9964
Valid - Loss: 31.0551 Accuracy: 0.9711
Epoch: 6
Train - Loss: 10.1592 Accuracy: 0.9964
Valid - Loss: 44.7457 Accuracy: 0.9617
Epoch: 7
Train - Loss: 6.0437 Accuracy: 0.9981
Valid - Loss: 19.5413 Accuracy: 0.9757
Epoch: 8
Train - Loss: 9.0102 Accuracy: 0.9967
Valid - Loss: 40.8513 Accuracy: 0.9555
Epoch: 9
Train - Loss: 5.7922 Accuracy: 0.9980
Valid - Loss: 20.7844 Accuracy: 0.9763
```

Submission Instructions

After you have chosen your final model (you work in groups, but each of you need to submit a copy).

- Please click `Runtime` -> `Restart session and run all`.
- Click `File` -> `Print` -> `Save as PDF`.
- Change the PDF file name to your `Firstname_LastName.pdf` (say `Donald_Trump.pdf`) and upload it to the box.ucdavis.edu submission folder.
- Copy the output from the previous cell, for example,

```
Epoch: 0
Train - Loss: 1053.0395 Accuracy: 0.6117
Valid - Loss: 165.4801 Accuracy: 0.7539
Epoch: 1
Train - Loss: 402.2355 Accuracy: 0.8408
Valid - Loss: 159.5908 Accuracy: 0.7818
Epoch: 2
Train - Loss: 271.9850 Accuracy: 0.8895
Valid - Loss: 160.4108 Accuracy: 0.8069
Epoch: 3
Train - Loss: 225.0550 Accuracy: 0.9048
Valid - Loss: 195.5503 Accuracy: 0.7927
Epoch: 4
Train - Loss: 185.9369 Accuracy: 0.9219
Valid - Loss: 238.5131 Accuracy: 0.7789
Epoch: 5
Train - Loss: 161.1346 Accuracy: 0.9307
Valid - Loss: 189.5530 Accuracy: 0.8211
Epoch: 6
```

```
Train - Loss: 150.2623 Accuracy: 0.9355
Valid - Loss: 232.6281 Accuracy: 0.8020
Epoch: 7
Train - Loss: 135.8071 Accuracy: 0.9416
Valid - Loss: 252.1603 Accuracy: 0.8246
Epoch: 8
Train - Loss: 124.8701 Accuracy: 0.9459
Valid - Loss: 240.1892 Accuracy: 0.8194
Epoch: 9
Train - Loss: 115.9816 Accuracy: 0.9500
Valid - Loss: 264.5139 Accuracy: 0.8105
```

And put it in a `txt` file named `submission.txt`.

- Submit the `submission.txt` file to gradescope. Your grade will show up within one minute after you make the submission on gradescope. If not, please check your file and make sure the format is correct (no additional lines or spaces).

The grading will be based on the performance of your final model. The key metric is min(Train Accuracy, Valid Accuracy) - abs(Train Accuracy - Valid Accuracy).

```
min(0.9912, 0.9999) - abs(0.9912- 0.9999)
```

```
0.9824999999999999
```

```
Start coding or generate with AI.
```