

In [ ]:

```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
from torchvision.datasets import FashionMNIST
from torchvision.transforms import ToTensor
from torchvision.utils import save_image
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
import os
from torchvision.datasets import ImageFolder
import torchvision.transforms as tt
from tqdm import tqdm
from torch.utils.data.dataloader import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:

```
! pip install -q kaggle
from google.colab import files
files.upload()
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

[Choose Files](#) No file chosen Upload widget is only available when the cell has been executed in the current browser session.

Please rerun this cell to enable.

Saving kaggle.json to Kaggle.json

In [ ]:

```
! kaggle datasets download grassknotted/asl-alphabet
! unzip asl-alphabet.zip
```

from google.colab import files

files.upload()

```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```







[illegible]



[illegible]







[illegible]



[illegible]

```

else:
    return torch.device('cpu')

def to_device(data, device):
    if isinstance(data, (list, tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

device = get_device()

In [ ]:
    batch_size=64

    train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
    val_dl = DataLoader(val_ds, batch_size=2, num_workers=4, pin_memory=True)
    test_dl = DataLoader(test_ds, batch_size=2, num_workers=4, pin_memory=True)

    /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create
    4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than
    what this DataLoader is going to create. Please be aware that excessive worker creation might not DataLoada
    r running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
    (cpuset_checked)

In [ ]:
    from torchvision.utils import make_grid

    def show_batch(dl):
        for images, labels in dl:
            fig, ax = plt.subplots(figsize=(16, 8))
            ax.set_title('||: ax.set_title(||:
            ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
            break

    show_batch(train_dl)

```

```

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will only
have worker processes in total. Our suggested max number of worker in current system is 2, which is smaller
than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader
a running slow or even freeze, lower the number of worker to avoid potential slowness/freeze if necessary.
  (torch.dataloader)
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
  (torch.dataloader)

```



```
In [ ]: def accuracy(outputs, labels):
    preds = torch.max(outputs, dim=1)
    feature_torch_tensor(torch.sum(preds == labels).item() / len(preds))
```

```
class ASLDeepNeuralNetwork(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(input_size, 128),
            nn.ReLU(),
            nn.Linear(128, output_size)
        )

    def forward(self, xb):
        return self.network(xb.view(xb.size(0), -1))

if __name__ == '__main__':
    torch.manual_seed(1)
```

```
def validation_step(self, batch):
    images, labels = batch
    out = self(images)
    loss = F.cross_entropy(out, labels)
    return loss

def validation_epoch_end(self, outputs):
```

```

        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean() # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean() # Combine accuracies
        return 'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()

def epoch_end(self, epoch, result):
    print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
        epoch, result['train_loss'], result['val_loss'], result['val_acc']))

#torch.no_grad()
def evaluate(model, val_dli):
    model.eval()
    for i, (inputs, targets) in enumerate(val_dli):
        outputs = model(inputs)
        batch_loss = torch.nn.Loss()
        batch_acc = torch.nn.AverageMeter()
        for j, (input, target) in enumerate(zip(inputs, targets)):
            output = model(input)
            loss = batch_loss(output, target)
            acc = batch_acc.update(output, target)
        batch_loss_val = batch_loss.item()
        batch_acc_val = batch_acc.avg
        batch_losses.append(batch_loss_val)
        batch_accs.append(batch_acc_val)
    epoch_loss = torch.stack(batch_losses).mean()
    epoch_acc = torch.stack(batch_accs).mean()
    return 'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()

```

```

        train_loader.load_state_dict(torch.load(train_path))
        return model, validation_epoch_end(outputs)

def fit(epochs, lr, model, train_dl, val_dl, opt_func=torch.optim.RMSProp):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in tqdm(train_dl):
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

```

```
optimizer.step()
optimizer.zero_grad()

# Validation phase
result = evaluate(model, val_dl)
result['train_loss'] = torch.stack(train_losses).mean().item()
model.epoch_end(epoch, result)
history.append(result)

return history

input_size = 3*32*32
output_size = 29
CNN = AlexNetWrapperNetwork(input_size, output_size)
train_dl = DeviceDataLoader(train_dl, device)
```

```

test_dl = DataLoader(data_loader_val, device=device, num_workers=num_workers)
device(DNN, device)
num_epochs = 10
opt_func = torch.optim.Adam
ls = 0.0001
history = fit(num_epochs, lr, DNN, train_dl, val_dl, opt_func)
torch.save(DNN.state_dict(), 'DNN.pth')

```

OK: | 0/1032 [00:00<, 717s]/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number of worker in your system is 2, which is below the maximum of 10. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the number of worker to avoid potential

```

[0] train_loss: 1.6879, val_loss: 1.6879, val_acc: 0.4533, epoch: 0
cpusat_checked)
100% |#####| 1032/1032 [01:09<00:00, 14.97r/s]
Epoch [0], train_loss: 2.1197, val_loss: 1.9398, val_acc: 0.4533
100% |#####| 1032/1032 [01:12<00:00, 14.16r/s]
Epoch [1], train_loss: 1.7223, val_loss: 1.5327, val_acc: 0.5779
100% |#####| 1032/1032 [01:09<00:00, 14.86r/s]
Epoch [2], train_loss: 1.3957, val_loss: 1.2867, val_acc: 0.6500
100% |#####| 1032/1032 [01:14<00:00, 13.83r/s]
Epoch [3], train_loss: 1.1819, val_loss: 1.1384, val_acc: 0.6968
100% |#####| 1032/1032 [01:12<00:00, 14.21r/s]
Epoch [4], train_loss: 1.0241, val_loss: 0.9707, val_acc: 0.7421

```

```
EPOCH [0], train_loss= 0.9012, val_loss= 0.8714, val_acc= 0.7696
100% |██████████| 1032/1032 [01:08<00:00, 15.12it/s]
EPOCH [0], train_loss= 0.8033, val_loss= 0.7872, val_acc= 0.7917
100% |██████████| 1032/1032 [01:11<00:00, 14.89it/s]
EPOCH [1], train_loss= 0.7176, val_loss= 0.7108, val_acc= 0.8184
100% |██████████| 1032/1032 [01:09<00:00, 14.94it/s]
EPOCH [1], train_loss= 0.6480, val_loss= 0.6535, val_acc= 0.8357
100% |██████████| 1032/1032 [01:07<00:00, 15.27it/s]
EPOCH [1], train_loss= 0.5884, val_loss= 0.5949, val_acc= 0.8640
```

```
In [ ]: def plot_accuracies(history):
```

```

        accuracies = [x['val_acc'] for x in history]
        plt.plot(accuracies, "--")
        plt.xlabel("epoch")
        plt.ylabel("accuracy")
        plt.title("Accuracy vs. No. of epochs");
        plt.show()

    def plot_losses(history):
        train_losses = [x.get('train_loss') for x in history]
        val_losses = [x['val_loss'] for x in history]
        plt.plot(train_losses, "-b")
        plt.plot(val_losses, "--r")
        plt.xlabel("epoch")
        plt.ylabel("loss")

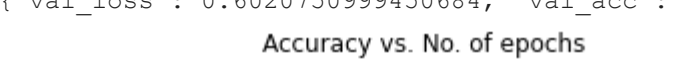
```

```
plt.legend(("Training", "Validation"))
plt.title("Loss vs. No. of epochs")
plt.show()

print(evaluate(DNN, test_dl))
plot_accuracies(history)
plot_losses(history)

#usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create
4 worker processes in total. Our suggested max number of worker processes in current system is 2, which is smaller than
5 the number of DataLoaders that are going to be created. Please be aware that excessive worker creation might get DataLoader
6 running slow or even freeze, lower the worker number to avoid potential slowness/freezes if necessary.
7 (check the docs)
8
```


["val\_loss": 0.6020750999450684, "val\_acc": 0.8466619253158569]



Epoch	Accuracy
0	0.55
1	0.58
2	0.62
3	0.65
4	0.68
5	0.70
6	0.72
7	0.74
8	0.76
9	0.78
10	0.79
11	0.80
12	0.81
13	0.82
14	0.83
15	0.85

The top graph shows training loss over 8 epochs. The x-axis is labeled 'epoch' and ranges from 0 to 8. The y-axis ranges from 0.40 to 0.50. The training loss starts at approximately 0.45 at epoch 0 and decreases to approximately 0.40 at epoch 8.

The bottom graph shows training and validation loss over 8 epochs. The x-axis is labeled 'epoch' and ranges from 0 to 8. The y-axis is labeled 'Loss vs. No. of epochs' and ranges from 1.75 to 2.50. The training loss (blue line with circles) starts at approximately 2.50 at epoch 0 and decreases to approximately 1.75 at epoch 8. The validation loss (red line with circles) starts at approximately 2.00 at epoch 0 and decreases to approximately 1.75 at epoch 8.



epoch	training loss	validation loss
0	1.50	1.40
1	1.30	1.20
2	1.15	1.05
3	1.05	0.95
4	0.95	0.90
5	0.85	0.80
6	0.80	0.75
7	0.75	0.70
8	0.70	0.65
9	0.65	0.60
10	0.60	0.65

```
In [ ]: def predict_single(input, target, model):
        predictions = model(input)
```

```
_, preds = torch.max(predictions, dim=-1)
print("Target:", target)
print("Prediction:", preds)

def show_example(img, label):
    print("Label: ", dataset.classes[label], "(" + str(label) + ")")
    plt.imshow(img.permute(1, 2, 0))

In [ ]:
for input, label in test_d:
    show_example(input.cpu(), label.cpu())
    negative = torch.tensor(0)
    negative = torch.tensor(0)
    negative = torch.tensor(0)
```

```

break
    _out_sample = torch.cat([_out_sample, _in_sample[_in_sample.size()[0]-1:_in_sample.size()[0]-1+1]], dim=0)

/usr/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create
4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller
than what this DataLoader is going to create. Please be aware that excessive worker creation might get
DataLoader slow down and possibly freeze your GPU. You can override this warning with
--warn-workers flag to silence this message. If you are unsure what this warning is good for, disable
cpuset_checked)
--input data to the valid range for inshaw with ROB data ([0..1] for floats or [0..255] for integers).
Label: 8 (tensor(1.77))
Target: tensor(117), device='cuda:0'
Prediction: tensor(117), device='cuda:0'
0

```