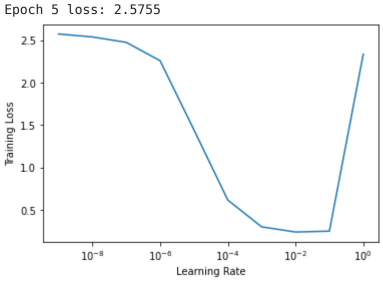
In [1]:
```
# Solution 2.1):

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np


# Define the batch size
batch_size = 64

# Define the transformations to be applied to the images
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)),
    transforms.Resize((32,32))
])

# Load the fashionMNIST dataset
train_set = datasets.FashionMNIST('./data', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True)

# ConvModule: a convolutional module in the above picture, consists a 2d convolutional layer, a 2d batchnorm layer, and a ReLU activation.
class ConvModule(nn.Module):
    def __init__(self, in_channels: int, out_channels: int, kernel_size, stride, padding='same'):
        super(ConvModule, self).__init__()
        self.conv2d = nn.Conv2d(
            in_channels, out_channels, kernel_size, stride=stride, padding=padding)

        self.batchnorm = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv2d(x)
        x = self.batchnorm(x)
        x = self.relu(x)

        return x

# InceptionModule: a inception module in the above picture, consists a convolution module with 1x1 filter,
# a convolution module with 3x3 filter, then concatenate these two outputs.
class InceptionModule(nn.Module):
    def __init__(self, in_channels, ch1x1, ch3x3):
        super(InceptionModule, self).__init__()

        self.conv1x1 = ConvModule(in_channels, ch1x1, (1, 1), 1)
        self.conv3x3 = ConvModule(in_channels, ch3x3, (3, 3), 1)

    def forward(self, x):
        out1 = self.conv1x1(x)
        out2 = self.conv3x3(x)
        x = torch.cat((out1, out2), 1)
        return x

# DownsampleModule: a downsample module in the above picture, consists a convolution module with 3x3 filter,
# a 2d maxpool layer, then concatenate these two outputs.
class DownsampleModule(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownsampleModule, self).__init__()

        self.conv3x3 = ConvModule(in_channels, out_channels, (3, 3), (2, 2), padding='valid')
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2)

    def forward(self, x):
        out1 = self.conv3x3(x)
        out2 = self.maxpool(x)

        #return out1
        x = torch.cat((out1, out2), 1)

        return x


# MiniGoogLeNet: the MiniGoogLeNet model. Input: input_channels * 32 * 32.
# When input_channels is 1, the input is a grayscale image. When input_channels is 3, the input is a RGB image.
# Output: a tensor with the shape of [-1, classes], where classes it the number of classes.

class MiniGoogLeNet(nn.Module):
    def __init__(self, classes, input_channels):
        super(MiniGoogLeNet, self).__init__()

        self.conv1 = ConvModule(input_channels, 96, kernel_size=(3, 3), stride=1) # input_channel is 3 if you want to deal with RGB image, 1 for grey scale image
        self.inception1 = InceptionModule(96, 32, 32)
        self.inception2 = InceptionModule(32+32, 32, 48)
        self.downsample1 = DownsampleModule(32+48, 80)

        self.inception3 = InceptionModule(80+80, 112, 48)
        self.inception4 = InceptionModule(112+48, 96, 64)
        self.inception5 = InceptionModule(96+64, 80, 80)
        self.inception6 = InceptionModule(80+80, 48, 96)
        self.downsample2 = DownsampleModule(48+96, 96)

        self.inception7 = InceptionModule(96+96, 176, 160)
        self.inception8 = InceptionModule(176+160, 176, 160)
        self.avgpool2d = nn.AvgPool2d(kernel_size=7)
        self.dropout = nn.Dropout2d(0.5)

        self.fc = nn.Linear(240, classes)
        #self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        x = self.conv1(x)
        #print(x.shape)
        x = self.inception1(x)
        x = self.inception2(x)
        x = self.downsample1(x)

        x = self.inception3(x)
        x = self.inception4(x)
        x = self.inception5(x)
```

```
        x = self.inception6(x)
        x = self.downsample2(x)

        x = self.avgpool2d(x)
        x = self.dropout(x)

        x = torch.flatten(x, 1)
        x = self.fc(x)
        #x = self.softmax(x), no need for softmax because PyTorch Cross Entropy Loss implemented softmax

        return x

# Define the learning rate candidates
learning_rate_candidates = [10**(-i) for i in range(10)]

# Define the learning rate candidates
learning_rate_candidates = [10**(-i) for i in range(10)]

# Train the model for 5 epochs for each learning rate candidate and store the results
losses = []
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
for lr in learning_rate_candidates:
    print(f'Training with learning rate {lr}')
    model = MiniGoogLeNet(10, 1).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
    for epoch in range(5):
        running_loss = 0.0
        for i, data in enumerate(train_loader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        epoch_loss = running_loss / len(train_loader)
        print(f'Epoch {epoch+1} loss: {epoch_loss:.4f}')
    losses.append(epoch_loss)

# Plot the losses as a function of the learning rate
plt.plot(learning_rate_candidates, losses)
plt.xscale('log')
plt.xlabel('Learning Rate')
plt.ylabel('Training Loss')
plt.show()

min_loss_index = losses.index(min(losses))
max_loss_index = losses.index(max(losses))
lrmin = learning_rate_candidates[min_loss_index]
lrmax = learning_rate_candidates[max_loss_index]
print(f'lrmin: {lrmin}')
print(f'lrmax: {lrmax}')
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz
  0%|          | 0/26421880 [00:00<?, ?it/s]
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
  0%|          | 0/29515 [00:00<?, ?it/s]
Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
  0%|          | 0/4422102 [00:00<?, ?it/s]
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
  0%|          | 0/5148 [00:00<?, ?it/s]
```

```
Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

Training with learning rate 1
Epoch 1 loss: 2.4475
Epoch 2 loss: 2.3377
Epoch 3 loss: 2.3393
Epoch 4 loss: 2.3403
Epoch 5 loss: 2.3368
Training with learning rate 0.1
Epoch 1 loss: 0.6276
Epoch 2 loss: 0.3635
Epoch 3 loss: 0.3105
Epoch 4 loss: 0.2753
Epoch 5 loss: 0.2508
Training with learning rate 0.01
Epoch 1 loss: 0.5694
Epoch 2 loss: 0.3545
Epoch 3 loss: 0.2952
Epoch 4 loss: 0.2661
Epoch 5 loss: 0.2406
Training with learning rate 0.001
Epoch 1 loss: 0.8012
Epoch 2 loss: 0.4481
Epoch 3 loss: 0.3725
Epoch 4 loss: 0.3299
Epoch 5 loss: 0.3014
Training with learning rate 0.0001
Epoch 1 loss: 1.5608
Epoch 2 loss: 0.9811
Epoch 3 loss: 0.7810
Epoch 4 loss: 0.6795
Epoch 5 loss: 0.6175
Training with learning rate 1e-05
Epoch 1 loss: 2.2330
Epoch 2 loss: 1.8995
Epoch 3 loss: 1.6914
Epoch 4 loss: 1.5528
Epoch 5 loss: 1.4458
Training with learning rate 1e-06
Epoch 1 loss: 2.5570
Epoch 2 loss: 2.4561
Epoch 3 loss: 2.3776
Epoch 4 loss: 2.3096
Epoch 5 loss: 2.2606
Training with learning rate 1e-07
Epoch 1 loss: 2.5137
Epoch 2 loss: 2.5013
Epoch 3 loss: 2.5007
Epoch 4 loss: 2.4887
Epoch 5 loss: 2.4767
Training with learning rate 1e-08
Epoch 1 loss: 2.5444
Epoch 2 loss: 2.5491
Epoch 3 loss: 2.5499
Epoch 4 loss: 2.5433
Epoch 5 loss: 2.5408
Training with learning rate 1e-09
Epoch 1 loss: 2.5761
Epoch 2 loss: 2.5769
Epoch 3 loss: 2.5747
Epoch 4 loss: 2.5764
Epoch 5 loss: 2.5755
```




```
lrmin: 0.01
lrmax: 1e-09
```

In [3]:

```python
#Solution 2.2):

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np


# Define the batch size
batch_size = 64

# Define the transformations to be applied to the images
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)),
    transforms.Resize((32,32))
])

# Load the fashionMNIST dataset
train_set = datasets.FashionMNIST('./data', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True)

test_set = datasets.FashionMNIST('./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=True)

# ConvModule: a convolutional module in the above picture, consists a 2d convolutional layer, a 2d batchnorm layer, and a ReLU activation.
class ConvModule(nn.Module):
    def __init__(self, in_channels: int, out_channels: int, kernel_size, stride, padding='same'):
        super(ConvModule, self).__init__()
        self.conv2d = nn.Conv2d(
            in_channels, out_channels, kernel_size, stride=stride, padding=padding)
```

```
        self.batchnorm = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv2d(x)
        x = self.batchnorm(x)
        x = self.relu(x)

        return x

# InceptionModule: a inception module in the above picture, consists a convolution module with 1x1 filter,
# a convolution module with 3x3 filter, then concatenate these two outputs.
class InceptionModule(nn.Module):
    def __init__(self, in_channels, ch1x1, ch3x3):
        super(InceptionModule, self).__init__()

        self.conv1x1 = ConvModule(in_channels, ch1x1, (1, 1), 1)
        self.conv3x3 = ConvModule(in_channels, ch3x3, (3, 3), 1)

    def forward(self, x):
        out1 = self.conv1x1(x)
        out2 = self.conv3x3(x)
        x = torch.cat((out1, out2), 1)
        return x

# DownsampleModule: a downsample module in the above picture, consists a convolution module with 3x3 filter,
# a 2d maxpool layer, then concatenate these two outputs.
class DownsampleModule(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownsampleModule, self).__init__()

        self.conv3x3 = ConvModule(in_channels, out_channels, (3, 3), (2, 2), padding='valid')
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2)

    def forward(self, x):
        out1 = self.conv3x3(x)
        out2 = self.maxpool(x)

        #return out1
        x = torch.cat((out1, out2), 1)

        return x


# MiniGoogLeNet: the MiniGoogLeNet model. Input: input_channels * 32 * 32.
# When input_channels is 1, the input is a grayscale image. When input_channels is 3, the input is a RGB image.
# Output: a tensor with the shape of [-1, classes], where classes it the number of classes.

class MiniGoogLeNet(nn.Module):
    def __init__(self, classes, input_channels):
        super(MiniGoogLeNet, self).__init__()

        self.conv1 = ConvModule(input_channels, 96, kernel_size=(3, 3), stride=1) # input_channel is 3 if you want to deal with RGB image, 1 for grey scale image
        self.inception1 = InceptionModule(96, 32, 32)
        self.inception2 = InceptionModule(32+32, 32, 48)
        self.downsample1 = DownsampleModule(32+48, 80)

        self.inception3 = InceptionModule(80+80, 112, 48)
        self.inception4 = InceptionModule(112+48, 96, 64)
        self.inception5 = InceptionModule(96+64, 80, 80)
        self.inception6 = InceptionModule(80+80, 48, 96)
        self.downsample2 = DownsampleModule(48+96, 96)

        self.inception7 = InceptionModule(96+96, 176, 160)
        self.inception8 = InceptionModule(176+160, 176, 160)
        self.avgpool2d = nn.AvgPool2d(kernel_size=7)
        self.dropout = nn.Dropout2d(0.5)

        self.fc = nn.Linear(240, classes)
        #self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        x = self.conv1(x)
        #print(x.shape)
        x = self.inception1(x)
        x = self.inception2(x)
        x = self.downsample1(x)

        x = self.inception3(x)
        x = self.inception4(x)
        x = self.inception5(x)
        x = self.inception6(x)
        x = self.downsample2(x)

        x = self.avgpool2d(x)
        x = self.dropout(x)

        x = torch.flatten(x, 1)
        x = self.fc(x)
        #x = self.softmax(x), no need for softmax because PyTorch Cross Entropy Loss implemented softmax

        return x

lrmin = 0.01
lrmax = 1e-09
step_size = 2000
mode = 'exp_range'

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = MiniGoogLeNet(10, 1).to(device)

# Create the cyclic learning rate scheduler
optimizer = optim.SGD(model.parameters(), lr=lrmax, momentum=0.9)
scheduler = optim.lr_scheduler.CyclicLR(optimizer, base_lr=lrmin,  max_lr=lrmax, step_size_up=step_size, mode=mode, gamma=0.99)
criterion = nn.CrossEntropyLoss()

# Train the model for 10 epochs using the cyclic learning rate policy
train_losses = []
train_accs = []
val_losses = []
val_accs = []

for epoch in range(10):
    model.train()
```

```
    running_loss = 0.0
    correct = 0
    total = 0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        scheduler.step()  # update the learning rate
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    epoch_train_loss = running_loss / len(train_loader)
    epoch_train_acc = correct / total
    train_losses.append(epoch_train_loss)
    train_accs.append(epoch_train_acc)
    print(f'Train Epoch {epoch+1} loss: {epoch_train_loss:.4f} accuracy: {epoch_train_acc:.4f}')

    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    epoch_val_loss = running_loss / len(test_loader)
    epoch_val_acc = correct / total
    val_losses.append(epoch_val_loss)
    val_accs.append(epoch_val_acc)
    print(f'Validation Epoch {epoch+1} loss: {epoch_val_loss:.4f} accuracy: {epoch_val_acc:.4f}')

# Plot the train/validation loss curve
plt.plot(train_losses, label='train')
plt.plot(val_losses, label='validation')
plt.title('Train/Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot the train/validation accuracy curve
plt.plot(train_accs, label='train')
plt.plot(val_accs, label='validation')
plt.title('Train/Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Train Epoch 1 loss: 0.5649 accuracy: 0.7940
Validation Epoch 1 loss: 0.3541 accuracy: 0.8745
Train Epoch 2 loss: 0.3512 accuracy: 0.8739
Validation Epoch 2 loss: 0.3390 accuracy: 0.8800
Train Epoch 3 loss: 0.2937 accuracy: 0.8958
Validation Epoch 3 loss: 0.3256 accuracy: 0.8886
Train Epoch 4 loss: 0.2655 accuracy: 0.9059
Validation Epoch 4 loss: 0.3259 accuracy: 0.8832
Train Epoch 5 loss: 0.2405 accuracy: 0.9149
Validation Epoch 5 loss: 0.2414 accuracy: 0.9142
Train Epoch 6 loss: 0.2221 accuracy: 0.9219
Validation Epoch 6 loss: 0.2316 accuracy: 0.9170
Train Epoch 7 loss: 0.2075 accuracy: 0.9262
Validation Epoch 7 loss: 0.2264 accuracy: 0.9210
Train Epoch 8 loss: 0.1934 accuracy: 0.9312
Validation Epoch 8 loss: 0.2157 accuracy: 0.9244
Train Epoch 9 loss: 0.1779 accuracy: 0.9355
Validation Epoch 9 loss: 0.2080 accuracy: 0.9276
Train Epoch 10 loss: 0.1716 accuracy: 0.9385
Validation Epoch 10 loss: 0.2382 accuracy: 0.9177
```