# 西安电子科技大学网络与信息安全学院
# 信息与内容安全 实验报告

班　　级: 1918039

学　　号: 19180300034

姓　　名: 肖畅

电子邮箱: 2857627720@qq.com

指导教师: 彭春蕾

2022 年 5 月 4 日

实验题目：对抗样本攻击实验和虚假人脸检测实验

实验摘要：

对抗样本攻击实验

1.在对攻击者知识的假设方面，**FGSM** 是白盒攻击，它假设攻击者拥有模型的全部知识和访问权限，包括架构、输入、输出和权重；在目标方面，**FGSM** 是无目标攻击，它不关心对抗样本经过模型后新分类是什么，只要分类错误即可。

2.我们在学习简单的神经网络时，学习过梯度下降算法，即基于反向传播的梯度调整权重，使损失函数最小化。而 **FGSM** 是基于同样的反向传播的梯度，通过调整输入数据使得损失函数最大化。

3.**FGSM** 的思路很简单：网络参数**θ**保持不变，损失函数 **L** 对原始输入 **x** 求偏导，若值为正数，取 **1**；值为负数，取**-1**，得到“梯度符号”。

4.因为希望产生对抗样本的速度更快，且 **FGSM** 是典型的无穷范数攻击，所以我们将小于阈值的扰动直接提升到阈值，而超出阈值的部分由约束进行截断。

5.随着 **epsilon** 的增加，测试精度会降低，但扰动变得更容易察觉。实际上，攻击者必须在准确性降低和可感知性之间进行权衡。

虚假人脸检测实验

1.通过 **GAN** 生成对抗网络及其扩展，我们可以伪造图像或视频。

2.视觉内容伪造方法可以分为有目标身份伪造和无目标身份伪造；有目标身份伪造又可以分为人脸替换和人脸编辑。人脸编辑又细分为属性编辑、表情重演和跨模态人脸编辑。

3.身份伪造的检测方法可以分类为空域线索、时域线索、面向未知伪造类型的泛化能力研究和面向对抗样本攻击的可信检测研究。

4.近期比较先进的算法有 **MesoNet**、**DeeperForensics** 和 **M2TR** 等。

题目描述：

对抗样本攻击实验

在白盒环境下，通过求出模型对原始输入的导数，然后用符号函数得到其具体的梯度方向，接着乘以一个步长，得到的“扰动”加在原来的输入上就得到了在 **FGSM** 攻击下的对抗样本。

$$perturbed\_image = image + epsilon * sign(data\_grad) = x + \epsilon * sign(\nabla_x J(\theta, \mathbf{x}, y))$$

原因:图像本身是高维特征，在随机方向修改 x 的话，一定程度内类别不会变化；但总是存在某个特定的具体方向，修改 x 后类别会急剧变化。对抗攻击的目标就是去寻找这个具体方向。

对 FGSM 而言，扰动造成的影响在神经网络中还会像滚雪球一样越滚越大，对于线性模型更是如此。而目前神经网络中倾向于使用 Relu 这种类线性的激活函数，使得网络整体趋近于线性。

虚假人脸检测实验

给定一个人脸数据集，其中包含 1999 张真实人脸，1999 张虚假人脸。将其中 500 张真实人脸和 500 张虚假人脸作为训练集，其余作为测试集。 根据给定数据集训练一个虚假人脸检测器，该检测器本质就是一个二分类分类器。

这要求我们先重新定义一个 Dataset 类，读取图片并将图片按比例划分为训练集和测试集，使用 Dataloader 加载数据。设计一个合适的 CNN 网络模型，定义损失函数和优化器，最后进行训练及测试。

# 实验内容

# 对抗样本攻击实验

## 一、实验基本原理及步骤

### 1.定义输入和受攻击的模型

```
epsilons = [0, .05, .1, .15, .2, .25, .3]
pretrained_model = "data/lenet_mnist_model.pth"
use_cuda=False

# LeNet Model definition
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

# MNIST Test dataset and dataloader declaration
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, download=True, transform=transforms.Compose([
            transforms.ToTensor(),
            ])),
        batch_size=1, shuffle=True)

# Define what device we are using
print("CUDA Available: ",torch.cuda.is_available())
device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")
# Initialize the network
model = Net().to(device)
# Load the pretrained model
model.load_state_dict(torch.load(pretrained_model, map_location='cpu'))
# Set the model in evaluation mode. In this case this is for the Dropout layers
model.eval()
```

设置了一些 **epsilon** 的取值，最后通过可视化分析，可以认识到降低分类准确性和

确保不可感知性之间的权衡。加载预训练的模型，设置为 **eval** 模式。

网络模型如图

```
CUDA Available:  False
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
```

## 2.FGSM 攻击

```
# FGSM attack code
def fgsm_attack(image, epsilon, data_grad):
    # Collect the element-wise sign of the data gradient
    sign_data_grad = data_grad.sign()
    # Create the perturbed image by adjusting each pixel of the input image
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # Return the perturbed image
    return perturbed_image
```

使用 **sign** 函数，将 **Loss** 对 **x** 求偏导的梯度进行符号化；乘上 **epsilon** 后生成对抗样本；最后做一个剪裁的工作，将 **torch.clamp** 内部大于 **1** 的数值变为 **1**，小于 **0** 的数值变为 **0**，防止越界。

## 3.测试功能

```
def test( model, device, test_loader, epsilon ):
    correct = 0
    adv_examples = []
    # Loop over all examples in test set
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        # Set requires_grad attribute of tensor. Important for Attack
        data.requires_grad = True
        # Forward pass the data through the model
        output = model(data)
        init_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
        # If the initial prediction is wrong, dont bother attacking, just move on
        if init_pred.item() != target.item():
            continue
        # Calculate the loss
        loss = F.nll_loss(output, target)
        # Zero all existing gradients
        model.zero_grad()
```

对于模型本身就无法正确分类的样本，我们不做处理。

```
        # Calculate gradients of model in backward pass
        loss.backward()
        # Collect datagrad
        data_grad = data.grad.data
        # Call FGSM Attack
        perturbed_data = fgsm_attack(data, epsilon, data_grad)
        # Re-classify the perturbed image
        output = model(perturbed_data)
        # Check for success
        final_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
        if final_pred.item() == target.item():
            correct += 1
            # Special case for saving 0 epsilon examples
            if (epsilon == 0) and (len(adv_examples) < 5):
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )
        else:
            # Save some adv examples for visualization later
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )

    # Calculate final accuracy for this epsilon
    final_acc = correct/float(len(test_loader))
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
    # Return the accuracy and an adversarial example
    return final_acc, adv_examples
```

**loss.backward()**函数的作用是根据 **loss** 来计算网络参数的梯度，其中包括损失函数关于 **x** 的偏导，获得其数值，构造对抗样本，并记录成功案例。

## 4.运行攻击

```
accuracies = []
examples = []

# Run test for each epsilon
for eps in epsilons:
    acc, ex = test(model, device, test_loader, eps)
    accuracies.append(acc)
    examples.append(ex)
```

我们为每个 **epsilon** 值运行一个完整的测试步骤。对于每个 **epsilon**，我们还保存最终准确度和一些成功的对抗性示例。

## 5.精度与 Epsilon

```
plt.figure(figsize=(5,5))
plt.plot(epsilons, accuracies, "*-")
plt.yticks(np.arange(0, 1.1, step=0.1))
```

```
plt.xticks(np.arange(0, .35, step=0.05))
plt.title("Accuracy vs Epsilon")
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.show()
```

展示随着 **epsilon** 的增加，测试精度的变化情况。

## 6.对抗样本示例

```
# Plot several examples of adversarial samples at each epsilon
cnt = 0
plt.figure(figsize=(8,10))
for i in range(len(epsilons)):
    for j in range(len(examples[i])):
        cnt += 1
        plt.subplot(len(epsilons),len(examples[0]),cnt)
        plt.xticks([], [])
        plt.yticks([], [])
        if j == 0:
            plt.ylabel("Eps: {}".format(epsilons[i]), fontsize=14)
        orig,adv,ex = examples[i][j]
        plt.title("{} -> {}".format(orig, adv))
        plt.imshow(ex, cmap="gray")
plt.tight_layout()
plt.show()
```

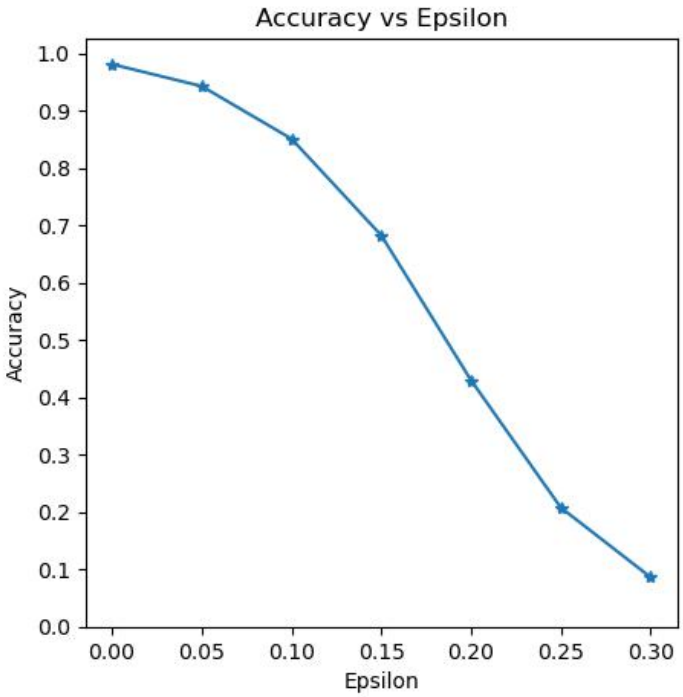
可视化分析，直观感受扰动是否能被肉眼观察，以及人类能否正确识别扰动图像的类别。

# 二、实验结果

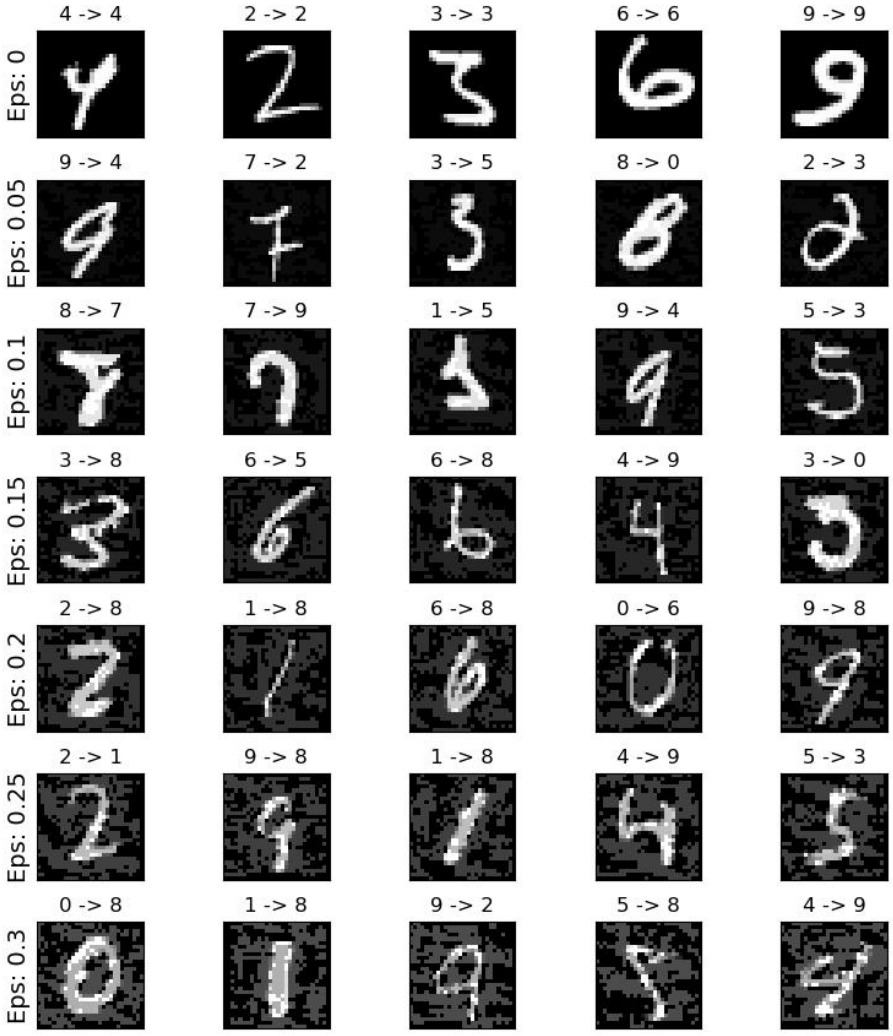
## 1.不同 epsilon 下的测试精度

```
(base) peer@ubuntu:~/Desktop/信息与内容安全$ cd /home/peer/
thon /home/peer/.vscode/extensions/ms-python.python-2022.4.1
top/信息与内容安全/FGSM.py
CUDA Available:  False
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.05   Test Accuracy = 9426 / 10000 = 0.9426
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.15   Test Accuracy = 6826 / 10000 = 0.6826
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.25   Test Accuracy = 2082 / 10000 = 0.2082
Epsilon: 0.3    Test Accuracy = 869 / 10000 = 0.0869
```

## 2.Accuracy vs Epsilon


Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30

## 3.对抗样本示例


Eps: 0
4 -> 4
2 -> 2
3 -> 3
6 -> 6
9 -> 9
Eps: 0.05
9 -> 4
7 -> 2
3 -> 5
8 -> 0
2 -> 3
Eps: 0.1
8 -> 7
7 -> 9
1 -> 5
9 -> 4
5 -> 3
Eps: 0.15
3 -> 8
6 -> 5
6 -> 8
4 -> 9
3 -> 0
Eps: 0.2
2 -> 8
1 -> 8
6 -> 8
0 -> 6
9 -> 8
Eps: 0.25
2 -> 1
9 -> 8
1 -> 8
4 -> 9
5 -> 3
Eps: 0.3
0 -> 8
1 -> 8
9 -> 2
5 -> 8
4 -> 9

三、实验结果的分析

随着 **epsilon** 的增加，测试精度不断降低。这是因为更大的 **epsilon** 意味着我们朝着使损失最大化的方向迈出了更大的一步，但这种下降并不是线性的。

第一行是 **epsilon=0** 表示没有扰动的原始"干净"图像的示例，可以发现，扰动在 **epsilon=0.15** 开始变得明显，并且在 **epsilon=0.3** 后变得非常明显。这意味着我们要在降低分类准确性和尽可能确保不可感知性之间做出权衡。

虚假人脸检测实验

一、实验基本原理及步骤

**1.**定义超参数

```
# Set random seed for reproducibility
manualSeed = random.randint(1, 10000)
print("Random Seed: ", manualSeed)
random.seed(manualSeed)
torch.manual_seed(manualSeed)

workers = 2
batch_size = 32
image_size = 200
num_epochs = 30
lr = 0.0002
beta1 = 0.5
ngpu = 0
```

设置 **batch_size**、图片的尺寸、训练次数和学习率等超参数。

**2.**继承 **Dataset** 类并重写关键方法

```
# define database class
class MyDataSet(Dataset):
    def __init__(self, images_path: list, images_class: list, transform=None):
        self.images_path = images_path
        self.images_class = images_class
        self.transform = transform

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

    def __getitem__(self, item):
        img = Image.open(self.images_path[item])

        if img.mode != 'RGB':
            raise ValueError("image: {} isn't RGB mode.".format(self.images_path[item]))
        label = self.images_class[item]
```

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

        return img, label
```

**pytorch 的 dataset 类有两种：Map-style datasets 和 Iterable-style datasets。前者是我们常用的结构，而后者是当数据集难以（或不可能）进行随机读取时使用。在这里我们实现 Map-style dataset。**

**继承 torch.utils.data.Dataset 后，需要重写的方法有：__len__与__getitem__方法，其中__len__方法需要返回所有数据的数量，而__getitem__则是要依照给出的数据索引获取对应的 tensor 类型的 Sample，除了这两个方法以外，一般还需要实现__init__方法来初始化一些变量。**

**3.将图片按比例划分为训练集和测试集**

```
# Split into training set and validation set
def read_split_data(root: str, val_rate: float = 0.75):      # modify the val_rate to change the rate of validation
    random.seed(0)
    assert os.path.exists(root), "dataset root: {} does not exist.".format(root)
    classes = [cla for cla in os.listdir(root) if os.path.isdir(os.path.join(root, cla))]
    classes.sort()
    class_indices = dict((k, v) for v, k in enumerate(classes))
    json_str = json.dumps(dict((val, key) for key, val in class_indices.items()), indent=4)
    with open('class_indices.json', 'w') as json_file:
        json_file.write(json_str)
    train_images_path = []
    train_images_label = []
    val_images_path = []
    val_images_label = []
    every_class_num = []
    supported = [".jpg", ".JPG", ".png", ".PNG"]
    for cla in classes:
        cla_path = os.path.join(root, cla)
        images = [os.path.join(root, cla, i) for i in os.listdir(cla_path)
                  if os.path.splitext(i)[-1] in supported]
        image_class = class_indices[cla]
        every_class_num.append(len(images))
        val_path = random.sample(images, k=int(len(images) * val_rate))
        for img_path in images:
            if img_path in val_path:
                val_images_path.append(img_path)
                val_images_label.append(image_class)
            else:
                train_images_path.append(img_path)
                train_images_label.append(image_class)
```

```
    print("{} images were found in the dataset.".format(sum(every_class_num)))
    print("{} images for training.".format(len(train_images_path)))
    print("{} images for validation.".format(len(val_images_path)))

    return train_images_path, train_images_label, val_images_path, val_images_label
```

输出如下

```
(base) peer@ubuntu:~/Desktop/信息与内容安全$ cd /home/peer/Desktop/虚
on /home/peer/.vscode/extensions/ms-python.python-2022.4.1/pythonFiles/
p/虚假人脸检测/fakeface.py
Random Seed:  4596
3998 images were found in the dataset.
1000 images for training.
2998 images for validation.
```

## 4.通过 transforms 函数来构造图像预处理序列

```
train_images_path, train_images_label, val_images_path, val_images_label = read_split_data(r"/home/peer/Desktop/
虚假人脸检测/CNN_synth_testset")
data_transform = {
    "train": transforms.Compose([
                                transforms.Resize(image_size),
                                transforms.CenterCrop(image_size),
                                transforms.ToTensor(),
                                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])]),
    "val": transforms.Compose([
                               transforms.Resize(image_size),
                               transforms.CenterCrop(image_size),
                               transforms.ToTensor(),
                               transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])])}
```

实现图像裁剪、将图片转成 Tensor 和把数值 normalize 到[0,1]等操作。

## 5.使用 Dataloader 加载数据

```
train_dataset = MyDataSet(images_path=train_images_path,
                          images_class=train_images_label,
                          transform=data_transform["train"])

val_dataset = MyDataSet(images_path=val_images_path,
                        images_class=val_images_label,
                        transform=data_transform["val"])

# Create the dataloader
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=workers)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=True, num_workers=workers)
```

实例化自定义的数据集类 MyDataSet 后，将其传给 DataLoader 作为参数，得到一个可遍历的数据加载器。可以通过参数 batch_size 控制批处理大小，shuffle 控制是否乱序读取，num_workers 控制用于读取数据的线程数量。

# 6.模型设计

```
# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)  # 第一个卷积层，输入通道数 3，输出通道数 16，卷积核大小 3×3，padding 大小 1
        self.conv2 = nn.Conv2d(16, 16, 3, padding=1)

        self.fc1 = nn.Linear(50 * 50 * 16, 128)     # 第一个全连接层，线性连接，输入节点数 50×50×16，输出节点数 128
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 2)

    def forward(self, x):       # 重写父类 forward 方法，通过该方法获取网络输入数据后的输出值
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)      # n * 16 * 50 * 50

        x = x.view(x.size()[0], -1)     # 将输入的[50×50×16]格式数据排列成[40000×1]形式
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return F.softmax(x, dim=1)      # 采用 SoftMax 方法将输出的 2 个输出值调整至[0.0, 1.0],且两者的和为 1

# Create the Discriminator
netD = Net().to(device)
```

设计了一个 CNN 网络，结构如下

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=40000, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=2, bias=True)
)
```

设计自己的模型结构需要继承 torch.nn.Module 这个类，然后实现其中的 forward 方法，一般在__init__中设定好网络模型的一些组件，然后在 forward 方法中依据输入输出顺序拼装组件。

## 7.训练及测试

```
# Train
optimizer = optim.Adam(netD.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()

def train(model, optimizer, data_loader, device):
    for epoch in range(num_epochs):
        data_loader = tqdm(data_loader)
        val_acc = 0.0
        val_loss = 0.0

        for step, data in enumerate(data_loader):
            image, label = data
            optimizer.zero_grad()
            image,  label = image.to(device), label.to(device)
            # print(label)
            pred = model(image)
            loss = criterion(pred, label.squeeze())
            loss.backward()
            optimizer.step()

        val_acc, val_loss = evaluate(netD, val_loader, device)
        print('epoch = {:>2d}, val_loss = {:>.5f}, val_acc = {:.3%}'.format(epoch + 1, val_loss, val_acc))


def evaluate(model, data_loader, device):
        correct_num = 0.0
        total_loss = 0.0
        for step, data in enumerate(data_loader):
            image, label = data
            image,  label = image.to(device), label.to(device)
            with torch.no_grad():
                pred = model(image)
                total_loss += criterion(pred, label.squeeze()).item()
                correct_num += (pred.argmax(1) == label ).type(torch.float).sum().item()

        val_loss = total_loss / len(data_loader)
        val_acc = correct_num / len(data_loader.dataset)
        return  val_acc, val_loss

train(netD, optimizer, train_loader, device)
```

实例化模型后，网络模型的训练需要定义损失函数与优化器，损失函数定义了网络输出与标签的差距，而优化器则定义了神经网络中的参数如何基于损失来更新。

训练过程就是使用 **dataloader** 依照 **batch** 读出数据后，将输入数据放入网络模型中计算得到网络的输出，然后基于标签通过损失函数计算 **Loss**，并将 **Loss** 反向传播回神经网络（在此之前需要清理上一次循环时的梯度），最后通过优化器更新权重。

测试和训练的步骤差不多，也是读取模型后通过 **dataloader** 获取数据，然后将其输入网络获得输出，但是不需要进行反向传播等操作了。

二、实验结果

```
... (more hidden) ...(base) peer@ubuntu:~/Desktop/虚假人脸检测$ cd /home/peer/Desktop/虚假人脸检测 ; /usr/bin/env /home/pee
 /home/peer/.vscode/extensions/ms-python.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 43029 -- /home/peer/Desktop/
虚假人脸检测/fakeface.py
Random Seed:  8115
3998 images were found in the dataset.
1000 images for training.
2998 images for validation.
100%|██████████| 32/32 [00:19<00:00,  1.67it/s]
epoch =  1, val_loss = 0.69236, val_acc = 52.535%
100%|██████████| 32/32 [00:19<00:00,  1.65it/s]
epoch =  2, val_loss = 0.69128, val_acc = 50.500%
100%|██████████| 32/32 [00:17<00:00,  1.88it/s]
epoch =  3, val_loss = 0.68736, val_acc = 56.271%
100%|██████████| 32/32 [00:17<00:00,  1.88it/s]
epoch =  4, val_loss = 0.68353, val_acc = 54.336%
100%|██████████| 32/32 [00:17<00:00,  1.82it/s]
epoch =  5, val_loss = 0.68158, val_acc = 56.971%
100%|██████████| 32/32 [00:15<00:00,  2.10it/s]
epoch =  6, val_loss = 0.68211, val_acc = 54.036%
100%|██████████| 32/32 [00:16<00:00,  1.92it/s]
epoch =  7, val_loss = 0.65710, val_acc = 62.675%
100%|██████████| 32/32 [00:17<00:00,  1.85it/s]
epoch =  8, val_loss = 0.64227, val_acc = 64.443%
100%|██████████| 32/32 [00:17<00:00,  1.79it/s]
epoch =  9, val_loss = 0.63489, val_acc = 65.110%
100%|██████████| 32/32 [00:18<00:00,  1.71it/s]
epoch = 10, val_loss = 0.63505, val_acc = 64.310%
100%|██████████| 32/32 [00:17<00:00,  1.81it/s]
epoch = 11, val_loss = 0.63470, val_acc = 63.476%
100%|██████████| 32/32 [00:17<00:00,  1.84it/s]
epoch = 12, val_loss = 0.60664, val_acc = 68.612%
100%|██████████| 32/32 [00:23<00:00,  1.36it/s]
epoch = 13, val_loss = 0.59461, val_acc = 70.013%
100%|██████████| 32/32 [00:17<00:00,  1.80it/s]
epoch = 14, val_loss = 0.58774, val_acc = 70.247%
100%|██████████| 32/32 [00:19<00:00,  1.61it/s]
epoch = 15, val_loss = 0.59096, val_acc = 70.480%
```

在训练集上经过 **15** 轮完整的训练后，在测试集上可以达到 **70.480%**的准确率。

三、实验结果的分析

设计的 **CNN** 模型较为简单，对于超参数的调整也较少，加上训练集相对较少带来的过拟合问题，使准确率有待提高。

实验总结：

**1**.依据网站上的教程完整学习了 **FGSM** 攻击的原理和实现，并阅读相关文章，进一步了解 **FGSM** 是如何实现攻击的。

**2**.同时学习了网站上的 **DCGAN** 教程，学习如何通过 **DCGAN** 生成伪造图片。并进一步学习了在 **pytorch** 中如何加载数据集，生成 **dataloader**，设计模型和进行训练及测试。

**3**.实验中遇到的主要难题是要学习各种函数，了解它们的功能和输入输出的格式要求。训练的时候也要选择合适的损失函数与优化器，定义各种参数，提升模型性能。

参考文献：

对抗样本攻击实验

https://pytorch.org/tutorials/beginner/fgsm_tutorial.html

https://zhuanlan.zhihu.com/p/371983112

https://blog.csdn.net/crystal_sugar/article/details/106023055

https://www.cnblogs.com/tangweijqxx/p/10615950.html

虚假人脸检测实验

https://blog.csdn.net/qq_34392457/article/details/113748534

https://github.com/satomiishihara/M2TR-pytorch

https://blog.csdn.net/weixin_44491423/article/details/121892838