**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：**

**学 号：**

**姓 名：**

**电子邮箱：**

**指导教师： 王善峰**

**2022年6月1日**

|  |
| --- |
| **实验题目：虚拟人脸检测实验** |
| **实验摘要（简单描述实验内容）：**  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。 |
| **题目描述（清楚描述题目中文，写出自己的理解，请勿原封复制原题目）：**  **使用ConvNet作为初始化或固定功能提取器执行感兴趣的人物。在本题中，将会使用传输学习训练一个卷积神经网络进行图像分类。**  **主要的转移学习场景：**   * ConvNet 作为固定特征提取器。在 ImageNet 上预先训练一个 ConvNet，删除最后一个完全连接的层（该层的输出是不同任务（如 ImageNet）的 1000 个类分数），然后将 ConvNet 的其余部分视为新数据集的固定特征提取器。在AlexNet中，这将为包含紧挨着分类器之前的隐藏层激活的每个图像计算一个4096-D向量。我们将这些功能称为CNN代码。对于性能而言，如果这些代码在 ImageNet 上训练 ConvNet 期间也被阈值（通常的情况），则这些代码是 ReLUd（即阈值为零），这一点很重要。提取所有图像的 4096-D 代码后，为新数据集训练线性分类器（例如线性 SVM 或 Softmax 分类器）**。** * **微调 ConvNet**。第二种策略是不仅要在新数据集上替换和重新训练ConvNet 顶部的分类器，还要通过继续反向传播来微调预训练网络的权重。可以微调ConvNet的所有层，也可以保持一些早期层的固定（由于过度拟合问题），并且只微调网络的一些更高级别部分。这是由于观察到ConvNet的早期特征包含更多通用特征（例如边缘检测器或色斑点检测器），这些特征应该对许多任务有用，但是ConvNet的后续层逐渐变得更加特定于原始数据集中包含的类的详细信息。例如，在包含许多狗品种的ImageNet的情况下，ConvNet的代表性的很大一部分可能致力于区分狗品种的特征。 * **预训练模型**。由于现代ConvNet需要在2-3周内在ImageNet上跨多个GPU进行训练，因此人们通常会发布最终的ConvNet检查点，以使可以使用网络进行微调的其他人受益。例如，Caffe 图书馆有一个模型动物园，人们可以在其中共享其网络权重。 |
| **实验内容（详细内容）：**   1. **实验基本原理及步骤**   构造一种基于卷积神经网络的人脸识别算法，由两个卷积层，两个池化层，一个全连接层和一个softmax回归层组成，可以自动提取人脸特征并进行分类，网络通过批量梯度下降法训练特征提取器和分类器，隐层利用dropout方法解决过拟合问题。该方法在各人脸识别数据库中的识别率达到99%+，并具有较强的鲁棒性。  卷积神经网络是一个受生物视觉启发、以最简化预处理操作为目的的多层感知器的变形，本质是一个前向反馈神经网络，卷积神经网络与多层感知器的最大区别是网络前几层由卷积层和池化层交替级联组成，模拟视觉皮层中用于高层次特征提取的简单细胞和复杂细胞交替级联结构。  卷积层的神经元对前一层输入的一部分区域（称为局部感受野，区域之间有重叠）有响应，提取输入的更高层次特征；池化层的神经元对前一层输入的一部分区域（区域之间无重叠）求平均值或最大值，抵抗输入的轻微形变或位移。卷积神经网络的后几层一般是若干个全连接层和一个分类器构成的输出层。  卷积神经网络用于人脸识别是一种基于特征的方法，区别于传统的人工特征提取和针对特征的高性能分类器设计，它的优点是通过逐层卷积降维进行特征提取，然后经过多层非线性映射，使网络可以从未经特殊处理的训练样本中，自动学习形成适应该识别任务的特征提取器和分类器，该方法降低了对训练样本的要求，而且网络的层数越多，学习到的特征更具有全局性。  LeNet-5是LeCun提出的一个具有高识别率的用于手写数字的卷积神经网络，本文借鉴LeNet-5的模型结构，提出6层卷积神经网络用于ORL人脸库和AR人脸库的人脸识别，网络前4层由卷积层和池化层交替级联用于特征提取随后接入一个全连接层，最后一层采用非线性分类能力强的Softmax分类器，该6层卷积神经网络采用sigmoid激活函数和固定的学习速率。  计算得到的梯度，这样就能使得损失值越来越小，从而模型预测对的概率越来越大。既然无目标攻击是希望模型将输入图像错分类成正确类别以外的其他任何一个类别都算攻击成功，那么只需要损失值越来越大就可以达到这个目标，也就是模型预测的概率中对应于真实标签的概率越小越好，这和原来的参数更新目的正好相反。因此我只需要在输入图像中加上计算得到的梯度方向，这样修改后的图像经过分类网络时的损失值就比修改前的图像经过分类网络时的损失值要大，换句话说，模型预测对的概率变小了。这就是FGSM算法的内容，一方面是基于输入图像计算梯度，另一方面更新输入图像时是加上梯度，而不是减去梯度，这和常见的分类模型更新参数正好背道而驰。  前面我们提到之所以采用梯度方向而不是采用梯度值是为了控制扰动的L∞距离，这只是其中的一部分。在Figure1中，梯度方向前一般会有一个权重参数e，这个权重参数可以用来控制攻击噪声的幅值，参数值越大，攻击强度也越大，肉眼也更容易观察到攻击噪声（因为输入图像归一化成0到1，所以图中e值只有0.07，换算成0到255的话差不多18个像素值），因此最终的攻击噪声就如下所示。因为FGSM算法的攻击噪声幅值评价指标是L∞，因此权重参数e和梯度方向就可以控制每个像素的最大变化值。  FGSM算法简单有效，在图像攻击领域扮演着非常重要的角色，后续的许多研究也都基于这个算法开展，比如为了提高攻击成功率而推出的迭代版FGSM等。  这里写图片描述  卷积层：  卷积层通过局部连接和权值共享的方法，模拟具有局部感受野的简单细胞，提取一些初级视觉特征的过程。局部连接指卷积层上的每一个神经元与前一层特征图中固定区域的神经元建立连接；权值共享指同一特征图中的神经元用一组相同的连接强度与前一层局部连接，可以减少网络训练参数，上述一组相同的连接强度即为一个特征提取器，在运算的过程中变现为一个卷积核，卷积核数值先随机初始化，最后由网络训练确定。  人脸比数字识别更加复杂，卷积层为了更充分提取人脸特征进行了以下改进：（1）增加2个卷积层的特征提取器——卷积核的数量；（2）卷积层2的特征图与池化层1的全部特征图均建立关系。  卷积层每个神经元的输入来自于前一层特征图中固定区域的神经元，区域的大小由卷积核大小决定，卷几层1的m张特征图由1张输入图片跟m个可学习的卷积核卷积，加上偏置后，通过激活函数作用得到卷积层2的n张特征图，由池化层1的m张特征图分别跟n\*m个卷积核卷积，每m个卷积结果结合，再加上偏置，通过激活函数作用得到。  池化层：  池化层模拟复杂细胞是将初级的视觉特征筛选并结合成更高级、抽象的视觉特征的过程，在网络中通过采样实现，经过池化层的采样后，输出特征图的数量不变，但是特征图的尺寸会变小，有减小计算复杂度、抵抗微小位移变化的作用。此文提出的池化层采用做大值采样，采样大小为2\*2，即把输入的特征图分割成不重叠的2\*2大小的矩形，对每个矩形取最大值，所以输出特征图的长和宽均是输入特征图的一半，本文定义池化层中的神经元不具备学习功能。  全连接层：  为了增强网络的非线性能力，同时限制网络规模的大小，网络在4个特征提取层提取特征后，接入一个全连接层，该层的每一个神经元与前一层的所有神经元互相连接，同层神经元之间不连接。  Softmax回归层：  LeNet-5最后一层是RBF分类器，采用由ASCII码表中的数字作为模板算出的纠错码或者以等概率的方式任选-1和1作为RBF的权值，每个RBF输入单元计算输入特征和参数向量之间的欧式距离，但是人脸特征比数字特征更复杂，而且人脸类别多又没有统一的模板，所以网络的最后一层采用非线性分类能力强的Softmax回归作为分类器。  这里写图片描述  **步骤：**  这两个主要的转移学习场景如下所示：  微调对流网：我们用预先训练的网络（如在图像数据集上训练的网络）来初始化网络，而不是随机初始化。其余的训练看起来和往常一样。  ConvNet 作为固定功能提取器：在这里，我们将冻结除最终完全连接层之外的所有网络的权重。最后一个完全连接的层被一个新的随机重量替换，并且只有此层经过训练。  # License: BSD  # Author: Sasank Chilamkurthy  from \_\_future\_\_ import print\_function, division  import torch  import torch.nn as nn  import torch.optim as optim  from torch.optim import lr\_scheduler  import numpy as np  import torchvision  from torchvision import datasets, models, transforms  import matplotlib.pyplot as plt  import time  import os  import copy  plt.ion() # interactive mode  使用的是torchvision and torch.utils.data来加载数据。  要解决的问题是训练一个模型来分类假人脸和真人脸。每类数据包含1999张图片，其中500张用作训练集，剩余作为测试集。  # Data augmentation and normalization for training  # Just normalization for validation  data\_transforms = {  'train': transforms.Compose([  transforms.RandomResizedCrop(224),  transforms.RandomHorizontalFlip(),  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  ]),  'val': transforms.Compose([  transforms.Resize(256),  transforms.CenterCrop(224),  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  ]),  }  data\_dir = 'data/hymenoptera\_data'  image\_datasets = {x: datasets.ImageFolder(os.path.join(data\_dir, x),  data\_transforms[x])  for x in ['train', 'val']}  dataloaders = {x: torch.utils.data.DataLoader(image\_datasets[x], batch\_size=4,  shuffle=True, num\_workers=4)  for x in ['train', 'val']}  dataset\_sizes = {x: len(image\_datasets[x]) for x in ['train', 'val']}  class\_names = image\_datasets['train'].classes  device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  可视化部分图像  让我们将一些培训图像可视化，以便了解数据增强。  def imshow(inp, title=None):  """Imshow for Tensor."""  inp = inp.numpy().transpose((1, 2, 0))  mean = np.array([0.485, 0.456, 0.406])  std = np.array([0.229, 0.224, 0.225])  inp = std \* inp + mean  inp = np.clip(inp, 0, 1)  plt.imshow(inp)  if title is not None:  plt.title(title)  plt.pause(0.001) # pause a bit so that plots are updated  # Get a batch of training data  inputs, classes = next(iter(dataloaders['train']))  # Make a grid from batch  out = torchvision.utils.make\_grid(inputs)  imshow(out, title=[class\_names[x] for x in classes])  训练模型  现在，让我们写一个通用功能来训练一个模型。在这里，我们将说明：   * 安排学习率 * 保存最佳型号   在下列中，参数是来自 LR 调度器对象。schedulertorch.optim.lr\_scheduler  def train\_model(model, criterion, optimizer, scheduler, num\_epochs=25):  since = time.time()  best\_model\_wts = copy.deepcopy(model.state\_dict())  best\_acc = 0.0  for epoch in range(num\_epochs):  print('Epoch {}/{}'.format(epoch, num\_epochs - 1))  print('-' \* 10)  # Each epoch has a training and validation phase  for phase in ['train', 'val']:  if phase == 'train':  model.train() # Set model to training mode  else:  model.eval() # Set model to evaluate mode  running\_loss = 0.0  running\_corrects = 0  # Iterate over data.  for inputs, labels in dataloaders[phase]:  inputs = inputs.to(device)  labels = labels.to(device)  # zero the parameter gradients  optimizer.zero\_grad()  # forward  # track history if only in train  with torch.set\_grad\_enabled(phase == 'train'):  outputs = model(inputs)  \_, preds = torch.max(outputs, 1)  loss = criterion(outputs, labels)  # backward + optimize only if in training phase  if phase == 'train':  loss.backward()  optimizer.step()  # statistics  running\_loss += loss.item() \* inputs.size(0)  running\_corrects += torch.sum(preds == labels.data)  if phase == 'train':  scheduler.step()  epoch\_loss = running\_loss / dataset\_sizes[phase]  epoch\_acc = running\_corrects.double() / dataset\_sizes[phase]  print('{} Loss: {:.4f} Acc: {:.4f}'.format(  phase, epoch\_loss, epoch\_acc))  # deep copy the model  if phase == 'val' and epoch\_acc > best\_acc:  best\_acc = epoch\_acc  best\_model\_wts = copy.deepcopy(model.state\_dict())  print()  time\_elapsed = time.time() - since  print('Training complete in {:.0f}m {:.0f}s'.format(  time\_elapsed // 60, time\_elapsed % 60))  print('Best val Acc: {:4f}'.format(best\_acc))  # load best model weights  model.load\_state\_dict(best\_model\_wts)  return model  可视化模型预测  显示一些图像预测的通用功能  def visualize\_model(model, num\_images=6):  was\_training = model.training  model.eval()  images\_so\_far = 0  fig = plt.figure()  with torch.no\_grad():  for i, (inputs, labels) in enumerate(dataloaders['val']):  inputs = inputs.to(device)  labels = labels.to(device)  outputs = model(inputs)  \_, preds = torch.max(outputs, 1)  for j in range(inputs.size()[0]):  images\_so\_far += 1  ax = plt.subplot(num\_images//2, 2, images\_so\_far)  ax.axis('off')  ax.set\_title('predicted: {}'.format(class\_names[preds[j]]))  imshow(inputs.cpu().data[j])  if images\_so\_far == num\_images:  model.train(mode=was\_training)  return  model.train(mode=was\_training)  微调卷积神经网络  加载经过预制的模型并重置最终完全连接的层。  model\_ft = models.resnet18(pretrained=True)  num\_ftrs = model\_ft.fc.in\_features  # Here the size of each output sample is set to 2.  # Alternatively, it can be generalized to nn.Linear(num\_ftrs, len(class\_names)).  model\_ft.fc = nn.Linear(num\_ftrs, 2)  model\_ft = model\_ft.to(device)  criterion = nn.CrossEntropyLoss()  # Observe that all parameters are being optimized  optimizer\_ft = optim.SGD(model\_ft.parameters(), lr=0.001, momentum=0.9)  # Decay LR by a factor of 0.1 every 7 epochs  exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_ft, step\_size=7, gamma=0.1)  训练与评估  CPU大约需要25分钟。不过，在GPU上，只需不到一分钟。这里我使用的是google-colab实验平台，为了提高实验效率，使用的是GPU来完成。  model\_ft = train\_model(model\_ft, criterion, optimizer\_ft, exp\_lr\_scheduler, num\_epochs=25)  然后使用这个代码展示分析结果  visualize\_model(model\_ft)   1. **实验结果**      1. **实验结果的分析**   为提高实验精确度，选择进行25轮。    训练模型准确度      在进行至20轮时，准确率第一次到达100%，随后进行了波动。  在这里，我们需要冻结除最后一层以外的所有网络。我们需要设置冻结参数，以便梯度不计算在。requires\_grad == Falsebackward()。具体原理：  评价模式实际上不是局部禁用梯度计算的一种机制。无论如何，它包括在这里，因为它有时是混淆这样的机制。  在功能上，（或等价物）完全正交到非毕业模式和推理模式。模型如何影响完全取决于模型中使用的特定模块，以及它们是否定义了任何培训模式特定行为。module.eval()module.train()model.eval()负责呼叫，如果您的模型依赖于诸如torch.nn.Dropout和手电筒.nn.BatchNorm2d等模块，则根据培训模式（例如，您）的行为可能会有所不同，以避免更新您的 BatchNorm 运行统计数据以验证数据。  model.eval()model.train()  model.train()model.eval()  model\_conv = torchvision.models.resnet18(pretrained=True)  for param in model\_conv.parameters():      param.requires\_grad = False  # Parameters of newly constructed modules have requires\_grad=True by default  num\_ftrs = model\_conv.fc.in\_features  model\_conv.fc = nn.Linear(num\_ftrs, 2)  model\_conv = model\_conv.to(device)  criterion = nn.CrossEntropyLoss()  # Observe that only parameters of final layer are being optimized as  # opposed to before.  optimizer\_conv = optim.SGD(model\_conv.fc.parameters(), lr=0.001, momentum=0.9)  # Decay LR by a factor of 0.1 every 7 epochs  exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_conv)  训练与评估  在 CPU 上，与之前的情况相比，这大约只需要一半的时间。这是预料之中的，因为大多数网络不需要计算梯度。但是牺牲了准确率。    训练结果： |
| **实验总结（完成心得与其它，主要讲自己碰到的问题和解决问题的方法）：**  **在不熟悉卷积网络与图像识别算法的基础下，解决人脸面部识别问题是具有一定难度的，在参考了资料并学习了部分知识后，可以按照一些指导一步步摸索。并设计一种神经网络模型进行分类。为提高实验效率，使用Google-colab平台。** |
| **参考文献（包括参考的书籍，论文，URL等，很重要）：**  <https://blog.csdn.net/Enjolras_fuu/article/details/60138274?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-0&spm=1001.2101.3001.4242>  <https://blog.csdn.net/weixin_43308332/article/details/93382141>  <https://cs231n.github.io/transfer-learning/>  <https://blog.csdn.net/weixin_39612110/article/details/110924335>  <https://pytorch.org/tutorials/intermediate/quantized_transfer_learning_tutorial.html> |