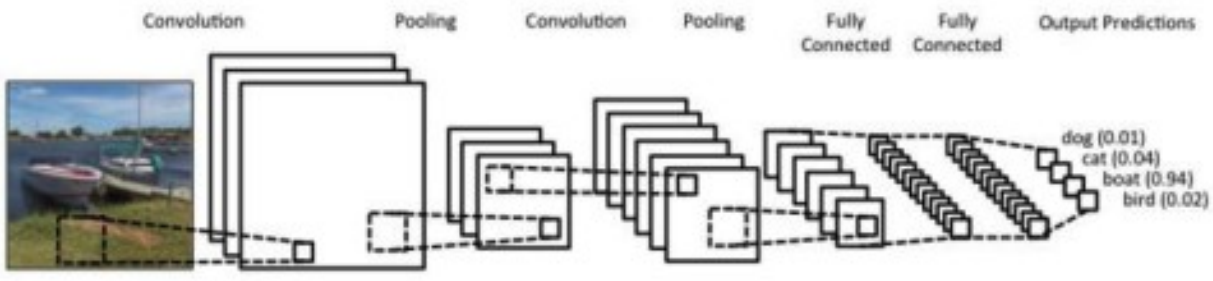
Assignment 2: Convolutional Neural Networks

Instructions: In Assignment 2, you will learn all about the convolutional neural networks. In particular, you will gain a first-hand experience of the training process, understand the architectural details, and familiarize with transfer learning with deep networks.

Part 1: Convolutional Neural Networks

In this part, you will experiment with a convolutional neural network implementation to perform image classification. The dataset we will use for this assignment was created by Zoya Bylinskii, and contains 451 works of art from 11 different artists all downsampled and padded to the same size. The task is to identify which artist produced each image. The original images can be found in the `art_data/artists` directory included with the data zip file. The composition of the dataset and a sample painting from each artist are shown in Table 1.

Figure 1 shows an example of the type of convolutional architecture typically employed for similar image recognition problems. Convolutional layers apply filters to the image, and produce layers of feature maps. Often, the convolutional layers are interspersed with pooling layers. The final layers of the network are fully connected, and lead to an output layer with one node for each of the K classes the network is trying to detect. We will use a similar architecture for our network.



The code for performing the data processing and training the network is provided in the starter pack. You will use PyTorch to implement convolutional neural networks. We create a dataset from the artists' images by downsampling them to 50x50 pixels, and transforming the RGB values to lie within the range $[-0.5, 0.5)$. We provide a lot of starter code below, but you will need to modify the hyperparameters and network structure.

Table 1: Artists and number of paintings in the dataset.

Artist	#	Example	Artist	#	Example
Canaletto	20		Paul Gauguin	37	
Claude Monet	54		Paul Sandby	36	
George Romney	63		Peter Paul Rubens	28	
J. M. W. Turner	70		Rembrandt	40	
John Robert Cozens	40		Richard Wilson	23	
Paul Cezanne	40				

Part 1.1: Convolutional Filter Receptive Field

First, it is important to develop an intuition for how a convolutional layer affects the feature representations that the network learns. Assume that you have a network in which the first

convolutional layer applies a 5x5 patch to the image, producing a feature map Z_1. The next layer of the network is also convolutional; in this case, a 3x3 patch is applied to the feature map Z_1 to produce a new feature map, Z_2. Assume the stride used in both cases is 1. Let the receptive field of a node in this network be the portion of the original image that contributes information to the node (that it can, through the filters of the network, "see"). What are the dimensions of the receptive field for a node in Z_2? Note that you can ignore padding, and just consider patches in the middle of the image and Z_1. Thinking about your answer, why is it effective to build convolutional networks deeper, i.e. with more layers?

Answer here:

The receptive field for each node in Z_2 is 7x7 pixels in the input image.

The first convolutional layer uses a 5x5 patch with stride 1, so each output in Z_1 is influenced by a 5x5 region of the input. The second layer applies a 3x3 patch (again stride 1) to Z_1, so each Z_2 node covers a 3x3 area of Z_1, each of which in turn corresponds to a 5x5 area of the original image. Calculating this, the total receptive field is 5 + (3-1) = 7.

Making networks deeper increases the receptive field of higher-level nodes, allowing them to detect more complex and larger-scale features in the input by integrating information from broader areas of the image. This enables hierarchical and abstract representations, making deep convolutional networks more effective at capturing complex structures in images.

Part 1.2: Run the PyTorch ConvNet

Study the provided SimpleCNN class below, and take a look at the hyperparameters. Answer the following questions about the initial implementation:

1) How many layers are there? Are they all convolutional? If not, what structure do they have? 2) Which activation function is used on the hidden nodes? 3) What loss function is being used to train the network? 4) How is the loss being minimized?

Answer here:

1) There are four layers in the provided SimpleCNN network: two are convolutional layers and two are fully connected (linear) layers. So, not all layers are convolutional; the network has both convolutional and linear (fully connected) layers. 2) The activation function used on the hidden nodes is the ReLU (Rectified Linear Unit) function. 3) The network is trained using the cross-entropy loss function (nn.CrossEntropyLoss). 4) The loss is minimized using an optimizer from torch.optim (such as Adam or SGD), which updates the network weights to reduce the loss during training.

Now that you are familiar with the code, try training the network. It should take between 60-120 seconds to train for 50 epochs. What is the training accuracy for your network after training? What is the validation accuracy? What do these two numbers tell you about what your network is doing?

Answer here:

Best val acc: 0.5604395604395604 and Best train acc: 0.6972222222222222

It is underfitting the triaining data since the both loss are high.

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from PIL import Image, ImageFile
import tqdm
from torch.nn import CrossEntropyLoss
import time
import random
from torchvision import transforms, utils
import numpy as np
import os
from torch import optim

device = torch.device("cuda") if torch.cuda.is_available() else
torch.device("cpu")
# device = torch.device("mps") if torch.backends.mps.is_available()
else torch.device("cpu")

class SimpleCNN(torch.nn.Module):
    def __init__(self,device,pooling= False):
        super(SimpleCNN, self).__init__()
        self.device = device
        self.pooling = pooling
        self.conv_layer1 =
torch.nn.Conv2d(in_channels=3,out_channels=16,kernel_size=5,stride=2,
device=device)
        self.pool_layer1 = torch.nn.MaxPool2d(kernel_size=2,stride=2)
        self.conv_layer2 =
torch.nn.Conv2d(in_channels=16,out_channels=16,kernel_size=5,stride=2,
device=device)
        self.pool_layer2 = torch.nn.MaxPool2d(kernel_size=2,stride=2)
        if pooling:
            self.fully_connected_layer = nn.Linear(64,64,
device=device)
            self.final_layer = nn.Linear(64,11, device=device)
        else:
            self.fully_connected_layer = nn.Linear(1600, 64,
device=device)
            self.final_layer = nn.Linear(64, 11, device=device)
    def forward(self,inp):
        x = torch.nn.functional.relu(self.conv_layer1(inp))
        if self.pooling:
            x = self.pool_layer1(x)
        x = torch.nn.functional.relu(self.conv_layer2(x))
        if self.pooling:
            x = self.pool_layer2(x)
        x = x.reshape(x.size(0),-1)
        x = torch.nn.functional.relu(self.fully_connected_layer(x))
```

```
        x = self.final_layer(x)
        return x

class LoaderClass(Dataset):
    def __init__(self,data,labels,phase,transforms):
        super(LoaderClass, self).__init__()
        self.transforms = transforms
        self.labels = labels[phase + "_labels"]
        self.data = data[phase + "_data"]
        self.phase = phase

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

    def __getitem__(self, idx):
        label = self.labels[idx]
        img = self.data[idx]
        img = Image.fromarray(img)
        img = self.transforms(img)
        return img,torch.from_numpy(label)

class Trainer():
    def __init__(self,model,criterion,tr_loader,val_loader,optimizer,
                 num_epoch,patience,batch_size,lr_scheduler=None):
        self.model = model
        self.tr_loader = tr_loader
        self.val_loader = val_loader
        self.optimizer = optimizer
        self.num_epoch = num_epoch
        self.patience = patience
        self.lr_scheduler = lr_scheduler
        self.criterion = criterion
        self.softmax = nn.Softmax()
        self.no_inc = 0
        self.best_loss = 9999
        self.phases = ["train","val"]
        self.best_model = []
        self.best_val_acc = 0
        self.best_train_acc = 0
        self.best_val_loss = 0
        self.best_train_loss = 0
        self.batch_size = batch_size

        pass
    def train(self):
        pbar = tqdm.tqdm(desc= "Epoch 0, phase:
Train",postfix="train_loss : ?, train_acc: ?")
        for i in range(self.num_epoch):
            last_train_acc = 0
            last_val_acc = 0
```

```
            last_val_loss = 0
            last_train_loss = 0
            pbar.update(1)

            for phase in self.phases:
                total_acc = 0
                total_loss = 0
                start = time.time()
                if phase == "train":
                    pbar.set_description_str("Epoch %d,"% i + "phase:
Training")
                    loader = self.tr_loader
                    self.model.train()
                else:
                    pbar.set_description_str("Epoch %d,"% i + "phase:
Validation")
                    loader = self.val_loader
                    self.model.eval()
                iter = 0
                for images,labels in loader:
                    iter += 1
                    images = images.to(self.model.device)
                    labels = labels.to(self.model.device)
                    self.optimizer.zero_grad()
                    logits = self.model(images)
                    softmaxed_scores = self.softmax(logits)
                    _, predictions = torch.max(softmaxed_scores,1)
                    _, labels = torch.max(labels,1)
                    loss =
self.criterion(softmaxed_scores.float(),labels.long())
                    total_loss += loss.item()
                    total_acc += torch.sum(predictions ==
labels).item()

                    if phase == "train":
                        pbar.set_postfix_str("train acc: %6.3f," %
(total_acc/ (iter*self.batch_size)) + ("train loss: %6.3f" %
(total_loss / iter)))
                        loss.backward()
                        self.optimizer.step()
                    else:
                        pbar.set_postfix_str("val acc: %6.3f," %
(total_acc/ (iter*self.batch_size)) + ("val loss: %6.3f" % (total_loss
/ iter)))


                if phase == "train":
                    if self.lr_scheduler:

                        self.lr_scheduler.step()
```

```
                end = time.time()
                if phase == "train":
                    loss_p = total_loss / iter
                    acc_p = total_acc / len(self.tr_loader.dataset)
                    last_train_acc = acc_p
                    last_train_loss = loss_p
                else:
                    loss_p = total_loss / iter
                    acc_p = total_acc / len(self.val_loader.dataset)
                    last_val_acc = acc_p
                    last_val_loss = loss_p

                    if loss_p < self.best_loss:
                        print("New best loss, loss is: ",str(loss_p),
"acc is: ",acc_p )
                        self.best_loss = loss_p
                        self.no_inc = 0
                        self.best_model = self.model
                        self.best_train_acc = last_train_acc
                        self.best_train_loss = last_train_loss
                        self.best_val_loss = last_val_loss
                        self.best_val_acc = last_val_acc
                    else:
                        print("Not a better score")


                        self.no_inc += 1
                        if self.no_inc == self.patience:
                            print("Out of patience returning the best
model")
                            print(
                                "Best val acc: {}, Best val loss: {},
Best train acc: {}, Best train loss: {} ".format(
                                    self.best_val_acc,
self.best_val_loss, self.best_train_acc, self.best_train_loss
                                ))  # Stats of the best model
                            return self.best_model
        print("Training ended returning the best model")
        print(
            "Best val acc: {}, Best val loss: {}, Best train acc: {},
Best train loss: {} ".format(
                self.best_val_acc, self.best_val_loss,
self.best_train_acc, self.best_train_loss
            ))  # Stats of the best model
        return self.best_model

LR = 1e-4
Momentum = 0.9 # If you use SGD with momentum
BATCH_SIZE = 16
POOLING = False
```

```
NUM_EPOCHS = 200
PATIENCE = -1
TRAIN_PERCENT = 0.8
VAL_PERCENT = 0.2
NUM_ARTISTS = 11
DATA_PATH = "./art_data/artists"
ImageFile.LOAD_TRUNCATED_IMAGES = True # Do not change this

def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = True

def load_artist_data():
    data = []
    labels = []
    artists = [x for x in os.listdir(DATA_PATH) if x != '.DS_Store']
    print(artists)
    for folder in os.listdir(DATA_PATH):
        class_index = artists.index(folder)
        for image_name in os.listdir(DATA_PATH + "/" + folder):
            img = Image.open(DATA_PATH + "/" + folder + "/" +
image_name)
            artist_label = (np.arange(NUM_ARTISTS) ==
class_index).astype(np.float32)
            data.append(np.array(img))
            labels.append(artist_label)
    shuffler = np.random.permutation(len(labels))
    data = np.array(data)[shuffler]
    labels = np.array(labels)[shuffler]

    length = len(data)
    val_size = int(length*0.2)
    val_data = data[0:val_size+1]
    train_data = data[val_size+1::]
    val_labels = labels[0:val_size+1]
    train_labels = labels[val_size+1::]
    print(val_labels)
    data_dict = {"train_data":train_data,"val_data":val_data}
    label_dict =
{"train_labels":np.array(train_labels),"val_labels":np.array(val_label
s)}

    return data_dict,label_dict
```

```
seed_everything(42)
data,labels = load_artist_data()
model = SimpleCNN(device=device,pooling=False)
optimizer = optim.Adam(model.parameters(), lr=LR)
transform = {
    'train': transforms.Compose([
        transforms.Resize(50),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(50),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
0.225])
    ]),
    }
```

```
['richard wilson', 'claude monet', 'paul sandby', 'peter paul rubens',
'canaletto', 'paul cezanne', 'j. m. w. turner', 'john robert cozens',
'paul gauguin', 'rembrandt', 'george romney']
[[0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 ...
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
```

```
train_dataset = LoaderClass(data,labels,"train",transform["train"])
valid_dataset = LoaderClass(data,labels,"val",transform["val"])
train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
num_workers=0, pin_memory=True)
val_loader = torch.utils.data.DataLoader(valid_dataset,
                                         batch_size=BATCH_SIZE,
                                         shuffle=True,
num_workers=0, pin_memory=True)

criterion = CrossEntropyLoss()
trainer_m = Trainer(model, criterion, train_loader, val_loader,
optimizer, num_epoch=NUM_EPOCHS,
patience=PATIENCE,batch_size=BATCH_SIZE,lr_scheduler= None)
best_model = trainer_m.train()
```

```
Epoch 0,phase: Training: 1it [00:00, 14926.35it/s, train_loss : ?,
train_acc: ?]/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-
packages/torch/utils/data/dataloader.py:684: UserWarning: 'pin_memory'
```

```
argument is set as true but not supported on MPS now, then device
pinned memory won't be used.
  warnings.warn(warn_msg)
/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-packages/
torch/nn/modules/module.py:1773: UserWarning: Implicit dimension
choice for softmax has been deprecated. Change the call to include
dim=X as an argument.
  return self._call_impl(*args, **kwargs)
Epoch 1,phase: Training: 2it [00:00,  6.34it/s, train acc:
0.269,train loss:  2.348]

New best loss, loss is:  2.3782220681508384 acc is:
0.14285714285714285

Epoch 2,phase: Training: 3it [00:00,  4.99it/s, train acc:
0.348,train loss:  2.241]

New best loss, loss is:  2.307943264643351 acc is:  0.3076923076923077

Epoch 3,phase: Training: 4it [00:01,  4.62it/s, train acc:
0.352,train loss:  2.165]

New best loss, loss is:  2.1777648528416953 acc is:
0.34065934065934067

Epoch 4,phase: Training: 5it [00:01,  4.43it/s, train acc:
0.403,train loss:  2.140]

New best loss, loss is:  2.1406694849332175 acc is:
0.4065934065934066

Epoch 5,phase: Training: 6it [00:01,  4.21it/s, train acc:
0.449,train loss:  2.105]

New best loss, loss is:  2.104274809360504 acc is:  0.4065934065934066

Epoch 6,phase: Training: 7it [00:01,  4.19it/s, train acc:
0.460,train loss:  2.090]

New best loss, loss is:  2.1002636353174844 acc is:
0.43956043956043955

Epoch 7,phase: Training: 8it [00:02,  4.15it/s, train acc:
0.497,train loss:  2.058]

Not a better score

Epoch 8,phase: Training: 9it [00:02,  4.05it/s, train acc:
0.497,train loss:  2.060]

New best loss, loss is:  2.066275159517924 acc is:  0.4945054945054945
```

```
Epoch 9,phase: Training: 10it [00:02,  4.07it/s, train acc:
0.529,train loss:  2.041]

Not a better score

Epoch 10,phase: Training: 11it [00:02,  3.71it/s, train acc:
0.509,train loss:  2.043]

Not a better score

Epoch 11,phase: Training: 12it [00:03,  3.73it/s, train acc:
0.516,train loss:  2.033]

Not a better score

Epoch 12,phase: Training: 13it [00:03,  3.86it/s, train acc:
0.533,train loss:  2.017]

New best loss, loss is:  2.0550614992777505 acc is:
0.5054945054945055

Epoch 13,phase: Training: 14it [00:03,  3.96it/s, train acc:
0.543,train loss:  2.009]

Not a better score

Epoch 14,phase: Training: 15it [00:03,  4.03it/s, train acc:
0.562,train loss:  2.004]

New best loss, loss is:  2.0540562868118286 acc is:
0.5164835164835165

Epoch 15,phase: Training: 16it [00:04,  3.98it/s, train acc:
0.554,train loss:  2.004]

New best loss, loss is:  2.052983601888021 acc is:  0.5164835164835165

Epoch 16,phase: Training: 17it [00:04,  3.95it/s, train acc:
0.592,train loss:  1.979]

New best loss, loss is:  2.0496842662493386 acc is:
0.5054945054945055

Epoch 17,phase: Training: 18it [00:04,  3.97it/s, train acc:
0.597,train loss:  1.967]

Not a better score

Epoch 18,phase: Training: 19it [00:04,  3.92it/s, train acc:
0.588,train loss:  1.978]

Not a better score
```

```
Epoch 19,phase: Training: 20it [00:05,  3.98it/s, train acc:
0.604,train loss:  1.962]

New best loss, loss is:  2.0460827151934304 acc is:
0.5054945054945055

Epoch 20,phase: Training: 21it [00:05,  4.01it/s, train acc:
0.608,train loss:  1.957]

New best loss, loss is:  2.0381917357444763 acc is:
0.5054945054945055

Epoch 21,phase: Training: 22it [00:05,  3.95it/s, train acc:
0.631,train loss:  1.947]

Not a better score

Epoch 22,phase: Training: 23it [00:05,  4.00it/s, train acc:
0.603,train loss:  1.954]

New best loss, loss is:  2.0280549923578897 acc is:
0.5274725274725275

Epoch 23,phase: Training: 24it [00:06,  3.94it/s, train acc:
0.622,train loss:  1.945]

Not a better score

Epoch 24,phase: Training: 25it [00:06,  4.00it/s, train acc:
0.611,train loss:  1.950]

Not a better score

Epoch 25,phase: Training: 26it [00:06,  4.04it/s, train acc:
0.625,train loss:  1.933]

Not a better score

Epoch 26,phase: Training: 27it [00:06,  3.98it/s, train acc:
0.636,train loss:  1.925]

Not a better score

Epoch 27,phase: Training: 28it [00:07,  4.02it/s, train acc:
0.639,train loss:  1.927]

New best loss, loss is:  2.026554306348165 acc is:  0.5164835164835165

Epoch 28,phase: Training: 29it [00:07,  4.05it/s, train acc:
0.647,train loss:  1.922]

New best loss, loss is:  2.0230512221654258 acc is:
0.5164835164835165
```

```
Epoch 29,phase: Training: 30it [00:07,  3.99it/s, train acc:
0.642,train loss:  1.917]

Not a better score

Epoch 30,phase: Training: 31it [00:07,  4.04it/s, train acc:
0.642,train loss:  1.920]

Not a better score

Epoch 31,phase: Training: 32it [00:08,  4.07it/s, train acc:
0.637,train loss:  1.919]

Not a better score

Epoch 32,phase: Training: 33it [00:08,  4.00it/s, train acc:
0.645,train loss:  1.919]

Not a better score

Epoch 33,phase: Training: 34it [00:08,  4.05it/s, train acc:
0.630,train loss:  1.921]

Not a better score

Epoch 34,phase: Training: 35it [00:08,  4.09it/s, train acc:
0.647,train loss:  1.909]

Not a better score

Epoch 35,phase: Training: 36it [00:09,  4.00it/s, train acc:
0.646,train loss:  1.909]

New best loss, loss is:  2.017632305622101 acc is:  0.5054945054945055

Epoch 36,phase: Training: 37it [00:09,  4.00it/s, train acc:
0.648,train loss:  1.909]

Not a better score

Epoch 37,phase: Training: 38it [00:09,  4.04it/s, train acc:
0.644,train loss:  1.913]

Not a better score

Epoch 38,phase: Training: 39it [00:09,  3.99it/s, train acc:
0.642,train loss:  1.909]

Not a better score

Epoch 39,phase: Training: 40it [00:10,  4.04it/s, train acc:
0.653,train loss:  1.901]

Not a better score
```

```
Epoch 40,phase: Training: 41it [00:10,  4.07it/s, train acc:
0.650,train loss:  1.906]

Not a better score

Epoch 41,phase: Training: 42it [00:10,  3.99it/s, train acc:
0.667,train loss:  1.884]

New best loss, loss is:  2.017304539680481 acc is:  0.5054945054945055

Epoch 42,phase: Training: 43it [00:10,  4.00it/s, train acc:
0.653,train loss:  1.898]

Not a better score

Epoch 43,phase: Training: 44it [00:11,  4.04it/s, train acc:
0.656,train loss:  1.895]

Not a better score

Epoch 44,phase: Training: 45it [00:11,  3.99it/s, train acc:
0.662,train loss:  1.889]

New best loss, loss is:  2.0112019181251526 acc is:
0.5274725274725275

Epoch 45,phase: Training: 46it [00:11,  4.03it/s, train acc:
0.642,train loss:  1.906]

Not a better score

Epoch 46,phase: Training: 47it [00:11,  4.06it/s, train acc:
0.659,train loss:  1.893]

Not a better score

Epoch 47,phase: Training: 48it [00:12,  3.99it/s, train acc:
0.659,train loss:  1.888]

Not a better score

Epoch 48,phase: Training: 49it [00:12,  4.02it/s, train acc:
0.656,train loss:  1.892]

Not a better score

Epoch 49,phase: Training: 50it [00:12,  4.04it/s, train acc:
0.644,train loss:  1.903]

Not a better score

Epoch 50,phase: Training: 51it [00:12,  3.98it/s, train acc:
0.656,train loss:  1.891]
```

```
Not a better score

Epoch 51,phase: Training: 52it [00:13,  4.02it/s, train acc:
0.644,train loss:  1.888]

Not a better score

Epoch 52,phase: Training: 53it [00:13,  4.08it/s, train acc:
0.650,train loss:  1.895]

Not a better score

Epoch 53,phase: Training: 54it [00:13,  4.01it/s, train acc:
0.656,train loss:  1.889]

Not a better score

Epoch 54,phase: Training: 55it [00:13,  4.05it/s, train acc:
0.665,train loss:  1.881]

Not a better score

Epoch 55,phase: Training: 56it [00:14,  4.08it/s, train acc:
0.644,train loss:  1.903]

Not a better score

Epoch 56,phase: Training: 57it [00:14,  4.03it/s, train acc:
0.659,train loss:  1.886]

Not a better score

Epoch 57,phase: Training: 58it [00:14,  4.07it/s, train acc:
0.659,train loss:  1.885]

Not a better score

Epoch 58,phase: Training: 59it [00:14,  4.09it/s, train acc:
0.641,train loss:  1.905]

Not a better score

Epoch 59,phase: Training: 60it [00:15,  4.01it/s, train acc:
0.656,train loss:  1.886]

Not a better score

Epoch 60,phase: Training: 61it [00:15,  4.06it/s, train acc:
0.668,train loss:  1.884]

Not a better score

Epoch 61,phase: Training: 62it [00:15,  4.08it/s, train acc:
0.648,train loss:  1.897]
```

```
Not a better score

Epoch 62,phase: Training: 63it [00:15,  3.99it/s, train acc:
0.665,train loss:  1.881]

Not a better score

Epoch 63,phase: Training: 64it [00:16,  4.03it/s, train acc:
0.664,train loss:  1.884]

Not a better score

Epoch 64,phase: Training: 65it [00:16,  4.05it/s, train acc:
0.669,train loss:  1.878]

Not a better score

Epoch 65,phase: Training: 66it [00:16,  3.97it/s, train acc:
0.665,train loss:  1.881]

Not a better score

Epoch 66,phase: Training: 67it [00:16,  4.01it/s, train acc:
0.676,train loss:  1.872]

Not a better score

Epoch 67,phase: Training: 68it [00:17,  4.02it/s, train acc:
0.672,train loss:  1.875]

Not a better score

Epoch 68,phase: Training: 69it [00:17,  3.96it/s, train acc:
0.668,train loss:  1.878]

Not a better score

Epoch 69,phase: Training: 70it [00:17,  4.01it/s, train acc:
0.665,train loss:  1.880]

Not a better score

Epoch 70,phase: Training: 71it [00:17,  4.03it/s, train acc:
0.666,train loss:  1.878]

Not a better score

Epoch 71,phase: Training: 72it [00:18,  3.97it/s, train acc:
0.668,train loss:  1.876]

Not a better score

Epoch 72,phase: Training: 73it [00:18,  4.02it/s, train acc:
0.668,train loss:  1.876]
```

```
Not a better score

Epoch 73,phase: Training: 74it [00:18,  4.04it/s, train acc:
0.666,train loss:  1.878]

Not a better score

Epoch 74,phase: Training: 75it [00:18,  3.98it/s, train acc:
0.668,train loss:  1.875]

Not a better score

Epoch 75,phase: Training: 76it [00:19,  4.02it/s, train acc:
0.668,train loss:  1.875]

Not a better score

Epoch 76,phase: Training: 77it [00:19,  4.06it/s, train acc:
0.684,train loss:  1.859]

Not a better score

Epoch 77,phase: Training: 78it [00:19,  4.00it/s, train acc:
0.665,train loss:  1.877]

Not a better score

Epoch 78,phase: Training: 79it [00:19,  4.05it/s, train acc:
0.673,train loss:  1.869]

Not a better score

Epoch 79,phase: Training: 80it [00:19,  4.09it/s, train acc:
0.662,train loss:  1.881]

Not a better score

Epoch 80,phase: Training: 81it [00:20,  4.02it/s, train acc:
0.658,train loss:  1.884]

Not a better score

Epoch 81,phase: Training: 82it [00:20,  4.03it/s, train acc:
0.670,train loss:  1.874]

Not a better score

Epoch 82,phase: Training: 83it [00:20,  4.06it/s, train acc:
0.659,train loss:  1.881]

Not a better score

Epoch 83,phase: Training: 84it [00:20,  3.99it/s, train acc:
0.670,train loss:  1.873]
```

```
Not a better score

Epoch 84,phase: Training: 85it [00:21,  4.04it/s, train acc:
0.668,train loss:  1.875]

Not a better score

Epoch 85,phase: Training: 86it [00:21,  4.07it/s, train acc:
0.669,train loss:  1.874]

Not a better score

Epoch 86,phase: Training: 87it [00:21,  4.00it/s, train acc:
0.668,train loss:  1.874]

Not a better score

Epoch 87,phase: Training: 88it [00:21,  4.03it/s, train acc:
0.661,train loss:  1.881]

Not a better score

Epoch 88,phase: Training: 89it [00:22,  3.96it/s, train acc:
0.670,train loss:  1.873]

Not a better score

Epoch 89,phase: Training: 90it [00:22,  3.98it/s, train acc:
0.670,train loss:  1.871]

Not a better score

Epoch 90,phase: Training: 91it [00:22,  4.02it/s, train acc:
0.670,train loss:  1.871]

Not a better score

Epoch 91,phase: Training: 92it [00:22,  3.98it/s, train acc:
0.664,train loss:  1.877]

Not a better score

Epoch 92,phase: Training: 93it [00:23,  3.99it/s, train acc:
0.679,train loss:  1.863]

Not a better score

Epoch 93,phase: Training: 94it [00:23,  4.01it/s, train acc:
0.670,train loss:  1.870]

New best loss, loss is:  2.0053385694821677 acc is:
0.5274725274725275
```

```
Epoch 94,phase: Training: 95it [00:23,  3.94it/s, train acc:
0.667,train loss:  1.874]

Not a better score

Epoch 95,phase: Training: 96it [00:23,  3.98it/s, train acc:
0.676,train loss:  1.864]

Not a better score

Epoch 96,phase: Training: 97it [00:24,  4.01it/s, train acc:
0.673,train loss:  1.867]

Not a better score

Epoch 97,phase: Training: 98it [00:24,  3.96it/s, train acc:
0.673,train loss:  1.867]

Not a better score

Epoch 98,phase: Training: 99it [00:24,  4.00it/s, train acc:
0.673,train loss:  1.867]

Not a better score

Epoch 99,phase: Training: 100it [00:24,  4.04it/s, train acc:
0.673,train loss:  1.867]

Not a better score

Epoch 100,phase: Training: 101it [00:25,  4.01it/s, train acc:
0.673,train loss:  1.868]

Not a better score

Epoch 101,phase: Training: 102it [00:25,  4.01it/s, train acc:
0.670,train loss:  1.870]

Not a better score

Epoch 102,phase: Training: 103it [00:25,  4.04it/s, train acc:
0.668,train loss:  1.872]

Not a better score

Epoch 103,phase: Training: 104it [00:25,  4.00it/s, train acc:
0.665,train loss:  1.875]

Not a better score

Epoch 104,phase: Training: 105it [00:26,  4.02it/s, train acc:
0.662,train loss:  1.881]

Not a better score
```

```
Epoch 105,phase: Training: 106it [00:26,  4.05it/s, train acc:
0.634,train loss:  1.912]

Not a better score

Epoch 106,phase: Training: 107it [00:26,  3.99it/s, train acc:
0.673,train loss:  1.871]

Not a better score

Epoch 107,phase: Training: 108it [00:26,  4.03it/s, train acc:
0.670,train loss:  1.872]

Not a better score

Epoch 108,phase: Training: 109it [00:27,  4.05it/s, train acc:
0.668,train loss:  1.876]

Not a better score

Epoch 109,phase: Training: 110it [00:27,  3.98it/s, train acc:
0.676,train loss:  1.870]

Not a better score

Epoch 110,phase: Training: 111it [00:27,  3.99it/s, train acc:
0.679,train loss:  1.868]

Not a better score

Epoch 111,phase: Training: 112it [00:27,  4.01it/s, train acc:
0.668,train loss:  1.876]

Not a better score

Epoch 112,phase: Training: 113it [00:28,  3.98it/s, train acc:
0.667,train loss:  1.873]

Not a better score

Epoch 113,phase: Training: 114it [00:28,  3.99it/s, train acc:
0.673,train loss:  1.868]

Not a better score

Epoch 114,phase: Training: 115it [00:28,  4.02it/s, train acc:
0.679,train loss:  1.863]

Not a better score

Epoch 115,phase: Training: 116it [00:28,  3.99it/s, train acc:
0.685,train loss:  1.856]

Not a better score
```

```
Epoch 116,phase: Training: 117it [00:29,  3.98it/s, train acc:
0.682,train loss:  1.859]

Not a better score

Epoch 117,phase: Training: 118it [00:29,  4.01it/s, train acc:
0.682,train loss:  1.859]

Not a better score

Epoch 118,phase: Training: 119it [00:29,  3.93it/s, train acc:
0.676,train loss:  1.865]

Not a better score

Epoch 119,phase: Training: 120it [00:29,  3.96it/s, train acc:
0.679,train loss:  1.861]

Not a better score

Epoch 120,phase: Training: 121it [00:30,  4.01it/s, train acc:
0.679,train loss:  1.861]

Not a better score

Epoch 121,phase: Training: 122it [00:30,  3.95it/s, train acc:
0.673,train loss:  1.867]

Not a better score

Epoch 122,phase: Training: 123it [00:30,  4.00it/s, train acc:
0.673,train loss:  1.865]

Not a better score

Epoch 123,phase: Training: 124it [00:30,  4.04it/s, train acc:
0.688,train loss:  1.854]

Not a better score

Epoch 124,phase: Training: 125it [00:31,  3.96it/s, train acc:
0.679,train loss:  1.864]

Not a better score

Epoch 125,phase: Training: 126it [00:31,  3.98it/s, train acc:
0.679,train loss:  1.861]

Not a better score

Epoch 126,phase: Training: 127it [00:31,  4.03it/s, train acc:
0.670,train loss:  1.871]

Not a better score
```

```
Epoch 127,phase: Training: 128it [00:31,  3.96it/s, train acc:
0.682,train loss:  1.859]

Not a better score

Epoch 128,phase: Training: 129it [00:32,  4.00it/s, train acc:
0.682,train loss:  1.858]

Not a better score

Epoch 129,phase: Training: 130it [00:32,  4.03it/s, train acc:
0.690,train loss:  1.851]

Not a better score

Epoch 130,phase: Training: 131it [00:32,  3.97it/s, train acc:
0.685,train loss:  1.856]

Not a better score

Epoch 131,phase: Training: 132it [00:32,  4.01it/s, train acc:
0.690,train loss:  1.853]

Not a better score

Epoch 132,phase: Training: 133it [00:33,  4.03it/s, train acc:
0.667,train loss:  1.875]

Not a better score

Epoch 133,phase: Training: 134it [00:33,  3.96it/s, train acc:
0.679,train loss:  1.862]

Not a better score

Epoch 134,phase: Training: 135it [00:33,  3.99it/s, train acc:
0.676,train loss:  1.866]

Not a better score

Epoch 135,phase: Training: 136it [00:33,  4.00it/s, train acc:
0.667,train loss:  1.875]

Not a better score

Epoch 136,phase: Training: 137it [00:34,  3.90it/s, train acc:
0.673,train loss:  1.868]

Not a better score

Epoch 137,phase: Training: 138it [00:34,  3.93it/s, train acc:
0.682,train loss:  1.859]

Not a better score
```

```
Epoch 138,phase: Training: 139it [00:34,  3.97it/s, train acc:
0.688,train loss:  1.853]

Not a better score

Epoch 139,phase: Training: 140it [00:35,  3.91it/s, train acc:
0.682,train loss:  1.858]

Not a better score

Epoch 140,phase: Training: 141it [00:35,  3.95it/s, train acc:
0.688,train loss:  1.856]

Not a better score

Epoch 141,phase: Training: 142it [00:35,  3.98it/s, train acc:
0.702,train loss:  1.840]

Not a better score

Epoch 142,phase: Training: 143it [00:35,  3.91it/s, train acc:
0.679,train loss:  1.864]

Not a better score

Epoch 143,phase: Training: 144it [00:36,  3.96it/s, train acc:
0.673,train loss:  1.868]

Not a better score

Epoch 144,phase: Training: 145it [00:36,  4.00it/s, train acc:
0.696,train loss:  1.843]

Not a better score

Epoch 145,phase: Training: 146it [00:36,  3.93it/s, train acc:
0.685,train loss:  1.855]

Not a better score

Epoch 146,phase: Training: 147it [00:36,  3.97it/s, train acc:
0.685,train loss:  1.855]

Not a better score

Epoch 147,phase: Training: 148it [00:37,  4.00it/s, train acc:
0.697,train loss:  1.845]

Not a better score

Epoch 148,phase: Training: 149it [00:37,  3.89it/s, train acc:
0.685,train loss:  1.857]

Not a better score
```

```
Epoch 149,phase: Training: 150it [00:37,  3.91it/s, train acc:
0.693,train loss:  1.848]

Not a better score

Epoch 150,phase: Training: 151it [00:37,  3.94it/s, train acc:
0.685,train loss:  1.855]

Not a better score

Epoch 151,phase: Training: 152it [00:38,  3.87it/s, train acc:
0.685,train loss:  1.858]

Not a better score

Epoch 152,phase: Training: 153it [00:38,  3.89it/s, train acc:
0.690,train loss:  1.850]

Not a better score

Epoch 153,phase: Training: 154it [00:38,  3.91it/s, train acc:
0.670,train loss:  1.867]

Not a better score

Epoch 154,phase: Training: 155it [00:38,  3.84it/s, train acc:
0.699,train loss:  1.842]

Not a better score

Epoch 155,phase: Training: 156it [00:39,  3.87it/s, train acc:
0.685,train loss:  1.856]

Not a better score

Epoch 156,phase: Training: 157it [00:39,  3.90it/s, train acc:
0.696,train loss:  1.843]

Not a better score

Epoch 157,phase: Training: 158it [00:39,  3.85it/s, train acc:
0.690,train loss:  1.850]

Not a better score

Epoch 158,phase: Training: 159it [00:39,  3.89it/s, train acc:
0.690,train loss:  1.852]

Not a better score

Epoch 159,phase: Training: 160it [00:40,  3.93it/s, train acc:
0.688,train loss:  1.855]

Not a better score
```

```
Epoch 160,phase: Training: 161it [00:40,  3.85it/s, train acc:
0.690,train loss:  1.856]

New best loss, loss is:  2.0019946098327637 acc is:
0.5384615384615384

Epoch 161,phase: Training: 162it [00:40,  3.87it/s, train acc:
0.696,train loss:  1.850]

Not a better score

Epoch 162,phase: Training: 163it [00:40,  3.89it/s, train acc:
0.679,train loss:  1.862]

Not a better score

Epoch 163,phase: Training: 164it [00:41,  3.85it/s, train acc:
0.690,train loss:  1.847]

Not a better score

Epoch 164,phase: Training: 165it [00:41,  3.87it/s, train acc:
0.679,train loss:  1.868]

Not a better score

Epoch 165,phase: Training: 166it [00:41,  3.90it/s, train acc:
0.658,train loss:  1.883]

Not a better score

Epoch 166,phase: Training: 167it [00:41,  3.85it/s, train acc:
0.670,train loss:  1.874]

New best loss, loss is:  1.98457537094752 acc is:  0.5494505494505495

Epoch 167,phase: Training: 168it [00:42,  3.93it/s, train acc:
0.676,train loss:  1.872]

Not a better score

Epoch 168,phase: Training: 169it [00:42,  3.99it/s, train acc:
0.693,train loss:  1.853]

Not a better score

Epoch 169,phase: Training: 170it [00:42,  3.93it/s, train acc:
0.693,train loss:  1.847]

New best loss, loss is:  1.977031449476878 acc is:  0.5604395604395604

Epoch 170,phase: Training: 171it [00:42,  4.00it/s, train acc:
0.682,train loss:  1.840]
```

```
Not a better score

Epoch 171,phase: Training: 172it [00:43,  4.06it/s, train acc:
0.685,train loss:  1.851]

Not a better score

Epoch 172,phase: Training: 173it [00:43,  4.00it/s, train acc:
0.696,train loss:  1.847]

Not a better score

Epoch 173,phase: Training: 174it [00:43,  3.99it/s, train acc:
0.693,train loss:  1.847]

Not a better score

Epoch 174,phase: Training: 175it [00:43,  3.99it/s, train acc:
0.693,train loss:  1.849]

Not a better score

Epoch 175,phase: Training: 176it [00:44,  3.87it/s, train acc:
0.700,train loss:  1.841]

Not a better score

Epoch 176,phase: Training: 177it [00:44,  3.84it/s, train acc:
0.690,train loss:  1.849]

Not a better score

Epoch 177,phase: Training: 178it [00:44,  3.92it/s, train acc:
0.684,train loss:  1.857]

Not a better score

Epoch 178,phase: Training: 179it [00:44,  3.83it/s, train acc:
0.705,train loss:  1.835]

Not a better score

Epoch 179,phase: Training: 180it [00:45,  3.87it/s, train acc:
0.696,train loss:  1.844]

Not a better score

Epoch 180,phase: Training: 181it [00:45,  3.91it/s, train acc:
0.679,train loss:  1.861]

Not a better score

Epoch 181,phase: Training: 182it [00:45,  3.93it/s, train acc:
0.702,train loss:  1.836]
```

```
Not a better score

Epoch 182,phase: Training: 183it [00:45,  3.95it/s, train acc:
0.702,train loss:  1.837]

Not a better score

Epoch 183,phase: Training: 184it [00:46,  3.96it/s, train acc:
0.705,train loss:  1.834]

Not a better score

Epoch 184,phase: Training: 185it [00:46,  3.99it/s, train acc:
0.705,train loss:  1.834]

Not a better score

Epoch 185,phase: Training: 186it [00:46,  4.00it/s, train acc:
0.693,train loss:  1.847]

Not a better score

Epoch 186,phase: Training: 187it [00:46,  4.02it/s, train acc:
0.693,train loss:  1.845]

Not a better score

Epoch 187,phase: Training: 188it [00:47,  4.02it/s, train acc:
0.688,train loss:  1.851]

Not a better score

Epoch 188,phase: Training: 189it [00:47,  4.01it/s, train acc:
0.697,train loss:  1.845]

Not a better score

Epoch 189,phase: Training: 190it [00:47,  3.93it/s, train acc:
0.702,train loss:  1.837]

Not a better score

Epoch 190,phase: Training: 191it [00:47,  3.96it/s, train acc:
0.693,train loss:  1.846]

Not a better score

Epoch 191,phase: Training: 192it [00:48,  3.99it/s, train acc:
0.691,train loss:  1.849]

Not a better score

Epoch 192,phase: Training: 193it [00:48,  3.95it/s, train acc:
0.682,train loss:  1.843]
```

```
Not a better score

Epoch 193,phase: Training: 194it [00:48,  4.04it/s, train acc:
0.699,train loss:  1.840]

Not a better score

Epoch 194,phase: Training: 195it [00:48,  4.09it/s, train acc:
0.700,train loss:  1.841]

Not a better score

Epoch 195,phase: Training: 196it [00:49,  4.01it/s, train acc:
0.690,train loss:  1.850]

Not a better score

Epoch 196,phase: Training: 197it [00:49,  4.06it/s, train acc:
0.705,train loss:  1.835]

Not a better score

Epoch 197,phase: Training: 198it [00:49,  4.08it/s, train acc:
0.694,train loss:  1.848]

Not a better score

Epoch 198,phase: Training: 199it [00:49,  4.03it/s, train acc:
0.688,train loss:  1.838]

Not a better score

Epoch 199,phase: Training: 200it [00:50,  4.09it/s, train acc:
0.682,train loss:  1.841]

Not a better score

Epoch 199,phase: Validation: 200it [00:50,  3.98it/s, val acc:
0.521,val loss:  1.989]

Not a better score
Training ended returning the best model
Best val acc: 0.5604395604395604, Best val loss: 1.977031449476878,
Best train acc: 0.6972222222222222, Best train loss:
1.8463929114134416
```

Part 1.3: Add Pooling Layers

We will now add max pooling layers after each of our convolutional layers. This code has already been provided for you; all you need to do is switch the pooling flag in the hyper-parameters to True, and choose different values for the pooling filter size and stride. After you applied max

pooling, what happened to your results? How did the training accuracy vs. validation accuracy change? What does that tell you about the effect of max pooling on your network?

```
model = SimpleCNN(device=device,pooling=True)
optimizer = optim.Adam(model.parameters(), lr=LR)
trainer_pooling = Trainer(model, criterion, train_loader, val_loader,
optimizer, num_epoch=NUM_EPOCHS,
patience=PATIENCE,batch_size=BATCH_SIZE,lr_scheduler= None)
best_model_pooling = trainer_pooling.train()
```

```
Epoch 0,phase: Training: 1it [00:00, 67650.06it/s, train_loss : ?,
train_acc: ?]/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-
packages/torch/utils/data/dataloader.py:684: UserWarning: 'pin_memory'
argument is set as true but not supported on MPS now, then device
pinned memory won't be used.
  warnings.warn(warn_msg)
/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-packages/
torch/nn/modules/module.py:1773: UserWarning: Implicit dimension
choice for softmax has been deprecated. Change the call to include
dim=X as an argument.
  return self._call_impl(*args, **kwargs)
Epoch 2,phase: Training: 3it [00:00,  6.51it/s, val acc:  0.292,val
loss:  2.388]

New best loss, loss is:  2.3930155436197915 acc is:
0.06593406593406594
New best loss, loss is:  2.3880619208017984 acc is:
0.3076923076923077

Epoch 3,phase: Validation: 4it [00:00,  5.89it/s, val acc:  0.275,val
loss:  2.330]

New best loss, loss is:  2.376756946245829 acc is:  0.3076923076923077

Epoch 4,phase: Validation: 5it [00:01,  5.54it/s, val acc:  0.312,val
loss:  2.230]

New best loss, loss is:  2.3328760067621865 acc is:
0.26373626373626374

Epoch 6,phase: Training: 7it [00:01,  5.21it/s, train acc:
0.375,train loss:  2.290]

New best loss, loss is:  2.2305721441904702 acc is:
0.34065934065934067
New best loss, loss is:  2.2106505632400513 acc is:
0.3516483516483517

Epoch 7,phase: Validation: 8it [00:01,  5.16it/s, val acc:  0.375,val
loss:  2.150]
```

```
New best loss, loss is:  2.1952991485595703 acc is:
0.37362637362637363

Epoch 9,phase: Training: 10it [00:01,  5.04it/s, train acc:
0.312,train loss:  2.248]

New best loss, loss is:  2.1799850861231485 acc is:
0.3626373626373626
New best loss, loss is:  2.1636544863382974 acc is:
0.3626373626373626

Epoch 10,phase: Validation: 11it [00:02,  5.07it/s, val acc:
0.344,val loss:  2.183]

Not a better score

Epoch 11,phase: Validation: 12it [00:02,  5.01it/s, val acc:
0.333,val loss:  2.191]

New best loss, loss is:  2.160581350326538 acc is:  0.3626373626373626

Epoch 13,phase: Training: 14it [00:02,  4.99it/s, val acc:  0.354,val
loss:  2.152]

Not a better score
New best loss, loss is:  2.1520769397417703 acc is:
0.37362637362637363

Epoch 14,phase: Validation: 15it [00:03,  5.03it/s, val acc:
0.391,val loss:  2.130]

New best loss, loss is:  2.1511805852254233 acc is:
0.37362637362637363

Epoch 15,phase: Validation: 16it [00:03,  4.98it/s, val acc:
0.375,val loss:  2.147]

Not a better score

Epoch 17,phase: Training: 18it [00:03,  4.99it/s, train acc:
0.438,train loss:  2.074]

New best loss, loss is:  2.1468201875686646 acc is:
0.37362637362637363
New best loss, loss is:  2.1424872080485025 acc is:
0.37362637362637363

Epoch 18,phase: Validation: 19it [00:03,  4.95it/s, val acc:
0.391,val loss:  2.152]

New best loss, loss is:  2.1246910889943442 acc is:
0.37362637362637363
```

```
Epoch 20,phase: Training: 21it [00:04,  4.96it/s, val acc:  0.375,val
loss:  2.132]

Not a better score
Not a better score

Epoch 22,phase: Training: 23it [00:04,  5.02it/s, val acc:  0.375,val
loss:  2.130]

Not a better score
Not a better score

Epoch 24,phase: Training: 25it [00:04,  5.00it/s, train acc:
0.562,train loss:  1.979]

Not a better score
Not a better score

Epoch 25,phase: Validation: 26it [00:05,  5.02it/s, val acc:
0.359,val loss:  2.172]

Not a better score

Epoch 27,phase: Training: 28it [00:05,  4.94it/s, val acc:  0.375,val
loss:  2.114]

New best loss, loss is:  2.121850232283274 acc is:  0.4065934065934066
New best loss, loss is:  2.1139520406723022 acc is:
0.3956043956043956

Epoch 29,phase: Training: 30it [00:05,  5.03it/s, val acc:  0.396,val
loss:  2.121]

Not a better score
Not a better score

Epoch 31,phase: Training: 32it [00:06,  4.97it/s, val acc:  0.396,val
loss:  2.109]

Not a better score
New best loss, loss is:  2.109045465787252 acc is:  0.4175824175824176

Epoch 33,phase: Training: 34it [00:06,  5.04it/s, train acc:
0.500,train loss:  2.096]

Not a better score
Not a better score

Epoch 35,phase: Training: 36it [00:07,  5.00it/s, train acc:
0.438,train loss:  2.099]
```

```
New best loss, loss is:  2.1018571853637695 acc is:
0.4065934065934066
Not a better score

Epoch 36,phase: Validation: 37it [00:07,  5.03it/s, val acc:
0.450,val loss:  2.078]

Not a better score

Epoch 37,phase: Validation: 38it [00:07,  4.99it/s, val acc:
0.375,val loss:  2.156]

Not a better score

Epoch 39,phase: Training: 40it [00:07,  4.99it/s, train acc:
0.500,train loss:  2.123]

New best loss, loss is:  2.085433761278788 acc is:
0.42857142857142855
New best loss, loss is:  2.0823932687441506 acc is:
0.42857142857142855

Epoch 40,phase: Validation: 41it [00:08,  4.98it/s, val acc:
0.438,val loss:  2.101]

Not a better score

Epoch 42,phase: Training: 43it [00:08,  4.94it/s, val acc:  0.406,val
loss:  2.108]

Not a better score
Not a better score

Epoch 43,phase: Validation: 44it [00:08,  5.00it/s, val acc:
0.406,val loss:  2.114]

Not a better score

Epoch 44,phase: Validation: 45it [00:09,  4.95it/s, val acc:
0.375,val loss:  2.123]

Not a better score

Epoch 46,phase: Training: 47it [00:09,  4.94it/s, val acc:  0.406,val
loss:  2.097]

Not a better score
Not a better score

Epoch 47,phase: Validation: 48it [00:09,  5.00it/s, val acc:
0.438,val loss:  2.096]

Not a better score
```

```
Epoch 48,phase: Validation: 49it [00:09,  4.97it/s, val acc:
0.542,val loss:  2.016]

Not a better score

Epoch 50,phase: Training: 51it [00:10,  4.96it/s, train acc:
0.438,train loss:  2.063]

Not a better score
Not a better score

Epoch 51,phase: Validation: 52it [00:10,  5.01it/s, val acc:
0.453,val loss:  2.101]

Not a better score

Epoch 53,phase: Training: 54it [00:10,  4.94it/s, val acc:  0.417,val
loss:  2.097]

Not a better score
Not a better score

Epoch 55,phase: Training: 56it [00:11,  4.91it/s, train acc:
0.625,train loss:  1.961]

Not a better score
Not a better score

Epoch 57,phase: Training: 58it [00:11,  5.00it/s, val acc:  0.417,val
loss:  2.103]

Not a better score
Not a better score

Epoch 59,phase: Training: 60it [00:11,  4.96it/s, train acc:
0.625,train loss:  1.939]

Not a better score
Not a better score

Epoch 60,phase: Validation: 61it [00:12,  5.00it/s, val acc:
0.463,val loss:  2.074]

Not a better score

Epoch 61,phase: Validation: 62it [00:12,  4.97it/s, val acc:
0.479,val loss:  2.053]

Not a better score

Epoch 63,phase: Training: 64it [00:12,  4.95it/s, train acc:
0.375,train loss:  2.170]
```

```
Not a better score
Not a better score

Epoch 65,phase: Training: 66it [00:13,  5.01it/s, val acc:  0.417,val
loss:  2.086]

Not a better score
Not a better score

Epoch 67,phase: Training: 68it [00:13,  5.00it/s, train acc:
0.562,train loss:  2.005]

Not a better score
Not a better score

Epoch 68,phase: Validation: 69it [00:13,  5.00it/s, val acc:
0.450,val loss:  2.085]

Not a better score

Epoch 69,phase: Validation: 70it [00:14,  4.97it/s, val acc:
0.396,val loss:  2.132]

Not a better score

Epoch 71,phase: Training: 72it [00:14,  4.96it/s, train acc:
0.625,train loss:  1.995]

Not a better score
Not a better score

Epoch 72,phase: Validation: 73it [00:14,  4.96it/s, val acc:
0.438,val loss:  2.106]

New best loss, loss is:  2.082081437110901 acc is:
0.43956043956043955

Epoch 74,phase: Training: 75it [00:14,  4.92it/s, train acc:
0.625,train loss:  1.934]

Not a better score
Not a better score

Epoch 76,phase: Training: 77it [00:15,  5.02it/s, val acc:  0.406,val
loss:  2.111]

New best loss, loss is:  2.0810994108517966 acc is:
0.45054945054945056
Not a better score

Epoch 78,phase: Training: 79it [00:15,  4.99it/s, train acc:
0.750,train loss:  1.807]
```

```
Not a better score
Not a better score

Epoch 80,phase: Training: 81it [00:16,  5.03it/s, val acc:  0.417,val
loss:  2.082]

Not a better score
Not a better score

Epoch 82,phase: Training: 83it [00:16,  5.00it/s, train acc:
0.438,train loss:  2.126]

Not a better score
Not a better score

Epoch 84,phase: Training: 85it [00:16,  5.05it/s, val acc:  0.417,val
loss:  2.088]

Not a better score
Not a better score

Epoch 86,phase: Training: 87it [00:17,  5.02it/s, train acc:
0.500,train loss:  2.041]

Not a better score
Not a better score

Epoch 87,phase: Validation: 88it [00:17,  5.01it/s, val acc:
0.388,val loss:  2.133]

Not a better score

Epoch 88,phase: Validation: 89it [00:17,  4.99it/s, val acc:
0.406,val loss:  2.113]

New best loss, loss is:  2.0787150462468467 acc is:
0.42857142857142855

Epoch 90,phase: Training: 91it [00:18,  4.92it/s, val acc:  0.417,val
loss:  2.090]

Not a better score
Not a better score

Epoch 91,phase: Validation: 92it [00:18,  4.96it/s, val acc:
0.625,val loss:  1.937]

Not a better score

Epoch 93,phase: Training: 94it [00:18,  4.92it/s, train acc:
0.375,train loss:  2.168]
```

```
Not a better score
Not a better score

Epoch 94,phase: Validation: 95it [00:19,  4.97it/s, val acc:
0.484,val loss:  2.075]

Not a better score

Epoch 95,phase: Validation: 96it [00:19,  4.95it/s, val acc:
0.479,val loss:  2.068]

Not a better score

Epoch 97,phase: Training: 98it [00:19,  4.93it/s, val acc:  0.417,val
loss:  2.095]

Not a better score
Not a better score

Epoch 98,phase: Validation: 99it [00:20,  4.96it/s, val acc:
0.406,val loss:  2.130]

Not a better score

Epoch 99,phase: Validation: 100it [00:20,  4.86it/s, val acc:
0.412,val loss:  2.125]

Not a better score

Epoch 101,phase: Training: 102it [00:20,  4.97it/s, train acc:
0.688,train loss:  1.901]

Not a better score
Not a better score

Epoch 102,phase: Validation: 103it [00:20,  4.96it/s, val acc:
0.406,val loss:  2.094]

Not a better score

Epoch 104,phase: Training: 105it [00:21,  4.95it/s, train acc:
0.438,train loss:  2.130]

Not a better score
Not a better score

Epoch 106,phase: Training: 107it [00:21,  5.00it/s, val acc:
0.406,val loss:  2.098]

Not a better score
Not a better score
```

```
Epoch 108,phase: Training: 109it [00:21,  4.98it/s, train acc:
0.562,train loss:  1.974]

Not a better score
Not a better score

Epoch 109,phase: Validation: 110it [00:22,  5.00it/s, val acc:
0.425,val loss:  2.109]

Not a better score

Epoch 110,phase: Validation: 111it [00:22,  4.99it/s, val acc:
0.406,val loss:  2.136]

Not a better score

Epoch 112,phase: Training: 113it [00:22,  4.97it/s, train acc:
0.438,train loss:  2.107]

Not a better score
Not a better score

Epoch 113,phase: Validation: 114it [00:23,  5.00it/s, val acc:
0.479,val loss:  2.038]

New best loss, loss is:  2.071227173010508 acc is:
0.45054945054945056

Epoch 115,phase: Training: 116it [00:23,  4.97it/s, train acc:
0.562,train loss:  2.005]

Not a better score
Not a better score

Epoch 116,phase: Validation: 117it [00:23,  5.03it/s, val acc:
0.475,val loss:  2.081]

Not a better score

Epoch 117,phase: Validation: 118it [00:23,  5.01it/s, val acc:
0.406,val loss:  2.112]

Not a better score

Epoch 119,phase: Training: 120it [00:24,  4.95it/s, train acc:
0.625,train loss:  1.951]

Not a better score
Not a better score

Epoch 120,phase: Validation: 121it [00:24,  5.01it/s, val acc:
0.412,val loss:  2.104]
```

```
Not a better score

Epoch 121,phase: Validation: 122it [00:24,  4.99it/s, val acc:
0.458,val loss:  2.057]

Not a better score

Epoch 123,phase: Training: 124it [00:24,  4.93it/s, val acc:
0.406,val loss:  2.095]

Not a better score
Not a better score

Epoch 124,phase: Validation: 125it [00:25,  5.00it/s, val acc:
0.531,val loss:  2.023]

Not a better score

Epoch 126,phase: Training: 127it [00:25,  4.95it/s, val acc:
0.417,val loss:  2.100]

Not a better score
Not a better score

Epoch 127,phase: Validation: 128it [00:25,  4.98it/s, val acc:
0.500,val loss:  2.071]

Not a better score

Epoch 128,phase: Validation: 129it [00:26,  4.93it/s, val acc:
0.500,val loss:  2.034]

Not a better score

Epoch 130,phase: Training: 131it [00:26,  4.93it/s, train acc:
0.750,train loss:  1.845]

Not a better score
Not a better score

Epoch 131,phase: Validation: 132it [00:26,  4.99it/s, val acc:
0.375,val loss:  2.164]

Not a better score

Epoch 133,phase: Training: 134it [00:26,  4.92it/s, val acc:
0.406,val loss:  2.099]

Not a better score
Not a better score

Epoch 135,phase: Training: 136it [00:27,  4.99it/s, val acc:
0.438,val loss:  2.097]
```

```
Not a better score
Not a better score

Epoch 137,phase: Training: 138it [00:27,  4.99it/s, train acc:
0.500,train loss:  2.039]

Not a better score
Not a better score

Epoch 138,phase: Validation: 140it [00:28,  5.01it/s, val acc:
0.448,val loss:  2.090]

Not a better score
Not a better score

Epoch 141,phase: Training: 142it [00:28,  4.97it/s, train acc:
0.438,train loss:  2.101]

Not a better score
Not a better score

Epoch 142,phase: Validation: 143it [00:28,  5.01it/s, val acc:
0.484,val loss:  2.095]

Not a better score

Epoch 144,phase: Training: 145it [00:29,  4.96it/s, val acc:
0.448,val loss:  2.083]

Not a better score
Not a better score

Epoch 145,phase: Validation: 146it [00:29,  5.01it/s, val acc:
0.475,val loss:  2.089]

Not a better score

Epoch 146,phase: Validation: 147it [00:29,  4.99it/s, val acc:
0.438,val loss:  2.112]

Not a better score

Epoch 148,phase: Training: 149it [00:29,  4.93it/s, val acc:
0.438,val loss:  2.090]

New best loss, loss is:  2.070476214090983 acc is:  0.4835164835164835
Not a better score

Epoch 149,phase: Validation: 150it [00:30,  4.97it/s, val acc:
0.500,val loss:  2.075]

Not a better score
```

```
Epoch 150,phase: Validation: 151it [00:30,  4.96it/s, val acc:
0.500,val loss:  2.042]

Not a better score

Epoch 152,phase: Training: 153it [00:30,  4.93it/s, val acc:
0.417,val loss:  2.076]

Not a better score
Not a better score

Epoch 153,phase: Validation: 154it [00:31,  4.98it/s, val acc:
0.484,val loss:  2.092]

Not a better score

Epoch 154,phase: Validation: 155it [00:31,  4.89it/s, val acc:
0.450,val loss:  2.088]

Not a better score

Epoch 156,phase: Training: 157it [00:31,  4.99it/s, train acc:
0.688,train loss:  1.856]

Not a better score
Not a better score

Epoch 157,phase: Validation: 158it [00:31,  5.00it/s, val acc:
0.396,val loss:  2.142]

Not a better score

Epoch 159,phase: Training: 160it [00:32,  4.95it/s, val acc:
0.448,val loss:  2.093]

Not a better score
Not a better score

Epoch 160,phase: Validation: 161it [00:32,  5.00it/s, val acc:
0.417,val loss:  2.091]

Not a better score

Epoch 161,phase: Validation: 162it [00:32,  5.00it/s, val acc:
0.396,val loss:  2.134]

Not a better score

Epoch 163,phase: Training: 164it [00:32,  4.98it/s, train acc:
0.750,train loss:  1.799]
```

```
New best loss, loss is:  2.0697630842526755 acc is:
0.45054945054945056
Not a better score

Epoch 164,phase: Validation: 165it [00:33,  5.01it/s, val acc:
0.453,val loss:  2.068]

Not a better score

Epoch 165,phase: Validation: 166it [00:33,  4.97it/s, val acc:
0.438,val loss:  2.111]

Not a better score

Epoch 167,phase: Training: 168it [00:33,  4.93it/s, train acc:
0.438,train loss:  2.106]

Not a better score
Not a better score

Epoch 169,phase: Training: 170it [00:34,  5.01it/s, val acc:
0.438,val loss:  2.086]

Not a better score
Not a better score

Epoch 171,phase: Training: 172it [00:34,  4.98it/s, train acc:
0.562,train loss:  1.981]

Not a better score
Not a better score

Epoch 172,phase: Validation: 173it [00:34,  4.98it/s, val acc:
0.450,val loss:  2.092]

Not a better score

Epoch 173,phase: Validation: 174it [00:35,  4.95it/s, val acc:
0.475,val loss:  2.061]

Not a better score

Epoch 175,phase: Training: 176it [00:35,  4.99it/s, val acc:
0.427,val loss:  2.078]

Not a better score
Not a better score

Epoch 176,phase: Validation: 177it [00:35,  5.01it/s, val acc:
0.562,val loss:  2.001]

Not a better score
```

```
Epoch 178,phase: Training: 179it [00:35,  4.91it/s, val acc:
0.438,val loss:  2.076]

Not a better score
Not a better score

Epoch 180,phase: Training: 181it [00:36,  5.00it/s, val acc:
0.427,val loss:  2.090]

Not a better score
Not a better score

Epoch 182,phase: Training: 183it [00:36,  4.96it/s, train acc:
0.562,train loss:  1.983]

Not a better score
Not a better score

Epoch 183,phase: Validation: 184it [00:37,  4.99it/s, val acc:
0.438,val loss:  2.078]

New best loss, loss is:  2.0640054742495217 acc is:
0.4725274725274725

Epoch 184,phase: Validation: 185it [00:37,  4.99it/s, val acc:
0.500,val loss:  2.042]

Not a better score

Epoch 186,phase: Training: 187it [00:37,  4.95it/s, val acc:
0.448,val loss:  2.080]

Not a better score
Not a better score

Epoch 187,phase: Validation: 188it [00:38,  4.97it/s, val acc:
0.500,val loss:  2.055]

Not a better score

Epoch 188,phase: Validation: 189it [00:38,  4.84it/s, val acc:
0.463,val loss:  2.093]

Not a better score

Epoch 190,phase: Training: 191it [00:38,  4.94it/s, val acc:
0.438,val loss:  2.077]

Not a better score
Not a better score

Epoch 191,phase: Validation: 192it [00:38,  4.96it/s, val acc:
0.406,val loss:  2.134]
```

```
Not a better score

Epoch 193,phase: Training: 194it [00:39,  4.92it/s, val acc:
0.458,val loss:  2.071]

Not a better score
Not a better score

Epoch 194,phase: Validation: 195it [00:39,  5.00it/s, val acc:
0.448,val loss:  2.065]

Not a better score

Epoch 195,phase: Validation: 196it [00:39,  4.99it/s, val acc:
0.406,val loss:  2.166]

Not a better score

Epoch 197,phase: Training: 198it [00:39,  4.94it/s, val acc:
0.469,val loss:  2.072]

Not a better score
Not a better score

Epoch 199,phase: Training: 200it [00:40,  5.00it/s, val acc:
0.448,val loss:  2.072]

Not a better score
Not a better score

Epoch 199,phase: Validation: 200it [00:40,  4.94it/s, val acc:
0.469,val loss:  2.070]

Not a better score
Training ended returning the best model
Best val acc: 0.4725274725274725, Best val loss: 2.0640054742495217,
Best train acc: 0.6055555555555555, Best train loss:
1.9404017717941948
```

Answer here:

Best val acc: 0.4725274725274725, Best val loss: 2.0640054742495217, Best train acc: 0.6055555555555555, Best train loss: 1.9404017717941948 It got worse since the max pooliing is for dealing with Overfiitiing and add more general.

Part 1.4: Regularize Your Network!

Because this is such a small dataset, your network is likely to overfit the data. Implement the following ways of regularizing your network. Test each one individually, and discuss how it affects your results.

- **Dropout**: In PyTorch, this is implemented using the `torch.nn.dropout` class, which takes a value called the `keep_prob`, representing the probability that an activation will be dropped out. This value should be between 0.1 and 0.5 during training, and 0 for evaluation and testing. An example of how this works is available here. You should add this to your network and try different values to find one that works well.
- **Weight Regularization**: You should try different optimizers, and different weight decay values for optimizers.
- **Early Stopping**: Stop training your model after your validation accuracy starts to plateau or decrease (so you do not overtrain your model). The number of steps can be controlled through the `patience` hyperparameter in the code.
- **Learning Rate Scheduling**: Learning rate scheduling is an important part of training neural networks. There are a lot of techniques for learning rate scheduling. You should try different schedulers such as `StepLR`, `CosineAnnealing`, etc.

Give your results for each of these regularization techniques, and discuss which ones were the most effective.

Answer here:

```
class New_Trainer():
    def __init__(self, model, criterion, tr_loader, val_loader,
optimizer, num_epoch, patience, batch_size, lr_scheduler=None):
        self.model = model
        self.tr_loader = tr_loader
        self.val_loader = val_loader
        self.optimizer = optimizer
        self.num_epoch = num_epoch
        self.patience = patience
        self.lr_scheduler = lr_scheduler
        self.criterion = criterion
        # Use correct device for tensors, set dim=1 for softmax over
classes
        self.softmax = nn.Softmax(dim=1)
        self.no_inc = 0
        self.best_loss = float('inf')
        self.phases = ["train", "val"]
        self.best_model = None
        self.best_val_acc = 0
        self.best_train_acc = 0
        self.best_val_loss = 0
        self.best_train_loss = 0
        self.batch_size = batch_size

    def train(self):
        pbar = tqdm.tqdm(desc="Epoch 0, phase: Training",
```

```
postfix="train_loss: ?, train_acc: ?")
        for epoch in range(self.num_epoch):
            last_train_acc = 0
            last_val_acc = 0
            last_val_loss = 0
            last_train_loss = 0
            pbar.update(1)

            for phase in self.phases:
                total_acc = 0
                total_loss = 0
                start = time.time()
                if phase == "train":
                    pbar.set_description_str(f"Epoch {epoch}, phase:
Training")
                    loader = self.tr_loader
                    self.model.train()
                else:
                    pbar.set_description_str(f"Epoch {epoch}, phase:
Validation")
                    loader = self.val_loader
                    self.model.eval()
                num_batches = 0
                for images, labels in loader:
                    num_batches += 1
                    images = images.to(self.model.device)
                    labels = labels.to(self.model.device)

                    self.optimizer.zero_grad()
                    logits = self.model(images)
                    # If labels are one-hot encoded, use argmax; else,
leave as is
                    if labels.ndim > 1 and labels.size(1) > 1:
                        labels_for_loss = torch.argmax(labels, dim=1)
                    else:
                        labels_for_loss = labels

                    softmaxed_scores = self.softmax(logits)
                    _, predictions = torch.max(softmaxed_scores, 1)
                    loss = self.criterion(softmaxed_scores.float(),
labels_for_loss.long())
                    total_loss += loss.item()
                    total_acc += (predictions ==
labels_for_loss).sum().item()

                    if phase == "train":
                        pbar.set_postfix_str(f"train acc: {total_acc /
(num_batches * self.batch_size):6.3f}, train loss: {total_loss /
num_batches:6.3f}")
                        loss.backward()
```

```
                        self.optimizer.step()
                    else:
                        pbar.set_postfix_str(f"val acc: {total_acc /
(num_batches * self.batch_size):6.3f}, val loss: {total_loss /
num_batches:6.3f}")

                if phase == "train":
                    if self.lr_scheduler:
                        self.lr_scheduler.step()
                end = time.time()
                if phase == "train":
                    loss_epoch = total_loss / num_batches
                    acc_epoch = total_acc /
len(self.tr_loader.dataset)
                    last_train_acc = acc_epoch
                    last_train_loss = loss_epoch
                else:
                    loss_epoch = total_loss / num_batches
                    acc_epoch = total_acc /
len(self.val_loader.dataset)
                    last_val_acc = acc_epoch
                    last_val_loss = loss_epoch

                    if loss_epoch < self.best_loss:
                        print(f"New best loss, loss is: {loss_epoch},
acc is: {acc_epoch}")
                        self.best_loss = loss_epoch
                        self.no_inc = 0
                        # Save a deep copy of the best model to avoid
in-place modification
                        self.best_model = self.model
                        self.best_train_acc = last_train_acc
                        self.best_train_loss = last_train_loss
                        self.best_val_loss = last_val_loss
                        self.best_val_acc = last_val_acc
                    else:
                        print("Not a better score")
                        self.no_inc += 1
                        if self.no_inc >= self.patience:
                            print("Out of patience returning the best
model")
                            print(
                                "Best val acc: {}, Best val loss: {},
Best train acc: {}, Best train loss: {} ".format(
                                    self.best_val_acc,
self.best_val_loss, self.best_train_acc, self.best_train_loss
                                )
                            )  # Stats of the best model
                            return self.best_model
```

```
        print("Training ended returning the best model")
        print(
            "Best val acc: {}, Best val loss: {}, Best train acc: {},
Best train loss: {} ".format(
                self.best_val_acc, self.best_val_loss,
self.best_train_acc, self.best_train_loss
            )
        )  # Stats of the best model
        return self.best_model
```

Part 1.5: Experiment with Your Architecture

All those parameters at the top of `SimpleCNN` still need to be set. You cannot possibly explore all combinations; so try to change some of them individually to get some feeling for their effect (if any). Optionally, you can explore adding more layers. Report which changes led to the biggest increases and decreases in performance. In particular, what is the effect of making the convolutional layers have (a) a larger filter size, (b) a larger stride and (c) greater depth? How does a pyramidal-shaped network in which the feature maps gradually decrease in height and width but increase in depth compare to a flat architecture, or one with the opposite shape?

Answer here:

Part 1.6: Optimize Your Architecture

Based on your experience with these tests, try to achieve the best performance that you can on the validation set by varying the hyperparameters, architecture, and regularization methods. You can even (optionally) try to think of additional ways to augment the data, or experiment with techniques like local response normalization layers using `torch.nn.LocalResponseNorm` or weight normalization using the implementation here. Report the best performance you are able to achieve, and the settings you used to obtain it.

Answer here:

Part 1.7: Test Your Final Architecture on Variations of the Data

In PyTorch data augmentation can be done dynamically while loading the data using what they call `transforms`. Note that some of the transforms are already implemented. You can try other transformations, such as the ones shown in Figure 3 and also try different probabilities for these transformations. You may find this link helpful. Note that the PyTorch data loader refreshes the data in each epoch and apply different transformations to the different instances.

Now that you have optimized your architecture, you are ready to test it on augmented data! Report your performance on each of the transformed datasets. Are you surprised by any of the results? Which transformations is your network most invariant to, and which lead it to be unable to recognize the images? What does that tell you about what features your network has learned to use to recognize artists' images?

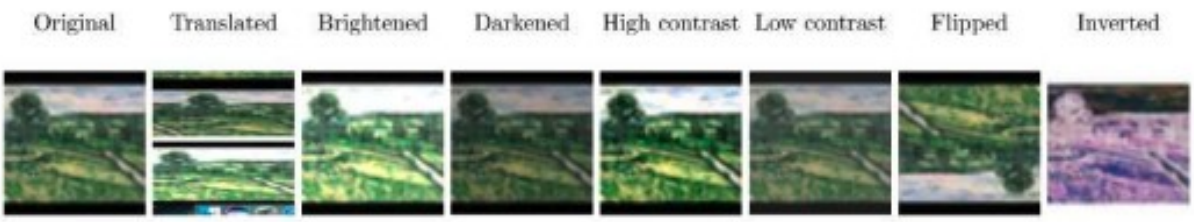


Figure 3: Image transformations used in data augmention.

Part 2: Transfer Learning with Deep Network

In this part, you will fine-tune AlexNet model pretrained on ImageNet to recognize faces. For the sake of simplicity you may use the pretrained AlexNet model provided in PyTorch Hub. You will work with a subset of the FaceScrub dataset. The subset of male actors is here and the subset of female actors is here. The dataset consists of URLs of images with faces, as well as the bounding boxes of the faces. The format of the bounding box is as follows (from the FaceScrub `readme.txt` file):

```
The format is x1,y1,x2,y2, where (x1,y1) is the coordinate of the top-left corner of the bounding box and (x2,y2) is that of the bottom-right corner, with (0,0) as the top-left corner of the image. Assuming the image is represented as a Python NumPy array I, a face in I can be obtained as I[y1:y2, x1:x2].
```

You may find it helpful to use and/or modify this script for downloading the image data. Note that you should crop out the images of the faces and resize them to appropriate size before proceeding further. Make sure to check the SHA-256 hashes, and make sure to only keep faces for which the hashes match. You should set aside 70 images per faces for the training set, and use the rest for the test and validation set.

Part 2.0: Get the Data

```
import os
import urllib.request
import pandas as pd
from tqdm import tqdm

if not os.path.exists('facescrub'):
    os.makedirs('facescrub', exist_ok=True)
if not os.path.exists('facescrub/actors'):
    os.makedirs('facescrub/actors', exist_ok=True)
if not os.path.exists('facescrub/actresses'):
    os.makedirs('facescrub/actresses', exist_ok=True)

ACTORS_URL =
"http://www.cs.toronto.edu/~guerzhoy/321/proj1/subset_actors.txt"
ACTRESSES_URL =
"http://www.cs.toronto.edu/~guerzhoy/321/proj1/subset_actresses.txt"

urllib.request.urlretrieve(ACTORS_URL, 'facescrub/subset_actors.txt')
```

```
urllib.request.urlretrieve(ACTRESSES_URL,
'facescrub/subset_actresses.txt')

# Read data from subset_actors.txt
with open('facescrub/subset_actors.txt', 'r') as f:
    actors_lines = f.readlines()

# Read data from subset_actresses.txt
with open('facescrub/subset_actresses.txt', 'r') as f:
    actresses_lines = f.readlines()

print(f"Number of actor lines: {len(actors_lines)}")
print(f"Number of actress lines: {len(actresses_lines)}")

def download_image(url, dest_path, timeout=2):
    try:
        if not os.path.exists(dest_path):
            with urllib.request.urlopen(url, timeout=timeout) as r,
open(dest_path, 'wb') as f:
                f.write(r.read())
            # print(f"Downloaded: {dest_path}")
        # else:
            # print(f"Already exists: {dest_path}")
    except Exception as e:
        pass
        # print(f"Could not download {url}: {e}")

Data = pd.DataFrame({"name": [], "cords": []})

# Download actor images
for file, dir in zip(["actors", "actresses"], [actors_lines,
actresses_lines]):
    for line in tqdm(dir):
        parts = line.strip().split()
        if len(parts) < 2:
            continue
        url = [p for p in parts if "http" in p][0]
        img_name = "-".join([parts[i] for i in
range(parts.index(url))]) + ".png"
        dest_path = os.path.join('facescrub', file, img_name)
        cord = [p for p in parts if p.count(",") == 3][0]
        Data = pd.concat([Data, pd.DataFrame([{"name": img_name,
"cords": cord}])], ignore_index=True)
        download_image(url, dest_path)

Data.to_csv("facescrub/data.csv")
```

```
Number of actor lines: 3167
Number of actress lines: 3169
```

```
100%|██████████| 3167/3167 [26:42<00:00,  1.98it/s]
100%|██████████| 3169/3169 [39:55<00:00,  1.32it/s]
```

Part 2.1: Train a Multilayer Perceptron

First resize the images to 28 × 28 pixels. Use a fully-connected neural network with a single hidden layer of size 300 units. Below, include the learning curve for the test, training, and validation sets, and the final performance classification on the test set. Include a text description of your system. In particular, describe how you preprocessed the input and initialized the weights, what activation function you used, and what the exact architecture of the network that you selected was. You might get performances close to 80-85% accuracy rate.

DATA

```
import os
import numpy as np
import pandas as pd
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt
from torchvision import models

def process_image(img_path, cords, target_size=(28,28)):
    try:
        img = Image.open(img_path).convert("L")
        x1, y1, x2, y2 = [int(float(val)) for val in cords.split(',')]
        img = img.crop((x1, y1, x2, y2))
        img = img.resize(target_size, Image.BILINEAR)
        img = np.asarray(img, dtype=np.float32) / 255.0
        return img
    except Exception as e:
        return None

def split_data(size=(28,28)):
    data_df = pd.read_csv("facescrub/data.csv")
    le = LabelEncoder()
    data_df['label'] = le.fit_transform(data_df['name'].apply(lambda
x: x.split('-')[0] + ' ' + x.split('-')[1]))

    images = []
    labels = []
    for idx, row in data_df.iterrows():
```

```
        img_name = row['name']
        label = row['label']
        cords = row['cords']

        for img_dir in ["facescrub/actors", "facescrub/actresses"]:
            img_path = os.path.join(img_dir, img_name)
            if os.path.exists(img_path):
                img_arr = process_image(img_path, cords, size)
                if img_arr is not None:
                    images.append(img_arr)
                    labels.append(label)
                break

    images = np.array(images)
    labels = np.array(labels)

    # Split into train, val, test
    X_train, X_temp, y_train, y_temp = train_test_split(images,
labels, test_size=0.1, random_state=42, stratify=labels)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42, stratify=y_temp)
    print(len(X_train), len(X_val), len(X_test))
    print(len(y_train), len(y_val), len(y_test))

    return X_train, X_val, X_test, y_train, y_val, y_test

X_train, X_val, X_test, y_train, y_val, y_test = split_data()

class FaceDataset(Dataset):
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels

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

    def __getitem__(self, idx):
        img = self.images[idx]
        img = np.expand_dims(img, axis=0) # (1,28,28) for grayscale
        return torch.tensor(img, dtype=torch.float32),
torch.tensor(self.labels[idx], dtype=torch.long)

batch_size = 64
train_dataset = FaceDataset(X_train, y_train)
val_dataset = FaceDataset(X_val, y_val)
test_dataset = FaceDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
```

```
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
```

```
1615 90 90
1615 90 90
```

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

input_dim = 28*28
hidden_dim = 300
output_dim = len(np.unique(y_train))
model = MLP(input_dim, hidden_dim, output_dim)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
def init_weights_xavier(m):
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        if m.bias is not None:
            nn.init.zeros_(m.bias)
model.apply(init_weights_xavier)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

def evaluate(model, data_loader):
    model.eval()
    total = 0
    correct = 0
    loss_sum = 0
    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss_sum += loss.item() * images.size(0)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += images.size(0)
```

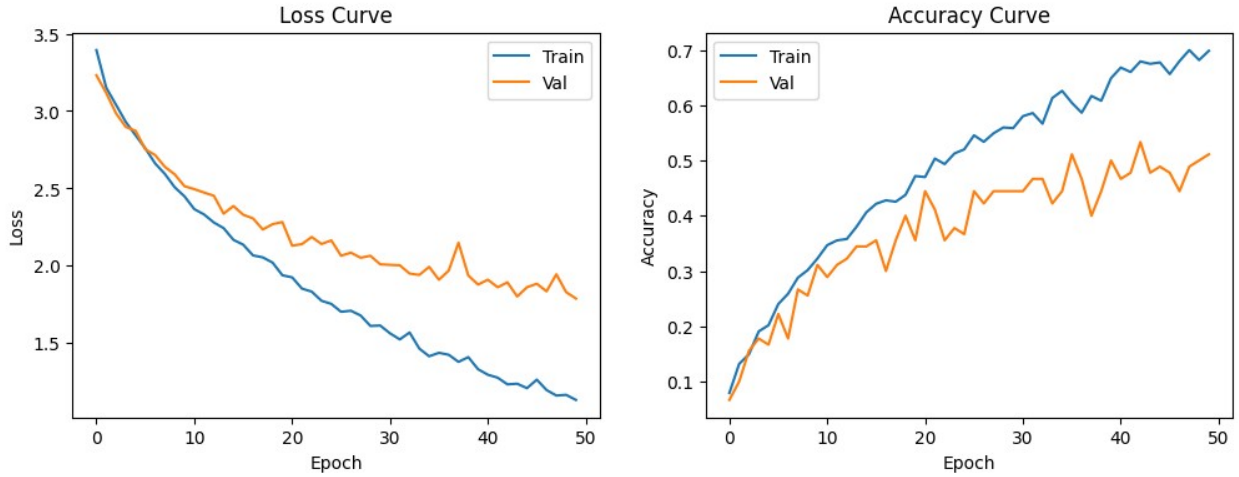
```
    return loss_sum/total, correct/total

num_epochs = 50
train_loss_hist = []
val_loss_hist = []
train_acc_hist = []
val_acc_hist = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0
    running_correct = 0
    running_total = 0
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        running_correct += (predicted == labels).sum().item()
        running_total += images.size(0)
    train_loss = running_loss / running_total
    train_acc = running_correct / running_total
    val_loss, val_acc = evaluate(model, val_loader)
    train_loss_hist.append(train_loss)
    val_loss_hist.append(val_loss)
    train_acc_hist.append(train_acc)
    val_acc_hist.append(val_acc)
    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss:
{train_loss:.4f} Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} Acc:
{val_acc:.4f}")

test_loss, test_acc = evaluate(model, test_loader)
print("\nTest Accuracy: {:.2f}%".format(test_acc*100))

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(train_loss_hist, label='Train')
plt.plot(val_loss_hist, label='Val')
plt.title("Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.subplot(1,2,2)
plt.plot(train_acc_hist, label='Train')
plt.plot(val_acc_hist, label='Val')
```

```
plt.title("Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
Epoch 1/50 | Train Loss: 3.3936 Acc: 0.0793 | Val Loss: 3.2312 Acc:
0.0667
Epoch 2/50 | Train Loss: 3.1510 Acc: 0.1319 | Val Loss: 3.1168 Acc:
0.1000
Epoch 3/50 | Train Loss: 3.0407 Acc: 0.1492 | Val Loss: 2.9855 Acc:
0.1556
Epoch 4/50 | Train Loss: 2.9294 Acc: 0.1907 | Val Loss: 2.8972 Acc:
0.1778
Epoch 5/50 | Train Loss: 2.8434 Acc: 0.2019 | Val Loss: 2.8723 Acc:
0.1667
Epoch 6/50 | Train Loss: 2.7583 Acc: 0.2402 | Val Loss: 2.7529 Acc:
0.2222
Epoch 7/50 | Train Loss: 2.6614 Acc: 0.2588 | Val Loss: 2.7155 Acc:
0.1778
Epoch 8/50 | Train Loss: 2.5932 Acc: 0.2879 | Val Loss: 2.6390 Acc:
0.2667
Epoch 9/50 | Train Loss: 2.5074 Acc: 0.3015 | Val Loss: 2.5910 Acc:
0.2556
Epoch 10/50 | Train Loss: 2.4488 Acc: 0.3226 | Val Loss: 2.5142 Acc:
0.3111
Epoch 11/50 | Train Loss: 2.3664 Acc: 0.3467 | Val Loss: 2.4955 Acc:
0.2889
Epoch 12/50 | Train Loss: 2.3317 Acc: 0.3554 | Val Loss: 2.4731 Acc:
0.3111
Epoch 13/50 | Train Loss: 2.2799 Acc: 0.3579 | Val Loss: 2.4527 Acc:
0.3222
Epoch 14/50 | Train Loss: 2.2433 Acc: 0.3802 | Val Loss: 2.3369 Acc:
0.3444
Epoch 15/50 | Train Loss: 2.1679 Acc: 0.4062 | Val Loss: 2.3863 Acc:
0.3444
Epoch 16/50 | Train Loss: 2.1365 Acc: 0.4217 | Val Loss: 2.3300 Acc:
0.3556
Epoch 17/50 | Train Loss: 2.0673 Acc: 0.4279 | Val Loss: 2.3064 Acc:
0.3000
Epoch 18/50 | Train Loss: 2.0546 Acc: 0.4254 | Val Loss: 2.2347 Acc:
0.3556
Epoch 19/50 | Train Loss: 2.0207 Acc: 0.4378 | Val Loss: 2.2691 Acc:
0.4000
Epoch 20/50 | Train Loss: 1.9391 Acc: 0.4718 | Val Loss: 2.2828 Acc:
0.3556
Epoch 21/50 | Train Loss: 1.9254 Acc: 0.4700 | Val Loss: 2.1302 Acc:
0.4444
Epoch 22/50 | Train Loss: 1.8530 Acc: 0.5034 | Val Loss: 2.1402 Acc:
0.4111
```

```
Epoch 23/50 | Train Loss: 1.8333 Acc: 0.4935 | Val Loss: 2.1863 Acc:
0.3556
Epoch 24/50 | Train Loss: 1.7742 Acc: 0.5127 | Val Loss: 2.1409 Acc:
0.3778
Epoch 25/50 | Train Loss: 1.7540 Acc: 0.5201 | Val Loss: 2.1642 Acc:
0.3667
Epoch 26/50 | Train Loss: 1.7029 Acc: 0.5455 | Val Loss: 2.0651 Acc:
0.4444
Epoch 27/50 | Train Loss: 1.7098 Acc: 0.5337 | Val Loss: 2.0850 Acc:
0.4222
Epoch 28/50 | Train Loss: 1.6788 Acc: 0.5492 | Val Loss: 2.0521 Acc:
0.4444
Epoch 29/50 | Train Loss: 1.6113 Acc: 0.5598 | Val Loss: 2.0648 Acc:
0.4444
Epoch 30/50 | Train Loss: 1.6141 Acc: 0.5585 | Val Loss: 2.0099 Acc:
0.4444
Epoch 31/50 | Train Loss: 1.5638 Acc: 0.5802 | Val Loss: 2.0063 Acc:
0.4444
Epoch 32/50 | Train Loss: 1.5245 Acc: 0.5858 | Val Loss: 2.0035 Acc:
0.4667
Epoch 33/50 | Train Loss: 1.5692 Acc: 0.5666 | Val Loss: 1.9499 Acc:
0.4667
Epoch 34/50 | Train Loss: 1.4649 Acc: 0.6130 | Val Loss: 1.9413 Acc:
0.4222
Epoch 35/50 | Train Loss: 1.4156 Acc: 0.6260 | Val Loss: 1.9933 Acc:
0.4444
Epoch 36/50 | Train Loss: 1.4378 Acc: 0.6043 | Val Loss: 1.9101 Acc:
0.5111
Epoch 37/50 | Train Loss: 1.4253 Acc: 0.5864 | Val Loss: 1.9690 Acc:
0.4667
Epoch 38/50 | Train Loss: 1.3794 Acc: 0.6167 | Val Loss: 2.1489 Acc:
0.4000
Epoch 39/50 | Train Loss: 1.4102 Acc: 0.6080 | Val Loss: 1.9375 Acc:
0.4444
Epoch 40/50 | Train Loss: 1.3318 Acc: 0.6489 | Val Loss: 1.8788 Acc:
0.5000
Epoch 41/50 | Train Loss: 1.2963 Acc: 0.6681 | Val Loss: 1.9102 Acc:
0.4667
Epoch 42/50 | Train Loss: 1.2772 Acc: 0.6601 | Val Loss: 1.8616 Acc:
0.4778
Epoch 43/50 | Train Loss: 1.2341 Acc: 0.6793 | Val Loss: 1.8931 Acc:
0.5333
Epoch 44/50 | Train Loss: 1.2381 Acc: 0.6749 | Val Loss: 1.8017 Acc:
0.4778
Epoch 45/50 | Train Loss: 1.2094 Acc: 0.6774 | Val Loss: 1.8616 Acc:
0.4889
Epoch 46/50 | Train Loss: 1.2645 Acc: 0.6563 | Val Loss: 1.8842 Acc:
0.4778
Epoch 47/50 | Train Loss: 1.1973 Acc: 0.6799 | Val Loss: 1.8352 Acc:
```

```
0.4444
Epoch 48/50 | Train Loss: 1.1619 Acc: 0.6997 | Val Loss: 1.9451 Acc:
0.4889
Epoch 49/50 | Train Loss: 1.1662 Acc: 0.6817 | Val Loss: 1.8301 Acc:
0.5000
Epoch 50/50 | Train Loss: 1.1334 Acc: 0.6985 | Val Loss: 1.7885 Acc:
0.5111

Test Accuracy: 52.22%
```



Part 2.2: AlexNet as a Fixed Feature Extractor

Extract the values of the activations of AlexNet on the face images. Use those as features in order to perform face classification: learn a fully-connected neural network that takes in the activations of the units in the AlexNet layer as inputs, and outputs the name of the person. Below, include a description of the system you built and its performance. It is recommended to start out with only using the `conv4` activations. Using `conv4` is sufficient here.

```
X_train, X_val, X_test, y_train, y_val, y_test = split_data((128,
128))

class TransferFaceDataset(Dataset):
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels

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

    def __getitem__(self, idx):
        img = self.images[idx]
        img = np.expand_dims(img, axis=0)  # (1,28,28)
        img = np.repeat(img, 3, axis=0)    # (3,28,28)
        return torch.tensor(img, dtype=torch.float32),
```

```
torch.tensor(self.labels[idx], dtype=torch.long)

batch_size = 64
train_dataset = TransferFaceDataset(X_train, y_train)
val_dataset = TransferFaceDataset(X_val, y_val)
test_dataset = TransferFaceDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
```

```
1615 90 90
1615 90 90
```

```
alexnet = models.alexnet(pretrained=True)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
alexnet = models.alexnet(pretrained=True)
alexnet = alexnet.to(device)
alexnet.eval()


class AlexNetConv4(nn.Module):
    def __init__(self, alexnet):
        super().__init__()
        # conv4 is features[8], so include features[0:9]
        self.features_up_to_conv4 =
nn.Sequential(*list(alexnet.features[:9]))

    def forward(self, x):
        return self.features_up_to_conv4(x)

alexnet_conv4 = AlexNetConv4(alexnet).to(device)
alexnet_conv4.eval()

def extract_conv4_features(dataloader):
    all_features = []
    all_labels = []
    with torch.no_grad():
        for images, labels in dataloader:
            images = images.to(device)
            acts = alexnet_conv4(images)
            feats = acts.view(acts.size(0), -1)   # Flatten spatial
dims
            all_features.append(feats.cpu())
            all_labels.append(labels)
    all_features = torch.cat(all_features, dim=0)
    all_labels = torch.cat(all_labels, dim=0)
    return all_features, all_labels
```

```
# Extract activations for train/val/test
X_train, y_train = extract_conv4_features(train_loader)
X_val, y_val = extract_conv4_features(val_loader)
X_test, y_test = extract_conv4_features(test_loader)

# Now define a simple fully-connected classifier on top of the conv4
features
num_classes = len(set(y_train.tolist()))
input_dim = X_train.size(1)

class FCClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_classes):
        super(FCClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, num_classes)
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Hyperparameters
hidden_dim = 128
num_epochs = 20
lr = 0.001
batch_size = 64

# Create model, criterion, and optimizer
fc_model = FCClassifier(input_dim, hidden_dim, num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(fc_model.parameters(), lr=lr)

# Create DataLoaders for feature tensors
train_dataset = torch.utils.data.TensorDataset(X_train, y_train)
val_dataset = torch.utils.data.TensorDataset(X_val, y_val)
test_dataset = torch.utils.data.TensorDataset(X_test, y_test)
train_feat_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
val_feat_loader = torch.utils.data.DataLoader(val_dataset,
batch_size=batch_size, shuffle=False)
test_feat_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=batch_size, shuffle=False)

# Training loop for FC classifier
train_loss_hist, val_loss_hist = [], []
train_acc_hist, val_acc_hist = [], []

def eval_classifier(model, loader):
    model.eval()
```

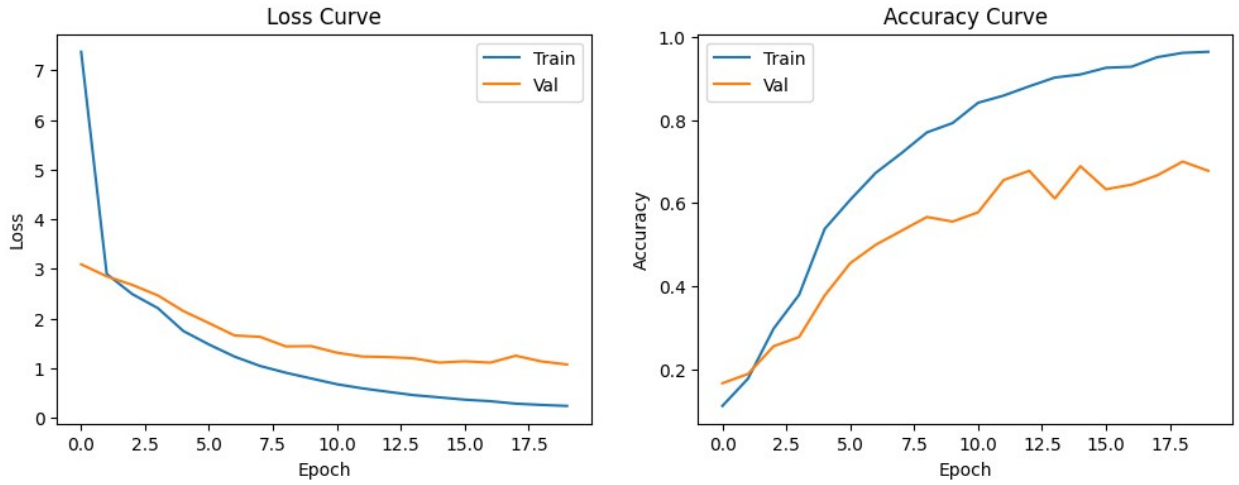
```
    running_loss = 0.0
    running_total = 0
    running_correct = 0
    with torch.no_grad():
        for X, y in loader:
            X = X.to(device)
            y = y.to(device)
            logits = model(X)
            loss = criterion(logits, y)
            running_loss += loss.item() * X.size(0)
            _, preds = torch.max(logits, 1)
            running_correct += (preds == y).sum().item()
            running_total += X.size(0)
    loss = running_loss / running_total
    acc = running_correct / running_total
    return loss, acc

for epoch in range(num_epochs):
    fc_model.train()
    running_loss = 0.0
    running_total = 0
    running_correct = 0
    for X, y in train_feat_loader:
        X = X.to(device)
        y = y.to(device)
        optimizer.zero_grad()
        logits = fc_model(X)
        loss = criterion(logits, y)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * X.size(0)
        _, preds = torch.max(logits, 1)
        running_correct += (preds == y).sum().item()
        running_total += X.size(0)
    train_loss = running_loss / running_total
    train_acc = running_correct / running_total
    val_loss, val_acc = eval_classifier(fc_model, val_feat_loader)
    train_loss_hist.append(train_loss)
    val_loss_hist.append(val_loss)
    train_acc_hist.append(train_acc)
    val_acc_hist.append(val_acc)
    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss:
{train_loss:.4f} Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f} Acc:
{val_acc:.4f}")

# Test set evaluation
test_loss, test_acc = eval_classifier(fc_model, test_feat_loader)
print(f"\nTest Accuracy with fixed conv4 features: {test_acc*100:.2f}
%")
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(train_loss_hist, label='Train')
plt.plot(val_loss_hist, label='Val')
plt.title("Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.subplot(1,2,2)
plt.plot(train_acc_hist, label='Train')
plt.plot(val_acc_hist, label='Val')
plt.title("Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-packages/
torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the
future, please use 'weights' instead.
  warnings.warn(
/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-packages/
torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may
be removed in the future. The current behavior is equivalent to
passing `weights=AlexNet_Weights.IMAGENET1K_V1`. You can also use
`weights=AlexNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
Epoch 1/20 | Train Loss: 7.3717 Acc: 0.1121 | Val Loss: 3.0904 Acc:
0.1667
Epoch 2/20 | Train Loss: 2.8961 Acc: 0.1777 | Val Loss: 2.8477 Acc:
0.1889
Epoch 3/20 | Train Loss: 2.4902 Acc: 0.2978 | Val Loss: 2.6755 Acc:
0.2556
Epoch 4/20 | Train Loss: 2.2072 Acc: 0.3796 | Val Loss: 2.4630 Acc:
0.2778
Epoch 5/20 | Train Loss: 1.7454 Acc: 0.5381 | Val Loss: 2.1480 Acc:
0.3778
Epoch 6/20 | Train Loss: 1.4749 Acc: 0.6080 | Val Loss: 1.9050 Acc:
0.4556
Epoch 7/20 | Train Loss: 1.2307 Acc: 0.6731 | Val Loss: 1.6564 Acc:
0.5000
Epoch 8/20 | Train Loss: 1.0421 Acc: 0.7201 | Val Loss: 1.6289 Acc:
0.5333
Epoch 9/20 | Train Loss: 0.9066 Acc: 0.7703 | Val Loss: 1.4367 Acc:
0.5667
Epoch 10/20 | Train Loss: 0.7902 Acc: 0.7926 | Val Loss: 1.4419 Acc:
```

```
0.5556
Epoch 11/20 | Train Loss: 0.6747 Acc: 0.8415 | Val Loss: 1.3094 Acc:
0.5778
Epoch 12/20 | Train Loss: 0.5911 Acc: 0.8588 | Val Loss: 1.2324 Acc:
0.6556
Epoch 13/20 | Train Loss: 0.5218 Acc: 0.8811 | Val Loss: 1.2212 Acc:
0.6778
Epoch 14/20 | Train Loss: 0.4550 Acc: 0.9022 | Val Loss: 1.1964 Acc:
0.6111
Epoch 15/20 | Train Loss: 0.4099 Acc: 0.9096 | Val Loss: 1.1099 Acc:
0.6889
Epoch 16/20 | Train Loss: 0.3636 Acc: 0.9257 | Val Loss: 1.1357 Acc:
0.6333
Epoch 17/20 | Train Loss: 0.3323 Acc: 0.9282 | Val Loss: 1.1092 Acc:
0.6444
Epoch 18/20 | Train Loss: 0.2829 Acc: 0.9511 | Val Loss: 1.2490 Acc:
0.6667
Epoch 19/20 | Train Loss: 0.2576 Acc: 0.9616 | Val Loss: 1.1334 Acc:
0.7000
Epoch 20/20 | Train Loss: 0.2375 Acc: 0.9641 | Val Loss: 1.0724 Acc:
0.6778

Test Accuracy with fixed conv4 features: 67.78%
```



Part 2.3: Visualize Weights

Train two networks the way you did in Part 2.1. Use 300 and 800 hidden units in the hidden layer. Visualize 2 different hidden features (neurons) for each of the two settings, and briefly explain why they are interesting. A sample visualization of a hidden feature is shown below. Note that you probably need to use L2 regularization while training to obtain nice weight visualizations.

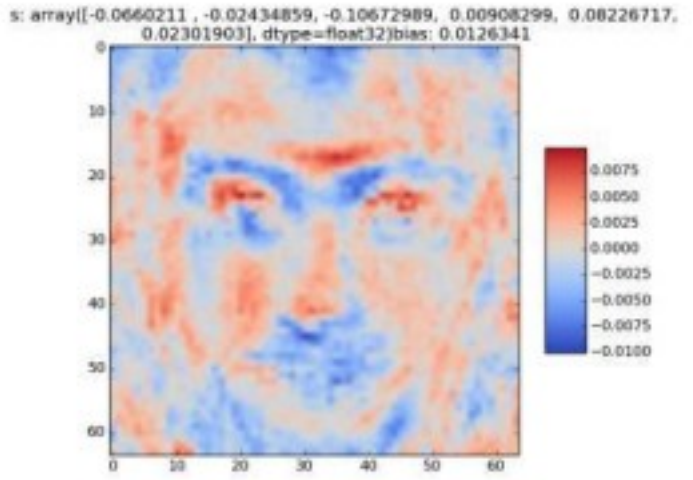


Figure 4: A sample visualization of a hidden feature.

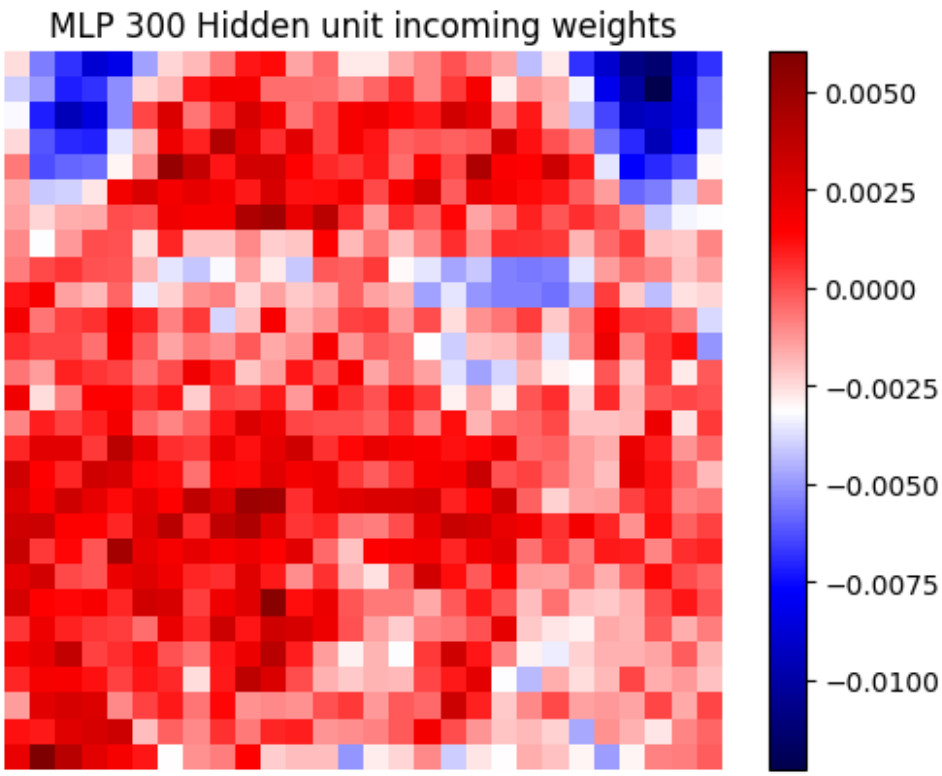
```
X_train, X_val, X_test, y_train, y_val, y_test = split_data()

class FaceDataset(Dataset):
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels

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

    def __getitem__(self, idx):
        img = self.images[idx]
        img = np.expand_dims(img, axis=0) # (1,28,28) for grayscale
        return torch.tensor(img, dtype=torch.float32),
torch.tensor(self.labels[idx], dtype=torch.long)

batch_size = 64
train_dataset = FaceDataset(X_train, y_train)
val_dataset = FaceDataset(X_val, y_val)
test_dataset = FaceDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, l2_reg):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.l2_reg = l2_reg  # Store L2 regularization value for use
```

```
if needed

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def l2_regularization(self):
        l2_norm = 0
        for param in self.parameters():
            l2_norm += torch.norm(param, 2)
        return self.l2_reg * l2_norm

def train_mlp(train_loader, val_loader, input_dim, hidden_units,
num_classes, l2_reg=1e-3, num_epochs=15):
    model = MLP(input_dim, hidden_dim, num_classes, l2_reg)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3,
weight_decay=l2_reg)
    train_loss_hist, val_loss_hist = [], []
    train_acc_hist, val_acc_hist = [], []

    for epoch in range(num_epochs):
        model.train()
        running_loss, running_correct, running_total = 0, 0, 0
        for X, y in train_loader:
            optimizer.zero_grad()
            logits = model(X)
            loss = criterion(logits, y)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * X.size(0)
            _, preds = torch.max(logits, 1)
            running_correct += (preds == y).sum().item()
            running_total += X.size(0)
        train_loss = running_loss / running_total
        train_acc = running_correct / running_total

        # validation
        model.eval()
        val_loss, val_correct, val_total = 0, 0, 0
        with torch.no_grad():
            for X, y in val_loader:
                logits = model(X)
                loss = criterion(logits, y)
                val_loss += loss.item() * X.size(0)
                _, preds = torch.max(logits, 1)
                val_correct += (preds == y).sum().item()
                val_total += X.size(0)
```

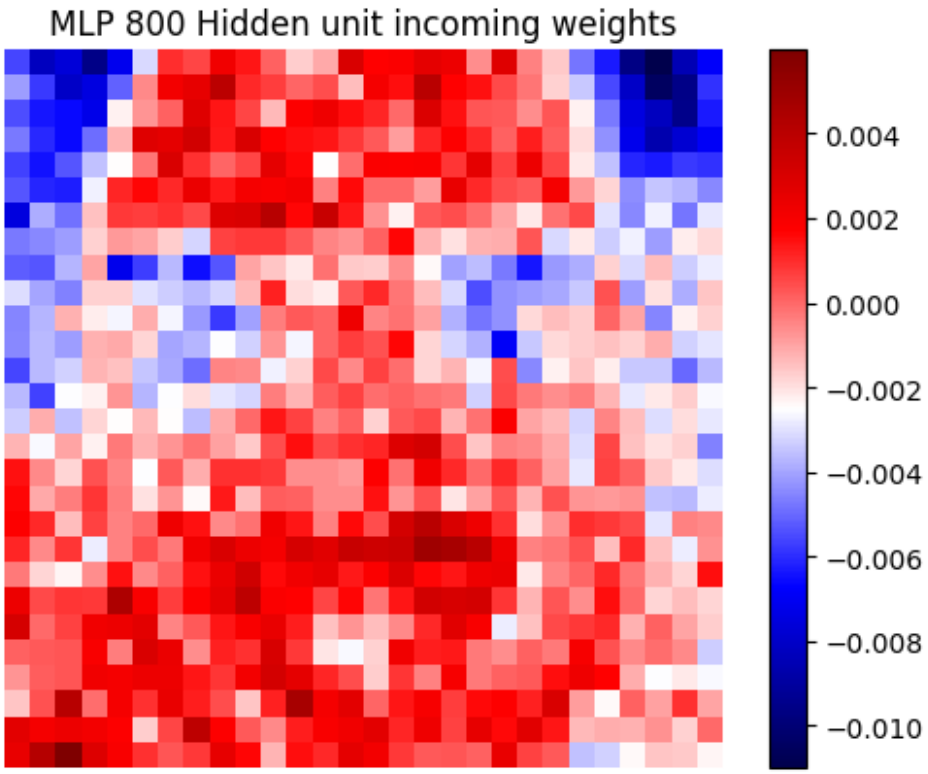
```
        val_loss = val_loss / val_total
        val_acc = val_correct / val_total

        train_loss_hist.append(train_loss)
        val_loss_hist.append(val_loss)
        train_acc_hist.append(train_acc)
        val_acc_hist.append(val_acc)
        print(f"H={hidden_units} Epoch {epoch+1}/{num_epochs} | Train
Loss: {train_loss:.4f} Acc: {train_acc:.4f} | Val Loss: {val_loss:.4f}
Acc: {val_acc:.4f}")

    return model

# Train models
input_dim = 28*28
num_classes = len(np.unique(y_train))

# Model with 300 hidden units
mlp_300 = train_mlp(train_loader, val_loader, input_dim, 300,
num_classes, l2_reg=5e-4, num_epochs=20)
# Model with 800 hidden units
mlp_800 = train_mlp(train_loader, val_loader, input_dim, 800,
num_classes, l2_reg=5e-4, num_epochs=20)

# Visualize two input weight vectors for 2 hidden neurons for both
models
def visualize_hidden_weights(model, title_prefix=""):
    weights = model.fc1.weight.detach().cpu().numpy()  # shape:
(hidden_units, feat_dim)
    print(weights.shape)
    weights = weights.reshape(weights.shape[0], 28, 28)
    w_img = weights.mean(0)
    print(w_img.shape)

    plt.figure()
    plt.imshow(w_img, cmap='seismic')
    plt.colorbar()
    plt.title(f"{title_prefix} Hidden unit incoming weights")
    plt.axis('off')
    plt.show()

# 300 hidden unit model
print("\nVisualizing hidden weights for 300 hidden units model:")
visualize_hidden_weights(mlp_300, title_prefix="MLP 300")

# 800 hidden unit model
print("\nVisualizing hidden weights for 800 hidden units model:")
visualize_hidden_weights(mlp_800, title_prefix="MLP 800")
```

```
1615 90 90
1615 90 90
H=300 Epoch 1/20 | Train Loss: 3.3374 Acc: 0.0774 | Val Loss: 3.2741
Acc: 0.0889
H=300 Epoch 2/20 | Train Loss: 3.2379 Acc: 0.1189 | Val Loss: 3.1669
Acc: 0.1222
H=300 Epoch 3/20 | Train Loss: 3.1581 Acc: 0.1288 | Val Loss: 3.0790
Acc: 0.1000
H=300 Epoch 4/20 | Train Loss: 3.0829 Acc: 0.1319 | Val Loss: 3.0193
Acc: 0.1222
H=300 Epoch 5/20 | Train Loss: 3.0295 Acc: 0.1449 | Val Loss: 2.9564
Acc: 0.1556
H=300 Epoch 6/20 | Train Loss: 2.9448 Acc: 0.1777 | Val Loss: 2.9357
Acc: 0.1111
H=300 Epoch 7/20 | Train Loss: 2.8847 Acc: 0.1944 | Val Loss: 2.8620
Acc: 0.1667
H=300 Epoch 8/20 | Train Loss: 2.8301 Acc: 0.1988 | Val Loss: 2.8336
Acc: 0.1667
H=300 Epoch 9/20 | Train Loss: 2.7680 Acc: 0.2365 | Val Loss: 2.7718
Acc: 0.1667
H=300 Epoch 10/20 | Train Loss: 2.7125 Acc: 0.2303 | Val Loss: 2.7322
Acc: 0.2111
H=300 Epoch 11/20 | Train Loss: 2.6607 Acc: 0.2693 | Val Loss: 2.7009
Acc: 0.1889
H=300 Epoch 12/20 | Train Loss: 2.5887 Acc: 0.2663 | Val Loss: 2.6500
Acc: 0.2333
H=300 Epoch 13/20 | Train Loss: 2.5500 Acc: 0.2935 | Val Loss: 2.6044
Acc: 0.2333
H=300 Epoch 14/20 | Train Loss: 2.5032 Acc: 0.2885 | Val Loss: 2.5943
Acc: 0.2556
H=300 Epoch 15/20 | Train Loss: 2.4550 Acc: 0.3276 | Val Loss: 2.5834
Acc: 0.2444
H=300 Epoch 16/20 | Train Loss: 2.4237 Acc: 0.3313 | Val Loss: 2.5382
Acc: 0.3000
H=300 Epoch 17/20 | Train Loss: 2.3873 Acc: 0.3257 | Val Loss: 2.5440
Acc: 0.2444
H=300 Epoch 18/20 | Train Loss: 2.3433 Acc: 0.3486 | Val Loss: 2.4793
Acc: 0.2333
H=300 Epoch 19/20 | Train Loss: 2.2971 Acc: 0.3598 | Val Loss: 2.4474
Acc: 0.2889
H=300 Epoch 20/20 | Train Loss: 2.2674 Acc: 0.3734 | Val Loss: 2.4476
Acc: 0.2667
H=800 Epoch 1/20 | Train Loss: 3.3539 Acc: 0.0724 | Val Loss: 3.2838
Acc: 0.1222
H=800 Epoch 2/20 | Train Loss: 3.2521 Acc: 0.1195 | Val Loss: 3.1783
Acc: 0.1000
H=800 Epoch 3/20 | Train Loss: 3.1718 Acc: 0.1183 | Val Loss: 3.0997
Acc: 0.1222
H=800 Epoch 4/20 | Train Loss: 3.1018 Acc: 0.1443 | Val Loss: 3.0490
Acc: 0.0889
```

```
H=800 Epoch 5/20 | Train Loss: 3.0261 Acc: 0.1467 | Val Loss: 2.9721
Acc: 0.1556
H=800 Epoch 6/20 | Train Loss: 2.9581 Acc: 0.1672 | Val Loss: 2.9312
Acc: 0.1222
H=800 Epoch 7/20 | Train Loss: 2.9081 Acc: 0.1808 | Val Loss: 2.8825
Acc: 0.1556
H=800 Epoch 8/20 | Train Loss: 2.8513 Acc: 0.2043 | Val Loss: 2.8477
Acc: 0.1778
H=800 Epoch 9/20 | Train Loss: 2.8005 Acc: 0.2155 | Val Loss: 2.8439
Acc: 0.1889
H=800 Epoch 10/20 | Train Loss: 2.7501 Acc: 0.2384 | Val Loss: 2.7685
Acc: 0.2000
H=800 Epoch 11/20 | Train Loss: 2.7062 Acc: 0.2316 | Val Loss: 2.7848
Acc: 0.1889
H=800 Epoch 12/20 | Train Loss: 2.6424 Acc: 0.2570 | Val Loss: 2.7022
Acc: 0.3111
H=800 Epoch 13/20 | Train Loss: 2.5940 Acc: 0.2861 | Val Loss: 2.6808
Acc: 0.2333
H=800 Epoch 14/20 | Train Loss: 2.5621 Acc: 0.2774 | Val Loss: 2.6417
Acc: 0.2778
H=800 Epoch 15/20 | Train Loss: 2.5118 Acc: 0.2947 | Val Loss: 2.6042
Acc: 0.2444
H=800 Epoch 16/20 | Train Loss: 2.4774 Acc: 0.3046 | Val Loss: 2.5777
Acc: 0.2667
H=800 Epoch 17/20 | Train Loss: 2.4340 Acc: 0.3226 | Val Loss: 2.5414
Acc: 0.3000
H=800 Epoch 18/20 | Train Loss: 2.3927 Acc: 0.3282 | Val Loss: 2.5025
Acc: 0.3000
H=800 Epoch 19/20 | Train Loss: 2.3628 Acc: 0.3406 | Val Loss: 2.4956
Acc: 0.3111
H=800 Epoch 20/20 | Train Loss: 2.3319 Acc: 0.3536 | Val Loss: 2.4789
Acc: 0.3222

Visualizing hidden weights for 300 hidden units model:
(128, 784)
(28, 28)
```




```
Visualizing hidden weights for 800 hidden units model:
(128, 784)
(28, 28)
```



Part 2.4: Finetuning AlexNet

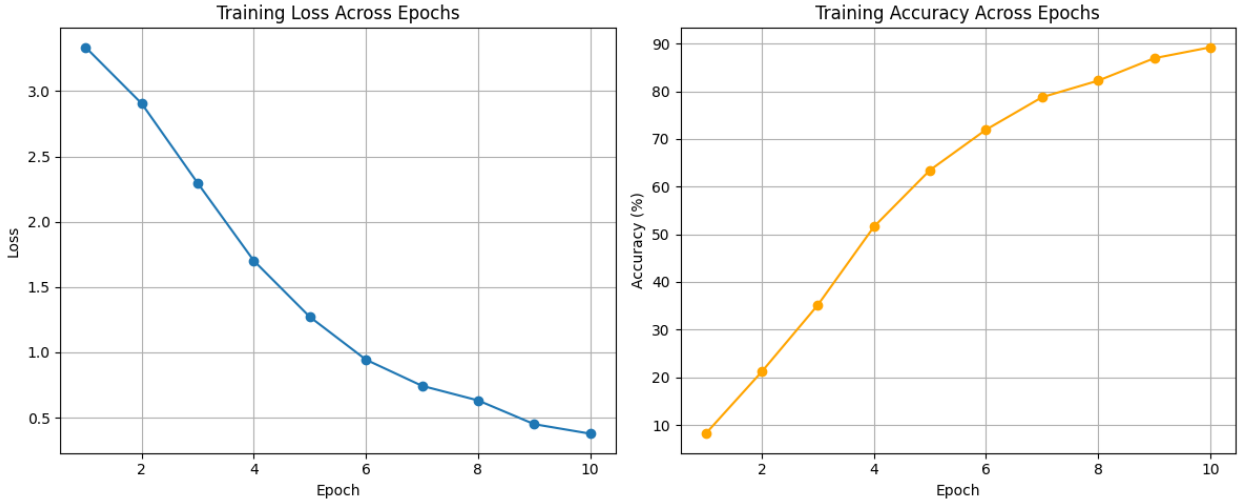
In Part 2.2, you used the (intermediate) outputs of AlexNet as inputs to a different network. You could view this as adding more layers to AlexNet (starting from e.g. conv4). Modify the classification layer of the AlexNet model so that you have the same output dimensions as in Part 2.2. In your report, include an example of using the network (i.e., plugging in an input face image into a placeholder, and getting the label as an output.)

```
X_train, X_val, X_test, y_train, y_val, y_test = split_data((128,
128))

class TransferFaceDataset(Dataset):
    def __init__(self, images, labels):
        self.images = images
        self.labels = labels

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

    def __getitem__(self, idx):
        img = self.images[idx]
        img = np.expand_dims(img, axis=0)  # (1,28,28)
        img = np.repeat(img, 3, axis=0)    # (3,28,28)
        return torch.tensor(img, dtype=torch.float32),
torch.tensor(self.labels[idx], dtype=torch.long)
```

```
batch_size = 64
train_dataset = TransferFaceDataset(X_train, y_train)
val_dataset = TransferFaceDataset(X_val, y_val)
test_dataset = TransferFaceDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

# Load pre-trained AlexNet
alexnet = models.alexnet(pretrained=True)

# Modify the classifier for our dataset (adjust num_hidden_units and
num_classes as needed)
num_hidden_units = 300  # or 800, as required
num_classes = len(set(y_train.tolist()))

alexnet.classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, num_hidden_units),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(num_hidden_units, num_classes)
)

# Optionally, enable requires_grad on all layers for full fine-tuning
for param in alexnet.parameters():
    param.requires_grad = True

# Move model to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
alexnet = alexnet.to(device)

# Define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(alexnet.parameters(), lr=1e-4, weight_decay=1e-
4)  # weight_decay for L2 regularization

# Example training loop for fine-tuning
num_epochs = 10  # Set the number of epochs as desired

train_losses = []
train_accuracies = []

for epoch in range(num_epochs):
    alexnet.train()
    running_loss = 0.0
    correct = 0
```

```
    total = 0

    for i, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()

        outputs = alexnet(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    epoch_loss = running_loss / total
    epoch_acc = 100. * correct / total
    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_acc)
    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss:
{epoch_loss:.4f} | Train Acc: {epoch_acc:.2f}%")

# After training, plot the training losses and accuracies
epochs = range(1, num_epochs+1)
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(epochs, train_losses, marker='o')
plt.title("Training Loss Across Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(epochs, train_accuracies, marker='o', color='orange')
plt.title("Training Accuracy Across Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.grid(True)

plt.tight_layout()
plt.show()


1615 90 90
1615 90 90

/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-packages/
torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the
```

```
future, please use 'weights' instead.
  warnings.warn(
/Users/alirezanoroozi/Library/Python/3.9/lib/python/site-packages/
torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may
be removed in the future. The current behavior is equivalent to
passing `weights=AlexNet_Weights.IMAGENET1K_V1`. You can also use
`weights=AlexNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)

Epoch 1/10 | Train Loss: 3.3350 | Train Acc: 8.17%
Epoch 2/10 | Train Loss: 2.9047 | Train Acc: 21.18%
Epoch 3/10 | Train Loss: 2.2950 | Train Acc: 35.17%
Epoch 4/10 | Train Loss: 1.7013 | Train Acc: 51.58%
Epoch 5/10 | Train Loss: 1.2711 | Train Acc: 63.47%
Epoch 6/10 | Train Loss: 0.9440 | Train Acc: 71.95%
Epoch 7/10 | Train Loss: 0.7439 | Train Acc: 78.76%
Epoch 8/10 | Train Loss: 0.6321 | Train Acc: 82.23%
Epoch 9/10 | Train Loss: 0.4505 | Train Acc: 86.93%
Epoch 10/10 | Train Loss: 0.3775 | Train Acc: 89.23%
```



Part 2.5: Bonus: Gradient Visualization

Here, you will use Utku Ozbulak's PyTorch CNN Visualizations Library to visualize the important parts of the input image for a particular output class. In particular, just select a specific picture of an actor, and then using your trained network in Part 2.4, perform Gradient visualization with guided backpropagation to understand the prediction for that actor with respect to the input image. Comment on your results.

What to Turn In

You have two **options** for submission: 1) Provide all the relevant answers to questions, images, figures, etc, in this Jupyter notebook, convert the jupyter notebook into a PDF, and upload the PDF. 2) Write all the answers to the questions and any relevant figures in a LaTeX report, convert the report to a PDF, and upload a zip file containing both the jupyter notebook and the report.

Grading

The assignment will be graded out of `100` points: `0` (no submission), `20` (an attempt at a solution), `40` (a partially correct solution), `60` (a mostly correct solution), `80` (a correct solution), `100` (a correct solution with lots of detail and analysis). The grading depends on both the content and clarity of your report.