EECS 504 PS5: Scene Recognition

Please provide the following information (e.g. Andrew Owens, ahowens):

[Your first name] [Your last name], [Your UMich uniqname]

Important: after you download the .ipynb file, please name it as **<your_uniquename>_<your_umid>.ip**
Example: adam_01101100.ipynb.

```
from tqdm import tqdm_notebook

for i in tqdm_notebook(range(10)):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Starting

Run the following code to import the modules you'll need. After your finish the assignment, remembe
your local machine as a .ipynb file for Canvas submission.

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
import os
import copy
from tqdm import tqdm

import torch
import torchvision
from torchvision import datasets, models, transforms
import torch.nn as nn
import torch.optim as optim

print("PyTorch Version: ",torch.__version__)
print("Torchvision Version: ",torchvision.__version__)
# Detect if we have a GPU available
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
    print("Using the GPU!")
else:
    print("WARNING: Could not find GPU! Using CPU only. If you want to enable GPU, please to

data_dir = "./data_miniplaces_modified"
```

```
PyTorch Version:  1.4.0
Torchvision Version:  0.5.0
Using the GPU!
```

Problem 5.1 Scene Recognition with VGG

You will build and train a convolutional neural network for scene recognition, i.e., classifying images i

1. Contruct dataloaders for train/val/test datasets
2. Build MiniVGG and MiniVGG-BN (MiniVGG with batch-normalization layers)
3. Train MiniVGG and MiniVGG-BN, compare their training progresses and their final top-1 and top-5 accuracies.
4. (Optional) Increase the size of the network by adding more layers and check whether top-1 and top-5 accurac

Step 0: Downloading the dataset.

```
# Download the miniplaces dataset
# Note: Restarting the runtime won't remove the downloaded dataset. You only need to re-downl
!wget http://www.eecs.umich.edu/courses/eecs504/data_miniplaces_modified.zip
```

```
--2020-02-11 01:17:21--  http://www.eecs.umich.edu/courses/eecs504/data_miniplaces_modif
Resolving www.eecs.umich.edu (www.eecs.umich.edu)... 141.212.113.199
Connecting to www.eecs.umich.edu (www.eecs.umich.edu)|141.212.113.199|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 534628730 (510M) [application/zip]
Saving to: ‘data_miniplaces_modified.zip’

data_miniplaces_mod 100%[===================>] 509.86M  11.2MB/s    in 46s

2020-02-11 01:18:08 (11.0 MB/s) - ‘data_miniplaces_modified.zip’ saved [534628730/53462
```

```
# Unzip the download dataset .zip file to your local colab dir
# Warning: this upzipping process may take a while. Please be patient.
!unzip -q data_miniplaces_modified.zip
```

Step 1: Build dataloaders for train, val, and test

```
def get_dataloaders(input_size, batch_size, shuffle = True):
```

```
    '''
    Build dataloaders with transformations.

    Args:
        input_size: int, the size of the tranformed images
        batch_size: int, minibatch size for dataloading

    Returns:
        dataloader_dict: dict, dict with "train", "val", "test" keys, each is mapped to a pyt

    '''

    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]

    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#
    # ============== YOUR CODE HERE ============== #

    # ========= Step 1: build transformations for the dataset ===========
    # You need to construct build a data transformation that does three preprocessings in orc
    # I. Resize the image to input_size using transforms.Resize
    # II. Convert the image to PyTorch tensor using transforms.ToTensor
    # III. Normalize the images with the provided mean and std parameters using transforms.Nc
    # You can use transforms.Compose to combine the above three transformations.

    composed_transform = transforms.Compose([transforms.Resize(input_size),
                                             transforms.ToTensor(),
                                             transforms.Normalize(mean, std)])

    data_transforms = {
        'train': composed_transform,
        'val': composed_transform,
        'test': composed_transform
    }

    # ========= Step 2: build dataloaders for the downloaded data ===========
    # I. use torch.datasets.ImageFolder with the provided data_dir and the data transfomatior
    # II. use torch.utils.data.DataLoader to build dataloaders with the constructed pytorch c
    # III. put the dataloaders into a dictionary

    # Create train/val/test datasets
    image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x])

    # Create training train/val/test dataloaders
    # Never shuffle the val and test datasets
    dataloaders_dict = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=batch_si

    # ============== END OF CODE ================= #
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#
```

```
    return dataloaders_dict


batch_size = 16
input_size = 128
dataloaders_dict = get_dataloaders(input_size, batch_size)

# Confirm your train/val/test sets contain 90,000/10,000/10,000 samples
print('# of training samples {}'.format(len(dataloaders_dict['train'].dataset)))
print('# of validation samples {}'.format(len(dataloaders_dict['val'].dataset)))
print('# of test samples {}'.format(len(dataloaders_dict['test'].dataset)))
```

```
# of training samples 90000
# of validation samples 10000
# of test samples 10000
```

```
# Visualize the data within the dataset
import json
with open('./data_miniplaces_modified/category_names.json', 'r') as f:
    class_names = json.load(f)['i2c']
class_names = {i:name for i, name in enumerate(class_names)}

def imshow(inp, title=None, ax=None, figsize=(5, 5)):
  """Imshow for Tensor."""
  inp = inp.numpy().transpose((1, 2, 0))
  mean = np.array([0.485, 0.456, 0.406])
  std = np.array([0.229, 0.224, 0.225])
  inp = std * inp + mean
  inp = np.clip(inp, 0, 1)
  if ax is None:
    fig, ax = plt.subplots(1, figsize=figsize)
  ax.imshow(inp)
  ax.set_xticks([])
  ax.set_yticks([])
  if title is not None:
    ax.set_title(title)

# Get a batch of training data
inputs, classes = next(iter(dataloaders_dict['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs, nrow=4)

fig, ax = plt.subplots(1, figsize=(10, 10))
title = [class_names[x.item()] if (i+1) % 4 != 0 else class_names[x.item()]+'\n' for i, x in
imshow(out, title=' | '.join(title), ax=ax)
```

clothing_store | yard | butchers_shop | martial_arts_gym
| baseball_field | swamp | phone_booth | butchers_shop
| martial_arts_gym | kitchen | temple | boat_deck
| ski_slope | airport_terminal | driveway | lobby

Step 2. Build MiniVGG and MiniVGG-BN

Please follow the instructions to build the two neural networks with architectures shown below.

MiniVGG architecture

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(5, 5))
  (classifier): Sequential(
    (0): Linear(in_features=3200, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.3, inplace=False)
    (6): Linear(in_features=256, out_features=100, bias=True)
  )
)
Number of trainable parameters 2166756
```

MiniVGG-BN architecure

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
    (10): ReLU(inplace=True)
    (11): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (13): ReLU(inplace=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(5, 5))
  (classifier): Sequential(
    (0): Linear(in_features=3200, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.3, inplace=False)
    (6): Linear(in_features=256, out_features=100, bias=True)
  )
)
Number of trainable parameters 2167652
```

```
# Helper function for counting number of trainable parameters.
def count_params(model):
    '''
    Counts the number of trainable parameters in PyTorch.

    Args:
        model: PyTorch model.

    Returns:
        num_params: int, number of trainable parameters.
    '''

    num_params = sum([item.numel() for item in model.parameters() if item.requires_grad])

    return num_params


# Network configurations for all layers before the final fully-connected layers.
# "M" corresponds to maxpooling layer, integers correspond to number of output channels of a
cfgs = {
    'MiniVGG': [64, 'M', 128, 'M', 128, 128, 'M'],
    'MiniVGG-BN': [64, 'M', 128, 'M', 128, 128, 'M']
```

```
}

def make_layers(cfg, batch_norm=False):

    '''
    Return a nn.Sequential object containing all layers before the fully-connected layers in

    Args:
      cfg: list
      batch_norm: bool, default: False. If set to True, a BatchNorm layer should be added aft

    Return:
      features: torch.nn.Sequential. Containers for all feature extraction layers. For use of
    '''
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#
    # ============= YOUR CODE HERE ============= #
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v

    features = nn.Sequential(*layers)
    # ============= END OF CODE ================ #
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#

    return features

class VGG(nn.Module):

    def __init__(self, features, num_classes=100, init_weights=True):
        super(VGG, self).__init__()

        self.features = features
        self.avgpool = nn.AdaptiveAvgPool2d((5, 5))

        # Construct the final FC layers using nn.Sequential.
        #++++++++++++++++++++++++++++++++++++++++++++++#
        #++++++++++++++++++++++++++++++++++++++++++++++#
        # ============= YOUR CODE HERE ============= #
        self.classifier = nn.Sequential(
            nn.Linear(128 * 5 * 5, 512),
            nn.ReLU(True),
```

```
            nn.Dropout(p=0.3),
            nn.Linear(512, 256),
            nn.ReLU(True),
            nn.Dropout(p=0.3),
            nn.Linear(256, num_classes),
        )
        # ============== END OF CODE ================= #
        #++++++++++++++++++++++++++++++++++++++++++++++++#
        #++++++++++++++++++++++++++++++++++++++++++++++++#

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)

features = make_layers(cfgs['MiniVGG'], batch_norm=False)
vgg = VGG(features)

features = make_layers(cfgs['MiniVGG-BN'], batch_norm=True)
vgg_bn = VGG(features)

# Print the network architectrue. Please compare the printed architecture with the one given
# Make sure your network has the same architecture as the one we give above.
print(vgg)
print('Number of trainable parameters {}'.format(count_params(vgg)))

print(vgg_bn)
print('Number of trainable parameters {}'.format(count_params(vgg_bn)))
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(5, 5))
  (classifier): Sequential(
    (0): Linear(in_features=3200, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.3, inplace=False)
    (6): Linear(in_features=256, out_features=100, bias=True)
  )
)
Number of trainable parameters 2166756
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (10): ReLU(inplace=True)
    (11): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
    (13): ReLU(inplace=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(5, 5))
  (classifier): Sequential(
    (0): Linear(in_features=3200, out_features=512, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=512, out_features=256, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.3, inplace=False)
    (6): Linear(in_features=256, out_features=100, bias=True)
  )
)
Number of trainable parameters 2167652
```

Step 3: Build training/validation loops

You will write a function for training and validating the network.

```
def make_optimizer(model):
    '''
    Args:
        model: NN to train

    Returns:
        optimizer: pytorch optmizer for updating the given model parameters.
    '''
    # Create an Adam optimizer with a learning rate 1e-3
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#
    # ============== YOUR CODE HERE ============== #
    params_to_update = model.parameters()
    optimizer = optim.Adam(params_to_update, lr=1e-3)
    # ============== END OF CODE ================= #
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#

    return optimizer

def get_loss():
    '''
    Returns:
        criterion: pytorch loss.
    '''

    # Create an instance of the cross entropy loss function
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#
    # ============== YOUR CODE HERE ============== #
    # The code should be a one-liner.
    criterion = nn.CrossEntropyLoss()
    # ============== END OF CODE ================= #
    #++++++++++++++++++++++++++++++++++++++++++++++#
    #++++++++++++++++++++++++++++++++++++++++++++++#

    return criterion


def train_model(model, dataloaders, criterion, optimizer, save_dir = None, num_epochs=25, moc
    '''
    Args:
        model: The NN to train
        dataloaders: A dictionary containing at least the keys
                    'train','val' that maps to Pytorch data loaders for the dataset
        criterion: The Loss function
```

```
        optimizer: Pytroch optimizer. The algorithm to update weights
        num_epochs: How many epochs to train for
        save_dir: Where to save the best model weights that are found. Using None will not wr

    Returns:
        model: The trained NN
        tr_acc_history: list, training accuracy history. Recording freq: one epoch.
        val_acc_history: list, validation accuracy history. Recording freq: one epoch.
    '''

    val_acc_history = []
    tr_acc_history = []

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            # loss and number of correct prediction for the current batch
            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            # TQDM has nice progress bars
            for inputs, labels in tqdm(dataloaders[phase]):

                #++++++++++++++++++++++++++++++++++++++++++++++#
                #++++++++++++++++++++++++++++++++++++++++++++++#
                # ============= YOUR CODE HERE ============= #
                # For "train" phase, compute the outputs, calculate the loss, update the mode
                # For "val" phase, compute the outputs, calculate the loss

                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    # Get model outputs and calculate loss
                    outputs = model(inputs)
```

```
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                # torch.max outputs the maximum value, and its index
                # Since the input is batched, we take the max along axis 1
                # (the meaningful outputs)
                _, preds = torch.max(outputs, 1)

                # backprop + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # ============== END OF CODE ================= #
            #++++++++++++++++++++++++++++++++++++++++++++++#
            #++++++++++++++++++++++++++++++++++++++++++++++#

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(dataloaders[phase].dataset)
        epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)

        print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))

        # deep copy the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

            # save the best model weights
            # ================================ IMPORTANT ===============================
            # Lossing connection to colab will lead to loss of trained weights.
            # You should download the trained weights to your local machine.
            # Later, you can load these weights directly without needing to train the neu
            # ==========================================================================
            if save_dir:
                torch.save(best_model_wts, os.path.join(save_dir, model_name + '.pth'))

        # record the train/val accuracies
        if phase == 'val':
            val_acc_history.append(epoch_acc)
        else:
            tr_acc_history.append(epoch_acc)

print('Best val Acc: {:4f}'.format(best_acc))

return model, tr_acc_history, val_acc_history
```

Step 4. Train MiniVGG and MiniVGG-BN

```
# Number of classes in the dataset
# Miniplaces has 100
num_classes = 100

# Batch size for training
batch_size = 128

# Shuffle the input data?
shuffle_datasets = True

# Number of epochs to train for
# During debugging, you can set this parameter to 1
# num_epochs = 1
# Training for 20 epochs. This will take about half an hour.
num_epochs = 20

### IO
# Directory to save weights to
save_dir = "weights"
os.makedirs(save_dir, exist_ok=True)

# get dataloders and criterion function
input_size = 64
dataloaders = get_dataloaders(input_size, batch_size, shuffle_datasets)
criterion = get_loss()


# Initialize MiniVGG
features = make_layers(cfgs['MiniVGG'], batch_norm=False)
model = VGG(features).to(device)
optimizer = make_optimizer(model)

# Train the model!
vgg, tr_his, val_his = train_model(model=model, dataloaders=dataloaders, criterion=criterion,
           save_dir=save_dir, num_epochs=num_epochs, model_name='MiniVGG')
```

```
  0%|          | 0/704 [00:00<?, ?it/s]Epoch 0/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.08it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 4.1074 Acc: 0.0599
100%|██████████| 79/79 [00:07<00:00, 10.90it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.7121 Acc: 0.1079
Epoch 1/19
----------
100%|██████████| 704/704 [01:10<00:00, 10.04it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.5748 Acc: 0.1381
100%|██████████| 79/79 [00:07<00:00, 10.96it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.3214 Acc: 0.1812
Epoch 2/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.06it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.3099 Acc: 0.1824
100%|██████████| 79/79 [00:07<00:00, 10.86it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.1342 Acc: 0.2143
Epoch 3/19
----------
100%|██████████| 704/704 [01:10<00:00, 10.05it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.1424 Acc: 0.2154
100%|██████████| 79/79 [00:07<00:00, 10.71it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.0483 Acc: 0.2344
Epoch 4/19
----------
100%|██████████| 704/704 [01:10<00:00,  9.95it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.0193 Acc: 0.2365
100%|██████████| 79/79 [00:07<00:00, 10.64it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.9567 Acc: 0.2492
Epoch 5/19
----------
100%|██████████| 704/704 [01:10<00:00,  9.97it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.9160 Acc: 0.2574
100%|██████████| 79/79 [00:07<00:00, 10.54it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8954 Acc: 0.2700
Epoch 6/19
----------
100%|██████████| 704/704 [01:10<00:00, 10.04it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.8104 Acc: 0.2774
100%|██████████| 79/79 [00:07<00:00, 10.34it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8824 Acc: 0.2724
Epoch 7/19
----------
100%|██████████| 704/704 [01:10<00:00,  9.98it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.7100 Acc: 0.2975
100%|██████████| 79/79 [00:07<00:00, 10.38it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8633 Acc: 0.2757
Epoch 8/19
----------
100%|██████████| 704/704 [01:11<00:00,  9.84it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.6300 Acc: 0.3147
100%|██████████| 79/79 [00:07<00:00, 12.27it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8548 Acc: 0.2820
Epoch 9/19
----------
100%|██████████| 704/704 [01:11<00:00,  9.91it/s]
```

```
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.5424 Acc: 0.3318
100%|██████████| 79/79 [00:07<00:00, 10.46it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8385 Acc: 0.2881
Epoch 10/19
----------
100%|██████████| 704/704 [01:11<00:00,  9.90it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.4690 Acc: 0.3460
100%|██████████| 79/79 [00:07<00:00, 10.69it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8236 Acc: 0.2893
Epoch 11/19
----------
100%|██████████| 704/704 [01:11<00:00,  9.91it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.3920 Acc: 0.3593
100%|██████████| 79/79 [00:07<00:00, 10.59it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8636 Acc: 0.2836
Epoch 12/19
----------
100%|██████████| 704/704 [01:10<00:00,  9.92it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.3120 Acc: 0.3773
100%|██████████| 79/79 [00:07<00:00, 10.92it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8894 Acc: 0.2878
Epoch 13/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.11it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.2434 Acc: 0.3891
100%|██████████| 79/79 [00:07<00:00, 11.20it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8991 Acc: 0.2898
Epoch 14/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.24it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.1762 Acc: 0.4030
100%|██████████| 79/79 [00:07<00:00, 11.23it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.9109 Acc: 0.2824
Epoch 15/19
----------
100%|██████████| 704/704 [01:07<00:00, 10.47it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.1155 Acc: 0.4155
100%|██████████| 79/79 [00:06<00:00, 11.58it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.9259 Acc: 0.2880
Epoch 16/19
----------
100%|██████████| 704/704 [01:06<00:00, 10.54it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.0527 Acc: 0.4305
100%|██████████| 79/79 [00:06<00:00, 11.61it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.9814 Acc: 0.2812
Epoch 17/19
----------
100%|██████████| 704/704 [01:06<00:00, 10.52it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 1.9872 Acc: 0.4430
100%|██████████| 79/79 [00:07<00:00, 10.98it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.0527 Acc: 0.2794
Epoch 18/19
----------
100%|██████████| 704/704 [01:07<00:00, 10.48it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 1.9365 Acc: 0.4554
100%|██████████| 79/79 [00:07<00:00, 11.13it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.0683 Acc: 0.2803
Epoch 19/19
```

```
----------
100%|██████████| 704/704 [01:07<00:00, 12.31it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 1.8920 Acc: 0.4643
100%|██████████| 79/79 [00:07<00:00, 11.25it/s]val Loss: 3.0798 Acc: 0.2773
Best val Acc: 0.289800
```

```
# Initialize MiniVGG-BN
features = make_layers(cfgs['MiniVGG-BN'], batch_norm=True)
model = VGG(features).to(device)
optimizer = make_optimizer(model)

# Train the model!
vgg_BN, tr_his_BN, val_his_BN = train_model(model=model, dataloaders=dataloaders, criterion=c
           save_dir=save_dir, num_epochs=num_epochs, model_name='MiniVGG-BN')
```

```
  0%|          | 0/704 [00:00<?, ?it/s]Epoch 0/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.09it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.9286 Acc: 0.0825
100%|██████████| 79/79 [00:07<00:00, 11.20it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.5533 Acc: 0.1350
Epoch 1/19
----------
100%|██████████| 704/704 [01:10<00:00, 10.04it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.5050 Acc: 0.1473
100%|██████████| 79/79 [00:06<00:00, 11.34it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.3160 Acc: 0.1826
Epoch 2/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.24it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.3273 Acc: 0.1795
100%|██████████| 79/79 [00:06<00:00, 11.32it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.1932 Acc: 0.2102
Epoch 3/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.28it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.2110 Acc: 0.2030
100%|██████████| 79/79 [00:07<00:00, 11.10it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.1652 Acc: 0.2058
Epoch 4/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.23it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.1048 Acc: 0.2215
100%|██████████| 79/79 [00:07<00:00, 11.17it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 3.0186 Acc: 0.2402
Epoch 5/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.19it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 3.0192 Acc: 0.2369
100%|██████████| 79/79 [00:06<00:00, 11.35it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.9868 Acc: 0.2478
Epoch 6/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.18it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.9417 Acc: 0.2518
100%|██████████| 79/79 [00:07<00:00, 11.19it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8364 Acc: 0.2788
Epoch 7/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.21it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.8726 Acc: 0.2667
100%|██████████| 79/79 [00:07<00:00, 11.02it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8409 Acc: 0.2745
Epoch 8/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.33it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.8027 Acc: 0.2789
100%|██████████| 79/79 [00:07<00:00, 10.91it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.7870 Acc: 0.2848
Epoch 9/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.32it/s]
```

```
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.7370 Acc: 0.2922
100%|██████████| 79/79 [00:06<00:00, 11.61it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6990 Acc: 0.3010
Epoch 10/19
----------
100%|██████████| 704/704 [01:08<00:00, 11.72it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.6785 Acc: 0.3028
100%|██████████| 79/79 [00:07<00:00, 11.08it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.8056 Acc: 0.2946
Epoch 11/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.22it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.6240 Acc: 0.3143
100%|██████████| 79/79 [00:06<00:00, 11.56it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6403 Acc: 0.3253
Epoch 12/19
----------
100%|██████████| 704/704 [01:08<00:00, 10.21it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.5666 Acc: 0.3271
100%|██████████| 79/79 [00:07<00:00, 11.27it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6886 Acc: 0.3142
Epoch 13/19
----------
100%|██████████| 704/704 [01:07<00:00, 10.35it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.5180 Acc: 0.3387
100%|██████████| 79/79 [00:07<00:00, 11.06it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.5947 Acc: 0.3317
Epoch 14/19
----------
100%|██████████| 704/704 [01:09<00:00, 10.15it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.4592 Acc: 0.3476
100%|██████████| 79/79 [00:07<00:00, 13.24it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6042 Acc: 0.3324
Epoch 15/19
----------
100%|██████████| 704/704 [01:10<00:00,  9.98it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.4132 Acc: 0.3558
100%|██████████| 79/79 [00:07<00:00, 11.06it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6191 Acc: 0.3306
Epoch 16/19
----------
100%|██████████| 704/704 [01:13<00:00,  9.55it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.3616 Acc: 0.3681
100%|██████████| 79/79 [00:07<00:00, 10.50it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6098 Acc: 0.3347
Epoch 17/19
----------
100%|██████████| 704/704 [01:11<00:00,  9.83it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.3178 Acc: 0.3779
100%|██████████| 79/79 [00:07<00:00, 10.91it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.5815 Acc: 0.3403
Epoch 18/19
----------
100%|██████████| 704/704 [01:11<00:00,  9.86it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.2736 Acc: 0.3845
100%|██████████| 79/79 [00:07<00:00, 10.91it/s]
  0%|          | 0/704 [00:00<?, ?it/s]val Loss: 2.6354 Acc: 0.3332
Epoch 19/19
```

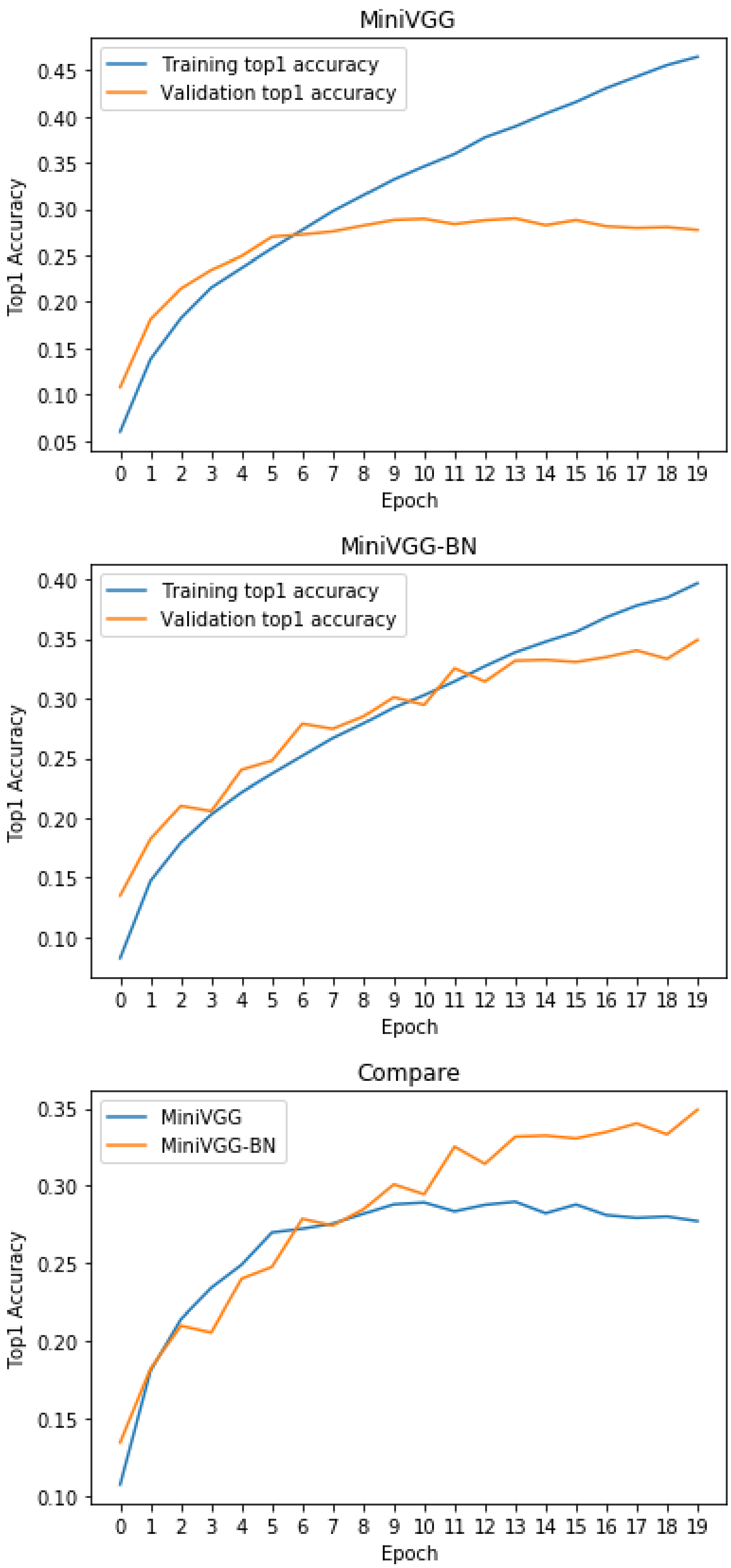
```
----------
100%|██████████| 704/704 [01:11<00:00,  9.84it/s]
  0%|          | 0/79 [00:00<?, ?it/s]train Loss: 2.2228 Acc: 0.3967
100%|██████████| 79/79 [00:07<00:00, 12.85it/s]val Loss: 2.5488 Acc: 0.3491
Best val Acc: 0.349100
```

```python
x = np.arange(num_epochs)

# train/val accuracies for MiniVGG
plt.figure()
plt.plot(x, tr_his)
plt.plot(x, val_his)
plt.legend(['Training top1 accuracy', 'Validation top1 accuracy'])
plt.xticks(x)
plt.xlabel('Epoch')
plt.ylabel('Top1 Accuracy')
plt.title('MiniVGG')
plt.show()

# train/val accuracies for MiniVGG-BN
plt.plot(x, tr_his_BN)
plt.plot(x, val_his_BN)
plt.legend(['Training top1 accuracy', 'Validation top1 accuracy'])
plt.xticks(x)
plt.xlabel('Epoch')
plt.ylabel('Top1 Accuracy')
plt.title('MiniVGG-BN')
plt.show()

# compare val accuracies of MiniVGG and MiniVGG-BN
plt.plot(x, val_his)
plt.plot(x, val_his_BN)
plt.legend(['MiniVGG', 'MiniVGG-BN'])
plt.xticks(x)
plt.xlabel('Epoch')
plt.ylabel('Top1 Accuracy')
plt.title('Compare')
plt.show()
```



Summarize the effect of batch normalization:

Please write a few sentences here to summarize the effect of batch nomalization.

```
pickle.dump(tr_his, open('tr_his.pkl', 'wb'))
pickle.dump(tr_his_BN, open('tr_his_BN.pkl', 'wb'))
pickle.dump(val_his, open('val_his.pkl', 'wb'))
pickle.dump(val_his_BN, open('val_his_BN.pkl', 'wb'))
```

Step 5. Measure top1 and top5 accuracies of MiniVGG and MiniVGG-B

Definition of top-k accuracy: if the correct label is within the *top k* predicted classes according to the prediction by the neural network as a correct prediction.

```
def accuracy(output, target, topk=(1,)):
    '''
    Computes the accuracy over the k top predictions for the specified values of k.

    Args:
        output: pytorch tensor, (batch_size x num_classes). Outputs of the network for one ba
        target: pytorch tensor, (batch_size,). True labels for one batch.

    Returns:
        res: list. Accuracies corresponding to topk[0], topk[1], ...
    '''
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)

        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))

        res = []
        for k in topk:
            correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res

def test(model, dataloader):

    model.eval()

    top1_acc = []
    top5_acc = []

    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
```

```
            res = accuracy(outputs, labels, topk=(1, 5))

            top1_acc.append(res[0] * len(outputs))
            top5_acc.append(res[1] * len(outputs))

    print('Top-1 accuracy {}%, Top-5 accuracy {}%'.format(sum(top1_acc).item()/10000, sum(top


##### Download pretrained weights (TODO: remove for student version) #####

!wget http://www.eecs.umich.edu/courses/eecs504/MiniVGG-BN.pth
!wget http://www.eecs.umich.edu/courses/eecs504/MiniVGG.pth

features = make_layers(cfgs['MiniVGG-BN'], batch_norm=True)
vgg_BN = VGG(features).to(device)

features = make_layers(cfgs['MiniVGG'], batch_norm=False)
vgg = VGG(features).to(device)

vgg_BN.load_state_dict(torch.load('MiniVGG-BN.pth'))
vgg.load_state_dict(torch.load('MiniVGG.pth'))

test(vgg_BN, dataloaders['test'])
test(vgg, dataloaders['test'])
```

```
Top-1 accuracy 34.96%, Top-5 accuracy 64.94%
Top-1 accuracy 29.21%, Top-5 accuracy 58.62%
```

```
##### To pass the test, both networks should have Top-5 accuracy above 50% #####
vgg_BN.load_state_dict(torch.load('./weights/MiniVGG-BN.pth'))
vgg.load_state_dict(torch.load('./weights/MiniVGG.pth'))

test(vgg_BN, dataloaders['test'])
test(vgg, dataloaders['test'])
```