```
!nvidia-smi
```

```
Thu Mar  9 01:15:59 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   31C    P0    51W / 400W |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
! pip install ptflops
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ptflops
  Downloading ptflops-0.6.9.tar.gz (12 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in /usr/local/lib/python3.9/dist-packages (from ptflops) (1.13.1+cu116)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.9/dist-packages (from torch->ptflops) (4.5.
Building wheels for collected packages: ptflops
  Building wheel for ptflops (setup.py) ... done
  Created wheel for ptflops: filename=ptflops-0.6.9-py3-none-any.whl size=11712 sha256=7e162d83a7afcbbecddec5c4536fe3b
  Stored in directory: /root/.cache/pip/wheels/86/07/9f/879035d99d7b639bbc564d23fed862a679aee7d1a2dced8c2e
Successfully built ptflops
Installing collected packages: ptflops
Successfully installed ptflops-0.6.9
```

```
import torch
import torchvision
from torch import nn
from torchvision import transforms
import torch.optim as optim
from torchsummary import summary
from ptflops import get_model_complexity_info
import matplotlib.pyplot as plt
import numpy as np
from torch.nn import functional as F
```

```
# Define how we want images transformed
resize = (64, 64)
trans = transforms.Compose([transforms.Resize(resize),
                            transforms.ToTensor()])

# Create training and validation sets
training_set = torchvision.datasets.CIFAR10('./data', train=True,
                                             transform=trans, download=True)
validation_set = torchvision.datasets.CIFAR10('./data', train=False,
                                               transform=trans, download=True)

# Create dataloaders for each set
training_loader = torch.utils.data.DataLoader(training_set, batch_size=128,
                                              shuffle=True, num_workers=2)
validation_loader = torch.utils.data.DataLoader(validation_set, batch_size=128,
                                                shuffle=False, num_workers=2)
```

```
print("Training set size:", len(training_set))
print("Validation set size:", len(validation_set))
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%                                   170498071/170498071 [00:05<00:00, 42066728.18it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Training set size: 50000
```

```
# Define the training loop for each epoch
def trainLoop(dataloader, model, loss_fn, optimizer):

    numBatches = len(dataloader)
    dataSize = len(dataloader.dataset)
    totalLoss = 0
    numCorrect = 0

    for batch, (X, y) in enumerate(dataloader):

        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        totalLoss = totalLoss + loss.item()

        if batch % 100 == 0:
            loss = loss.item()
            interLosses.append(loss)
            avgLoss = totalLoss / (batch + 1)
            avgLosses.append(avgLoss)
            print("loss:", loss)

        pred = model(X)
        numCorrect = numCorrect + (pred.argmax(1) == y).type(torch.float).sum().item()

    trainAcc = numCorrect / dataSize
    trainHist.append(trainAcc)
    trainAccPercent = trainAcc * 100

    epochLoss = totalLoss / len(dataloader)
    trainLosses.append(epochLoss)

    print("Training Accuracy: ", trainAccPercent,  "    Training Loss: ", epochLoss)


# Define the validation loop for each epoch
def valLoop(dataloader, model, loss_fn):

    numBatches = len(dataloader)
    dataSize = len(dataloader.dataset)
    valLoss = 0
    numCorrect = 0

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            valLoss = valLoss + loss_fn(pred, y).item()
            numCorrect = numCorrect + (pred.argmax(1) == y).type(torch.float).sum().item()


    valAcc = numCorrect / dataSize
    valHist.append(valAcc)
    valAccPercent = valAcc * 100

    avgLoss = valLoss / numBatches
    valLosses.append(avgLoss)

    print("Validation Accuracy:", valAccPercent, "    Validation Loss: ", avgLoss)
    print(" ")
```

1a) Baseline VGG-11

```
def VGGBlock(numConv, outChannels):
  layers = []
  for _ in range(numConv):
     layers.append(nn.LazyConv2d(outChannels, kernel_size=3, padding=1))
     layers.append(nn.LazyBatchNorm2d())
     layers.append(nn.ReLU())

  layers.append(nn.MaxPool2d(kernel_size=2, stride=2))
  return nn.Sequential(*layers)



def BaseVGG(arch):
  convBlocks = []
  for (numConv, outChannels) in arch:
    convBlocks.append(VGGBlock(numConv, outChannels))

  VGGNet = nn.Sequential(
            *convBlocks,
            nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
            nn.LazyLinear(512), nn.ReLU(), nn.Dropout(0.5),       # Adjusted from 4096 to better accomodate only 10 classes
            nn.LazyLinear(10)
        )
  return VGGNet


VGGEleven = BaseVGG(arch=((1, 64), (1, 128), (2, 256), (2, 512), (2, 512)))
#summary(VGGEleven, input_size = (3, 64, 64), batch_size = 128)
```

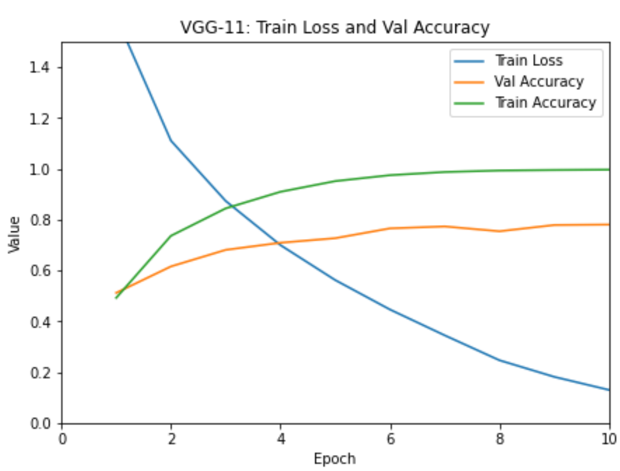
```
/usr/local/lib/python3.8/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
  warnings.warn('Lazy modules are a new feature under heavy development '
```

```
#from torchvision import models
#model = models.vgg11()
#summary(model, input_size = (3, 64, 64), batch_size = 128)


criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(VGGEleven.parameters(), lr=0.01)


# Begin training over 10 epochs
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, VGGEleven, criterion, optimizer)
    valLoop(validation_loader, VGGEleven, criterion)
```

```
Epoch 1
loss: 2.322709798812866
loss: 1.7706522941589355
loss: 1.3530608415603638
loss: 1.3835805654525757
Training Accuracy:  49.206     Training Loss:  1.5973757987132158
Validation Accuracy: 51.190000000000005     Validation Loss:  1.3320431256596046

Epoch 2
loss: 1.2362195253372192
loss: 1.1561284065246582
loss: 1.2826095819473267
loss: 1.0891155004501343
Training Accuracy:  73.636     Training Loss:  1.1106922954244687
Validation Accuracy: 61.58     Validation Loss:  1.058244418494309

Epoch 3
loss: 0.9438884854316711
```

```
loss: 1.0084092617034912
loss: 0.854917049407959
loss: 0.83794105052948
Training Accuracy:  84.428     Training Loss:  0.8733290932367525
Validation Accuracy: 68.10000000000001     Validation Loss:  0.9052490395835683

Epoch 4
loss: 0.8517128825187683
loss: 0.8558422327041626
loss: 0.5831639170646667
loss: 0.8300999402999878
Training Accuracy:  90.956     Training Loss:  0.6997149263501472
Validation Accuracy: 70.89999999999999     Validation Loss:  0.8479296225535718

Epoch 5
loss: 0.6483731269836426
loss: 0.5351250767707825
loss: 0.7217350602149963
loss: 0.401886522769928
Training Accuracy:  95.15599999999999     Training Loss:  0.562122440856436
Validation Accuracy: 72.69     Validation Loss:  0.8275535581987116

Epoch 6
loss: 0.5183454155921936
loss: 0.3113739490509033
loss: 0.4868917465209961
loss: 0.3523615300655365
Training Accuracy:  97.49     Training Loss:  0.446201415470494
Validation Accuracy: 76.55999999999999     Validation Loss:  0.6968071970004069

Epoch 7
loss: 0.3252418637275696
loss: 0.3342866897583008
loss: 0.3068305253982544
loss: 0.2835935652256012
Training Accuracy:  98.74000000000001     Training Loss:  0.3446421988327485
Validation Accuracy: 77.32     Validation Loss:  0.7261034318163425

Epoch 8
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("VGG-11: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: ~100ish mins for 10 epochs w/A100 GPU
```




```
macs, params = get_model_complexity_info(VGGEleven, (3, 64, 64), as_strings=True,
```

```
                                             print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module Flatten is treated as a zero-op.
Warning: module Dropout is treated as a zero-op.
Sequential(
  19.72 M, 100.000% Params, 624.48 MMac, 100.000% MACs,
  (0): Sequential(
    1.92 k, 0.010% Params, 8.39 MMac, 1.343% MACs,
    (0): Conv2d(1.79 k, 0.009% Params, 7.34 MMac, 1.175% MACs, 3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1
    (1): BatchNorm2d(128, 0.001% Params, 524.29 KMac, 0.084% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 262.14 KMac, 0.042% MACs, )
    (3): MaxPool2d(0, 0.000% Params, 262.14 KMac, 0.042% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (1): Sequential(
    74.11 k, 0.376% Params, 76.15 MMac, 12.195% MACs,
    (0): Conv2d(73.86 k, 0.374% Params, 75.63 MMac, 12.111% MACs, 64, 128, kernel_size=(3, 3), stride=(1, 1), padding=
    (1): BatchNorm2d(256, 0.001% Params, 262.14 KMac, 0.042% MACs, 128, eps=1e-05, momentum=0.1, affine=True, track_ru
    (2): ReLU(0, 0.000% Params, 131.07 KMac, 0.021% MACs, )
    (3): MaxPool2d(0, 0.000% Params, 131.07 KMac, 0.021% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (2): Sequential(
    886.27 k, 4.494% Params, 227.08 MMac, 36.363% MACs,
    (0): Conv2d(295.17 k, 1.497% Params, 75.56 MMac, 12.100% MACs, 128, 256, kernel_size=(3, 3), stride=(1, 1), paddin
    (1): BatchNorm2d(512, 0.003% Params, 131.07 KMac, 0.021% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.010% MACs, )
    (3): Conv2d(590.08 k, 2.992% Params, 151.06 MMac, 24.190% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
    (4): BatchNorm2d(512, 0.003% Params, 131.07 KMac, 0.021% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (5): ReLU(0, 0.000% Params, 65.54 KMac, 0.010% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.010% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
  )
  (3): Sequential(
    3.54 M, 17.960% Params, 226.79 MMac, 36.316% MACs,
    (0): Conv2d(1.18 M, 5.984% Params, 75.53 MMac, 12.095% MACs, 256, 512, kernel_size=(3, 3), stride=(1, 1), padding=
    (1): BatchNorm2d(1.02 k, 0.005% Params, 65.54 KMac, 0.010% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
    (2): ReLU(0, 0.000% Params, 32.77 KMac, 0.005% MACs, )
    (3): Conv2d(2.36 M, 11.966% Params, 151.03 MMac, 24.184% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), paddin
    (4): BatchNorm2d(1.02 k, 0.005% Params, 65.54 KMac, 0.010% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
    (5): ReLU(0, 0.000% Params, 32.77 KMac, 0.005% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 32.77 KMac, 0.005% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
  )
  (4): Sequential(
    4.72 M, 23.942% Params, 75.57 MMac, 12.101% MACs,
    (0): Conv2d(2.36 M, 11.966% Params, 37.76 MMac, 6.046% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
    (1): BatchNorm2d(1.02 k, 0.005% Params, 16.38 KMac, 0.003% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
    (2): ReLU(0, 0.000% Params, 8.19 KMac, 0.001% MACs, )
    (3): Conv2d(2.36 M, 11.966% Params, 37.76 MMac, 6.046% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
    (4): BatchNorm2d(1.02 k, 0.005% Params, 16.38 KMac, 0.003% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
    (5): ReLU(0, 0.000% Params, 8.19 KMac, 0.001% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 8.19 KMac, 0.001% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
  )
  (5): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
  (6): Linear(8.39 M, 42.556% Params, 8.39 MMac, 1.344% MACs, in_features=2048, out_features=4096, bias=True)
  (7): ReLU(0, 0.000% Params, 4.1 KMac, 0.001% MACs, )
  (8): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
  (9): Linear(2.1 M, 10.636% Params, 2.1 MMac, 0.336% MACs, in_features=4096, out_features=512, bias=True)
  (10): ReLU(0, 0.000% Params, 512.0 Mac, 0.000% MACs, )
  (11): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
  (12): Linear(5.13 k, 0.026% Params, 5.13 KMac, 0.001% MACs, in_features=512, out_features=10, bias=True)
)
Computational complexity:  624.48 MMac
```

1b) VGG-16

```
VGGSixteen = BaseVGG(arch=((2, 64), (2, 128), (3, 256), (3, 512), (3, 512)))
#summary(VGGSixteen, input_size = (3, 64, 64), batch_size = 128)
```

```
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
  warnings.warn('Lazy modules are a new feature under heavy development '
```

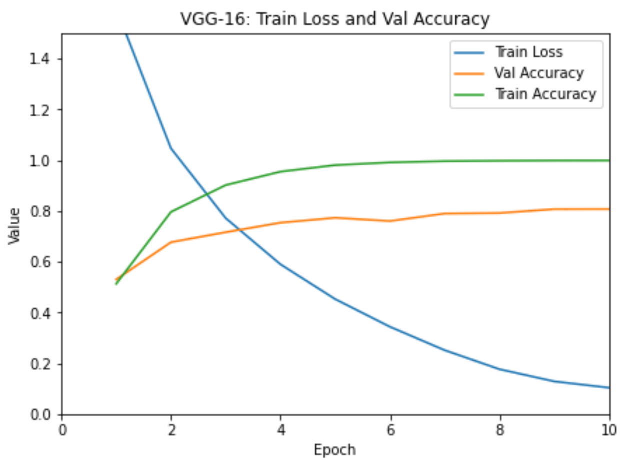
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(VGGSixteen.parameters(), lr=0.01)
```

```
# Begin training over 10 epochs
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, VGGSixteen, criterion, optimizer)
    valLoop(validation_loader, VGGSixteen, criterion)
```

```
Epoch 1
loss: 2.3159642219543457
loss: 1.7693490982055664
loss: 1.4039890766143799
loss: 1.376631498336792
Training Accuracy:  51.286     Training Loss:  1.5964810982384645
Validation Accuracy: 53.02     Validation Loss:  1.3110757688932781

Epoch 2
loss: 1.289994716644287
loss: 1.1914608478546143
loss: 1.2127760648727417
loss: 0.9775689840316772
Training Accuracy:  79.61399999999999     Training Loss:  1.0464261959275931
Validation Accuracy: 67.67     Validation Loss:  0.9252960327305372

Epoch 3
loss: 0.8221198320388794
loss: 0.9339970946311951
loss: 0.5893039703369141
loss: 0.7952086925506592
Training Accuracy:  90.17399999999999     Training Loss:  0.771403071368137
Validation Accuracy: 71.61     Validation Loss:  0.8031428788281694

Epoch 4
loss: 0.5304207801818848
loss: 0.3293275237083435
loss: 0.5665963888168335
loss: 0.5413843989372253
Training Accuracy:  95.468     Training Loss:  0.5896911217885858
Validation Accuracy: 75.36     Validation Loss:  0.7284090673621697

Epoch 5
loss: 0.5566954612731934
loss: 0.35142502188682556
loss: 0.46066609025001526
loss: 0.4674033224582672
Training Accuracy:  98.044     Training Loss:  0.4524637028536833
Validation Accuracy: 77.29     Validation Loss:  0.6810789225222189

Epoch 6
loss: 0.3067588806152344
loss: 0.32145750522613525
loss: 0.3884941637516022
loss: 0.4495769739151001
Training Accuracy:  99.104     Training Loss:  0.3434785594949332
Validation Accuracy: 76.01     Validation Loss:  0.7775940551787992

Epoch 7
loss: 0.3063439428806305
loss: 0.224542498588562
loss: 0.21085478365421295
loss: 0.30594658851623535
Training Accuracy:  99.61     Training Loss:  0.2512144137678854
Validation Accuracy: 78.97     Validation Loss:  0.689447547816023

Epoch 8
loss: 0.2563817501068115
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("VGG-16: Train Loss and Val Accuracy")
```

```
plt.plot(x, trainLosses, label="Train Loss")
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: ~3hrs 15mins for 10 epochs
```




```
macs, params = get_model_complexity_info(VGGSixteen, (3, 64, 64), as_strings=True,
                                         print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module Flatten is treated as a zero-op.
Warning: module Dropout is treated as a zero-op.
Sequential(
  25.22 M, 100.000% Params, 1.27 GMac, 100.000% MACs,
  (0): Sequential(
    38.98 k, 0.155% Params, 160.43 MMac, 12.650% MACs,
    (0): Conv2d(1.79 k, 0.007% Params, 7.34 MMac, 0.579% MACs, 3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1
    (1): BatchNorm2d(128, 0.001% Params, 524.29 KMac, 0.041% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 262.14 KMac, 0.021% MACs, )
    (3): Conv2d(36.93 k, 0.146% Params, 151.26 MMac, 11.927% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
    (4): BatchNorm2d(128, 0.001% Params, 524.29 KMac, 0.041% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (5): ReLU(0, 0.000% Params, 262.14 KMac, 0.021% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 262.14 KMac, 0.021% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (1): Sequential(
    221.95 k, 0.880% Params, 227.67 MMac, 17.952% MACs,
    (0): Conv2d(73.86 k, 0.293% Params, 75.63 MMac, 5.963% MACs, 64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
    (1): BatchNorm2d(256, 0.001% Params, 262.14 KMac, 0.021% MACs, 128, eps=1e-05, momentum=0.1, affine=True, track_ru
    (2): ReLU(0, 0.000% Params, 131.07 KMac, 0.010% MACs, )
    (3): Conv2d(147.58 k, 0.585% Params, 151.13 MMac, 11.916% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), paddi
    (4): BatchNorm2d(256, 0.001% Params, 262.14 KMac, 0.021% MACs, 128, eps=1e-05, momentum=0.1, affine=True, track_ru
    (5): ReLU(0, 0.000% Params, 131.07 KMac, 0.010% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 131.07 KMac, 0.010% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (2): Sequential(
    1.48 M, 5.856% Params, 378.34 MMac, 29.833% MACs,
    (0): Conv2d(295.17 k, 1.170% Params, 75.56 MMac, 5.958% MACs, 128, 256, kernel_size=(3, 3), stride=(1, 1), padding
    (1): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.010% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.005% MACs, )
    (3): Conv2d(590.08 k, 2.340% Params, 151.06 MMac, 11.911% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
    (4): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.010% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (5): ReLU(0, 0.000% Params, 65.54 KMac, 0.005% MACs, )
    (6): Conv2d(590.08 k, 2.340% Params, 151.06 MMac, 11.911% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
    (7): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.010% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (8): ReLU(0, 0.000% Params, 65.54 KMac, 0.005% MACs, )
    (9): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.005% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
  )
  (3): Sequential(
    5.9 M, 23.407% Params, 377.91 MMac, 29.799% MACs,
    (0): Conv2d(1.18 M, 4.680% Params, 75.53 MMac, 5.956% MACs, 256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
    (1): BatchNorm2d(1.02 k, 0.004% Params, 65.54 KMac, 0.005% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
```

```
      (2): ReLU(0, 0.000% Params, 32.77 KMac, 0.003% MACs, )
      (3): Conv2d(2.36 M, 9.357% Params, 151.03 MMac, 11.909% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding
      (4): BatchNorm2d(1.02 k, 0.004% Params, 65.54 KMac, 0.005% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
      (5): ReLU(0, 0.000% Params, 32.77 KMac, 0.003% MACs, )
      (6): Conv2d(2.36 M, 9.357% Params, 151.03 MMac, 11.909% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding
      (7): BatchNorm2d(1.02 k, 0.004% Params, 65.54 KMac, 0.005% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
      (8): ReLU(0, 0.000% Params, 32.77 KMac, 0.003% MACs, )
      (9): MaxPool2d(0, 0.000% Params, 32.77 KMac, 0.003% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
    )
    (4): Sequential(
      7.08 M, 28.084% Params, 113.35 MMac, 8.938% MACs,
      (0): Conv2d(2.36 M, 9.357% Params, 37.76 MMac, 2.977% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
      (1): BatchNorm2d(1.02 k, 0.004% Params, 16.38 KMac, 0.001% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
      (2): ReLU(0, 0.000% Params, 8.19 KMac, 0.001% MACs, )
      (3): Conv2d(2.36 M, 9.357% Params, 37.76 MMac, 2.977% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
      (4): BatchNorm2d(1.02 k, 0.004% Params, 16.38 KMac, 0.001% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
```

1c) VGG-19

```
VGGNineteen = BaseVGG(arch=((2, 64), (2, 128), (4, 256), (4, 512), (4, 512)))
#summary(VGGNineteen, input_size = (3, 64, 64), batch_size = 128)
```

```
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
  warnings.warn('Lazy modules are a new feature under heavy development '
```

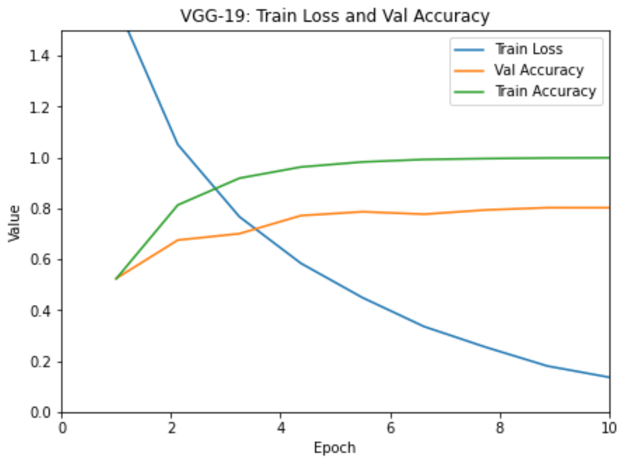
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(VGGNineteen.parameters(), lr=0.01)
```

```
# Begin training over 10 epochs
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, VGGNineteen, criterion, optimizer)
    valLoop(validation_loader, VGGNineteen, criterion)
```

```
Epoch 1
loss: 2.331387758255005
loss: 1.7032594680786133
loss: 1.526144027709961
loss: 1.3765671253204346
Training Accuracy:  52.334     Training Loss:  1.6046475733027739
Validation Accuracy: 52.459999999999994     Validation Loss:  1.2993437293209606

Epoch 2
loss: 1.4717752933502197
loss: 1.3382675647735596
loss: 1.0125163793563843
loss: 0.863140881061554
Training Accuracy:  81.314     Training Loss:  1.050951664709984
Validation Accuracy: 67.53     Validation Loss:  0.925742282143122

Epoch 3
loss: 0.9831904768943787
loss: 0.7725498080253601
loss: 0.6669642329216003
loss: 0.8635208606719971
Training Accuracy:  91.85600000000001     Training Loss:  0.76711285967961
Validation Accuracy: 70.05     Validation Loss:  0.8731882028941866

Epoch 4
loss: 0.7225092053413391
loss: 0.5570181608200073
loss: 0.5194321274757385
loss: 0.580623984336853
Training Accuracy:  96.26400000000001     Training Loss:  0.5837884024738351
Validation Accuracy: 77.17     Validation Loss:  0.6869940467273132

Epoch 5
loss: 0.4886320233345032
loss: 0.514672040939331
loss: 0.5179904699325562
loss: 0.35854268074035645
Training Accuracy:  98.25     Training Loss:  0.4488447916019908
Validation Accuracy: 78.66     Validation Loss:  0.659045045511632

Epoch 6
loss: 0.35212236642837524
loss: 0.19586272537708282
loss: 0.259711354970932
loss: 0.34487879276275635
Training Accuracy:  99.21799999999999     Training Loss:  0.33521658315530517
Validation Accuracy: 77.71000000000001     Validation Loss:  0.7025533322292038

Epoch 7
loss: 0.26394909620285034
loss: 0.2867668867111206
loss: 0.32079559564590454
loss: 0.3416512906551361
Training Accuracy:  99.58200000000001     Training Loss:  0.2542842682784476
Validation Accuracy: 79.36999999999999     Validation Loss:  0.6789268756969066

Epoch 8
loss: 0.2204064577817917
loss: 0.19845499098300934
loss: 0.1552555412054062
loss: 0.13485829532146454
Training Accuracy:  99.798     Training Loss:  0.18023010173721996
Validation Accuracy: 80.25999999999999     Validation Loss:  0.6903623685806612

Epoch 9
loss: 0.09689150005578995
loss: 0.11360768973827362
loss: 0.09631191194057465
loss: 0.2283611297607422
```

```
# Plot results
x = np.linspace(1, epochs, epochs-1)
plt.figure(figsize=(7,5))
plt.title("VGG-19: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
```

```
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: ~3hrs 30mins for 9 epochs
```




```
macs, params = get_model_complexity_info(VGGNineteen, (3, 64, 64), as_strings=True,
                                          print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module Flatten is treated as a zero-op.
Warning: module Dropout is treated as a zero-op.
Sequential(
  30.53 M, 100.000% Params, 1.61 GMac, 100.000% MACs,
  (0): Sequential(
    38.98 k, 0.128% Params, 160.43 MMac, 9.975% MACs,
    (0): Conv2d(1.79 k, 0.006% Params, 7.34 MMac, 0.456% MACs, 3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1
    (1): BatchNorm2d(128, 0.000% Params, 524.29 KMac, 0.033% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 262.14 KMac, 0.016% MACs, )
    (3): Conv2d(36.93 k, 0.121% Params, 151.26 MMac, 9.404% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(
    (4): BatchNorm2d(128, 0.000% Params, 524.29 KMac, 0.033% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (5): ReLU(0, 0.000% Params, 262.14 KMac, 0.016% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 262.14 KMac, 0.016% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (1): Sequential(
    221.95 k, 0.727% Params, 227.67 MMac, 14.155% MACs,
    (0): Conv2d(73.86 k, 0.242% Params, 75.63 MMac, 4.702% MACs, 64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
    (1): BatchNorm2d(256, 0.001% Params, 262.14 KMac, 0.016% MACs, 128, eps=1e-05, momentum=0.1, affine=True, track_ru
    (2): ReLU(0, 0.000% Params, 131.07 KMac, 0.008% MACs, )
    (3): Conv2d(147.58 k, 0.483% Params, 151.13 MMac, 9.396% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), paddin
    (4): BatchNorm2d(256, 0.001% Params, 262.14 KMac, 0.016% MACs, 128, eps=1e-05, momentum=0.1, affine=True, track_ru
    (5): ReLU(0, 0.000% Params, 131.07 KMac, 0.008% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 131.07 KMac, 0.008% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (2): Sequential(
    2.07 M, 6.772% Params, 529.6 MMac, 32.927% MACs,
    (0): Conv2d(295.17 k, 0.967% Params, 75.56 MMac, 4.698% MACs, 128, 256, kernel_size=(3, 3), stride=(1, 1), padding
    (1): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.008% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.004% MACs, )
    (3): Conv2d(590.08 k, 1.933% Params, 151.06 MMac, 9.392% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
    (4): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.008% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (5): ReLU(0, 0.000% Params, 65.54 KMac, 0.004% MACs, )
    (6): Conv2d(590.08 k, 1.933% Params, 151.06 MMac, 9.392% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
    (7): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.008% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_ru
    (8): ReLU(0, 0.000% Params, 65.54 KMac, 0.004% MACs, )
    (9): Conv2d(590.08 k, 1.933% Params, 151.06 MMac, 9.392% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
    (10): BatchNorm2d(512, 0.002% Params, 131.07 KMac, 0.008% MACs, 256, eps=1e-05, momentum=0.1, affine=True, track_r
    (11): ReLU(0, 0.000% Params, 65.54 KMac, 0.004% MACs, )
    (12): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.004% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
  )
  (3): Sequential(
    8.26 M, 27.067% Params, 529.04 MMac, 32.893% MACs,
    (0): Conv2d(1.18 M, 3.865% Params, 75.53 MMac, 4.696% MACs, 256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
    (1): BatchNorm2d(1.02 k, 0.003% Params, 65.54 KMac, 0.004% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
    (2): ReLU(0, 0.000% Params, 32.77 KMac, 0.002% MACs, )
    (3): Conv2d(2.36 M, 7.729% Params, 151.03 MMac, 9.390% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
    (4): BatchNorm2d(1.02 k, 0.003% Params, 65.54 KMac, 0.004% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
```

```
      (5): ReLU(0, 0.000% Params, 32.77 KMac, 0.002% MACs, )
      (6): Conv2d(2.36 M, 7.729% Params, 151.03 MMac, 9.390% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
      (7): BatchNorm2d(1.02 k, 0.003% Params, 65.54 KMac, 0.004% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track_
      (8): ReLU(0, 0.000% Params, 32.77 KMac, 0.002% MACs, )
      (9): Conv2d(2.36 M, 7.729% Params, 151.03 MMac, 9.390% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
      (10): BatchNorm2d(1.02 k, 0.003% Params, 65.54 KMac, 0.004% MACs, 512, eps=1e-05, momentum=0.1, affine=True, track
      (11): ReLU(0, 0.000% Params, 32.77 KMac, 0.002% MACs, )
      (12): MaxPool2d(0, 0.000% Params, 32.77 KMac, 0.002% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
    )
    (4): Sequential(
      9.44 M, 30.930% Params, 151.13 MMac, 9.397% MACs,
```

2a) Baseline GoogleNet

```
class Inception(nn.Module):

    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)
        # Branch 1
        self.b1_1 = nn.LazyConv2d(c1, kernel_size=1)
        # Branch 2
        self.b2_1 = nn.LazyConv2d(c2[0], kernel_size=1)
        self.b2_2 = nn.LazyConv2d(c2[1], kernel_size=3, padding=1)
        # Branch 3
        self.b3_1 = nn.LazyConv2d(c3[0], kernel_size=1)
        self.b3_2 = nn.LazyConv2d(c3[1], kernel_size=5, padding=2)
        # Branch 4
        self.b4_1 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
        self.b4_2 = nn.LazyConv2d(c4, kernel_size=1)

    def forward(self, x):
        b1 = F.relu(self.b1_1(x))
        b2 = F.relu(self.b2_2(F.relu(self.b2_1(x))))
        b3 = F.relu(self.b3_2(F.relu(self.b3_1(x))))
        b4 = F.relu(self.b4_2(self.b4_1(x)))
        return torch.cat((b1, b2, b3, b4), dim=1)
```

```
bigBlock1 = nn.Sequential(
                   nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3), nn.ReLU(),
                   nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
/usr/local/lib/python3.8/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
  warnings.warn('Lazy modules are a new feature under heavy development '
```

```
bigBlock2 = nn.Sequential(
                   nn.LazyConv2d(64, kernel_size=1), nn.ReLU(),
                   nn.LazyConv2d(192, kernel_size=3, padding=1), nn.ReLU(),
                   nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock3 = nn.Sequential(Inception(64, (96, 128), (16, 32), 32),
                   Inception(128, (128, 192), (32, 96), 64),
                   nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock4 = nn.Sequential(Inception(192, (96, 208), (16, 48), 64),
                   Inception(160, (112, 224), (24, 64), 64),
                   Inception(128, (128, 256), (24, 64), 64),
                   Inception(112, (144, 288), (32, 64), 64),
                   Inception(256, (160, 320), (32, 128), 128),
                   nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock5 = nn.Sequential(Inception(256, (160, 320), (32, 128), 128),
                   Inception(384, (192, 384), (48, 128), 128),
                   nn.AdaptiveAvgPool2d((1,1)),
                   nn.Flatten())


GoogleNet = nn.Sequential(
    bigBlock1,
    bigBlock2,
```

```
    bigBlock3,
    bigBlock4,
    bigBlock5,
    nn.LazyLinear(10)
)


summary(GoogleNet, input_size = (3, 64, 64), batch_size = 128)
```

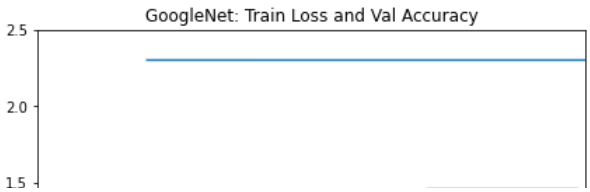
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [128, 64, 32, 32]           9,472
              ReLU-2          [128, 64, 32, 32]               0
         MaxPool2d-3          [128, 64, 16, 16]               0
            Conv2d-4          [128, 64, 16, 16]           4,160
              ReLU-5          [128, 64, 16, 16]               0
            Conv2d-6         [128, 192, 16, 16]         110,784
              ReLU-7         [128, 192, 16, 16]               0
         MaxPool2d-8           [128, 192, 8, 8]               0
            Conv2d-9            [128, 64, 8, 8]          12,352
           Conv2d-10            [128, 96, 8, 8]          18,528
           Conv2d-11           [128, 128, 8, 8]         110,720
           Conv2d-12            [128, 16, 8, 8]           3,088
           Conv2d-13            [128, 32, 8, 8]          12,832
        MaxPool2d-14           [128, 192, 8, 8]               0
           Conv2d-15            [128, 32, 8, 8]           6,176
        Inception-16           [128, 256, 8, 8]               0
           Conv2d-17           [128, 128, 8, 8]          32,896
           Conv2d-18           [128, 128, 8, 8]          32,896
           Conv2d-19           [128, 192, 8, 8]         221,376
           Conv2d-20            [128, 32, 8, 8]           8,224
           Conv2d-21            [128, 96, 8, 8]          76,896
        MaxPool2d-22           [128, 256, 8, 8]               0
           Conv2d-23            [128, 64, 8, 8]          16,448
        Inception-24           [128, 480, 8, 8]               0
        MaxPool2d-25           [128, 480, 4, 4]               0
           Conv2d-26           [128, 192, 4, 4]          92,352
           Conv2d-27            [128, 96, 4, 4]          46,176
           Conv2d-28           [128, 208, 4, 4]         179,920
           Conv2d-29            [128, 16, 4, 4]           7,696
           Conv2d-30            [128, 48, 4, 4]          19,248
        MaxPool2d-31           [128, 480, 4, 4]               0
           Conv2d-32            [128, 64, 4, 4]          30,784
        Inception-33           [128, 512, 4, 4]               0
           Conv2d-34           [128, 160, 4, 4]          82,080
           Conv2d-35           [128, 112, 4, 4]          57,456
           Conv2d-36           [128, 224, 4, 4]         226,016
           Conv2d-37            [128, 24, 4, 4]          12,312
           Conv2d-38            [128, 64, 4, 4]          38,464
        MaxPool2d-39           [128, 512, 4, 4]               0
           Conv2d-40            [128, 64, 4, 4]          32,832
        Inception-41           [128, 512, 4, 4]               0
           Conv2d-42           [128, 128, 4, 4]          65,664
           Conv2d-43           [128, 128, 4, 4]          65,664
           Conv2d-44           [128, 256, 4, 4]         295,168
           Conv2d-45            [128, 24, 4, 4]          12,312
           Conv2d-46            [128, 64, 4, 4]          38,464
        MaxPool2d-47           [128, 512, 4, 4]               0
           Conv2d-48            [128, 64, 4, 4]          32,832
        Inception-49           [128, 512, 4, 4]               0
           Conv2d-50           [128, 112, 4, 4]          57,456
           Conv2d-51           [128, 144, 4, 4]          73,872
           Conv2d-52           [128, 288, 4, 4]         373,536
           Conv2d-53            [128, 32, 4, 4]          16,416
           Conv2d-54            [128, 64, 4, 4]          51,264
        MaxPool2d-55           [128, 512, 4, 4]               0
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(GoogleNet.parameters(), lr=0.01)


# Begin training over 5 epochs
epochs = 5
valHist = []
valLosses = []
trainHist = []
trainLosses = []
```

```
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, GoogleNet, criterion, optimizer)
    valLoop(validation_loader, GoogleNet, criterion)
```

```
Epoch 1
loss: 2.304359197616577
loss: 2.302676200866699
loss: 2.302182674407959
loss: 2.302480936050415
Training Accuracy:  10.0     Training Loss:  2.302677970408174
Validation Accuracy: 10.0     Validation Loss:  2.302629262586183

Epoch 2
loss: 2.3016629219055176
loss: 2.3033292293548584
loss: 2.302647590637207
loss: 2.302816152572632
Training Accuracy:  10.07     Training Loss:  2.302648488213034
Validation Accuracy: 10.0     Validation Loss:  2.3026042165635507

Epoch 3
loss: 2.302588939666748
loss: 2.3023459911346436
loss: 2.3028881549835205
loss: 2.303091526031494
Training Accuracy:  10.25     Training Loss:  2.3026394051359134
Validation Accuracy:  10.0     Validation Loss:  2.302596753156638

Epoch 4
loss: 2.302618980407715
loss: 2.3027374744415283
loss: 2.3028383255004883
loss: 2.302997350692749
Training Accuracy:  10.209999999999999     Training Loss:  2.302632306847731
Validation Accuracy: 10.0     Validation Loss:  2.3025873974908757

Epoch 5
loss: 2.3026015758514404
loss: 2.3025219440460205
loss: 2.3024256229400635
loss: 2.302746295928955
Training Accuracy:  10.198     Training Loss:  2.3026275384761488
Validation Accuracy: 10.0     Validation Loss:  2.30258980582032
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("GoogleNet: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 2.5)
plt.show()

### Train time: ~18ish minutes for 5 epochs
```




```
macs, params = get_model_complexity_info(GoogleNet, (3, 64, 64), as_strings=True,
                                           print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
    Warning: module Inception is treated as a zero-op.
    Warning: module Flatten is treated as a zero-op.
    Sequential(
      5.98 M, 100.000% Params, 129.76 MMac, 100.000% MACs,
      (0): Sequential(
        9.47 k, 0.158% Params, 9.83 MMac, 7.576% MACs,
        (0): Conv2d(9.47 k, 0.158% Params, 9.7 MMac, 7.475% MACs, 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3)
        (1): ReLU(0, 0.000% Params, 65.54 KMac, 0.051% MACs, )
        (2): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.051% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
      )
      (1): Sequential(
        114.94 k, 1.921% Params, 29.54 MMac, 22.766% MACs,
        (0): Conv2d(4.16 k, 0.070% Params, 1.06 MMac, 0.821% MACs, 64, 64, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU(0, 0.000% Params, 16.38 KMac, 0.013% MACs, )
        (2): Conv2d(110.78 k, 1.851% Params, 28.36 MMac, 21.857% MACs, 64, 192, kernel_size=(3, 3), stride=(1, 1), padding
        (3): ReLU(0, 0.000% Params, 49.15 KMac, 0.038% MACs, )
        (4): MaxPool2d(0, 0.000% Params, 49.15 KMac, 0.038% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
      )
      (2): Sequential(
        552.43 k, 9.232% Params, 35.42 MMac, 27.293% MACs,
        (0): Inception(
          163.7 k, 2.736% Params, 10.49 MMac, 8.083% MACs,
          (b1_1): Conv2d(12.35 k, 0.206% Params, 790.53 KMac, 0.609% MACs, 192, 64, kernel_size=(1, 1), stride=(1, 1))
          (b2_1): Conv2d(18.53 k, 0.310% Params, 1.19 MMac, 0.914% MACs, 192, 96, kernel_size=(1, 1), stride=(1, 1))
          (b2_2): Conv2d(110.72 k, 1.850% Params, 7.09 MMac, 5.461% MACs, 96, 128, kernel_size=(3, 3), stride=(1, 1), padd
          (b3_1): Conv2d(3.09 k, 0.052% Params, 197.63 KMac, 0.152% MACs, 192, 16, kernel_size=(1, 1), stride=(1, 1))
          (b3_2): Conv2d(12.83 k, 0.214% Params, 821.25 KMac, 0.633% MACs, 16, 32, kernel_size=(5, 5), stride=(1, 1), padd
          (b4_1): MaxPool2d(0, 0.000% Params, 12.29 KMac, 0.009% MACs, kernel_size=3, stride=1, padding=1, dilation=1, cei
          (b4_2): Conv2d(6.18 k, 0.103% Params, 395.26 KMac, 0.305% MACs, 192, 32, kernel_size=(1, 1), stride=(1, 1))
        )
        (1): Inception(
          388.74 k, 6.496% Params, 24.9 MMac, 19.186% MACs,
          (b1_1): Conv2d(32.9 k, 0.550% Params, 2.11 MMac, 1.623% MACs, 256, 128, kernel_size=(1, 1), stride=(1, 1))
          (b2_1): Conv2d(32.9 k, 0.550% Params, 2.11 MMac, 1.623% MACs, 256, 128, kernel_size=(1, 1), stride=(1, 1))
          (b2_2): Conv2d(221.38 k, 3.700% Params, 14.17 MMac, 10.919% MACs, 128, 192, kernel_size=(3, 3), stride=(1, 1), p
          (b3_1): Conv2d(8.22 k, 0.137% Params, 526.34 KMac, 0.406% MACs, 256, 32, kernel_size=(1, 1), stride=(1, 1))
          (b3_2): Conv2d(76.9 k, 1.285% Params, 4.92 MMac, 3.793% MACs, 32, 96, kernel_size=(5, 5), stride=(1, 1), padding
          (b4_1): MaxPool2d(0, 0.000% Params, 16.38 KMac, 0.013% MACs, kernel_size=3, stride=1, padding=1, dilation=1, cei
          (b4_2): Conv2d(16.45 k, 0.275% Params, 1.05 MMac, 0.811% MACs, 256, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (2): MaxPool2d(0, 0.000% Params, 30.72 KMac, 0.024% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
      )
      (3): Sequential(
        2.81 M, 46.946% Params, 45.0 MMac, 34.681% MACs,
        (0): Inception(
          376.18 k, 6.287% Params, 6.03 MMac, 4.644% MACs,
          (b1_1): Conv2d(92.35 k, 1.543% Params, 1.48 MMac, 1.139% MACs, 480, 192, kernel_size=(1, 1), stride=(1, 1))
          (b2_1): Conv2d(46.18 k, 0.772% Params, 738.82 KMac, 0.569% MACs, 480, 96, kernel_size=(1, 1), stride=(1, 1))
          (b2_2): Conv2d(179.92 k, 3.007% Params, 2.88 MMac, 2.219% MACs, 96, 208, kernel_size=(3, 3), stride=(1, 1), padd
          (b3_1): Conv2d(7.7 k, 0.129% Params, 123.14 KMac, 0.095% MACs, 480, 16, kernel_size=(1, 1), stride=(1, 1))
          (b3_2): Conv2d(19.25 k, 0.322% Params, 307.97 KMac, 0.237% MACs, 16, 48, kernel_size=(5, 5), stride=(1, 1), padd
          (b4_1): MaxPool2d(0, 0.000% Params, 7.68 KMac, 0.006% MACs, kernel_size=3, stride=1, padding=1, dilation=1, ceil_
          (b4_2): Conv2d(30.78 k, 0.514% Params, 492.54 KMac, 0.380% MACs, 480, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (1): Inception(
          449.16 k, 7.506% Params, 7.19 MMac, 5.545% MACs,
          (b1_1): Conv2d(82.08 k, 1.372% Params, 1.31 MMac, 1.012% MACs, 512, 160, kernel_size=(1, 1), stride=(1, 1))
          (b2_1): Conv2d(57.46 k, 0.960% Params, 919.3 KMac, 0.708% MACs, 512, 112, kernel_size=(1, 1), stride=(1, 1))
```

2b) GoogleNet w/ Batchnorm

```
class BatchInception(nn.Module):
```

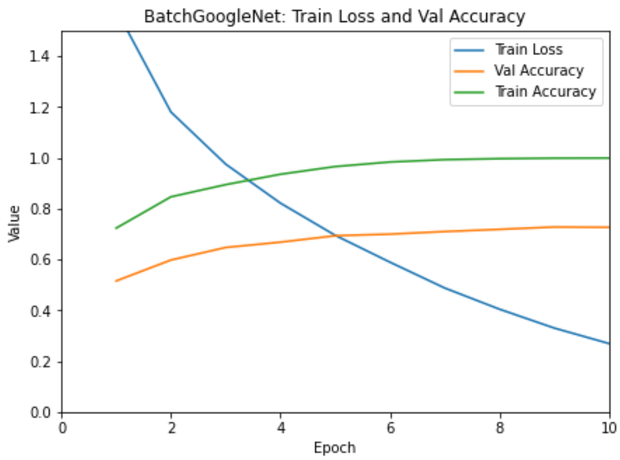
```python
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(BatchInception, self).__init__(**kwargs)
        # Branch 1
        self.b1_1 = nn.LazyConv2d(c1, kernel_size=1)
        self.b1_bn = nn.LazyBatchNorm2d()
        # Branch 2
        self.b2_1 = nn.LazyConv2d(c2[0], kernel_size=1)
        self.b2_1_bn = nn.LazyBatchNorm2d()
        self.b2_2 = nn.LazyConv2d(c2[1], kernel_size=3, padding=1)
        self.b2_2_bn = nn.LazyBatchNorm2d()
        # Branch 3
        self.b3_1 = nn.LazyConv2d(c3[0], kernel_size=1)
        self.b3_1_bn = nn.LazyBatchNorm2d()
        self.b3_2 = nn.LazyConv2d(c3[1], kernel_size=5, padding=2)
        self.b3_2_bn = nn.LazyBatchNorm2d()
        # Branch 4
        self.b4_1 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
        self.b4_2 = nn.LazyConv2d(c4, kernel_size=1)
        self.b4_bn = nn.LazyBatchNorm2d()

    def forward(self, x):
        b1 = self.b1_bn( F.relu(self.b1_1(x)) )
        b2 = F.relu(self.b2_2_bn( self.b2_2(F.relu(self.b2_1_bn( self.b2_1(x) ))) ))
        b3 = F.relu(self.b3_2_bn( self.b3_2(F.relu(self.b3_1_bn( self.b3_1(x) ))) ))
        b4 = F.relu(self.b4_bn( self.b4_2(self.b4_1(x)) ))
        return torch.cat((b1, b2, b3, b4), dim=1)


bigBlock1 = nn.Sequential(
                          nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3), nn.LazyBatchNorm2d(), nn.ReLU(),
                          nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock2 = nn.Sequential(
                          nn.LazyConv2d(64, kernel_size=1), nn.LazyBatchNorm2d(), nn.ReLU(),
                          nn.LazyConv2d(192, kernel_size=3, padding=1), nn.LazyBatchNorm2d(), nn.ReLU(),
                          nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock3 = nn.Sequential(BatchInception(64, (96, 128), (16, 32), 32),
                          BatchInception(128, (128, 192), (32, 96), 64),
                          nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock4 = nn.Sequential(BatchInception(192, (96, 208), (16, 48), 64),
                          BatchInception(160, (112, 224), (24, 64), 64),
                          BatchInception(128, (128, 256), (24, 64), 64),
                          BatchInception(112, (144, 288), (32, 64), 64),
                          BatchInception(256, (160, 320), (32, 128), 128),
                          nn.MaxPool2d(kernel_size=3, stride=2, padding=1))


bigBlock5 = nn.Sequential(BatchInception(256, (160, 320), (32, 128), 128),
                          BatchInception(384, (192, 384), (48, 128), 128),
                          nn.AdaptiveAvgPool2d((1,1)),
                          nn.Flatten())


BatchGoogleNet = nn.Sequential(
    bigBlock1,
    bigBlock2,
    bigBlock3,
    bigBlock4,
    bigBlock5,
    nn.LazyLinear(10)
)


summary(BatchGoogleNet, input_size = (3, 64, 64), batch_size = 128)


criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(BatchGoogleNet.parameters(), lr=0.01)


# Begin training over 10 epochs
```

```
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, BatchGoogleNet, criterion, optimizer)
    valLoop(validation_loader, BatchGoogleNet, criterion)
```

```
Epoch 1
loss: 2.4031636714935303
loss: 1.5241669416427612
loss: 1.5519462823867798
loss: 1.412006139755249
Training Accuracy:  72.332     Training Loss:  1.5833526383275571
Validation Accuracy: 51.55     Validation Loss:  1.3400464057922363

Epoch 2
loss: 1.2369741201400757
loss: 1.211909532546997
loss: 1.2887994050979614
loss: 1.0961852073669434
Training Accuracy:  84.646     Training Loss:  1.1801138774818167
Validation Accuracy: 59.78     Validation Loss:  1.1172009359432171

Epoch 3
loss: 0.9839455485343933
loss: 1.060718297958374
loss: 0.8656967282295227
loss: 0.9972230792045593
Training Accuracy:  89.446     Training Loss:  0.9750528414840893
Validation Accuracy: 64.71000000000001     Validation Loss:  0.9899978622605529

Epoch 4
loss: 0.8221462368965149
loss: 0.6781387329101562
loss: 0.63700270652771
loss: 0.7727214694023132
Training Accuracy:  93.53200000000001     Training Loss:  0.8216175238799561
Validation Accuracy: 66.81     Validation Loss:  0.9690882918200915

Epoch 5
loss: 0.8320279121398926
loss: 0.4910655915737152
loss: 0.6911700963973999
loss: 0.7061962485313416
Training Accuracy:  96.55799999999999     Training Loss:  0.6949193696384235
Validation Accuracy: 69.34     Validation Loss:  0.8760639089572279

Epoch 6
loss: 0.6218411922454834
loss: 0.6399492025375366
loss: 0.6702315211296082
loss: 0.5875455737113953
Training Accuracy:  98.35000000000001     Training Loss:  0.58926835648544
Validation Accuracy: 69.96     Validation Loss:  0.8721806414519684

Epoch 7
loss: 0.3940551280975342
loss: 0.32410064339637756
loss: 0.5813966393470764
loss: 0.5567288398742676
Training Accuracy:  99.278     Training Loss:  0.48764182044112164
Validation Accuracy: 70.98     Validation Loss:  0.9099403778208962

Epoch 8
loss: 0.4223775863647461
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("BatchGoogleNet: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
```

```
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: ~45ish minutes for 10 epochs
```




```
macs, params = get_model_complexity_info(BatchGoogleNet, (3, 64, 64), as_strings=True,
                                           print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module BatchInception is treated as a zero-op.
Warning: module Flatten is treated as a zero-op.
Sequential(
  6.0 M, 100.000% Params, 130.28 MMac, 100.000% MACs,
  (0): Sequential(
    9.6 k, 0.160% Params, 9.96 MMac, 7.646% MACs,
    (0): Conv2d(9.47 k, 0.158% Params, 9.7 MMac, 7.445% MACs, 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3)
    (1): BatchNorm2d(128, 0.002% Params, 131.07 KMac, 0.101% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.050% MACs, )
    (3): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.050% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
  )
  (1): Sequential(
    115.46 k, 1.925% Params, 29.67 MMac, 22.774% MACs,
    (0): Conv2d(4.16 k, 0.069% Params, 1.06 MMac, 0.817% MACs, 64, 64, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(128, 0.002% Params, 32.77 KMac, 0.025% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_runn
    (2): ReLU(0, 0.000% Params, 16.38 KMac, 0.013% MACs, )
    (3): Conv2d(110.78 k, 1.847% Params, 28.36 MMac, 21.768% MACs, 64, 192, kernel_size=(3, 3), stride=(1, 1), padding
    (4): BatchNorm2d(384, 0.006% Params, 98.3 KMac, 0.075% MACs, 192, eps=1e-05, momentum=0.1, affine=True, track_runn
    (5): ReLU(0, 0.000% Params, 49.15 KMac, 0.038% MACs, )
    (6): MaxPool2d(0, 0.000% Params, 49.15 KMac, 0.038% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
  )
  (2): Sequential(
    554.45 k, 9.243% Params, 35.54 MMac, 27.282% MACs,
    (0): BatchInception(
      164.43 k, 2.741% Params, 10.54 MMac, 8.087% MACs,
      (b1_1): Conv2d(12.35 k, 0.206% Params, 790.53 KMac, 0.607% MACs, 192, 64, kernel_size=(1, 1), stride=(1, 1))
      (b1_bn): BatchNorm2d(128, 0.002% Params, 8.19 KMac, 0.006% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_
      (b2_1): Conv2d(18.53 k, 0.309% Params, 1.19 MMac, 0.910% MACs, 192, 96, kernel_size=(1, 1), stride=(1, 1))
      (b2_1_bn): BatchNorm2d(192, 0.003% Params, 12.29 KMac, 0.009% MACs, 96, eps=1e-05, momentum=0.1, affine=True, tr
      (b2_2): Conv2d(110.72 k, 1.846% Params, 7.09 MMac, 5.439% MACs, 96, 128, kernel_size=(3, 3), stride=(1, 1), padd
      (b2_2_bn): BatchNorm2d(256, 0.004% Params, 16.38 KMac, 0.013% MACs, 128, eps=1e-05, momentum=0.1, affine=True, t
      (b3_1): Conv2d(3.09 k, 0.051% Params, 197.63 KMac, 0.152% MACs, 192, 16, kernel_size=(1, 1), stride=(1, 1))
      (b3_1_bn): BatchNorm2d(32, 0.001% Params, 2.05 KMac, 0.002% MACs, 16, eps=1e-05, momentum=0.1, affine=True, trac
      (b3_2): Conv2d(12.83 k, 0.214% Params, 821.25 KMac, 0.630% MACs, 16, 32, kernel_size=(5, 5), stride=(1, 1), padd
      (b3_2_bn): BatchNorm2d(64, 0.001% Params, 4.1 KMac, 0.003% MACs, 32, eps=1e-05, momentum=0.1, affine=True, track_
      (b4_1): MaxPool2d(0, 0.000% Params, 12.29 KMac, 0.009% MACs, kernel_size=3, stride=1, padding=1, dilation=1, cei
      (b4_2): Conv2d(6.18 k, 0.103% Params, 395.26 KMac, 0.303% MACs, 192, 32, kernel_size=(1, 1), stride=(1, 1))
      (b4_bn): BatchNorm2d(64, 0.001% Params, 4.1 KMac, 0.003% MACs, 32, eps=1e-05, momentum=0.1, affine=True, track_r
    )
    (1): BatchInception(
      390.02 k, 6.502% Params, 24.98 MMac, 19.171% MACs,
      (b1_1): Conv2d(32.9 k, 0.548% Params, 2.11 MMac, 1.616% MACs, 256, 128, kernel_size=(1, 1), stride=(1, 1))
```

```
    (b1_bn): BatchNorm2d(256, 0.004% Params, 16.38 KMac, 0.013% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
    (b2_1): Conv2d(32.9 k, 0.548% Params, 2.11 MMac, 1.616% MACs, 256, 128, kernel_size=(1, 1), stride=(1, 1))
    (b2_1_bn): BatchNorm2d(256, 0.004% Params, 16.38 KMac, 0.013% MACs, 128, eps=1e-05, momentum=0.1, affine=True, t
    (b2_2): Conv2d(221.38 k, 3.691% Params, 14.17 MMac, 10.875% MACs, 128, 192, kernel_size=(3, 3), stride=(1, 1), p
    (b2_2_bn): BatchNorm2d(384, 0.006% Params, 24.58 KMac, 0.019% MACs, 192, eps=1e-05, momentum=0.1, affine=True, t
    (b3_1): Conv2d(8.22 k, 0.137% Params, 526.34 KMac, 0.404% MACs, 256, 32, kernel_size=(1, 1), stride=(1, 1))
    (b3_1_bn): BatchNorm2d(64, 0.001% Params, 4.1 KMac, 0.003% MACs, 32, eps=1e-05, momentum=0.1, affine=True, track_
    (b3_2): Conv2d(76.9 k, 1.282% Params, 4.92 MMac, 3.777% MACs, 32, 96, kernel_size=(5, 5), stride=(1, 1), padding
    (b3_2_bn): BatchNorm2d(192, 0.003% Params, 12.29 KMac, 0.009% MACs, 96, eps=1e-05, momentum=0.1, affine=True, tr
    (b4_1): MaxPool2d(0, 0.000% Params, 16.38 KMac, 0.013% MACs, kernel_size=3, stride=1, padding=1, dilation=1, cei
    (b4_2): Conv2d(16.45 k, 0.274% Params, 1.05 MMac, 0.808% MACs, 256, 64, kernel_size=(1, 1), stride=(1, 1))
    (b4_bn): BatchNorm2d(128, 0.002% Params, 8.19 KMac, 0.006% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_
  )
  (2): MaxPool2d(0, 0.000% Params, 30.72 KMac, 0.024% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
)
```

3a) Baseline ResNet-18

```python
class Residual(nn.Module):

    def __init__(self, num_channels, use_1x1conv=False, strides=1):
        super().__init__()
        self.conv1 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1,
                                   stride=strides)
        self.conv2 = nn.LazyConv2d(num_channels, kernel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.LazyConv2d(num_channels, kernel_size=1,
                                       stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.LazyBatchNorm2d()
        self.bn2 = nn.LazyBatchNorm2d()

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return F.relu(Y)


def ResBlock(num_residuals, num_channels, first_block=False):
    blk = []
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.append(Residual(num_channels, use_1x1conv=True, strides=2))
        else:
            blk.append(Residual(num_channels))

    return nn.Sequential(*blk)


def body(arch):
    bodyBlocks = []
    for i, b in enumerate(arch):
        bodyBlocks.append(ResBlock(*b, first_block=(i==0)))

    return nn.Sequential(*bodyBlocks)


resBigBlock1 = nn.Sequential(
            nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
            nn.LazyBatchNorm2d(), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
 /usr/local/lib/python3.8/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
   warnings.warn('Lazy modules are a new feature under heavy development '
```

```python
resBigBlock2 = body(((2, 64), (2, 128), (2, 256), (2, 512)))


resBigBlock3 = nn.Sequential(
```

```
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten(),
            nn.LazyLinear(10))


ResNet18 = nn.Sequential(
    resBigBlock1,
    resBigBlock2,
    resBigBlock3
)


summary(ResNet18, input_size = (3, 64, 64), batch_size = 128)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [128, 64, 32, 32]           9,472
       BatchNorm2d-2         [128, 64, 32, 32]             128
              ReLU-3         [128, 64, 32, 32]               0
         MaxPool2d-4         [128, 64, 16, 16]               0
            Conv2d-5         [128, 64, 16, 16]          36,928
       BatchNorm2d-6         [128, 64, 16, 16]             128
            Conv2d-7         [128, 64, 16, 16]          36,928
       BatchNorm2d-8         [128, 64, 16, 16]             128
          Residual-9         [128, 64, 16, 16]               0
           Conv2d-10         [128, 64, 16, 16]          36,928
      BatchNorm2d-11         [128, 64, 16, 16]             128
           Conv2d-12         [128, 64, 16, 16]          36,928
      BatchNorm2d-13         [128, 64, 16, 16]             128
         Residual-14         [128, 64, 16, 16]               0
           Conv2d-15          [128, 128, 8, 8]          73,856
      BatchNorm2d-16          [128, 128, 8, 8]             256
           Conv2d-17          [128, 128, 8, 8]         147,584
      BatchNorm2d-18          [128, 128, 8, 8]             256
           Conv2d-19          [128, 128, 8, 8]           8,320
         Residual-20          [128, 128, 8, 8]               0
           Conv2d-21          [128, 128, 8, 8]         147,584
      BatchNorm2d-22          [128, 128, 8, 8]             256
           Conv2d-23          [128, 128, 8, 8]         147,584
      BatchNorm2d-24          [128, 128, 8, 8]             256
         Residual-25          [128, 128, 8, 8]               0
           Conv2d-26          [128, 256, 4, 4]         295,168
      BatchNorm2d-27          [128, 256, 4, 4]             512
           Conv2d-28          [128, 256, 4, 4]         590,080
      BatchNorm2d-29          [128, 256, 4, 4]             512
           Conv2d-30          [128, 256, 4, 4]          33,024
         Residual-31          [128, 256, 4, 4]               0
           Conv2d-32          [128, 256, 4, 4]         590,080
      BatchNorm2d-33          [128, 256, 4, 4]             512
           Conv2d-34          [128, 256, 4, 4]         590,080
      BatchNorm2d-35          [128, 256, 4, 4]             512
         Residual-36          [128, 256, 4, 4]               0
           Conv2d-37          [128, 512, 2, 2]       1,180,160
      BatchNorm2d-38          [128, 512, 2, 2]           1,024
           Conv2d-39          [128, 512, 2, 2]       2,359,808
      BatchNorm2d-40          [128, 512, 2, 2]           1,024
           Conv2d-41          [128, 512, 2, 2]         131,584
         Residual-42          [128, 512, 2, 2]               0
           Conv2d-43          [128, 512, 2, 2]       2,359,808
      BatchNorm2d-44          [128, 512, 2, 2]           1,024
           Conv2d-45          [128, 512, 2, 2]       2,359,808
      BatchNorm2d-46          [128, 512, 2, 2]           1,024
         Residual-47          [128, 512, 2, 2]               0
AdaptiveAvgPool2d-48          [128, 512, 1, 1]               0
          Flatten-49                [128, 512]               0
           Linear-50                 [128, 10]           5,130
================================================================
Total params: 11,184,650
Trainable params: 11,184,650
Non-trainable params: 0
----------------------------------------------------------------
```

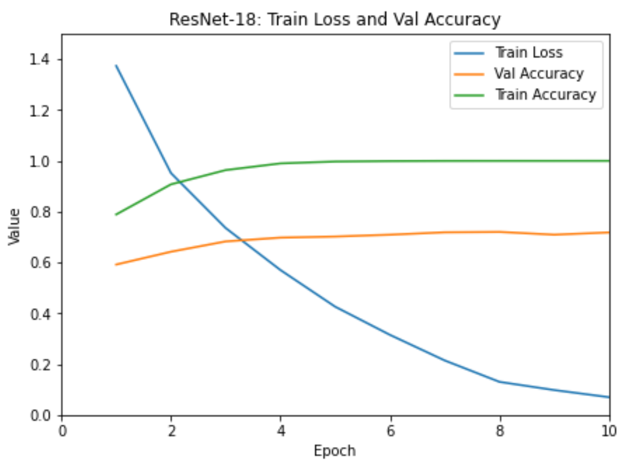
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(ResNet18.parameters(), lr=0.01)


# Begin training over 10 epochs
```

```
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, ResNet18, criterion, optimizer)
    valLoop(validation_loader, ResNet18, criterion)
```

```
Epoch 1
loss: 2.462340831756592
loss: 1.4582386016845703
loss: 1.2948322296142578
loss: 1.19211745262146
Training Accuracy:  78.89     Training Loss:  1.373415850160067
Validation Accuracy: 59.18     Validation Loss:  1.1354249217842198

Epoch 2
loss: 0.8900984525680542
loss: 0.8996784090995789
loss: 1.100325584411621
loss: 0.9876412153244019
Training Accuracy:  90.736     Training Loss:  0.9518261593016212
Validation Accuracy: 64.22     Validation Loss:  1.0047412620315068

Epoch 3
loss: 0.9186182618141174
loss: 0.7010466456413269
loss: 0.7943851351737976
loss: 0.7852962017059326
Training Accuracy:  96.352     Training Loss:  0.7352850441737553
Validation Accuracy: 68.28     Validation Loss:  0.9230640470227108

Epoch 4
loss: 0.5784720182418823
loss: 0.5407695770263672
loss: 0.555677592754364
loss: 0.4433528780937195
Training Accuracy:  98.976     Training Loss:  0.570110267842822
Validation Accuracy: 69.77     Validation Loss:  0.8741352377058584

Epoch 5
loss: 0.588233232498169
loss: 0.3058212995529175
loss: 0.4595407247543335
loss: 0.4960896074771881
Training Accuracy:  99.714     Training Loss:  0.42578436159874167
Validation Accuracy: 70.19     Validation Loss:  0.9387022875532319

Epoch 6
loss: 0.3579128086566925
loss: 0.2689177095890045
loss: 0.3291059732437134
loss: 0.3779588043689728
Training Accuracy:  99.894     Training Loss:  0.3149978412539148
Validation Accuracy: 70.92     Validation Loss:  0.9325100636180443

Epoch 7
loss: 0.13786502182483673
loss: 0.2778322994709015
loss: 0.20755711197853088
loss: 0.27415210008621216
Training Accuracy:  99.96000000000001     Training Loss:  0.21426948436233392
Validation Accuracy: 71.86     Validation Loss:  1.0094171053246608

Epoch 8
loss: 0.18470793962478638
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("ResNet-18: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
```

```
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: ~32ish minutes for 10 epochs
```




```
macs, params = get_model_complexity_info(ResNet18, (3, 64, 64), as_strings=True,
                                          print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module Residual is treated as a zero-op.
Warning: module Flatten is treated as a zero-op.
Sequential(
  11.18 M, 100.000% Params, 148.76 MMac, 100.000% MACs,
  (0): Sequential(
    9.6 k, 0.086% Params, 9.96 MMac, 6.696% MACs,
    (0): Conv2d(9.47 k, 0.085% Params, 9.7 MMac, 6.520% MACs, 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3)
    (1): BatchNorm2d(128, 0.001% Params, 131.07 KMac, 0.088% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.044% MACs, )
    (3): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.044% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
  )
  (1): Sequential(
    11.17 M, 99.868% Params, 138.8 MMac, 93.299% MACs,
    (0): Sequential(
      148.22 k, 1.325% Params, 37.95 MMac, 25.507% MACs,
      (0): Residual(
        74.11 k, 0.663% Params, 18.97 MMac, 12.754% MACs,
        (conv1): Conv2d(36.93 k, 0.330% Params, 9.45 MMac, 6.355% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (conv2): Conv2d(36.93 k, 0.330% Params, 9.45 MMac, 6.355% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (bn1): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.022% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
        (bn2): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.022% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
      )
      (1): Residual(
        74.11 k, 0.663% Params, 18.97 MMac, 12.754% MACs,
        (conv1): Conv2d(36.93 k, 0.330% Params, 9.45 MMac, 6.355% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (conv2): Conv2d(36.93 k, 0.330% Params, 9.45 MMac, 6.355% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (bn1): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.022% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
        (bn2): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.022% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
      )
    )
    (1): Sequential(
      525.95 k, 4.702% Params, 33.66 MMac, 22.627% MACs,
      (0): Residual(
        230.27 k, 2.059% Params, 14.74 MMac, 9.907% MACs,
        (conv1): Conv2d(73.86 k, 0.660% Params, 4.73 MMac, 3.177% MACs, 64, 128, kernel_size=(3, 3), stride=(2, 2), pa
        (conv2): Conv2d(147.58 k, 1.320% Params, 9.45 MMac, 6.349% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1),
        (conv3): Conv2d(8.32 k, 0.074% Params, 532.48 KMac, 0.358% MACs, 64, 128, kernel_size=(1, 1), stride=(2, 2))
        (bn1): BatchNorm2d(256, 0.002% Params, 16.38 KMac, 0.011% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
        (bn2): BatchNorm2d(256, 0.002% Params, 16.38 KMac, 0.011% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
      )
      (1): Residual(
        295.68 k, 2.644% Params, 18.92 MMac, 12.721% MACs,
```

```
      (conv1): Conv2d(147.58 k, 1.320% Params, 9.45 MMac, 6.349% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), 
      (conv2): Conv2d(147.58 k, 1.320% Params, 9.45 MMac, 6.349% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), 
      (bn1): BatchNorm2d(256, 0.002% Params, 16.38 KMac, 0.011% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
      (bn2): BatchNorm2d(256, 0.002% Params, 16.38 KMac, 0.011% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
    )
  )
  (2): Sequential(
    2.1 M, 18.780% Params, 33.61 MMac, 22.591% MACs,
    (0): Residual(
      919.3 k, 8.219% Params, 14.71 MMac, 9.887% MACs,
      (conv1): Conv2d(295.17 k, 2.639% Params, 4.72 MMac, 3.175% MACs, 128, 256, kernel_size=(3, 3), stride=(2, 2), 
      (conv2): Conv2d(590.08 k, 5.276% Params, 9.44 MMac, 6.346% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), 
      (conv3): Conv2d(33.02 k, 0.295% Params, 528.38 KMac, 0.355% MACs, 128, 256, kernel_size=(1, 1), stride=(2, 2))
      (bn1): BatchNorm2d(512, 0.005% Params, 8.19 KMac, 0.006% MACs, 256, eps=1e-05, momentum=0.1, affine=True, trac
      (bn2): BatchNorm2d(512, 0.005% Params, 8.19 KMac, 0.006% MACs, 256, eps=1e-05, momentum=0.1, affine=True, trac
```

3b) ResNet-26

```
res26BigBlock1 = nn.Sequential(
                 nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
                 nn.LazyBatchNorm2d(), nn.ReLU(),
                 nn.MaxPool2d(kernel_size=3, stride=2, padding=1))

res26BigBlock2 = body(((2, 64), (3, 128), (4, 256), (2, 512)))

res26BigBlock3 = nn.Sequential(
                 nn.AdaptiveAvgPool2d((1, 1)),
                 nn.Flatten(),
                 nn.LazyLinear(10))

ResNet26 = nn.Sequential(
    res26BigBlock1,
    res26BigBlock2,
    res26BigBlock3
)
```

3c) ResNet-32

```
res32BigBlock1 = nn.Sequential(
                 nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
                 nn.LazyBatchNorm2d(), nn.ReLU(),
                 nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
/usr/local/lib/python3.8/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
  warnings.warn('Lazy modules are a new feature under heavy development '
```

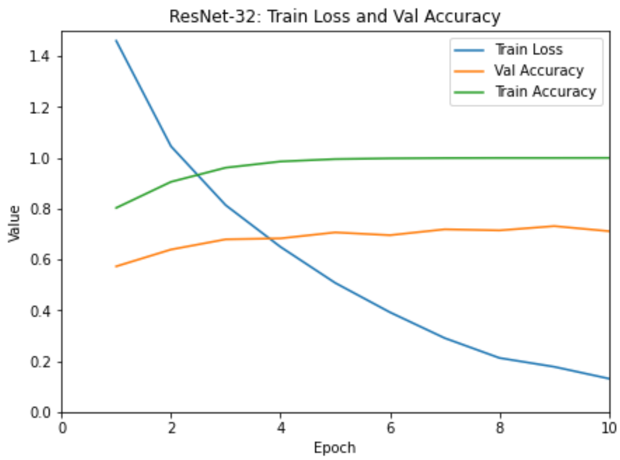
```
res32BigBlock2 = body(((3, 64), (4, 128), (6, 256), (3, 512)))

res32BigBlock3 = nn.Sequential(
                 nn.AdaptiveAvgPool2d((1, 1)),
                 nn.Flatten(),
                 nn.LazyLinear(10))

ResNet32 = nn.Sequential(
    res32BigBlock1,
    res32BigBlock2,
    res32BigBlock3
)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(ResNet32.parameters(), lr=0.01)

# Begin training over 10 epochs
```

```
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, ResNet32, criterion, optimizer)
    valLoop(validation_loader, ResNet32, criterion)
```

```
Epoch 1
loss: 2.511591911315918
loss: 1.5542881488800049
loss: 1.4366211891174316
loss: 1.3692954778671265
Training Accuracy:  80.294     Training Loss:  1.4604977520225604
Validation Accuracy: 57.28     Validation Loss:  1.1836518237862406

Epoch 2
loss: 0.9406509399414062
loss: 1.0219050645828247
loss: 0.9520958065986633
loss: 1.0801548957824707
Training Accuracy:  90.53999999999999     Training Loss:  1.0455130475866214
Validation Accuracy: 63.9     Validation Loss:  1.0262688322912288

Epoch 3
loss: 0.9648037552833557
loss: 1.0323445796966553
loss: 0.870963454246521
loss: 0.7506881952285767
Training Accuracy:  96.11     Training Loss:  0.8135033060827523
Validation Accuracy: 67.92     Validation Loss:  0.926001138324979

Epoch 4
loss: 0.6214495897293091
loss: 0.5398983359336853
loss: 0.7081708312034607
loss: 0.6848787069320679
Training Accuracy:  98.556     Training Loss:  0.6495592630732699
Validation Accuracy: 68.30000000000001     Validation Loss:  0.9299038437348378

Epoch 5
loss: 0.5414183735847473
loss: 0.4739900827407837
loss: 0.45866838097572327
loss: 0.5279505848884583
Training Accuracy:  99.492     Training Loss:  0.507748502523393
Validation Accuracy: 70.64     Validation Loss:  0.9080400036860116

Epoch 6
loss: 0.5411632061004639
loss: 0.451568603515625
loss: 0.36261674761772156
loss: 0.37999582290649414
Training Accuracy:  99.78399999999999     Training Loss:  0.3921046669754531
Validation Accuracy: 69.55     Validation Loss:  0.9883706252786177

Epoch 7
loss: 0.43339353799819946
loss: 0.19347110390663147
loss: 0.2977222204208374
loss: 0.4343850016593933
Training Accuracy:  99.90599999999999     Training Loss:  0.2905266760179149
Validation Accuracy: 71.86     Validation Loss:  0.9686806730077236

Epoch 8
loss: 0.17181630432605743
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("ResNet-32: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
```

```
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: 60 minutes for 10 epochs
```




```
macs, params = get_model_complexity_info(ResNet32, (3, 64, 64), as_strings=True,
                                          print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module Residual is treated as a zero-op.
Warning: module Flatten is treated as a zero-op.
Sequential(
  21.3 M, 100.000% Params, 300.07 MMac, 100.000% MACs,
  (0): Sequential(
    9.6 k, 0.045% Params, 9.96 MMac, 3.320% MACs,
    (0): Conv2d(9.47 k, 0.044% Params, 9.7 MMac, 3.232% MACs, 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3)
    (1): BatchNorm2d(128, 0.001% Params, 131.07 KMac, 0.044% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.022% MACs, )
    (3): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.022% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
  )
  (1): Sequential(
    21.28 M, 99.931% Params, 290.1 MMac, 96.678% MACs,
    (0): Sequential(
      222.34 k, 1.044% Params, 56.92 MMac, 18.969% MACs,
      (0): Residual(
        74.11 k, 0.348% Params, 18.97 MMac, 6.323% MACs,
        (conv1): Conv2d(36.93 k, 0.173% Params, 9.45 MMac, 3.150% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (conv2): Conv2d(36.93 k, 0.173% Params, 9.45 MMac, 3.150% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (bn1): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.011% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
        (bn2): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.011% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
      )
      (1): Residual(
        74.11 k, 0.348% Params, 18.97 MMac, 6.323% MACs,
        (conv1): Conv2d(36.93 k, 0.173% Params, 9.45 MMac, 3.150% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (conv2): Conv2d(36.93 k, 0.173% Params, 9.45 MMac, 3.150% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (bn1): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.011% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
        (bn2): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.011% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
      )
      (2): Residual(
        74.11 k, 0.348% Params, 18.97 MMac, 6.323% MACs,
        (conv1): Conv2d(36.93 k, 0.173% Params, 9.45 MMac, 3.150% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (conv2): Conv2d(36.93 k, 0.173% Params, 9.45 MMac, 3.150% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), pad
        (bn1): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.011% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
        (bn2): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.011% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
      )
    )
    (1): Sequential(
      1.12 M, 5.246% Params, 71.51 MMac, 23.831% MACs,
      (0): Residual(
        230.27 k, 1.081% Params, 14.74 MMac, 4.911% MACs,
        (conv1): Conv2d(73.86 k, 0.347% Params, 4.73 MMac, 1.575% MACs, 64, 128, kernel_size=(3, 3), stride=(2, 2), pa
```

```
        (conv2): Conv2d(147.58 k, 0.693% Params, 9.45 MMac, 3.148% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), 
        (conv3): Conv2d(8.32 k, 0.039% Params, 532.48 KMac, 0.177% MACs, 64, 128, kernel_size=(1, 1), stride=(2, 2))
        (bn1): BatchNorm2d(256, 0.001% Params, 16.38 KMac, 0.005% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
        (bn2): BatchNorm2d(256, 0.001% Params, 16.38 KMac, 0.005% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
      )
      (1): Residual(
        295.68 k, 1.388% Params, 18.92 MMac, 6.306% MACs,
        (conv1): Conv2d(147.58 k, 0.693% Params, 9.45 MMac, 3.148% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), 
        (conv2): Conv2d(147.58 k, 0.693% Params, 9.45 MMac, 3.148% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), 
        (bn1): BatchNorm2d(256, 0.001% Params, 16.38 KMac, 0.005% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
        (bn2): BatchNorm2d(256, 0.001% Params, 16.38 KMac, 0.005% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
      )
      (2): Residual(
        295.68 k, 1.388% Params, 18.92 MMac, 6.306% MACs,
        (conv1): Conv2d(147.58 k, 0.693% Params, 9.45 MMac, 3.148% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), 
```

Extra) DenseNet vs. ResNet

```
def convBlockDense(num_channels):
    return nn.Sequential(
        nn.LazyBatchNorm2d(), nn.ReLU(),
        nn.LazyConv2d(num_channels, kernel_size=3, padding=1))


class DenseBlock(nn.Module):
    def __init__(self, num_convs, num_channels):
        super(DenseBlock, self).__init__()
        layer = []
        for i in range(num_convs):
            layer.append(convBlockDense(num_channels))
        self.net = nn.Sequential(*layer)

    def forward(self, X):
        for blk in self.net:
            Y = blk(X)
            # Concatenate input and output of each block along the channels
            X = torch.cat((X, Y), dim=1)
        return X


def transitionBlock(num_channels):
    return nn.Sequential(
                        nn.LazyBatchNorm2d(), nn.ReLU(),
                        nn.LazyConv2d(num_channels, kernel_size=1),
                        nn.AvgPool2d(kernel_size=2, stride=2)
                 )


def denseBody(arch):
    growthRate = 32
    bodyBlocks = []
    numChannels = 64

    for i, numConvs in enumerate(arch):
        bodyBlocks.append(DenseBlock(numConvs, growthRate))
        numChannels += numConvs * growthRate

        if i != len(arch) - 1:
            numChannels //= 2
            bodyBlocks.append(transitionBlock(numChannels))

    return nn.Sequential(*bodyBlocks)


denseBigBlock1 = nn.Sequential(
                        nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
                        nn.LazyBatchNorm2d(), nn.ReLU(),
                        nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
                 )
```

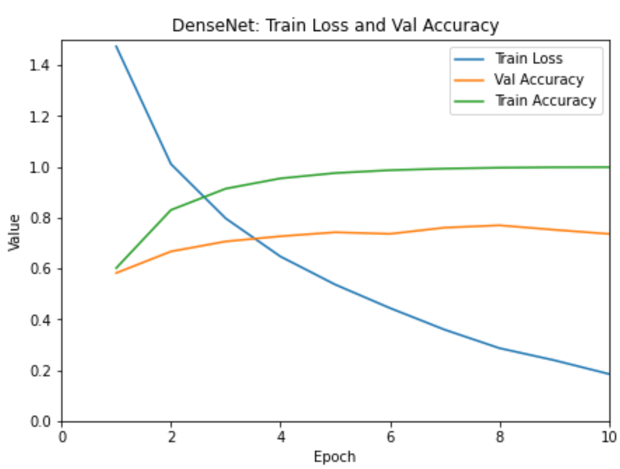
```
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/lazy.py:180: UserWarning: Lazy modules are a new feature under
  warnings.warn('Lazy modules are a new feature under heavy development '
```

```
denseBigBlock2 = denseBody((4,16,9,18,21))


denseBigBlock3 = nn.Sequential(
                        nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten(),
                        nn.LazyLinear(10)
)


denseNet = nn.Sequential(
                        denseBigBlock1,
                        denseBigBlock2,
                        denseBigBlock3
)


criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(denseNet.parameters(), lr=0.01)


# Begin training over 10 epochs
epochs = 10
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
    print("Epoch", actualEpoch)
    trainLoop(training_loader, denseNet, criterion, optimizer)
    valLoop(validation_loader, denseNet, criterion)
```

```
Epoch 1
loss: 2.305583953857422
loss: 1.5611282587051392
loss: 1.2717264890670776
loss: 1.160510540008545
Training Accuracy:  60.132     Training Loss:  1.4740133767237749
Validation Accuracy: 58.209999999999994     Validation Loss:  1.1806575874739056

Epoch 2
loss: 1.1500059366226196
loss: 1.0796420574188232
loss: 1.113866925239563
loss: 0.9303102493286133
Training Accuracy:  83.016     Training Loss:  1.0109011655878228
Validation Accuracy: 66.69     Validation Loss:  0.9380505990378464

Epoch 3
loss: 0.7960896492004395
loss: 0.725610613822937
loss: 0.8983271718025208
loss: 0.9390447735786438
Training Accuracy:  91.4     Training Loss:  0.7970694048935191
Validation Accuracy: 70.63000000000001     Validation Loss:  0.8357654304444035

Epoch 4
loss: 0.6795663833618164
loss: 0.499089777469635
loss: 0.7130162119865417
loss: 0.5280178189277649
Training Accuracy:  95.44200000000001     Training Loss:  0.6470762312869587
Validation Accuracy: 72.67     Validation Loss:  0.7892459238631816

Epoch 5
loss: 0.6986221075057983
loss: 0.4100870192050934
loss: 0.6042727828025818
loss: 0.45775923132896423
Training Accuracy:  97.548     Training Loss:  0.5366347064752408
Validation Accuracy: 74.24     Validation Loss:  0.7567275023158593

Epoch 6
loss: 0.47053924202919006
loss: 0.3864079713821411
```

```
loss: 0.4116270840167999
loss: 0.37730154395103455
Training Accuracy:  98.664     Training Loss:  0.4440811030242754
Validation Accuracy: 73.61999999999999     Validation Loss:  0.7830685724185992

Epoch 7
loss: 0.3975030183792114
loss: 0.3060373067855835
loss: 0.3046684265136719
loss: 0.4185643494129181
Training Accuracy:  99.28200000000001     Training Loss:  0.3589277767464328
Validation Accuracy: 76.06     Validation Loss:  0.7428525852251656

Epoch 8
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("DenseNet: Train Loss and Val Accuracy")
plt.plot(x, trainLosses, label="Train Loss")
plt.plot(x, valHist, label="Val Accuracy")
plt.plot(x, trainHist, label="Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.xlim(0, epochs)
plt.ylim(0, 1.5)
plt.show()

### Train time: ~60 minutes for 10 epochs
```




```
macs, params = get_model_complexity_info(denseNet, (3, 64, 64), as_strings=True,
                                           print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

```
Warning: module DenseBlock is treated as a zero-op.
Warning: module Flatten is treated as a zero-op.
Sequential(
  11.17 M, 100.000% Params, 199.44 MMac, 100.000% MACs,
  (0): Sequential(
    9.6 k, 0.086% Params, 9.96 MMac, 4.995% MACs,
    (0): Conv2d(9.47 k, 0.085% Params, 9.7 MMac, 4.863% MACs, 3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3)
    (1): BatchNorm2d(128, 0.001% Params, 131.07 KMac, 0.066% MACs, 64, eps=1e-05, momentum=0.1, affine=True, track_run
    (2): ReLU(0, 0.000% Params, 65.54 KMac, 0.033% MACs, )
    (3): MaxPool2d(0, 0.000% Params, 65.54 KMac, 0.033% MACs, kernel_size=3, stride=2, padding=1, dilation=1, ceil_mod
  )
  (1): Sequential(
    11.15 M, 99.815% Params, 189.47 MMac, 94.999% MACs,
    (0): DenseBlock(
      130.05 k, 1.164% Params, 33.41 MMac, 16.750% MACs,
      (net): Sequential(
        130.05 k, 1.164% Params, 33.41 MMac, 16.750% MACs,
        (0): Sequential(
          18.59 k, 0.166% Params, 4.78 MMac, 2.395% MACs,
          (0): BatchNorm2d(128, 0.001% Params, 32.77 KMac, 0.016% MACs, 64, eps=1e-05, momentum=0.1, affine=True, trac
          (1): ReLU(0, 0.000% Params, 16.38 KMac, 0.008% MACs, )
```

```
        (2): Conv2d(18.46 k, 0.165% Params, 4.73 MMac, 2.370% MACs, 64, 32, kernel_size=(3, 3), stride=(1, 1), paddi
      )
      (1): Sequential(
        27.87 k, 0.249% Params, 7.16 MMac, 3.590% MACs,
        (0): BatchNorm2d(192, 0.002% Params, 49.15 KMac, 0.025% MACs, 96, eps=1e-05, momentum=0.1, affine=True, trac
        (1): ReLU(0, 0.000% Params, 24.58 KMac, 0.012% MACs, )
        (2): Conv2d(27.68 k, 0.248% Params, 7.09 MMac, 3.553% MACs, 96, 32, kernel_size=(3, 3), stride=(1, 1), paddi
      )
      (2): Sequential(
        37.15 k, 0.333% Params, 9.54 MMac, 4.785% MACs,
        (0): BatchNorm2d(256, 0.002% Params, 65.54 KMac, 0.033% MACs, 128, eps=1e-05, momentum=0.1, affine=True, tra
        (1): ReLU(0, 0.000% Params, 32.77 KMac, 0.016% MACs, )
        (2): Conv2d(36.9 k, 0.330% Params, 9.45 MMac, 4.736% MACs, 128, 32, kernel_size=(3, 3), stride=(1, 1), paddi
      )
      (3): Sequential(
        46.43 k, 0.416% Params, 11.93 MMac, 5.980% MACs,
        (0): BatchNorm2d(320, 0.003% Params, 81.92 KMac, 0.041% MACs, 160, eps=1e-05, momentum=0.1, affine=True, tra
        (1): ReLU(0, 0.000% Params, 40.96 KMac, 0.021% MACs, )
        (2): Conv2d(46.11 k, 0.413% Params, 11.8 MMac, 5.919% MACs, 160, 32, kernel_size=(3, 3), stride=(1, 1), padd
      )
    )
  )
  (1): Sequential(
    18.91 k, 0.169% Params, 4.92 MMac, 2.464% MACs,
    (0): BatchNorm2d(384, 0.003% Params, 98.3 KMac, 0.049% MACs, 192, eps=1e-05, momentum=0.1, affine=True, track_ru
    (1): ReLU(0, 0.000% Params, 49.15 KMac, 0.025% MACs, )
    (2): Conv2d(18.53 k, 0.166% Params, 4.74 MMac, 2.378% MACs, 192, 96, kernel_size=(1, 1), stride=(1, 1))
    (3): AvgPool2d(0, 0.000% Params, 24.58 KMac, 0.012% MACs, kernel_size=2, stride=2, padding=0)
  )
  (2): DenseBlock(
    1.56 M, 13.959% Params, 100.16 MMac, 50.217% MACs,
    (net): Sequential(
      1.56 M, 13.959% Params, 100.16 MMac, 50.217% MACs,
      (0): Sequential(
        27.87 k, 0.249% Params, 1.79 MMac, 0.897% MACs,
        (0): BatchNorm2d(192, 0.002% Params, 12.29 KMac, 0.006% MACs, 96, eps=1e-05, momentum=0.1, affine=True, trac
        (1): ReLU(0, 0.000% Params, 6.14 KMac, 0.003% MACs, )
```