```
! 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/python3.8/dist-packages (from ptflops) (1.13.1+cu116)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torch->ptflops) (4.5.0)
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=573aa52379379f15b60def1503efad9c768691422b979d93eb9cc6745516a053
  Stored in directory: /root/.cache/pip/wheels/b6/86/d5/cf62a3571b005f91cd9accefc5e10f40214538be997198afad
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
```

```
# Define how we want images transformed
resize = (28, 28)
trans = transforms.Compose([transforms.Resize(resize),
                            transforms.ToTensor()])

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

# Create dataloaders for each set
training_loader = torch.utils.data.DataLoader(training_set, batch_size=32,
                                              shuffle=True, num_workers=2)
validation_loader = torch.utils.data.DataLoader(validation_set, batch_size=32,
                                                shuffle=False, num_workers=2)

classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')

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

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz
100%                                          26421880/26421880 [00:01<00:00, 26287683.24it/s]
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
100%                                          29515/29515 [00:00<00:00, 298289.42it/s]
Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
100%                                          4422102/4422102 [00:00<00:00, 7200833.18it/s]
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
100%                                          5148/5148 [00:00<00:00, 84372.85it/s]
Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw
```

1) MaxNet w/ ReLU

```
MaxLeNet = nn.Sequential(

    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.ReLU(),     #LazyConv2d(6, kernel_size=5, padding=2), nn.Sigmoid(),
    nn.MaxPool2d(2),                                          #nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=5), nn.ReLU(),               #LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
    nn.MaxPool2d(2),                                          #nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(400, 120), nn.ReLU(),                           #LazyLinear(120), nn.Sigmoid(),
    nn.Linear(120, 84), nn.ReLU(),                            #LazyLinear(84), nn.Sigmoid(),
    nn.Linear(84, 10)                                         #LazyLinear(num_classes))
)


summary(MaxLeNet, input_size = (1, 28, 28), batch_size = 32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [32, 6, 28, 28]             156
              ReLU-2           [32, 6, 28, 28]               0
         MaxPool2d-3           [32, 6, 14, 14]               0
            Conv2d-4          [32, 16, 10, 10]           2,416
              ReLU-5          [32, 16, 10, 10]               0
         MaxPool2d-6            [32, 16, 5, 5]               0
           Flatten-7                 [32, 400]               0
            Linear-8                 [32, 120]          48,120
              ReLU-9                 [32, 120]               0
           Linear-10                  [32, 84]          10,164
             ReLU-11                  [32, 84]               0
           Linear-12                  [32, 10]             850
================================================================
```

```
    Total params: 61,706
    Trainable params: 61,706
    Non-trainable params: 0
    ----------------------------------------------------------------
    Input size (MB): 0.10
    Forward/backward pass size (MB): 3.66
    Params size (MB): 0.24
    Estimated Total Size (MB): 3.99
    ----------------------------------------------------------------
```

```
# --- APPLY TO ALL EXPERIMENTS ---
macs, params = get_model_complexity_info(MaxLeNet, (1, 28, 28), 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.
    Sequential(
      61.71 k, 100.000% Params, 435.85 KMac, 100.000% MACs,
      (0): Conv2d(156, 0.253% Params, 122.3 KMac, 28.061% MACs, 1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): ReLU(0, 0.000% Params, 4.7 KMac, 1.079% MACs, )
      (2): MaxPool2d(0, 0.000% Params, 4.7 KMac, 1.079% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): Conv2d(2.42 k, 3.915% Params, 241.6 KMac, 55.432% MACs, 6, 16, kernel_size=(5, 5), stride=(1, 1))
      (4): ReLU(0, 0.000% Params, 1.6 KMac, 0.367% MACs, )
      (5): MaxPool2d(0, 0.000% Params, 1.6 KMac, 0.367% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (6): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
      (7): Linear(48.12 k, 77.983% Params, 48.12 KMac, 11.040% MACs, in_features=400, out_features=120, bias=True)
      (8): ReLU(0, 0.000% Params, 120.0 Mac, 0.028% MACs, )
      (9): Linear(10.16 k, 16.472% Params, 10.16 KMac, 2.332% MACs, in_features=120, out_features=84, bias=True)
      (10): ReLU(0, 0.000% Params, 84.0 Mac, 0.019% MACs, )
      (11): Linear(850, 1.377% Params, 850.0 Mac, 0.195% MACs, in_features=84, out_features=10, bias=True)
    )
    Computational complexity:  435.85 KMac
    Number of parameters:  61.71 k
```

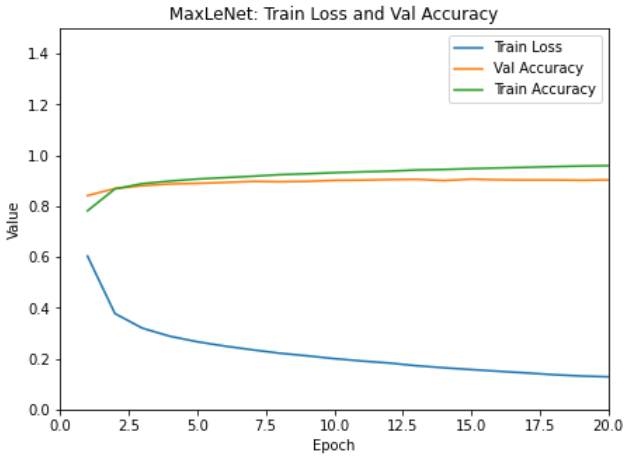
```
# Define the training loop for each epoch
def trainLoop(dataloader, model, loss_fn, optimizer):

    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 % 1000 == 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)

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


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


# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

for t in range(epochs):
    actualEpoch = t+1
```

```
print("Epoch", actualEpoch)
trainLoop(training_loader, MaxLeNet, criterion, optimizer)
valLoop(validation_loader, MaxLeNet, criterion)
```

```
Validation Accuracy: 89.74     Validation Loss:  0.29269843495572906

Epoch 10
loss: 0.26472964882850647
loss: 0.058733873069286346
Validation Accuracy: 90.14999999999999     Validation Loss:  0.27200968651630625

Epoch 11
loss: 0.059090666472911835
loss: 0.15066036581993103
Validation Accuracy: 90.22     Validation Loss:  0.26984099586741234

Epoch 12
loss: 0.175387904047966
loss: 0.14507068693637848
Validation Accuracy: 90.46     Validation Loss:  0.27179381690514737

Epoch 13
loss: 0.17627651989459991
loss: 0.21958458423614502
Validation Accuracy: 90.55     Validation Loss:  0.27978623022857946

Epoch 14
loss: 0.13553816080093384
loss: 0.17807172238826752
Validation Accuracy: 90.07     Validation Loss:  0.2844237531550205

Epoch 15
loss: 0.07433485239744186
loss: 0.03841002658009529
Validation Accuracy: 90.62     Validation Loss:  0.28944059723791793

Epoch 16
loss: 0.025794582441449165
loss: 0.03645904362201691
Validation Accuracy: 90.38000000000001     Validation Loss:  0.2970541306494619

Epoch 17
loss: 0.09444670379161835
loss: 0.2125445306301117
Validation Accuracy: 90.28     Validation Loss:  0.3068658906442765

Epoch 18
loss: 0.3425464332103729
loss: 0.22572341561317444
Validation Accuracy: 90.3     Validation Loss:  0.3051984432799081

Epoch 19
loss: 0.10637159645557404
loss: 0.1293027549982071
Validation Accuracy: 90.18     Validation Loss:  0.31781230903804875

Epoch 20
loss: 0.18390409648418427
loss: 0.14328217506408691
Validation Accuracy: 90.33     Validation Loss:  0.31958482676325517
```

```
#trainLosses, valHist, interLosses, avgLosses


# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("MaxLeNet: 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()
```



2a) Experimental LeNet - Window/Kernel Size

```
ExpLeNetWind = nn.Sequential(

            nn.Conv2d(1, 6, kernel_size=3, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(6, 16, kernel_size=3), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(576, 120), nn.ReLU(),
            nn.Linear(120, 84), nn.ReLU(),
            nn.Linear(84, 10)
)
```

```
summary(ExpLeNetWind, input_size = (1, 28, 28), batch_size =32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [32, 6, 30, 30]              60
              ReLU-2           [32, 6, 30, 30]               0
         MaxPool2d-3           [32, 6, 15, 15]               0
            Conv2d-4          [32, 16, 13, 13]             880
              ReLU-5          [32, 16, 13, 13]               0
         MaxPool2d-6            [32, 16, 6, 6]               0
           Flatten-7                 [32, 576]               0
            Linear-8                 [32, 120]          69,240
              ReLU-9                 [32, 120]               0
           Linear-10                  [32, 84]          10,164
             ReLU-11                  [32, 84]               0
           Linear-12                  [32, 10]             850
================================================================
Total params: 81,194
Trainable params: 81,194
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.10
Forward/backward pass size (MB): 4.67
Params size (MB): 0.31
Estimated Total Size (MB): 5.08
----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(ExpLeNetWind, (1, 28, 28), 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.
Sequential(
  81.19 k, 100.000% Params, 299.39 KMac, 100.000% MACs,
  (0): Conv2d(60, 0.074% Params, 54.0 KMac, 18.037% MACs, 1, 6, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (1): ReLU(0, 0.000% Params, 5.4 KMac, 1.804% MACs, )
  (2): MaxPool2d(0, 0.000% Params, 5.4 KMac, 1.804% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(880, 1.084% Params, 148.72 KMac, 49.675% MACs, 6, 16, kernel_size=(3, 3), stride=(1, 1))
  (4): ReLU(0, 0.000% Params, 2.7 KMac, 0.903% MACs, )
  (5): MaxPool2d(0, 0.000% Params, 2.7 KMac, 0.903% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
  (7): Linear(69.24 k, 85.277% Params, 69.24 KMac, 23.127% MACs, in_features=576, out_features=120, bias=True)
  (8): ReLU(0, 0.000% Params, 120.0 Mac, 0.040% MACs, )
  (9): Linear(10.16 k, 12.518% Params, 10.16 KMac, 3.395% MACs, in_features=120, out_features=84, bias=True)
  (10): ReLU(0, 0.000% Params, 84.0 Mac, 0.028% MACs, )
  (11): Linear(850, 1.047% Params, 850.0 Mac, 0.284% MACs, in_features=84, out_features=10, bias=True)
)
Computational complexity:  299.39 KMac
Number of parameters:  81.19 k
```

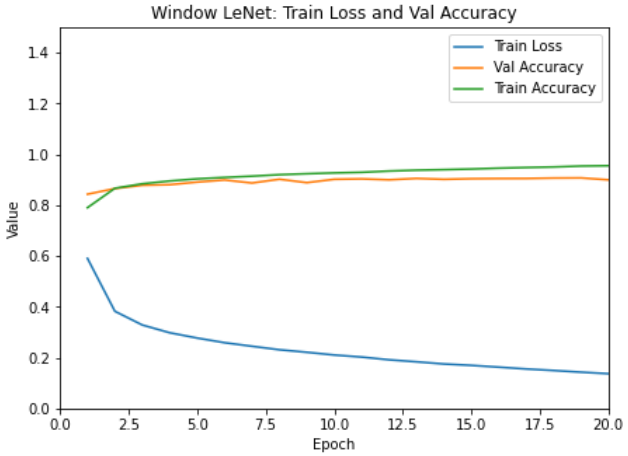
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(ExpLeNetWind.parameters(), lr=0.001)
```

```
# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

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

```
Validation Accuracy: 88.85     Validation Loss:  0.2982958235358373


Epoch 10
loss: 0.28388839960098267
loss: 0.14293548464775085
Validation Accuracy: 90.16999999999999     Validation Loss:  0.267375189001663

Epoch 11
loss: 0.1129353865981102
loss: 0.26229333877563477
Validation Accuracy: 90.33     Validation Loss:  0.264691455861012

Epoch 12
loss: 0.14219126105308533
loss: 0.2240721434354782
Validation Accuracy: 89.97     Validation Loss:  0.28073896977086416

Epoch 13
loss: 0.2418632209300995
loss: 0.17405880987644196
Validation Accuracy: 90.51     Validation Loss:  0.2779467262970373

Epoch 14
loss: 0.11901608854532242
loss: 0.04241831600666046
Validation Accuracy: 90.16     Validation Loss:  0.2886403740678256

Epoch 15
loss: 0.21292787790298462
loss: 0.08505263179540634
Validation Accuracy: 90.4     Validation Loss:  0.27278012166412685

Epoch 16
loss: 0.0945063978433609
loss: 0.24951516091823578
Validation Accuracy: 90.47     Validation Loss:  0.2857096066799598

Epoch 17
loss: 0.18068593740463257
loss: 0.049970388412475586
Validation Accuracy: 90.49000000000001     Validation Loss:  0.2936227724325067

Epoch 18
loss: 0.18012750148773193
```

```
loss: 0.180588960647583
Validation Accuracy: 90.66     Validation Loss:  0.28525481897349747

Epoch 19
loss: 0.13082930445671082
loss: 0.22957293689250946
Validation Accuracy: 90.71000000000001     Validation Loss:  0.2929377834291789

Epoch 20
loss: 0.10809242725372314
loss: 0.21371260285377502
Validation Accuracy: 89.96     Validation Loss:  0.3274085383398083
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("Window LeNet: 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()
```



2b) Experimental LeNet - Output Channels

```
ExpLeNetChan = nn.Sequential(

            nn.Conv2d(1, 32, kernel_size=5, padding=2), nn.ReLU(),
```

```
            nn.MaxPool2d(2),
            nn.Conv2d(32, 32, kernel_size=5), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(800, 256), nn.ReLU(),
            nn.Linear(256, 64), nn.ReLU(),
            nn.Linear(64, 10)
)

summary(ExpLeNetChan, input_size = (1, 28, 28), batch_size =32)
```

```
    ----------------------------------------------------------------
            Layer (type)               Output Shape         Param #
    ================================================================
                Conv2d-1          [32, 32, 28, 28]             832
                  ReLU-2          [32, 32, 28, 28]               0
             MaxPool2d-3          [32, 32, 14, 14]               0
                Conv2d-4          [32, 32, 10, 10]          25,632
                  ReLU-5          [32, 32, 10, 10]               0
             MaxPool2d-6            [32, 32, 5, 5]               0
               Flatten-7                 [32, 800]               0
                Linear-8                 [32, 256]         205,056
                  ReLU-9                 [32, 256]               0
               Linear-10                  [32, 64]          16,448
                 ReLU-11                  [32, 64]               0
               Linear-12                  [32, 10]             650
    ================================================================
    Total params: 248,618
    Trainable params: 248,618
    Non-trainable params: 0
    ----------------------------------------------------------------
    Input size (MB): 0.10
    Forward/backward pass size (MB): 15.89
    Params size (MB): 0.95
    Estimated Total Size (MB): 16.94
    ----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(ExpLeNetChan, (1, 28, 28), 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.
    Sequential(
      248.62 k, 100.000% Params, 3.49 MMac, 100.000% MACs,
      (0): Conv2d(832, 0.335% Params, 652.29 KMac, 18.666% MACs, 1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (1): ReLU(0, 0.000% Params, 25.09 KMac, 0.718% MACs, )
      (2): MaxPool2d(0, 0.000% Params, 25.09 KMac, 0.718% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): Conv2d(25.63 k, 10.310% Params, 2.56 MMac, 73.349% MACs, 32, 32, kernel_size=(5, 5), stride=(1, 1))
      (4): ReLU(0, 0.000% Params, 3.2 KMac, 0.092% MACs, )
      (5): MaxPool2d(0, 0.000% Params, 3.2 KMac, 0.092% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (6): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
      (7): Linear(205.06 k, 82.478% Params, 205.06 KMac, 5.868% MACs, in_features=800, out_features=256, bias=True)
      (8): ReLU(0, 0.000% Params, 256.0 Mac, 0.007% MACs, )
      (9): Linear(16.45 k, 6.616% Params, 16.45 KMac, 0.471% MACs, in_features=256, out_features=64, bias=True)
      (10): ReLU(0, 0.000% Params, 64.0 Mac, 0.002% MACs, )
      (11): Linear(650, 0.261% Params, 650.0 Mac, 0.019% MACs, in_features=64, out_features=10, bias=True)
    )
```

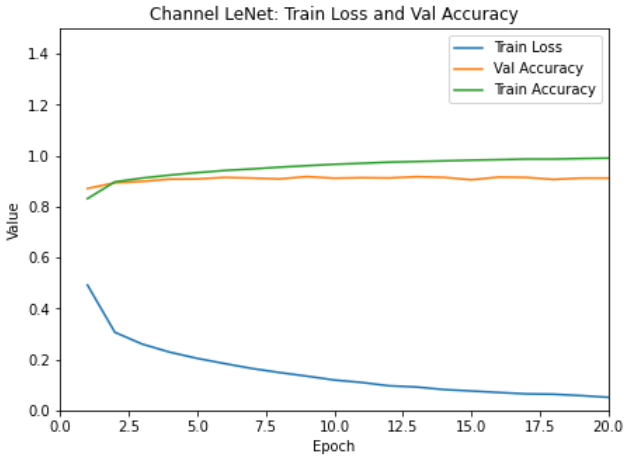
```
Computational complexity:  3.49 MMac
Number of parameters:  248.62 k
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(ExpLeNetChan.parameters(), lr=0.001)


# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

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

```
loss: 0.10545231401920319
Validation Accuracy: 91.77     Validation Loss:  0.2635595384544839

Epoch 10
loss: 0.13737992942333221
loss: 0.10598868876695633
Validation Accuracy: 91.14999999999999     Validation Loss:  0.2752029822704891

Epoch 11
loss: 0.05570889264345169
loss: 0.01653691940009594
Validation Accuracy: 91.34     Validation Loss:  0.3112751868026801

Epoch 12
loss: 0.0803329274058342
loss: 0.12001214921474457
Validation Accuracy: 91.23     Validation Loss:  0.30967554425338684

Epoch 13
loss: 0.06855528056621552
loss: 0.03943931683897972
Validation Accuracy: 91.73     Validation Loss:  0.3185548261760142

Epoch 14
loss: 0.1979738175868988
loss: 0.037628620862960815
Validation Accuracy: 91.47     Validation Loss:  0.325786305490298

Epoch 15
loss: 0.00550600653514266
loss: 0.02556084655225277
Validation Accuracy: 90.53     Validation Loss:  0.392708921642489

Epoch 16
loss: 0.0077915857546031475
loss: 0.009563696570694447
```

```
Validation Accuracy: 91.58     Validation Loss:  0.39452129550510273

Epoch 17
loss: 0.019824210554361343
loss: 0.11915914714336395
Validation Accuracy: 91.45     Validation Loss:  0.3854067757504043

Epoch 18
loss: 0.07205665856599808
loss: 0.03600491210818291
Validation Accuracy: 90.67     Validation Loss:  0.4403740083278059

Epoch 19
loss: 0.004822830203920603
loss: 0.01705688238143921
Validation Accuracy: 91.18     Validation Loss:  0.38978407756029704

Epoch 20
loss: 0.015321574173867702
loss: 0.007182026281952858
Validation Accuracy: 91.18     Validation Loss:  0.47341194060242797
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("Channel LeNet: 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()
```



2c) Experimental LeNet - Conv Layers

```
ExpLeNetConv = nn.Sequential(

            nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(6, 16, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Linear(32, 16), nn.ReLU(),
            nn.Linear(16, 10)
)
```

```
summary(ExpLeNetConv, input_size = (1, 28, 28), batch_size =32)
```

```
    ----------------------------------------------------------------
            Layer (type)               Output Shape         Param #
    ================================================================
                Conv2d-1           [32, 6, 28, 28]             156
                  ReLU-2           [32, 6, 28, 28]               0
             MaxPool2d-3           [32, 6, 14, 14]               0
                Conv2d-4          [32, 16, 14, 14]           2,416
                  ReLU-5          [32, 16, 14, 14]               0
             MaxPool2d-6            [32, 16, 7, 7]               0
                Conv2d-7            [32, 32, 7, 7]          12,832
                  ReLU-8            [32, 32, 7, 7]               0
             MaxPool2d-9            [32, 32, 3, 3]               0
               Conv2d-10            [32, 64, 3, 3]          51,264
                 ReLU-11            [32, 64, 3, 3]               0
            MaxPool2d-12            [32, 64, 1, 1]               0
              Flatten-13                  [32, 64]               0
               Linear-14                  [32, 32]           2,080
                 ReLU-15                  [32, 32]               0
               Linear-16                  [32, 16]             528
                 ReLU-17                  [32, 16]               0
               Linear-18                  [32, 10]             170
    ================================================================
    Total params: 69,446
    Trainable params: 69,446
    Non-trainable params: 0
    ----------------------------------------------------------------
    Input size (MB): 0.10
    Forward/backward pass size (MB): 5.48
    Params size (MB): 0.26
    Estimated Total Size (MB): 5.84
    ----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(ExpLeNetConv, (1, 28, 28), 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.
Sequential(
  69.45 k, 100.000% Params, 1.71 MMac, 100.000% MACs,
  (0): Conv2d(156, 0.225% Params, 122.3 KMac, 7.157% MACs, 1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU(0, 0.000% Params, 4.7 KMac, 0.275% MACs, )
  (2): MaxPool2d(0, 0.000% Params, 4.7 KMac, 0.275% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(2.42 k, 3.479% Params, 473.54 KMac, 27.712% MACs, 6, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (4): ReLU(0, 0.000% Params, 3.14 KMac, 0.184% MACs, )
  (5): MaxPool2d(0, 0.000% Params, 3.14 KMac, 0.184% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(12.83 k, 18.478% Params, 628.77 KMac, 36.796% MACs, 16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (7): ReLU(0, 0.000% Params, 1.57 KMac, 0.092% MACs, )
  (8): MaxPool2d(0, 0.000% Params, 1.57 KMac, 0.092% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Conv2d(51.26 k, 73.819% Params, 461.38 KMac, 27.000% MACs, 32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (10): ReLU(0, 0.000% Params, 576.0 Mac, 0.034% MACs, )
  (11): MaxPool2d(0, 0.000% Params, 576.0 Mac, 0.034% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
  (13): Linear(2.08 k, 2.995% Params, 2.08 KMac, 0.122% MACs, in_features=64, out_features=32, bias=True)
  (14): ReLU(0, 0.000% Params, 32.0 Mac, 0.002% MACs, )
  (15): Linear(528, 0.760% Params, 528.0 Mac, 0.031% MACs, in_features=32, out_features=16, bias=True)
  (16): ReLU(0, 0.000% Params, 16.0 Mac, 0.001% MACs, )
  (17): Linear(170, 0.245% Params, 170.0 Mac, 0.010% MACs, in_features=16, out_features=10, bias=True)
)
Computational complexity:  1.71 MMac
Number of parameters:  69.45 k
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(ExpLeNetConv.parameters(), lr=0.001)
```

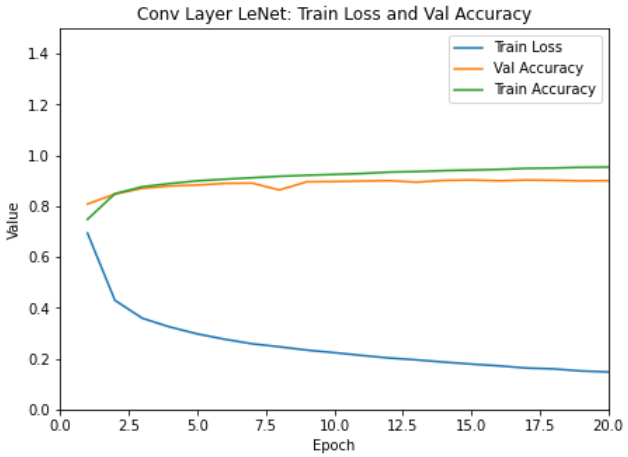
```
# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

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

```
loss: 0.18912243843078613
Validation Accuracy: 89.59     Validation Loss:  0.30505555555129205

Epoch 10
loss: 0.22251185774803162
loss: 0.1909756362438202
Validation Accuracy: 89.67     Validation Loss:  0.29852879498475277

Epoch 11
loss: 0.05390220135450363
```

```
loss: 0.5136352181434631
Validation Accuracy: 89.86     Validation Loss:  0.2922220514795651

Epoch 12
loss: 0.30488497018814087
loss: 0.16639462113380432
Validation Accuracy: 90.01     Validation Loss:  0.2882175712992018

Epoch 13
loss: 0.07017980515956879
loss: 0.16630296409130096
Validation Accuracy: 89.42999999999999     Validation Loss:  0.3089518357818119

Epoch 14
loss: 0.21424993872642517
loss: 0.05710185319185257
Validation Accuracy: 90.12     Validation Loss:  0.303217229138786

Epoch 15
loss: 0.512250542640686
loss: 0.2508762776851654
Validation Accuracy: 90.29     Validation Loss:  0.3028508698026212

Epoch 16
loss: 0.16260407865047455
loss: 0.22891202569007874
Validation Accuracy: 89.96     Validation Loss:  0.3137348194639332

Epoch 17
loss: 0.05506058782339096
loss: 0.1931689828634262
Validation Accuracy: 90.28     Validation Loss:  0.3300599384660157

Epoch 18
loss: 0.09002522379159927
loss: 0.23958276212215424
Validation Accuracy: 90.16     Validation Loss:  0.33050147842127864

Epoch 19
loss: 0.046834371984004974
loss: 0.12266209721565247
Validation Accuracy: 89.92     Validation Loss:  0.3062375673827843

Epoch 20
loss: 0.07435212284326553
loss: 0.08859160542488098
Validation Accuracy: 90.02     Validation Loss:  0.3225944591656375
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("Conv Layer LeNet: 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()
```



2d) Experimental LeNet - FC Layers

```
ExpLeNetFC = nn.Sequential(

            nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(6, 16, kernel_size=5), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(400, 256), nn.ReLU(),
            nn.Linear(256, 128), nn.ReLU(),
            nn.Linear(128, 64), nn.ReLU(),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Linear(32, 10)

)

summary(ExpLeNetFC, input_size = (1, 28, 28), batch_size =32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [32, 6, 28, 28]             156
              ReLU-2            [32, 6, 28, 28]               0
         MaxPool2d-3            [32, 6, 14, 14]               0
            Conv2d-4           [32, 16, 10, 10]           2,416
              ReLU-5           [32, 16, 10, 10]               0
         MaxPool2d-6             [32, 16, 5, 5]               0
```

```
            Flatten-7                [32, 400]               0
             Linear-8                [32, 256]         102,656
               ReLU-9                [32, 256]               0
            Linear-10                [32, 128]          32,896
              ReLU-11                [32, 128]               0
            Linear-12                 [32, 64]           8,256
              ReLU-13                 [32, 64]               0
            Linear-14                 [32, 32]           2,080
              ReLU-15                 [32, 32]               0
            Linear-16                 [32, 10]             330
================================================================
Total params: 148,790
Trainable params: 148,790
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.10
Forward/backward pass size (MB): 3.80
Params size (MB): 0.57
Estimated Total Size (MB): 4.46
----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(ExpLeNetFC, (1, 28, 28), 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.
Sequential(
  148.79 k, 100.000% Params, 523.21 KMac, 100.000% MACs,
  (0): Conv2d(156, 0.105% Params, 122.3 KMac, 23.376% MACs, 1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU(0, 0.000% Params, 4.7 KMac, 0.899% MACs, )
  (2): MaxPool2d(0, 0.000% Params, 4.7 KMac, 0.899% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(2.42 k, 1.624% Params, 241.6 KMac, 46.176% MACs, 6, 16, kernel_size=(5, 5), stride=(1, 1))
  (4): ReLU(0, 0.000% Params, 1.6 KMac, 0.306% MACs, )
  (5): MaxPool2d(0, 0.000% Params, 1.6 KMac, 0.306% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
  (7): Linear(102.66 k, 68.994% Params, 102.66 KMac, 19.620% MACs, in_features=400, out_features=256, bias=True)
  (8): ReLU(0, 0.000% Params, 256.0 Mac, 0.049% MACs, )
  (9): Linear(32.9 k, 22.109% Params, 32.9 KMac, 6.287% MACs, in_features=256, out_features=128, bias=True)
  (10): ReLU(0, 0.000% Params, 128.0 Mac, 0.024% MACs, )
  (11): Linear(8.26 k, 5.549% Params, 8.26 KMac, 1.578% MACs, in_features=128, out_features=64, bias=True)
  (12): ReLU(0, 0.000% Params, 64.0 Mac, 0.012% MACs, )
  (13): Linear(2.08 k, 1.398% Params, 2.08 KMac, 0.398% MACs, in_features=64, out_features=32, bias=True)
  (14): ReLU(0, 0.000% Params, 32.0 Mac, 0.006% MACs, )
  (15): Linear(330, 0.222% Params, 330.0 Mac, 0.063% MACs, in_features=32, out_features=10, bias=True)
)
Computational complexity:  523.21 KMac
Number of parameters:  148.79 k
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(ExpLeNetFC.parameters(), lr=0.001)
```

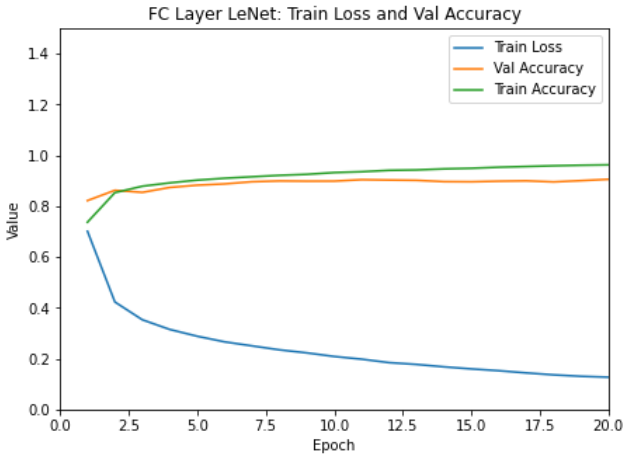
```
# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
```

```
trainLosses = []
interLosses = []
avgLosses = []

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

```
Epoch 1
loss: 2.290706157684326
loss: 0.307341068983078
Validation Accuracy: 82.17999999999999     Validation Loss:  0.48632207193884985

Epoch 2
loss: 0.4437580108642578
loss: 0.35284027457237244
Validation Accuracy: 86.22999999999999     Validation Loss:  0.38711444512247656

Epoch 3
loss: 0.37868165969848633
loss: 0.2912781834602356
Validation Accuracy: 85.41     Validation Loss:  0.397343154841909

Epoch 4
loss: 0.4657308757305145
loss: 0.3759935200214386
Validation Accuracy: 87.35000000000001     Validation Loss:  0.3495084028750563

Epoch 5
loss: 0.17693762481212616
loss: 0.2395053654909134
Validation Accuracy: 88.24     Validation Loss:  0.3206745018831457

Epoch 6
loss: 0.28645333647727966
loss: 0.15221655368804932
Validation Accuracy: 88.75     Validation Loss:  0.3223715439653054

Epoch 7
loss: 0.5499658584594727
loss: 0.26621702313423157
Validation Accuracy: 89.60000000000001     Validation Loss:  0.2914217749116615

Epoch 8
loss: 0.1569925844669342
loss: 0.20477882027626038
Validation Accuracy: 89.9     Validation Loss:  0.29118004319862056

Epoch 9
loss: 0.3153538107872009
loss: 0.14894960820674896
Validation Accuracy: 89.82     Validation Loss:  0.2950234032286623

Epoch 10
loss: 0.22837993502616882
loss: 0.26834988594055176
Validation Accuracy: 89.84     Validation Loss:  0.3026259756935671
```

```
Epoch 11
loss: 0.2310730218887329
loss: 0.11507314443588257
Validation Accuracy: 90.41     Validation Loss:  0.28189363941169393

Epoch 12
loss: 0.13440990447998047
loss: 0.28459158539772034
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("FC Layer LeNet: 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()
```



2e) Experimental LeNet - Learning Rate

```
ExpLeNetLearn = nn.Sequential(

    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(6, 16, kernel_size=5), nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Flatten(),
    nn.Linear(400, 120), nn.ReLU(),
```

```
    nn.Linear(120, 84), nn.ReLU(),
    nn.Linear(84, 10)
)


summary(ExpLeNetLearn, input_size = (1, 28, 28), batch_size =32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [32, 6, 28, 28]             156
              ReLU-2           [32, 6, 28, 28]               0
         MaxPool2d-3           [32, 6, 14, 14]               0
            Conv2d-4          [32, 16, 10, 10]           2,416
              ReLU-5          [32, 16, 10, 10]               0
         MaxPool2d-6            [32, 16, 5, 5]               0
           Flatten-7                 [32, 400]               0
            Linear-8                 [32, 120]          48,120
              ReLU-9                 [32, 120]               0
           Linear-10                  [32, 84]          10,164
             ReLU-11                  [32, 84]               0
           Linear-12                  [32, 10]             850
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.10
Forward/backward pass size (MB): 3.66
Params size (MB): 0.24
Estimated Total Size (MB): 3.99
----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(ExpLeNetLearn, (1, 28, 28), as_strings=True,
                                         print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

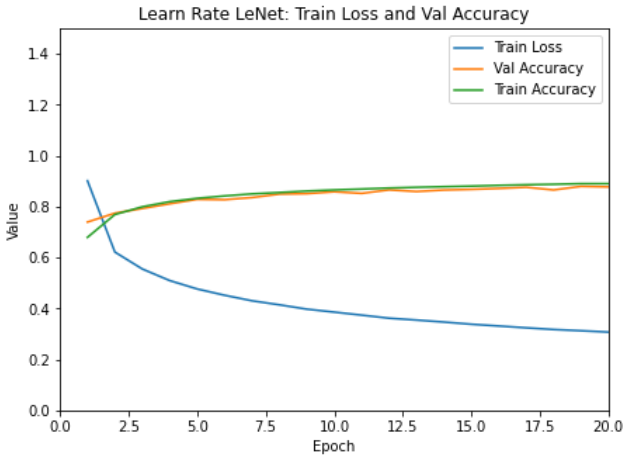
```
Warning: module Flatten is treated as a zero-op.
Sequential(
  61.71 k, 100.000% Params, 435.85 KMac, 100.000% MACs,
  (0): Conv2d(156, 0.253% Params, 122.3 KMac, 28.061% MACs, 1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU(0, 0.000% Params, 4.7 KMac, 1.079% MACs, )
  (2): MaxPool2d(0, 0.000% Params, 4.7 KMac, 1.079% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(2.42 k, 3.915% Params, 241.6 KMac, 55.432% MACs, 6, 16, kernel_size=(5, 5), stride=(1, 1))
  (4): ReLU(0, 0.000% Params, 1.6 KMac, 0.367% MACs, )
  (5): MaxPool2d(0, 0.000% Params, 1.6 KMac, 0.367% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
  (7): Linear(48.12 k, 77.983% Params, 48.12 KMac, 11.040% MACs, in_features=400, out_features=120, bias=True)
  (8): ReLU(0, 0.000% Params, 120.0 Mac, 0.028% MACs, )
  (9): Linear(10.16 k, 16.472% Params, 10.16 KMac, 2.332% MACs, in_features=120, out_features=84, bias=True)
  (10): ReLU(0, 0.000% Params, 84.0 Mac, 0.019% MACs, )
  (11): Linear(850, 1.377% Params, 850.0 Mac, 0.195% MACs, in_features=84, out_features=10, bias=True)
)
Computational complexity:  435.85 KMac
Number of parameters:  61.71 k
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(ExpLeNetLearn.parameters(), lr=0.0001)

# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

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

```
loss: 0.46455809473991394
Validation Accuracy: 85.08     Validation Loss:  0.4200473606300811

Epoch 10
loss: 0.20845942199230194
loss: 0.4548656940460205
Validation Accuracy: 85.81     Validation Loss:  0.40200565555415596

Epoch 11
loss: 0.28233468532562256
loss: 0.3258947432041168
Validation Accuracy: 85.19     Validation Loss:  0.4096351430748408

Epoch 12
loss: 0.25641652941703796
loss: 0.2643296718597412
Validation Accuracy: 86.55000000000001     Validation Loss:  0.381915259915895

Epoch 13
loss: 0.29789093136787415
loss: 0.37787431478500366
Validation Accuracy: 85.92     Validation Loss:  0.39304941247541686

Epoch 14
loss: 0.3465706408023834
loss: 0.4797326624393463
Validation Accuracy: 86.5     Validation Loss:  0.37464668042362687

Epoch 15
loss: 0.4175805449485779
loss: 0.44802987575531006
Validation Accuracy: 86.76     Validation Loss:  0.3647009155001884

Epoch 16
loss: 0.43517136573791504
loss: 0.36722302436828613
Validation Accuracy: 87.12     Validation Loss:  0.35968529384214276

Epoch 17
loss: 0.15792155265808105
loss: 0.1539457142353058
Validation Accuracy: 87.59     Validation Loss:  0.3501290382025912
```

```
Epoch 18
loss: 0.387608140707016
loss: 0.3448188602924347
Validation Accuracy: 86.53     Validation Loss:  0.36654538631486816

Epoch 19
loss: 0.3941614329814911
loss: 0.3907373547554016
Validation Accuracy: 87.94     Validation Loss:  0.33603902344410413

Epoch 20
loss: 0.21067094802856445
loss: 0.40098559856414795
Validation Accuracy: 87.72999999999999     Validation Loss:  0.3397259250426064
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("Learn Rate LeNet: 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()
```




```
trainLosses, trainHist
```

```
([0.9013984229405722,
  0.6214531558195749,
  0.5549947891314825,
```

```
       0.5094297926028569,
       0.47651880617936454,
       0.451864084148407,
       0.4300498764872551,
       0.41432707730134327,
       0.3972409782886505,
       0.38560177760918934,
       0.3737924686113993,
       0.3616999568462372,
       0.3544420238931974,
       0.34659421243866284,
       0.3379843934893608,
       0.3310685098648071,
       0.3239920284807682,
       0.3173277738412221,
       0.31247236766616504,
       0.30721893773476283],
      [0.6790833333333334,
       0.76885,
       0.7992166666666667,
       0.81935,
       0.8323833333333334,
       0.8418666666666667,
       0.8499,
       0.8552333333333333,
       0.8609666666666667,
       0.8653833333333333,
       0.8689333333333333,
       0.8729,
       0.8757166666666667,
       0.8782,
       0.8804666666666666,
       0.8833833333333333,
       0.88575,
       0.8875333333333333,
       0.8900333333333333,
       0.8899833333333333])
```

3) Best Model w/ Dropout

```
DropLeNetChan = nn.Sequential(

            nn.Conv2d(1, 32, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.3),
            nn.Conv2d(32, 32, kernel_size=5), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.3),
            nn.Flatten(),
            nn.Linear(800, 256), nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 64), nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 10)
)
```

```
summary(DropLeNetChan, input_size = (1, 28, 28), batch_size =32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [32, 32, 28, 28]             832
              ReLU-2          [32, 32, 28, 28]               0
         MaxPool2d-3          [32, 32, 14, 14]               0
           Dropout-4          [32, 32, 14, 14]               0
            Conv2d-5          [32, 32, 10, 10]          25,632
              ReLU-6          [32, 32, 10, 10]               0
         MaxPool2d-7            [32, 32, 5, 5]               0
           Dropout-8            [32, 32, 5, 5]               0
           Flatten-9                 [32, 800]               0
           Linear-10                 [32, 256]         205,056
             ReLU-11                 [32, 256]               0
          Dropout-12                 [32, 256]               0
           Linear-13                  [32, 64]          16,448
             ReLU-14                  [32, 64]               0
          Dropout-15                  [32, 64]               0
           Linear-16                  [32, 10]             650
================================================================
Total params: 248,618
Trainable params: 248,618
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.10
Forward/backward pass size (MB): 17.70
Params size (MB): 0.95
Estimated Total Size (MB): 18.74
----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(DropLeNetChan, (1, 28, 28), as_strings=True,
                                         print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

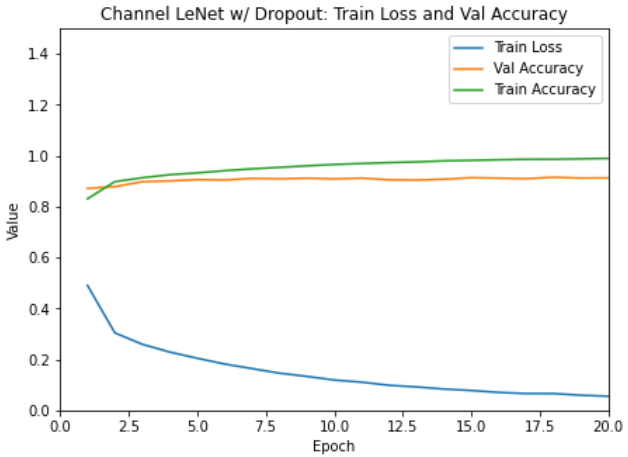
```
Warning: module Dropout is treated as a zero-op.
Warning: module Flatten is treated as a zero-op.
Sequential(
  248.62 k, 100.000% Params, 3.49 MMac, 100.000% MACs,
  (0): Conv2d(832, 0.335% Params, 652.29 KMac, 18.666% MACs, 1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU(0, 0.000% Params, 25.09 KMac, 0.718% MACs, )
  (2): MaxPool2d(0, 0.000% Params, 25.09 KMac, 0.718% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.3, inplace=False)
  (4): Conv2d(25.63 k, 10.310% Params, 2.56 MMac, 73.349% MACs, 32, 32, kernel_size=(5, 5), stride=(1, 1))
  (5): ReLU(0, 0.000% Params, 3.2 KMac, 0.092% MACs, )
  (6): MaxPool2d(0, 0.000% Params, 3.2 KMac, 0.092% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (7): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.3, inplace=False)
  (8): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
  (9): Linear(205.06 k, 82.478% Params, 205.06 KMac, 5.868% MACs, in_features=800, out_features=256, bias=True)
  (10): ReLU(0, 0.000% Params, 256.0 Mac, 0.007% MACs, )
  (11): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.3, inplace=False)
  (12): Linear(16.45 k, 6.616% Params, 16.45 KMac, 0.471% MACs, in_features=256, out_features=64, bias=True)
  (13): ReLU(0, 0.000% Params, 64.0 Mac, 0.002% MACs, )
  (14): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.3, inplace=False)
  (15): Linear(650, 0.261% Params, 650.0 Mac, 0.019% MACs, in_features=64, out_features=10, bias=True)
```

```
    )
    Computational complexity:  3.49 MMac
    Number of parameters:  248.62 k


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


# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

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

```
    loss: 0.2781029939651489
    Validation Accuracy: 91.16     Validation Loss:  0.28653141904991275

    Epoch 10
    loss: 0.045927003026008606
    loss: 0.07794604450464249
    Validation Accuracy: 90.89     Validation Loss:  0.2941940748117888

    Epoch 11
    loss: 0.10345877707004547
    loss: 0.11044761538505554
    Validation Accuracy: 91.17     Validation Loss:  0.2977535161234367

    Epoch 12
    loss: 0.1692483127117157
    loss: 0.38319969177246094
    Validation Accuracy: 90.51     Validation Loss:  0.3777781583550282

    Epoch 13
    loss: 0.0576220266520977
    loss: 0.02092619426548481
    Validation Accuracy: 90.42     Validation Loss:  0.3792904548974821

    Epoch 14
    loss: 0.006482385098934174
    loss: 0.07958652079105377
    Validation Accuracy: 90.74     Validation Loss:  0.3468707030841384

    Epoch 15
    loss: 0.09907671809196472
    loss: 0.2174682468175888
    Validation Accuracy: 91.34     Validation Loss:  0.3576213141067174

    Epoch 16
    loss: 0.0863342434167862
    loss: 0.029391279444098473
```

```
Validation Accuracy: 91.18     Validation Loss:  0.35718619907661653

Epoch 17
loss: 0.018787117674946785
loss: 0.030273647978901863
Validation Accuracy: 90.96     Validation Loss:  0.40946935341892815

Epoch 18
loss: 0.09341385960578918
loss: 0.015946049243211746
Validation Accuracy: 91.51     Validation Loss:  0.41649855165400157

Epoch 19
loss: 0.01702222228050232
loss: 0.015503406524658203
Validation Accuracy: 91.19     Validation Loss:  0.44161497947191614

Epoch 20
loss: 0.15391159057617188
loss: 0.001051229308359325
Validation Accuracy: 91.25999999999999     Validation Loss:  0.44329784055479976
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("Channel LeNet w/ Dropout: 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()
```



4) AlexNet

```
# First analyze AlexNet's complexity
AlexNet = nn.Sequential(

            nn.Conv2d(1, 96, kernel_size=11, stride=4, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 384, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(384, 384, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Flatten(),
            nn.Linear(6400, 4096), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, 10)

)
```

```
summary(AlexNet, input_size = (1, 224, 224), batch_size =32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [32, 96, 54, 54]          11,712
              ReLU-2          [32, 96, 54, 54]               0
         MaxPool2d-3          [32, 96, 26, 26]               0
            Conv2d-4         [32, 256, 26, 26]         614,656
              ReLU-5         [32, 256, 26, 26]               0
         MaxPool2d-6         [32, 256, 12, 12]               0
            Conv2d-7         [32, 384, 12, 12]         885,120
              ReLU-8         [32, 384, 12, 12]               0
            Conv2d-9         [32, 384, 12, 12]       1,327,488
             ReLU-10         [32, 384, 12, 12]               0
           Conv2d-11         [32, 256, 12, 12]         884,992
             ReLU-12         [32, 256, 12, 12]               0
        MaxPool2d-13           [32, 256, 5, 5]               0
          Flatten-14                [32, 6400]               0
           Linear-15                [32, 4096]      26,218,496
             ReLU-16                [32, 4096]               0
          Dropout-17                [32, 4096]               0
           Linear-18                [32, 4096]      16,781,312
             ReLU-19                [32, 4096]               0
          Dropout-20                [32, 4096]               0
           Linear-21                  [32, 10]          40,970
================================================================
Total params: 46,764,746
Trainable params: 46,764,746
Non-trainable params: 0
----------------------------------------------------------------
```

```
      Input size (MB): 6.12
      Forward/backward pass size (MB): 327.16
      Params size (MB): 178.39
      Estimated Total Size (MB): 511.68
      ----------------------------------------------------------------


macs, params = get_model_complexity_info(AlexNet, (1, 224, 224), as_strings=True,
                                         print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)

      Warning: variables __flops__ or __params__ are already defined for the moduleConv2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleMaxPool2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleConv2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleMaxPool2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleConv2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleConv2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleConv2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleMaxPool2d ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleReLU ptflops can affect your code!
      Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflops can affect your code!
      Warning: module Flatten is treated as a zero-op.
      Warning: module Dropout is treated as a zero-op.
      Sequential(
        46.76 M, 100.000% Params, 939.85 MMac, 100.000% MACs,
        (0): Conv2d(11.71 k, 0.025% Params, 34.15 MMac, 3.634% MACs, 1, 96, kernel_size=(11, 11), stride=(4, 4), padding=(1, 1))
        (1): ReLU(0, 0.000% Params, 279.94 KMac, 0.030% MACs, )
        (2): MaxPool2d(0, 0.000% Params, 279.94 KMac, 0.030% MACs, kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
        (3): Conv2d(614.66 k, 1.314% Params, 415.51 MMac, 44.210% MACs, 96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
        (4): ReLU(0, 0.000% Params, 173.06 KMac, 0.018% MACs, )
        (5): MaxPool2d(0, 0.000% Params, 173.06 KMac, 0.018% MACs, kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
        (6): Conv2d(885.12 k, 1.893% Params, 127.46 MMac, 13.561% MACs, 256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (7): ReLU(0, 0.000% Params, 55.3 KMac, 0.006% MACs, )
        (8): Conv2d(1.33 M, 2.839% Params, 191.16 MMac, 20.339% MACs, 384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (9): ReLU(0, 0.000% Params, 55.3 KMac, 0.006% MACs, )
        (10): Conv2d(884.99 k, 1.892% Params, 127.44 MMac, 13.559% MACs, 384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (11): ReLU(0, 0.000% Params, 36.86 KMac, 0.004% MACs, )
        (12): MaxPool2d(0, 0.000% Params, 36.86 KMac, 0.004% MACs, kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
        (13): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
        (14): Linear(26.22 M, 56.065% Params, 26.22 MMac, 2.790% MACs, in_features=6400, out_features=4096, bias=True)
        (15): ReLU(0, 0.000% Params, 4.1 KMac, 0.000% MACs, )
        (16): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
        (17): Linear(16.78 M, 35.885% Params, 16.78 MMac, 1.786% MACs, in_features=4096, out_features=4096, bias=True)
        (18): ReLU(0, 0.000% Params, 4.1 KMac, 0.000% MACs, )
        (19): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
        (20): Linear(40.97 k, 0.088% Params, 40.97 KMac, 0.004% MACs, in_features=4096, out_features=10, bias=True)
      )
      Computational complexity:  939.85 MMac
      Number of parameters:  46.76 M
```

Simplified AlexNet

```
# Simplifying AlexNet
SimpAlexNet = nn.Sequential(

            nn.Conv2d(1, 32, kernel_size=3, stride=2, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(32, 64, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(128, 96, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.Linear(96, 64), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(32, 10)

)
```

```
summary(SimpAlexNet, input_size = (1, 28, 28), batch_size =32)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [32, 32, 14, 14]             320
              ReLU-2          [32, 32, 14, 14]               0
         MaxPool2d-3            [32, 32, 6, 6]               0
            Conv2d-4            [32, 64, 6, 6]          51,264
              ReLU-5            [32, 64, 6, 6]               0
         MaxPool2d-6            [32, 64, 2, 2]               0
            Conv2d-7           [32, 128, 2, 2]          73,856
              ReLU-8           [32, 128, 2, 2]               0
            Conv2d-9           [32, 128, 2, 2]         147,584
             ReLU-10           [32, 128, 2, 2]               0
           Conv2d-11            [32, 96, 2, 2]         110,688
             ReLU-12            [32, 96, 2, 2]               0
        MaxPool2d-13            [32, 96, 1, 1]               0
          Flatten-14                  [32, 96]               0
           Linear-15                  [32, 64]           6,208
             ReLU-16                  [32, 64]               0
          Dropout-17                  [32, 64]               0
           Linear-18                  [32, 32]           2,080
             ReLU-19                  [32, 32]               0
          Dropout-20                  [32, 32]               0
           Linear-21                  [32, 10]             330
================================================================
Total params: 392,330
Trainable params: 392,330
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.10
```

```
     Forward/backward pass size (MB): 5.34
     Params size (MB): 1.50
     Estimated Total Size (MB): 6.93
     ----------------------------------------------------------------
```

```
macs, params = get_model_complexity_info(SimpAlexNet, (1, 28, 28), as_strings=True,
                                         print_per_layer_stat=True, verbose=True)
print('Computational complexity: ', macs)
print('Number of parameters: ', params)
```

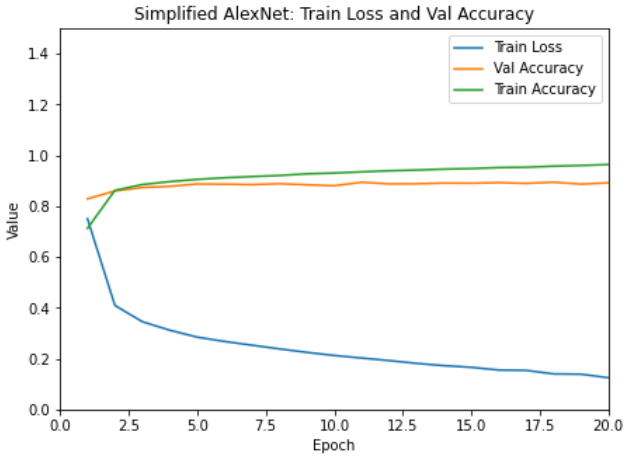
```
     Warning: module Flatten is treated as a zero-op.
     Warning: module Dropout is treated as a zero-op.
     Sequential(
       392.33 k, 100.000% Params, 3.26 MMac, 100.000% MACs,
       (0): Conv2d(320, 0.082% Params, 62.72 KMac, 1.921% MACs, 1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
       (1): ReLU(0, 0.000% Params, 6.27 KMac, 0.192% MACs, )
       (2): MaxPool2d(0, 0.000% Params, 6.27 KMac, 0.192% MACs, kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
       (3): Conv2d(51.26 k, 13.067% Params, 1.85 MMac, 56.534% MACs, 32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
       (4): ReLU(0, 0.000% Params, 2.3 KMac, 0.071% MACs, )
       (5): MaxPool2d(0, 0.000% Params, 2.3 KMac, 0.071% MACs, kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
       (6): Conv2d(73.86 k, 18.825% Params, 295.42 KMac, 9.050% MACs, 64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (7): ReLU(0, 0.000% Params, 512.0 Mac, 0.016% MACs, )
       (8): Conv2d(147.58 k, 37.617% Params, 590.34 KMac, 18.084% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (9): ReLU(0, 0.000% Params, 512.0 Mac, 0.016% MACs, )
       (10): Conv2d(110.69 k, 28.213% Params, 442.75 KMac, 13.563% MACs, 128, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (11): ReLU(0, 0.000% Params, 384.0 Mac, 0.012% MACs, )
       (12): MaxPool2d(0, 0.000% Params, 384.0 Mac, 0.012% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (13): Flatten(0, 0.000% Params, 0.0 Mac, 0.000% MACs, start_dim=1, end_dim=-1)
       (14): Linear(6.21 k, 1.582% Params, 6.21 KMac, 0.190% MACs, in_features=96, out_features=64, bias=True)
       (15): ReLU(0, 0.000% Params, 64.0 Mac, 0.002% MACs, )
       (16): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
       (17): Linear(2.08 k, 0.530% Params, 2.08 KMac, 0.064% MACs, in_features=64, out_features=32, bias=True)
       (18): ReLU(0, 0.000% Params, 32.0 Mac, 0.001% MACs, )
       (19): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
       (20): Linear(330, 0.084% Params, 330.0 Mac, 0.010% MACs, in_features=32, out_features=10, bias=True)
     )
     Computational complexity:  3.26 MMac
     Number of parameters:  392.33 k
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(SimpAlexNet.parameters(), lr=0.001)
```

```
# Begin training over 20 epochs
epochs = 20
valHist = []
valLosses = []
trainHist = []
trainLosses = []
interLosses = []
avgLosses = []

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

```
loss: 0.18720115721225739
Validation Accuracy: 88.39     Validation Loss:  0.35834061316312693

Epoch 10
loss: 0.44304436445236206
loss: 0.1485912948846817
Validation Accuracy: 88.03     Validation Loss:  0.35558546795108065

Epoch 11
loss: 0.07590296864509583
loss: 0.24297630786895752
Validation Accuracy: 89.39     Validation Loss:  0.30821049189605654

Epoch 12
loss: 0.3486815392971039
loss: 0.4212316870689392
Validation Accuracy: 88.75999999999999     Validation Loss:  0.3630248645719248

Epoch 13
loss: 0.15323634445667267
loss: 0.09813195466995239
Validation Accuracy: 88.81     Validation Loss:  0.33460332328876174

Epoch 14
loss: 0.2781636714935303
loss: 0.5736841559410095
Validation Accuracy: 89.08     Validation Loss:  0.3559204632826983

Epoch 15
loss: 0.32782769203186035
loss: 0.16226975619792938
Validation Accuracy: 89.03     Validation Loss:  0.33801869034898074

Epoch 16
loss: 0.18509599566459656
loss: 0.09188417345285416
Validation Accuracy: 89.24     Validation Loss:  0.3793918632637388

Epoch 17
loss: 0.13456179201602936
loss: 0.08670203387737274
Validation Accuracy: 88.97     Validation Loss:  0.40783253519203716

Epoch 18
loss: 0.16297142207622528
loss: 0.2289642095565796
Validation Accuracy: 89.41     Validation Loss:  0.3860520369733294

Epoch 19
loss: 0.07365482300519943
loss: 0.13822397589683533
Validation Accuracy: 88.67     Validation Loss:  0.4406382368859677

Epoch 20
loss: 0.05559298396110535
loss: 0.12641765177249908
Validation Accuracy: 89.17     Validation Loss:  0.3869175660593536
```

```
# Plot results
x = np.linspace(1, epochs, epochs)
plt.figure(figsize=(7,5))
plt.title("Simplified AlexNet: 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()
```



✓ 0s completed at 4:17 PM