In [1]:
```
import torch
from torch import nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets
import torchvision.transforms as transforms
from torchvision.transforms import ToTensor
import time
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch.nn.functional as F
import gc
from ptflops import get_model_complexity_info
from torchmetrics.classification import MulticlassConfusionMatrix
```

Problem 1

In [2]:
```
ata-unversioned/ecgr4106/'
s.CIFAR10(data_path, train=True, download=True, transform=transforms.Compose([transforms.ToTensor(), transforms.Resize(size=
asets.CIFAR10(data_path, train=False, download=True, transform=transforms.Compose([transforms.ToTensor(), transforms.Resize(
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [3]:
```
def try_gpu(i=0):
    if torch.cuda.device_count() >= i+1:
        return torch.device(f'cuda:{i}')
    return torch.device('cpu')
```

In [4]:

```
def training_loop(n_epochs, optimizer, model, loss_fn, train_loader, val_loader, update_freq):
    train_loss_hist = []
    train_acc_hist = []
    val_acc_hist = []
    main_tic = time.perf_counter()

    for epoch in range(1, n_epochs + 1):
        tic = time.perf_counter()
        loss_train = 0.0
        correct_train = 0
        correct_val = 0
        model_argmax = []
        labels_argmax = []

        for imgs, lbls in train_loader:
            images = imgs.to(device=try_gpu())
            labels = lbls.to(device=try_gpu())
            outputs = model(images)
            del images
            loss = loss_fn(outputs, labels)
            del labels
            del outputs
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss_train += loss.item()
            gc.collect()
            torch.cuda.empty_cache()

        toc = time.perf_counter()

        with torch.no_grad():
            total = 0
            for imgs, lbls in train_loader:
                images = imgs.to(device=try_gpu())
                labels = lbls.to(device=try_gpu())
                outputs = model(images)
                del images
                _, predicted = torch.max(outputs, dim=1)
                del outputs
                total += labels.shape[0]
                correct_train += int((predicted == labels).sum())
                del labels
                del predicted
            train_acc = round(correct_train/total, 3)
            total = 0
            for imgs, lbls in val_loader:
                images = imgs.to(device=try_gpu())
                labels = lbls.to(device=try_gpu())
                outputs = model(images)
                del images
                _, predicted = torch.max(outputs, dim=1)
                del outputs
                if epoch == 1 or epoch == n_epochs or epoch % update_freq == 0:
                    model_argmax = model_argmax + predicted.tolist()
                    labels_argmax = labels_argmax + labels.tolist()
                total += labels.shape[0]
                correct_val += int((predicted == labels).sum())
                del labels
                del predicted
            val_acc = round(correct_val/total, 3)

        train_loss_hist.append(round(loss_train / len(train_loader), 5))
        train_acc_hist.append(train_acc)
        val_acc_hist.append(val_acc)
        label_set = set(labels_argmax)

        if epoch == 1 or epoch == n_epochs or epoch % update_freq == 0:
            print(f"Epoch {epoch}:\n\tDuration = {round(toc - tic, 3)} seconds\n\tTraining Loss: {train_loss_hist[-1]}\n\tT
            metric = MulticlassConfusionMatrix(num_classes=len(label_set))
            print(metric(torch.ByteTensor(model_argmax), torch.ByteTensor(labels_argmax)))

    main_toc = time.perf_counter()
    print(f"\nTotal Training Time = {round(main_toc - main_tic, 3)} seconds\nAverage Training Time per Epoch (including val
    return train_loss_hist, train_acc_hist, val_acc_hist
```

In [5]:
```
def plot_model(title, loss_hist, train_hist, test_hist, leg_loc):
    fig, ax1 = plt.subplots()
    x = range(1, len(loss_hist)+1)
    ax1.plot(x, loss_hist, color='k')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Error')
    ax1.tick_params(axis='y')

    ax2 = ax1.twinx()
    ax2.set_ylabel('Accuracy')
    ax2.plot(x, train_hist)
    ax2.plot(x, test_hist)
    ax2.set_ylim([0, 1])
    ax1.tick_params(axis='y')

    fig.legend(["Training Loss", "Training Accuracy", "Testing Accuracy"], loc=leg_loc, bbox_to_anchor=(1, 1), bbox_transfo
    plt.title(title)
```

In [6]:
```
def vgg_block(num_convs, out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(nn.LazyConv2d(out_channels, 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)
```

In [82]:
```
def build_vgg(arch, num_classes=10):
    conv_blks = []
    for (num_convs, out_channels) in arch:
        conv_blks.append(vgg_block(num_convs, out_channels))
    model = nn.Sequential(
        *conv_blks, nn.Flatten(),
        nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
        nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(0.5),
        nn.LazyLinear(num_classes))
    return model
```

In [ ]:
```
train_loader_1 = DataLoader(cifar10, batch_size=64, shuffle=True)
val_loader_1 = DataLoader(cifar10_val, batch_size=64, shuffle=False)
```

In [83]:
```
class tinyVGG(nn.Module):
    def __init__(self, arch, num_classes=10):
        super(tinyVGG, self).__init__()
        conv_blks = []
        for (num_convs, out_channels) in arch:
            conv_blks.append(vgg_block(num_convs, out_channels))
        self.conv_blks = nn.Sequential(*conv_blks, nn.Flatten())
        self.fc1 = nn.LazyLinear(128)
        self.fc2 = nn.LazyLinear(64)
        self.fc3 = nn.LazyLinear(num_classes)
        self.fc_drop = nn.Dropout(p=0.5)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.conv_blks(x)
        out = self.fc_drop(self.relu(self.fc1(out)))
        out = self.fc_drop(self.relu(self.fc2(out)))
        out = self.fc3(out)
        return out

model_0 = tinyVGG(arch=((1, 64), (1, 128))).to(device=try_gpu())
optimizer_0 = optim.SGD(model_0.parameters(), lr=0.1)
model_0.eval()
```

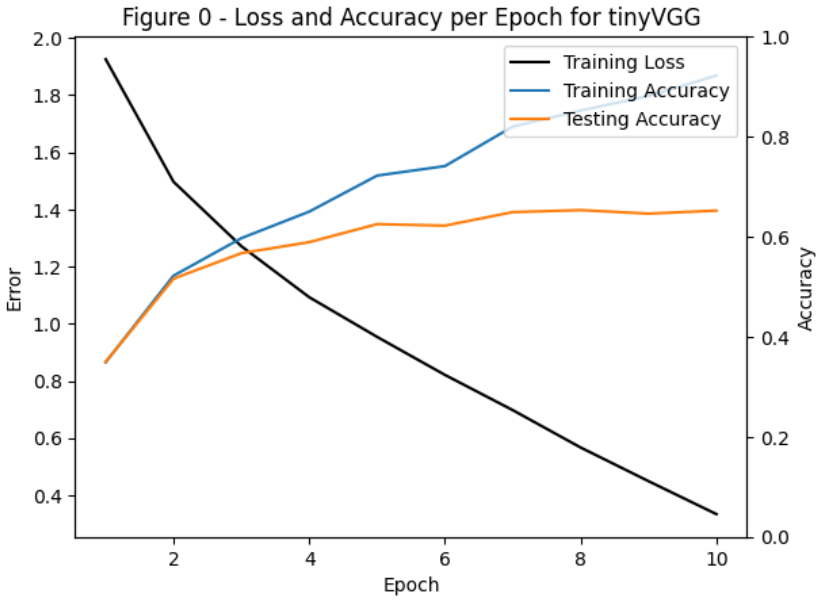
Out[83]:
```
tinyVGG(
  (vgg): Sequential(
    (0): Sequential(
      (0): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Flatten(start_dim=1, end_dim=-1)
    (3): LazyLinear(in_features=0, out_features=4096, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.5, inplace=False)
    (6): LazyLinear(in_features=0, out_features=4096, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.5, inplace=False)
    (9): LazyLinear(in_features=0, out_features=10, bias=True)
  )
)
```

In [ ]:
```
with torch.cuda.device(0):
    macs, params = get_model_complexity_info(model_1, (1, 3, 64, 64), as_strings=True,
                                             print_per_layer_stat=False, verbose=False)
```

In [9]:
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_0, t_acc_hist_0, v_acc_hist_0= training_loop(10,
                                                         optimizer_0,
                                                         model_0,
                                                         nn.CrossEntropyLoss(),
                                                         train_loader_1,
                                                         val_loader_1,
                                                         1)
```

```
        [ 10,  17,  40,  49,  71,  37, 720,  15,  13,  28],
        [ 20,  23,  38,  21,  59,  50,   8, 717,   5,  59],
        [ 92, 119,   8,   5,   6,   7,   5,   5, 683,  70],
        [ 12, 186,   6,   5,   3,   6,   6,  11,  17, 748]])
Epoch 9:
        Duration = 163.986 seconds
        Training Loss: 0.44921
        Training Accuracy: 0.881
        Validation Accuracy: 0.646
tensor([[766,  15,  67,   8,   9,  23,   5,  16,  24,  67],
        [ 29, 675,  28,  11,   4,  13,  10,   5,  18, 207],
        [ 78,   6, 647,  36,  61,  86,  39,  27,   7,  13],
        [ 22,   8, 149, 334,  33, 331,  66,  30,  15,  12],
        [ 28,   2, 186,  44, 513,  75,  75,  70,   4,   3],
        [ 12,   3, 110,  74,  38, 696,  25,  33,   4,   5],
        [  8,   2,  91,  47,  43,  82, 703,   8,   4,  12],
        [ 15,   4,  89,  31,  49,  99,  11, 681,   4,  17],
        [168,  32,  31,  14,   7,  17,   7,   6, 665,  53],
        [ 37,  64,  22,  15,   6,  34,   5,  16,  20, 781]])
Epoch 10:
```

In [19]:
```
title_0 = "Figure 0 - Loss and Accuracy per Epoch for tinyVGG"
plot_model(title_0, t_loss_hist_0, t_acc_hist_0, v_acc_hist_0, 'upper right')
```



In [84]:
```
class VGG_11(nn.Module):
    def __init__(self, arch, num_classes=10):
        super(VGG_11, self).__init__()
        self.vgg = build_vgg(arch)

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

model_1 = VGG_11(arch=((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))).to(device=try_gpu())
optimizer_1 = optim.SGD(model_1.parameters(), lr=0.1)
model_1.eval()
```

Out[84]:
```
VGG_11(
  (vgg): Sequential(
    (0): Sequential(
      (0): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (4): Sequential(
      (0): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (5): Flatten(start_dim=1, end_dim=-1)
    (6): LazyLinear(in_features=0, out_features=4096, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.5, inplace=False)
    (9): LazyLinear(in_features=0, out_features=4096, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.5, inplace=False)
    (12): LazyLinear(in_features=0, out_features=10, bias=True)
  )
)
```

In [85]:

```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_1, t_acc_hist_1, v_acc_hist_1 = training_loop(10,
                                                          optimizer_1,
                                                          model_1,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          2)
```

```
Epoch 1:
        Duration = 306.019 seconds
        Training Loss: 2.30314
        Training Accuracy: 0.1
        Validation Accuracy: 0.1
tensor([[   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0],
        [   0,    0,    0, 1000,    0,    0,    0,    0,    0,    0]])

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11660\3450204939.py in <module>
      1 torch.cuda.empty_cache()
      2 gc.collect()
----> 3 t_loss_hist_1, t_acc_hist_1, v_acc_hist_1 = training_loop(10,
      4                                                           optimizer_1,
      5                                                           model_1,

~\AppData\Local\Temp\ipykernel_11660\310723436.py in training_loop(n_epochs, optimizer, model, loss_fn, train_loader, val
_loader, update_freq)
     24             loss.backward()
     25             optimizer.step()
---> 26             loss_train += loss.item()
     27             gc.collect()
     28             torch.cuda.empty_cache()

KeyboardInterrupt:
```

In [105]:

```
torch.cuda.empty_cache()
gc.collect()
```

Out[105]: 0

In [ ]:

```
title_1 = "Figure 1 - Loss and Accuracy per Epoch for VGG-11"
plot_model(title_1, t_loss_hist_1, t_acc_hist_1, v_acc_hist_1, 'upper right')
```

In [8]:
```
class VGG_16(nn.Module):
    def __init__(self, arch, num_classes=10):
        super(VGG_16, self).__init__()
        self.vgg = build_VGG(arch)

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

model_2 = VGG_16(arch=((2, 64), (2, 128), (3, 256), (3, 512), (3, 512))).to(device=try_gpu())
optimizer_2 = optim.SGD(model_2.parameters(), lr=0.1)
model_2.eval()
```

```
C:\Users\ccm51\anaconda3\lib\site-packages\torch\nn\modules\lazy.py:180: UserWarning: Lazy modules are a new feature unde
r heavy development so changes to the API or functionality can happen at any moment.
  warnings.warn('Lazy modules are a new feature under heavy development '
```

Out[8]:
```
VGG_16(
  (conv_blks): Sequential(
    (0): Sequential(
      (0): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU()
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU()
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (4): Sequential(
      (0): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU()
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (fc1): LazyLinear(in_features=0, out_features=4096, bias=True)
  (fc2): LazyLinear(in_features=0, out_features=4096, bias=True)
  (fc3): LazyLinear(in_features=0, out_features=10, bias=True)
  (fc_drop): Dropout(p=0.5, inplace=False)
  (relu): ReLU()
)
```

In [ ]:
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_2, t_acc_hist_2, v_acc_hist_2 = training_loop(3,
                                                          optimizer_2,
                                                          model_2,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          1)
```

In [ ]:
```
title_2 = "Figure 2 - Loss and Accuracy per Epoch for VGG-16"
plot_model(title_2, t_loss_hist_2, t_acc_hist_2, v_acc_hist_2, 'upper right')
```

In [54]:

```
class VGG_19(nn.Module):
    def __init__(self, arch, num_classes=10):
        super(VGG_19, self).__init__()
        conv_blks = []
        for (num_convs, out_channels) in arch:
            conv_blks.append(vgg_block(num_convs, out_channels))
        self.conv_blks = nn.Sequential(*conv_blks, nn.Flatten())
        self.fc1 = nn.LazyLinear(4096)
        self.fc2 = nn.LazyLinear(4096)
        self.fc3 = nn.LazyLinear(num_classes)
        self.fc_drop = nn.Dropout(p=0.5)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.conv_blks(x)
        out = self.fc_drop(self.relu(self.fc1(out)))
        out = self.fc_drop(self.relu(self.fc2(out)))
        out = self.fc3(out)
        return out

model_3 = VGG_19(arch=((2, 64), (2, 128), (4, 256), (4, 512), (4, 512))).to(device=try_gpu())
optimizer_3 = optim.SGD(model_3.parameters(), lr=0.1)
model_3.eval()
```

Out[54]:
```
VGG_19(
  (conv_blks): Sequential(
    (0): Sequential(
      (0): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU()
      (9): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (10): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (11): ReLU()
      (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU()
      (9): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (10): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (11): ReLU()
      (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (4): Sequential(
      (0): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU()
      (6): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (8): ReLU()
      (9): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (10): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (11): ReLU()
      (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (5): Flatten(start_dim=1, end_dim=-1)
  )
  (fc1): LazyLinear(in_features=0, out_features=4096, bias=True)
  (fc2): LazyLinear(in_features=0, out_features=4096, bias=True)
  (fc3): LazyLinear(in_features=0, out_features=10, bias=True)
  (fc_drop): Dropout(p=0.5, inplace=False)
  (relu): ReLU()
)
```

In [ ]:
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_3, t_acc_hist_3, v_acc_hist_3 = training_loop(10,
                                                          optimizer_3,
                                                          model_3,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          2)
```

In [ ]:
```
title_3 = "Figure 3 - Loss and Accuracy per Epoch for VGG-19"
plot_model(title_3, t_loss_hist_3, t_acc_hist_3, v_acc_hist_3, 'upper right')
```

Problem 2

In [11]:
```
class Inception(nn.Module):
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)
        self.b1_1 = nn.LazyConv2d(c1, kernel_size=1)
        self.b2_1 = nn.LazyConv2d(c2[0], kernel_size=1)
        self.b2_2 = nn.LazyConv2d(c2[1], kernel_size=3, padding=1)
        self.b3_1 = nn.LazyConv2d(c3[0], kernel_size=1)
        self.b3_2 = nn.LazyConv2d(c3[1], kernel_size=5, padding=2)
        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)
```

In [12]:

```
class GoogLeNet(nn.Module):
    def __init__(self, num_classes=10):
        super(GoogLeNet, self).__init__()
        self.stem = nn.Sequential(nn.LazyConv2d(64, kernel_size=7, stride=2, padding=3),
                                  nn.ReLU(),
                                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
                                  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))
        self.body1 = 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))
        self.body2 = nn.Sequential(Inception(192, (96, 208), (16, 48), 64),
                                   Inception(160, (112, 124), (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))
        self.body3 = nn.Sequential(Inception(256, (160, 320), (32, 128), 128),
                                   Inception(384, (192, 384), (48, 128), 128),
                                   nn.AdaptiveAvgPool2d((1,1)),
                                   nn.Flatten())
        self.fc = nn.LazyLinear(num_classes)


    def forward(self, x):
        out = self.stem(x)
        out = self.body1(out)
        out = self.body2(out)
        out = self.body3(out)
        out = self.fc(out)
        return out

model_4 = GoogLeNet().to(device=try_gpu())
optimizer_4 = optim.SGD(model_4.parameters(), lr=0.1)
model_4.eval()
```

Out[12]:
```
GoogLeNet(
  (stem): Sequential(
    (0): LazyConv2d(0, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (3): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
    (4): ReLU()
    (5): LazyConv2d(0, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (body1): Sequential(
    (0): Inception(
      (b1_1): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 96, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 16, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 32, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): Inception(
      (b1_1): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 32, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 96, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
    )
    (2): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (body2): Sequential(
    (0): Inception(
      (b1_1): LazyConv2d(0, 192, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 96, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 208, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 16, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 48, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): Inception(
      (b1_1): LazyConv2d(0, 160, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 112, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 124, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 24, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
    )
    (2): Inception(
      (b1_1): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 24, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
    )
    (3): Inception(
      (b1_1): LazyConv2d(0, 112, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 144, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 288, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 32, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 64, kernel_size=(1, 1), stride=(1, 1))
    )
    (4): Inception(
      (b1_1): LazyConv2d(0, 256, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 160, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 32, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
    )
    (5): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (body3): Sequential(
    (0): Inception(
      (b1_1): LazyConv2d(0, 256, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 160, kernel_size=(1, 1), stride=(1, 1))
```

```
      (b2_2): LazyConv2d(0, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 32, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): Inception(
      (b1_1): LazyConv2d(0, 384, kernel_size=(1, 1), stride=(1, 1))
      (b2_1): LazyConv2d(0, 192, kernel_size=(1, 1), stride=(1, 1))
      (b2_2): LazyConv2d(0, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (b3_1): LazyConv2d(0, 48, kernel_size=(1, 1), stride=(1, 1))
      (b3_2): LazyConv2d(0, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
      (b4_1): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
      (b4_2): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
    )
    (2): AdaptiveAvgPool2d(output_size=(1, 1))
    (3): Flatten(start_dim=1, end_dim=-1)
  )
  (fc): LazyLinear(in_features=0, out_features=10, bias=True)
)
```

In [13]:

```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_4, t_acc_hist_4, v_acc_hist_4 = training_loop(3,
                                                        optimizer_4,
                                                        model_4,
                                                        nn.CrossEntropyLoss(),
                                                        train_loader_1,
                                                        val_loader_1,
                                                        1)
```

```
Epoch 1:
        Duration = 243.455 seconds
        Training Loss: 2.30299
        Training Accuracy: 0.1
        Validation Accuracy: 0.1
tensor([[   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0],
        [   0,    0,    0,    0,    0, 1000,    0,    0,    0,    0]])
Epoch 2:
        Duration = 264.773 seconds
        Training Loss: 2.30299
        Training Accuracy: 0.1
        Validation Accuracy: 0.1
tensor([[   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0],
        [   0,    0, 1000,    0,    0,    0,    0,    0,    0,    0]])
Epoch 3:
        Duration = 266.93 seconds
        Training Loss: 2.30306
        Training Accuracy: 0.1
        Validation Accuracy: 0.1
tensor([[   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0],
        [   0,    0,    0,    0, 1000,    0,    0,    0,    0,    0]])

Total Training Time = 1035.319 seconds
Average Training Time per Epoch (including validation) = 345.106 seconds
```

In [ ]:
```
title_4 = "Figure 4 - Loss and Accuracy per Epoch for GoogLeNet"
plot_model(title_4, t_loss_hist_4, t_acc_hist_4, v_acc_hist_4, 'upper right')
```

In [ ]:
```
class AltGoogLeNet(nn.Module):
    def __init__(self, num_classes=10):
        super(GoogLeNet, self).__init__()
        self.stem = 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),
                                  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))
        self.body1 = nn.Sequential(Inception(64, (96, 128), (16, 32), 32),
                                  Inception(128, (128, 192), (32, 96), 64),
                                  nn.LazyBatchNorm2d(),
                                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
        self.body2 = nn.Sequential(Inception(192, (96, 208), (16, 48), 64),
                                  Inception(160, (112, 124), (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.LazyBatchNorm2d(),
                                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
        self.body3 = nn.Sequential(Inception(256, (160, 320), (32, 128), 128),
                                  Inception(384, (192, 384), (48, 128), 128),
                                  nn.LazyBatchNorm2d(),
                                  nn.AdaptiveAvgPool2d((1,1)),
                                  nn.Flatten())
        self.fc = nn.LazyLinear(num_classes)


    def forward(self, x):
        out = self.stem(x)
        out = self.body1(out)
        out = self.body2(out)
        out = self.body3(out)
        out = self.fc(out)
        return out

model_5 = GoogLeNet().to(device=try_gpu())
optimizer_5 = optim.SGD(model_5.parameters(), lr=0.1)
model_5.eval()
```

In [ ]:
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_5, t_acc_hist_5, v_acc_hist_5 = training_loop(5,
                                                          optimizer_5,
                                                          model_5,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          3)
```

In [ ]:
```
title_5 = "Figure 5 - Loss and Accuracy per Epoch for GoogLeNet with Batch Norm"
plot_model(title_5, t_loss_hist_5, t_acc_hist_5, v_acc_hist_5, 'upper right')
```

Problem 3

In [20]:
```
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.bn = nn.LazyBatchNorm2d()

    def forward(self, x):
        y = F.relu(self.bn(self.conv1(x)))
        y = self.bn(self.conv2(y))
        if self.conv3:
            x = self.bn(self.conv3(x))
        y += x
        return F.relu(y)
```

In [48]:
```
def block(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)
```

In [51]:

```
class ResNet(nn.Module):
    def __init__(self, arch, num_classes=10):
        super(ResNet, self).__init__()
        self.stem = 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))
        blks = []
        for i, b in enumerate(arch):
            blks.append(block(*b, first_block=(i==0)))
        self.blks = nn.Sequential(*blks)
        self.head = nn.Sequential(nn.AdaptiveAvgPool2d((1, 1)),
                                  nn.Flatten(),
                                  nn.LazyLinear(num_classes))

    def forward(self, x):
        out = self.stem(x)
        out = self.blks(out)
        out = self.head(out)
        return out

model_6 = ResNet(arch=((2, 64), (2, 128), (2, 256), (2,512))).to(device=try_gpu())
optimizer_6 = optim.SGD(model_6.parameters(), lr=0.01)
model_6.eval()
```

Out[51]:
```
ResNet(
  (stem): Sequential(
    (0): LazyConv2d(0, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
    (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (blks): Sequential(
    (0): Sequential(
      (0): Residual(
        (conv1): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): Residual(
        (conv1): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Sequential(
      (0): Residual(
        (conv1): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (conv2): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv3): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(2, 2))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): Residual(
        (conv1): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (2): Sequential(
      (0): Residual(
        (conv1): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (conv2): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv3): LazyConv2d(0, 256, kernel_size=(1, 1), stride=(2, 2))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): Residual(
        (conv1): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (3): Sequential(
      (0): Residual(
        (conv1): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (conv2): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv3): LazyConv2d(0, 512, kernel_size=(1, 1), stride=(2, 2))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): Residual(
        (conv1): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
  (head): Sequential(
    (0): AdaptiveAvgPool2d(output_size=(1, 1))
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): LazyLinear(in_features=0, out_features=10, bias=True)
  )
)
```

In [53]:

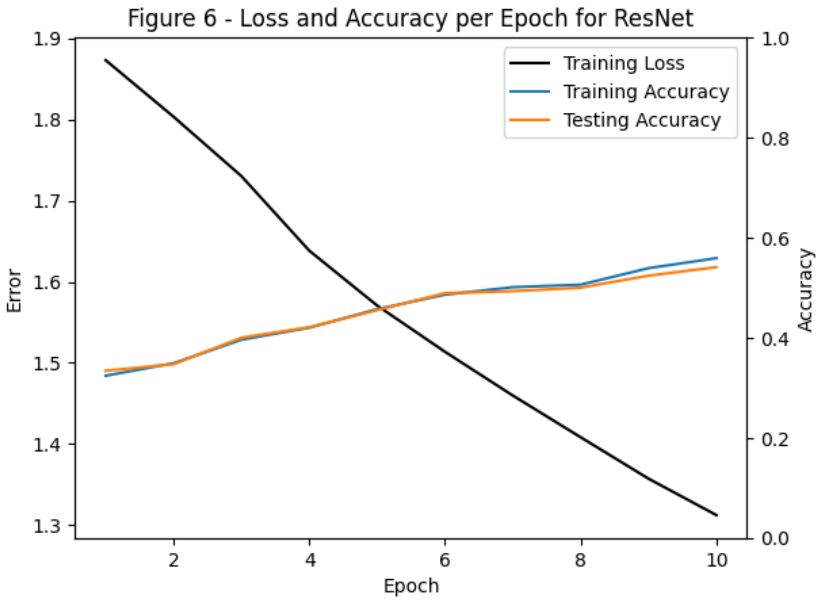
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_6, t_acc_hist_6, v_acc_hist_6 = training_loop(10,
                                                          optimizer_6,
                                                          model_6,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          2)
```

```
Epoch 1:
        Duration = 215.568 seconds
        Training Loss: 1.87298
        Training Accuracy: 0.324
        Validation Accuracy: 0.334
tensor([[480, 101,  26,  85,   7,  54,  26,  71,  94,  56],
        [ 97, 462,  10,  47,   3,  92,  22,  56, 101, 110],
        [ 75,  35,  78, 119, 193, 113, 245, 121,  10,  11],
        [ 17,  28,  23, 233,  53, 181, 307, 135,  12,  11],
        [ 42,  15,  23,  74, 345,  71, 299, 112,  11,   8],
        [ 15,  22,  25, 169,  64, 274, 291, 123,   8,   9],
        [  4,  10,  23, 102, 232,  63, 468,  92,   2,   4],
        [ 26,  34,  28,  86,  82,  80, 175, 456,   5,  28],
        [313, 128,  12,  74,   0,  63,  10,  47, 278,  75],
        [ 84, 161,  13,  68,   3,  68,  27, 196, 110, 270]])
Epoch 2:
        Duration = 237.357 seconds
        Training Loss: 1.80335
        Training Accuracy: 0.349
        Validation Accuracy: 0.347
tensor([[422,  87,   3,  25,   1,  40,   4,  24, 319,  75],
        [ 56, 466,   0,  10,   0,  35,   3,  10, 251, 169],
        [107,  64,  27,  88,  48, 282, 163, 115,  66,  40],
        [ 49,  53,   8, 215,   5, 378,  82,  68,  64,  78],
        [ 52,  28,   4,  82, 102, 244, 251, 134,  58,  45],
        [ 31,  57,   6, 142,  11, 463,  65,  87,  79,  59],
        [ 12,  58,   2, 140,  26, 276, 333,  86,  24,  43],
        [ 46,  78,   5,  93,  18, 172,  37, 351,  36, 164],
        [144,  86,   1,  18,   0,  30,   0,   9, 623,  89],
        [ 54, 150,   0,  25,   1,  31,   2,  25, 246, 466]])
Epoch 4:
        Duration = 248.569 seconds
        Training Loss: 1.6383
        Training Accuracy: 0.42
        Validation Accuracy: 0.421
tensor([[413,  94,  17,  35,   3,   5,  32,  22, 323,  56],
        [ 16, 653,   2,  31,   0,   5,  29,  10, 119, 135],
        [ 82,  48, 137, 178, 116,  30, 259,  79,  49,  22],
        [ 31,  46,  37, 430,  18,  45, 259,  39,  51,  44],
        [ 53,  29,  49, 151, 191,  20, 336, 109,  51,  11],
        [ 20,  40,  40, 390,  28, 142, 197,  66,  49,  28],
        [  7,  33,  18, 152,  25,   4, 691,  21,  22,  27],
        [ 39,  46,  24, 185,  39,  27, 104, 431,  32,  73],
        [ 81, 121,   7,  36,   1,   5,  11,  11, 671,  56],
        [ 27, 259,   6,  44,   0,   1,  36,  21, 153, 453]])
Epoch 6:
        Duration = 237.792 seconds
        Training Loss: 1.51366
        Training Accuracy: 0.486
        Validation Accuracy: 0.489
tensor([[526,  55,  68,  21,  16,  19,  16,  27, 175,  77],
        [ 36, 647,  11,  21,   1,  20,  17,  18,  61, 168],
        [ 58,  30, 411,  60, 140,  99,  90,  72,  21,  19],
        [ 28,  27, 105, 304,  58, 226, 135,  66,  17,  34],
        [ 49,  13, 178,  55, 366,  75, 109, 121,  18,  16],
        [ 12,  15, 121, 146,  65, 436,  80,  84,  22,  19],
        [  3,  15,  83, 119, 127,  59, 535,  35,   5,  19],
        [ 30,  17,  45,  70,  71, 114,  40, 555,  10,  48],
        [160,  85,  35,  40,  13,   7,   6,  18, 543,  93],
        [ 37, 195,  16,  32,   1,  22,  21,  41,  71, 564]])
Epoch 8:
        Duration = 271.673 seconds
        Training Loss: 1.4083
        Training Accuracy: 0.506
        Validation Accuracy: 0.5
tensor([[544,  62,  74,  15,  13,  47,  38,  36,  75,  96],
        [ 15, 627,  16,  18,   1,  41,  23,  24,  15, 220],
        [ 46,  17, 340,  83,  88, 163, 146,  79,  13,  25],
        [  9,  12,  46, 339,  24, 297, 156,  72,   6,  39],
        [ 27,   9, 101,  98, 234, 145, 227, 139,   4,  16],
        [  7,   7,  55, 147,  19, 561,  92,  87,   4,  21],
        [  3,   6,  37, 114,  32,  71, 690,  28,   1,  18],
        [ 12,   9,  23,  71,  38, 167,  51, 586,   5,  38],
        [185,  93,  34,  38,  10,  36,  20,  21, 416, 147],
        [ 33, 148,  11,  27,   2,  37,  32,  36,  12, 662]])
Epoch 10:
        Duration = 255.405 seconds
        Training Loss: 1.31234
        Training Accuracy: 0.559
        Validation Accuracy: 0.541
tensor([[659,  40,  44,  11,  31,  15,  12,  19, 114,  55],
        [ 37, 741,   8,   6,   3,  13,   5,   5,  27, 155],
        [ 83,  28, 377,  22, 202, 126,  56,  59,  22,  25],
        [ 28,  23,  74, 172, 102, 336, 129,  71,  22,  43],
```

```
[ 51,  16,  95,  24, 515,  92,  73, 109,  13,  12],
[ 20,  14,  78,  50,  81, 561,  61,  96,  17,  22],
[ 18,  18,  56,  42, 160,  67, 578,  31,   8,  22],
[ 40,  13,  25,  25,  88, 141,  27, 585,  12,  44],
[190,  85,  24,  15,  10,  19,   6,  13, 589,  49],
[ 60, 187,  12,   5,   9,  17,  12,  28,  40, 630]])

Total Training Time = 3276.498 seconds
Average Training Time per Epoch (including validation) = 327.65 seconds
```

In [55]:
```
title_6 = "Figure 6 - Loss and Accuracy per Epoch for ResNet"
plot_model(title_6, t_loss_hist_6, t_acc_hist_6, v_acc_hist_6, 'upper right')
```



In [102]:

```
class ResNet26(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet26, self).__init__()
        self.stem = 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))
        self.b1_1 = nn.Sequential(nn.LazyConv2d(64, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(64, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(256, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b1_2 = nn.Sequential(nn.LazyConv2d(256, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b2 = nn.Sequential(nn.LazyConv2d(64, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(64, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(256, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b3_1 = nn.Sequential(nn.LazyConv2d(128, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(128, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(512, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b3_2 = nn.Sequential(nn.LazyConv2d(512, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b4 = nn.Sequential(nn.LazyConv2d(128, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(128, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(512, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b5_1 = nn.Sequential(nn.LazyConv2d(256, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(256, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(1024, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b5_2 = nn.Sequential(nn.LazyConv2d(1024, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b6 = nn.Sequential(nn.LazyConv2d(256, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(256, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(1024, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b7_1 = nn.Sequential(nn.LazyConv2d(512, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(512, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(2048, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b7_2 = nn.Sequential(nn.LazyConv2d(2048, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.b8 = nn.Sequential(nn.LazyConv2d(512, kernel_size=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(512, kernel_size=3, padding=1),
                                  nn.LazyBatchNorm2d(),
                                  nn.ReLU(),
                                  nn.LazyConv2d(2048, kernel_size=1),
                                  nn.LazyBatchNorm2d())
        self.relu = nn.ReLU()
        self.head = nn.Sequential(nn.AdaptiveAvgPool2d((1, 1)),
                                  nn.Flatten(),
                                  nn.LazyLinear(num_classes))
```

```
    def forward(self, x):
        out = self.stem(x)
        out = self.relu(self.b1_1(out) + self.b1_2(out))
        out = self.relu(out + self.b2(out))
        out = self.relu(self.b3_1(out) + self.b3_2(out))
        out = self.relu(out + self.b4(out))
        out = self.relu(self.b5_1(out) + self.b5_2(out))
        out = self.relu(out + self.b6(out))
        out = self.relu(self.b7_1(out) + self.b7_2(out))
        out = self.relu(out + self.b8(out))
        out = self.head(out)
        return out

model_7 = ResNet26().to(device=try_gpu())
optimizer_7 = optim.SGD(model_7.parameters(), lr=0.01)
model_7.eval()
```

```
    (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): LazyConv2d(0, 256, kernel_size=(1, 1), stride=(1, 1))
    (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (b3_1): Sequential(
    (0): LazyConv2d(0, 128, kernel_size=(1, 1), stride=(1, 1))
    (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): LazyConv2d(0, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): LazyConv2d(0, 512, kernel_size=(1, 1), stride=(1, 1))
    (7): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (b3_2): Sequential(
    (0): LazyConv2d(0, 512, kernel_size=(1, 1), stride=(1, 1))
```

In [106]:
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_7, t_acc_hist_7, v_acc_hist_7 = training_loop(3,
                                                          optimizer_7,
                                                          model_7,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          1)
```

```
---------------------------------------------------------------------------
OutOfMemoryError                          Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11660\179788224.py in <module>
      1 torch.cuda.empty_cache()
      2 gc.collect()
----> 3 t_loss_hist_7, t_acc_hist_7, v_acc_hist_7 = training_loop(3,
      4                                                          optimizer_7,
      5                                                          model_7,

~\AppData\Local\Temp\ipykernel_11660\310723436.py in training_loop(n_epochs, optimizer, model, loss_fn, train_loader, val
_loader, update_freq)
     16             images = imgs.to(device=try_gpu())
     17             labels = lbls.to(device=try_gpu())
---> 18             outputs = model(images)
     19             del images
     20             loss = loss_fn(outputs, labels)

~\anaconda3\lib\site-packages\torch\nn\modules\module.py in _call_impl(self, *input, **kwargs)
   1188         if not (self._backward_hooks or self._forward_hooks or self._forward_pre_hooks or _global_backward_hooks
   1189                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1190             return forward_call(*input, **kwargs)
   1191         # Do not call functions when jit is used
   1192         full_backward_hooks, non_full_backward_hooks = [], []

~\AppData\Local\Temp\ipykernel_11660\2953130020.py in forward(self, x)
     86         out = self.stem(x)
     87         out = self.relu(self.b1_1(out) + self.b1_2(out))
---> 88         out = self.relu(out + self.b2(out))
     89         out = self.relu(self.b3_1(out) + self.b3_2(out))
     90         out = self.relu(out + self.b4(out))

~\anaconda3\lib\site-packages\torch\nn\modules\module.py in _call_impl(self, *input, **kwargs)
   1188         if not (self._backward_hooks or self._forward_hooks or self._forward_pre_hooks or _global_backward_hooks
   1189                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1190             return forward_call(*input, **kwargs)
   1191         # Do not call functions when jit is used
   1192         full_backward_hooks, non_full_backward_hooks = [], []

~\anaconda3\lib\site-packages\torch\nn\modules\container.py in forward(self, input)
    202     def forward(self, input):
    203         for module in self:
--> 204             input = module(input)
    205         return input
    206

~\anaconda3\lib\site-packages\torch\nn\modules\module.py in _call_impl(self, *input, **kwargs)
   1188         if not (self._backward_hooks or self._forward_hooks or self._forward_pre_hooks or _global_backward_hooks
   1189                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1190             return forward_call(*input, **kwargs)
   1191         # Do not call functions when jit is used
   1192         full_backward_hooks, non_full_backward_hooks = [], []

~\anaconda3\lib\site-packages\torch\nn\modules\batchnorm.py in forward(self, input)
    169         used for normalization (i.e. in eval mode when buffers are not None).
    170         """
--> 171         return F.batch_norm(
    172             input,
    173             # If buffers are not to be tracked, ensure that they won't be updated

~\anaconda3\lib\site-packages\torch\nn\functional.py in batch_norm(input, running_mean, running_var, weight, bias, traini
ng, momentum, eps)
   2448         _verify_batch_size(input.size())
   2449
-> 2450     return torch.batch_norm(
   2451         input, weight, bias, running_mean, running_var, training, momentum, eps, torch.backends.cudnn.enabled
   2452     )

OutOfMemoryError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0; 2.00 GiB total capacity; 1.12 GiB already alloc
ated; 0 bytes free; 1.15 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_spli
t_size_mb to avoid fragmentation.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

In [ ]:
```
title_7 = "Figure 7 - Loss and Accuracy per Epoch for ResNet26"
plot_model(title_7, t_loss_hist_7, t_acc_hist_7, v_acc_hist_7, 'upper right')
```

In [86]:
```
class ResNet34(nn.Module):
    def __init__(self, arch, num_classes=10):
        super(ResNet34, self).__init__()
        self.stem = 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))
        blks = []
        for i, b in enumerate(arch):
            blks.append(block(*b, first_block=(i==0)))
        self.blks = nn.Sequential(*blks)
        self.head = nn.Sequential(nn.AdaptiveAvgPool2d((1, 1)),
                                  nn.Flatten(),
                                  nn.LazyLinear(num_classes))

    def forward(self, x):
        out = self.stem(x)
        out = self.blks(out)
        out = self.head(out)
        return out

model_8 = ResNet34(arch=((3, 64), (4, 128), (6, 256), (3,512))).to(device=try_gpu())
optimizer_8 = optim.SGD(model_8.parameters(), lr=0.01)
model_8.eval()
```

Out[86]:
```
ResNet34(
  (stem): Sequential(
    (0): LazyConv2d(0, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
    (1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (blks): Sequential(
    (0): Sequential(
      (0): Residual(
        (conv1): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (bn2): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): Residual(
        (conv1): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): LazyConv2d(0, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn1): LazyBatchNorm2d(0, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

In [ ]:
```
torch.cuda.empty_cache()
gc.collect()
t_loss_hist_7, t_acc_hist_7, v_acc_hist_7 = training_loop(3,
                                                          optimizer_7,
                                                          model_7,
                                                          nn.CrossEntropyLoss(),
                                                          train_loader_1,
                                                          val_loader_1,
                                                          1)
```

In [ ]:
```
title_8 = "Figure 8 - Loss and Accuracy per Epoch for ResNet34"
plot_model(title_8, t_loss_hist_8, t_acc_hist_8, v_acc_hist_8, 'upper right')
```