HW5: Image Classification & Detection

CS4610/5335: Robotic Science and Systems (Spring 2023) | Robert Platt | Northeastern University

---

Please remember the following policies:

- Submissions should be made electronically via the Canvas. For this assignment, you should submit both a *.ipynb and *.pdf version of your completed Colab notebook as a single zipped file.
- You are welcome to discuss the programming questions (but not the written questions) with other students in the class. However, you must understand and write all code yourself. Also, you must list all students (if any) with whom you discussed your solutions to the programming questions.
- Please provide comments in your code to make it understandable to the graders.

Collaborators: [*Optional*]

```
%%capture
! pip install pytorch-lightning
```

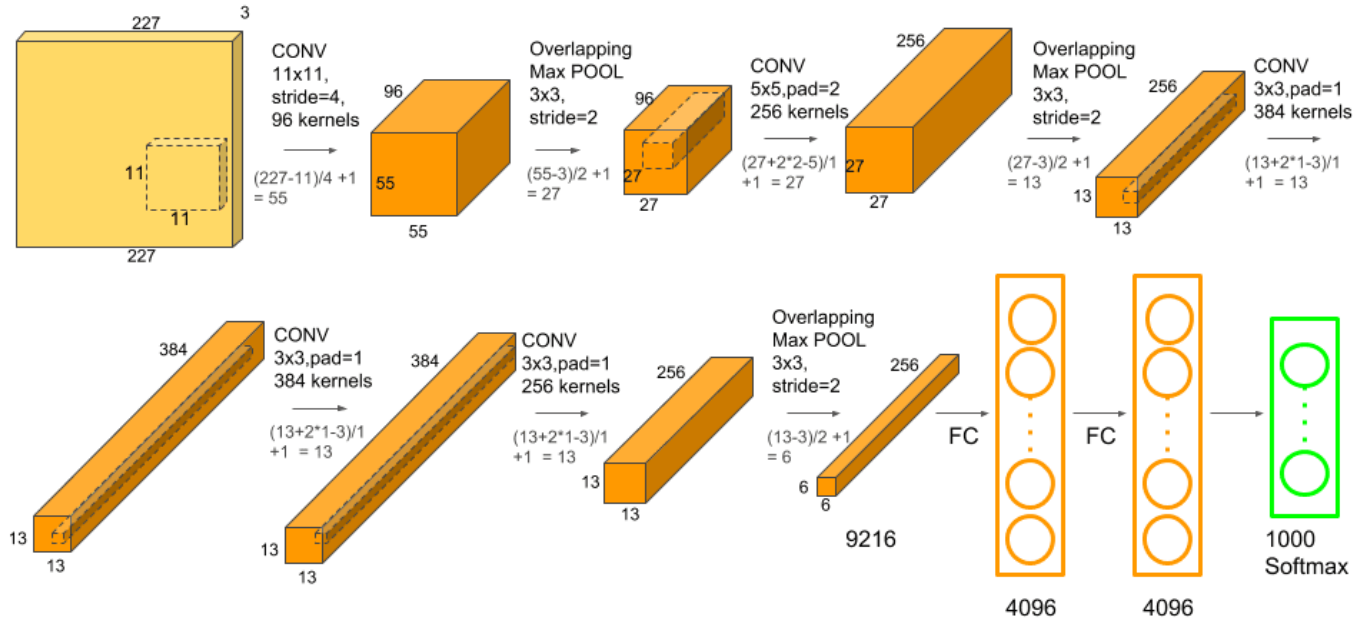
```
## UPLOAD utils.py: click "Choose Files" then click on "utils.py"
from google.colab import files
uploaded = files.upload()

try:
  import utils
except ModuleNotFoundError:
  raise ModuleNotFoundError(
    'ERROR: you did not upload "utils.py" correctly, run the cell again.'
  )
```

Browse... No files selected. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving utils.py to utils (2).py

```
import torch
from torch import Tensor
import torch.nn as nn
import torchvision
import pytorch_lightning as pl
import matplotlib.pyplot as plt
```

In this question, you will implement AlexNet in PyTorch using the model achitecture diagram provided below. For an introduction to creating neural networks with PyTorch, see this guide. If you want to look up specific pytorch modules, try searching the docs.




```
class AlexNet(nn.Module):
  def __init__(self):
    """
    You need to add ReLU activations after every internal convolution or linear lay
    Do not add BatchNorm layers
    """
    super().__init__()

    # define your layers here, the first operation is done for you
    # you may use nn.Sequential if you wish
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11, stride=4
    self.relu1 = nn.ReLU(True)
    self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.conv2 = nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5, stride=
    self.relu2 = nn.ReLU(True)
    self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.conv3 = nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3, stride
    self.relu3 = nn.ReLU(True)
    self.conv4 = nn.Conv2d(in_channels=384, out_channels=384, kernel_size=3, stride
    self.relu4 = nn.ReLU(True)
    self.conv5 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, stride
```

```
        self.relu5 = nn.ReLU(True)
        self.maxpool3 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.fc1 = nn.Linear(in_features=9216, out_features=4096)
        self.relu6 = nn.ReLU(inplace=True)
        self.fc2 = nn.Linear(in_features=4096, out_features=4096)
        self.relu7 = nn.ReLU(inplace=True)
        self.fc3 = nn.Linear(in_features=4096, out_features=1000)
        self.relu8 = nn.ReLU(inplace=True)
        self.softmax = nn.Softmax(dim=1)

        # ...

    def forward(self, x):
        """Performs forward pass

        Arguments
        ---------
        x: Tensor
            image tensor of shape (B, 3, 227, 227)

        Returns
        -------
        Tensor
            logits (ranging from 0 to 1) tensor with shape (B, 1000)
        """
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.maxpool2(x)
        x = self.conv3(x)
        x = self.relu3(x)
        x = self.conv4(x)
        x = self.relu4(x)
        x = self.conv5(x)
        x = self.relu5(x)
        x = self.maxpool3(x)
        x = torch.flatten(x, start_dim=1)
        x = self.fc1(x)
        x = self.relu6(x)
        x = self.fc2(x)
        x = self.relu7(x)
        x = self.fc3(x)
        x = self.relu8(x)
        x = self.softmax(x)

        return x


# To help you debug, you may want to print the shapes of tensors
```

```
# produced after each layer in the network
model = AlexNet()
img = torch.randn((1, 3, 227, 227), dtype=torch.float32)
model(img)

utils.check_q1a(AlexNet())
```

```
PASSED
```

Q1b. Customizing AlexNet for Smaller Images

The original AlexNet was designed for 227x227 images. Modify the network to handle 37x37 images while keeping the model parameters constant. In other words, do not change the channel dimensions or kernel sizes. Instead modify the convolution padding or stride or remove maxpool operations.

```
class SmallAlexNet(nn.Module):
  def __init__(self):
    '''
    You need to add ReLU activations after every internal convolution or linear lay
    Do not add BatchNorm layers

    Hint
    ----
    You want the feature map produced by the final conv layer to be 6x6
    '''
    super().__init__()

    # define your layers here, the first operation is done for you
    # you may use nn.Sequential if you wish
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=96, kernel_size=11, stride=1
    self.relu1 = nn.ReLU(True)
    #self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.conv2 = nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5, stride=
    self.relu2 = nn.ReLU(True)
    self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.conv3 = nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3, stride
    self.relu3 = nn.ReLU(True)
    self.conv4 = nn.Conv2d(in_channels=384, out_channels=384, kernel_size=3, stride
    self.relu4 = nn.ReLU(True)
    self.conv5 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, stride
    self.relu5 = nn.ReLU(True)
    self.maxpool3 = nn.MaxPool2d(kernel_size=3, stride=2)
    self.fc1 = nn.Linear(in_features=9216, out_features=4096)
    self.relu6 = nn.ReLU(inplace=True)
    self.fc2 = nn.Linear(in_features=4096, out_features=4096)
    self.relu7 = nn.ReLU(inplace=True)
```

```
    self.relu7 = nn.ReLU(inplace=True)
    self.fc3 = nn.Linear(in_features=4096, out_features=1000)
    self.relu8 = nn.ReLU(inplace=True)
    self.softmax = nn.Softmax(dim=1)


  def forward(self, x):
    """Performs forward pass

    Arguments
    ---------
    x: Tensor
      image tensor of shape (B, 3, 37, 37)

    Returns
    -------
    Tensor
      logits (ranging from 0 to 1) tensor with shape (B, 1000)
    """

    x = self.conv1(x)
    x = self.relu1(x)
    #x = self.maxpool1(x)
    x = self.conv2(x)
    x = self.relu2(x)
    x = self.maxpool2(x)
    x = self.conv3(x)
    x = self.relu3(x)
    x = self.conv4(x)
    x = self.relu4(x)
    x = self.conv5(x)
    x = self.relu5(x)
    x = self.maxpool3(x)
    x = torch.flatten(x, start_dim=1)
    x = self.fc1(x)
    x = self.relu6(x)
    x = self.fc2(x)
    x = self.relu7(x)
    x = self.fc3(x)
    x = self.relu8(x)
    x = self.softmax(x)
    return x

utils.check_q1b(SmallAlexNet())
```

```
PASSED
```

Q2. Implementing Basic Block

One of the most common network architectures for computer vision is the Residual Network

One of the most common network architectures for computer vision is the Residual Network (ResNet), first introduced by He et al. (2015). ResNets are constructed by stacking many residual blocks in a row. In this problem, you will implement the BasicBlock, which is used in the ResNet18 and ResNet34 networks.

```
class BasicBlock(nn.Module):
  def __init__(self, c_in: int, c_out: int, stride):
    """
    x -> conv3x3(c_in, c_out, stride) -> bn -> relu -> conv3x3(c_out, c_out) -> bn
    |
    '------------------> conv1x1(c_in, c_out, stride) -> bn ----------------------
    """
    super(BasicBlock, self).__init__()

    self.conv1 = nn.Conv2d(c_in, c_out, kernel_size=3, stride=stride, padding=1, bi
    self.bn1 = nn.BatchNorm2d(c_out)
    self.relu = nn.ReLU(inplace=True)
    self.conv2 = nn.Conv2d(c_out, c_out, kernel_size=3, stride=1, padding=1, bias=F
    self.bn2 = nn.BatchNorm2d(c_out)
    self.conv3 = nn.Conv2d(c_in, c_out, kernel_size=1, stride=stride,bias=False)
    self.bn3 = nn.BatchNorm2d(c_out)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
      residual = x

      out = self.conv1(x)
      out = self.bn1(x)
      out = self.relu(x)

      out = self.conv2(x)
      out = self.bn2(x)

      residual = self.conv3(residual)
      residual = self.bn3(residual)

      x += residual
      x = self.relu(x)

      return x
```

Why are residual connections beneficial when designing deep networks?

Some of the reasons why residual connections are benificial when designing deep networks are:
1)Residual connections helps fix the issue of vanishing gradientwhich occures when we train very deep networks. This is because in deep networks as the network backpropagates the gradient becomes very small, making it difficult to update the weights in the lower layers.
2)Residual connections allow information from earlier layers to bypass later layers and be

directly propagated to the output of the block, thus enabling better optimization of the weights in the earlier layers.
3) Residual connections can prevent overfitting, as they enable the network to learn both low-level and high-level features simultaneously.

Why is there a conv1x1 layer in the residual pathway? What purpose does it serve? Hint: see what happens when it is removed.

The conv 1x1 layer in the residual pathway of neural network is used to reduce the dimensionality of the feature map. it is also used to improve the performance of the model. **1)**Dimensionality reduction: The conv1x1 layer reduces the number of feature maps, which helps in lowering the network's computational complexity. This is critical, particularly for extremely deep networks where the parameter values to go into huge numbers. **2)**Non-linearity: The conv1x1 layer introduces non-linearity into the network, which is necessary for the network to learn complex representations of the input data. **3)**Performance: The network might still be able to train if the conv1x1 layer is removed, but the performance is likely going to decrease. This is due to the conv1x1 layer's contribution to enhancing the network's accuracy and stability, which can be essential for tasks like image classification.

Q3. Fine Tuning on Smaller Dataset

You are given MobileNetV2 pretrained on ImageNet and need to modify it so that it can train on a smaller dataset of flowers. To do this, you need to switch the network head to a linear layer with the proper output dimension, in this case 102 classes. Then you need to configure the optimizer to apply gradient descent on the head.

```
class MobileNetV2_finetune(pl.LightningModule):
  def __init__(self):
    super().__init__()
    self.pretrained_model = torchvision.models.mobilenet_v2(
        weights=torchvision.models.MobileNet_V2_Weights
    )
    # MODIFY MODEL HEAD
    #self.pretrained_model.classifier = nn.Linear(self.pretrained_model.last_channe
    self.parameters = nn.Sequential(
        nn.Dropout(0.5),
        nn.Linear(1280, 102),
        nn.Softmax(dim=1))

  def forward(self, x):
    """Performs forward pass
```

```
    Arguments
    ---------
    x: Tensor
      image tensor of shape (B, 3, 128, 128)

    Returns
    -------
    Tensor
      logits (ranging from 0 to 1) tensor with shape (B, 102)
    """
    # MAKE SURE TO PERFORM SOFTMAX on output
    #x = self.pretrained_model(x)
    # Perform softmax on output
    return self.pretrained_model(x)

  def configure_optimizers(self):
    # FIX THIS SO IT ONLY OPTIMIZES CERTAIN PARAMETERS
    trainable_params = self.parameters()
    optimizer = torch.optim.Adam(trainable_params, lr=1e-3)
    return optimizer

  def training_step(self, batch, batch_idx):
    x, y = batch
    y_pred = self(x)
    loss = nn.functional.nll_loss(y_pred, y)
    acc = (torch.argmax(y_pred, 1) == y).float().mean()
    self.log("train_acc", acc, prog_bar=True)
    return loss

  def validation_step(self, batch, batch_idx):
    x, y = batch
    y_pred = self(x)
    loss = nn.functional.nll_loss(y_pred, y)
    acc = (torch.argmax(y_pred, 1) == y).float().mean()
    self.log("val_acc", acc, prog_bar=True)
    return loss


model = MobileNetV2_finetune()
pl.seed_everything(42, workers=True)
train_dl, val_dl = utils.get_flowers_dataloaders(batch_size=512)
trainer = pl.Trainer(max_epochs=20, accelerator='gpu')
trainer.fit(model=model, train_dataloaders=train_dl, val_dataloaders=val_dl)

# this may take about 5 min to run
# you should achieve a validation accuracy of ~0.68 if done correctly
```

```
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:223: User\
  warnings.warn(msg)
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
```

```
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name             | Type        | Params
-------------------------------------------------
0 | pretrained_model | MobileNetV2 | 3.5 M
1 | parameters       | Sequential  | 130 K
-------------------------------------------------
3.6 M     Trainable params
0         Non-trainable params
3.6 M     Total params
14.542    Total estimated model params size (MB)


/usr/local/lib/python3.9/dist-packages/pytorch_lightning/loops/fit_loop.py:280
  rank_zero_warn(
```

Epoch 19: 100% 2/2 [00:17<00:00, 8.95s/it, v_num=10, train_acc=0.947, val_acc=0.748]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:223: UserW
  warnings.warn(msg)
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name             | Type        | Params
-------------------------------------------------
0 | pretrained_model | MobileNetV2 | 3.5 M
1 | parameters       | Sequential  | 130 K
-------------------------------------------------
3.6 M     Trainable params
0         Non-trainable params
3.6 M     Total params
14.542    Total estimated model params size (MB)


/usr/local/lib/python3.9/dist-packages/pytorch_lightning/loops/fit_loop.py:280
  rank_zero_warn(
Epoch 19: 100%          2/2 [00:15<00:00, 7.78s/it, v_num=13, train_acc=0.951, val_acc=0.753]
```

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
```

Q4. Data Augmentation to Improve Generalization

This is good but I bet we can do better. Try out some data augmentations and see what performance you can get (see full list of supported transforms here). For two different transformations (or compositions of transformations), explain why you believe this transformation could improve generalization and what validation accuracy you achieve (it's okay if it does not improve performance).

1. I have used RandomVerticalFlip(0.5) as I felt that the images are more symmetrical in nature because the dataset contains many pictures of flowers. THis gives me train_acc=89.4%, val_acc=70.8%.

2. I have used RandomHorizontalFlip(0.5) as I felt that the images are more symmetrical in nature because the dataset contains many pictures of flowers. This gives me train_acc=93.7%, val_acc=76.8% and on comparing the accuracies between the vertical and horizontal filter it can be observed that HorizontalFlip has better accuracy this might be because the dataset might have contained more images that are identical on the X axis.

```
# if you want to visualize dataset to think about useful transforms
_, val_dl = utils.get_flowers_dataloaders(batch_size=15)
imgs, _ = next(iter(val_dl))
plt.figure()
grid_img = torchvision.utils.make_grid(imgs, nrow=5, normalize=True).permute((1, 2,
plt.imshow(grid_img)
plt.axis('off')
plt.show()
```

```
import torchvision.transforms as transforms
# try your first transform here
TRAIN_TFM = torchvision.transforms.Compose([torchvision.transforms.RandomVerticalFl
model = MobileNetV2_finetune()
pl.seed_everything(42, workers=True)
train_dl, val_dl = utils.get_flowers_dataloaders(batch_size=512, train_tfm=TRAIN_TF
trainer = pl.Trainer(max_epochs=20, accelerator='gpu')
trainer.fit(model=model, train_dataloaders=train_dl, val_dataloaders=val_dl)
```

```
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:223: UserW
  warnings.warn(msg)
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name             | Type        | Params
-------------------------------------------------
0 | pretrained_model | MobileNetV2 | 3.5 M
1 | parameters       | Sequential  | 130 K
-------------------------------------------------
3.6 M     Trainable params
0         Non-trainable params
3.6 M     Total params
14.542    Total estimated model params size (MB)
```

```
Epoch 19: 100%          2/2 [00:14<00:00, 7.46s/it, v_num=11, train_acc=0.894, val_acc=0.708]
```

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
```

```
import torchvision.transforms as transforms
# try your second transform here
TRAIN_TFM = torchvision.transforms.Compose([transforms.RandomHorizontalFlip(p=0.5)
    ])

model = MobileNetV2_finetune()
pl.seed_everything(42, workers=True)
train_dl, val_dl = utils.get_flowers_dataloaders(batch_size=512, train_tfm=TRAIN_TF
trainer = pl.Trainer(max_epochs=20, accelerator='gpu')
trainer.fit(model=model, train_dataloaders=train_dl, val_dataloaders=val_dl)
```

```
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name              | Type        | Params
-------------------------------------------------
```

```
0 | pretrained_model | MobileNetV2 | 3.5 M
1 | parameters       | Sequential  | 130 K
-------------------------------------------------
3.6 M     Trainable params
0         Non-trainable params
3.6 M     Total params
14.542    Total estimated model params size (MB)
```

Epoch 19: 100% 2/2 [00:18<00:00, 9.00s/it, v_num=12, train_acc=0.937, val_acc=0.768]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
```

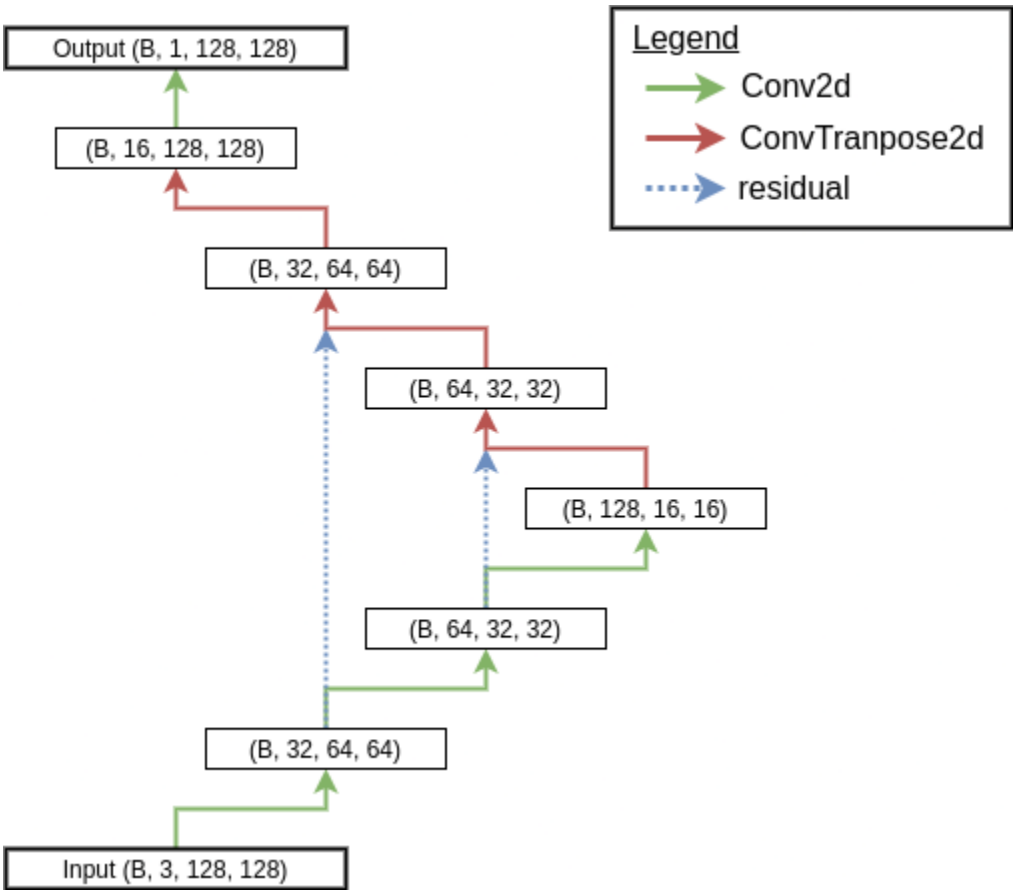
```
  | Name             | Type        | Params
-------------------------------------------------
0 | pretrained_model | MobileNetV2 | 3.5 M
1 | parameters       | Sequential  | 130 K
-------------------------------------------------
3.6 M     Trainable params
0         Non-trainable params
3.6 M     Total params
14.542    Total estimated model params size (MB)
```

Epoch 19: 100% 2/2 [00:14<00:00, 7.50s/it, v_num=14, train_acc=0.939, val_acc=0.762]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
```

Q5. Segmentation with UNet-style Architecture

Implement a UNet-style network as specified in the diagram below. Place a BatchNorm and ReLU after every conv or convtranspose layer.




```
class UNet(utils.SegmentationModule):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=2,
    self.bn1 = nn.BatchNorm2d(num_features=32)
    self.relu1 = nn.ReLU(True)

    self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=2
    self.bn2 = nn.BatchNorm2d(num_features=64)
    self.relu2 = nn.ReLU(True)

    self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=
    self.bn3 = nn.BatchNorm2d(num_features=128)
    self.relu3 = nn.ReLU(True)

    self.conv4 = nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=4
    self.bn4 = nn.BatchNorm2d(num_features=64)
    self.relu4 = nn.ReLU(True)

    self.conv5 = nn.ConvTranspose2d(in_channels=64, out_channels=32, kernel_size=4,
    self.bn5 = nn.BatchNorm2d(num_features=32)
    self.relu5 = nn.ReLU(True)
```

```
    self.conv6 = nn.ConvTranspose2d(in_channels=32, out_channels=16, kernel_size=4,
    self.bn6 = nn.BatchNorm2d(num_features=16)
    self.relu6 = nn.ReLU(True)

    self.conv7 = nn.Conv2d(in_channels=16, out_channels=1, kernel_size=1, stride=1,

  def forward(self, x):
    '''
    Arguments
    ---------
    x : Tensor
      image tensor of shape (B, 3, 128, 128)

    Returns
    -------
    Tensor
      image tensor of shape (B, 1, 128, 128) where every pixel is within (0,1)
      and describes the probability that the pixel is in the foreground
    '''
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu1(x)

    # saving the layer copy on block 2
    res1 = x
    x = self.conv2(x)
    x = self.bn2(x)
    x = self.relu2(x)

    # saving the layer copy on block 3
    res2 = x
    x = self.conv3(x)
    x = self.bn3(x)
    x = self.relu3(x)

    x = self.conv4(x)
    x = self.bn4(x)
    # adding the residual connection
    x += res2
    x = self.relu4(x)

    x = self.conv5(x)
    x = self.bn5(x)
    # adding the residual connection
    x += res1
    x = self.relu5(x)

    x = self.conv6(x)
    x = self.bn6(x)
    x = self.relu6(x)
```

```
        x = self.relu6(x)

        x = self.conv7(x)
        x = torch.sigmoid(x)

        return x


    # run this cell to debug the model using a dummy input
model = UNet()
dummy_input = torch.randn((30, 3, 128, 128), dtype=torch.float32)
output = model(dummy_input)

# MAKE SURE OUTPUT IMAGE HAS SAME HEIGHT AND WIDTH AS INPUT
assert output.shape[2:] == dummy_input.shape[2:]

# MAKE SURE THE OUTPUT IMAGE PIXELS VARY FROM 0 to 1
assert output.min().item() > 0 and output.max().item() < 1


# Train model once you have debugged the forward pass
train_dl, val_dl = utils.get_voc_dataloaders(batch_size=256)

model = UNet()
pl.seed_everything(42, workers=True)
trainer = pl.Trainer(max_epochs=10, accelerator='gpu',
                     logger=pl.loggers.TensorBoardLogger('./voc'),
                     log_every_n_steps=1)


# this will take about 8 minutes
trainer.fit(model=model, train_dataloaders=train_dl, val_dataloaders=val_dl)
```

```
Using downloaded and verified file: ./VOCtrainval_11-May-2012.tar
Extracting ./VOCtrainval_11-May-2012.tar to ./
Using downloaded and verified file: ./VOCtrainval_11-May-2012.tar
Extracting ./VOCtrainval_11-May-2012.tar to ./
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
   | Name       | Type            | Params
-----------------------------------------------
0  | acc_metric | BinaryAccuracy  | 0
1  | criterion  | BCELoss         | 0
2  | conv1      | Conv2d          | 896
3  | bn1        | BatchNorm2d     | 64
4  | relu1      | ReLU            | 0
5  | conv2      | Conv2d          | 18.5 K
6  | bn2        | BatchNorm2d     | 128
```

```
7  | relu2      | ReLU            | 0
8  | conv3      | Conv2d          | 73.9 K
9  | bn3        | BatchNorm2d     | 256
10 | relu3      | ReLU            | 0
11 | conv4      | ConvTranspose2d | 131 K
12 | bn4        | BatchNorm2d     | 128
13 | relu4      | ReLU            | 0
14 | conv5      | ConvTranspose2d | 32.8 K
15 | bn5        | BatchNorm2d     | 64
16 | relu5      | ReLU            | 0
17 | conv6      | ConvTranspose2d | 8.2 K
18 | bn6        | BatchNorm2d     | 32
19 | relu6      | ReLU            | 0
20 | conv7      | Conv2d          | 17
------------------------------------------------
266 K     Trainable params
0         Non-trainable params
266 K     Total params
1.064     Total estimated model params size (MB)
```

Epoch 9: 100% 2/2 [00:07<00:00, 3.61s/it, v_num=4, train_acc=0.665, val_acc=0.685]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
Using downloaded and verified file: ./VOCtrainval_11-May-2012.tar
Extracting ./VOCtrainval_11-May-2012.tar to ./
Using downloaded and verified file: ./VOCtrainval_11-May-2012.tar
Extracting ./VOCtrainval_11-May-2012.tar to ./
INFO:lightning_fabric.utilities.seed:Global seed set to 42
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: T
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
   | Name       | Type            | Params
------------------------------------------------
```

```
0  | acc_metric | BinaryAccuracy  | 0
1  | criterion  | BCELoss         | 0
2  | conv1      | Conv2d          | 896
3  | bn1        | BatchNorm2d     | 64
4  | relu1      | ReLU            | 0
5  | conv2      | Conv2d          | 18.5 K
6  | bn2        | BatchNorm2d     | 128
7  | relu2      | ReLU            | 0
8  | conv3      | Conv2d          | 73.9 K
9  | bn3        | BatchNorm2d     | 256
10 | relu3      | ReLU            | 0
11 | conv4      | ConvTranspose2d | 131 K
12 | bn4        | BatchNorm2d     | 128
13 | relu4      | ReLU            | 0
14 | conv5      | ConvTranspose2d | 32.8 K
15 | bn5        | BatchNorm2d     | 64
16 | relu5      | ReLU            | 0
17 | conv6      | ConvTranspose2d | 8.2 K
18 | bn6        | BatchNorm2d     | 32
19 | relu6      | ReLU            | 0
20 | conv7      | Conv2d          | 17
------------------------------------------------
266 K     Trainable params
0         Non-trainable params
266 K     Total params
1.064     Total estimated model params size (MB)
```

Epoch 9: 100% 2/2 [00:07<00:00, 3.68s/it, v_num=5, train_acc=0.665, val_acc=0.686]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=
```

Why is it useful to perform downsampling then upsampling within the network? Think about how information is processed within the network.

The network may obtain higher level characteristics from larger receptive fields with the use of

downsampling. Yet, it also results in the loss of certain significant spatial details required for successful segmentation. So, by utilizing transposed convolution or upsampling followed by convolution, upsampling aids in enlarging the feature maps to their original size.

The upsampling process restores the spatial resolution of the image and merges the feature maps from the downsampling stage with the corresponding feature maps from the upsampling stage, effectively combining low-level and high-level information to generate accurate segmentations.

Hence, the U-Net design may extract high-level features while maintaining spatial information, leading to effective segmentation of input images, by performing downsampling followed by upsampling.

Run the code cell below to view TensorBoard. Look at the plots of the train and validation losses. Is model overfitting here? Explain why you think it is or is not.
No the model is not overfitting because there is no significant difference between the training accuracy and validation accuracy. Here it can be observed that the training accuracy is 72.8% and the validation accuracy is 76%.1. The difference between the accuracies is only 3.3% so the model is not overfitting. We can say the model is overfitting only if training accuracy is significantly higher than the validation accuracy.

```
%load_ext tensorboard
%tensorboard --logdir voc/lightning_logs
# it make take a few seconds to load
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 36576), started 0:29:15 ago. (Use
'!kill 36576' to kill it.)
```