Open Lab 3

Convolution Networks

CSCI 7850 - Deep Learning

Due: Thursday, Oct. 12 @ 11:00pm

Assignment

Here are the details of what you need to do for this assignment:

- Create two of python scripts (`cifar10-resnet-relu.py` and `cifar10-resnet-gelu.py`) that run the corresponding model type (plain ResNet50 as below and ResNet50 *rebuilt by replacing any ReLU activations with GELU activations*) on the corresponding data set (cifar10):
  - You will need to reconstruct ResNet from scratch utilizing these tools:
    - `torch.nn.GELU`
    - Reference for GELU: https://pytorch.org/docs/stable/generated/torch.nn.GELU.html
    - Primary literature for GELU (in case you are curious, which you *should* be): https://arxiv.org/abs/1606.08415
  - Most settings should be maintained across all scripts (number of epochs=50, learning rate=0.001, etc.) and your final scripts only print the results line (validation accuracy) **exactly matching the format provided above**.
  - You will **need to use hamilton/babbage** to run these models within a reasonable time-frame. While the models can theoretically run on JupyterHub (azuread/biosim), you will not be able to train these models on these systems in the end since they have CPU-only resources.
- Perform the same process above on models for the **cifar100** data set as well. Be sure to clearly name your scripts accordingly (i.e. `cifar100-resnet-relu.py`).
- Use your scripts to run your models - **perform 5 independent runs for each combination**.
- Compile your data into *uniquely named* text files for each combination of model architectures that can be read in using `np.loadtxt` .
- Create an iPython Notebook file named `OL3.ipynb` which reads in the compiled results to produce a learning curve with mean and standard error: one for cifar10 and one for cifar100. No code for model training/testing should be in this notebook file: it should only read in the results text files, plot them, and include answers to the questions below. **Be sure to label your learning curves using the `plt.legend()` function.**
- At the end of your notebook file, create a Markdown cell and compile answers to the following questions:
  1. Which of the architectures performs best overall?
  2. Why do you think this may be happening?
  3. What other choices (hyperparameters, architecture changes, data prep, etc.) do you think might be explored which could impact the performance?
  4. What outcome would you expect from changing the code in this way (hypothesis)?
  5. What process would you use to attempt to confirm your hypothesis and what steps would testing it involve?
  6. Which part(s) of the lab/code/experiments are still *unclear* to you after finishing this assignment?
  7. Which parts are you interested in learning more about?

Submission

Create a zip archive which contains the following contents:

- cifar10-resnet-relu.py
- cifar10-resnet-gelu.py
- cifar100-resnet-relu.py
- cifar100-resnet-gelu.py
- OL3.ipynb
- **All** processed results text files (needed by np.loadtxt in your notebook)

Upload your zip archive to the course assignment system by the deadline at the top of this document.

Exploring Modified Convolution Architectures

We will utilize some of the same coding and experimental protocols and principles that were explored in previous labs to complete this lab, so be sure to refer back to previous Open Labs if needed for review.

Let's start with some imports that will help us explain our problem of interest...

In [1]:
```
import numpy as np
import torch
import lightning.pytorch as pl
import torchmetrics
import torchvision
from torchinfo import summary
from torchview import draw_graph
from IPython.display import display
import sympy as sp
sp.init_printing(use_latex=True)
import pandas as pd
import matplotlib.pyplot as plt
```

A list of physical compute devices present on a machine can be requested from the PyTorch configuration as follows:

In [2]:
```
if torch.cuda.is_available():
    print(torch.cuda.get_device_name())
    print(torch.cuda.get_device_properties("cuda"))
    print("Number of devices:",torch.cuda.device_count())
    device = ("cuda")
else:
    print("Only CPU is available...")
    device = ("cpu")
```

```
NVIDIA GeForce RTX 2080 Ti
_CudaDeviceProperties(name='NVIDIA GeForce RTX 2080 Ti', major=7, minor=5, total_memory=11011MB, mu
lti_processor_count=68)
Number of devices: 1
```

Now that we have set up our computing resources, we are ready to start the modeling/training/validation pipeline.

We will load up and prepare CIFAR which has 50,000 images for training/validation and 10,000 images for testing using 10 categories: 0) airplane, 1) automobile, 2) bird, 3) cat, 4) deer, 5) dog, 6) frog, 7) horse, 8) ship, and 9) truck.

Let's load the data:

In [40]:
```
# CIFAR 10
training_dataset = torchvision.datasets.CIFAR10(root='datasets',download=True, train=True)
testing_dataset = torchvision.datasets.CIFAR10(root='datasets',download=True, train=False)
```

```
x_train = torch.Tensor(training_dataset.data).permute(0, 3, 1, 2)
y_train = torch.Tensor(training_dataset.targets).to(torch.long)
x_test = torch.Tensor(testing_dataset.data).permute(0, 3, 1 ,2)
y_test = torch.Tensor(testing_dataset.targets).to(torch.long)
print(x_train.shape)
print(x_test.shape)
```

```
Files already downloaded and verified
Files already downloaded and verified
torch.Size([50000, 3, 32, 32])
torch.Size([10000, 3, 32, 32])
```

For simplicity, we will utilize all of the training data for training, and all of the testing data for validation...

In [78]:
```
batch_size = 250
xy_train = torch.utils.data.DataLoader(list(zip(x_train,
                                                y_train)),
                                       shuffle=True, batch_size=batch_size,
                                       num_workers=4)
xy_val = torch.utils.data.DataLoader(list(zip(x_test, y_test)),
                                     shuffle=False, batch_size=batch_size,
                                     num_workers=4)
```

```
/opt/conda/lib/python3.11/site-packages/torch/utils/data/dataloader.py:560: UserWarning: This DataL
oader will create 4 worker processes in total. Our suggested max number of worker in current system
is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive
worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid
potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

Pre-processing steps...

Note that I am permuting the image tensors to match the default PyTorch image format of CxHxW (while the images were originally in WxHxC format).

In all cases, I am converting targets into `torch.long` (long integers).

In [59]:
```
x_train.shape
```

Out[59]:
```
torch.Size([50000, 3, 32, 32])
```

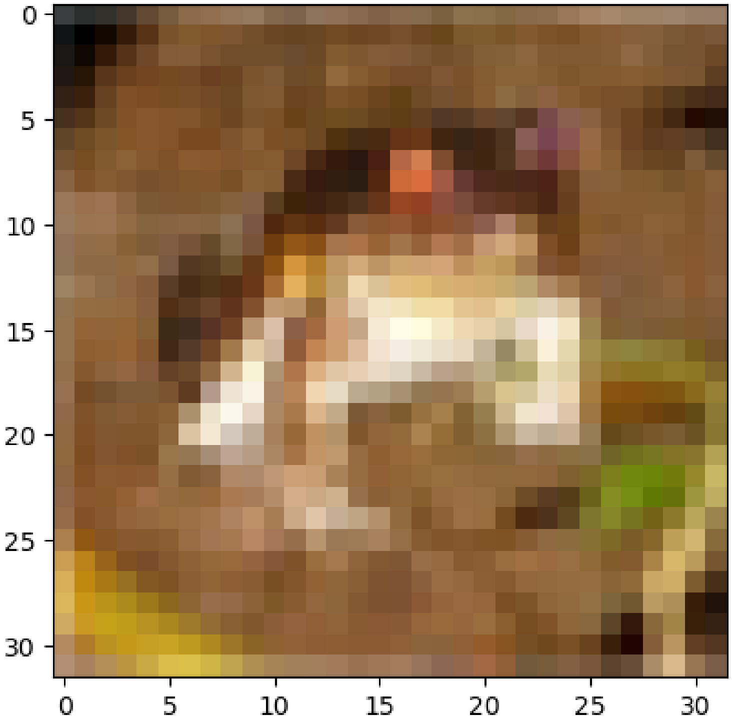
In [60]:
```
y_train.shape
```

Out[60]:
```
torch.Size([50000])
```

In [61]:
```
np.unique(y_train)
```

Out[61]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [62]:
```
plt.imshow(x_train[0].permute(1,2,0) / 255.0)
plt.show()
```



In [63]:
```
y_train[0]
```

Out[63]:
```
tensor(6)
```

Utilize an un-trained ResNet

Let's start with the control model for our experiments, the ResNet50 network:

We are going to add a data augmentation pipeline to the training phase as the primary means of avoiding overfitting. So see the comments for the `self.normalize` and `self.transform` compositions defined below...

In [64]:
```
class ResNet50(pl.LightningModule):
    def __init__(self,
                 input_shape,
                 output_size,
                 **kwargs):
        super().__init__(**kwargs)

        # Needs to always be applied to any incoming
        # image for this model. The Compose operation
        # takes a list of torchvision transforms and
        # applies them in sequential order, similar
        # to neural layers...
        self.normalize = torchvision.transforms.Compose([
            torchvision.transforms.Lambda(lambda x: x / 255.0),
            torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                             std=[0.229, 0.224, 0.225]),
        ])
        # Besides just scaling, the images can also undergo
        # augmentation using torchvision. Again, we compose
        # these operations together - ranges are provided for
        # each of these augmentations.
        self.transform = torchvision.transforms.Compose([
            torchvision.transforms.RandomAffine(degrees=(-10.0,10.0),
                                                translate=(0.1,0.1),
                                                scale=(0.9,1.1),
                                                shear=(-10.0,10.0)),
            torchvision.transforms.RandomHorizontalFlip(0.5),
        ])
```

```python
        # Linear projection - learned upsampling
        self.projection = torch.nn.ConvTranspose2d(3,3,
                                                   (4,4), # 8x
                                                   (4,4)) # 8+

        self.resnet = torchvision.models.resnet50(weights=None,
                                                  num_classes=output_size)
        self.mc_acc = torchmetrics.classification.Accuracy(task='multiclass',
                                                           num_classes=output_size)
        self.cce_loss = torch.nn.CrossEntropyLoss()

    def forward(self, x):
        y = x
        # Always normalize
        y = self.normalize(y)
        # Only augment when training
        if self.training:
            y = self.transform(y)
        y = self.projection(y)
        y = self.resnet(y)
        return y

    def predict(self, x):
        return torch.softmax(self(x),-1)

    def configure_optimizers(self):
        optimizer = torch.optim.SGD(self.parameters(), lr=0.01)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        x, y_true = train_batch
        y_pred = self(x)
        acc = self.mc_acc(y_pred,y_true)
        loss = self.cce_loss(y_pred,y_true)
        self.log('train_acc', acc, on_step=False, on_epoch=True)
        self.log('train_loss', loss, on_step=False, on_epoch=True)
        return loss

    def validation_step(self, val_batch, batch_idx):
        x, y_true = val_batch
        y_pred = self(x)
        acc = self.mc_acc(y_pred,y_true)
        loss = self.cce_loss(y_pred,y_true)
        self.log('val_acc', acc, on_step=False, on_epoch=True)
        self.log('val_loss', loss, on_step=False, on_epoch=True)
        return loss
```

In [65]:

```python
model = ResNet50(x_train.shape[1:],
                 len(torch.unique(y_train)))
summary(model, input_size=(1,)+x_train.shape[1:],
        depth=4)
```

```
Out[65]: ==========================================================================================
         Layout (type:depth-idx)                     Output Shape              Param #
         ==========================================================================================
         ResNet50                                    [1, 10]                   --
         ├─ConvTranspose2d: 1-1                      [1, 3, 128, 128]          147
         ├─ResNet: 1-2                               [1, 10]                   --
         │    └─Conv2d: 2-1                          [1, 64, 64, 64]           9,408
         │    └─BatchNorm2d: 2-2                     [1, 64, 64, 64]           128
         │    └─ReLU: 2-3                            [1, 64, 64, 64]           --
         │    └─MaxPool2d: 2-4                       [1, 64, 32, 32]           --
         │    └─Sequential: 2-5                      [1, 256, 32, 32]          --
         │    │    └─Bottleneck: 3-1                 [1, 256, 32, 32]          --
         │    │    │    └─Conv2d: 4-1                [1, 64, 32, 32]           4,096
         │    │    │    └─BatchNorm2d: 4-2           [1, 64, 32, 32]           128
         │    │    │    └─ReLU: 4-3                  [1, 64, 32, 32]           --
         │    │    │    └─Conv2d: 4-4                [1, 64, 32, 32]           36,864
         │    │    │    └─BatchNorm2d: 4-5           [1, 64, 32, 32]           128
         │    │    │    └─ReLU: 4-6                  [1, 64, 32, 32]           --
         │    │    │    └─Conv2d: 4-7                [1, 256, 32, 32]          16,384
         │    │    │    └─BatchNorm2d: 4-8           [1, 256, 32, 32]          512
         │    │    │    └─Sequential: 4-9            [1, 256, 32, 32]          16,896
         │    │    │    └─ReLU: 4-10                 [1, 256, 32, 32]          --
         │    │    └─Bottleneck: 3-2                 [1, 256, 32, 32]          --
         │    │    │    └─Conv2d: 4-11               [1, 64, 32, 32]           16,384
         │    │    │    └─BatchNorm2d: 4-12          [1, 64, 32, 32]           128
         │    │    │    └─ReLU: 4-13                 [1, 64, 32, 32]           --
         │    │    │    └─Conv2d: 4-14               [1, 64, 32, 32]           36,864
         │    │    │    └─BatchNorm2d: 4-15          [1, 64, 32, 32]           128
         │    │    │    └─ReLU: 4-16                 [1, 64, 32, 32]           --
         │    │    │    └─Conv2d: 4-17               [1, 256, 32, 32]          16,384
         │    │    │    └─BatchNorm2d: 4-18          [1, 256, 32, 32]          512
         │    │    │    └─ReLU: 4-19                 [1, 256, 32, 32]          --
         │    │    └─Bottleneck: 3-3                 [1, 256, 32, 32]          --
         │    │    │    └─Conv2d: 4-20               [1, 64, 32, 32]           16,384
         │    │    │    └─BatchNorm2d: 4-21          [1, 64, 32, 32]           128
         │    │    │    └─ReLU: 4-22                 [1, 64, 32, 32]           --
         │    │    │    └─Conv2d: 4-23               [1, 64, 32, 32]           36,864
         │    │    │    └─BatchNorm2d: 4-24          [1, 64, 32, 32]           128
         │    │    │    └─ReLU: 4-25                 [1, 64, 32, 32]           --
         │    │    │    └─Conv2d: 4-26               [1, 256, 32, 32]          16,384
         │    │    │    └─BatchNorm2d: 4-27          [1, 256, 32, 32]          512
         │    │    │    └─ReLU: 4-28                 [1, 256, 32, 32]          --
         │    └─Sequential: 2-6                      [1, 512, 16, 16]          --
         │    │    └─Bottleneck: 3-4                 [1, 512, 16, 16]          --
         │    │    │    └─Conv2d: 4-29               [1, 128, 32, 32]          32,768
         │    │    │    └─BatchNorm2d: 4-30          [1, 128, 32, 32]          256
         │    │    │    └─ReLU: 4-31                 [1, 128, 32, 32]          --
         │    │    │    └─Conv2d: 4-32               [1, 128, 16, 16]          147,456
         │    │    │    └─BatchNorm2d: 4-33          [1, 128, 16, 16]          256
         │    │    │    └─ReLU: 4-34                 [1, 128, 16, 16]          --
         │    │    │    └─Conv2d: 4-35               [1, 512, 16, 16]          65,536
         │    │    │    └─BatchNorm2d: 4-36          [1, 512, 16, 16]          1,024
         │    │    │    └─Sequential: 4-37           [1, 512, 16, 16]          132,096
         │    │    │    └─ReLU: 4-38                 [1, 512, 16, 16]          --
         │    │    └─Bottleneck: 3-5                 [1, 512, 16, 16]          --
         │    │    │    └─Conv2d: 4-39               [1, 128, 16, 16]          65,536
         │    │    │    └─BatchNorm2d: 4-40          [1, 128, 16, 16]          256
         │    │    │    └─ReLU: 4-41                 [1, 128, 16, 16]          --
         │    │    │    └─Conv2d: 4-42               [1, 128, 16, 16]          147,456
         │    │    │    └─BatchNorm2d: 4-43          [1, 128, 16, 16]          256
         │    │    │    └─ReLU: 4-44                 [1, 128, 16, 16]          --
         │    │    │    └─Conv2d: 4-45               [1, 512, 16, 16]          65,536
         │    │    │    └─BatchNorm2d: 4-46          [1, 512, 16, 16]          1,024
         │    │    │    └─ReLU: 4-47                 [1, 512, 16, 16]          --
         │    │    └─Bottleneck: 3-6                 [1, 512, 16, 16]          --
         │    │    │    └─Conv2d: 4-48               [1, 128, 16, 16]          65,536
         │    │    │    └─BatchNorm2d: 4-49          [1, 128, 16, 16]          256
```

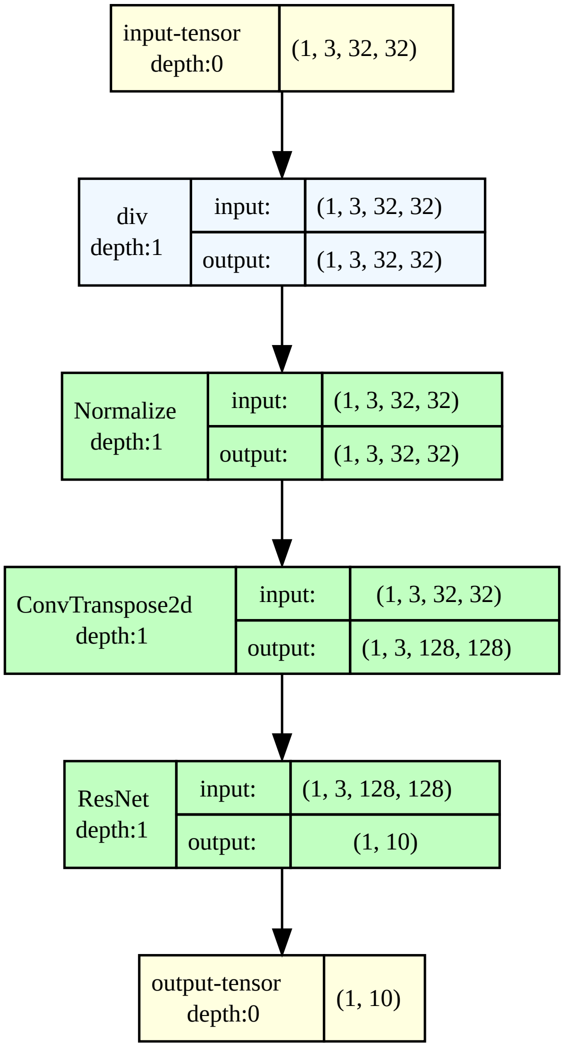
```
│    │    │    └─ReLU: 4-50                        [1, 128, 16, 16]          --
│    │    │    └─Conv2d: 4-51                      [1, 128, 16, 16]          147,456
│    │    │    └─BatchNorm2d: 4-52                 [1, 128, 16, 16]          256
│    │    │    └─ReLU: 4-53                        [1, 128, 16, 16]          --
│    │    │    └─Conv2d: 4-54                      [1, 512, 16, 16]          65,536
│    │    │    └─BatchNorm2d: 4-55                 [1, 512, 16, 16]          1,024
│    │    │    └─ReLU: 4-56                        [1, 512, 16, 16]          --
│    │    └─Bottleneck: 3-7                        [1, 512, 16, 16]          --
│    │    │    └─Conv2d: 4-57                      [1, 128, 16, 16]          65,536
│    │    │    └─BatchNorm2d: 4-58                 [1, 128, 16, 16]          256
│    │    │    └─ReLU: 4-59                        [1, 128, 16, 16]          --
│    │    │    └─Conv2d: 4-60                      [1, 128, 16, 16]          147,456
│    │    │    └─BatchNorm2d: 4-61                 [1, 128, 16, 16]          256
│    │    │    └─ReLU: 4-62                        [1, 128, 16, 16]          --
│    │    │    └─Conv2d: 4-63                      [1, 512, 16, 16]          65,536
│    │    │    └─BatchNorm2d: 4-64                 [1, 512, 16, 16]          1,024
│    │    │    └─ReLU: 4-65                        [1, 512, 16, 16]          --
│    └─Sequential: 2-7                             [1, 1024, 8, 8]           --
│    │    └─Bottleneck: 3-8                        [1, 1024, 8, 8]           --
│    │    │    └─Conv2d: 4-66                      [1, 256, 16, 16]          131,072
│    │    │    └─BatchNorm2d: 4-67                 [1, 256, 16, 16]          512
│    │    │    └─ReLU: 4-68                        [1, 256, 16, 16]          --
│    │    │    └─Conv2d: 4-69                      [1, 256, 8, 8]            589,824
│    │    │    └─BatchNorm2d: 4-70                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-71                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-72                      [1, 1024, 8, 8]           262,144
│    │    │    └─BatchNorm2d: 4-73                 [1, 1024, 8, 8]           2,048
│    │    │    └─Sequential: 4-74                  [1, 1024, 8, 8]           526,336
│    │    │    └─ReLU: 4-75                        [1, 1024, 8, 8]           --
│    │    └─Bottleneck: 3-9                        [1, 1024, 8, 8]           --
│    │    │    └─Conv2d: 4-76                      [1, 256, 8, 8]            262,144
│    │    │    └─BatchNorm2d: 4-77                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-78                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-79                      [1, 256, 8, 8]            589,824
│    │    │    └─BatchNorm2d: 4-80                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-81                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-82                      [1, 1024, 8, 8]           262,144
│    │    │    └─BatchNorm2d: 4-83                 [1, 1024, 8, 8]           2,048
│    │    │    └─ReLU: 4-84                        [1, 1024, 8, 8]           --
│    │    └─Bottleneck: 3-10                       [1, 1024, 8, 8]           --
│    │    │    └─Conv2d: 4-85                      [1, 256, 8, 8]            262,144
│    │    │    └─BatchNorm2d: 4-86                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-87                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-88                      [1, 256, 8, 8]            589,824
│    │    │    └─BatchNorm2d: 4-89                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-90                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-91                      [1, 1024, 8, 8]           262,144
│    │    │    └─BatchNorm2d: 4-92                 [1, 1024, 8, 8]           2,048
│    │    │    └─ReLU: 4-93                        [1, 1024, 8, 8]           --
│    │    └─Bottleneck: 3-11                       [1, 1024, 8, 8]           --
│    │    │    └─Conv2d: 4-94                      [1, 256, 8, 8]            262,144
│    │    │    └─BatchNorm2d: 4-95                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-96                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-97                      [1, 256, 8, 8]            589,824
│    │    │    └─BatchNorm2d: 4-98                 [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-99                        [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-100                     [1, 1024, 8, 8]           262,144
│    │    │    └─BatchNorm2d: 4-101                [1, 1024, 8, 8]           2,048
│    │    │    └─ReLU: 4-102                       [1, 1024, 8, 8]           --
│    │    └─Bottleneck: 3-12                       [1, 1024, 8, 8]           --
│    │    │    └─Conv2d: 4-103                     [1, 256, 8, 8]            262,144
│    │    │    └─BatchNorm2d: 4-104                [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-105                       [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-106                     [1, 256, 8, 8]            589,824
│    │    │    └─BatchNorm2d: 4-107                [1, 256, 8, 8]            512
│    │    │    └─ReLU: 4-108                       [1, 256, 8, 8]            --
│    │    │    └─Conv2d: 4-109                     [1, 1024, 8, 8]           262,144
│    │    │    └─BatchNorm2d: 4-110                [1, 1024, 8, 8]           2,048
```

```
│    │    │    └─ReLU: 4-111                    [1, 1024, 8, 8]          --
│    │    └─Bottleneck: 3-13                    [1, 1024, 8, 8]          --
│    │    │    └─Conv2d: 4-112                  [1, 256, 8, 8]           262,144
│    │    │    └─BatchNorm2d: 4-113             [1, 256, 8, 8]           512
│    │    │    └─ReLU: 4-114                    [1, 256, 8, 8]           --
│    │    │    └─Conv2d: 4-115                  [1, 256, 8, 8]           589,824
│    │    │    └─BatchNorm2d: 4-116             [1, 256, 8, 8]           512
│    │    │    └─ReLU: 4-117                    [1, 256, 8, 8]           --
│    │    │    └─Conv2d: 4-118                  [1, 1024, 8, 8]          262,144
│    │    │    └─BatchNorm2d: 4-119             [1, 1024, 8, 8]          2,048
│    │    │    └─ReLU: 4-120                    [1, 1024, 8, 8]          --
│    └─Sequential: 2-8                          [1, 2048, 4, 4]          --
│    │    └─Bottleneck: 3-14                    [1, 2048, 4, 4]          --
│    │    │    └─Conv2d: 4-121                  [1, 512, 8, 8]           524,288
│    │    │    └─BatchNorm2d: 4-122             [1, 512, 8, 8]           1,024
│    │    │    └─ReLU: 4-123                    [1, 512, 8, 8]           --
│    │    │    └─Conv2d: 4-124                  [1, 512, 4, 4]           2,359,296
│    │    │    └─BatchNorm2d: 4-125             [1, 512, 4, 4]           1,024
│    │    │    └─ReLU: 4-126                    [1, 512, 4, 4]           --
│    │    │    └─Conv2d: 4-127                  [1, 2048, 4, 4]          1,048,576
│    │    │    └─BatchNorm2d: 4-128             [1, 2048, 4, 4]          4,096
│    │    │    └─Sequential: 4-129              [1, 2048, 4, 4]          2,101,248
│    │    │    └─ReLU: 4-130                    [1, 2048, 4, 4]          --
│    │    └─Bottleneck: 3-15                    [1, 2048, 4, 4]          --
│    │    │    └─Conv2d: 4-131                  [1, 512, 4, 4]           1,048,576
│    │    │    └─BatchNorm2d: 4-132             [1, 512, 4, 4]           1,024
│    │    │    └─ReLU: 4-133                    [1, 512, 4, 4]           --
│    │    │    └─Conv2d: 4-134                  [1, 512, 4, 4]           2,359,296
│    │    │    └─BatchNorm2d: 4-135             [1, 512, 4, 4]           1,024
│    │    │    └─ReLU: 4-136                    [1, 512, 4, 4]           --
│    │    │    └─Conv2d: 4-137                  [1, 2048, 4, 4]          1,048,576
│    │    │    └─BatchNorm2d: 4-138             [1, 2048, 4, 4]          4,096
│    │    │    └─ReLU: 4-139                    [1, 2048, 4, 4]          --
│    │    └─Bottleneck: 3-16                    [1, 2048, 4, 4]          --
│    │    │    └─Conv2d: 4-140                  [1, 512, 4, 4]           1,048,576
│    │    │    └─BatchNorm2d: 4-141             [1, 512, 4, 4]           1,024
│    │    │    └─ReLU: 4-142                    [1, 512, 4, 4]           --
│    │    │    └─Conv2d: 4-143                  [1, 512, 4, 4]           2,359,296
│    │    │    └─BatchNorm2d: 4-144             [1, 512, 4, 4]           1,024
│    │    │    └─ReLU: 4-145                    [1, 512, 4, 4]           --
│    │    │    └─Conv2d: 4-146                  [1, 2048, 4, 4]          1,048,576
│    │    │    └─BatchNorm2d: 4-147             [1, 2048, 4, 4]          4,096
│    │    │    └─ReLU: 4-148                    [1, 2048, 4, 4]          --
│    └─AdaptiveAvgPool2d: 2-9                   [1, 2048, 1, 1]          --
│    └─Linear: 2-10                             [1, 10]                  20,490
===============================================================================================
Total params: 23,528,669
Trainable params: 23,528,669
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 1.34
===============================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 58.46
Params size (MB): 94.11
Estimated Total Size (MB): 152.59
===============================================================================================
```

Note that you need to set the `depth` flag to a larger value if you want see ResNet in its entirety.

In [66]:
```
model_graph = draw_graph(model, input_size=(1,)+x_train.shape[1:], device=device,
                        hide_inner_tensors=True,hide_module_functions=True,
                        expand_nested=False, depth=1)
model_graph.visual_graph
```

Out[66]:



Initial predictions with the random initial weights...

In [67]:
```
predictions = model.predict(x_train[:5].to(device)).cpu().detach().numpy()
print(predictions)
```

```
[[0.11970423 0.08197451 0.05884943 0.16465342 0.10559594 0.11571512
  0.11629636 0.07668598 0.0743398  0.0861852 ]
 [0.16799873 0.09725221 0.05188746 0.1326848  0.0771573  0.13584383
  0.12215839 0.10219138 0.03518614 0.07763977]
 [0.14733356 0.11618423 0.04384485 0.15868603 0.06847981 0.15053093
  0.08264749 0.09010516 0.04073072 0.10145719]
 [0.14150819 0.07442706 0.06278161 0.17160246 0.08209937 0.12416675
  0.11422311 0.07424976 0.06892944 0.08601223]
 [0.13802421 0.08854379 0.04661158 0.14588535 0.05871483 0.14705016
  0.11837248 0.09624066 0.05401621 0.10654077]]
```

In [68]:
```
predictions.shape
```

Out[68]: $(5, 10)$

Highest probabilities for each output vector: not good just yet of course!

In [69]:
```
predictions.argmax(-1)
```

Out[69]:
```
array([3, 0, 3, 3, 5])
```

In [70]:
```
y_train[:5]
```

Out[70]:
```
tensor([6, 9, 9, 4, 1])
```

Training...

In [71]:
```python
logger = pl.loggers.CSVLogger("lightning_logs",
                              name="Open_Lab_3",
                              version="demo-0")
```

In [72]:
```python
trainer = pl.Trainer(logger=logger,
                     max_epochs=50,
                     enable_progress_bar=True,
                     log_every_n_steps=0,
                     enable_checkpointing=False,
                     callbacks=[pl.callbacks.TQDMProgressBar(refresh_rate=50)])
```

```
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
```

In [73]:
```python
trainer.fit(model, xy_train, xy_val)
```

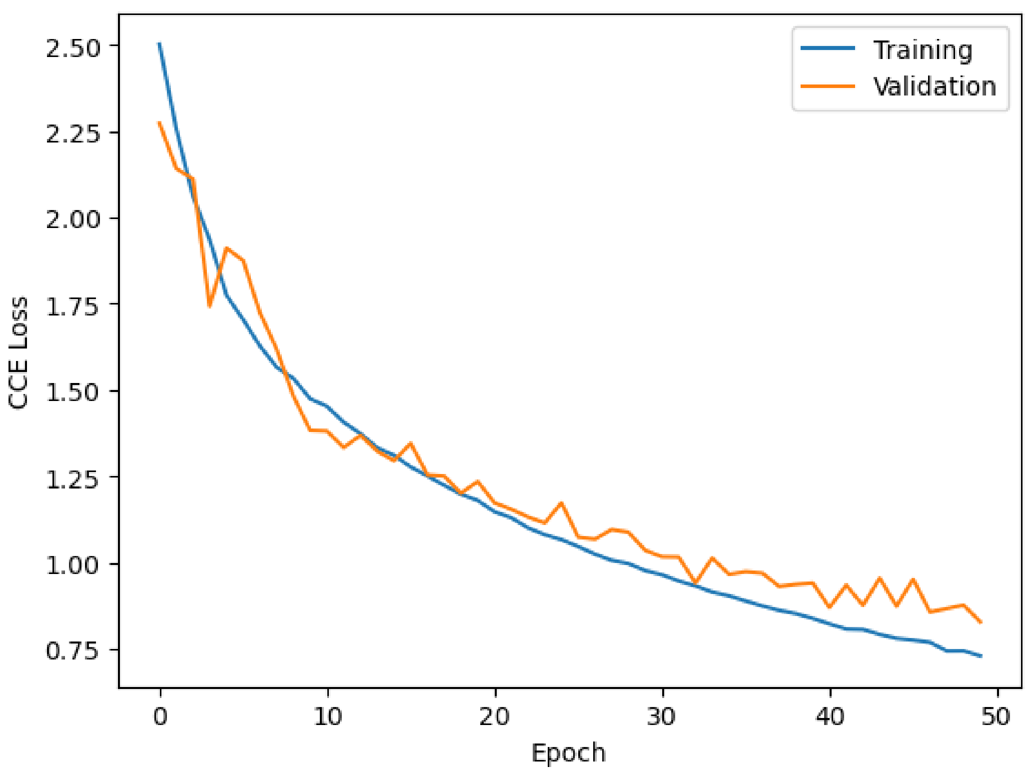
```
/opt/conda/lib/python3.11/site-packages/lightning/fabric/loggers/csv_logs.py:195: UserWarning: Expe
riment logs directory lightning_logs/Open_Lab_3/demo-0 exists and is not empty. Previous log files
in this directory will be deleted when the new ones are saved!
  rank_zero_warn(
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name       | Type                 | Params
--------------------------------------------------
0 | projection | ConvTranspose2d      | 147
1 | resnet     | ResNet               | 23.5 M
2 | mc_acc     | MulticlassAccuracy   | 0
3 | cce_loss   | CrossEntropyLoss     | 0
--------------------------------------------------
23.5 M    Trainable params
0         Non-trainable params
23.5 M    Total params
94.115    Total estimated model params size (MB)
SLURM auto-requeueing enabled. Setting signal handlers.
Sanity Checking: 0it [00:00, ?it/s]
/opt/conda/lib/python3.11/site-packages/lightning/pytorch/trainer/connectors/data_connector.py:442:
PossibleUserWarning: The dataloader, val_dataloader, does not have many workers which may be a bott
leneck. Consider increasing the value of the `num_workers` argument` (try 20 which is the number of
cpus on this machine) in the `DataLoader` init to improve performance.
  rank_zero_warn(
/opt/conda/lib/python3.11/site-packages/lightning/pytorch/trainer/connectors/data_connector.py:442:
PossibleUserWarning: The dataloader, train_dataloader, does not have many workers which may be a bo
ttleneck. Consider increasing the value of the `num_workers` argument` (try 20 which is the number
of cpus on this machine) in the `DataLoader` init to improve performance.
  rank_zero_warn(
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
```

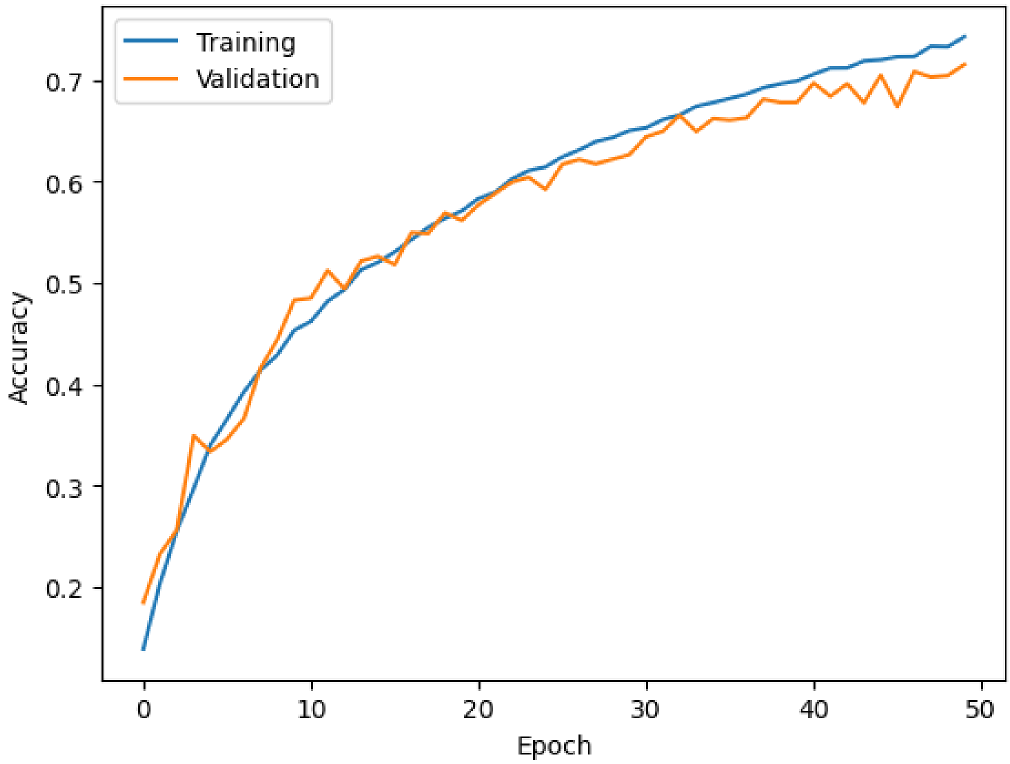
```
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
`Trainer.fit` stopped: `max_epochs=50` reached.
```

In [74]:
```
results = pd.read_csv(logger.log_dir+"/metrics.csv")
```

In [75]:
```
plt.plot(results["epoch"][np.logical_not(np.isnan(results["train_loss"]))],
         results["train_loss"][np.logical_not(np.isnan(results["train_loss"]))],
         label="Training")
plt.plot(results["epoch"][np.logical_not(np.isnan(results["val_loss"]))],
         results["val_loss"][np.logical_not(np.isnan(results["val_loss"]))],
         label="Validation")
plt.legend()
plt.ylabel("CCE Loss")
plt.xlabel("Epoch")
plt.show()
```



In [76]:
```
plt.plot(results["epoch"][np.logical_not(np.isnan(results["train_acc"]))],
         results["train_acc"][np.logical_not(np.isnan(results["train_acc"]))],
         label="Training")
plt.plot(results["epoch"][np.logical_not(np.isnan(results["val_acc"]))],
         results["val_acc"][np.logical_not(np.isnan(results["val_acc"]))],
         label="Validation")
plt.legend()
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.show()
```



In [77]:
```
print("Validation accuracy:",*["%.8f"%(x) for x in
                               results['val_acc'][np.logical_not(np.isnan(results["val_acc"]))]])
```

```
Validation accuracy: 0.18470000 0.23260000 0.25619999 0.34920001 0.33340001 0.34570000 0.36610001
0.41610000 0.44420001 0.48289999 0.48480001 0.51209998 0.49399999 0.52160001 0.52609998 0.51789999
0.54960001 0.54860002 0.56910002 0.56140000 0.57700002 0.58810002 0.59939998 0.60409999 0.59189999
0.61690003 0.62159997 0.61729997 0.62180001 0.62639999 0.64389998 0.64950001 0.66520000 0.64920002
0.66200000 0.66049999 0.66270000 0.68120003 0.67799997 0.67799997 0.69709998 0.68409997 0.69630003
0.67750001 0.70480001 0.67369998 0.70840001 0.70310003 0.70450002 0.71539998
```

Decent performance here overall, but we now have a **baseline model** for performing comparisons with modifications to the ResNet architecture...