Modified ResNet

We present the modified ResNet model with number of parameter unders 5 milions parameter which can achieve 94% accuracy in CIFAR10 testing dataset, and 82.6 % on custom data (NoLabel dataset).

Necessary libraries

```
!pip install "pytorch-lightning>=1.4" "torchmetrics>=0.6" "lightning-
bolts"
import torch
import torchvision
from torchvision import transforms
import matplotlib.pyplot as plt
import numpy as np
import pickle
import os
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.model_selection import train_test_split
from pl_bolts.datamodules import CIFAR10DataModule
from pl_bolts.transforms.dataset_normalizations import
cifar10_normalization
import torchvision.datasets as datasets
import torch.utils.data as data
import copy
import torch.nn as nn
import torch.nn.functional as F
from torch.optim.lr_scheduler import OneCycleLR

import os

import pandas as pd
import seaborn as sn
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision

from pl_bolts.datamodules import CIFAR10DataModule
from pl_bolts.transforms.dataset_normalizations import
cifar10_normalization
from pytorch_lightning import LightningModule, Trainer,
seed_everything
from pytorch_lightning.callbacks import LearningRateMonitor
from pytorch_lightning.callbacks.progress import TQDMProgressBar
from pytorch_lightning.loggers import CSVLogger
from torch.optim.lr_scheduler import OneCycleLR
from torch.optim.swa_utils import AveragedModel, update_bn
# from torchmetrics.functional import accuracy
```

```
from torchmetrics import Accuracy
import torchsummary
```

Loading and visualizing the dataset

We obtained the dataset from the `CIFAR10DataModule` within the `pl_bolts.datamodules` module of PyTorch Lightning Bolts. This module conveniently includes pre-split training, testing, and validation datasets. Additionally, it provides a `transform` function which we utilized to augment the data, enhancing the diversity of the training samples and improving the generalization ability of our model.

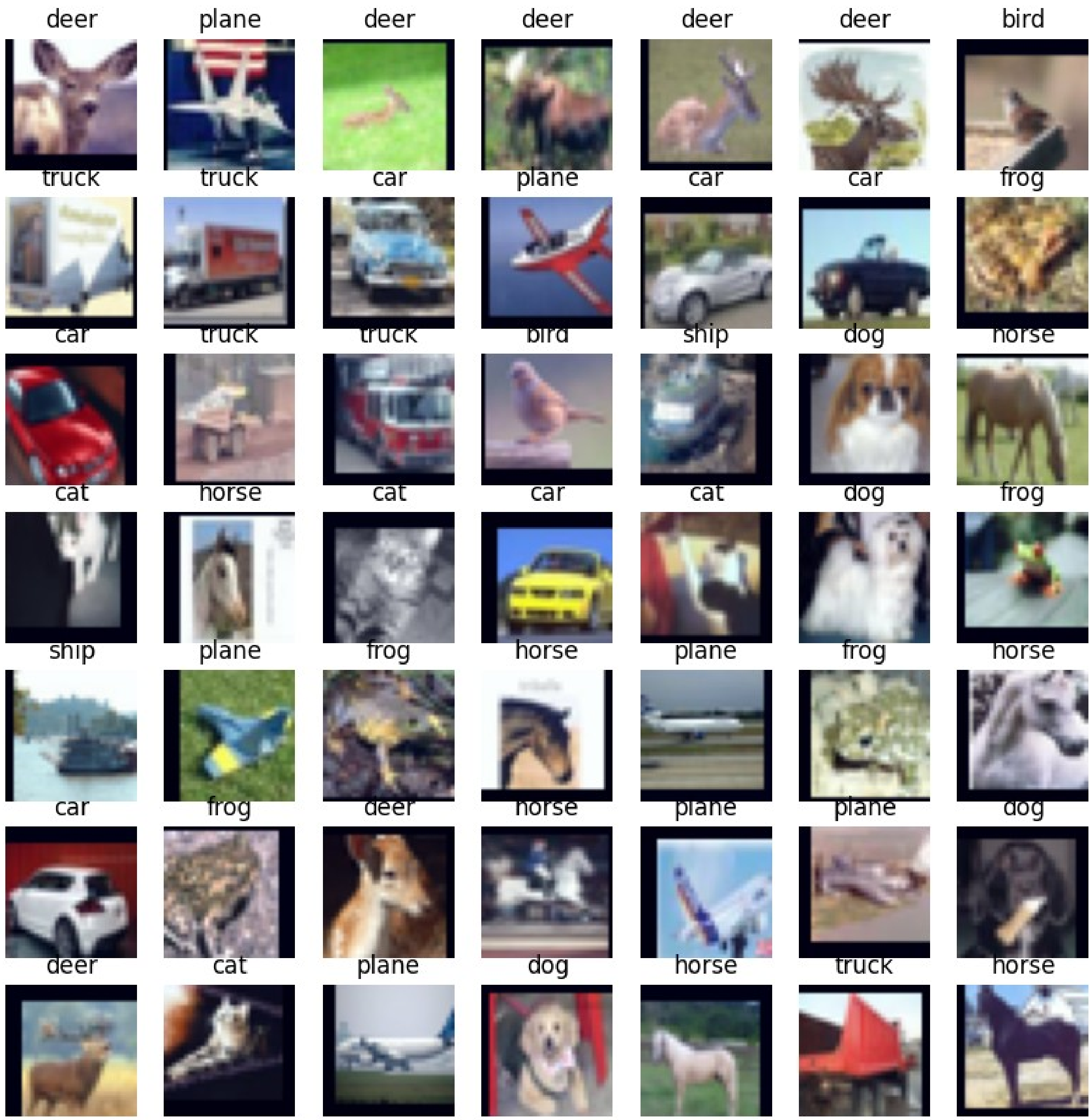
Here we defined a sequence of transformations to be applied to the training and Testing datasets. It includes: RandomCrop: Randomly crops the input image to the specified size (32x32) with padding. RandomHorizontalFlip: Randomly flips the input image horizontally with a probability of 0.5. ToTensor: Converts the input image to a PyTorch tensor. cifar10_normalization(): Applies normalization process, including scaling the pixel values to a specific range between 0 and 1 and standardizing them using the mean and standard deviation of the CIFAR-10 dataset.

We also can visualize random images from dataset in the figure below.

```
seed_everything(7)

PATH_DATASETS = os.environ.get("PATH_DATASETS", "/")
BATCH_SIZE = 256 if torch.cuda.is_available() else 64
NUM_WORKERS = int(os.cpu_count() / 2)
train_transforms = torchvision.transforms.Compose(
    [
        torchvision.transforms.RandomCrop(32, padding=4),
        torchvision.transforms.RandomHorizontalFlip(),
        torchvision.transforms.ToTensor(),
        cifar10_normalization(),
    ]
)

test_transforms = torchvision.transforms.Compose(
    [
        torchvision.transforms.ToTensor(),
        cifar10_normalization(),
    ]
)

cifar10_dm = CIFAR10DataModule(
    data_dir=PATH_DATASETS,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    train_transforms=train_transforms,
    test_transforms=test_transforms,
    val_transforms=test_transforms,
)


import matplotlib.pyplot as plt


def plot_images(images, labels, classes, normalize=False):
    """Plot a grid of subplots with the given images and their
corresponding labels."""
    n_images = len(images)
    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure(figsize=(10, 10))
    for i in range(rows*cols):
        ax = fig.add_subplot(rows, cols, i+1)
        img = images[i]
        if normalize:
            img = img.clone()
            img_min = img.min()
            img_max = img.max()
            img.clamp_(min=img_min, max=img_max)
            img.add_(-img_min).div_(img_max - img_min + 1e-5)

        ax.imshow(img.permute(1, 2, 0).cpu().numpy())
        ax.set_title(classes[labels[i]])
        ax.axis('off')
    plt.show()

def display_random_images(data_module, num_images=50):
    """Fetches images from the dataset and displays them."""
    data_module.prepare_data()
    data_module.setup()

    train_loader = data_module.train_dataloader()
    images, labels = next(iter(train_loader))

    # Randomly select `num_images` images
    indices = np.random.choice(len(images), num_images, replace=False)
    selected_images = images[indices]
    selected_labels = labels[indices]
    classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
    plot_images(selected_images, selected_labels, classes,
normalize=True)

display_random_images(cifar10_dm)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
```

```
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
INFO:lightning_fabric.utilities.seed:Global seed set to 7

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
/kaggle/working/cifar-10-python.tar.gz

100%|██████████| 170498071/170498071 [00:03<00:00, 43324048.68it/s]

Extracting /kaggle/working/cifar-10-python.tar.gz to /kaggle/working/
Files already downloaded and verified

/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded
code, and JAX is multithreaded, so this will likely lead to a
deadlock.
  self.pid = os.fork()
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded
code, and JAX is multithreaded, so this will likely lead to a
deadlock.
  self.pid = os.fork()
```



Defining the Model

We defind the Basic Blocks which is the construction of Resnet via Pytorch. Each Basic Block consists of several convolutional layers coupled with batch normalization and ReLU activations.The most distinguish feature "shortcut connections", which address vanishing gradient. In our work, the basic block contains:

1. Two Convolutional Layers: The first convolutional layer applies a set filters to the input feature map and the second convolutional layer further processes the output of the first, applying an additional set of filters. Both layers typically use a 3x3 kernel and include a bias term.

2. Batch Normalization Layers: Following each convolutional layer, a batch normalization layer is used. This layer normalizes the output of the convolution by adjusting and scaling the activations.

3. ReLU Activations: After each batch normalization layer, a ReLU (Rectified Linear Unit) activation function is applied. The ReLU function introduces non-linearity to the processing.

4. Shortcut Connections: This connection directly forwards the input x to the end of the block.

```
class BasicBlock(nn.Module):

    expansion = 1
    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, 
padding=1, bias=True)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=True)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=True),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):

        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

We define the ResNet model contains numbes of layers, each containing multiple BasicBlock which mentioned in previous section. To be specific, it contains:

1. Initial Convolutional Layer: a convolution layer with 64 filters 5x5 dimension followed by a batch normalization layer.
2. Stacked BasicBlock Layers: four layers, each made up of multiple 'BasicBlock'.
3. Final linear layer: a fully connected layer to get classification result for the 10 classes.

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=5,
                               stride=1, padding=1, bias=True)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0],
stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1],
stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2],
stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3],
stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):

        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
#         out =
        return out
```

We present our model by reducing the number of blocks from the last 3 layers presented by RESNET18 Structure to get the number of parameters under 5M.So our Resnet model structure [2,1,1,1] representing number of blocks at each layer.

```
def Our_ResNet():
    return ResNet(BasicBlock, [2, 1, 1, 1])
```

We calculate the total number of parameters in the model via torchsummary.

```
!pip install torchsummary
import torchsummary
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Our_ResNet()
model = model.to(device)

torchsummary.summary(model, input_size=(3,32,32))
```

```
Requirement already satisfied: torchsummary in
/usr/local/lib/python3.10/dist-packages (1.5.1)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 30, 30]           4,864
       BatchNorm2d-2           [-1, 64, 30, 30]             128
            Conv2d-3           [-1, 64, 30, 30]          36,928
       BatchNorm2d-4           [-1, 64, 30, 30]             128
            Conv2d-5           [-1, 64, 30, 30]          36,928
       BatchNorm2d-6           [-1, 64, 30, 30]             128
        BasicBlock-7           [-1, 64, 30, 30]               0
            Conv2d-8           [-1, 64, 30, 30]          36,928
       BatchNorm2d-9           [-1, 64, 30, 30]             128
           Conv2d-10           [-1, 64, 30, 30]          36,928
      BatchNorm2d-11           [-1, 64, 30, 30]             128
       BasicBlock-12           [-1, 64, 30, 30]               0
           Conv2d-13          [-1, 128, 15, 15]          73,856
      BatchNorm2d-14          [-1, 128, 15, 15]             256
           Conv2d-15          [-1, 128, 15, 15]         147,584
      BatchNorm2d-16          [-1, 128, 15, 15]             256
           Conv2d-17          [-1, 128, 15, 15]           8,320
      BatchNorm2d-18          [-1, 128, 15, 15]             256
       BasicBlock-19          [-1, 128, 15, 15]               0
           Conv2d-20            [-1, 256, 8, 8]         295,168
      BatchNorm2d-21            [-1, 256, 8, 8]             512
           Conv2d-22            [-1, 256, 8, 8]         590,080
      BatchNorm2d-23            [-1, 256, 8, 8]             512
           Conv2d-24            [-1, 256, 8, 8]          33,024
      BatchNorm2d-25            [-1, 256, 8, 8]             512
       BasicBlock-26            [-1, 256, 8, 8]               0
           Conv2d-27            [-1, 512, 4, 4]       1,180,160
      BatchNorm2d-28            [-1, 512, 4, 4]           1,024
           Conv2d-29            [-1, 512, 4, 4]       2,359,808
      BatchNorm2d-30            [-1, 512, 4, 4]           1,024
           Conv2d-31            [-1, 512, 4, 4]         131,584
      BatchNorm2d-32            [-1, 512, 4, 4]           1,024
       BasicBlock-33            [-1, 512, 4, 4]               0
           Linear-34                   [-1, 10]           5,130
================================================================
Total params: 4,983,306
Trainable params: 4,983,306
Non-trainable params: 0
----------------------------------------------------------------
```

```
Input size (MB): 0.01
Forward/backward pass size (MB): 8.12
Params size (MB): 19.01
Estimated Total Size (MB): 27.15
----------------------------------------------------------------
```

The initial value of the weights and biases is set by the following function.

```
def initialize_parameters(m):

    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight.data, nonlinearity = 'relu')
        nn.init.constant_(m.bias.data, 0)
    elif isinstance(m, nn.Linear):
        nn.init.xavier_normal_(m.weight.data, gain =
nn.init.calculate_gain('relu'))
        nn.init.constant_(m.bias.data, 0)

model.apply(initialize_parameters)

ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1),
padding=(1, 1))
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1))
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2))
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1))
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1))
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2))
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (linear): Linear(in_features=512, out_features=10, bias=True)
)
```

Training the Model.

We intergrate our Model with PyTorch Lightning Trainer which can simplifies the training workflow and is designed for a multiclass classification task. It contains:

1. Accuracy Metric: Setup for multiclass accuracy evaluation.
2. Model Initialization: constructs the ResNet model and initializes its parameters.
3. LightningModule Configuration: Defines training, validation, testing steps, and the optimization setup.

Our optimazing parameters:

1. Optimizer : we chose Stochastic Gradient Descent (SGD) with momentum. SGD is a widely used optimization algorithm as itt updates the model parameters in the direction of the negative gradient of the loss function with respect to the parameters. The momentum term in SGD helps speed up convergence by taking information from past gradients. This helps smooth out fluctuations in the optimization process, leading to more stable and efficient training
2. OneCycleLR Scheduler: The OneCycleLR scheduler is chosen to dynamically adjust the learning rate which leads to improving convergence and preventing overfitting.

```
accuracy = Accuracy(task="multiclass",
num_classes=10).to(device='cuda')
def create_model():
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    model = Our_ResNet()
    model = model.to(device)
    model.apply(initialize_parameters)

    return model

class LitResnet(LightningModule):
    def __init__(self, lr=0.05):
        super().__init__()

        self.save_hyperparameters()
        self.model = create_model()

    def forward(self, x):
        out = self.model(x)
        return F.log_softmax(out, dim=1)

    def training_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = F.nll_loss(logits, y)

        preds = torch.argmax(logits, dim=1)
        acc = accuracy(preds, y)

        self.log("train_loss", loss)
        self.log("train_acc", acc)

        return loss

    def evaluate(self, batch, stage=None):
        x, y = batch
        logits = self(x)
        loss = F.nll_loss(logits, y)
        preds = torch.argmax(logits, dim=1)
        acc = accuracy(preds, y)

        if stage:
            self.log(f"{stage}_loss", loss, prog_bar=True)
            self.log(f"{stage}_acc", acc, prog_bar=True)

    def validation_step(self, batch, batch_idx):
        self.evaluate(batch, "val")

    def test_step(self, batch, batch_idx):
        self.evaluate(batch, "test")

    def configure_optimizers(self):
        optimizer = torch.optim.SGD(
            self.parameters(),
            lr=self.hparams.lr,
            momentum=0.9,
            weight_decay=5e-4,
        )
        steps_per_epoch = 45000 // BATCH_SIZE
        scheduler_dict = {
            "scheduler": OneCycleLR(
                optimizer,
                0.1,
                epochs=self.trainer.max_epochs,
                steps_per_epoch=steps_per_epoch,
            ),
            "interval": "step",
        }
        return {"optimizer": optimizer, "lr_scheduler":
scheduler_dict}
```

We utilize the `LitResnet` class, to train our model with CIFAR10 dataset, including:

1. Learning rate 0.05
2. Number epochs of training 200

Through this process, we save the train and validation accuracy in log file and save the final model at the final.

```
from google.colab import drive
drive.mount('/content/drive')
model = LitResnet(lr=0.05)

trainer = Trainer(
    max_epochs=200,
    accelerator="auto",
    devices=1 if torch.cuda.is_available() else None,  # limiting got
iPython runs
    logger=CSVLogger(save_dir=r"/logs"),
    callbacks=[LearningRateMonitor(logging_interval="step"),
TQDMProgressBar(refresh_rate=10)],
)

trainer.fit(model, cifar10_dm)
# trainer.test(model, datamodule=cifar10_dm)
sum(p.numel() for p in model.parameters() if p.requires_grad)
torch.save(model.model.state_dict(), '/200_model.pth')
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda),
used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False,
using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False,
using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False,
using: 0 HPUs

Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
Files already downloaded and verified
Files already downloaded and verified

WARNING:lightning_fabric.loggers.csv_logs:Missing logger folder:
/logs/lightning_logs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 -
CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name  | Type   | Params
---------------------------------
0 | model | ResNet | 5.0 M
---------------------------------
5.0 M     Trainable params
0         Non-trainable params
```

```
5.0 M     Total params
19.933    Total estimated model params size (MB)

{"model_id":"2d3a4489cec64904b3efa015a1199ffd","version_major":2,"vers
ion_minor":0}

/usr/local/lib/python3.10/dist-packages/torchmetrics/utilities/
prints.py:32: DeprecationWarning: This property will be removed in
2.0.0. Use `Metric.updated_called` instead.
  return fn(*args, **kwargs)

{"model_id":"0b0bb1ef957649b2b1fc78ebbfc49030","version_major":2,"vers
ion_minor":0}

{"model_id":"640057d5cddf48fa99b8b6e3f2f979d9","version_major":2,"vers
ion_minor":0}

{"model_id":"4e119500f5ee4d638fafa042c45bb97e","version_major":2,"vers
ion_minor":0}

{"model_id":"6850f8913bf44645a9ad8de8c68aa478","version_major":2,"vers
ion_minor":0}

{"model_id":"695eceaaa7ab4e679f7ea576d9930796","version_major":2,"vers
ion_minor":0}

{"model_id":"944c36addc3e45ccb12f508365c7fa39","version_major":2,"vers
ion_minor":0}

{"model_id":"ec07ffc564004a2081151bddb318000e","version_major":2,"vers
ion_minor":0}

{"model_id":"c1e877566a664cbfa3f91ac31daa1838","version_major":2,"vers
ion_minor":0}

{"model_id":"b3ba0a00f60f493faa2df3739e51d524","version_major":2,"vers
ion_minor":0}

{"model_id":"a4c2c5b3d70b49569bf230a7076f37e4","version_major":2,"vers
ion_minor":0}

{"model_id":"ddb5d87738d34cd09459320d2f57c61a","version_major":2,"vers
ion_minor":0}

{"model_id":"1554fd3e56034907a4e4782def15ca44","version_major":2,"vers
ion_minor":0}

{"model_id":"37f2c63928eb495ea3743d1c0b66d9cb","version_major":2,"vers
ion_minor":0}

{"model_id":"a1829fd2fa484c77ab763e6324abbc57","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"579983f9cf9744c4b20740c1661fb1c4","version_major":2,"version_minor":0}

{"model_id":"0da37d999a084d54837484a5749aa855","version_major":2,"version_minor":0}

{"model_id":"595eb2afc6294518b444c28103be43f9","version_major":2,"version_minor":0}

{"model_id":"88050728fda64f5f9f3627eba1176500","version_major":2,"version_minor":0}

{"model_id":"60ea8979f8904630a44eb4627985a250","version_major":2,"version_minor":0}

{"model_id":"eda4e6809c4841b48ab73df8698f477b","version_major":2,"version_minor":0}

{"model_id":"25faff7509734f6f91dade9746641656","version_major":2,"version_minor":0}

{"model_id":"bae185a6e6c748a1b754d656c3844983","version_major":2,"version_minor":0}

{"model_id":"c5ae30746b2b41c8bcfe18fa85b64aa3","version_major":2,"version_minor":0}

{"model_id":"42aced9d5aa149a6bd511219992f3941","version_major":2,"version_minor":0}

{"model_id":"e3d2c6be2e8843b5b841ca5dae08d694","version_major":2,"version_minor":0}

{"model_id":"7d30a2412ae74bbdbd8b7132c79f7046","version_major":2,"version_minor":0}

{"model_id":"ae8917dc4c12440dad2c71aef2708799","version_major":2,"version_minor":0}

{"model_id":"78622ea99246474183f2b6286d1eaa30","version_major":2,"version_minor":0}

{"model_id":"52220edc9c6c4376973ae032e40e1662","version_major":2,"version_minor":0}

{"model_id":"f520acd8e79045158aafb9e886e26230","version_major":2,"version_minor":0}

{"model_id":"15ada16860bf438895ad2e13d04c6d15","version_major":2,"version_minor":0}

{"model_id":"d037186b60fb4ed3afb0fbb91a66cdec","version_major":2,"version_minor":0}
```

```
{"model_id":"ccddd8dcfb584026809ea4c0f4ad1188","version_major":2,"version_minor":0}

{"model_id":"80559351acc64dd2a9ba026810e1c09c","version_major":2,"version_minor":0}

{"model_id":"6821305d112e4e2e916dcb6b57dbd398","version_major":2,"version_minor":0}

{"model_id":"41fb4b9776c14961afc656534c2be599","version_major":2,"version_minor":0}

{"model_id":"db6a6bb8e6124af48c7a7bb263ccd246","version_major":2,"version_minor":0}

{"model_id":"788555f7ee944b0587354423afa41ab2","version_major":2,"version_minor":0}

{"model_id":"d428d1526f7947c4a28eaf1b3e6d9e86","version_major":2,"version_minor":0}

{"model_id":"0e64617d47564856b3ddca7251d3f88f","version_major":2,"version_minor":0}

{"model_id":"3ece9d8b1b4b4c94a446818dda792853","version_major":2,"version_minor":0}

{"model_id":"6a825da425d84315889000b6f2dd9111","version_major":2,"version_minor":0}

{"model_id":"a1e30ab4204245669fafdf2b1bf21c1e","version_major":2,"version_minor":0}

{"model_id":"5df4c965b3c0447aa3360a46d67b83d1","version_major":2,"version_minor":0}

{"model_id":"e48d24af01574c4a9ffa7347f28a8fcd","version_major":2,"version_minor":0}

{"model_id":"597203377e484e24b7576b703783b9fb","version_major":2,"version_minor":0}

{"model_id":"2b0327c84db345cfa6ba723eb979a402","version_major":2,"version_minor":0}

{"model_id":"b9fb2db8e05b4b5386073c59dbf77afc","version_major":2,"version_minor":0}

{"model_id":"f3a986598eb349b9af69b9ce04e2d3fd","version_major":2,"version_minor":0}

{"model_id":"64937a6710e64d449c36f8ac5dcdf862","version_major":2,"version_minor":0}
```

```
{"model_id":"fac9211b412c43bf832cb40a384e6d3f","version_major":2,"version_minor":0}

{"model_id":"31d4d07f0de94db48068c052859575e5","version_major":2,"version_minor":0}

{"model_id":"e7997b847bbc4195912ce85cd3dd868d","version_major":2,"version_minor":0}

{"model_id":"963297f958c647d5ad2869a538311c25","version_major":2,"version_minor":0}

{"model_id":"9b0acc3855a14b38b1dfc6fc6ddeb4f2","version_major":2,"version_minor":0}

{"model_id":"077a872248404613a20abdbb577b0ce9","version_major":2,"version_minor":0}

{"model_id":"64ba805fe2c84815bd44f28705cfe386","version_major":2,"version_minor":0}

{"model_id":"ded94a38a7ff4a2bb8a70dee7afcca96","version_major":2,"version_minor":0}

{"model_id":"5b2c4242cd14403bb37aac4d4720e6b6","version_major":2,"version_minor":0}

{"model_id":"b02d4466937442ca91e1931cfd1f287f","version_major":2,"version_minor":0}

{"model_id":"01adabb039514fcab2e54533899c48d8","version_major":2,"version_minor":0}

{"model_id":"93417eeb55584354b2fbdc4c57d5c63c","version_major":2,"version_minor":0}

{"model_id":"bb34bd832da842a79ebda742b9bb77da","version_major":2,"version_minor":0}

{"model_id":"66a7c1d2fff04048bdee77664be9b060","version_major":2,"version_minor":0}

{"model_id":"a91416afea514c40843400403266754d","version_major":2,"version_minor":0}

{"model_id":"94dab7a1f9f5400fa3e1c9d59f631754","version_major":2,"version_minor":0}

{"model_id":"7475136bbf44425bb5279f06cd6b5fc9","version_major":2,"version_minor":0}

{"model_id":"9d089169552e410fb8a90079eaadb618","version_major":2,"version_minor":0}
```

```
{"model_id":"062a48ba33db49cba8bc4f5f1c0cc688","version_major":2,"version_minor":0}

{"model_id":"5321d232dec64781ace3b4a5da0ca6d7","version_major":2,"version_minor":0}

{"model_id":"f08a78457a8841ac840614fdc9cdbf7d","version_major":2,"version_minor":0}

{"model_id":"81fef6d471f84b3e8da7b85cb3535253","version_major":2,"version_minor":0}

{"model_id":"420f253855cc451a9439e37c5f7262af","version_major":2,"version_minor":0}

{"model_id":"a22310d98bfa493e90c8526fbb83a19e","version_major":2,"version_minor":0}

{"model_id":"039ace12ad4a41c0aad5c533a3ac48d1","version_major":2,"version_minor":0}

{"model_id":"1bf865c0df1340fdad43786f25a29cfc","version_major":2,"version_minor":0}

{"model_id":"31a6c51a37d943dba6590899a3d0ff27","version_major":2,"version_minor":0}

{"model_id":"9e8386b6afcd4b55880f8ab465c33be9","version_major":2,"version_minor":0}

{"model_id":"ccaad56332c74fc6bf3ba68f9b78e2a3","version_major":2,"version_minor":0}

{"model_id":"2186e13bfae64df1a99be26c2d3ab543","version_major":2,"version_minor":0}

{"model_id":"8a1f66e58ebe45f1b9fdb9c1c4b09e27","version_major":2,"version_minor":0}

{"model_id":"2c2a03ca1f4e42e6a1415b7dc6203d69","version_major":2,"version_minor":0}

{"model_id":"448c57efd4634f07b1c30dc3006e9171","version_major":2,"version_minor":0}

{"model_id":"515698bf8ff344d6a1ab015b4446c843","version_major":2,"version_minor":0}

{"model_id":"131a446773da4c819bf1eced9a004dce","version_major":2,"version_minor":0}

{"model_id":"6b5577870e27493687a1177b4b78f669","version_major":2,"version_minor":0}
```

```
{"model_id":"3b2685e8fcf6498791b5f71e6a2bfafd","version_major":2,"version_minor":0}

{"model_id":"8bc0b5ccc22b482b8a6229ec37e47160","version_major":2,"version_minor":0}

{"model_id":"d365c77be3ce4b85b50ccd34900a6f9e","version_major":2,"version_minor":0}

{"model_id":"a28d9465c2c14bdeae40d369f9ecb69e","version_major":2,"version_minor":0}

{"model_id":"4c20646c22d842639bc56f48b7296a98","version_major":2,"version_minor":0}

{"model_id":"705712d6512840faa0a6eaf0669dcaf8","version_major":2,"version_minor":0}

{"model_id":"ea5161142eed415c90cc784074c77e39","version_major":2,"version_minor":0}

{"model_id":"9e09b3bcc0c04aa9bc1383d843c2e927","version_major":2,"version_minor":0}

{"model_id":"ef40017eef7245f18ed8afb0966ce61b","version_major":2,"version_minor":0}

{"model_id":"7562d0f0c3c74a46b5740d84b6e91286","version_major":2,"version_minor":0}

{"model_id":"0f9ee3ef756e4e5b816af210352ae14e","version_major":2,"version_minor":0}

{"model_id":"a87aa4a6a2034a159aadc537bd61130d","version_major":2,"version_minor":0}

{"model_id":"df4ae15ab212494e9df9960b6ef8c597","version_major":2,"version_minor":0}

{"model_id":"421c8b8b3b55488d85fd0f8ad07e1ba6","version_major":2,"version_minor":0}

{"model_id":"87fc4e4bbdf94cc5bf9dfb33836278eb","version_major":2,"version_minor":0}

{"model_id":"d04586801c164e34b562c8754c12b50c","version_major":2,"version_minor":0}

{"model_id":"48cea1fde5af40d18fb5cc09ad9ab79b","version_major":2,"version_minor":0}

{"model_id":"bd139fca008346b1b46c0ac8409543d0","version_major":2,"version_minor":0}
```

```
{"model_id":"c7d75d488aa44963a20f48323f1d00a9","version_major":2,"version_minor":0}

{"model_id":"0aed7b076f4c4a19918a904d26e3d361","version_major":2,"version_minor":0}

{"model_id":"a305f6d93a4f4382ab25335ae3384ae1","version_major":2,"version_minor":0}

{"model_id":"3514be939a624898be698feec61a4af8","version_major":2,"version_minor":0}

{"model_id":"5900e8b5e12943588ff108fbcbc20e68","version_major":2,"version_minor":0}

{"model_id":"dfc2193084c44f49b1362f5e8abb04c8","version_major":2,"version_minor":0}

{"model_id":"bc0b20f502a74487893ae3471c64277c","version_major":2,"version_minor":0}

{"model_id":"1c73ab10454b42f5bf0e5f67bda88834","version_major":2,"version_minor":0}

{"model_id":"e3b2780dffc64c168ccc029435cabaae","version_major":2,"version_minor":0}

{"model_id":"c5d94f1866ef483d8c883bdd410af39f","version_major":2,"version_minor":0}

{"model_id":"e13a8fa8411543cc8c6ac0504032740f","version_major":2,"version_minor":0}

{"model_id":"9bbfd4fcb2d8476096d47a5921dcc4d4","version_major":2,"version_minor":0}

{"model_id":"e4e69cae96364f748e7f7da24a630aeb","version_major":2,"version_minor":0}

{"model_id":"3ceafd3c23f64bd7921482c8813aae1f","version_major":2,"version_minor":0}

{"model_id":"12f734ce8248453a86309855799b007a","version_major":2,"version_minor":0}

{"model_id":"c7193b11e2574bfbaf0da356dce46438","version_major":2,"version_minor":0}

{"model_id":"7a3ef782af6446e191e82f4fd8aba005","version_major":2,"version_minor":0}

{"model_id":"b9672fe3757948e3bd2e199f3297f8c8","version_major":2,"version_minor":0}
```

```
{"model_id":"e717aefe98e14db4b36a34ba0c154842","version_major":2,"version_minor":0}

{"model_id":"ddf49d3987174dfbadba9d3304d2657e","version_major":2,"version_minor":0}

{"model_id":"fdfb2423a4b149798c5eab4beb5ce77d","version_major":2,"version_minor":0}

{"model_id":"94cfb06921b340a3a17805a30540849d","version_major":2,"version_minor":0}

{"model_id":"20b6b8900f1a49b6afe18fd5476bca00","version_major":2,"version_minor":0}

{"model_id":"481a668fd09946bfa601158699a69e9c","version_major":2,"version_minor":0}

{"model_id":"410fd6c100904bcf995675056d52d3a3","version_major":2,"version_minor":0}

{"model_id":"b6a8bfd2bf3f416d871aa3bd1efce7b9","version_major":2,"version_minor":0}

{"model_id":"bdf6fe8d87214bb994dbe4026f0659bf","version_major":2,"version_minor":0}

{"model_id":"d299b1f412f34ca3ba2f39facbf7e24e","version_major":2,"version_minor":0}

{"model_id":"2471be978e7a43969919b7b8203b9a10","version_major":2,"version_minor":0}

{"model_id":"27234bbe622e4f4c960cb05607ed3b8e","version_major":2,"version_minor":0}

{"model_id":"98b1ebe012f546f9b04fbd43328e6d9c","version_major":2,"version_minor":0}

{"model_id":"2bf879f34e5e47398f80205489d3d4e5","version_major":2,"version_minor":0}

{"model_id":"fc3affee6b9b4370b2d128047a665d7d","version_major":2,"version_minor":0}

{"model_id":"a860947b8c3544bf86e85980de6d526f","version_major":2,"version_minor":0}

{"model_id":"a730e83702f54aa698e48157102befb3","version_major":2,"version_minor":0}

{"model_id":"0c38d4fc05b24c1c970ee9cd9f74add8","version_major":2,"version_minor":0}
```

```
{"model_id":"369eb195241744e481953b8c40693520","version_major":2,"version_minor":0}

{"model_id":"a7d646ba38d74580a3e86ca9f666d90f","version_major":2,"version_minor":0}

{"model_id":"9dbdbcde9e1d4717b95be9930a08d974","version_major":2,"version_minor":0}

{"model_id":"3db05c30e92744789d32550d37195989","version_major":2,"version_minor":0}

{"model_id":"eeb49b52d0e44ca0bfecb462a4acdba9","version_major":2,"version_minor":0}

{"model_id":"ec461e8b2232413ea0238c5a329bea45","version_major":2,"version_minor":0}

{"model_id":"5d9f169206e04f71b0f51444bf7eca63","version_major":2,"version_minor":0}

{"model_id":"5bb4ee3d7dad4a92af04723da69a0848","version_major":2,"version_minor":0}

{"model_id":"5502660ae30440b6883f1b8522eeb16d","version_major":2,"version_minor":0}

{"model_id":"4e33336410804f3c928aa411f7da800f","version_major":2,"version_minor":0}

{"model_id":"c48b2f4dd24642ca8724fbdcdf608536","version_major":2,"version_minor":0}

{"model_id":"42be6f099a514c1895cf88ec65f5b5af","version_major":2,"version_minor":0}

{"model_id":"9dc06faddf7448ea9405d5e37fc1b3b5","version_major":2,"version_minor":0}

{"model_id":"965c0f4659064e11b48b5489083a6ea8","version_major":2,"version_minor":0}

{"model_id":"5a3d1d76c7194957bb87dcbdcd2998cf","version_major":2,"version_minor":0}

{"model_id":"88835c02b093467e96187360b0891f54","version_major":2,"version_minor":0}

{"model_id":"9f1660db9d3544dfbbd872f43836e77e","version_major":2,"version_minor":0}

{"model_id":"3b7aba8bb6774e0a9f15089e095b1832","version_major":2,"version_minor":0}
```

```
{"model_id":"d156c3bd6564430a9483174584939f0f","version_major":2,"version_minor":0}

{"model_id":"d1f369035433436b949234a18b00b938","version_major":2,"version_minor":0}

{"model_id":"67cc089595684aadbd35f7c46bb21351","version_major":2,"version_minor":0}

{"model_id":"360344b2512d4341a2f27bd7bfb67dea","version_major":2,"version_minor":0}

{"model_id":"e1084faa2f2c4ce5a6942d55135a7189","version_major":2,"version_minor":0}

{"model_id":"1a888b7f5f1c40f08c5b37289bba5fc3","version_major":2,"version_minor":0}

{"model_id":"51163c080ecb4471bedbd4fe303d3262","version_major":2,"version_minor":0}

{"model_id":"89399642bd2a48daa97737b5f2b2f92b","version_major":2,"version_minor":0}

{"model_id":"8e59263d773249f1847cf6d4ecee01a9","version_major":2,"version_minor":0}

{"model_id":"2183b76f041f48c49722b3efc605f000","version_major":2,"version_minor":0}

{"model_id":"509cda571402473e8758a5179f705e3b","version_major":2,"version_minor":0}

{"model_id":"12c3828301604dd29729ae448dbfb597","version_major":2,"version_minor":0}

{"model_id":"816ff2e29f4d42ea820f15e8c39af3dc","version_major":2,"version_minor":0}

{"model_id":"d7487254daf5428dbcc7bced10654ec6","version_major":2,"version_minor":0}

{"model_id":"778bd1a1ce6047dabe49d7aa377d09ad","version_major":2,"version_minor":0}

{"model_id":"2751752d6e77420889e341f8cb732583","version_major":2,"version_minor":0}

{"model_id":"3b208c169c0c41d599638eb1de8f4754","version_major":2,"version_minor":0}

{"model_id":"c15cc0ba9b064b8a9c13aac967ab2706","version_major":2,"version_minor":0}
```

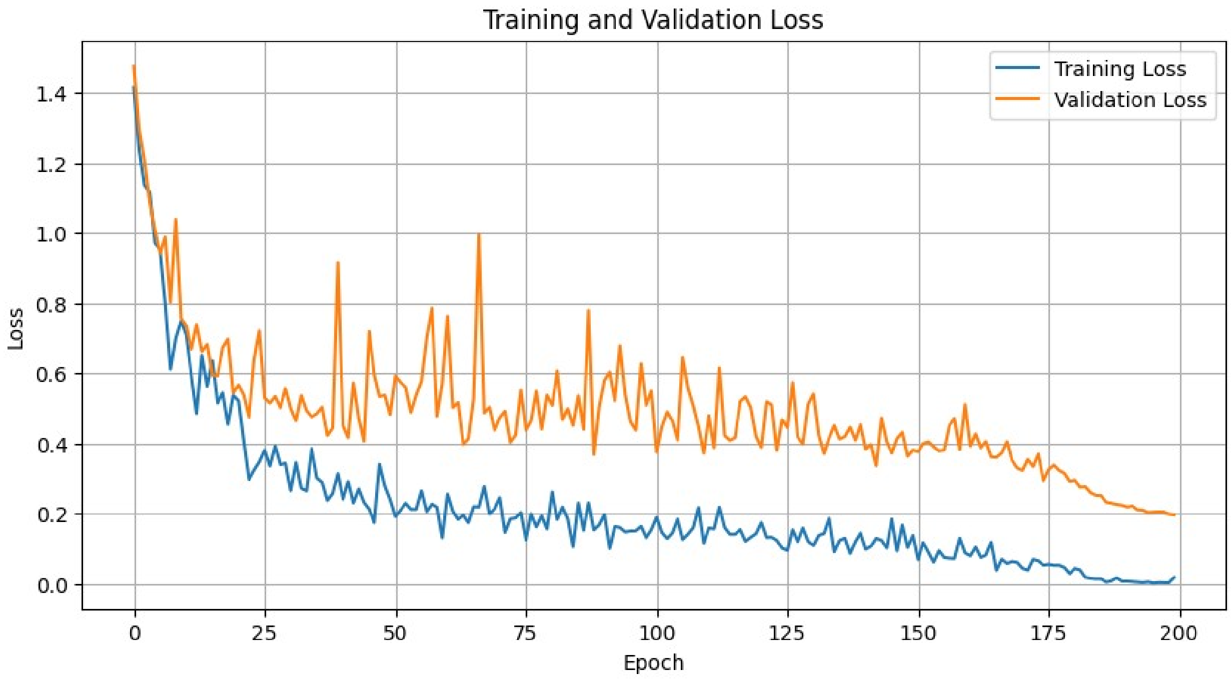
```
{"model_id":"a1ac2425e9154450aead7f40827c9b72","version_major":2,"version_minor":0}

{"model_id":"9266480f5a95474991c0112cfd423d78","version_major":2,"version_minor":0}

{"model_id":"40c4275bc61d495a82f490d099a6e557","version_major":2,"version_minor":0}

{"model_id":"755c5b3e140a42f5abe30af039111abf","version_major":2,"version_minor":0}

{"model_id":"32b931cdba20482bb3d9c549465e95ab","version_major":2,"version_minor":0}

{"model_id":"8e84596349a04002a331d21ebd2953d3","version_major":2,"version_minor":0}

{"model_id":"acb58cf378234dd38bed5aee48d9d710","version_major":2,"version_minor":0}

{"model_id":"4d3305cd13344ee5981bec1c8264d4f7","version_major":2,"version_minor":0}

{"model_id":"9cbe571a2bc74421b751a83b1fc52475","version_major":2,"version_minor":0}

{"model_id":"b96a05839da54fa9a11f897685243766","version_major":2,"version_minor":0}

{"model_id":"8c4fc5e4691b4bc5993338ac831329e2","version_major":2,"version_minor":0}

{"model_id":"f0a182e1f27b47d893d54de2e9095949","version_major":2,"version_minor":0}

{"model_id":"ab9f4033002d46d5b955a7ff7361ea59","version_major":2,"version_minor":0}

{"model_id":"8f3741713bec4a1b8dc1c40e84b72664","version_major":2,"version_minor":0}

{"model_id":"02df528b18e446389322f647ed43cdbf","version_major":2,"version_minor":0}

{"model_id":"8d2ba8da38694acc91a9d56563c789e0","version_major":2,"version_minor":0}

{"model_id":"e48e39b9e0514aa5a2bb5d118ec09141","version_major":2,"version_minor":0}

{"model_id":"ca3366a9a0a843a9bf0391c9774a3f64","version_major":2,"version_minor":0}
```

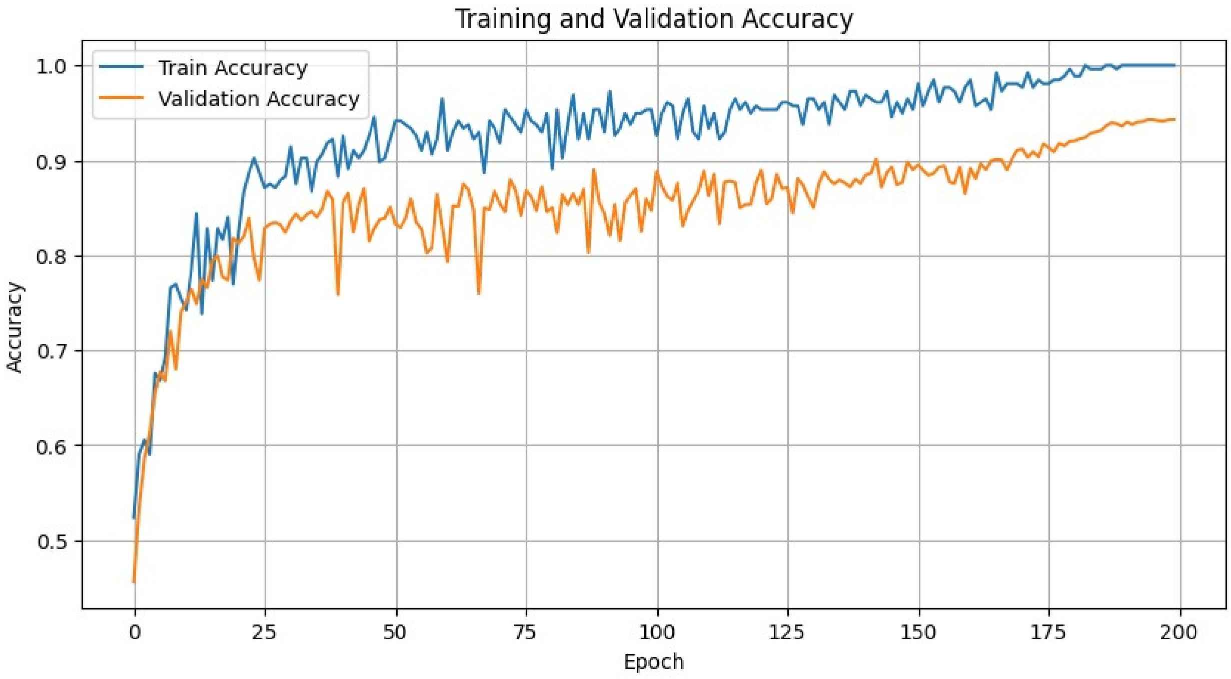
```
{"model_id":"ecc9f969df4442619f84505f202992fe","version_major":2,"vers
ion_minor":0}

{"model_id":"101c5b191d1242c5963c09cc11102788","version_major":2,"vers
ion_minor":0}

{"model_id":"5ceb68184c55495e8cb375c56276d202","version_major":2,"vers
ion_minor":0}

{"model_id":"4d9c4ded632544f79cc450cbd1724227","version_major":2,"vers
ion_minor":0}

{"model_id":"da8e1c1f305e43d48e07e344d4f76fbf","version_major":2,"vers
ion_minor":0}

{"model_id":"5058dda8e3dc4922b7fd2fe6ac8aa322","version_major":2,"vers
ion_minor":0}

{"model_id":"c846966d15084352b249fbb4657049f4","version_major":2,"vers
ion_minor":0}

INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=200` reached.
```

```
torch.save( model.state_dict(), r'/200_model.pth' )
```

The training and validation accuracy can be plot based on the logfile.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
Traning_accuracy_final =  0
validation_accuracy_final = 0
def plot_training_metrics(csv_file_path):
    # Load the CSV file into a DataFrame
    df = pd.read_csv(csv_file_path)
    # Remove rows with empty values in the specified columns
    val_acc_series = df['val_acc'].values
    val_acc_array = val_acc_series[~np.isnan(val_acc_series)]
    train_acc_series = df['train_acc'].values
    train_acc_array = train_acc_series[~np.isnan(train_acc_series)]
    val_loss_series = df['val_loss'].values
    val_loss_array = val_loss_series[~np.isnan(val_loss_series)]
    epoch_series = df['epoch'].values
    indices_train_not_nan = np.where(df['train_acc'].notna())[0]
    indices_epoch_not_nan = np.where(df['epoch'].notna())[0]

    common_indices =
list(set(indices_train_not_nan).intersection(indices_epoch_not_nan))
    answer =[0]*len(val_loss_array)
    for epoch_i in range(len(val_loss_array)):
        for id, real_id in enumerate(common_indices):
            if(epoch_series[real_id] == epoch_i):
                answer[epoch_i] = real_id
    train_loss_series = df['train_loss'].values
    train_loss_array = train_loss_series[~np.isnan(train_loss_series)]
    selected_train_acc_array = train_acc_series[answer]
    selected_train_loss_array = train_loss_series[answer]
    epoch = list(range(len(selected_train_loss_array)))
    # Plot training and validation loss
    plt.figure(figsize=(10, 5))
    plt.plot(epoch, selected_train_loss_array, label='Training Loss')
    plt.plot(epoch, val_loss_array , label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.grid()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    epoch = list(range(len(val_acc_array)))
    # Plot training and validation accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(epoch, selected_train_acc_array, label='Train Accuracy')
    plt.plot(epoch, val_acc_array, label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.grid()
    plt.legend()
    plt.show()
    traning_accuracy_final =  selected_train_acc_array[-1]
    validation_accuracy_final = val_acc_array[-1]
    return traning_accuracy_final , validation_accuracy_final
traning_accuracy_final, validation_accuracy_fina =
plot_training_metrics(r"/logs/lightning_logs/version_0/metrics.csv")
```





```
from google.colab import drive
drive.mount('/content/drive')
```

We got the accuracy of 0.94 with test dataset from CIFAR-10 Dataset.

```
trainer.test(model, datamodule=cifar10_dm)
```

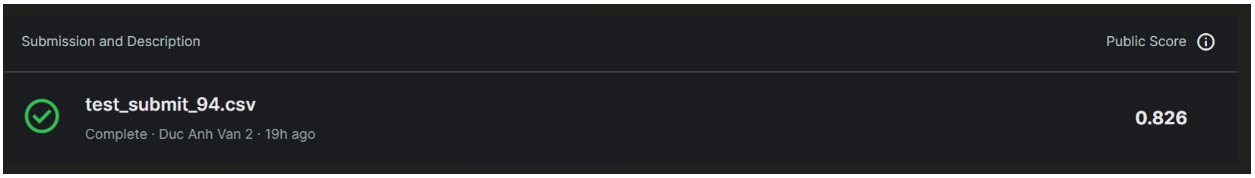
```
Files already downloaded and verified
Files already downloaded and verified

INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 -
CUDA_VISIBLE_DEVICES: [0]
```

```
{"model_id":"3e1b71abbc884f99bf020589984cab03","version_major":2,"vers
ion_minor":0}
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃        Test metric         ┃        DataLoader 0        ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│          test_acc          │     0.940500020980835      │
│         test_loss          │    0.22141557931900024     │
└────────────────────────────┴────────────────────────────┘
```

Evaluating the Model with the unlabled dataset.

We reuse the model saved from the previous one to predict the label of the custom dataset. Compared with the result in Kaggle, we achieved around 83% of accuracy.



Submission and Description | Public Score

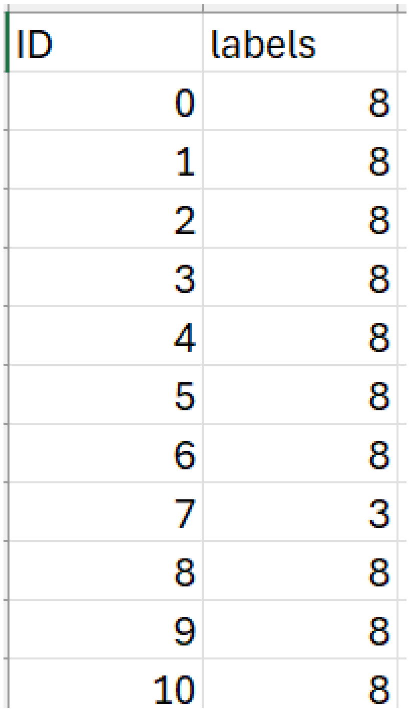
✓ test_submit_94.csv
Complete · Duc Anh Van 2 · 19h ago | 0.826

```python
from torchvision.transforms import v2
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

model.load_state_dict(torch.load('/200_model.pth'))

test_for_submit = unpickle('/cifar_test_nolabels.pkl')
data_for_submit = test_for_submit[b'data']/255.0
data_for_submit_tensor = torch.tensor(data_for_submit.reshape(-1, 3,
32, 32)).float()  # Adjusting for channel position

test_transforms = v2.Compose(
    [
        v2.ToImage(),
        v2.ToDtype(torch.float32, scale=True),
        cifar10_normalization(),
    ]
)
data_for_submit_tensor = test_transforms(data_for_submit_tensor)
data_for_submit_tensor = data_for_submit_tensor.to(device)
```

```
model.model.eval() # evaluate
model.model.to(device)
y_predict = []
for i in range(0, 10000, 200):
    logits = model.model(data_for_submit_tensor[i:i+200])
    preds = torch.argmax(logits, dim=1).cpu().numpy()
#     print(preds)
    y_predict.append(preds)


my_answer = np.array(y_predict).flatten()

id = list(range(10000))
value = my_answer

# create a dictionary with the three lists
dict = {'ID': id, 'labels': value}

# create a Pandas DataFrame from the dictionary
df = pd.DataFrame(dict)

# write the DataFrame to a CSV file
df.to_csv('/test_submit_94.csv', index=False)
df[['labels']].value_counts()
# df.
```

```
labels
9         1089
5         1082
3         1072
8         1054
7         1031
4          960
6          946
0          931
1          923
2          912
Name: count, dtype: int64
```

Here's an image of how the csv file looks like



ID	labels
0	8
1	8
2	8
3	8
4	8
5	8
6	8
7	3
8	8
9	8
10	8

Summary of our ResNet Model Results

```
print(f"1. Training Data Accuracy: {0.99:.2f} \n"
      f"2. Validation Data Accuracy: {0.94:.2f}\n"
      f"3. No-Label Data Accuracy: {0.828:.2f}\n"
      f"4. Number of Parameters for our Model: {4_983_306:,}")

1. Training Data Accuracy: 0.99
2. Validation Data Accuracy: 0.94
3. No-Label Data Accuracy: 0.83
4. Number of Parameters for our Model: 4,983,306
```

Lastly, we plot some images from the unlabeled dataset and their predicted class to visually inspect our model's performance.

```
import pickle
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

def unpickle(file):
    with open(file, 'rb') as fo:
        data = pickle.load(fo, encoding='bytes')
    return data

image_data = unpickle('/cifar_test_nolabels.pkl')
```

```
df = pd.read_csv('/test_submit_94.csv')
image_ids = df['ID']
prediction_labels = df['labels']

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
predicted_classes = [classes[label] for label in prediction_labels]

N = 25
random_indices = np.random.choice(len(image_data[b'data']), N,
replace=False)

fig, axes = plt.subplots(N//5, 5, figsize=(15, 7))
for i, idx in enumerate(random_indices):
    ax = axes[i // 5, i % 5]
    reshaped_image = image_data[b'data'][idx].reshape(3, 32, 32)
    ax.imshow(np.transpose(reshaped_image, (1, 2, 0)))
    ax.set_title(f"Prediction: {predicted_classes[idx]}")
    ax.axis('off')

plt.tight_layout()
plt.show()
```