```
%cd /content/drive/MyDrive/ImageBind finetune/ImageBind-LoRA
```

```
/content/drive/MyDrive/ImageBind finetune/ImageBind-LoRA
```

```
!pip install -r requirements.txt --quiet
!pip install pytorch_lightning --quiet
```

```
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 890.1/890.1 MB 1.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.3/24.3 MB 59.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 95.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.0/510.0 kB 51.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.4/54.4 kB 8.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.2/43.2 kB 6.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.2/50.2 kB 7.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.6/13.6 MB 82.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.2/42.2 kB 6.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.1/7.1 MB 61.6 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 107.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 57.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 849.3/849.3 kB 969.1 kB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 557.1/557.1 MB 2.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.1/317.1 MB 2.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.0/21.0 MB 53.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.4/66.4 kB 9.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 80.8/80.8 kB 12.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.5/55.5 kB 8.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 55.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.9/71.9 kB 10.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 868.8/868.8 kB 65.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.4/62.4 kB 9.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.9/129.9 kB 23.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 812.3/812.3 kB 68.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.3/34.3 MB 51.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 230.0/230.0 kB 32.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.9/268.9 kB 33.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 74.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.1/5.1 MB 17.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 78.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 105.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.3/64.3 kB 10.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.4/58.4 kB 2.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.2/139.2 kB 19.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 9.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 812.3/812.3 kB 67.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 812.3/812.3 kB 62.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 812.2/812.2 kB 64.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 802.3/802.3 kB 62.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 116.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.7/82.7 kB 12.2 MB/s eta 0:00:00
  Building wheel for pytorchvideo (setup.py) ... done
  Building wheel for fvcore (setup.py) ... done
  Building wheel for iopath (setup.py) ... done
  Building wheel for mayavi (pyproject.toml) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are ir
torchtext 0.18.0 requires torch>=2.3.0, but you have torch 1.13.0 which is incompatible.
```

```
import logging
logging.basicConfig(level=logging.INFO, force=True)
```

```
import os
num_workers = os.cpu_count()
```

```
from pytorch_lightning import seed_everything
seed_everything(43, workers=True)
```

```
INFO:lightning_fabric.utilities.seed:Seed set to 43
43
```

```
import os
from torch.utils.data import Dataset
from sklearn.model_selection import train_test_split
```

```
import data
```

Resources

You are not subscribed. Learn more

Available: 71.5 compute units

Usage rate: approximately 0 per hour

You have 0 active sessions.

Manage sessions

Want more memory and disk space?

Upgrade to Colab Pro

Not connected to runtime.

Change runtime type

```
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/_functional_video.py:6: UserWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/_transforms_video.py:22: UserWarnir
  warnings.warn(
```

```
import torch
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, ConcatDataset
import torchvision
from torchvision import transforms

from models import imagebind_model
from models import lora as LoRA
from models.imagebind_model import ModalityType, load_module, save_module
```

```
import pytorch_lightning as L
from pytorch_lightning import Trainer, seed_everything
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning import loggers as pl_loggers
```

```
self_contrast = False
batch_size = 8
num_workers= os.cpu_count()
lora_modality_names_123 = ["vision", "audio", "text"]
LOG_ON_STEP = False
LOG_ON_EPOCH = True
lora= True
full_model_checkpointing = False
full_model_checkpoint_dir="./.checkpoints/full"
lora_checkpoint_dir="./.checkpoints/lora"
device_name="cuda:0" if torch.cuda.is_available() else "cpu"
max_epochs = 5
gradient_clip_val=1.0
loggers = None
linear_probing = False
```

```
class ImageBindTrain(L.LightningModule):
    def __init__(self, lr=5e-4, weight_decay=1e-4, max_epochs=500, batch_size=32, num_workers=4, seed=
                 self_contrast=False, temperature=0.07,  momentum_betas=(0.9, 0.95),
                 lora=False, lora_rank=4, lora_checkpoint_dir="./.checkpoints/lora",
                 lora_layer_idxs=None, lora_modality_names=None,
                 linear_probing=False
                 ):
        super().__init__()
        assert not (linear_probing and lora), \
            "Linear probing is a subset of LoRA training procedure for ImageBind. " \
            "Cannot set both linear_probing=True and lora=True. " \
            "Linear probing stores params in lora_checkpoint_dir"
        self.save_hyperparameters()

        # Load full pretrained ImageBind model
        self.model = imagebind_model.imagebind_huge(pretrained=True)
        if lora:
            for modality_preprocessor in self.model.modality_preprocessors.children():
                modality_preprocessor.requires_grad_(False)
            for modality_trunk in self.model.modality_trunks.children():
                modality_trunk.requires_grad_(False)

            self.model.modality_trunks.update(LoRA.apply_lora_modality_trunks(self.model.modality_trur
                                                                              layer_idxs=lora_layer_id
                                                                              modality_names=lora_moda
            LoRA.load_lora_modality_trunks(self.model.modality_trunks, checkpoint_dir=lora_checkpoint_

            # Load postprocessors & heads
            load_module(self.model.modality_postprocessors, module_name="postprocessors",
                        checkpoint_dir=lora_checkpoint_dir)
            load_module(self.model.modality_heads, module_name="heads",
                        checkpoint_dir=lora_checkpoint_dir)
        elif linear_probing:
            for modality_preprocessor in self.model.modality_preprocessors.children():
                modality_preprocessor.requires_grad_(False)
            for modality_trunk in self.model.modality_trunks.children():
                modality_trunk.requires_grad_(False)
            for modality_postprocessor in self.model.modality_postprocessors.children():
                modality_postprocessor.requires_grad_(False)

            load_module(self.model.modality_heads, module_name="heads",
                        checkpoint_dir=lora_checkpoint_dir)
            for modality_head in self.model.modality_heads.children():
                modality_head.requires_grad_(False)
                final_layer = list(modality_head.children())[-1]
                final_layer.requires_grad_(True)

    def configure_optimizers(self):
        optimizer = optim.AdamW(self.parameters(), lr=self.hparams.lr, weight_decay=self.hparams.weigh
                                betas=self.hparams.momentum_betas)
        lr_scheduler = optim.lr_scheduler.CosineAnnealingLR(
            optimizer, T_max=self.hparams.max_epochs, eta_min=self.hparams.lr / 50
        )
        return [optimizer], [lr_scheduler]

    def info_nce_loss(self, batch, mode="train"):
      data_a, class_a, data_b, class_b, data_c, class_c = batch

      # class_a is always "vision" according to ImageBind
      feats_a = [self.model({class_a[0]: data_a_i}) for data_a_i in data_a]
      feats_a_tensor = torch.cat([list(dict_.values())[0] for dict_ in feats_a], dim=0)

      # class_b is always "audio"
      feats_b = [self.model({class_b[0]: data_b_i}) for data_b_i in data_b]
      feats_b_tensor = torch.cat([list(dict_.values())[0] for dict_ in feats_b], dim=0)

      # class_c is always "text"
      feats_c = [self.model({class_c[0]: data_c_i}) for data_c_i in data_c]
      feats_c_tensor = torch.cat([list(dict_.values())[0] for dict_ in feats_c], dim=0)

      if self.hparams.self_contrast:
          feats_a_b_c_tensor = torch.cat([feats_a_tensor.chunk(3)[0], feats_b_tensor, feats_c_tensor],
          feats_tensors = [feats_a_tensor, feats_a_b_c_tensor]
          temperatures = [1, self.hparams.temperature]
          contrast = ["self", "cross"]
      else:
          feats_a_b_c_tensor = torch.cat([feats_a_tensor, feats_b_tensor, feats_c_tensor], dim=0)
          feats_tensors = [feats_a_b_c_tensor]
          temperatures = [self.hparams.temperature]
          contrast = ["cross"]

      dual_nll = False
```

```
    for feats_idx, feats_tensor in enumerate(feats_tensors):
        cos_sim = F.cosine_similarity(feats_tensor[:, None, :], feats_tensor[None, :, :], dim=-1)
        self_mask = torch.eye(cos_sim.shape[0], dtype=torch.bool, device=cos_sim.device)
        cos_sim.masked_fill_(self_mask, -9e15)
        #pos_mask = self_mask.roll(shifts=cos_sim.shape[0] // 3, dims=0)
        pos_mask_1 = self_mask.roll(shifts=cos_sim.shape[0]//3, dims=0)
        pos_mask_2 = self_mask.roll(shifts=2 * cos_sim.shape[0]//3, dims=0)
        pos_mask = pos_mask_1 | pos_mask_2
        cos_sim = cos_sim / temperatures[feats_idx]
        nll = -cos_sim[pos_mask] + torch.logsumexp(cos_sim, dim=-1)
        nll = nll.mean()
        if not dual_nll:
            dual_nll = nll
        else:
            dual_nll += nll
            dual_nll /= 2
        self.log(mode + "_loss_" + contrast[feats_idx], nll, prog_bar=True,
                on_step=LOG_ON_STEP, on_epoch=LOG_ON_EPOCH, batch_size=self.hparams.batch_size)
        comb_sim = torch.cat(
            [cos_sim[pos_mask][:, None], cos_sim.masked_fill(pos_mask, -9e15)],
            dim=-1,
        )
        sim_argsort = comb_sim.argsort(dim=-1, descending=True).argmin(dim=-1)
        self.log(mode + "_acc_top1", (sim_argsort == 0).float().mean(), prog_bar=True,
                on_step=LOG_ON_STEP, on_epoch=LOG_ON_EPOCH, batch_size=self.hparams.batch_size)
        self.log(mode + "_acc_top5", (sim_argsort < 5).float().mean(), prog_bar=True,
                on_step=LOG_ON_STEP, on_epoch=LOG_ON_EPOCH, batch_size=self.hparams.batch_size)
        self.log(mode + "_acc_mean_pos", 1 + sim_argsort.float().mean(), prog_bar=True,
                on_step=LOG_ON_STEP, on_epoch=LOG_ON_EPOCH, batch_size=self.hparams.batch_size)

    self.log(mode + "_loss", dual_nll, prog_bar=True,
            on_step=LOG_ON_STEP, on_epoch=LOG_ON_EPOCH, batch_size=self.hparams.batch_size)
    return dual_nll


def training_step(self, batch, batch_idx):
    return self.info_nce_loss(batch, mode="train")

def validation_step(self, batch, batch_idx):
    self.info_nce_loss(batch, mode="val")

def on_validation_epoch_end(self):
    if self.hparams.lora:
        # Save LoRA checkpoint
        LoRA.save_lora_modality_trunks(self.model.modality_trunks, checkpoint_dir=self.hparams.lor
        # Save postprocessors & heads
        save_module(self.model.modality_postprocessors, module_name="postprocessors",
                    checkpoint_dir=self.hparams.lora_checkpoint_dir)
        save_module(self.model.modality_heads, module_name="heads",
                    checkpoint_dir=self.hparams.lora_checkpoint_dir)
    elif self.hparams.linear_probing:
        # Save postprocessors & heads
        save_module(self.model.modality_heads, module_name="heads",
                    checkpoint_dir=self.hparams.lora_checkpoint_dir)
```

```
class ImageAudioDataset(Dataset):
    def __init__(self, root_dir, transform=None, split='train', train_size=0.9, random_seed=42, device
        self.root_dir = root_dir
        self.transform = transform
        self.device = device

        self.classes = [d for d in os.listdir(os.path.join(root_dir, 'images')) if os.path.isdir(os.pa
        self.class_to_idx = {cls: idx for idx, cls in enumerate(self.classes)}

        self.image_paths = []
        self.audio_paths = []
        for cls in self.classes:
            cls_image_dir = os.path.join(root_dir, 'images', cls)
            cls_audio_dir = os.path.join(root_dir, 'audio', cls)
            for filename in os.listdir(cls_image_dir):
                filename_temp=filename[:-4]
                if filename_temp[:-4] == ".DS_S":
                  continue
                self.image_paths.append((os.path.join(cls_image_dir, filename_temp+".jpg"), cls))
                self.audio_paths.append((os.path.join(cls_audio_dir, filename_temp+".wav"), cls))

        # Split dataset
        self.train_image_paths, self.test_image_paths = train_test_split(self.image_paths, train_size=
        self.train_audio_paths, self.test_audio_paths = train_test_split(self.audio_paths, train_size=

        if split == 'train':
            self.image_paths = self.train_image_paths
            self.audio_paths = self.train_audio_paths
        elif split == 'test':
            self.image_paths = self.test_image_paths
            self.audio_paths = self.test_audio_paths
        else:
            raise ValueError(f"Invalid split argument. Expected 'train' or 'test', got {split}")

    def __len__(self):
        return min(len(self.image_paths), len(self.audio_paths))

    def __getitem__(self, index):
        img_path, class_text = self.image_paths[index]
        audio_path, _ = self.audio_paths[index]
        # Load and transform image
        images = data.load_and_transform_vision_data([img_path], self.device, to_tensor=False)
        if self.transform is not None:
            image = images[0]
            images = self.transform(image)

        # Load and transform audio
        audios = data.load_and_transform_audio_data([audio_path], self.device)

        # Load and transform text
        texts = data.load_and_transform_text([class_text], self.device)

        return images, ModalityType.VISION, audios, ModalityType.AUDIO, texts, ModalityType.TEXT
```

```
contrast_transforms = transforms.Compose(
        [
            transforms.RandomHorizontalFlip(),
            transforms.RandomResizedCrop(size=224),
            transforms.RandomApply([transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.
                                    p=0.8),
            transforms.RandomGrayscale(p=0.2),
            transforms.GaussianBlur(kernel_size=9),
            transforms.ToTensor(),
            transforms.Normalize(
                mean=(0.48145466, 0.4578275, 0.40821073),
                std=(0.26862954, 0.26130258, 0.27577711),
            ),
        ]
    )

class ContrastiveTransformations:
    def __init__(self, base_transforms, n_views=2):
        self.base_transforms = base_transforms
        self.n_views = n_views

    def __call__(self, x):
        return [self.base_transforms(x) for _ in range(self.n_views)]
```

```
train_datasets = []
test_datasets = []
```

```
train_datasets.append(ImageAudioDataset(
            root_dir=os.getcwd()+"/new_data/", split="train",
            transform=ContrastiveTransformations(contrast_transforms,
                                                 n_views=2 if self_contrast else 1)))
```

```
test_datasets.append(ImageAudioDataset(
            root_dir=os.getcwd()+"/new_data/", split="test",
            transform=ContrastiveTransformations(contrast_transforms,
                                                 n_views=2 if self_contrast else 1)))
```

```
train_dataset = train_datasets[0]
test_dataset = test_datasets[0]
```

```
train_loader = DataLoader(
        train_dataset,
        batch_size=batch_size,
        shuffle=True,
        drop_last=True,
        pin_memory=False,
        num_workers=num_workers,
    )
val_loader = DataLoader(
        test_dataset,
        batch_size=batch_size,
        shuffle=False,
        drop_last=False,
        pin_memory=False,
        num_workers=num_workers,
    )
```

```
lora_layer_idxs = {}
lora_modality_names = []
modalities = ["vision", "text", "audio", "thermal", "depth", "imu"]
for modality_name in lora_modality_names_123:
    if modality_name in modalities:
        modality_type = getattr(ModalityType, modality_name.upper())
        #lora_layer_idxs[modality_type] = getattr(args, f'lora_layer_idxs_{modality_name}', None)
        # if not lora_layer_idxs[modality_type]:
        #     lora_layer_idxs[modality_type] = None
        lora_layer_idxs[modality_type] = None
        lora_modality_names.append(modality_type)
    else:
```