

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

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

✓ Step 1: Download a dataset and preview images

```
!tar -xvf '/content/cifar100.tar'
```

↗

```
./cifar100/test/kangaroo_4347.jpg
./cifar100/test/caterpillar_529.jpg
./cifar100/test/keyboard_4800.jpg
./cifar100/test/porcupine_7513.jpg
./cifar100/test/orchid_2459.jpg
./cifar100/test/train_2241.jpg
./cifar100/test/crab_4790.jpg
./cifar100/test/oak_tree_7631.jpg
./cifar100/test/tank_1007.jpg
./cifar100/test/flatfish_8893.jpg
./cifar100/test/skunk_9341.jpg
./cifar100/test/apple_3775.jpg
./cifar100/test/chair_1740.jpg
./cifar100/test/bee_964.jpg
./cifar100/test/plate_8278.jpg
./cifar100/test/man_5831.jpg
./cifar100/test/baby_3577.jpg
./cifar100/test/bus_1616.jpg
./cifar100/test/cup_907.jpg
./cifar100/test/willow_tree_6974.jpg
./cifar100/test/possum_3690.jpg
./cifar100/test/butterfly_680.jpg
./cifar100/test/woman_3469.jpg
./cifar100/test/wolf_1752.jpg
./cifar100/test/lamp_355.jpg
./cifar100/test/crocodile_1769.jpg
./cifar100/test/sweet_pepper_2069.jpg
./cifar100/test/lawn_mower_3230.jpg
./cifar100/test/kangaroo_2035.jpg
./cifar100/test/hamster_7327.jpg
./cifar100/test/rocket_6867.jpg
./cifar100/test/lobster_9844.jpg
./cifar100/test/road_733.jpg
./cifar100/test/tulip_6755.jpg
./cifar100/test/plain_9006.jpg
./cifar100/test/boy_6241.jpg
./cifar100/test/raccoon_2158.jpg
./cifar100/test/rabbit_9988.jpg
./cifar100/test/bowl_2176.jpg
./cifar100/test/bowl_3091.jpg
./cifar100/test/baby_5960.jpg
./cifar100/test/couch_4306.jpg
./cifar100/test/ray_6722.jpg
./cifar100/test/baby_6875.jpg
./cifar100/test/elephant_3401.jpg
./cifar100/test/sunflower_3386.jpg
./cifar100/test/spider_2248.jpg
./cifar100/test/streetcar_4076.jpg
./cifar100/test/bee_6732.jpg
./cifar100/test/whale_736.jpg
./cifar100/test/otter_6029.jpg
./cifar100/test/beaver_248.jpg
./cifar100/test/road_424.jpg
./cifar100/test/wardrobe_328.jpg
./cifar100/test/bus_4307.jpg
./cifar100/test/tulip_9297.jpg
./cifar100/test/shark_8220.jpg
./cifar100/test/couch_7672.jpg
```

✓ Step 2: Custom Data Loading

```

import os
import time
import glob
import torch
import shutil
import torchvision
import torch.nn as nn
from torch.nn import functional as F
import torchvision.transforms as transforms
from PIL import Image
from torch.utils.data import Dataset

config = {
    "data_path": "./cifar100",
    "batch_size": 125,
    "shape": 50
}

class mydataset(Dataset):
    def __init__(self, data_dir, flag, transform):
        super(mydataset, self).__init__()
        self.root = data_dir
        self.label = flag
        self.transform = transform
        self.img_dir = os.path.join(self.root, self.label)
        self.img_names = glob.glob(os.path.join(self.img_dir, '*.jpg'))
        self.tags = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 't

    def RGB2FF(self, img: torch.tensor):
        fft_result = torch.fft.fftn(img, dim=[1, 2])
        fft_result = torch.fft.fftshift(fft_result, dim=[1, 2])
        height, width = img.shape[1], img.shape[2]
        center_width = width // 8
        center_height = height // 8
        fft_result[:, height//2-center_height//2:height//2+center_height//2,
                    width//2-center_width//2:width//2+center_width//2] = 0.0
        fft_result = torch.fft.ifftshift(fft_result, dim=[1, 2])
        fft_result = torch.fft.ifftn(fft_result, dim=[1, 2]).real
        return fft_result

    def __getitem__(self, idx):
        img_name = self.img_names[idx]
        img = Image.open(os.path.join(img_name))
        img = self.transform(img)
        img_ff = self.RGB2FF(img)
        for i in range(len(self.tags)):
            if self.tags[i] in img_name:
                tag = i
                break
        return img, img_ff, tag

    def __len__(self):
        return len(self.img_names)

# 自定义data augmentation
transform_train = transforms.Compose([
    transforms.Resize([config['shape'], config['shape']]),
    transforms.RandomHorizontalFlip(), # 随机水平翻转
    transforms.RandomRotation(20), # 随机旋转20度以内
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1), # 颜色抖动
    transforms.ToTensor(), # 转换为Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # 归一化
])

transform_test = torchvision.transforms.Compose([
    transforms.Resize([config['shape'], config['shape']]),

```

```


transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

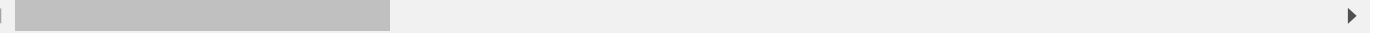
```

```

train_dataset = mydataset(data_dir=config['data_path'], flag= "train", transform=transform_train)
test_dataset = mydataset(data_dir=config['data_path'], flag= "test", transform=transform_test)
train_loader = torch.utils.data.DataLoader( train_dataset, batch_size=config['batch_size'], shuffle=True, num_workers=4, pin_memor
test_loader = torch.utils.data.DataLoader( test_dataset, batch_size=config['batch_size'], shuffle=True, num_workers=4, pin_memor

```

 /usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker



✓ Step 3: Configure the Neural Network

```

import torch.nn as nn
from torch.nn import functional as F
from torch.nn import Sequential as Seq
import torch.optim as optim
import torchvision.models as models

```

```

class SEBlock(nn.Module):
    def __init__(self, channel, reduction=128):
        super(SEBlock, self).__init__()
        self.fc1 = nn.Linear(channel, channel // reduction, bias=False)
        self.fc2 = nn.Linear(channel // reduction, channel, bias=False)
    def forward(self, x):
        b, c, _, _ = x.size()
        y = F.adaptive_avg_pool2d(x, 1).view(b, c)
        y = F.relu(self.fc1(y), inplace=True)
        y = torch.sigmoid(self.fc2(y)).view(b, c, 1, 1)
        return x * y

```

```

class CommonBlock(nn.Module):
    def __init__(self, in_channel, out_channel, stride):
        # 普通Block简单完成两次卷积操作
        super(CommonBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)
        self.se = SEBlock(out_channel)
        self.bn3 = nn.BatchNorm2d(out_channel)
    def forward(self, x):
        identity = x
        x = F.relu(self.bn1(self.conv1(x)), inplace=True)
        x = self.bn2(self.conv2(x))
        x = self.bn3(self.se(x))
        x += identity
        return F.relu(x, inplace=True)

```

```

class SpecialBlock(nn.Module):
    def __init__(self, in_channel, out_channel, stride):
        # 特殊Block完成两次卷积操作 · 以及一次升维下采样
        # 注意这里的stride传入一个数组 · shortcut和残差部分stride不同
        super(SpecialBlock, self).__init__()
        self.change_channel = nn.Sequential(
            # 负责升维下采样的卷积网络change_channel
            nn.Conv2d(in_channel, out_channel, kernel_size=1, stride=stride[0], padding=0, bias=False),
            nn.BatchNorm2d(out_channel))
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size=3, stride=stride[0], padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3, stride=stride[1], padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)
        self.se = SEBlock(out_channel)
        self.bn3 = nn.BatchNorm2d(out_channel)
    def forward(self, x):
        identity = self.change_channel(x)
        x = F.relu(self.bn1(self.conv1(x)), inplace=True)
        x = self.bn2(self.conv2(x))

```

```

    x = self.bn3(self.se(x))
    x += identity
    return F.relu(x, inplace=True)                                # 输出卷积单元

class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        # 使用预训练的ResNet的前几层作为特征提取
        resnet = models.resnet18(pretrained=True)
        self.prepare = nn.Sequential(
            resnet.conv1, resnet.bn1, resnet.relu, resnet.maxpool,
            resnet.layer1, resnet.layer2, resnet.layer3
        )
        self.layer1 = nn.Sequential(SpecialBlock(256, 512, [2, 1]), CommonBlock(512, 512, 1))
        self.layer2 = nn.Sequential(CommonBlock(512, 512, 1), CommonBlock(512, 512, 1))
        self.dropout = nn.Dropout(0.2)
        self.head = Seq(nn.AdaptiveAvgPool2d(1), nn.Flatten(start_dim=1))

    def forward(self, x):
        x = self.prepare(x)
        x = self.layer1(x)
        x = self.layer2(x)
        return self.head(self.dropout(x))

class DualBranchConvNet(nn.Module):
    def __init__(self, classes_num):
        super(DualBranchConvNet, self).__init__()
        self.original_branch = ConvNet() # Branch for original images
        self.frequency_branch = ConvNet() # Branch for frequency-domain images
        self.fc = nn.Sequential(
            nn.Linear(1024, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, classes_num))

    def forward(self, img, img_ff):
        original_out = self.original_branch(img)
        freq_out = self.frequency_branch(img_ff)
        combined = torch.cat([original_out, freq_out], dim=1)
        return self.fc(combined)

config = {
    "lr":0.025,
    "momentum":0.9,
    "weight_decay":5e-4,
}

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
net = DualBranchConvNet(classes_num=100).to(device)
criterion = torch.nn.CrossEntropyLoss().to(device)
# weight-decay L2正则化
optimizer = torch.optim.SGD(net.parameters(), lr=config["lr"], momentum=config["momentum"], weight_decay=config["weight_decay"])
# optimizer = torch.optim.AdamW(net.parameters(), lr=config["lr"], weight_decay=config["weight_decay"])
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.8)

# 钩子函数
features = {}

def get_features(name):
    def hook(model, input, output):
        features[name] = output.detach()
    return hook

# 注册钩子·提取融合前的两个分支的特征
net.original_branch.head.register_forward_hook(get_features('original_branch'))
net.frequency_branch.head.register_forward_hook(get_features('frequency_branch'))

# 注册钩子·提取融合后的特征
net.fc.register_forward_hook(get_features('fusion'))

```

```

↗ cuda:0
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f370
100%|██████████| 44.7M/44.7M [00:00<00:00, 173MB/s]
<torch.utils.hooks.RemovableHandle at 0x7fa95fe101c0>

```

✓ Step 4: Train the network and save model

```
import time
```

```

class AverageMeter(object):
    def __init__(self):
        self.reset()
    def reset(self):
        self.val = self.avg = self.sum = self.count = 0
    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count

def accuracy(output, target, topk=(1,1)):
    """Computes the precision@k for the specified values of k"""
    maxk = max(topk)
    batch_size = target.size(0)
    _, pred = output.topk(maxk, 1, True, True)
    pred = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    res = []
    for k in topk:
        correct_k = correct[:,k].view(-1).float().sum(0)
        res.append(correct_k.mul_(100.0 / batch_size))
    return res[0]

def train(train_loader, net, optimizer, criterion, epoch):
    batch_time = AverageMeter()
    data_time = AverageMeter()
    top1 = AverageMeter()
    LOSS = AverageMeter()
    net.train()
    end = time.time()
    for i, (img, img_ff, target) in enumerate(train_loader, start=1):
        data_time.update(time.time() - end)
        img = img.to(device)
        img_ff = img_ff.to(device)
        target = target.to(device)
        out = net(img, img_ff)
        loss = criterion(out, target)
        prec1 = accuracy(out, target) # prec1: list
        top1.update(prec1.item(), img.size(0))
        LOSS.update(loss.item(), img.size(0))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        batch_time.update(time.time() - end)
        end = time.time()
        if i % 50 == 0:
            log_str = ('Epoch[{0}]:[{1:03}/{2:03}] '
                       'Time:{batch_time.val:.4f} '
                       'Data:{data_time.val:.4f} '
                       'loss:{loss.val:.4f}({loss.avg:.4f}) '
                       'prec@1:{top1.val:.2f}({top1.avg:.2f}) '.format(
                           epoch, i, len(train_loader), batch_time=batch_time, data_time=data_time,

```

```
        loss=LOSS,
        top1=top1))
    print(log_str)

    return LOSS.avg

def save_checkpoint(state, is_best, save_root, epoch):
    if not os.path.exists(save_root):
        os.makedirs(save_root)
    save_path = os.path.join(save_root, 'epoch_{}.pth.tar'.format(str(epoch)))
    torch.save(state, save_path)
    if is_best:
        best_save_path = os.path.join(save_root, 'model_best.pth.tar')
        shutil.copyfile(save_path, best_save_path)

config = {
    "save_root": "./result",
    "epochs": 26,
}

best_top1 = 0
test_top1 = 0
for epoch in range(1, config["epochs"]+1):
    epoch_start_time = time.time()
    train_loss = train(train_loader, net, optimizer, criterion, epoch)
    epoch_duration = time.time() - epoch_start_time
    print('Epoch time: {}'.format(int(epoch_duration)))
    is_best = False
    print('Saving models.....')
    save_checkpoint({
        'epoch': epoch,
        'net': net.state_dict(),
        'prec@1': test_top1,
    }, is_best, config["save_root"], epoch)
    scheduler.step()
```



```

Epoch[25]:[050/400] Time:0.1491 Data:0.0003 loss:0.4760(0.3093) prec@1:85.60(90.54)
Epoch[25]:[100/400] Time:0.1351 Data:0.0003 loss:0.3262(0.3211) prec@1:89.60(90.15)
Epoch[25]:[150/400] Time:0.1797 Data:0.0002 loss:0.2326(0.3193) prec@1:92.00(90.23)
Epoch[25]:[200/400] Time:0.1611 Data:0.0045 loss:0.2952(0.3195) prec@1:90.40(90.12)
Epoch[25]:[250/400] Time:0.1632 Data:0.0003 loss:0.2613(0.3233) prec@1:91.20(89.97)
Epoch[25]:[300/400] Time:0.1605 Data:0.0003 loss:0.3220(0.3253) prec@1:89.60(89.96)
Epoch[25]:[350/400] Time:0.2257 Data:0.0004 loss:0.3437(0.3254) prec@1:90.40(89.98)
Epoch[25]:[400/400] Time:0.1222 Data:0.0001 loss:0.2866(0.3274) prec@1:92.80(89.90)
Epoch time: 95s
Saving models.....
Epoch[26]:[050/400] Time:0.1481 Data:0.0002 loss:0.2283(0.3196) prec@1:92.80(90.08)
Epoch[26]:[100/400] Time:0.1557 Data:0.0004 loss:0.3473(0.3206) prec@1:88.80(90.05)
Epoch[26]:[150/400] Time:0.1217 Data:0.0003 loss:0.4332(0.3228) prec@1:86.40(89.95)
Epoch[26]:[200/400] Time:0.1781 Data:0.0003 loss:0.2271(0.3200) prec@1:92.80(89.95)
Epoch[26]:[250/400] Time:0.2040 Data:0.0003 loss:0.3282(0.3219) prec@1:91.20(89.95)
Epoch[26]:[300/400] Time:0.1655 Data:0.0004 loss:0.3685(0.3228) prec@1:88.00(89.99)
Epoch[26]:[350/400] Time:0.1650 Data:0.0003 loss:0.3035(0.3221) prec@1:93.60(90.05)
Epoch[26]:[400/400] Time:0.1221 Data:0.0001 loss:0.2468(0.3219) prec@1:90.40(90.00)
Epoch time: 100s
Saving models.....

```

✓ Step 5: Test on single image

```

net.eval()
img = Image.open("./cifar100/test/apple_9904.jpg")
img = transform_test(img).unsqueeze(0)
fft_result = torch.fft.fftn(img, dim=[1, 2])
fft_result = torch.fft.fftshift(fft_result, dim=[1, 2])
height, width = img.shape[1], img.shape[2]
center_width = width // 8
center_height = height // 8
fft_result[:, height//2-center_height//2:height//2+center_height//2, width//2-center_width//2:width//2+center_width//2] = 0.0
fft_result = torch.fft.ifftshift(fft_result, dim=[1, 2])
fft_result = torch.fft.ifftn(fft_result, dim=[1, 2]).real
img = img.to(device)
img_ff = fft_result.to(device)
out = net(img, img_ff)
predicted_classes = torch.argmax(out, dim=1)
tags = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 'boy', 'bridge']
print(tags[predicted_classes[0]])

```

🔍 apple

```

def test(test_loader, net, criterion):
    losses = AverageMeter()
    top1 = AverageMeter()
    net.eval()
    for i, (img, img_ff, target) in enumerate(test_loader, start=1):
        img = img.to(device)
        img_ff = img_ff.to(device)
        target = target.to(device)
        with torch.no_grad():
            out = net(img, img_ff)
            loss = criterion(out, target)

        prec1 = accuracy(out, target) # prec1:list
        losses.update(loss.item(), img.size(0))
        top1.update(prec1.item(), img.size(0))

    f_1 = [losses.avg, top1.avg]
    print('-----test classification result-----')
    print('Loss: {:.4f}, Prec@1: {:.2f}%'.format(*f_1))

    return top1.avg

test_top1 = test(test_loader, net, criterion)

```

```

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker
warnings.warn(_create_warning_msg(
-----test classification result-----
Loss: 3.4360, Prec@1: 59.52%

```

✓ Step 7: T-SNE Visualization

Use hooks in PyTorch to extract feature representations from the intermediate layers of the model for the test set "testloader", and visualize them using the T-SNE method. The specific requirements are as follows:

Visualize the features before and after the dual-branch feature fusion. If there are multiple fusions, you may choose specific layers for visualization.

```

from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# 从捕获的特征中提取 numpy 数组
original_features = features['original_branch'].cpu().numpy()
frequency_features = features['frequency_branch'].cpu().numpy()
fusion_features = features['fusion'].cpu().numpy()


# 使用T-SNE进行降维
tsne = TSNE(n_components=2, random_state=42)

# 分别对两个分支的特征以及融合后的特征进行T-SNE处理
original_features_tsne = tsne.fit_transform(original_features)
frequency_features_tsne = tsne.fit_transform(frequency_features)
fusion_features_tsne = tsne.fit_transform(fusion_features)

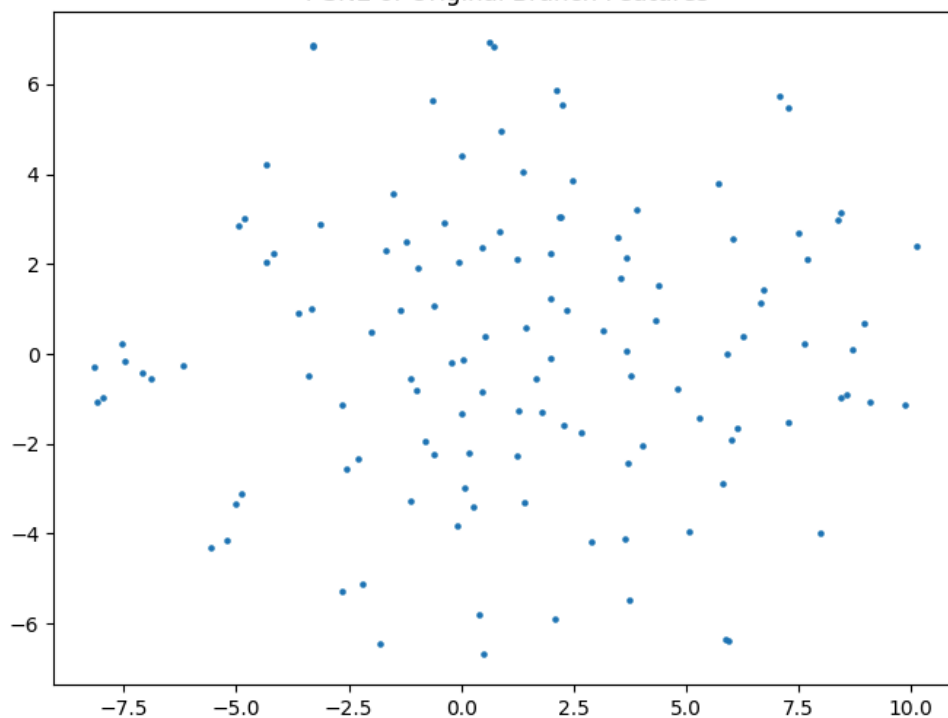
# 可视化
def plot_tsne(features, title, name):
    plt.figure(figsize=(8, 6))
    plt.scatter(features[:, 0], features[:, 1], s=5, cmap='viridis')
    plt.title(title)
    plt.show()
    plt.savefig(name)

# 可视化原始图像分支特征
plot_tsne(original_features_tsne, "T-SNE of Original Branch Features",
          "T-SNE of Original Branch Features.png")
# 可视化频域图像分支特征
plot_tsne(frequency_features_tsne, "T-SNE of Frequency Branch Features",
          "T-SNE of Frequency Branch Features.png")
# 可视化融合后的特征
plot_tsne(fusion_features_tsne, "T-SNE of Fusion Features",
          "T-SNE of Fusion Features.png")

```

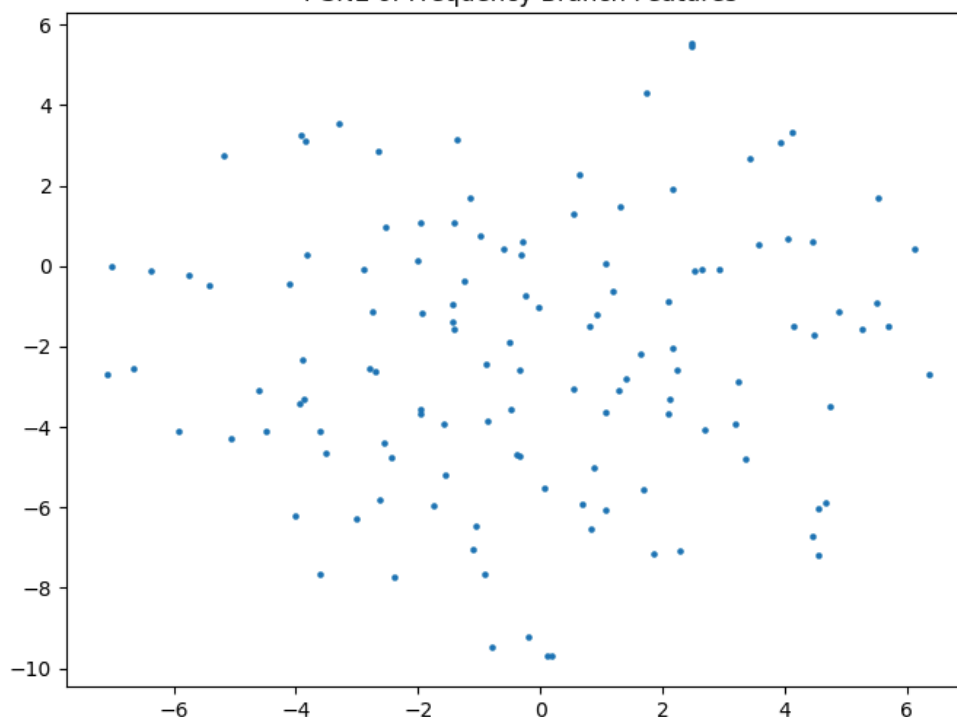

 <ipython-input-9-8e57d77159b2>:20: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
plt.scatter(features[:, 0], features[:, 1], s=5, cmap='viridis')

T-SNE of Original Branch Features



<Figure size 640x480 with 0 Axes>

T-SNE of Frequency Branch Features



<Figure size 640x480 with 0 Axes>

T-SNE of Fusion Features

