Question1

```
import os
import random
import numpy as np
import cv2
import time
from PIL import Image, ImageDraw, ImageFont
from tqdm import tqdm
from multiprocessing import Process, Value, Lock
import matplotlib.pyplot as plt
import torch

# Set the device to GPU or CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Convert your image data from numpy array to torch tensor, assuming you're
working with numpy arrays
def process_image_with_gpu(image_path):
    image = cv2.imread(image_path)
    # Assuming we need to process the image, convert it to grayscale and perform
certain calculations
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Convert the image to a PyTorch tensor
    image_tensor = torch.from_numpy(gray_image).float().to(device)

    # matrix operations
    processed_image = image_tensor * 0.5  # Resize/scale the image

    # Convert the result back to NumPy array for further operations
    processed_image = processed_image.cpu().numpy()

    return processed_image

labels = "账树并擦最仓莱桌蔑编锐柒畦汪苞舵秦便息饿；雏央偷母葱浙忱碾衅运辰忿魏豁嗓嘲吓货
叫谒摧飘拖砚养亏咱倦砾天摘坛琳铐拟谬书逻翅箍缝例丑好历溃依掠银稿惜霞椰尽仁蛮工稳亘严礼葵暑
倘保酒遭仇许做痪橡丝疮件穆败愤滤幅闻翻辅隙铲弦糙代赠殷锡租仙位鲸舷符橱酷猩汗氓挚地饵安阿样
泄唬憔干贩屹曼芥姜捎疾纱拴羔桂帜8插凌驼统寻阁杉丐凡验鬓瘫惩虏毕日环趟瘟边惰测卢溺住劈蜗摊
祠组木恤琉引疙旦晦射萌策昆犬推q脱预晚抵奉二中违萍瘤诚藕常烟萤菲涌攘吕贯劣俱势诞水余妖限肌
美漂参宋昭达吆说蕊咐臀旅挟苹控均阐殴涝祷轨摩融聪速踪驴扩谅辉形颖蝗皱北壁勤肠域妹畏社芋摄南
忌贴捣蛋鳄充逛箫遥衍豌尸异嚷狸俘核糯酥篡扒沸觉震跷妻践状弄沛迫瞬漱沐伶绘氯提纪疯记始沥锦农
哮详乘残栈杜技音现扼谴碌税晕先辕屈卦刺〉吉楞辨别煮撕稚镣倡生t锋度撞徘敦券唆喇菜k俺盗酸梭恼
叼肛轴河允咪穿耀芳菱悉画急矫版业嘶挫莹炼桐拘怔捐润进鸭肾埂胯芯双狰彰坦霸的挤馋滚铆膨，侯
+要蟋隐修栋乙倾虫怪么亮矮串腔带呛淡培霹儡李宿眯她瓮夸哟扛谁氮搅迅辞空诫雹截映滴哪肉诲陈俐
易耗茁搀械砖容泌泡癌兽心剂首岛灰憋桶添澜把巾斗壳众卓益莉）传抹葫遂防译杂侧烛煤岁哑煞揭言分
给向岖痘颗谚咒宠宅筒殃雀锉箩荒宛畔壹醇睦筐源蹭坞百但薪轿击喝厂姥牍花渡赴显酵诊泳贱络恐杠梅
盟铃擎苦旷《半窒捡漠奢侵牺棠挣己丙刊混确胡准腾毫悔云蔬枯师远钞搭史早量恕祈英鸦呼家敢武乍腮
坟陷悬思秃爬楔瞻米叮殖皇坏浆狈濒多锈享窝剃甫畜汁腰器饰拄殊搏川宽蓄撮公轩钉侣纽屏竣爸淌鸳摇
崖蛔蟥敏绎缀逗扶茅乳肢蕴彻般节晶恬民p秩瞭巡志果系婴惹簇恶绒操茂览追吁召疑流袋直包肿玫味幽
垒昨构超古呀竞顾赚钓咕e欠凹绕咏探腿覆波抒未骄狼性破乔鞋忆听竿掸普顿者句立兑榕将碱脆钝哥定
```

拗冰韵璧淫挺兴讨查同窜秽如利敛叙跋虱汛蝇种捧莺纫歉卵权矩缳境缨睬展仲肯悼套浑铅评妈谍致傀妒跳阵拣糖兰陆疗床晰髓膝妙具市毯赁驾痴伤皿眨踊模委较涂暂攻等制喧触肖畴粹抗钻灿创崇抚批憨助赵拌卜期哲深孕貌豺哀谆番谋执搬淤响情待嗤马叶站烦茬什训洞庞廉窖歼谜蔽寨患盔肘阎幌遵弹翁名涮斋外涤掰刮椅昔晾采剑檬赋篱拾怖高励a率裸慎乓搞卑奠暮议媚末野括诽币乏布智佣邻紊彤长嘉粗葡酣欺虎锤衫数狞绩龄拱国倒得孽盘张讥回寝食饺刁艘炸秸寺其匈啦咬媳\\觅蘸辈惨储痊企躬蜒我胰缔士晃傅恒溢镶循淮针贷囱妨糊醉偎颂亲满惋冻盛所剧盐共蟀仿址寇撼石帝选前臣倚俏置挨雪俭笆淑转宫曲淋噪喘意侮罚啸遮〈蛛奖嚼良翩宏绳挥硬鞭卷搪惫累弛巨锻各怕咳坎昙烘窿死怒陌姿朽诅行褪烹了夺旨斧禀蚀z罐不诉印蝶爆单拯嚣焚黎徒鹊启雕次栖}奥诈蓖泥卡庶佩姨绣着蛾万慌赔嗦法j吞取氏肺庄哗闸宴放谢轰案供卧漫椭冤碎盖蚪缩欣甜堵顷蔚蚕炒皮肋爷骗戚每能喻请扎王潮护齐汞株颈勾阶亥售缅摸仆倔猜扬麦富羹资蓬浩丢你敌躯族营沧借辑身锄舟锹萎刻灸旁怎捌酱伦燃暗愧鸵垮匪6悯熄拂捺粪烧迈盏搂窃输上堤撤答畅昏懦之帽珍终溶吝朱光缕涧缺蓝陡投唱契蟆窑湿至词冀程春类员邀油猫阻浪笔持随本厌乎加车芦迷锯课冠升嫁栏质拦消瀑舱垫症琢榛橙叉饮郑冷蝎作算仰尖买幸娜披琅逸瑟辜知个优坯软g糜隘洋旧霎相摔竹h蝙讯饭愕谓蹋魔b矛斯滨凯阴约损蒸述压菊筋惑楼码片徽潭讹割台停胀务鳍菠馅九曙路哨誓式巴略棒履田敲匙播拳琴毡朋失坡州煌按吮染扯裕实辐邢餐啥岗闹题赏嗡梧抑-计瞎洽胚鉴耽皆填唇彪负柜救讳火澎i愉锣用植抽忙忧下默呕笑温夫蝉艇兢础槐胎熙几与“闰罪腥毛求蛉气尼鸥篇艺寄阅亿欲凰扔它螺娘铡松玻井泽峰淹揍唾适薄京有拼硫您邑跛扣秋唯诬协吱骂罩欧足晤辫纳教丹密塑巍冲亩朗缴脐婿县豪糕再扭鸠絮往炬照退瞄尚复s凸另役喳似念褥鹃墅点踏完绸跨蛙欢缸毅悲蜘鞍扁森雁掂起柬碟庆埋可夏铁维股拷舆禁秉刃池筥硝香滩沈林杏辖押仪科沫撇娄垂演垦栗吸俄肴鹏链牧傻峦豹浮鲜杖吟忠登袱馁鞠吊浸过尊步泼猬蝌剖伊阀躁臊咸圈旱梁然玩蜀佳金善反荠惠锭薇块拐端焕是联赤坠挎刹孵狱抖永配烁鸿呈嘱芝鸯遗贬霉掘祸也况内垄捂猎时镐瘩激殉谐土蚓字逮拨警柱笙明‘型凝趾溅漆喜角箭腌黄耘世袄削藏交吠财飞矾茸劲剔贼奋跪茵于蚯膀馒肮束纺堂久椎甚绪胸迹碳经褒受躺隅逃脏镇攀篙狭邦月陶需尝弥专炮惭争厅沮颊膊抓菌掷替慰靴里招象原秧d岳渊f贫衬造刚橄勇自#慢聚狐唠溉临都手孔继弱猛讶吴热练俯告宗血岭慕那梗炕脑恳泛瓜幕含湖繁董间喉廓氢开年揖渣兜苟表浦砌癣童磷声产玄椿隔嫌揩攒隆坐簿胧寞闺装僻烂屁蜈真蜻涡鲫骤娩疼赐逼鸟蚂钾篷颁予些湃侨跟喷郭屑讼捷梳卸柴尤栽撬熟捻沦闽抡圆称旋渔被汇估茧乒右糠儒曹框叹剪示避除徙面屡寓司檩鳞蹄窥吗惶学派频铣虑总虾兄责呜免倍勿七瓢险递叔恍十斟雇叨晋铺条鹅灌越衷换揽牛楷吐申天迁唉三队谈彬贺枉栅丁蜒喂禽泪贮债轻力戏征熏全伪辣镰铜懒脊炭愚锌螃舅淀去戴伍猴及号耻呆职牡沉宜颜份鹤革肩惯纯穗凤斥究镊歇汰贝薯朦露耐茫滓浓2雾铝接座秀库禾蜡榆颠吵灵拢焙昌蒋物翼袒鹞才没侥入男枣丰西精衔奔津沪医陪掉痰」n四宁肚雨胁憎躺躲额差梨膘棱闲槽毙沃惦茶文锰打函盅圾注杭%勃撩缓祭鲤猖佑塞邓柔掏潜梯绑姓碰旬赦蚊目恨翎补妓筑少鼓话导坑匕肪襟嘀痛球匠夷奶薛戳艰狡仅成捅庙瘦俗须涵考腕叭逞蔓宰军备萄闪宝姑圣裤疟{见x掌问希恭道贾介判乐收捍谷囤钦醒誊蜂唁捏捉沙宾假涩支吩封y隧缭党盹亦忽榔认青服掀苛移吭俩潦厉局初街磕纬误捞蒲紧散梦管庭铭苗拥帘友斤仗凛穷荷妆墩盼材勋键绊壕癫织山骆耕婶霍帖耍访简蛤奇腺突湘鸡r裆托骏堡段9牵匀否叛脸秣崔坷尔洼蚜翰口蚌昵蹂擅藤辛弓朝愈郊塌1到：坪1刑姻振伏料誉衣曾窘豫垃黍连仍尾赛鹿惕龙剥枝谤落饥屉拧骑辽跺范款婆洒靶稀感岂脖群撵游渴焦顶醋腐、庐整兼崎桑哼卫止嘹幔机熊绿闷官忘窍基绅灶呻一特踢宵两贞祝格哺焰幻=雅胳狮舍胖馍吃诱届摆撑尘厦旭邮婉萧羊虹图门缚嘴芹燥以建滞钳宪洪鸣牌厘俊颤粘涨人毁姐部尿袖趋衡墨署吻廊绽甥迟签战僚赃囊厚风班杆怠辱3积户菇芙途贤洗雄棉胞玷笋藐肆掖！怯箕滑正挖蒙购赫隶污城轧快新拭奄舰膳匣办秆白喊兔敷荣灾检关背弟今讽凳拇嘿碉剩增毒御他酗姊静规液鹰删钙来为钧校峻脯凑钟炉蹬抱慧更妇区监奕伙吹鹦赌们揣兵底帮玉爪奏拜呢错磁素靖熬劝笼虽乾授柑私延纹罢送甘榜折改绷销妄漾峭慨唤裳盲歌命诗憾蚤偏苫砂票稼陋舀腊耸奴凶粤逆疏鱼啰承横怀幢舌贵琼檀察沼枷歹谊魄v锚蛀调赊瓣爵剿头而诀怨梆费坊缎晒粟夕当缆冗堕澳千驻船莽爱遣扑店帆报信脓痒害必棋无侄褂珊斜尉耿壮逝会悟绢桅哈凫萨习列叁聋劳啤碴汽弃蠢犀迄非蕉诸矢哩兆牢洛绰比荸樱权厨4疚钢踱女蜜挪孝园帅付影冶粒勉父还哆勺只奸太孙渠瞧徐泣写抄肝悄嗅因耳o檐乱帕指烫沟岸厢恢傲呵蚣拉榴窗帐呐佃匿绍附瓤蒿就飒订熔驮罗刨功审睛揉层羞姚汉鼻苔赖炫饶滥泰砰赂疤叽伞稽哄淘w舔蓉爽炊贸靠孟效册获谱康驹侠暇u咽墙裂揪跑蝴裁腻丧决眉伸铛蒜草啡药像镜副惧政伴冯悦螟老虐守柏障缠绝祖擒杯周疹距贰纵走扮勒集麸谭艾韭重枢芍伟低棵眶愿嫉邪漓锁姆璃舶磅搁驳？让逊樟嗜籍崩变夹捶骚帚巢蚁透遍漩态聂候催神撒褐危活庸丽娇疫广掩威斑腋这矿芜钠咨袜挑很领侈元咆蛇赡讲澄酪析疲级衙靡滔嫩标废韧梢痕垛搜柿困殿狗寂黑主甲逢沾盯瞪体寒卤锥跌综柄典雷咖卿硼平幼挂泉。荐昼谎左弧颇渐僵轮啃缤拙踩彭东通胜筏譬占榄切冈芒在闯舒眠肄汤仑馆握由饲聘昂猪粱悴德饼骇钱戒竖断赘纷刷悍降牙驰理胆猿凭发楣客郁蠕猾净询亭瓦寥祥堰碘线近湾航塔澈晓枪桃乖荧m叠脉蹦鄙碧偶钥挽却价羡逐篓塘弊视泞玛潘狠嘁冕使独茄挠商妥秕席纤嗽甸极措凿钩担岔怜翠茎细且举撰舞小根懈晌努爹抛豆坝克架任葬港辙瘸皂搔玲腹围哎陕趣衩棘鼎丘敬溪抠档存”绞杨吼孤暖栓纸冒棚陨搓漏稻&柳牲昧解清袭嫡酌糟筹红筷耙簸室固盒罕涣噩茉眷府膜砍谨啄钮督懂翘旺值项灼苇畸蔼辩渗烤菩碗电纠痢后冬霜盈*犁难孩荞侦氧骨续蟹荡施藻则卖焊烈棕眼瑰矗语摹臭诺观既狂酿陵吨唧僧娶归酝擂驱网芭该八溯返恰减趴够休瓷闭泻佛涉胶鼠涯伯甩瑞某桩疆稍拆惊诵

```
苍结品巷逾博翔概慈迎歪斩袁玖君即此瘾崭洁泊乃刽色芬岩铸找若彼睡埃涎描粥旗录沿留属淆济赎泵遇壶荆丸扰弯慷浴扫咙刘抢睛劫拍援抬睹令掺膏氨朴恋筝坤贡@榨子哭碑蛆锨挡硅愁虚恃强裹庇雳炎屿樊羽巧场著凉乡脚橘艳伺偿燕房蔗诡寸序宣乌蛹界磨驯拒赶晨吧阳巫窄苏笨扇五楚滋郎捆秘龟秤识桥病福离箱动戈柠棺齿秒枕想笛嵌燎奈犹渤肥捕望亚堆团啊c嚎合献馏对缘纲臂顽胃桦试微荔浊蒂拓研索零（棍涛氛拔遏珠凄粉脂海顺窟匆仔季礁悠魁驶脾茴篮崇趁携丈迂阔烙镀鲁盆赢麻阱埠庵枫乞萝屯荚术瞒廷衰蹲袍筛第宦杰莫华板鳖星裙涕5壤杀已或短肤勘敞澡尺据竭儿页酬荤堪砸婚丛读淳浇》夜义硕溜夯际坚巩村忍寿方匾】颓午贿治臼痹择锅大犯应谦排椒化侍贪囚0朵划故蘑墓蹈院桨载枚圃芽碍从睁省江伐彩【盾设亡瓶景居掐琐六魂促卒蔫穴暴论厕健「媒鬼辆紫娃饱屎宇刀看垢跃何募膛灭蜕瘪浅和瞳证灯懊渺谣汹又径骡唐蝠寡律辟章嫂籽屋育啼事颅粮7蕾雌出处莲恩肃韩峡葛煎吏徊拿绵嬉释傍黔鸽屠沽咧娱竟歧匹稠扳洲赞宙聊QWERTYUIOPASDFGHJKLZXCVBNM"
# Assuming batch processing of images is needed, we can use multiprocessing
(parallelization)
def process_images_in_parallel(image_paths):
    processes = []
    results = []

    for path in image_paths:
        p = Process(target=process_image_with_gpu, args=(path,))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

# Ensure the generate_data directory exists
os.makedirs("./generate_data", exist_ok=True)

# Define the font file storage path
font_directory = "./kaggle/input/12123334/Font"
# Get a list of font files in the font directory
font_files = [os.path.join(font_directory, f) for f in os.listdir(font_directory)
if f.endswith('.ttf') or f.endswith('.ttc')]

# Set the number of samples to generate per label
count = 100

# Calculate the total number of samples
total_samples = count * len(labels)

# Create a multi-process shared variable to track the total number of generated
samples
progress = Value('i', 0)
# Create a Lock object to manage access to shared resources
lock = Lock()

# Define the number of processes to use
num_processes = os.cpu_count()
# Split labels into num_processes chunks
chunk_size = len(labels) // num_processes
label_chunks = [labels[i:i + chunk_size] for i in range(0, len(labels),
chunk_size)]

def generate_sample(char, font_path):
    image_size = (64, 64)
```

```
    background_color = (255, 255, 255)

    image = Image.new('RGB', image_size, background_color)
    draw = ImageDraw.Draw(image)

    max_font_size = 48
    min_font_size = 10
    font_size = max_font_size
    font = ImageFont.truetype(font_path, font_size)

    while font_size >= min_font_size:
        bbox = draw.textbbox((0, 0), char, font=font)
        text_width = bbox[2] - bbox[0]
        text_height = bbox[3] - bbox[1]

        if text_width <= image_size[0] and text_height <= image_size[1]:
            break
        font_size -= 1
        font = ImageFont.truetype(font_path, font_size)

    if font_size < min_font_size:
        print("Character cannot fit into the image size")
        return image

    position = ((image_size[0] - text_width) // 2, (image_size[1] - text_height)
// 2)
    draw.text(position, char, fill=(0, 0, 0), font=font)

    image_np = np.array(image)
    noise_level = 10
    noise = np.random.randint(-noise_level, noise_level, image_np.shape,
dtype='int16')
    noisy_image_np = np.clip(image_np + noise, 0, 255).astype('uint8')

    kernel = np.ones((3, 3), np.uint8)
    processed_image_np = noisy_image_np
    if random.random() < 0.5:
        processed_image_np = cv2.erode(noisy_image_np, kernel, iterations=1)

    processed_image = Image.fromarray(processed_image_np)
    if char not in '._ ':
        rotation_angle = random.uniform(-5, 5)
        processed_image = processed_image.rotate(rotation_angle, expand=False,
fillcolor=background_color)

    return processed_image

def is_blank_image(image, threshold=50):
    gray_image = np.array(image.convert('L'))
    variance = gray_image.var()
    return variance < threshold


import re
```

```
def sanitize_filename(filename):
    # Replace invalid characters with an underscore (_)
    return re.sub(r'[<>:"/\\|?*]', '_', filename)


def save_sample(image, label):
    # Sanitize label to ensure it's a valid directory name
    label = sanitize_filename(label)

    folder_name = os.path.join("./generate_data", label)  # Use os.path.join for
cross-platform compatibility
    os.makedirs(folder_name, exist_ok=True)

    file_count = len(os.listdir(folder_name))
    file_name = f"{file_count}.png"
    file_path = os.path.join(folder_name, file_name)

    image.save(file_path)


def is_char_supported(char, font_path, font_size=48):
    try:
        font = ImageFont.truetype(font_path, font_size)
        if font.getbbox(char):
            return True
    except OSError:
        pass
    return False

def process_labels(labels, font_files, count, progress, lock):
    for char in labels:
        font_paths = random.choices(font_files, k=count)
        for font_path in font_paths:
            if not is_char_supported(char, font_path):
                print(f"Font {font_path} does NOT support character '{char}'")
                continue

            tryCnt = 0
            image = generate_sample(char, font_path)
            while is_blank_image(image) and tryCnt <= 10:
                tryCnt += 1
                newfont = random.choice([f for f in font_files if
is_char_supported(char, f)])
                image = generate_sample(char, newfont)
            if tryCnt > 9:
                continue
            save_sample(image, char)

            with lock:
                progress.value += 1

def create_random_overview_image(data_dir, grid_size=(10, 10), image_size=(64,
```

```
64)):
    canvas_size = (grid_size[0] * image_size[0], grid_size[1] * image_size[1])
    overview_image = Image.new('RGB', canvas_size, (255, 255, 255))

    labels = os.listdir(data_dir)
    random.shuffle(labels)
    label_count = 0

    for row in range(grid_size[0]):
        for col in range(grid_size[1]):
            if label_count < len(labels):
                label = labels[label_count]
                label_path = os.path.join(data_dir, label)
                images = os.listdir(label_path)
                if images:
                    image_path = os.path.join(label_path, random.choice(images))
                    image = Image.open(image_path).resize(image_size)
                    overview_image.paste(image, (col * image_size[0], row *
image_size[1]))
                    label_count += 1
            else:
                break

    plt.figure(figsize=(10, 10))
    plt.imshow(overview_image)
    plt.axis('off')
    plt.show()


def main():
    # Create a Process instance for each chunk of labels
    processes = []
    for chunk in label_chunks:
        process = Process(target=process_labels, args=(chunk, font_files, count,
progress, lock))
        processes.append(process)

    # Start each process
    for process in processes:
        process.start()

    # Use tqdm to display the total progress of sample generation
    with tqdm(total=total_samples) as pbar:
        while any(p.is_alive() for p in processes):
            with lock:
                pbar.update(progress.value - pbar.n)
            time.sleep(1)

    # Wait for all processes to finish
    for process in processes:
        process.join()

    # Print the completion message
    print("All samples generated")
```

```
    # Call the function to create the random overview image
    # Example usage
    data_dir = "./generate_data"
    create_random_overview_image(data_dir)

if __name__ == '__main__':
    main()
```

Output:

```
D:\python\.conda\envs\chemist\python.exe D:\python\Code\ML1\generate.py
100%|██████████| 359300/359300 [04:41<00:00, 1276.63it/s]
All samples generated

进程已结束，退出代码为 0
```

Question2

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from PIL import Image
from tqdm import tqdm
from torch.amp import GradScaler, autocast

# Use GradScaler for mixed precision training
scaler = GradScaler()  # Updated to the new syntax
```

```
# Safe loader function to handle image loading errors
def safe_loader(path):
    try:
        img = Image.open(path)
        img.verify()  # Verify if the image is valid
        img = Image.open(path)  # Re-open the image after verification
        return img
    except (IOError, SyntaxError) as e:
        print(f"Error loading image {path}: {e}")
        return None

class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        # Shortcut connection (if input and output channels don't match, use 1x1
convolution)
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride,
bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out += self.shortcut(x)  # Add the shortcut (residual connection)
        out = self.relu(out)
        return out

class ResNetModel(nn.Module):
    def __init__(self, num_classes):
        super(ResNetModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,
bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # Create the layers of residual blocks
        self.layer1 = self._make_layer(64, 128, 2, stride=1)
```

```
        self.layer2 = self._make_layer(128, 256, 2, stride=2)
        self.layer3 = self._make_layer(256, 512, 2, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))  # Global Average Pooling
        self.fc = nn.Linear(512, num_classes)  # Fully connected output layer
        self.dropout = nn.Dropout(0.5)  # Add Dropout layer to prevent overfitting

    def _make_layer(self, in_channels, out_channels, num_blocks, stride):
        layers = []
        layers.append(ResidualBlock(in_channels, out_channels, stride))
        for _ in range(1, num_blocks):
            layers.append(ResidualBlock(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)  # Flatten the output for the fully connected
layer
        x = self.dropout(x)  # Apply dropout before the fully connected layer
        x = self.fc(x)
        return x

def train():
    # Step 1: Load and Split Dataset
    data_dir = './generate_data'

    # Define transformations for data augmentation and normalization
    transform = transforms.Compose([
        transforms.Resize((128, 128)),              # Resize images to 128x128
        transforms.RandomHorizontalFlip(),          # Random horizontal flip for
augmentation
        transforms.RandomRotation(30),              # Random rotation up to 30
degrees
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
hue=0.2),  # Color jitter
        transforms.ToTensor(),                      # Convert image to tensor
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])  # Standard normalization
    ])

    # Load the dataset using ImageFolder with the safe loader
    dataset = datasets.ImageFolder(data_dir, transform=transform,
loader=safe_loader)

    # Filter out any invalid images that returned None
```

```
    dataset.samples = [sample for sample in dataset.samples if sample[0] is not
None]

    # Split the dataset into 80% training and 20% validation
    train_size = int(0.8 * len(dataset))
    val_size = len(dataset) - train_size
    train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

    # Define DataLoader for training and validation datasets
    batch_size = 32
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
num_workers=4, pin_memory=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False,
num_workers=4, pin_memory=True)

    # Step 2: Define Model Architecture
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    num_classes = len(dataset.classes)
    model = ResNetModel(num_classes).to(device)

    # Step 3: Set Loss Function and Optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=0.001)  # Use AdamW optimizer
for better regularization
    scheduler = StepLR(optimizer, step_size=5, gamma=0.7)  # Learning rate
scheduler

    # Step 4: Training Loop with Mixed Precision
    num_epochs = 20  # Number of epochs to train the model

    for epoch in range(num_epochs):
        model.train()  # Set model to training mode
        running_loss = 0.0

        # Iterate over the training data
        for inputs, labels in tqdm(train_loader, desc=f"Epoch
{epoch+1}/{num_epochs}", unit="batch"):
            # Move inputs and labels to the device (GPU)
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()  # Zero out the gradients from the previous step

            # Mixed precision: using autocast for automatic casting to FP16
            with autocast(device_type=device.type):  # Specify the device type
(CPU or CUDA)
                outputs = model(inputs)  # Forward pass
                loss = criterion(outputs, labels)  # Compute loss

            # Scale the loss and backpropagate
            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()

            running_loss += loss.item()
```

```
        # Calculate the average training loss for this epoch
        avg_train_loss = running_loss / len(train_loader)

        # Step 5: Validation Loop
        model.eval()  # Set model to evaluation mode
        val_loss = 0.0
        correct = 0
        total = 0

        # Disable gradient calculation during validation
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        # Calculate the average validation loss and accuracy
        avg_val_loss = val_loss / len(val_loader)
        val_accuracy = correct / total

        # Step 6: Output Training Results
        print(f"Epoch {epoch+1}/{num_epochs}: "
              f"Train Loss: {avg_train_loss:.4f}, "
              f"Val Loss: {avg_val_loss:.4f}, "
              f"Val Accuracy: {val_accuracy*100:.2f}%")

        scheduler.step()  # Update learning rate

    # Step 7: Save the Model
    torch.save(model.state_dict(), "resnet_model.pth")
    print("Model saved to resnet_model.pth")

if __name__ == '__main__':
    train()
```

Output:

```
D:\python\.conda\envs\chemist\python.exe D:\python\Code\ML1\CNN.py
Epoch 1/10: 100%|██████████| 8983/8983 [26:00<00:00,  5.76batch/s]
Epoch 1/10: Train Loss: 4.5307, Val Loss: 0.3580, Val Accuracy: 90.21%
Epoch 2/10: 100%|██████████| 8983/8983 [41:46<00:00,  3.58batch/s]
Epoch 2/10: Train Loss: 0.2037, Val Loss: 0.0983, Val Accuracy: 97.00%
Epoch 3/10: 100%|██████████| 8983/8983 [11:26<00:00, 13.09batch/s]
Epoch 3/10: Train Loss: 0.0822, Val Loss: 0.0752, Val Accuracy: 97.64%
Epoch 4/10: 100%|██████████| 8983/8983 [11:17<00:00, 13.26batch/s]
```

```
Epoch 4/10: Train Loss: 0.0493, Val Loss: 0.0343, Val Accuracy: 98.82%
Epoch 5/10: 100%|██████████| 8983/8983 [11:10<00:00, 13.40batch/s]
Epoch 5/10: Train Loss: 0.0354, Val Loss: 0.0241, Val Accuracy: 99.22%
Epoch 6/10: 100%|██████████| 8983/8983 [11:10<00:00, 13.40batch/s]
Epoch 6/10: Train Loss: 0.0266, Val Loss: 0.0222, Val Accuracy: 99.23%
Epoch 7/10: 100%|██████████| 8983/8983 [10:21<00:00, 14.45batch/s]
Epoch 7/10: Train Loss: 0.0219, Val Loss: 0.0180, Val Accuracy: 99.37%
Epoch 8/10: 100%|██████████| 8983/8983 [07:50<00:00, 19.09batch/s]
Epoch 8/10: Train Loss: 0.0182, Val Loss: 0.0143, Val Accuracy: 99.47%
Epoch 9/10: 100%|██████████| 8983/8983 [10:59<00:00, 13.63batch/s]
Epoch 9/10: Train Loss: 0.0157, Val Loss: 0.0214, Val Accuracy: 99.29%
Epoch 10/10: 100%|██████████| 8983/8983 [11:23<00:00, 13.14batch/s]
Epoch 10/10: Train Loss: 0.0138, Val Loss: 0.0146, Val Accuracy: 99.50%
Model saved to resnet_model.pth

进程已结束，退出代码为 0
```

Question3

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm

# Define the tokenizer as a regular function
def tokenizer_function(x, max_seq_length=256):
    return [min(ord(c), 255) for c in x[:max_seq_length]]

class CustomDataset(Dataset):
    def __init__(self, texts, labels, tokenizer):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer

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

    def __getitem__(self, idx):
        text = torch.tensor(self.tokenizer(self.texts[idx]), dtype=torch.long)
        label = torch.tensor(self.labels[idx], dtype=torch.float32)
        return text, label

def collate_fn(batch):
    texts, labels = zip(*batch)
    lengths = torch.tensor([len(text) for text in texts])
    texts = pad_sequence(texts, batch_first=True, padding_value=0)
    labels = torch.tensor(labels, dtype=torch.float32)
```

```
    return texts, lengths, labels

class SentimentRNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
        super(SentimentRNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

    def forward(self, text, text_lengths):
        embedded = self.embedding(text)
        # In train_model and evaluate_model
        packed_embedded = pack_padded_sequence(embedded, text_lengths.cpu(),
batch_first=True, enforce_sorted=False)

        packed_output, (hidden, cell) = self.rnn(packed_embedded)
        dense_outputs = self.fc(hidden[-1])
        return self.sigmoid(dense_outputs)

def train_model(model, train_loader, optimizer, criterion, num_epochs):
    train_losses = []
    train_accuracies = []
    for epoch in range(num_epochs):
        model.train()
        epoch_loss = 0
        correct = 0
        total = 0
        for texts, lengths, labels in tqdm(train_loader, desc=f"Epoch
{epoch+1}/{num_epochs}", ncols=100):
            texts, lengths, labels = texts.to(device), lengths.to(device),
labels.to(device)
            optimizer.zero_grad()
            predictions = model(texts, lengths.cpu()).squeeze()  # Ensure lengths
are on CPU
            loss = criterion(predictions, labels)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
            predicted = (predictions >= 0.5).float()
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
        avg_loss = epoch_loss / len(train_loader)
        accuracy = correct / total
        train_losses.append(avg_loss)
        train_accuracies.append(accuracy)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}, Accuracy:
{accuracy:.4f}")
    return train_losses, train_accuracies

def evaluate_model(model, val_loader, criterion):
    model.eval()
    val_loss = 0
    correct = 0
```

```
    total = 0
    with torch.no_grad():
        for texts, lengths, labels in tqdm(val_loader, desc="Evaluating",
ncols=100):
            texts, lengths, labels = texts.to(device), lengths.to(device),
labels.to(device)
            predictions = model(texts, lengths.cpu()).squeeze()  # Ensure lengths
are on CPU
            loss = criterion(predictions, labels)
            val_loss += loss.item()
            predicted = (predictions >= 0.5).float()
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    avg_loss = val_loss / len(val_loader)
    accuracy = correct / total
    return avg_loss, accuracy


if __name__ == '__main__':  # Add this line to protect the entry point

    # Load datasets
    train_df = pd.read_csv('./kaggle/input/12123334/train.tsv', sep='\t')
    test_df = pd.read_csv('./kaggle/input/12123334/test.tsv', sep='\t')

    train_texts = train_df['text_a'].tolist()
    train_labels = train_df['label'].tolist()

    val_texts = test_df['text_a'].tolist()
    val_labels = test_df['label'].tolist()

    # Tokenizer and other configurations
    max_seq_length = 256
    # Pass the tokenizer function as an argument
    tokenizer = tokenizer_function

    vocab_size = 256
    embedding_dim = 32
    hidden_dim = 64
    output_dim = 1
    num_epochs = 10
    batch_size = 16
    learning_rate = 0.001

    # Prepare datasets and dataloaders
    train_dataset = CustomDataset(train_texts, train_labels, tokenizer)
    val_dataset = CustomDataset(val_texts, val_labels, tokenizer)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
collate_fn=collate_fn, num_workers=2)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False,
collate_fn=collate_fn, num_workers=2)

    # Device configuration
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
    # Initialize model, criterion, and optimizer
    model = SentimentRNN(vocab_size, embedding_dim, hidden_dim,
output_dim).to(device)
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Train the model
    train_losses, train_accuracies = train_model(model, train_loader, optimizer,
criterion, num_epochs)

    # Evaluate the model
    val_loss, val_accuracy = evaluate_model(model, val_loader, criterion)

    print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy:
{val_accuracy:.4f}")

    # Plot the results
    plt.figure(figsize=(12, 5))

    # Plot Training Loss
    plt.subplot(1, 2, 1)
    plt.plot(range(1, num_epochs + 1), train_losses, label="Training Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title("Training Loss Trend")
    plt.legend()

    # Plot Training Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(range(1, num_epochs + 1), train_accuracies, label="Training
Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("Training Accuracy Trend")
    plt.legend()

    plt.show()
```

Output:

```
D:\python\.conda\envs\chemist\python.exe D:\python\Code\ML1\visual1.py
Epoch 1/10: 100%|██████████████████████████████████████████████████| 572/572
[00:34<00:00, 16.71it/s]
Epoch 1/10, Loss: 0.6940, Accuracy: 0.5078
Epoch 2/10: 100%|██████████████████████████████████████████████████| 572/572
[00:26<00:00, 21.57it/s]
Epoch 2/10, Loss: 0.6925, Accuracy: 0.5045
Epoch 3/10: 100%|██████████████████████████████████████████████████| 572/572
[00:25<00:00, 22.31it/s]
```

```
Epoch 3/10, Loss: 0.6915, Accuracy: 0.5087
Epoch 4/10: 100%|██████████| 572/572
[00:24<00:00, 22.93it/s]
Epoch 4/10, Loss: 0.6905, Accuracy: 0.5054
Epoch 5/10: 100%|██████████| 572/572
[00:26<00:00, 21.99it/s]
Epoch 5/10, Loss: 0.6852, Accuracy: 0.5358
Epoch 6/10: 100%|██████████| 572/572
[00:25<00:00, 22.04it/s]
Epoch 6/10, Loss: 0.6739, Accuracy: 0.5573
Epoch 7/10: 100%|██████████| 572/572
[00:25<00:00, 22.10it/s]
Epoch 7/10, Loss: 0.6659, Accuracy: 0.5786
Epoch 8/10: 100%|██████████| 572/572
[00:25<00:00, 22.76it/s]
Epoch 8/10, Loss: 0.6543, Accuracy: 0.5865
Epoch 9/10: 100%|██████████| 572/572
[00:26<00:00, 21.57it/s]
Epoch 9/10, Loss: 0.6549, Accuracy: 0.5877
Epoch 10/10: 100%|██████████| 572/572
[00:25<00:00, 22.62it/s]
Epoch 10/10, Loss: 0.6537, Accuracy: 0.5890
Evaluating: 100%|██████████| 75/75
[00:08<00:00,  9.34it/s]
Validation Loss: 0.6795, Validation Accuracy: 0.5567

进程已结束，退出代码为 0
```