## Vue前端代码

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8" />  
 <link rel="icon" href="/favicon.ico" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <meta name="description" content="vue-element-admin的vue3版本" />  
 <meta name="keywords" content="vue-element-admin,vue3-element-admin" />  
 <title>图像去雾系统</title>  
</head>  
<body>  
<div id="app" class="app"></div>  
<script type="module" src="/src/main.ts"></script>  
</body>  
</html>

import { createApp } from "vue";
  
import App from "./App.vue";
  
import router from "@/router";
  
import { setupStore } from "@/store";
  
import { setupDirective } from "@/directive";
  
import "@/permission";
  
// 本地SVG图标
  
import "virtual:svg-icons-register";
  
// 国际化
  
import i18n from "@/lang/index";
  
// 样式
  
import "element-plus/theme-chalk/dark/css-vars.css";
  
import "@/styles/index.scss";
  
import "uno.css";
  
const app = createApp(App);
  
// 全局注册 自定义指令(directive)
  
setupDirective(app);
  
// 全局注册 状态管理(store)
  
setupStore(app);
  
app.use(router).use(i18n).mount("#app");
  
import { DehazeIndex, ImageInfo, ModelInfo } from "@/api/dehaze/types";
  
import requestPy from "@/utils/request-py";
  
import { AxiosProgressEvent, AxiosPromise } from "axios";
  
export function getModelApi(): AxiosPromise<ModelInfo[]> {
  
 return requestPy({
  
 url: "/model/",
  
 method: "get",
  
 });
  
}
  
export function uploadImageApi(file: File): AxiosPromise<ImageInfo> {
  
 return requestPy({
  
 url: "/upload/",
  
 method: "post",
  
 data: file,
  
 headers: {
  
 "Content-Type": "Image/png",
  
 },
  
 });
  
}
  
export function downloadApi(image\_name: string): AxiosPromise<File> {
  
 return requestPy({
  
 url: `/download/${image\_name}/`,
  
 method: "get",
  
 });
  
}
  
export function dehazeApi(
  
 haze\_image: string,
  
 model\_name: string
  
): AxiosPromise<ImageInfo> {
  
 return requestPy({
  
 url: "/dehazeImage/",
  
 method: "post",
  
 data: { haze\_image, model\_name },
  
 });
  
}
  
export function calculateIndexApi(
  
 haze\_image: string,
  
 clear\_image: string,
  
 onUpload: ((progressEvent: AxiosProgressEvent) => void) | undefined
  
): AxiosPromise<DehazeIndex> {
  
 return requestPy({
  
 url: "/calculateIndex/",
  
 method: "post",
  
 data: { haze\_image, clear\_image },
  
 onUploadProgress: onUpload,
  
 });
  
}
  
export interface ModelInfo {
  
 value: string;
  
 label: string;
  
 children?: ModelInfo;
  
}
  
export interface DehazeIndex {
  
 psnr: string;
  
 ssim: string;
  
}
  
export interface ImageInfo {
  
 image\_name: string;
  
}
  
// 系统设置
  
interface DefaultSettings {
  
 title: string;
  
 showSettings: boolean;
  
 tagsView: boolean;
  
 fixedHeader: boolean;
  
 sidebarLogo: boolean;
  
 layout: string;
  
 theme: string;
  
 size: string;
  
 language: string;
  
}
  
const defaultSettings: DefaultSettings = {
  
 title: "图像去雾系统",
  
 showSettings: true,
  
 tagsView: false,
  
 fixedHeader: false,
  
 sidebarLogo: true,
  
 layout: "left",
  
 theme: "light",
  
 size: "default", // default |large |small
  
 language: "zh-cn", // zh-cn| en
  
};
  
export default {
  
 // 路由国际化
  
 route: {
  
 dashboard: "首页",
  
 document: "项目文档",
  
 },
  
 // 登录页面国际化
  
 login: {
  
 title: "图像去雾系统",
  
 username: "用户名",
  
 password: "密码",
  
 login: "登 录",
  
 verifyCode: "验证码",
  
 },
  
 // 导航栏国际化
  
 navbar: {
  
 dashboard: "首页",
  
 logout: "注销",
  
 },
  
};
  
// 创建 axios 实例
  
import axios, { AxiosResponse } from "axios";
  
  
const service = axios.create({
  
 baseURL: import.meta.env.VITE\_APP\_PYTHON\_API,
  
 timeout: 50000,
  
});
  
service.interceptors.response.use(
  
 (response: AxiosResponse) => {
  
 const { code, msg } = response.data;
  
 if (code === "00000") {
  
 return response.data;
  
 }
  
 if (response.data instanceof ArrayBuffer) {
  
 return response;
  
 }
  
 ElMessage.error(msg || response.data);
  
 return Promise.reject(new Error(msg || "Error"));
  
 },
  
 (error: any) => {
  
 if (error.response.data) {
  
 const { msg } = error.response.data;
  
 ElMessage.error(msg || "系统出错");
  
 }
  
 return Promise.reject(error.message);
  
 }
  
);
  
export const imageBaseURL: string = import.meta.env.VITE\_APP\_IMG\_URL;
  
// 导出 axios 实例
  
export default service;
  
export default defaultSettings;
  
import vue from "@vitejs/plugin-vue";
  
import { UserConfig, ConfigEnv, loadEnv, defineConfig } from "vite";
  
import AutoImport from "unplugin-auto-import/vite";
  
import Components from "unplugin-vue-components/vite";
  
import { ElementPlusResolver } from "unplugin-vue-components/resolvers";
  
import Icons from "unplugin-icons/vite";
  
import IconsResolver from "unplugin-icons/resolver";
  
import { createSvgIconsPlugin } from "vite-plugin-svg-icons";
  
import { viteMockServe } from "vite-plugin-mock";
  
import visualizer from "rollup-plugin-visualizer";
  
import UnoCSS from "unocss/vite";
  
import path from "path";
  
import viteCompression from "vite-plugin-compression";
  
const pathSrc = path.resolve(\_\_dirname, "src");
  
export default defineConfig(({ mode }: ConfigEnv): UserConfig => {
  
 const env = loadEnv(mode, process.cwd());
  
 return {
  
 resolve: {
  
 alias: {
  
 "@": pathSrc,
  
 },
  
 },
  
 css: {
  
 // CSS 预处理器
  
 preprocessorOptions: {
  
 //define global scss variable
  
 scss: {
  
 javascriptEnabled: true,
  
 additionalData: `
  
 @use "@/styles/variables.scss" as \*;
  
 `,
  
 },
  
 },
  
 },
  
 server: {
  
 host: "0.0.0.0",
  
 port: Number(env.VITE\_APP\_PORT),
  
 open: true, // 运行是否自动打开浏览器
  
 proxy: {
  
 // 反向代理解决跨域
  
 [env.VITE\_APP\_BASE\_API]: {
  
 target: env.VITE\_APP\_TARGET\_URL,
  
 changeOrigin: true,
  
 rewrite: (path) =>
  
 path.replace(
  
 new RegExp("^" + env.VITE\_APP\_BASE\_API),
  
 env.VITE\_APP\_TARGET\_BASE\_API
  
 ), // 替换 /dev-api 为 target 接口地址
  
 },
  
 [env.VITE\_APP\_PYTHON\_API]: {
  
 target: env.VITE\_APP\_PYTHON\_URL,
  
 changeOrigin: true,
  
 rewrite: (path) =>
  
 path.replace(
  
 new RegExp("^" + env.VITE\_APP\_PYTHON\_API),
  
 env.VITE\_APP\_TARGET\_BASE\_API
  
 ),
  
 },
  
 },
  
 },
  
 plugins: [
  
 vue(),
  
 UnoCSS({}),
  
 AutoImport({
  
 // 自动导入 Vue 相关函数，如：ref, reactive, toRef 等
  
 imports: ["vue", "@vueuse/core"],
  
 eslintrc: {
  
 enabled: false,
  
 filepath: "./.eslintrc-auto-import.json",
  
 globalsPropValue: true,
  
 },
  
 resolvers: [
  
 // 自动导入 Element Plus 相关函数，如：ElMessage, ElMessageBox... (带样式)
  
 ElementPlusResolver(),
  
 IconsResolver({}),
  
 ],
  
 vueTemplate: true,
  
 // 配置文件生成位置(false:关闭自动生成)
  
 // dts: false,
  
 dts: "src/types/auto-imports.d.ts",
  
 }),
  
 Components({
  
 resolvers: [
  
 // 自动导入 Element Plus 组件
  
 ElementPlusResolver(),
  
 // 自动导入图标组件
  
 IconsResolver({
  
 // @iconify-json/ep 是 Element Plus 的图标库
  
 enabledCollections: ["ep"],
  
 }),
  
 ],
  
 // 指定自定义组件位置(默认:src/components)
  
 dirs: ["src/\*\*/components"],
  
 // 配置文件位置(false:关闭自动生成)
  
 // dts: false,
  
 dts: "src/types/components.d.ts",
  
 }),
  
 Icons({
  
 // 自动安装图标库
  
 autoInstall: true,
  
 }),
  
 createSvgIconsPlugin({
  
 // 指定需要缓存的图标文件夹
  
 iconDirs: [path.resolve(pathSrc, "assets/icons")],
  
 // 指定symbolId格式
  
 symbolId: "icon-[dir]-[name]",
  
 }),
  
 // 代码压缩
  
 viteCompression({
  
 verbose: true, // 默认即可
  
 disable: true, // 是否禁用压缩，默认禁用，true为禁用,false为开启，打开压缩需配置nginx支持
  
 deleteOriginFile: true, // 删除源文件
  
 threshold: 10240, // 压缩前最小文件大小
  
 algorithm: "gzip", // 压缩算法
  
 ext: ".gz", // 文件类型
  
 }),
  
 viteMockServe({
  
 ignore: /^\_/,
  
 mockPath: "mock",
  
 enable: mode === "development",
  
 // https://github.com/anncwb/vite-plugin-mock/issues/9
  
 }),
  
 visualizer({
  
 filename: "./stats.html",
  
 open: false,
  
 gzipSize: true,
  
 brotliSize: true,
  
 }),
  
 ],
  
 // 预加载项目必需的组件
  
 optimizeDeps: {
  
 include: [
  
 "vue",
  
 "vue-router",
  
 "pinia",
  
 "axios",
  
 "element-plus/es/components/form/style/css",
  
 "element-plus/es/components/form-item/style/css",
  
 "element-plus/es/components/button/style/css",
  
 "element-plus/es/components/input/style/css",
  
 "element-plus/es/components/input-number/style/css",
  
 "element-plus/es/components/switch/style/css",
  
 "element-plus/es/components/upload/style/css",
  
 "element-plus/es/components/menu/style/css",
  
 "element-plus/es/components/col/style/css",
  
 "element-plus/es/components/icon/style/css",
  
 "element-plus/es/components/row/style/css",
  
 "element-plus/es/components/tag/style/css",
  
 "element-plus/es/components/dialog/style/css",
  
 "element-plus/es/components/loading/style/css",
  
 "element-plus/es/components/radio/style/css",
  
 "element-plus/es/components/radio-group/style/css",
  
 "element-plus/es/components/popover/style/css",
  
 "element-plus/es/components/scrollbar/style/css",
  
 "element-plus/es/components/tooltip/style/css",
  
 "element-plus/es/components/dropdown/style/css",
  
 "element-plus/es/components/dropdown-menu/style/css",
  
 "element-plus/es/components/dropdown-item/style/css",
  
 "element-plus/es/components/sub-menu/style/css",
  
 "element-plus/es/components/menu-item/style/css",
  
 "element-plus/es/components/divider/style/css",
  
 "element-plus/es/components/card/style/css",
  
 "element-plus/es/components/link/style/css",
  
 "element-plus/es/components/breadcrumb/style/css",
  
 "element-plus/es/components/breadcrumb-item/style/css",
  
 "element-plus/es/components/table/style/css",
  
 "element-plus/es/components/tree-select/style/css",
  
 "element-plus/es/components/table-column/style/css",
  
 "element-plus/es/components/select/style/css",
  
 "element-plus/es/components/option/style/css",
  
 "element-plus/es/components/pagination/style/css",
  
 "element-plus/es/components/tree/style/css",
  
 "element-plus/es/components/alert/style/css",
  
 "element-plus/es/components/radio-button/style/css",
  
 "element-plus/es/components/checkbox-group/style/css",
  
 "element-plus/es/components/checkbox/style/css",
  
 "element-plus/es/components/tabs/style/css",
  
 "element-plus/es/components/tab-pane/style/css",
  
 "element-plus/es/components/rate/style/css",
  
 "element-plus/es/components/date-picker/style/css",
  
 "element-plus/es/components/notification/style/css",
  
 "@vueuse/core",
  
 "sortablejs",
  
 "path-to-regexp",
  
 "echarts",
  
 "@wangeditor/editor",
  
 "@wangeditor/editor-for-vue",
  
 "vue-i18n",
  
 "codemirror",
  
 ],
  
 },
  
 };
  
});

<script setup lang="ts">  
 import { uploadImageApi } from "@/api/dehaze";  
 import { imageBaseURL } from "@/utils/request-py";  
 import { UploadRawFile, UploadRequestOptions } from "element-plus";  
 const props = defineProps({  
 modelValue: {  
 type: String,  
 default: "",  
 },  
 title: {  
 type: String,  
 default: "点击上传文件",  
 },  
 });  
 const emit = defineEmits(["update:modelValue", "getImageInfo"]);  
 const imageName = useVModel(props, "modelValue", emit);  
 const imgUrl = computed(() => {  
 return imageName.value ? imageBaseURL + "/" + imageName.value + "/" : "";  
 });  
 const isLoading = ref(false);  
 async function uploadFile(options: UploadRequestOptions): Promise<any> {  
 isLoading.value = true;  
 try {  
 const { data } = await uploadImageApi(options.file);  
 imageName.value = data.image\_name;  
 } finally {  
 isLoading.value = false;  
 }  
 }  
 async function handleBeforeUpload(file: UploadRawFile) {  
 if (file.size > 20 \* 1024 \* 1024) {  
 ElMessage.warning("上传图片不能大于20M");  
 return false;  
 }  
 const image = new Image();  
 image.src = URL.createObjectURL(file);  
 await image.decode();  
 // 向父组件传递用户所上传的图片的分辨率  
 emit("getImageInfo", image.naturalWidth, image.naturalHeight);  
 return true;  
 }  
</script>  
<template>  
 <el-upload  
 v-model="imgUrl"  
 class="single-uploader"  
 :show-file-list="false"  
 list-type="picture"  
 :before-upload="handleBeforeUpload"  
 :http-request="uploadFile"  
 >  
 <img v-if="imgUrl" :src="imgUrl" class="single" alt="" />  
 <div v-else-if="isLoading" class="single-uploader-placeholder">  
 <span>上传中...</span>  
 </div>  
 <div v-else class="single-uploader-placeholder">  
 <el-icon class="single-uploader-icon"><i-ep-plus /></el-icon>  
 <span style="margin-top: 5px">{{ title }}</span>  
 </div>  
 </el-upload>  
</template>  
<style scoped>  
 .single-uploader .single {  
 display: block;  
 width: 35vmax;  
 height: 35vmax;  
 }  
</style>  
<style>  
 .single-uploader .el-upload {  
 position: relative;  
 overflow: hidden;  
 cursor: pointer;  
 border: 1px dashed var(--el-border-color);  
 border-radius: 6px;  
 transition: var(--el-transition-duration-fast);  
 }  
 .single-uploader .el-upload:hover {  
 border-color: var(--el-color-primary);  
 }  
 .single-uploader-placeholder {  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 justify-content: center;  
 width: 35vmax;  
 height: 35vmax;  
 }  
 .single-uploader-icon {  
 font-size: 28px;  
 color: #8c939d;  
 text-align: center;  
 }  
</style>  
<script setup lang="ts">  
import { useUserStore } from "@/store/modules/user";  
import { useTransition, TransitionPresets } from "@vueuse/core";  
defineOptions({  
 // eslint-disable-next-line  
 name: "Dashboard",  
 inheritAttrs: false,  
});  
const userStore = useUserStore();  
const date: Date = new Date();  
const greetings = computed(() => {  
 const hours = date.getHours();  
 if (hours >= 6 && hours < 8) {  
 return "晨起披衣出草堂，轩窗已自喜微凉🌅！";  
 } else if (hours >= 8 && hours < 12) {  
 return "上午好🌞！";  
 } else if (hours >= 12 && hours < 18) {  
 return "下午好☕！";  
 } else if (hours >= 18 && hours < 24) {  
 return "晚上好🌃！";  
 } else if (hours >= 0 && hours < 6) {  
 return "偷偷向银河要了一把碎星，只等你闭上眼睛撒入你的梦中，晚安🌛！";  
 }  
});  
const duration = 5000;  
// 收入金额  
const amount = ref(0);  
const amountOutput = useTransition(amount, {  
 duration: duration,  
 transition: TransitionPresets.easeOutExpo,  
});  
amount.value = 150;  
// 访问数  
const visitCount = ref(0);  
const visitCountOutput = useTransition(visitCount, {  
 duration: duration,  
 transition: TransitionPresets.easeOutExpo,  
});  
visitCount.value = 2180;  
//消息数  
const messageCount = ref(0);  
const messageCountOutput = useTransition(messageCount, {  
 duration: duration,  
 transition: TransitionPresets.easeOutExpo,  
});  
messageCount.value = 15;  
// 订单数  
const orderCount = ref(0);  
const orderCountOutput = useTransition(orderCount, {  
 duration: duration,  
 transition: TransitionPresets.easeOutExpo,  
});  
orderCount.value = 154;  
</script>  
<template>  
 <div class="dashboard-container">  
 <!-- 用户信息 -->  
 <el-row class="mb-8">  
 <el-card class="w-full">  
 <div class="flex justify-between flex-wrap">  
 <div class="flex items-center">  
 <img  
 class="user-avatar"  
 :src="userStore.avatar + '?imageView2/1/w/80/h/80'"  
 />  
 <span class="ml-[10px] text-[16px]">  
 {{ userStore.nickname }}  
 </span>  
 </div>  
 <div class="leading-[40px]">  
 {{ greetings }}  
 </div>  
 <div class="space-x-2 flex items-center justify-end">  
 <el-link target="\_blank" type="danger" href="http://10.16.90.26/"  
 >💥土味锌的阅读笔记</el-link  
 >  
 <el-divider direction="vertical" />  
 <el-link  
 target="\_blank"  
 type="success"  
 href="https://gitee.com/earthy-zinc"  
 >Gitee</el-link  
 >  
 <el-divider direction="vertical" />  
 <el-link  
 target="\_blank"  
 type="primary"  
 href="https://github.com/earthy-zinc"  
 >GitHub  
 </el-link>  
 </div>  
 </div>  
 </el-card>  
 </el-row>  
 <!-- 数据卡片 -->  
 <el-row :gutter="40" class="mb-4">  
 <el-col :xs="24" :sm="12" :lg="6" class="mb-4">  
 <div class="data-box">  
 <div  
 class="text-[#40c9c6] hover:!text-white hover:bg-[#40c9c6] p-3 rounded"  
 >  
 <svg-icon icon-class="uv" size="3em" />  
 </div>  
 <div class="flex flex-col space-y-3">  
 <div class="text-[var(--el-text-color-secondary)]">访问数</div>  
 <div class="text-lg text-right">  
 {{ Math.round(visitCountOutput) }}  
 </div>  
 </div>  
 </div>  
 </el-col>  
 <!--消息数-->  
 <el-col :xs="24" :sm="12" :lg="6" class="mb-4">  
 <div class="data-box">  
 <div  
 class="text-[#36a3f7] hover:!text-white hover:bg-[#36a3f7] p-3 rounded"  
 >  
 <svg-icon icon-class="message" size="3em" />  
 </div>  
 <div class="flex flex-col space-y-3">  
 <div class="text-[var(--el-text-color-secondary)]">模型数</div>  
 <div class="text-lg text-right">  
 {{ Math.round(messageCountOutput) }}  
 </div>  
 </div>  
 </div>  
 </el-col>  
 <el-col :xs="24" :sm="12" :lg="6" class="mb-4">  
 <div class="data-box">  
 <div  
 class="text-[#f4516c] hover:!text-white hover:bg-[#f4516c] p-3 rounded"  
 >  
 <svg-icon icon-class="money" size="3em" />  
 </div>  
 <div class="flex flex-col space-y-3">  
 <div class="text-[var(--el-text-color-secondary)]">  
 累计去雾数量  
 </div>  
 <div class="text-lg text-right">  
 {{ Math.round(amountOutput) }}  
 </div>  
 </div>  
 </div>  
 </el-col>  
 <el-col :xs="24" :sm="12" :lg="6" class="mb-2">  
 <div class="data-box">  
 <div  
 class="text-[#34bfa3] hover:!text-white hover:bg-[#34bfa3] p-3 rounded"  
 >  
 <svg-icon icon-class="shopping" size="3em" />  
 </div>  
 <div class="flex flex-col space-y-3">  
 <div class="text-[var(--el-text-color-secondary)]">  
 累计评估数量  
 </div>  
 <div class="text-lg text-right">  
 {{ Math.round(orderCountOutput) }}  
 </div>  
 </div>  
 </div>  
 </el-col>  
 </el-row>  
 <!-- Echarts 图表 -->  
 <el-row :gutter="40">  
 <el-col :sm="24" :lg="8" class="mb-4">  
 <BarChart  
 id="barChart"  
 height="400px"  
 width="100%"  
 class="bg-[var(--el-bg-color-overlay)]"  
 />  
 </el-col>  
 <el-col :xs="24" :sm="12" :lg="8" class="mb-4">  
 <PieChart  
 id="pieChart"  
 height="400px"  
 width="100%"  
 class="bg-[var(--el-bg-color-overlay)]"  
 />  
 </el-col>  
 <el-col :xs="24" :sm="12" :lg="8" class="mb-4">  
 <RadarChart  
 id="radarChart"  
 height="400px"  
 width="100%"  
 class="bg-[var(--el-bg-color-overlay)]"  
 />  
 </el-col>  
 </el-row>  
 </div>  
</template>  
<style lang="scss" scoped>  
.dashboard-container {  
 position: relative;  
 padding: 24px;  
 .user-avatar {  
 width: 40px;  
 height: 40px;  
 border-radius: 50%;  
 }  
 .data-box {  
 display: flex;  
 justify-content: space-between;  
 padding: 20px;  
 font-weight: bold;  
 color: var(--el-text-color-regular);  
 background: var(--el-bg-color-overlay);  
 border-color: var(--el-border-color);  
 box-shadow: var(--el-box-shadow-dark);  
 }  
 .svg-icon {  
 fill: currentcolor !important;  
 }  
}  
</style>  
<!-- 线 + 柱混合图 -->  
<template>  
 <el-card>  
 <template #header>  
 <div class="title">  
 去雾效果柱状图  
 <el-tooltip effect="dark" content="点击试试下载" placement="bottom">  
 <i-ep-download class="download" @click="downloadEchart" />  
 </el-tooltip>  
 </div>  
 </template>  
 <div :id="id" :class="className" :style="{ height, width }"></div>  
 </el-card>  
</template>  
<script setup lang="ts">  
 import \* as echarts from "echarts";  
 const props = defineProps({  
 id: {  
 type: String,  
 default: "barChart",  
 },  
 className: {  
 type: String,  
 default: "",  
 },  
 width: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 height: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 });  
 const options = {  
 grid: {  
 left: "2%",  
 right: "2%",  
 bottom: "10%",  
 containLabel: true,  
 },  
 tooltip: {  
 trigger: "axis",  
 axisPointer: {  
 type: "cross",  
 crossStyle: {  
 color: "#999",  
 },  
 },  
 },  
 legend: {  
 x: "center",  
 y: "bottom",  
 data: ["有雾图像", "无雾图像", "PSNR", "SSIM"],  
 textStyle: {  
 color: "#999",  
 },  
 },  
 xAxis: [  
 {  
 type: "category",  
 data: [  
 "C2PNet",  
 "DehazeFormer",  
 "MB-TaylorFormer",  
 "MixDehazeNet",  
 "RIDCP",  
 ],  
 axisPointer: {  
 type: "shadow",  
 },  
 },  
 ],  
 yAxis: [  
 {  
 type: "value",  
 min: 0,  
 max: 3000,  
 interval: 500,  
 axisLabel: {  
 formatter: "{value} ",  
 },  
 },  
 {  
 type: "value",  
 min: 0,  
 max: 50,  
 interval: 10,  
 axisLabel: {  
 formatter: "{value}%",  
 },  
 },  
 ],  
 series: [  
 {  
 name: "有雾图像",  
 type: "bar",  
 data: [1200, 500, 2500, 1800, 800],  
 barWidth: 20,  
 itemStyle: {  
 color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [  
 { offset: 0, color: "#83bff6" },  
 { offset: 0.5, color: "#188df0" },  
 { offset: 1, color: "#188df0" },  
 ]),  
 },  
 },  
 {  
 name: "无雾图像",  
 type: "bar",  
 data: [3000, 1000, 2400, 1600, 800],  
 barWidth: 20,  
 itemStyle: {  
 color: new echarts.graphic.LinearGradient(0, 0, 0, 1, [  
 { offset: 0, color: "#25d73c" },  
 { offset: 0.5, color: "#1bc23d" },  
 { offset: 1, color: "#179e61" },  
 ]),  
 },  
 },  
 {  
 name: "PSNR",  
 type: "line",  
 yAxisIndex: 1,  
 data: [25, 30, 26, 38, 41],  
 itemStyle: {  
 color: "#67C23A",  
 },  
 },  
 {  
 name: "SSIM",  
 type: "line",  
 yAxisIndex: 1,  
 data: [14, 25, 30, 35, 40],  
 itemStyle: {  
 color: "#409EFF",  
 },  
 },  
 ],  
 };  
 const chart = ref<any>("");  
 onMounted(() => {  
 // 图表初始化  
 chart.value = markRaw(  
 echarts.init(document.getElementById(props.id) as HTMLDivElement)  
 );  
 chart.value.setOption(options);  
 // 大小自适应  
 window.addEventListener("resize", () => {  
 chart.value.resize();  
 });  
 });  
 const downloadEchart = () => {  
 // 获取画布图表地址信息  
 const img = new Image();  
 img.src = chart.value.getDataURL({  
 type: "png",  
 pixelRatio: 1,  
 backgroundColor: "#fff",  
 });  
 // 当图片加载完成后，生成 URL 并下载  
 img.onload = () => {  
 const canvas = document.createElement("canvas");  
 canvas.width = img.width;  
 canvas.height = img.height;  
 const ctx = canvas.getContext("2d");  
 if (ctx) {  
 ctx.drawImage(img, 0, 0, img.width, img.height);  
 const link = document.createElement("a");  
 link.download = `去雾效果图.png`;  
 link.href = canvas.toDataURL("image/png", 0.9);  
 document.body.appendChild(link);  
 link.click();  
 link.remove();  
 }  
 };  
 };  
</script>  
<style lang="scss" scoped>  
 .title {  
 display: flex;  
 justify-content: space-between;  
 .download {  
 cursor: pointer;  
 &:hover {  
 color: #409eff;  
 }  
 }  
 }  
</style>  
<!-- 漏斗图 -->  
<template>  
 <div :id="id" :class="className" :style="{ height, width }"></div>  
</template>  
<script setup lang="ts">  
 import \* as echarts from "echarts";  
 const props = defineProps({  
 id: {  
 type: String,  
 default: "funnelChart",  
 },  
 className: {  
 type: String,  
 default: "",  
 },  
 width: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 height: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 });  
 const options = {  
 title: {  
 show: true,  
 text: "订单线索转化漏斗图",  
 x: "center",  
 padding: 15,  
 textStyle: {  
 fontSize: 18,  
 fontStyle: "normal",  
 fontWeight: "bold",  
 color: "#337ecc",  
 },  
 },  
 grid: {  
 left: "2%",  
 right: "2%",  
 bottom: "10%",  
 containLabel: true,  
 },  
 legend: {  
 x: "center",  
 y: "bottom",  
 data: ["Show", "Click", "Visit", "Inquiry", "Order"],  
 },  
 series: [  
 {  
 name: "Funnel",  
 type: "funnel",  
 left: "20%",  
 top: 60,  
 bottom: 60,  
 width: "60%",  
 sort: "descending",  
 gap: 2,  
 label: {  
 show: true,  
 position: "inside",  
 },  
 labelLine: {  
 length: 10,  
 lineStyle: {  
 width: 1,  
 type: "solid",  
 },  
 },  
 itemStyle: {  
 borderColor: "#fff",  
 borderWidth: 1,  
 },  
 emphasis: {  
 label: {  
 fontSize: 20,  
 },  
 },  
 data: [  
 { value: 60, name: "Visit" },  
 { value: 40, name: "Inquiry" },  
 { value: 20, name: "Order" },  
 { value: 80, name: "Click" },  
 { value: 100, name: "Show" },  
 ],  
 },  
 ],  
 };  
 onMounted(() => {  
 const chart = echarts.init(  
 document.getElementById(props.id) as HTMLDivElement  
 );  
 chart.setOption(options);  
 window.addEventListener("resize", () => {  
 chart.resize();  
 });  
 });  
</script>  
<!-- 饼图 -->  
<template>  
 <el-card>  
 <template #header> 去雾方法饼图 </template>  
 <div :id="id" :class="className" :style="{ height, width }"></div>  
 </el-card>  
</template>  
<script setup lang="ts">  
 import \* as echarts from "echarts";  
 const props = defineProps({  
 id: {  
 type: String,  
 default: "pieChart",  
 },  
 className: {  
 type: String,  
 default: "",  
 },  
 width: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 height: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 });  
 const options = {  
 grid: {  
 left: "2%",  
 right: "2%",  
 bottom: "10%",  
 containLabel: true,  
 },  
 legend: {  
 top: "bottom",  
 textStyle: {  
 color: "#999",  
 },  
 },  
 series: [  
 {  
 name: "Nightingale Chart",  
 type: "pie",  
 radius: [50, 130],  
 center: ["50%", "50%"],  
 roseType: "area",  
 itemStyle: {  
 borderRadius: 1,  
 color: function (params: any) {  
 //自定义颜色  
 const colorList = ["#409EFF", "#67C23A", "#E6A23C", "#F56C6C"];  
 return colorList[params.dataIndex];  
 },  
 },  
 data: [  
 { value: 58, name: "解码器-编码器" },  
 { value: 27, name: "Transformer" },  
 { value: 10, name: "无监督" },  
 { value: 5, name: "物理模型" },  
 ],  
 },  
 ],  
 };  
 onMounted(() => {  
 const chart = echarts.init(  
 document.getElementById(props.id) as HTMLDivElement  
 );  
 chart.setOption(options);  
 window.addEventListener("resize", () => {  
 chart.resize();  
 });  
 });  
</script>  
<!-- 雷达图 -->  
<template>  
 <el-card>  
 <template #header> 数据集情况雷达图 </template>  
 <div :id="id" :class="className" :style="{ height, width }"></div>  
 </el-card>  
</template>  
<script setup lang="ts">  
 import \* as echarts from "echarts";  
 const props = defineProps({  
 id: {  
 type: String,  
 default: "radarChart",  
 },  
 className: {  
 type: String,  
 default: "",  
 },  
 width: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 height: {  
 type: String,  
 default: "200px",  
 required: true,  
 },  
 });  
 const options = {  
 grid: {  
 left: "2%",  
 right: "2%",  
 bottom: "10%",  
 containLabel: true,  
 },  
 legend: {  
 x: "center",  
 y: "bottom",  
 data: ["简单雾霾图", "困难雾霾图", "真实雾霾图"],  
 textStyle: {  
 color: "#999",  
 },  
 },  
 radar: {  
 // shape: 'circle',  
 radius: "60%",  
 indicator: [  
 { name: "RESIDE" },  
 { name: "Dense-Haze" },  
 { name: "I-Haze" },  
 { name: "O-Haze" },  
 { name: "RS-Haze" },  
 { name: "NH-Haze" },  
 ],  
 },  
 series: [  
 {  
 name: "Budget vs spending",  
 type: "radar",  
 itemStyle: {  
 borderRadius: 6,  
 color: function (params: any) {  
 //自定义颜色  
 const colorList = ["#409EFF", "#67C23A", "#E6A23C", "#F56C6C"];  
 return colorList[params.dataIndex];  
 },  
 },  
 data: [  
 {  
 value: [400, 100, 200, 600, 300, 100],  
 name: "简单雾霾图",  
 },  
 {  
 value: [300, 100, 100, 200, 600, 100],  
 name: "困难雾霾图",  
 },  
 {  
 value: [800, 300, 200, 100, 600, 500],  
 name: "真实雾霾图",  
 },  
 ],  
 },  
 ],  
 };  
 onMounted(() => {  
 const chart = echarts.init(  
 document.getElementById(props.id) as HTMLDivElement  
 );  
 chart.setOption(options);  
 window.addEventListener("resize", () => {  
 chart.resize();  
 });  
 });  
</script>  
<script setup lang="ts">  
 import { calculateIndexApi, dehazeApi, getModelApi } from "@/api/dehaze";  
 import { ModelInfo } from "@/api/dehaze/types";  
 import SingleUploadPy from "@/components/Upload/SingleUploadPy.vue";  
 import { useAppStore } from "@/store/modules/app";  
 import { imageBaseURL } from "@/utils/request-py";  
 import { ElMessage } from "element-plus";  
 const dehazeModels = ref<ModelInfo[]>();  
 const selectedDehazeModel = ref("");  
 const appStore = useAppStore();  
 onMounted(async () => {  
 const { data } = await getModelApi();  
 dehazeModels.value = data;  
 });  
 const hazeImage = reactive({  
 name: "",  
 height: 0,  
 width: 0,  
 });  
 const clearImage = reactive({  
 name: "",  
 height: 0,  
 width: 0,  
 });  
 const outputImage = reactive({  
 name: "",  
 height: 0,  
 width: 0,  
 });  
 const dehazedImgUrl = computed(() => {  
 return outputImage.name ? imageBaseURL + "/" + outputImage.name + "/" : "";  
 });  
 const isLoading = ref(false);  
 const loadingText = ref("开始去雾");  
 async function dehazeImage() {  
 if (selectedDehazeModel.value.length === 0) {  
 ElMessage.warning("请选择去雾模型！");  
 return;  
 }  
 if (hazeImage.name.length === 0) {  
 ElMessage.warning("请上传有雾图像！");  
 return;  
 }  
 isLoading.value = true;  
 loadingText.value = "正在去雾";  
 try {  
 const { data } = await dehazeApi(hazeImage.name, selectedDehazeModel.value);  
 outputImage.name = data.image\_name;  
 } finally {  
 isLoading.value = false;  
 loadingText.value = "开始去雾";  
 }  
 }  
 const psnr = ref(0);  
 const psnrEvaluate = computed(() => {  
 if (psnr.value > 40) return "奇迹";  
 else if (psnr.value > 30) return "优秀";  
 else if (psnr.value > 25) return "良好";  
 else if (psnr.value > 20) return "一般";  
 else if (psnr.value > 1) return "不及格";  
 else return "-";  
 });  
 const ssim = ref(0);  
 const ssimEvaluate = computed(() => {  
 if (ssim.value > 0.95) return "奇迹";  
 else if (ssim.value > 0.8) return "优秀";  
 else if (ssim.value > 0.6) return "良好";  
 else if (ssim.value > 0.4) return "一般";  
 else if (ssim.value > 0.01) return "不及格";  
 else return "-";  
 });  
 const vi = ref(0);  
 const viEvaluate = computed(() => "-");  
 const ri = ref(0);  
 const riEvaluate = computed(() => "-");  
 const comprehensiveReview = computed(() => {  
 if (ssim.value === 0 || psnr.value === 0) return "-";  
 const score = (psnr.value / 40 + ssim.value / 1) / 2;  
 let result;  
 if (score > 0.95) result = "简直是奇迹";  
 else if (score > 0.8) result = "非常优秀";  
 else if (score > 0.6) result = "良好";  
 else if (score > 0.4) result = "一般";  
 else if (score > 0) result = "太差了！不及格";  
 else result = "无法评价";  
 return `该图像去雾效果${result}，在PSNR表现${psnrEvaluate.value}、SSIM指标上的表现${ssimEvaluate.value}！`;  
 });  
 async function calculateDehazeIndex() {  
 if (  
 hazeImage.height !== clearImage.height ||  
 hazeImage.width !== clearImage.width  
 ) {  
 ElMessage.warning("基准无雾图像和传入的有雾图像分辨率不对应，无法计算指标");  
 return;  
 }  
 if (outputImage.name.length === 0 || clearImage.name.length === 0) {  
 ElMessage.warning("未点击去雾或者未上传基准无雾图像，无法去雾");  
 return;  
 }  
 const { data } = await calculateIndexApi(  
 outputImage.name,  
 clearImage.name,  
 (progressEvent) => {  
 if (progressEvent.total) {  
 evaluatePercentage.value = Math.round(  
 (progressEvent.loaded / progressEvent.total) \* 100  
 );  
 } else {  
 evaluatePercentage.value = 100;  
 }  
 }  
 );  
 psnr.value = Math.floor(parseFloat(data.psnr) \* 100) / 100;  
 ssim.value = Math.floor(parseFloat(data.ssim) \* 10000) / 10000;  
 }  
 function clearEvaluateResult() {  
 clearImage.name = "";  
 clearImage.height = 0;  
 clearImage.width = 0;  
 evaluatePercentage.value = 0;  
 psnr.value = 0;  
 ssim.value = 0;  
 }  
 function getHazeImageInfo(width: number, height: number) {  
 hazeImage.width = width;  
 hazeImage.height = height;  
 }  
 function getClearImageInfo(width: number, height: number) {  
 clearImage.width = width;  
 clearImage.height = height;  
 }  
 function resetForm() {  
 clearEvaluateResult();  
 outputImage.name = "";  
 outputImage.height = 0;  
 outputImage.width = 0;  
 hazeImage.name = "";  
 hazeImage.width = 0;  
 hazeImage.height = 0;  
 selectedDehazeModel.value = "";  
 }  
 const dialogVisible = ref(false);  
 const drawerVisible = ref(false);  
 const evaluatePercentage = ref(0);  
</script>  
<template>  
 <div class="app-container">  
 <h1 style="margin: 5px 0 10px; text-align: center">多模型图像去雾系统</h1>  
 <div class="operate-panel">  
 <el-cascader  
 placeholder="请选择去雾模型"  
 v-model="selectedDehazeModel"  
 :options="dehazeModels"  
 filterable  
 :props="{ emitPath: false }"  
 :clearable="true"  
 />  
 <div class="operate-panel-right">  
 <el-button type="primary" @click="dehazeImage" :loading="isLoading"  
 >{{ loadingText }}  
 </el-button>  
 <el-button @click="dialogVisible = !dialogVisible">评估效果</el-button>  
 <el-button @click="drawerVisible = !drawerVisible">历史记录</el-button>  
 <el-button type="warning" @click="resetForm">重置页面</el-button>  
 </div>  
 </div>  
 <div class="image-show-container">  
 <single-upload-py  
 v-model="hazeImage.name"  
 title="上传有雾图像"  
 @get-image-info="getHazeImageInfo"  
 />  
 <span style="margin: 0 20px"></span>  
 <el-image  
 style="width: 35vmax; height: 35vmax"  
 :src="dehazedImgUrl"  
 fit="fill"  
 alt="@/assets/photo.png"  
 >  
 <template #error>  
 <div class="image-show-placeholder">  
 <div class="image-show-text">  
 请上传有雾图像<br />并点击"开始去雾"获取无雾图像  
 </div>  
 </div>  
 </template>  
 </el-image>  
 </div>  
 <el-dialog  
 v-model="dialogVisible"  
 :width="appStore.device === 'mobile' ? '100%' : '70%'"  
 title="评估效果"  
 top="2vh"  
 class="dialog-class"  
 >  
 <el-alert  
 :closable="false"  
 show-icon  
 description="评估去雾模型的去雾效果需要上传用于比较的基准无雾图像，系统才能够计算出模型去雾效果和真实的无雾图像之间的差距。真实的无雾图像需要和原始的有雾图像拍摄位置、图像宽高大小一致，否则会出现错误"  
 />  
 <div class="dialog-content">  
 <single-upload-py  
 style="margin-right: 20px"  
 v-model="clearImage.name"  
 title="上传基准无雾图像"  
 @get-image-info="getClearImageInfo"  
 />  
 <div class="dialog-content-right">  
 <div class="dialog-content-right-up">  
 <el-button type="primary" @click="calculateDehazeIndex"  
 >开始评估  
 </el-button>  
 <el-button type="info" @click="clearEvaluateResult"  
 >清空结果  
 </el-button>  
 </div>  
 <div class="dialog-content-right-center">  
 <div style="display: flex; justify-content: center; width: 100%">  
 <el-progress  
 v-if="evaluatePercentage !== 100"  
 type="dashboard"  
 :percentage="evaluatePercentage"  
 :color="0"  
 />  
 </div>  
 <el-result v-if="evaluatePercentage === 100" icon="success" />  
 </div>  
 <div>  
 <div class="text-class">  
 <span style="margin-right: 16px">雾霾图像分辨率</span>  
 {{ hazeImage.width + " \* " + hazeImage.height }}  
 </div>  
 <div class="text-class">  
 <span style="margin-right: 16px">基准图像分辨率</span>  
 {{ clearImage.width + " \* " + clearImage.height }}  
 </div>  
 <el-descriptions :column="2" size="large">  
 <el-descriptions-item :span="1" :min-width="50" label="PSNR"  
 >{{ psnr }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="评价"  
 >{{ psnrEvaluate }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="SSIM"  
 >{{ ssim }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="评价"  
 >{{ ssimEvaluate }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="VI"  
 >{{ vi }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="评价"  
 >{{ viEvaluate }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="RI"  
 >{{ ri }}  
 </el-descriptions-item>  
 <el-descriptions-item :min-width="50" label="评价"  
 >{{ riEvaluate }}  
 </el-descriptions-item>  
 <el-descriptions-item :width="150" label="综合点评">  
 {{ comprehensiveReview }}  
 </el-descriptions-item>  
 </el-descriptions>  
 </div>  
 </div>  
 </div>  
 </el-dialog>  
 <el-drawer  
 :size="appStore.device === 'mobile' ? '100%' : '650'"  
 v-model="drawerVisible"  
 direction="rtl"  
 title="历史记录"  
 >  
 <el-table>  
 <el-table-column prop="id" label="序号" />  
 <el-table-column prop="hazeModel" label="去雾模型" />  
 <el-table-column prop="type" label="操作类别" />  
 <el-table-column prop="operationTime" label="操作时间" />  
 <el-table-column prop="detail" label="详情" />  
 </el-table>  
 </el-drawer>  
 </div>  
</template>  
<style>  
 .operate-panel {  
 display: flex;  
 flex-wrap: wrap;  
 justify-content: space-between;  
 margin: 0 15px 20px;  
 }  
 .operate-panel-right {  
 display: flex;  
 flex-wrap: wrap;  
 align-content: space-between;  
 }  
 .image-show-container {  
 display: flex;  
 flex-wrap: wrap;  
 justify-content: center;  
 margin: 0 15px;  
 }  
 .image-show-placeholder {  
 display: flex;  
 align-content: center;  
 align-items: center;  
 justify-content: center;  
 width: 35vmax;  
 height: 35vmax;  
 border: 1px dashed var(--el-border-color);  
 border-radius: 6px;  
 }  
 .image-show-text {  
 text-align: center;  
 }  
 .dialog-content {  
 display: flex;  
 flex-wrap: wrap;  
 justify-content: space-between;  
 margin-top: 15px;  
 }  
 .dialog-content-right {  
 display: flex;  
 flex-direction: column;  
 flex-grow: 1;  
 justify-content: space-between;  
 }  
 .dialog-content-right-up {  
 display: flex;  
 justify-content: space-evenly;  
 }  
 .dialog-content-right-center {  
 height: 130px;  
 }  
 .text-class {  
 padding: 1px 1px 16px;  
 color: var(--el-text-color-primary);  
 }  
</style>  
<template>  
 <div class="login-container">  
 <el-form  
 ref="loginFormRef"  
 :model="loginData"  
 :rules="loginRules"  
 class="login-form"  
 >  
 <div class="flex text-white items-center py-4 title-wrap">  
 <span class="text-2xl flex-1 text-center title">  
 {{ $t("login.title") }}  
 </span>  
 <lang-select class="text-white! cursor-pointer" />  
 </div>  
 <el-form-item prop="username">  
 <div class="p-2 text-white">  
 <svg-icon icon-class="user" />  
 </div>  
 <el-input  
 ref="username"  
 v-model="loginData.username"  
 class="flex-1"  
 size="large"  
 :placeholder="$t('login.username')"  
 name="username"  
 />  
 </el-form-item>  
 <el-tooltip  
 :disabled="isCapslock === false"  
 content="Caps lock is On"  
 placement="right"  
 >  
 <el-form-item prop="password">  
 <span class="p-2 text-white">  
 <svg-icon icon-class="password" />  
 </span>  
 <el-input  
 v-model="loginData.password"  
 class="flex-1"  
 placeholder="密码"  
 :type="passwordVisible === false ? 'password' : 'input'"  
 size="large"  
 name="password"  
 @keyup="checkCapslock"  
 @keyup.enter="handleLogin"  
 />  
 <span class="mr-2" @click="passwordVisible = !passwordVisible">  
 <svg-icon  
 :icon-class="passwordVisible === false ? 'eye' : 'eye-open'"  
 class="text-white cursor-pointer"  
 />  
 </span>  
 </el-form-item>  
 </el-tooltip>  
 <!-- 验证码 -->  
 <el-form-item prop="verifyCode">  
 <span class="p-2 text-white">  
 <svg-icon icon-class="verify\_code" />  
 </span>  
 <el-input  
 v-model="loginData.verifyCode"  
 auto-complete="off"  
 :placeholder="$t('login.verifyCode')"  
 class="w-[60%]"  
 @keyup.enter="handleLogin"  
 />  
 <div class="captcha">  
 <img :src="captchaBase64" @click="getCaptcha" />  
 </div>  
 </el-form-item>  
 <el-button  
 size="default"  
 :loading="loading"  
 type="primary"  
 class="w-full"  
 @click.prevent="handleLogin"  
 >{{ $t("login.login") }}  
 </el-button>  
 <!-- 账号密码提示 -->  
 <div class="mt-4 text-white text-sm">  
 <span>去雾体验账号: dehaze</span>  
 <span class="ml-4"> 密码: 123456</span>  
 </div>  
 </el-form>  
 </div>  
</template>  
<script setup lang="ts">  
 import router from "@/router";  
 import LangSelect from "@/components/LangSelect/index.vue";  
 import SvgIcon from "@/components/SvgIcon/index.vue";  
 // 状态管理依赖  
 import { useUserStore } from "@/store/modules/user";  
 // API依赖  
 import { LocationQuery, LocationQueryValue, useRoute } from "vue-router";  
 import { getCaptchaApi } from "@/api/auth";  
 import { LoginData } from "@/api/auth/types";  
 const userStore = useUserStore();  
 const route = useRoute();  
 /\*\*  
 \* 按钮loading  
 \*/  
 const loading = ref(false);  
 /\*\*  
 \* 是否大写锁定  
 \*/  
 const isCapslock = ref(false);  
 /\*\*  
 \* 密码是否可见  
 \*/  
 const passwordVisible = ref(false);  
 /\*\*  
 \* 验证码图片Base64字符串  
 \*/  
 const captchaBase64 = ref();  
 /\*\*  
 \* 登录表单引用  
 \*/  
 const loginFormRef = ref(ElForm);  
 const loginData = ref<LoginData>({  
 username: "",  
 password: "",  
 });  
 const loginRules = {  
 username: [{ required: true, trigger: "blur" }],  
 password: [{ required: true, trigger: "blur", validator: passwordValidator }],  
 verifyCode: [{ required: true, trigger: "blur" }],  
 };  
 /\*\*  
 \* 密码校验器  
 \*/  
 function passwordValidator(rule: any, value: any, callback: any) {  
 if (value.length < 6) {  
 callback(new Error("The password can not be less than 6 digits"));  
 } else {  
 callback();  
 }  
 }  
 /\*\*  
 \* 检查输入大小写状态  
 \*/  
 function checkCapslock(e: any) {  
 const { key } = e;  
 isCapslock.value = key && key.length === 1 && key >= "A" && key <= "Z";  
 }  
 /\*\*  
 \* 获取验证码  
 \*/  
 function getCaptcha() {  
 getCaptchaApi().then(({ data }) => {  
 const { verifyCodeBase64, verifyCodeKey } = data;  
 loginData.value.verifyCodeKey = verifyCodeKey;  
 captchaBase64.value = verifyCodeBase64;  
 });  
 }  
 /\*\*  
 \* 登录  
 \*/  
 function handleLogin() {  
 loginFormRef.value.validate((valid: boolean) => {  
 if (valid) {  
 loading.value = true;  
 userStore  
 .login(loginData.value)  
 .then(() => {  
 const query: LocationQuery = route.query;  
 const redirect = (query.redirect as LocationQueryValue) ?? "/";  
 const otherQueryParams = Object.keys(query).reduce(  
 (acc: any, cur: string) => {  
 if (cur !== "redirect") {  
 acc[cur] = query[cur];  
 }  
 return acc;  
 },  
 {}  
 );  
 router.push({ path: redirect, query: otherQueryParams });  
 })  
 .catch(() => {  
 // 验证失败，重新生成验证码  
 getCaptcha();  
 })  
 .finally(() => {  
 loading.value = false;  
 });  
 }  
 });  
 }  
 onMounted(() => {  
 getCaptcha();  
 });  
</script>  
<style lang="scss" scoped>  
 .login-container {  
 width: 100%;  
 min-height: 100%;  
 overflow: hidden;  
 background-color: #2d3a4b;  
 .title-wrap {  
 filter: contrast(30);  
 .title {  
 letter-spacing: 4px;  
 animation: showup 3s forwards;  
 }  
 @keyframes showup {  
 0% {  
 letter-spacing: -20px;  
 }  
 100% {  
 letter-spacing: 4px;  
 }  
 }  
 }  
 .login-form {  
 width: 520px;  
 max-width: 100%;  
 padding: 160px 35px 0;  
 margin: 0 auto;  
 overflow: hidden;  
 .captcha {  
 position: absolute;  
 top: 0;  
 right: 0;  
 img {  
 width: 120px;  
 height: 48px;  
 cursor: pointer;  
 }  
 }  
 }  
 }  
 .el-form-item {  
 background: rgb(0 0 0 / 10%);  
 border: 1px solid rgb(255 255 255 / 10%);  
 border-radius: 5px;  
 }  
 .el-input {  
 background: transparent;  
 // 子组件 scoped 无效，使用 :deep  
 :deep(.el-input\_\_wrapper) {  
 padding: 0;  
 background: transparent;  
 box-shadow: none;  
 .el-input\_\_inner {  
 color: #fff;  
 background: transparent;  
 border: 0;  
 border-radius: 0;  
 caret-color: #fff;  
 &:-webkit-autofill {  
 box-shadow: 0 0 0 1000px transparent inset !important;  
 -webkit-text-fill-color: #fff !important;  
 }

// 设置输入框自动填充的延迟属性  
 &:-webkit-autofill,  
 &:-webkit-autofill:hover,  
 &:-webkit-autofill:focus,  
 &:-webkit-autofill:active {  
 transition: color 99999s ease-out, background-color 99999s ease-out;  
 transition-delay: 99999s;  
 }  
 }  
 }  
 }  
</style>

## JAVA后端代码

package com.pei.dehaze;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
@SpringBootApplication  
public class SystemApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(SystemApplication.class, args);  
 }  
}  
package com.youlai.system.controller;  
import cn.hutool.core.util.StrUtil;  
import com.pei.dehaze.common.constant.SecurityConstants;  
import com.pei.dehaze.common.result.Result;  
import com.pei.dehaze.common.util.RequestUtils;  
import com.pei.dehaze.security.captcha.EasyCaptchaService;  
import com.pei.dehaze.model.dto.CaptchaResult;  
import com.pei.dehaze.model.dto.LoginResult;  
import com.pei.dehaze.security.JwtTokenManager;  
import io.jsonwebtoken.Claims;  
import io.swagger.v3.oas.annotations.Parameter;  
import io.swagger.v3.oas.annotations.security.SecurityRequirement;  
import io.swagger.v3.oas.annotations.tags.Tag;  
import io.swagger.v3.oas.annotations.Operation;  
import jakarta.servlet.http.HttpServletRequest;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.web.bind.annotation.\*;  
import java.util.Date;  
import java.util.concurrent.TimeUnit;  
@Tag(name = "认证中心")  
@RestController  
@RequestMapping("/api/v1/auth")  
@RequiredArgsConstructor  
@Slf4j  
public class AuthController {  
 private final AuthenticationManager authenticationManager;  
 private final JwtTokenManager jwtTokenManager;  
 private final EasyCaptchaService easyCaptchaService;  
 private final RedisTemplate redisTemplate;  
 @Operation(summary = "登录")  
 @PostMapping("/login")  
 public Result<LoginResult> login(  
 @Parameter(description = "用户名", example = "admin") @RequestParam String username,  
 @Parameter(description = "密码", example = "123456") @RequestParam String password  
 ) {  
 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(  
 username.toLowerCase().trim(),  
 password  
 );  
 Authentication authentication = authenticationManager.authenticate(authenticationToken);  
 // 生成token  
 String accessToken = jwtTokenManager.createToken(authentication);  
 LoginResult loginResult = LoginResult.builder()  
 .tokenType("Bearer")  
 .accessToken(accessToken)  
 .build();  
 return Result.success(loginResult);  
 }  
 @Operation(summary = "注销", security = {@SecurityRequirement(name = SecurityConstants.TOKEN\_KEY)})  
 @DeleteMapping("/logout")  
 public Result logout(HttpServletRequest request) {  
 String token = RequestUtils.resolveToken(request);  
 if (StrUtil.isNotBlank(token)) {  
 Claims claims = jwtTokenManager.getTokenClaims(token);  
 String jti = claims.get("jti", String.class);  
 Date expiration = claims.getExpiration();  
 if (expiration != null) {  
 // 有过期时间，在token有效时间内存入黑名单，超出时间移除黑名单节省内存占用  
 long ttl = (expiration.getTime() - System.currentTimeMillis());  
 redisTemplate.opsForValue().set(SecurityConstants.BLACK\_TOKEN\_CACHE\_PREFIX + jti, null, ttl, TimeUnit.MILLISECONDS);  
 } else {  
 // 无过期时间，永久加入黑名单  
 redisTemplate.opsForValue().set(SecurityConstants.BLACK\_TOKEN\_CACHE\_PREFIX + jti, null);  
 }  
 }  
 SecurityContextHolder.clearContext();  
 return Result.success("注销成功");  
 }  
 @Operation(summary = "获取验证码")  
 @GetMapping("/captcha")  
 public Result getCaptcha() {  
 CaptchaResult captcha = easyCaptchaService.getCaptcha();  
 return Result.success(captcha);  
 }  
}

## python后端代码

#!/usr/bin/env python
  
"""Django's command-line utility for administrative tasks."""
  
import os
  
import sys
  
def main():
  
 """Run administrative tasks."""
  
 os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'dehazing\_system.settings')
  
 try:
  
 from django.core.management import execute\_from\_command\_line
  
 except ImportError as exc:
  
 raise ImportError(
  
 "Couldn't import Django. Are you sure it's installed and "
  
 "available on your PYTHONPATH environment variable? Did you "
  
 "forget to activate a virtual environment?"
  
 ) from exc
  
 execute\_from\_command\_line(sys.argv)
  
  
if \_\_name\_\_ == '\_\_main\_\_':
  
 main()
  
  
from pathlib import Path
  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent
  
  
DEBUG = True
  
  
ALLOWED\_HOSTS = ['\*']
  
  
# APPEND\_SLASH = False
  
# 最大文件上传大小 20MB（单位：字节）
  
DATA\_UPLOAD\_MAX\_MEMORY\_SIZE = 20971520
  
  
INSTALLED\_APPS = [
  
 'django.contrib.admin',
  
 'django.contrib.auth',
  
 'django.contrib.contenttypes',
  
 'django.contrib.sessions',
  
 'django.contrib.messages',
  
 'django.contrib.staticfiles',
  
]
  
  
MIDDLEWARE = [
  
 'django.middleware.security.SecurityMiddleware',
  
 'django.contrib.sessions.middleware.SessionMiddleware',
  
 'django.middleware.common.CommonMiddleware',
  
 'django.contrib.auth.middleware.AuthenticationMiddleware',
  
 'django.contrib.messages.middleware.MessageMiddleware',
  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',
  
]
  
  
ROOT\_URLCONF = 'dehazing\_system.urls'
  
  
TEMPLATES = [
  
 {
  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',
  
 'DIRS': [BASE\_DIR / 'templates']
  
 ,
  
 'APP\_DIRS': True,
  
 'OPTIONS': {
  
 'context\_processors': [
  
 'django.template.context\_processors.debug',
  
 'django.template.context\_processors.request',
  
 'django.contrib.auth.context\_processors.auth',
  
 'django.contrib.messages.context\_processors.messages',
  
 ],
  
 },
  
 },
  
]
  
  
WSGI\_APPLICATION = 'dehazing\_system.wsgi.application'
  
  
DATABASES = {
  
 'default': {
  
 'ENGINE': 'django.db.backends.sqlite3',
  
 'NAME': BASE\_DIR / 'db.sqlite3',
  
 }
  
}
  
  
AUTH\_PASSWORD\_VALIDATORS = [
  
 {
  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',
  
 },
  
 {
  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',
  
 },
  
 {
  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',
  
 },
  
 {
  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',
  
 },
  
]
  
LANGUAGE\_CODE = 'en-US'
  
TIME\_ZONE = 'UTC'
  
USE\_I18N = True
  
USE\_TZ = True
  
  
STATIC\_URL = 'static/'
  
  
DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'
  
  
import os
  
import uuid
  
import torch
  
DEVICE = 'cuda:0' if torch.cuda.is\_available() else 'cpu'
  
PROJECT\_PATH = os.path.dirname(os.path.abspath(\_\_file\_\_))
  
DATA\_PATH = os.path.join(PROJECT\_PATH, "data")
  
MODEL\_PATH = os.path.join(PROJECT\_PATH, "trained\_model")
  
if \_\_name\_\_ == '\_\_main\_\_':
  
 image\_name = str(uuid.uuid4()) + ".png"
  
 image\_path = os.path.join(DATA\_PATH, image\_name)
  
 print(image\_path)
  
  
import json
  
import os.path
  
import traceback
  
import uuid
  
from django.http import HttpResponse, HttpRequest
  
import benchmark.C2PNet.run
  
import benchmark.DehazeFormer.run
  
import benchmark.MixDehazeNet.run
  
import benchmark.CMFNet.run
  
import benchmark.DEANet.run
  
import benchmark.FogRemoval.run
  
import benchmark.ITBdehaze.run
  
import benchmark.RIDCP.run
  
from benchmark.metrics import calculate
  
from global\_variable import DATA\_PATH
  
  
dehaze\_model = {
  
 'C2PNet/OTS.pkl': benchmark.C2PNet.run.dehaze,
  
 'C2PNet/ITS.pkl': benchmark.C2PNet.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-b.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-d.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-l.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-m.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-s.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-t.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/indoor/dehazeformer-w.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/outdoor/dehazeformer-b.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/outdoor/dehazeformer-m.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/outdoor/dehazeformer-s.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/outdoor/dehazeformer-t.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/reside6k/dehazeformer-b.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/reside6k/dehazeformer-m.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/reside6k/dehazeformer-s.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/reside6k/dehazeformer-t.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/rshaze/dehazeformer-b.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/rshaze/dehazeformer-m.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/rshaze/dehazeformer-s.pth': benchmark.DehazeFormer.run.dehaze,
  
 'DehazeFormer/rshaze/dehazeformer-t.pth': benchmark.DehazeFormer.run.dehaze,
  
 'MixDehazeNet/haze4k/MixDehazeNet-l.pth': benchmark.MixDehazeNet.run.dehaze,
  
 'MixDehazeNet/indoor/MixDehazeNet-l.pth': benchmark.MixDehazeNet.run.dehaze,
  
 'MixDehazeNet/indoor/MixDehazeNet-b.pth': benchmark.MixDehazeNet.run.dehaze,
  
 'MixDehazeNet/outdoor/MixDehazeNet-b.pth': benchmark.MixDehazeNet.run.dehaze,
  
 'MixDehazeNet/outdoor/MixDehazeNet-l.pth': benchmark.MixDehazeNet.run.dehaze,
  
 'MixDehazeNet/outdoor/MixDehazeNet-s.pth': benchmark.MixDehazeNet.run.dehaze,
  
 'CMFNet/dehaze\_I\_OHaze\_CMFNet.pth': benchmark.CMFNet.run.dehaze,
  
 'DEA-Net/HAZE4K/PSNR3426\_SSIM9885.pth': benchmark.DEANet.run.dehaze,
  
 'DEA-Net/ITS/PSNR4131\_SSIM9945.pth': benchmark.DEANet.run.dehaze,
  
 'DEA-Net/OTS/PSNR3659\_SSIM9897.pth': benchmark.DEANet.run.dehaze,
  
 'FogRemoval/NH-HAZE\_params\_0100000.pt': benchmark.FogRemoval.run.dehaze,
  
 'ITBdehaze/best.pkl': benchmark.ITBdehaze.run.dehaze,
  
 'RIDCP/pretrained\_RIDCP.pth': benchmark.RIDCP.run.dehaze,
  
}
  
  
  
def ok\_response(data):
  
 message = {
  
 'code': '00000',
  
 'msg': '一切ok',
  
 'data': data
  
 }
  
 return HttpResponse(json.dumps(message), content\_type='application/json')
  
  
  
def error\_response(code, msg):
  
 message = {
  
 'code': code,
  
 'msg': msg,
  
 'data': None
  
 }
  
 return HttpResponse(json.dumps(message), content\_type='application/json')
  
  
  
def get\_model(request: HttpRequest):
  
 result = []
  
 for index, key in enumerate(dehaze\_model):
  
 # 首先将字符串按照 / 分割成数组
  
 parts = key.split('/')
  
 # 然后获取当前已经组装好的结果，准备继续向内部添加当前结点
  
 current = result
  
 # 遍历该数组，创建嵌套的数组
  
 for i, part in enumerate(parts):
  
 # 如果当前元素是数组的最后一个元素，也就是'DehazeFormer/indoor/dehazeformer-b.pth' 中的 'dehazeformer-b.pth'
  
 # 那么就将当前元素放入结果数组中
  
 if i == len(parts) - 1:
  
 current.append({'value': key, 'label': part.split(".")[0]})
  
 else:
  
 # 如果不是最后一个元素，则遍历结果数组，直到找到一个key和当前的元素一样的
  
 # 就更改当前结果数组
  
 found = False
  
 for child in current:
  
 if child['value'] == part:
  
 current = child['children']
  
 found = True
  
 break
  
 # 如果没有找到则创建一个新元素，插入到结果数组中，并且更新当前结果数组
  
 if not found:
  
 new\_node = {'value': part, 'label': part, 'children': []}
  
 current.append(new\_node)
  
 current = new\_node['children']
  
 return ok\_response(result)
  
  
  
def upload\_image(request: HttpRequest):
  
 image\_name = str(uuid.uuid4()) + ".png"
  
 image\_path = os.path.join(DATA\_PATH, image\_name)
  
 image = request.body
  
 # 保存前端传来的图片
  
 with open(image\_path, "wb") as destination:
  
 destination.write(image)
  
 return ok\_response({'image\_name': image\_name})
  
  
  
def download\_image(request: HttpRequest, image\_name: str):
  
 image\_path = os.path.join(DATA\_PATH, image\_name)
  
 with open(image\_path, "rb") as destination:
  
 return HttpResponse(destination.read(), content\_type="image/png")
  
  
  
def dehaze\_image(request: HttpRequest):
  
 data = json.loads(request.body)
  
 haze\_image\_name = data["haze\_image"]
  
 model\_name = data["model\_name"]
  
  
 output\_image\_name = str(uuid.uuid4()) + ".png"
  
 haze\_image\_path = os.path.join(DATA\_PATH, haze\_image\_name)
  
 output\_image\_path = os.path.join(DATA\_PATH, output\_image\_name)
  
  
 try:
  
 dehaze = dehaze\_model.get(model\_name, None)
  
 if dehaze is not None:
  
 dehaze(haze\_image\_path, output\_image\_path, model\_name)
  
 else:
  
 return error\_response('1', "无法找到模型")
  
 except RuntimeError as e:
  
 traceback.print\_exc()
  
 return error\_response('1', e.\_\_str\_\_())
  
  
 return ok\_response({'image\_name': output\_image\_name})
  
  
  
def calculate\_dehaze\_index(request: HttpRequest):
  
 data = json.loads(request.body)
  
 haze\_image\_name = data["haze\_image"]
  
 clear\_image\_name = data["clear\_image"]
  
 haze\_image\_path = os.path.join(DATA\_PATH, haze\_image\_name)
  
 clear\_image\_path = os.path.join(DATA\_PATH, clear\_image\_name)
  
  
 psnr, ssim = calculate(haze\_image\_path, clear\_image\_path)
  
 return ok\_response({'psnr': psnr, 'ssim': ssim})
  
  
from django.contrib import admin
  
from django.urls import path
  
import dehazing\_system.photo
  
urlpatterns = [
  
 path('admin/', admin.site.urls),
  
 path('model/', dehazing\_system.photo.get\_model),
  
 path("upload/", dehazing\_system.photo.upload\_image),
  
 path('download/<str:image\_name>/', dehazing\_system.photo.download\_image),
  
 path('dehazeImage/', dehazing\_system.photo.dehaze\_image),
  
 path('calculateIndex/', dehazing\_system.photo.calculate\_dehaze\_index),
  
]
  
import numpy as np
  
from PIL import Image
  
from skimage.metrics import peak\_signal\_noise\_ratio, structural\_similarity
  
  
  
def calculate(haze\_image\_path: str, clear\_image\_path: str):
  
 haze = Image.open(haze\_image\_path).convert('RGB')
  
 clear = Image.open(clear\_image\_path).convert('RGB')
  
 haze = np.array(haze)
  
 clear = np.array(clear)
  
 current\_psnr = peak\_signal\_noise\_ratio(haze, clear)
  
 current\_ssim = structural\_similarity(haze, clear, channel\_axis=2)
  
 return current\_psnr, current\_ssim
  
  
import torch
  
from PIL import Image
  
import torchvision.transforms as tfs
  
import torchvision.utils as torch\_utils
  
from skimage.metrics import peak\_signal\_noise\_ratio, structural\_similarity
  
from benchmark.C2PNet.model import C2PNet
  
import os
  
# from benchmark.C2PNet.metrics import psnr, ssim
  
from global\_variable import MODEL\_PATH, DEVICE
  
  
def get\_model(model\_name: str):
  
 # 构造模型文件的绝对路径
  
 model\_dir = os.path.join(MODEL\_PATH, model\_name)
  
 net = C2PNet(gps=3, blocks=19)
  
 ckp = torch.load(model\_dir)
  
 net = net.to(DEVICE)
  
 net.load\_state\_dict(ckp['model'])
  
 net.eval()
  
 return net
  
  
def dehaze(haze\_image\_path: str, output\_image\_path: str, model\_name: str = 'C2PNet/OTS.pkl'):
  
 net = get\_model(model\_name)
  
 haze = Image.open(haze\_image\_path).convert('RGB')
  
 haze = tfs.ToTensor()(haze)[None, ::]
  
 haze = haze.to(DEVICE)
  
 with torch.no\_grad():
  
 pred = net(haze)
  
 ts = torch.squeeze(pred.clamp(0, 1).cpu())
  
 torch\_utils.save\_image(ts, output\_image\_path)
  
  
import torch
  
import torch.nn as nn
  
  
  
def default\_conv(in\_channels, out\_channels, kernel\_size, bias=True):
  
 return nn.Conv2d(in\_channels, out\_channels, kernel\_size, padding=(kernel\_size // 2), bias=bias)
  
  
  
class CALayer(nn.Module):
  
 def \_\_init\_\_(self, channel):
  
 super(CALayer, self).\_\_init\_\_()
  
 self.avg\_pool = nn.AdaptiveAvgPool2d(1)
  
 self.ca = nn.Sequential(
  
 nn.Conv2d(channel, channel // 8, 1, padding=0, bias=True),
  
 nn.ReLU(inplace=True),
  
 nn.Conv2d(channel // 8, channel, 1, padding=0, bias=True),
  
 nn.Sigmoid()
  
 )
  
  
 def forward(self, x):
  
 y = self.avg\_pool(x)
  
 y = self.ca(y)
  
 return x \* y
  
  
  
class PDU(nn.Module): # physical block
  
 def \_\_init\_\_(self, channel):
  
 super(PDU, self).\_\_init\_\_()
  
 self.avg\_pool = nn.AdaptiveAvgPool2d(1)
  
 self.ka = nn.Sequential(
  
 nn.Conv2d(channel, channel // 8, 1, padding=0, bias=True),
  
 nn.ReLU(inplace=True),
  
 nn.Conv2d(channel // 8, channel, 1, padding=0, bias=True),
  
 nn.Sigmoid()
  
 )
  
 self.td = nn.Sequential(
  
 default\_conv(channel, channel, 3),
  
 default\_conv(channel, channel // 8, 3),
  
 nn.ReLU(inplace=True),
  
 default\_conv(channel // 8, channel, 3),
  
 nn.Sigmoid()
  
 )
  
  
 def forward(self, x):
  
 a = self.avg\_pool(x)
  
 a = self.ka(a)
  
 t = self.td(x)
  
 j = torch.mul((1 - t), a) + torch.mul(t, x)
  
 return j
  
  
  
class Block(nn.Module): # origin
  
 def \_\_init\_\_(self, conv, dim, kernel\_size, ):
  
 super(Block, self).\_\_init\_\_()
  
 self.conv1 = conv(dim, dim, kernel\_size, bias=True)
  
 self.act1 = nn.ReLU(inplace=True)
  
 self.conv2 = conv(dim, dim, kernel\_size, bias=True)
  
 self.calayer = CALayer(dim)
  
 self.pdu = PDU(dim)
  
  
 def forward(self, x):
  
 res = self.act1(self.conv1(x))
  
 res = res + x
  
 res = self.conv2(res)
  
 res = self.calayer(res)
  
 res = self.pdu(res)
  
 res += x
  
 return res
  
  
  
class Group(nn.Module):
  
 def \_\_init\_\_(self, conv, dim, kernel\_size, blocks):
  
 super(Group, self).\_\_init\_\_()
  
 modules = [Block(conv, dim, kernel\_size) for \_ in range(blocks)]
  
 modules.append(conv(dim, dim, kernel\_size))
  
 self.gp = nn.Sequential(\*modules)
  
  
 def forward(self, x):
  
 res = self.gp(x)
  
 res += x
  
 return res
  
  
  
class C2PNet(nn.Module):
  
 def \_\_init\_\_(self, gps, blocks, conv=default\_conv):
  
 super(C2PNet, self).\_\_init\_\_()
  
 self.gps = gps
  
 self.dim = 64
  
 kernel\_size = 3
  
 pre\_process = [conv(3, self.dim, kernel\_size)]
  
 assert self.gps == 3
  
 self.g1 = Group(conv, self.dim, kernel\_size, blocks=blocks)
  
 self.g2 = Group(conv, self.dim, kernel\_size, blocks=blocks)
  
 self.g3 = Group(conv, self.dim, kernel\_size, blocks=blocks)
  
 self.ca = nn.Sequential(\*[
  
 nn.AdaptiveAvgPool2d(1),
  
 nn.Conv2d(self.dim \* self.gps, self.dim // 16, 1, padding=0),
  
 nn.ReLU(inplace=True),
  
 nn.Conv2d(self.dim // 16, self.dim \* self.gps, 1, padding=0, bias=True),
  
 nn.Sigmoid()
  
 ])
  
 self.pdu = PDU(self.dim)
  
  
 post\_precess = [
  
 conv(self.dim, self.dim, kernel\_size),
  
 conv(self.dim, 3, kernel\_size)]
  
  
 self.pre = nn.Sequential(\*pre\_process)
  
 self.post = nn.Sequential(\*post\_precess)
  
  
 def forward(self, x1):
  
 x = self.pre(x1)
  
 res1 = self.g1(x)
  
 res2 = self.g2(res1)
  
 res3 = self.g3(res2)
  
 w = self.ca(torch.cat([res1, res2, res3], dim=1))
  
 w = w.view(-1, self.gps, self.dim)[:, :, :, None, None]
  
 out = w[:, 0, ::] \* res1 + w[:, 1, ::] \* res2 + w[:, 2, ::] \* res3
  
 out = self.pdu(out)
  
 x = self.post(out)
  
 return x + x1
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":
  
 net = C2PNet(gps=3, blocks=19)
  
 print(net)
  
  
import math
  
import torch
  
import torch.nn as nn
  
import torch.nn.functional as F
  
import torch.utils.checkpoint as checkpoint
  
from timm.models.layers import DropPath, to\_2tuple, trunc\_normal\_
  
  
  
class Mlp(nn.Module):
  
 def \_\_init\_\_(self, in\_features, hidden\_features=None, out\_features=None, act\_layer=nn.GELU, drop=0.):
  
 super().\_\_init\_\_()
  
 out\_features = out\_features or in\_features
  
 hidden\_features = hidden\_features or in\_features
  
 self.fc1 = nn.Linear(in\_features, hidden\_features)
  
 self.act = act\_layer()
  
 self.fc2 = nn.Linear(hidden\_features, out\_features)
  
 self.drop = nn.Dropout(drop)
  
  
 def forward(self, x):
  
 x = self.fc1(x)
  
 x = self.act(x)
  
 x = self.drop(x)
  
 x = self.fc2(x)
  
 x = self.drop(x)
  
 return x
  
  
  
def window\_partition(x, window\_size):
  
 """
  
 Args:
  
 x: (B, H, W, C)
  
 window\_size (int): window size
  
  
 Returns:
  
 windows: (num\_windows\*B, window\_size, window\_size, C)
  
 """
  
 B, H, W, C = x.shape
  
 x = x.view(B, H // window\_size, window\_size, W // window\_size, window\_size, C)
  
 windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, window\_size, window\_size, C)
  
 return windows
  
  
  
def window\_reverse(windows, window\_size, H, W):
  
 """
  
 Args:
  
 windows: (num\_windows\*B, window\_size, window\_size, C)
  
 window\_size (int): Window size
  
 H (int): Height of image
  
 W (int): Width of image
  
  
 Returns:
  
 x: (B, H, W, C)
  
 """
  
 B = int(windows.shape[0] / (H \* W / window\_size / window\_size))
  
 x = windows.view(B, H // window\_size, W // window\_size, window\_size, window\_size, -1)
  
 x = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(B, H, W, -1)
  
 return x
  
  
  
class WindowAttention(nn.Module):
  
 r""" Window based multi-head self attention (W-MSA) module with relative position bias.
  
 It supports both of shifted and non-shifted window.
  
  
 Args:
  
 dim (int): Number of input channels.
  
 window\_size (tuple[int]): The height and width of the window.
  
 num\_heads (int): Number of attention heads.
  
 qkv\_bias (bool, optional): If True, add a learnable bias to query, key, value. Default: True
  
 qk\_scale (float | None, optional): Override default qk scale of head\_dim \*\* -0.5 if set
  
 attn\_drop (float, optional): Dropout ratio of attention weight. Default: 0.0
  
 proj\_drop (float, optional): Dropout ratio of output. Default: 0.0
  
 """
  
  
 def \_\_init\_\_(self, dim, window\_size, num\_heads, qkv\_bias=True, qk\_scale=None, attn\_drop=0., proj\_drop=0.):
  
  
 super().\_\_init\_\_()
  
 self.dim = dim
  
 self.window\_size = window\_size # Wh, Ww
  
 self.num\_heads = num\_heads
  
 head\_dim = dim // num\_heads
  
 self.scale = qk\_scale or head\_dim \*\* -0.5
  
  
 # define a parameter table of relative position bias
  
 self.relative\_position\_bias\_table = nn.Parameter(
  
 torch.zeros((2 \* window\_size[0] - 1) \* (2 \* window\_size[1] - 1), num\_heads)) # 2\*Wh-1 \* 2\*Ww-1, nH
  
  
 # get pair-wise relative position index for each token inside the window
  
 coords\_h = torch.arange(self.window\_size[0])
  
 coords\_w = torch.arange(self.window\_size[1])
  
 coords = torch.stack(torch.meshgrid([coords\_h, coords\_w])) # 2, Wh, Ww
  
 coords\_flatten = torch.flatten(coords, 1) # 2, Wh\*Ww
  
 relative\_coords = coords\_flatten[:, :, None] - coords\_flatten[:, None, :] # 2, Wh\*Ww, Wh\*Ww
  
 relative\_coords = relative\_coords.permute(1, 2, 0).contiguous() # Wh\*Ww, Wh\*Ww, 2
  
 relative\_coords[:, :, 0] += self.window\_size[0] - 1 # shift to start from 0
  
 relative\_coords[:, :, 1] += self.window\_size[1] - 1
  
 relative\_coords[:, :, 0] \*= 2 \* self.window\_size[1] - 1
  
 relative\_position\_index = relative\_coords.sum(-1) # Wh\*Ww, Wh\*Ww
  
 self.register\_buffer("relative\_position\_index", relative\_position\_index)
  
  
 self.qkv = nn.Linear(dim, dim \* 3, bias=qkv\_bias)
  
 self.attn\_drop = nn.Dropout(attn\_drop)
  
 self.proj = nn.Linear(dim, dim)
  
  
 self.proj\_drop = nn.Dropout(proj\_drop)
  
  
 trunc\_normal\_(self.relative\_position\_bias\_table, std=.02)
  
 self.softmax = nn.Softmax(dim=-1)
  
  
 def forward(self, x, mask=None):
  
 """
  
 Args:
  
 x: input features with shape of (num\_windows\*B, N, C)
  
 mask: (0/-inf) mask with shape of (num\_windows, Wh\*Ww, Wh\*Ww) or None
  
 """
  
 B\_, N, C = x.shape
  
 qkv = self.qkv(x).reshape(B\_, N, 3, self.num\_heads, C // self.num\_heads).permute(2, 0, 3, 1, 4)
  
 q, k, v = qkv[0], qkv[1], qkv[2] # make torchscript happy (cannot use tensor as tuple)
  
  
 q = q \* self.scale
  
 attn = (q @ k.transpose(-2, -1))
  
  
 relative\_position\_bias = self.relative\_position\_bias\_table[self.relative\_position\_index.view(-1)].view(
  
 self.window\_size[0] \* self.window\_size[1], self.window\_size[0] \* self.window\_size[1], -1) # Wh\*Ww,Wh\*Ww,nH
  
 relative\_position\_bias = relative\_position\_bias.permute(2, 0, 1).contiguous() # nH, Wh\*Ww, Wh\*Ww
  
 attn = attn + relative\_position\_bias.unsqueeze(0)
  
  
 if mask is not None:
  
 nW = mask.shape[0]
  
 attn = attn.view(B\_ // nW, nW, self.num\_heads, N, N) + mask.unsqueeze(1).unsqueeze(0)
  
 attn = attn.view(-1, self.num\_heads, N, N)
  
 attn = self.softmax(attn)
  
 else:
  
 attn = self.softmax(attn)
  
  
 attn = self.attn\_drop(attn)
  
  
 x = (attn @ v).transpose(1, 2).reshape(B\_, N, C)
  
 x = self.proj(x)
  
 x = self.proj\_drop(x)
  
 return x
  
  
 def extra\_repr(self) -> str:
  
 return f'dim={self.dim}, window\_size={self.window\_size}, num\_heads={self.num\_heads}'
  
  
 def flops(self, N):
  
 # calculate flops for 1 window with token length of N
  
 flops = 0
  
 # qkv = self.qkv(x)
  
 flops += N \* self.dim \* 3 \* self.dim
  
 # attn = (q @ k.transpose(-2, -1))
  
 flops += self.num\_heads \* N \* (self.dim // self.num\_heads) \* N
  
 # x = (attn @ v)
  
 flops += self.num\_heads \* N \* N \* (self.dim // self.num\_heads)
  
 # x = self.proj(x)
  
 flops += N \* self.dim \* self.dim
  
 return flops
  
  
  
class SwinTransformerBlock(nn.Module):
  
 def \_\_init\_\_(self, dim, input\_resolution, num\_heads, window\_size=7, shift\_size=0,
  
 mlp\_ratio=4., qkv\_bias=True, qk\_scale=None, drop=0., attn\_drop=0., drop\_path=0.,
  
 act\_layer=nn.GELU, norm\_layer=nn.LayerNorm):
  
 super().\_\_init\_\_()
  
 self.dim = dim
  
 self.input\_resolution = input\_resolution
  
 self.num\_heads = num\_heads
  
 self.window\_size = window\_size
  
 self.shift\_size = shift\_size
  
 self.mlp\_ratio = mlp\_ratio
  
 if min(self.input\_resolution) <= self.window\_size:
  
 # if window size is larger than input resolution, we don't partition windows
  
 self.shift\_size = 0
  
 self.window\_size = min(self.input\_resolution)
  
 assert 0 <= self.shift\_size < self.window\_size, "shift\_size must in 0-window\_size"
  
  
 self.norm1 = norm\_layer(dim)
  
 self.attn = WindowAttention(
  
 dim, window\_size=to\_2tuple(self.window\_size), num\_heads=num\_heads,
  
 qkv\_bias=qkv\_bias, qk\_scale=qk\_scale, attn\_drop=attn\_drop, proj\_drop=drop)
  
  
 self.drop\_path = DropPath(drop\_path) if drop\_path > 0. else nn.Identity()
  
 self.norm2 = norm\_layer(dim)
  
 mlp\_hidden\_dim = int(dim \* mlp\_ratio)
  
 self.mlp = Mlp(in\_features=dim, hidden\_features=mlp\_hidden\_dim, act\_layer=act\_layer, drop=drop)
  
  
 if self.shift\_size > 0:
  
 attn\_mask = self.calculate\_mask(self.input\_resolution)
  
 else:
  
 attn\_mask = None
  
  
 self.register\_buffer("attn\_mask", attn\_mask)
  
  
 def calculate\_mask(self, x\_size):
  
 # calculate attention mask for SW-MSA
  
 H, W = x\_size
  
 img\_mask = torch.zeros((1, H, W, 1)) # 1 H W 1
  
 h\_slices = (slice(0, -self.window\_size),
  
 slice(-self.window\_size, -self.shift\_size),
  
 slice(-self.shift\_size, None))
  
 w\_slices = (slice(0, -self.window\_size),
  
 slice(-self.window\_size, -self.shift\_size),
  
 slice(-self.shift\_size, None))
  
 cnt = 0
  
 for h in h\_slices:
  
 for w in w\_slices:
  
 img\_mask[:, h, w, :] = cnt
  
 cnt += 1
  
  
 mask\_windows = window\_partition(img\_mask, self.window\_size) # nW, window\_size, window\_size, 1
  
 mask\_windows = mask\_windows.view(-1, self.window\_size \* self.window\_size)
  
 attn\_mask = mask\_windows.unsqueeze(1) - mask\_windows.unsqueeze(2)
  
 attn\_mask = attn\_mask.masked\_fill(attn\_mask != 0, float(-100.0)).masked\_fill(attn\_mask == 0, float(0.0))
  
  
 return attn\_mask
  
  
 def forward(self, x, x\_size):
  
 H, W = x\_size
  
 B, L, C = x.shape
  
 # assert L == H \* W, "input feature has wrong size"
  
  
 shortcut = x
  
 x = self.norm1(x)
  
 x = x.view(B, H, W, C)
  
  
 # cyclic shift
  
 if self.shift\_size > 0:
  
 shifted\_x = torch.roll(x, shifts=(-self.shift\_size, -self.shift\_size), dims=(1, 2))
  
 else:
  
 shifted\_x = x
  
  
 # partition windows
  
 x\_windows = window\_partition(shifted\_x, self.window\_size) # nW\*B, window\_size, window\_size, C
  
 x\_windows = x\_windows.view(-1, self.window\_size \* self.window\_size, C) # nW\*B, window\_size\*window\_size, C
  
  
 # W-MSA/SW-MSA (to be compatible for testing on images whose shapes are the multiple of window size
  
 if self.input\_resolution == x\_size:
  
 attn\_windows = self.attn(x\_windows, mask=self.attn\_mask) # nW\*B, window\_size\*window\_size, C
  
 else:
  
 attn\_windows = self.attn(x\_windows, mask=self.calculate\_mask(x\_size).to(x.device))
  
  
 # merge windows
  
 attn\_windows = attn\_windows.view(-1, self.window\_size, self.window\_size, C)
  
 shifted\_x = window\_reverse(attn\_windows, self.window\_size, H, W) # B H' W' C
  
  
 # reverse cyclic shift
  
 if self.shift\_size > 0:
  
 x = torch.roll(shifted\_x, shifts=(self.shift\_size, self.shift\_size), dims=(1, 2))
  
 else:
  
 x = shifted\_x
  
 x = x.view(B, H \* W, C)
  
  
 # FFN
  
 x = shortcut + self.drop\_path(x)
  
 x = x + self.drop\_path(self.mlp(self.norm2(x)))
  
  
 return x
  
  
 def extra\_repr(self) -> str:
  
 return f"dim={self.dim}, input\_resolution={self.input\_resolution}, num\_heads={self.num\_heads}, "
  
 f"window\_size={self.window\_size}, shift\_size={self.shift\_size}, mlp\_ratio={self.mlp\_ratio}"
  
  
 def flops(self):
  
 flops = 0
  
 H, W = self.input\_resolution
  
 # norm1
  
 flops += self.dim \* H \* W
  
 # W-MSA/SW-MSA
  
 nW = H \* W / self.window\_size / self.window\_size
  
 flops += nW \* self.attn.flops(self.window\_size \* self.window\_size)
  
 # mlp
  
 flops += 2 \* H \* W \* self.dim \* self.dim \* self.mlp\_ratio
  
 # norm2
  
 flops += self.dim \* H \* W
  
 return flops
  
  
  
class PatchMerging(nn.Module):
  
 def \_\_init\_\_(self, input\_resolution, dim, norm\_layer=nn.LayerNorm):
  
 super().\_\_init\_\_()
  
 self.input\_resolution = input\_resolution
  
 self.dim = dim
  
 self.reduction = nn.Linear(4 \* dim, 2 \* dim, bias=False)
  
 self.norm = norm\_layer(4 \* dim)
  
  
 def forward(self, x):
  
 """
  
 x: B, H\*W, C
  
 """
  
 H, W = self.input\_resolution
  
 B, L, C = x.shape
  
 assert L == H \* W, "input feature has wrong size"
  
 assert H % 2 == 0 and W % 2 == 0, f"x size ({H}\*{W}) are not even."
  
  
 x = x.view(B, H, W, C)
  
  
 x0 = x[:, 0::2, 0::2, :] # B H/2 W/2 C
  
 x1 = x[:, 1::2, 0::2, :] # B H/2 W/2 C
  
 x2 = x[:, 0::2, 1::2, :] # B H/2 W/2 C
  
 x3 = x[:, 1::2, 1::2, :] # B H/2 W/2 C
  
 x = torch.cat([x0, x1, x2, x3], -1) # B H/2 W/2 4\*C
  
 x = x.view(B, -1, 4 \* C) # B H/2\*W/2 4\*C
  
  
 x = self.norm(x)
  
 x = self.reduction(x)
  
  
 return x
  
  
 def extra\_repr(self) -> str:
  
 return f"input\_resolution={self.input\_resolution}, dim={self.dim}"
  
  
 def flops(self):
  
 H, W = self.input\_resolution
  
 flops = H \* W \* self.dim
  
 flops += (H // 2) \* (W // 2) \* 4 \* self.dim \* 2 \* self.dim
  
 return flops
  
  
  
class BasicLayer(nn.Module):
  
 def \_\_init\_\_(self, dim, input\_resolution, depth, num\_heads, window\_size,
  
 mlp\_ratio=4., qkv\_bias=True, qk\_scale=None, drop=0., attn\_drop=0.,
  
 drop\_path=0., norm\_layer=nn.LayerNorm, downsample=None, use\_checkpoint=False):
  
  
 super().\_\_init\_\_()
  
 self.dim = dim
  
 self.input\_resolution = input\_resolution
  
 self.depth = depth
  
 self.use\_checkpoint = use\_checkpoint
  
  
 # build blocks
  
 self.blocks = nn.ModuleList([
  
 SwinTransformerBlock(dim=dim, input\_resolution=input\_resolution,
  
 num\_heads=num\_heads, window\_size=window\_size,
  
 shift\_size=0 if (i % 2 == 0) else window\_size // 2,
  
 mlp\_ratio=mlp\_ratio,
  
 qkv\_bias=qkv\_bias, qk\_scale=qk\_scale,
  
 drop=drop, attn\_drop=attn\_drop,
  
 drop\_path=drop\_path[i] if isinstance(drop\_path, list) else drop\_path,
  
 norm\_layer=norm\_layer)
  
 for i in range(depth)])
  
  
 # patch merging layer
  
 if downsample is not None:
  
 self.downsample = downsample(input\_resolution, dim=dim, norm\_layer=norm\_layer)
  
 else:
  
 self.downsample = None
  
  
 def forward(self, x, x\_size):
  
 for blk in self.blocks:
  
 if self.use\_checkpoint:
  
 x = checkpoint.checkpoint(blk, x, x\_size)
  
 else:
  
 x = blk(x, x\_size)
  
 if self.downsample is not None:
  
 x = self.downsample(x)
  
 return x
  
  
 def extra\_repr(self) -> str:
  
 return f"dim={self.dim}, input\_resolution={self.input\_resolution}, depth={self.depth}"
  
  
 def flops(self):
  
 flops = 0
  
 for blk in self.blocks:
  
 flops += blk.flops()
  
 if self.downsample is not None:
  
 flops += self.downsample.flops()
  
 return flops
  
  
class RSTB(nn.Module):
  
 def \_\_init\_\_(self, dim, input\_resolution, depth, num\_heads, window\_size,
  
 mlp\_ratio=4., qkv\_bias=True, qk\_scale=None, drop=0., attn\_drop=0.,
  
 drop\_path=0., norm\_layer=nn.LayerNorm, downsample=None, use\_checkpoint=False,
  
 img\_size=224, patch\_size=4, resi\_connection='1conv'):
  
 super(RSTB, self).\_\_init\_\_()
  
 self.dim = dim
  
 self.input\_resolution = input\_resolution
  
 self.residual\_group = BasicLayer(dim=dim,
  
 input\_resolution=input\_resolution,
  
 depth=depth,
  
 num\_heads=num\_heads,
  
 window\_size=window\_size,
  
 mlp\_ratio=mlp\_ratio,
  
 qkv\_bias=qkv\_bias, qk\_scale=qk\_scale,
  
 drop=drop, attn\_drop=attn\_drop,
  
 drop\_path=drop\_path,
  
 norm\_layer=norm\_layer,
  
 downsample=downsample,
  
 use\_checkpoint=use\_checkpoint)
  
  
 if resi\_connection == '1conv':
  
 self.conv = nn.Conv2d(dim, dim, 3, 1, 1)
  
 elif resi\_connection == '3conv':
  
 # to save parameters and memory
  
 self.conv = nn.Sequential(nn.Conv2d(dim, dim // 4, 3, 1, 1), nn.LeakyReLU(negative\_slope=0.2, inplace=True),
  
 nn.Conv2d(dim // 4, dim // 4, 1, 1, 0),
  
 nn.LeakyReLU(negative\_slope=0.2, inplace=True),
  
 nn.Conv2d(dim // 4, dim, 3, 1, 1))
  
  
 self.patch\_embed = PatchEmbed(
  
 img\_size=img\_size, patch\_size=patch\_size, in\_chans=0, embed\_dim=dim,
  
 norm\_layer=None)
  
  
 self.patch\_unembed = PatchUnEmbed(
  
 img\_size=img\_size, patch\_size=patch\_size, in\_chans=0, embed\_dim=dim,
  
 norm\_layer=None)
  
  
 def forward(self, x, x\_size):
  
 # with torch.backends.cudnn.flags(enabled=False):
  
 return self.patch\_embed(self.conv(self.patch\_unembed(self.residual\_group(x, x\_size), x\_size))) + x
  
  
 def flops(self):
  
 flops = 0
  
 flops += self.residual\_group.flops()
  
 H, W = self.input\_resolution
  
 flops += H \* W \* self.dim \* self.dim \* 9
  
 flops += self.patch\_embed.flops()
  
 flops += self.patch\_unembed.flops()
  
  
 return flops
  
  
  
class PatchEmbed(nn.Module):
  
 def \_\_init\_\_(self, img\_size=224, patch\_size=4, in\_chans=3, embed\_dim=96, norm\_layer=None):
  
 super().\_\_init\_\_()
  
 img\_size = to\_2tuple(img\_size)
  
 patch\_size = to\_2tuple(patch\_size)
  
 patches\_resolution = [img\_size[0] // patch\_size[0], img\_size[1] // patch\_size[1]]
  
 self.img\_size = img\_size
  
 self.patch\_size = patch\_size
  
 self.patches\_resolution = patches\_resolution
  
 self.num\_patches = patches\_resolution[0] \* patches\_resolution[1]
  
  
 self.in\_chans = in\_chans
  
 self.embed\_dim = embed\_dim
  
  
 if norm\_layer is not None:
  
 self.norm = norm\_layer(embed\_dim)
  
 else:
  
 self.norm = None
  
  
 def forward(self, x):
  
 x = x.flatten(2).transpose(1, 2) # B Ph\*Pw C
  
 if self.norm is not None:
  
 x = self.norm(x)
  
 return x
  
  
 def flops(self):
  
 flops = 0
  
 H, W = self.img\_size
  
 if self.norm is not None:
  
 flops += H \* W \* self.embed\_dim
  
 return flops
  
  
class PatchUnEmbed(nn.Module):
  
 def \_\_init\_\_(self, img\_size=224, patch\_size=4, in\_chans=3, embed\_dim=96, norm\_layer=None):
  
 super().\_\_init\_\_()
  
 img\_size = to\_2tuple(img\_size)
  
 patch\_size = to\_2tuple(patch\_size)
  
 patches\_resolution = [img\_size[0] // patch\_size[0], img\_size[1] // patch\_size[1]]
  
 self.img\_size = img\_size
  
 self.patch\_size = patch\_size
  
 self.patches\_resolution = patches\_resolution
  
 self.num\_patches = patches\_resolution[0] \* patches\_resolution[1]
  
  
 self.in\_chans = in\_chans
  
 self.embed\_dim = embed\_dim
  
  
 def forward(self, x, x\_size):
  
 B, HW, C = x.shape
  
 x = x.transpose(1, 2).view(B, self.embed\_dim, x\_size[0], x\_size[1]) # B Ph\*Pw C
  
 return x
  
  
 def flops(self):
  
 flops = 0
  
 return flops
  
  
class Upsample(nn.Sequential):
  
 def \_\_init\_\_(self, scale, num\_feat):
  
 m = []
  
 if (scale & (scale - 1)) == 0: # scale = 2^n
  
 for \_ in range(int(math.log(scale, 2))):
  
 m.append(nn.Conv2d(num\_feat, 4 \* num\_feat, 3, 1, 1))
  
 m.append(nn.PixelShuffle(2))
  
 elif scale == 3:
  
 m.append(nn.Conv2d(num\_feat, 9 \* num\_feat, 3, 1, 1))
  
 m.append(nn.PixelShuffle(3))
  
 else:
  
 raise ValueError(f'scale {scale} is not supported. ' 'Supported scales: 2^n and 3.')
  
 super(Upsample, self).\_\_init\_\_(\*m)
  
  
  
class UpsampleOneStep(nn.Sequential):
  
 def \_\_init\_\_(self, scale, num\_feat, num\_out\_ch, input\_resolution=None):
  
 self.num\_feat = num\_feat
  
 self.input\_resolution = input\_resolution
  
 m = []
  
 m.append(nn.Conv2d(num\_feat, (scale \*\* 2) \* num\_out\_ch, 3, 1, 1))
  
 m.append(nn.PixelShuffle(scale))
  
 super(UpsampleOneStep, self).\_\_init\_\_(\*m)
  
  
 def flops(self):
  
 H, W = self.input\_resolution
  
 flops = H \* W \* self.num\_feat \* 3 \* 9
  
 return flops
  
  
  
class SwinIR(nn.Module):
  
 def \_\_init\_\_(self, img\_size=64, patch\_size=1, in\_chans=3,
  
 embed\_dim=96, depths=[6, 6, 6, 6], num\_heads=[6, 6, 6, 6],
  
 window\_size=7, mlp\_ratio=4., qkv\_bias=True, qk\_scale=None,
  
 drop\_rate=0., attn\_drop\_rate=0., drop\_path\_rate=0.1,
  
 norm\_layer=nn.LayerNorm, ape=False, patch\_norm=True,
  
 use\_checkpoint=False, upscale=2, img\_range=1., upsampler='', resi\_connection='1conv',
  
 \*\*kwargs):
  
 super(SwinIR, self).\_\_init\_\_()
  
 num\_in\_ch = in\_chans
  
 num\_out\_ch = in\_chans
  
 num\_feat = 64
  
 self.img\_range = img\_range
  
 if in\_chans == 3:
  
 rgb\_mean = (0.4488, 0.4371, 0.4040)
  
 self.mean = torch.Tensor(rgb\_mean).view(1, 3, 1, 1)
  
 else:
  
 self.mean = torch.zeros(1, 1, 1, 1)
  
 self.upscale = upscale
  
 self.upsampler = upsampler
  
 self.window\_size = window\_size
  
  
 self.conv\_first = nn.Conv2d(num\_in\_ch, embed\_dim, 3, 1, 1)
  
  
 self.num\_layers = len(depths)
  
 self.embed\_dim = embed\_dim
  
 self.ape = ape
  
 self.patch\_norm = patch\_norm
  
 self.num\_features = embed\_dim
  
 self.mlp\_ratio = mlp\_ratio
  
  
 # split image into non-overlapping patches
  
 self.patch\_embed = PatchEmbed(
  
 img\_size=img\_size, patch\_size=patch\_size, in\_chans=embed\_dim, embed\_dim=embed\_dim,
  
 norm\_layer=norm\_layer if self.patch\_norm else None)
  
 num\_patches = self.patch\_embed.num\_patches
  
 patches\_resolution = self.patch\_embed.patches\_resolution
  
 self.patches\_resolution = patches\_resolution
  
  
 # merge non-overlapping patches into image
  
 self.patch\_unembed = PatchUnEmbed(
  
 img\_size=img\_size, patch\_size=patch\_size, in\_chans=embed\_dim, embed\_dim=embed\_dim,
  
 norm\_layer=norm\_layer if self.patch\_norm else None)
  
  
 # absolute position embedding
  
 if self.ape:
  
 self.absolute\_pos\_embed = nn.Parameter(torch.zeros(1, num\_patches, embed\_dim))
  
 trunc\_normal\_(self.absolute\_pos\_embed, std=.02)
  
  
 self.pos\_drop = nn.Dropout(p=drop\_rate)
  
  
 # stochastic depth
  
 dpr = [x.item() for x in torch.linspace(0, drop\_path\_rate, sum(depths))] # stochastic depth decay rule
  
  
 # build Residual Swin Transformer blocks (RSTB)
  
 self.layers = nn.ModuleList()
  
 for i\_layer in range(self.num\_layers):
  
 layer = RSTB(dim=embed\_dim,
  
 input\_resolution=(patches\_resolution[0],
  
 patches\_resolution[1]),
  
 depth=depths[i\_layer],
  
 num\_heads=num\_heads[i\_layer],
  
 window\_size=window\_size,
  
 mlp\_ratio=self.mlp\_ratio,
  
 qkv\_bias=qkv\_bias, qk\_scale=qk\_scale,
  
 drop=drop\_rate, attn\_drop=attn\_drop\_rate,
  
 drop\_path=dpr[sum(depths[:i\_layer]):sum(depths[:i\_layer + 1])], # no impact on SR results
  
 norm\_layer=norm\_layer,
  
 downsample=None,
  
 use\_checkpoint=use\_checkpoint,
  
 img\_size=img\_size,
  
 patch\_size=patch\_size,
  
 resi\_connection=resi\_connection
  
  
 )
  
 self.layers.append(layer)
  
 self.norm = norm\_layer(self.num\_features)
  
  
 # build the last conv layer in deep feature extraction
  
 if resi\_connection == '1conv':
  
 self.conv\_after\_body = nn.Conv2d(embed\_dim, embed\_dim, 3, 1, 1)
  
 elif resi\_connection == '3conv':
  
 # to save parameters and memory
  
 self.conv\_after\_body = nn.Sequential(nn.Conv2d(embed\_dim, embed\_dim // 4, 3, 1, 1),
  
 nn.LeakyReLU(negative\_slope=0.2, inplace=True),
  
 nn.Conv2d(embed\_dim // 4, embed\_dim // 4, 1, 1, 0),
  
 nn.LeakyReLU(negative\_slope=0.2, inplace=True),
  
 nn.Conv2d(embed\_dim // 4, embed\_dim, 3, 1, 1))
  
  
 if self.upsampler == 'pixelshuffle':
  
 # for classical SR
  
 self.conv\_before\_upsample = nn.Sequential(nn.Conv2d(embed\_dim, num\_feat, 3, 1, 1),
  
 nn.LeakyReLU(inplace=True))
  
 self.upsample = Upsample(upscale, num\_feat)
  
 self.conv\_last = nn.Conv2d(num\_feat, num\_out\_ch, 3, 1, 1)
  
 elif self.upsampler == 'pixelshuffledirect':
  
 # for lightweight SR (to save parameters)
  
 self.upsample = UpsampleOneStep(upscale, embed\_dim, num\_out\_ch,
  
 (patches\_resolution[0], patches\_resolution[1]))
  
 elif self.upsampler == 'nearest+conv':
  
 # for real-world SR (less artifacts)
  
 assert self.upscale == 4, 'only support x4 now.'
  
 self.conv\_before\_upsample = nn.Sequential(nn.Conv2d(embed\_dim, num\_feat, 3, 1, 1),
  
 nn.LeakyReLU(inplace=True))
  
 self.conv\_up1 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)
  
 self.conv\_up2 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)
  
 self.conv\_hr = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)
  
 self.conv\_last = nn.Conv2d(num\_feat, num\_out\_ch, 3, 1, 1)
  
 self.lrelu = nn.LeakyReLU(negative\_slope=0.2, inplace=True)
  
 else:
  
 # for image denoising and JPEG compression artifact reduction
  
 self.conv\_last = nn.Conv2d(embed\_dim, num\_out\_ch, 3, 1, 1)
  
  
 self.apply(self.\_init\_weights)
  
  
 def \_init\_weights(self, m):
  
 if isinstance(m, nn.Linear):
  
 trunc\_normal\_(m.weight, std=.02)
  
 if isinstance(m, nn.Linear) and m.bias is not None:
  
 nn.init.constant\_(m.bias, 0)
  
 elif isinstance(m, nn.LayerNorm):
  
 nn.init.constant\_(m.bias, 0)
  
 nn.init.constant\_(m.weight, 1.0)
  
  
 @torch.jit.ignore
  
 def no\_weight\_decay(self):
  
 return {'absolute\_pos\_embed'}
  
  
 @torch.jit.ignore
  
 def no\_weight\_decay\_keywords(self):
  
 return {'relative\_position\_bias\_table'}
  
  
 def check\_image\_size(self, x):
  
 \_, \_, h, w = x.size()
  
 mod\_pad\_h = (self.window\_size - h % self.window\_size) % self.window\_size
  
 mod\_pad\_w = (self.window\_size - w % self.window\_size) % self.window\_size
  
 x = F.pad(x, (0, mod\_pad\_w, 0, mod\_pad\_h), 'reflect')
  
 return x
  
  
 def forward\_features(self, x):
  
 x\_size = (x.shape[2], x.shape[3])
  
 x = self.patch\_embed(x)
  
 if self.ape:
  
 x = x + self.absolute\_pos\_embed
  
 x = self.pos\_drop(x)
  
  
 for layer in self.layers:
  
 x = layer(x, x\_size)
  
  
 x = self.norm(x) # B L C
  
 x = self.patch\_unembed(x, x\_size)
  
  
 return x
  
  
 def forward(self, x):
  
 H, W = x.shape[2:]
  
 x = self.check\_image\_size(x)
  
  
 self.mean = self.mean.type\_as(x)
  
 x = (x - self.mean) \* self.img\_range
  
  
 if self.upsampler == 'pixelshuffle':
  
 # for classical SR
  
 x = self.conv\_first(x)
  
 x = self.conv\_after\_body(self.forward\_features(x)) + x
  
 x = self.conv\_before\_upsample(x)
  
 x = self.conv\_last(self.upsample(x))
  
 elif self.upsampler == 'pixelshuffledirect':
  
 # for lightweight SR
  
 x = self.conv\_first(x)
  
 x = self.conv\_after\_body(self.forward\_features(x)) + x
  
 x = self.upsample(x)
  
 elif self.upsampler == 'nearest+conv':
  
 # for real-world SR
  
 x = self.conv\_first(x)
  
 x = self.conv\_after\_body(self.forward\_features(x)) + x
  
 x = self.conv\_before\_upsample(x)
  
 x = self.lrelu(self.conv\_up1(torch.nn.functional.interpolate(x, scale\_factor=2, mode='nearest')))
  
 x = self.lrelu(self.conv\_up2(torch.nn.functional.interpolate(x, scale\_factor=2, mode='nearest')))
  
 x = self.conv\_last(self.lrelu(self.conv\_hr(x)))
  
 else:
  
 # for image denoising and JPEG compression artifact reduction
  
 x\_first = self.conv\_first(x)
  
 res = self.conv\_after\_body(self.forward\_features(x\_first)) + x\_first
  
 x = x + self.conv\_last(res)
  
  
 x = x / self.img\_range + self.mean
  
  
 return x[:, :, :H\*self.upscale, :W\*self.upscale]
  
  
 def flops(self):
  
 flops = 0
  
 H, W = self.patches\_resolution
  
 flops += H \* W \* 3 \* self.embed\_dim \* 9
  
 flops += self.patch\_embed.flops()
  
 for i, layer in enumerate(self.layers):
  
 flops += layer.flops()
  
 flops += H \* W \* 3 \* self.embed\_dim \* self.embed\_dim
  
 flops += self.upsample.flops()
  
 return flops
  
  
if \_\_name\_\_ == '\_\_main\_\_':
  
 upscale = 4
  
 window\_size = 8
  
 height = (1024 // upscale // window\_size + 1) \* window\_size
  
 width = (720 // upscale // window\_size + 1) \* window\_size
  
 model = SwinIR(upscale=2, img\_size=(height, width),
  
 window\_size=window\_size, img\_range=1., depths=[6, 6, 6, 6],
  
 embed\_dim=60, num\_heads=[6, 6, 6, 6], mlp\_ratio=2, upsampler='pixelshuffledirect')
  
 print(model)
  
 print(height, width, model.flops() / 1e9)
  
  
 x = torch.randn((1, 3, height, width))
  
 x = model(x)
  
 print(x.shape)
  
  
import os.path
  
import torch
  
import torch.nn.functional as F
  
from torch import nn as nn
  
import numpy as np
  
import math
  
from global\_variable import MODEL\_PATH
  
from .dcn import ModulatedDeformConvPack, modulated\_deform\_conv
  
from .network\_swinir import RSTB
  
from .ridcp\_utils import ResBlock, CombineQuantBlock
  
from .vgg\_arch import VGGFeatureExtractor
  
  
WEIGHT\_PATH = os.path.join(MODEL\_PATH, 'RIDCP/weight\_for\_matching\_dehazing\_Flickr.pth')
  
  
  
class DCNv2Pack(ModulatedDeformConvPack):
  
 def forward(self, x, feat):
  
 out = self.conv\_offset(feat)
  
 o1, o2, mask = torch.chunk(out, 3, dim=1)
  
 offset = torch.cat((o1, o2), dim=1)
  
 mask = torch.sigmoid(mask)
  
  
 offset\_absmean = torch.mean(torch.abs(offset))
  
 if offset\_absmean > 50:
  
 print(f'Offset abs mean is {offset\_absmean}, larger than 50.')
  
  
 return modulated\_deform\_conv(x, offset, mask, self.weight, self.bias, self.stride, self.padding,
  
 self.dilation, self.groups, self.deformable\_groups)
  
  
  
class VectorQuantizer(nn.Module):
  
 def \_\_init\_\_(self, n\_e, e\_dim, weight\_path=WEIGHT\_PATH, beta=0.25,
  
 LQ\_stage=False, use\_weight=True, weight\_alpha=1.0):
  
 super().\_\_init\_\_()
  
 self.n\_e = int(n\_e)
  
 self.e\_dim = int(e\_dim)
  
 self.LQ\_stage = LQ\_stage
  
 self.beta = beta
  
 self.use\_weight = use\_weight
  
 self.weight\_alpha = weight\_alpha
  
 if self.use\_weight:
  
 self.weight = nn.Parameter(torch.load(weight\_path))
  
 self.weight.requires\_grad = False
  
 self.embedding = nn.Embedding(self.n\_e, self.e\_dim)
  
  
 def dist(self, x, y):
  
 if x.shape == y.shape:
  
 return (x - y) \*\* 2
  
 else:
  
 return torch.sum(x \*\* 2, dim=1, keepdim=True) +
  
 torch.sum(y \*\* 2, dim=1) - 2 \*
  
 torch.matmul(x, y.t())
  
  
 def gram\_loss(self, x, y):
  
 b, h, w, c = x.shape
  
 x = x.reshape(b, h \* w, c)
  
 y = y.reshape(b, h \* w, c)
  
  
 gmx = x.transpose(1, 2) @ x / (h \* w)
  
 gmy = y.transpose(1, 2) @ y / (h \* w)
  
  
 return (gmx - gmy).square().mean()
  
  
 def forward(self, z, gt\_indices=None, current\_iter=None, weight\_alpha=None):
  
 """
  
 Args:
  
 z: input features to be quantized, z (continuous) -> z\_q (discrete)
  
 z.shape = (batch, channel, height, width)
  
 gt\_indices: feature map of given indices, used for visualization.
  
 """
  
 # reshape z -> (batch, height, width, channel) and flatten
  
 z = z.permute(0, 2, 3, 1).contiguous()
  
 z\_flattened = z.view(-1, self.e\_dim)
  
  
 codebook = self.embedding.weight
  
  
 d = self.dist(z\_flattened, codebook)
  
 if self.use\_weight and self.LQ\_stage:
  
 if weight\_alpha is not None:
  
 self.weight\_alpha = weight\_alpha
  
 d = d \* torch.exp(self.weight\_alpha \* self.weight)
  
  
 # find closest encodings
  
 min\_encoding\_indices = torch.argmin(d, dim=1).unsqueeze(1)
  
 min\_encodings = torch.zeros(min\_encoding\_indices.shape[0], codebook.shape[0]).to(z)
  
 min\_encodings.scatter\_(1, min\_encoding\_indices, 1)
  
  
 if gt\_indices is not None:
  
 gt\_indices = gt\_indices.reshape(-1)
  
  
 gt\_min\_indices = gt\_indices.reshape\_as(min\_encoding\_indices)
  
 gt\_min\_onehot = torch.zeros(gt\_min\_indices.shape[0], codebook.shape[0]).to(z)
  
 gt\_min\_onehot.scatter\_(1, gt\_min\_indices, 1)
  
  
 z\_q\_gt = torch.matmul(gt\_min\_onehot, codebook)
  
 z\_q\_gt = z\_q\_gt.view(z.shape)
  
  
 # get quantized latent vectors
  
 z\_q = torch.matmul(min\_encodings, codebook)
  
 z\_q = z\_q.view(z.shape)
  
  
 e\_latent\_loss = torch.mean((z\_q.detach() - z) \*\* 2)
  
 q\_latent\_loss = torch.mean((z\_q - z.detach()) \*\* 2)
  
  
 if self.LQ\_stage and gt\_indices is not None:
  
 # codebook\_loss = self.dist(z\_q, z\_q\_gt.detach()).mean() \
  
 # + self.beta \* self.dist(z\_q\_gt.detach(), z)
  
 codebook\_loss = self.beta \* self.dist(z\_q\_gt.detach(), z)
  
 texture\_loss = self.gram\_loss(z, z\_q\_gt.detach())
  
 # print("codebook loss:", codebook\_loss.mean(), "\ntexture\_loss: ", texture\_loss.mean())
  
 codebook\_loss = codebook\_loss + texture\_loss
  
 else:
  
 codebook\_loss = q\_latent\_loss + e\_latent\_loss \* self.beta
  
  
 # preserve gradients
  
 z\_q = z + (z\_q - z).detach()
  
  
 # reshape back to match original input shape
  
 z\_q = z\_q.permute(0, 3, 1, 2).contiguous()
  
  
 return z\_q, codebook\_loss, min\_encoding\_indices.reshape(z\_q.shape[0], 1, z\_q.shape[2], z\_q.shape[3])
  
  
 def get\_codebook\_entry(self, indices):
  
 b, \_, h, w = indices.shape
  
  
 indices = indices.flatten().to(self.embedding.weight.device)
  
 min\_encodings = torch.zeros(indices.shape[0], self.n\_e).to(indices)
  
 min\_encodings.scatter\_(1, indices[:, None], 1)
  
  
 # get quantized latent vectors
  
 z\_q = torch.matmul(min\_encodings.float(), self.embedding.weight)
  
 z\_q = z\_q.view(b, h, w, -1).permute(0, 3, 1, 2).contiguous()
  
 return z\_q
  
  
  
class SwinLayers(nn.Module):
  
 def \_\_init\_\_(self, input\_resolution=(32, 32), embed\_dim=256,
  
 blk\_depth=6,
  
 num\_heads=8,
  
 window\_size=8,
  
 \*\*kwargs):
  
 super().\_\_init\_\_()
  
 self.swin\_blks = nn.ModuleList()
  
 for i in range(4):
  
 layer = RSTB(embed\_dim, input\_resolution, blk\_depth, num\_heads, window\_size, patch\_size=1, \*\*kwargs)
  
 self.swin\_blks.append(layer)
  
  
 def forward(self, x):
  
 b, c, h, w = x.shape
  
 x = x.reshape(b, c, h \* w).transpose(1, 2)
  
 for m in self.swin\_blks:
  
 x = m(x, (h, w))
  
 x = x.transpose(1, 2).reshape(b, c, h, w)
  
 return x
  
  
  
class MultiScaleEncoder(nn.Module):
  
 def \_\_init\_\_(self,
  
 in\_channel,
  
 max\_depth,
  
 input\_res=256,
  
 channel\_query\_dict=None,
  
 norm\_type='gn',
  
 act\_type='leakyrelu',
  
 LQ\_stage=True,
  
 \*\*swin\_opts,
  
 ):
  
 super().\_\_init\_\_()
  
 self.LQ\_stage = LQ\_stage
  
 ksz = 3
  
  
 self.in\_conv = nn.Conv2d(in\_channel, channel\_query\_dict[input\_res], 4, padding=1)
  
  
 self.blocks = nn.ModuleList()
  
 self.up\_blocks = nn.ModuleList()
  
 self.max\_depth = max\_depth
  
 res = input\_res
  
 for i in range(max\_depth):
  
 in\_ch, out\_ch = channel\_query\_dict[res], channel\_query\_dict[res // 2]
  
 tmp\_down\_block = [
  
 nn.Conv2d(in\_ch, out\_ch, ksz, stride=2, padding=1),
  
 ResBlock(out\_ch, out\_ch, norm\_type, act\_type),
  
 ResBlock(out\_ch, out\_ch, norm\_type, act\_type),
  
 ]
  
 self.blocks.append(nn.Sequential(\*tmp\_down\_block))
  
 res = res // 2
  
  
 if LQ\_stage:
  
 self.blocks.append(SwinLayers(\*\*swin\_opts))
  
  
 def forward(self, input):
  
 # input.requires\_grad = True
  
 x = self.in\_conv(input)
  
 for idx, m in enumerate(self.blocks):
  
 with torch.backends.cudnn.flags(enabled=False):
  
 x = m(x)
  
 return x
  
  
class DecoderBlock(nn.Module):
  
  
 def \_\_init\_\_(self, in\_channel, out\_channel, norm\_type='gn', act\_type='leakyrelu'):
  
 super().\_\_init\_\_()
  
  
 self.block = []
  
 self.block += [
  
 nn.Upsample(scale\_factor=2),
  
 nn.Conv2d(in\_channel, out\_channel, 3, stride=1, padding=1),
  
 ResBlock(out\_channel, out\_channel, norm\_type, act\_type),
  
 ResBlock(out\_channel, out\_channel, norm\_type, act\_type),
  
 ]
  
  
 self.block = nn.Sequential(\*self.block)
  
  
 def forward(self, input):
  
 return self.block(input)
  
  
  
class WarpBlock(nn.Module):
  
 def \_\_init\_\_(self, in\_channel):
  
 super().\_\_init\_\_()
  
 self.offset = nn.Conv2d(in\_channel \* 2, in\_channel, 3, stride=1, padding=1)
  
 self.dcn = DCNv2Pack(in\_channel, in\_channel, 3, padding=1, deformable\_groups=4)
  
  
 def forward(self, x\_vq, x\_residual):
  
 x\_residual = self.offset(torch.cat([x\_vq, x\_residual], dim=1))
  
 feat\_after\_warp = self.dcn(x\_vq, x\_residual)
  
  
 return feat\_after\_warp
  
  
  
class MultiScaleDecoder(nn.Module):
  
 def \_\_init\_\_(self,
  
 in\_channel,
  
 max\_depth,
  
 input\_res=256,
  
 channel\_query\_dict=None,
  
 norm\_type='gn',
  
 act\_type='leakyrelu',
  
 only\_residual=False,
  
 use\_warp=True
  
 ):
  
 super().\_\_init\_\_()
  
 self.only\_residual = only\_residual
  
 self.use\_warp = use\_warp
  
 self.upsampler = nn.ModuleList()
  
 self.warp = nn.ModuleList()
  
 res = input\_res // (2 \*\* max\_depth)
  
 for i in range(max\_depth):
  
 in\_channel, out\_channel = channel\_query\_dict[res], channel\_query\_dict[res \* 2]
  
 self.upsampler.append(nn.Sequential(
  
 nn.Upsample(scale\_factor=2),
  
 nn.Conv2d(in\_channel, out\_channel, 3, stride=1, padding=1),
  
 ResBlock(out\_channel, out\_channel, norm\_type, act\_type),
  
 ResBlock(out\_channel, out\_channel, norm\_type, act\_type),
  
 )
  
 )
  
 self.warp.append(WarpBlock(out\_channel))
  
 res = res \* 2
  
  
 def forward(self, input, code\_decoder\_output):
  
 x = input
  
 for idx, m in enumerate(self.upsampler):
  
 with torch.backends.cudnn.flags(enabled=False):
  
 if not self.only\_residual:
  
 x = m(x)
  
 if self.use\_warp:
  
 x\_vq = self.warp[idx](code\_decoder\_output[idx], x)
  
 # print(idx, x.mean(), x\_vq.mean())
  
 x = x + x\_vq \* (x.mean() / x\_vq.mean())
  
 else:
  
 x = x + code\_decoder\_output[idx]
  
 else:
  
 x = m(x)
  
 # print()
  
 return x
  
  
  
class VQWeightDehazeNet(nn.Module):
  
 def \_\_init\_\_(self,
  
 \*,
  
 in\_channel=3,
  
 codebook\_params=None,
  
 gt\_resolution=256,
  
 LQ\_stage=False,
  
 norm\_type='gn',
  
 act\_type='silu',
  
 use\_quantize=True,
  
 use\_semantic\_loss=False,
  
 use\_residual=True,
  
 only\_residual=False,
  
 use\_weight=False,
  
 use\_warp=True,
  
 weight\_alpha=1.0,
  
 \*\*ignore\_kwargs):
  
 super().\_\_init\_\_()
  
  
 codebook\_params = np.array(codebook\_params)
  
  
 self.codebook\_scale = codebook\_params[:, 0]
  
 codebook\_emb\_num = codebook\_params[:, 1].astype(int)
  
 codebook\_emb\_dim = codebook\_params[:, 2].astype(int)
  
  
 self.use\_quantize = use\_quantize
  
 self.in\_channel = in\_channel
  
 self.gt\_res = gt\_resolution
  
 self.LQ\_stage = LQ\_stage
  
 self.use\_residual = use\_residual
  
 self.only\_residual = only\_residual
  
 self.use\_weight = use\_weight
  
 self.use\_warp = use\_warp
  
 self.weight\_alpha = weight\_alpha
  
  
 channel\_query\_dict = {
  
 8: 256,
  
 16: 256,
  
 32: 256,
  
 64: 256,
  
 128: 128,
  
 256: 64,
  
 512: 32,
  
 }
  
  
 # build encoder
  
 self.max\_depth = int(np.log2(gt\_resolution // self.codebook\_scale[0]))
  
 self.multiscale\_encoder = MultiScaleEncoder(
  
 in\_channel,
  
 self.max\_depth,
  
 self.gt\_res,
  
 channel\_query\_dict,
  
 norm\_type, act\_type, LQ\_stage
  
 )
  
 if self.LQ\_stage and self.use\_residual:
  
 self.multiscale\_decoder = MultiScaleDecoder(
  
 in\_channel,
  
 self.max\_depth,
  
 self.gt\_res,
  
 channel\_query\_dict,
  
 norm\_type, act\_type, only\_residual, use\_warp=self.use\_warp
  
 )
  
  
 # build decoder
  
 self.decoder\_group = nn.ModuleList()
  
 for i in range(self.max\_depth):
  
 res = gt\_resolution // 2 \*\* self.max\_depth \* 2 \*\* i
  
 in\_ch, out\_ch = channel\_query\_dict[res], channel\_query\_dict[res \* 2]
  
 self.decoder\_group.append(DecoderBlock(in\_ch, out\_ch, norm\_type, act\_type))
  
  
 self.out\_conv = nn.Conv2d(out\_ch, 3, 3, 1, 1)
  
 self.residual\_conv = nn.Conv2d(out\_ch, 3, 3, 1, 1)
  
  
 # build multi-scale vector quantizers
  
 self.quantize\_group = nn.ModuleList()
  
 self.before\_quant\_group = nn.ModuleList()
  
 self.after\_quant\_group = nn.ModuleList()
  
  
 for scale in range(0, codebook\_params.shape[0]):
  
 quantize = VectorQuantizer(
  
 codebook\_emb\_num[scale],
  
 codebook\_emb\_dim[scale],
  
 LQ\_stage=self.LQ\_stage,
  
 use\_weight=self.use\_weight,
  
 weight\_alpha=self.weight\_alpha
  
 )
  
 self.quantize\_group.append(quantize)
  
  
 scale\_in\_ch = channel\_query\_dict[self.codebook\_scale[scale]]
  
 if scale == 0:
  
 quant\_conv\_in\_ch = scale\_in\_ch
  
 comb\_quant\_in\_ch1 = codebook\_emb\_dim[scale]
  
 comb\_quant\_in\_ch2 = 0
  
 else:
  
 quant\_conv\_in\_ch = scale\_in\_ch \* 2
  
 comb\_quant\_in\_ch1 = codebook\_emb\_dim[scale - 1]
  
 comb\_quant\_in\_ch2 = codebook\_emb\_dim[scale]
  
  
 self.before\_quant\_group.append(nn.Conv2d(quant\_conv\_in\_ch, codebook\_emb\_dim[scale], 1))
  
 self.after\_quant\_group.append(CombineQuantBlock(comb\_quant\_in\_ch1, comb\_quant\_in\_ch2, scale\_in\_ch))
  
  
 # semantic loss for HQ pretrain stage
  
 self.use\_semantic\_loss = use\_semantic\_loss
  
 if use\_semantic\_loss:
  
 self.conv\_semantic = nn.Sequential(
  
 nn.Conv2d(512, 512, 1, 1, 0),
  
 nn.ReLU(),
  
 )
  
 self.vgg\_feat\_layer = 'relu4\_4'
  
 self.vgg\_feat\_extractor = VGGFeatureExtractor([self.vgg\_feat\_layer])
  
  
 def encode\_and\_decode(self, input, gt\_indices=None, current\_iter=None, weight\_alpha=None):
  
 # if self.training:
  
 # for p in self.multiscale\_encoder.parameters():
  
 # p.requires\_grad = True
  
 enc\_feats = self.multiscale\_encoder(input)
  
  
 if self.use\_semantic\_loss:
  
 with torch.no\_grad():
  
 vgg\_feat = self.vgg\_feat\_extractor(input)[self.vgg\_feat\_layer]
  
  
 codebook\_loss\_list = []
  
 indices\_list = []
  
 semantic\_loss\_list = []
  
 code\_decoder\_output = []
  
  
 quant\_idx = 0
  
 prev\_dec\_feat = None
  
 prev\_quant\_feat = None
  
 out\_img = None
  
 out\_img\_residual = None
  
 x = enc\_feats
  
 for i in range(self.max\_depth):
  
 cur\_res = self.gt\_res // 2 \*\* self.max\_depth \* 2 \*\* i
  
 if cur\_res in self.codebook\_scale: # needs to perform quantize
  
 if prev\_dec\_feat is not None:
  
 before\_quant\_feat = torch.cat((x, prev\_dec\_feat), dim=1)
  
 else:
  
 before\_quant\_feat = x
  
 feat\_to\_quant = self.before\_quant\_group[quant\_idx](before\_quant\_feat)
  
  
 if weight\_alpha is not None:
  
 self.weight\_alpha = weight\_alpha
  
 if gt\_indices is not None:
  
 z\_quant, codebook\_loss, indices = self.quantize\_group[quant\_idx](feat\_to\_quant,
  
 gt\_indices[quant\_idx],
  
 weight\_alpha=self.weight\_alpha)
  
 else:
  
 z\_quant, codebook\_loss, indices = self.quantize\_group[quant\_idx](feat\_to\_quant,
  
 weight\_alpha=self.weight\_alpha)
  
 if self.use\_semantic\_loss:
  
 semantic\_z\_quant = self.conv\_semantic(z\_quant)
  
 semantic\_loss = F.mse\_loss(semantic\_z\_quant, vgg\_feat)
  
 semantic\_loss\_list.append(semantic\_loss)
  
 if not self.use\_quantize:
  
 z\_quant = feat\_to\_quant
  
 after\_quant\_feat = self.after\_quant\_group[quant\_idx](z\_quant, prev\_quant\_feat)
  
 codebook\_loss\_list.append(codebook\_loss)
  
 indices\_list.append(indices)
  
 quant\_idx += 1
  
 prev\_quant\_feat = z\_quant
  
 x = after\_quant\_feat
  
 x = self.decoder\_group[i](x)
  
 code\_decoder\_output.append(x)
  
 prev\_dec\_feat = x
  
 out\_img = self.out\_conv(x)
  
 if self.LQ\_stage and self.use\_residual:
  
 if self.only\_residual:
  
 residual\_feature = self.multiscale\_decoder(enc\_feats, code\_decoder\_output)
  
 else:
  
 residual\_feature = self.multiscale\_decoder(enc\_feats.detach(), code\_decoder\_output)
  
 out\_img\_residual = self.residual\_conv(residual\_feature)
  
 if len(codebook\_loss\_list) > 0:
  
 codebook\_loss = sum(codebook\_loss\_list)
  
 else:
  
 codebook\_loss = 0
  
 semantic\_loss = sum(semantic\_loss\_list) if len(semantic\_loss\_list) else codebook\_loss \* 0
  
 return out\_img, out\_img\_residual, codebook\_loss, semantic\_loss, feat\_to\_quant, z\_quant, indices\_list
  
  
 def decode\_indices(self, indices):
  
 assert len(indices.shape) == 4, f'shape of indices must be (b, 1, h, w), but got {indices.shape}'
  
 z\_quant = self.quantize\_group[0].get\_codebook\_entry(indices)
  
 x = self.after\_quant\_group[0](z\_quant)
  
 for m in self.decoder\_group:
  
 x = m(x)
  
 out\_img = self.out\_conv(x)
  
 return out\_img
  
  
 @torch.no\_grad()
  
 def test\_tile(self, input, tile\_size=240, tile\_pad=16):
  
 batch, channel, height, width = input.shape
  
 output\_height = height
  
 output\_width = width
  
 output\_shape = (batch, channel, output\_height, output\_width)
  
 # start with black image
  
 output = input.new\_zeros(output\_shape)
  
 tiles\_x = math.ceil(width / tile\_size)
  
 tiles\_y = math.ceil(height / tile\_size)
  
 # loop over all tiles
  
 for y in range(tiles\_y):
  
 for x in range(tiles\_x):
  
 # extract tile from input image
  
 ofs\_x = x \* tile\_size
  
 ofs\_y = y \* tile\_size
  
 # input tile area on total image
  
 input\_start\_x = ofs\_x
  
 input\_end\_x = min(ofs\_x + tile\_size, width)
  
 input\_start\_y = ofs\_y
  
 input\_end\_y = min(ofs\_y + tile\_size, height)
  
 # input tile area on total image with padding
  
 input\_start\_x\_pad = max(input\_start\_x - tile\_pad, 0)
  
 input\_end\_x\_pad = min(input\_end\_x + tile\_pad, width)
  
 input\_start\_y\_pad = max(input\_start\_y - tile\_pad, 0)
  
 input\_end\_y\_pad = min(input\_end\_y + tile\_pad, height)
  
 # input tile dimensions
  
 input\_tile\_width = input\_end\_x - input\_start\_x
  
 input\_tile\_height = input\_end\_y - input\_start\_y
  
 tile\_idx = y \* tiles\_x + x + 1
  
 input\_tile = input[:, :, input\_start\_y\_pad:input\_end\_y\_pad, input\_start\_x\_pad:input\_end\_x\_pad]
  
 # upscale tile
  
 output\_tile = self.test(input\_tile)
  
 # output tile area on total image
  
 output\_start\_x = input\_start\_x
  
 output\_end\_x = input\_end\_x
  
 output\_start\_y = input\_start\_y
  
 output\_end\_y = input\_end\_y
  
 # output tile area without padding
  
 output\_start\_x\_tile = (input\_start\_x - input\_start\_x\_pad)
  
 output\_end\_x\_tile = output\_start\_x\_tile + input\_tile\_width
  
 output\_start\_y\_tile = (input\_start\_y - input\_start\_y\_pad)
  
 output\_end\_y\_tile = output\_start\_y\_tile + input\_tile\_height
  
 # put tile into output image
  
 output[:, :, output\_start\_y:output\_end\_y,
  
 output\_start\_x:output\_end\_x] = output\_tile[:, :, output\_start\_y\_tile:output\_end\_y\_tile,
  
 output\_start\_x\_tile:output\_end\_x\_tile]
  
 return output
  
  
 @torch.no\_grad()
  
 def test(self, input, weight\_alpha=None):
  
 org\_use\_semantic\_loss = self.use\_semantic\_loss
  
 self.use\_semantic\_loss = False
  
  
 # padding to multiple of window\_size \* 8
  
 wsz = 32
  
 \_, \_, h\_old, w\_old = input.shape
  
 h\_pad = (h\_old // wsz + 1) \* wsz - h\_old
  
 w\_pad = (w\_old // wsz + 1) \* wsz - w\_old
  
 input = torch.cat([input, torch.flip(input, [2])], 2)[:, :, :h\_old + h\_pad, :]
  
 input = torch.cat([input, torch.flip(input, [3])], 3)[:, :, :, :w\_old + w\_pad]
  
 output\_vq, output, \_, \_, \_, after\_quant, index = self.encode\_and\_decode(input, None, None,
  
 weight\_alpha=weight\_alpha)
  
 if output is not None:
  
 output = output[..., :h\_old, :w\_old]
  
 if output\_vq is not None:
  
 output\_vq = output\_vq[..., :h\_old, :w\_old]
  
 self.use\_semantic\_loss = org\_use\_semantic\_loss
  
 return output, index
  
  
 def forward(self, input, gt\_indices=None, weight\_alpha=None):
  
 if gt\_indices is not None:
  
 # in LQ training stage, need to pass GT indices for supervise.
  
 dec, dec\_residual, codebook\_loss, semantic\_loss, quant\_before\_feature, quant\_after\_feature, indices = self.encode\_and\_decode(
  
 input, gt\_indices, weight\_alpha=weight\_alpha)
  
 else:
  
 # in HQ stage, or LQ test stage, no GT indices needed.
  
 dec, dec\_residual, codebook\_loss, semantic\_loss, quant\_before\_feature, quant\_after\_feature, indices = self.encode\_and\_decode(
  
 input, weight\_alpha=weight\_alpha)
  
 return dec, dec\_residual, codebook\_loss, semantic\_loss, quant\_before\_feature, quant\_after\_feature, indices