Skill-13

2100030910

Sec-23

Main.py

```
import numpy as np
import pandas as pd
import os

from keras import Model
from keras.src.applications import ResNet50
from keras.src.layers import GlobalAveragePooling2D
from keras.src.optimizers import Adam

import model8 as mc
import tensorflow as tf
import cv2

train_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\train'
valid_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\valid'

cloud_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\data\cloudy'
cloud_train_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\train\cloudy'
cloud_valid_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\valid\cloudy'

water_directory = r'C:\Users\dell\PycharmProjex\deepLearn\Skill\data\water'
water_train_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\train\water'
water_valid_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\valid\water'

green_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\data\green_area'
green_train_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\train\green'
green_valid_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\valid\green'

desert_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\data\desert'
desert_train_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\train\desert'
desert_valid_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\valid\desert'
```

```
cloud_image_files = [f for f in os.listdir(cloud_directory) if
                     f.lower().endswith(('.jpg', '.jpeg'))]


water_image_files = [f for f in os.listdir(water_directory) if
                     f.lower().endswith(('.jpg', '.jpeg'))]


desert_image_files = [f for f in os.listdir(desert_directory) if
                      f.lower().endswith(('.jpg', '.jpeg'))]


green_image_files = [f for f in os.listdir(green_directory) if
                     f.lower().endswith(('.jpg', '.jpeg'))]



os.makedirs(train_directory, exist_ok=True)
os.makedirs(valid_directory, exist_ok=True)


import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
BatchNormalization, Dropout

# Directories
train_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\train'
valid_directory =
r'C:\Users\dell\PycharmProjex\deepLearn\Skill\genData\valid'

# Image data preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')



train_generator = train_datagen.flow_from_directory(
```

```
    train_directory,
    batch_size=128,
    target_size=(224, 224),
    class_mode='input',
    shuffle=False
)

validation_generator = validation_datagen.flow_from_directory(
    valid_directory,
    batch_size=128,
    target_size=(224, 224),
    class_mode='input',
    shuffle=False
)

validation_generator2 = validation_datagen.flow_from_directory(
    valid_directory,
    batch_size=128,
    target_size=(224, 224),
    class_mode='categorical',
    shuffle=False
)

all_images = []
all_labels=[]
total_samples =len(validation_generator)
for i in range(total_samples):
    images, _ = next(validation_generator)
    all_images.extend(images)

train_images = np.array(all_images)
print(validation_generator2.classes.shape)
print(train_images.shape)
print(validation_generator2.num_classes)
print(validation_generator.classes)
target=np.array(validation_generator2.classes)
onehot=tf.keras.utils.to_categorical(target,num_classes=4)
print(onehot.shape)

# Sample CNN model
mc2=mc.Modelsc()
model2=mc2.Autoencoder()

# Display model summary
model2.summary()
model2.fit(train_generator, epochs=2, validation_data=validation_generator)




train_EncoImages = model2.predict(train_images)



print(train_EncoImages.shape)


generated_images = model2.predict(validation_generator,verbose=2)
print(generated_images.shape)

plt.figure(figsize=(20, 4))
```

```
generated_images_resize = tf.image.resize(generated_images, (224, 224))
generated_images_resized=generated_images_resize.numpy()

print(generated_images_resized.shape)
for i in range(len(generated_images_resized)):
    image = generated_images_resized[i]
    image=image*255
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    filename
=f":\\Users\\dell\\PycharmProjex\\deepLearn\\Skill\\genData3\\Img_{i}.jpg"
    cv2.imwrite(filename, image)

# Define ResNet50 base model
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

# Freeze the base model

# Add custom classification head
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation='relu')(x)
predictions = Dense(4, activation='softmax')(x)

# Combine base model and custom head
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
  layer.trainable = False

# Compile the model
model.compile(optimizer=Adam(), loss="categorical_crossentropy",
metrics=['accuracy'])




history = model.fit(train_EncoImages,onehot,batch_size=64, epochs=2)

result = model.evaluate(validation_generator2)

print(result)
```

```
import tensorflow as tf
from keras import Sequential, Input, Model
from keras.src import losses
from keras.src.applications import VGG16
from keras.src.layers import Flatten, Dense, Conv2D, MaxPool2D, Dropout,
SimpleRNN, Reshape, LSTM, MaxPooling2D, \
    UpSampling2D
from tensorflow import keras
from tensorflow.keras import layers
from keras.applications.vgg16 import VGG16, preprocess_input
class Modelsc:
    def Autoencoder(self):
        input = Input(shape=(224, 224, 3))
        x = Conv2D(32, (3, 3), activation='relu', padding='same')(input)
        x = MaxPooling2D((2, 2), padding='same')(x)
        x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
        encoded = MaxPooling2D((2, 2), padding='same')(x)
```

```
        x = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded)
        x = UpSampling2D((2, 2))(x)
        x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
        x = UpSampling2D((2, 2))(x)
        decoded = Conv2D(3, kernel_size=(3, 3), activation='sigmoid',
padding='same')(x)

        autoencoder = Model(input, decoded)
        autoencoder.compile(optimizer='adam', loss='mse',
metrics=['accuracy'])

        return autoencoder
```