Code

train.py

**import** numpy **as** np  
**import** pandas **as** pd  
  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sn  
**from** matplotlib.image **import** imread  
**import** cv2  
**import** random  
**from** os **import** listdir  
**from** PIL **import** Image  
  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.preprocessing **import** LabelBinarizer  
**from** sklearn.metrics **import** classification\_report  
  
**from** tensorflow.keras.preprocessing **import** image  
**from** tensorflow.keras.preprocessing.image **import** img\_to\_array,array\_to\_img  
**from** tensorflow.keras.optimizers **import** Adam  
**from** tensorflow.keras **import** Sequential  
**from** tensorflow.keras.layers **import** BatchNormalization  
**from** tensorflow.keras.layers **import** Conv2D, MaxPooling2D, Activation,Flatten, Dense, Dropout, LeakyReLU  
**from** tensorflow.math **import** confusion\_matrix  
**from** tensorflow.keras **import** callbacks  
**from** tensorflow.keras.models **import** load\_model  
  
  
*#show the content of mountain folder  
#data = listdir('./dataset/mountain/')  
#print(data)  
  
  
#show the items in the dataset and their total***import** os  
**import** glob **as** gb  
data\_all = **'./dataset/'  
for** myfolder **in** os.listdir(data\_all):  
 files=gb.glob(pathname=str(data\_all+myfolder+**'/\*.jpg'**))  
 print(**f"The training data found in {**myfolder**} is {**len(files)**}"**)  
   
  
*#Handling data preprocessing  
#create empty list*mydir= **'./dataset/'**rt\_dir=listdir(mydir)  
*#image\_list, label\_list=[], []*image\_list = []  
label\_list = []  
print(rt\_dir)  
  
*#convert image to array  
#Don't be in a haste, wait for some seconds to see the output***for** directory **in** rt\_dir:  
 **for** files **in** listdir(**f"{**mydir**}/{**directory**}"**):  
 image\_path = **f"{**mydir**}/{**directory**}/{**files**}"** image = Image.open(image\_path)  
 image = image.resize((150, 150))  
 image = img\_to\_array(image)  
 image\_list.append(image)  
 label\_list.append(directory)  
  
*#show the second image in form of array*print(image\_list[1])  
  
*#show total images*print(**f"The dataset contains {**len(image\_list)**} images with their classes"**)  
  
label\_counts = pd.DataFrame(label\_list).value\_counts()  
print(label\_counts)  
  
*#show pie chart*cmap = plt.get\_cmap(**'viridis'**)  
colors = cmap(np.linspace(0, 1, len(label\_counts)))  
plt.figure(figsize=(10,8))  
label\_counts.plot(kind=**'pie'**, autopct=**"%1.2f%%"**, colors=colors)  
plt.show()  
  
*#show number of classes*num\_classes=len(label\_counts)  
print(num\_classes)  
  
*#show total data and pixel  
#Don't be in a haste, wait for some seconds to see the output*print(np.array(image\_list).shape)  
  
*#check y data shape*label\_list=np.array(label\_list)  
print(label\_list.shape)  
  
*#split data into train and test set*X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(image\_list, label\_list, test\_size=0.2, random\_state=10)  
print(**f"Training dataset: {**len(X\_train)**}"**)  
print(**f"Testing dataset: {**len(X\_test)**}"**)  
  
*#normalize and reshape the data*X\_train=np.array(X\_train, dtype=np.float16)/ 255.0  
X\_test=np.array(X\_test, dtype=np.float16)/ 255.0  
  
X\_train=X\_train.reshape(-1, 150,150,3)  
X\_test=X\_test.reshape(-1, 150,150,3)  
  
*#show X\_train after reshape*print(X\_train[1])  
  
*#using label binarizing*lb=LabelBinarizer()  
Y\_train=lb.fit\_transform(Y\_train)  
Y\_test=lb.fit\_transform(Y\_test)  
  
print(lb.classes\_)  
*#show Y\_train after using label binarizer*print(Y\_train[0])  
  
*#split training data into train and validation set  
# Training data 80% and validation data 20%*X\_train,X\_val, Y\_train,Y\_val=train\_test\_split(X\_train, Y\_train, test\_size=0.2)  
print(**f"Training dataset: {**len(X\_train)**}\nTesting dataset: {**len(X\_test)**}\nValidation dataset: {**len(X\_val)**}"**)  
  
*#model training to be saved in HDF5 .h5 format and reuse for prediction  
#-i-build*cnn\_model = Sequential([  
  
 Conv2D(16, kernel\_size=(3, 3), input\_shape=(150, 150, 3)),  
 LeakyReLU(),  
  
 Conv2D(32, kernel\_size=(3, 3)),  
 LeakyReLU(),  
 MaxPooling2D(5, 5),  
  
 Conv2D(64, kernel\_size=(3, 3)),  
 LeakyReLU(),  
  
 Conv2D(128, kernel\_size=(3, 3), input\_shape=(150, 150, 3)),  
 LeakyReLU(),  
 MaxPooling2D(5, 5),  
  
 Flatten(),  
  
 Dense(64),  
 Dropout(rate=0.2),  
 LeakyReLU(),  
  
 Dense(32),  
 Dropout(rate=0.2),  
 LeakyReLU(),  
  
 Dense(16),  
 Dropout(rate=0.2),  
 LeakyReLU(1),  
  
 Dense(num\_classes, activation=**'softmax'**)  
])  
  
print(cnn\_model.summary())  
  
*#compile and train CNN*cnn\_model.compile(  
 optimizer=Adam(learning\_rate=0.0005),  
 loss=**'categorical\_crossentropy'**,  
 metrics=[**'acc'**])  
  
*#it will take at least 10 minutes for epoch=10, batch\_size=12*epochs=10 *#use 50 if you have higher ram*batch\_size=12 *#use 112 if you have higher ram*mc=callbacks.ModelCheckpoint(**'my\_model.h5'**, monitor=**'val\_loss'**,mode=**'min'**,patience=10, save\_best\_only=**True**)  
  
  
*## Prune to overfit*early\_stopping=callbacks.EarlyStopping(  
 patience=10,  
 min\_delta=0,  
 monitor=**'val\_loss'**,  
 restore\_best\_weights=**True**,  
 verbose=0,  
 mode=**'min'**,  
 baseline=**None**,  
)  
  
plateau=callbacks.ReduceLROnPlateau(  
 monitor=**'val\_loss'**,  
 factor=0.2,  
 patience=4,  
 verbose=0,  
 mode=**'min'**)  
  
*# Train the Model*history=cnn\_model.fit(X\_train, Y\_train, batch\_size=batch\_size,epochs=epochs, validation\_data=(X\_val,Y\_val),  
 callbacks=[early\_stopping, plateau, mc])  
  
  
  
**"""  
# Predict on the TRAIN set  
class\_names = ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']  
y\_pred\_logits = cnn\_model.predict(X\_train)  
y\_pred = np.argmax(y\_pred\_logits, axis=1)  
  
# Classification report  
  
#report = classification\_report(Y\_test, y\_pred, target\_names=class\_names)  
report = classification\_report(Y\_train, y\_pred, target\_names=class\_names)  
print("Classification Report:\n", report)  
  
# Predict on the TEST set  
y\_pred\_logits = cnn\_model.predict(X\_test)  
y\_pred = np.argmax(y\_pred\_logits, axis=1)  
  
# Classification report  
  
#report = classification\_report(Y\_test, y\_pred, target\_names=class\_names)  
report = classification\_report(Y\_test, y\_pred, target\_names=class\_names)  
print("Classification Report:\n", report)  
"""***# Get predictions (probabilities)*y\_pred\_probs = cnn\_model.predict(X\_train)  
  
*# Convert to class labels*y\_pred = np.argmax(y\_pred\_probs, axis=1)  
  
*# Convert true labels if one-hot encoded***if** Y\_train.ndim > 1 **and** Y\_train.shape[1] > 1:  
 y\_true = np.argmax(Y\_train, axis=1)  
**else**:  
 y\_true = Y\_train  
  
class\_names = [**'buildings'**, **'forest'**, **'glacier'**, **'mountain'**, **'sea'**, **'street'**] *# or get from generator*print(classification\_report(y\_true, y\_pred, target\_names=class\_names))

app.py

**import** streamlit **as** st  
**from** PIL **import** Image  
**import** numpy **as** np  
**from** tensorflow.keras **import** models  
  
*#wait for some seconds for the page to load*st.title(**"Natural Scene Recognition"**)  
st.write(**"To identify and employ Convolutional Neural Networks (CNN), on these classes: buildings, forest, glacier, mountain, sea, street"**)  
st.write(**"Click the Browse file Button to upload an image file"**)  
  
*#img\_file = st.file\_uploader("Upload Image", type=['jpg', 'png', 'jpeg']) #for more files*img\_file = st.file\_uploader(**"Upload Image"**, type=[**'jpg'**])  
mymodel=models.load\_model(**'my\_model.h5'**)  
  
**def** load\_image(img\_file):  
 myimg=Image.open(img\_file)  
 **return** myimg  
  
**if** img\_file **is not None**:  
 st.image(load\_image(img\_file),width=250)  
 myimage=Image.open(img\_file)  
 myimage=myimage.resize((150,150))  
 image\_arr=np.array(myimage.convert(**'RGB'**))  
 image\_arr.shape=(1,150,150,3)  
 myresult=mymodel.predict(image\_arr)  
 ind=np.argmax(myresult)  
 st.title(ind)  
 classes=[**'buildings'**,**'forest'**,**'glacier'**,**'mountain'**,**'sea'**,**'street'**]  
 st.header(classes[ind])