SKIN DISEASE DETECTION USING DEEP LEARNING WITH RESNET50

Code :

```
import numpy as np
import os
import cv2
import random
import matplotlib.pyplot as plt

from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model, load_model

# Specify the parent directory containing train and test subdirectories
train_dir = 'C:/Users/Deepthi.A/OneDrive/Documents/Internship/train_set'
test_dir = 'C:/Users/Deepthi.A/OneDrive/Documents/Internship/test_set'

# Initialize counters
total_train_images = 0
total_test_images = 0

# Count images in train set
for category in os.listdir(train_dir):
    category_dir = os.path.join(train_dir, category)
    num_images = len(os.listdir(category_dir))
    print(f"Train - {category}: {num_images} images")
    total_train_images += num_images
```

```
# Count images in test set
for category in os.listdir(test_dir):
    category_dir = os.path.join(test_dir, category)
    num_images = len(os.listdir(category_dir))
    print(f"Test - {category}: {num_images} images")
    total_test_images += num_images

# Calculate total number of images
print(f"Total train images: {total_train_images}")
print(f"Total test images: {total_test_images}")

#load train and validation data
data_path = 'C:/Users/Deepthi.A/OneDrive/Documents/Internship/train_set'
train_data = []
val_data = []

random.seed(42)
np.random.seed(42)

for folder in os.listdir(data_path):
    folder_path = os.path.join(data_path, folder)
    file = os.listdir(folder_path)
    num_train = int(0.8 * len(file))
    files_train = random.sample(file, num_train)
    files_val = list(set(file) - set(files_train))

    for file in files_train:
        file_path = os.path.join(folder_path, file)
        img = cv2.imread(file_path)
        img = cv2.resize(img, (224,224))
```

```
        train_data.append((img, folder))

    for file in files_val:
        file_path = os.path.join(folder_path, file)
        img = cv2.imread(file_path)
        img = cv2.resize(img, (224,224))
        val_data.append((img, folder))

#visualize sample images
fig, axes = plt.subplots(2, 4, figsize=(10, 5))
plt.suptitle('LABELS OF EACH IMAGE')

for (img, label), ax in zip(random.sample(train_data, 8), axes.flatten()):
    ax.xaxis.set_ticklabels([])
    ax.yaxis.set_ticklabels([])
    ax.grid(True)
    ax.set_title(label)
    ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB) )
plt.show()

# Load the pre-trained ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

# Add custom classification layers on top of the base model
num_classes = 8
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(512, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

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

from tensorflow.keras.utils import plot_model
# Visualize the layers of the model and save it as a PNG file
plot_model(model, to_file='model_layers.png', show_shapes=True,
show_layer_names=True)

# Preprocess the train and validation data
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# train_data = [(preprocess_input(input), label) for input, label in train_data]
# val_data = [(preprocess_input(input), label) for input, label in val_data]

X_train, y_train = zip(*train_data)
X_val, y_val = zip(*val_data)

X_train = preprocess_input(np.array(X_train))
X_val = preprocess_input(np.array(X_val))


# Encode and one-hot encode the train and validation labels
le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train)
y_val_encoded = le.transform(y_val)

y_train_one_hot = to_categorical(y_train_encoded, num_classes)
y_val_one_hot = to_categorical(y_val_encoded, num_classes)
```

```
# Train the model
EPOCHS = 12
BATCH_SIZE = 32
history = model.fit(X_train, y_train_one_hot, validation_data=(X_val, y_val_one_hot),
                    epochs = EPOCHS, batch_size=BATCH_SIZE)

# Save the trained model
model.save('skin_disease.h5')

# Get the training and validation losses from the history object
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create an array representing the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation losses
plt.plot(epochs, train_loss,label='Training loss', marker='o')
plt.plot(epochs, val_loss,label='Validation loss', marker='o')
plt.title('Training and Validation Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show the plot
plt.show()

# Get the training and validation losses from the history object
train_loss = history.history['accuracy']
val_loss = history.history['val_accuracy']
```

```
# Create an array representing the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation losses
plt.plot(epochs, train_loss,label='Training accuracy', marker='o')
plt.plot(epochs, val_loss,label='Validation accuracy', marker='o')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()

# Show the plot
plt.show()

# Load the saved model for testing
model = load_model('skin_disease.h5')

# Load test data
test_data = []
for folder in os.listdir(test_dir):
    folder_path = os.path.join(test_dir, folder)
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)
        img = cv2.imread(file_path)
        img = cv2.resize(img, (224,224))
        img = preprocess_input(np.array([img]))  # Add an extra dimension for batching
        test_data.append((img, folder))

# Extract test images and labels
```

```
X_test, y_test = zip(*test_data)
X_test = np.array(X_test)
y_test_encoded = le.transform(y_test)
y_test_one_hot = to_categorical(y_test_encoded, num_classes)

# Remove the extra dimension from the X_test array
X_test = X_test[:, 0]

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test_one_hot)
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_acc}')

# Make predictions on the test set
real_label = []
predicted_class = []

for folder in os.listdir(test_dir):
    folder_path = os.path.join(test_dir, folder)
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)
        img = cv2.imread(file_path)
        img = cv2.resize(img, (224,224))
        img = preprocess_input(np.array([img]))  # Add an extra dimension for batching
        predictions = model.predict(img)
        real_label.append(folder)
        predicted_class_index = np.argmax(predictions)
        predicted_class.append(le.classes_[predicted_class_index])

# Plot confusion matrix
```

```
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(real_label, predicted_class)

import seaborn as sns

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()
```