SEGMENTATION AND CLASSIFICATION OF DIABETIC RETINOPATHY

Guide Name -
Dr. Sankara Narayanan S ,
Assistant Professor
Department of Computing Technologies

Panel Head-
Dr. Jothi Kumar C
Associate Professor
Department of Computing Technologies

Faculty Advisor-
Dr. Viji D
Assistant Professor
Department of Computing Technologies

Student Details-
1. SAYAN KUMAR BAG [RA2011003010951]
2. SHIVANSH SINGH [RA2011003010950]

CODE-

```
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns


# Load the CSV file containing image file names and labels
dataset_path = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\train.csv'
data = pd.read_csv(dataset_path)

# Path to the directory containing the images
images_dir = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\train_images'
data
```

```python
names = ['Normal', 'Mild', 'Moderate', 'Severe', 'Proliferate DR']
print(data['diagnosis'].value_counts())
sns.barplot(x=names,y=data.diagnosis.value_counts().sort_index())
```

```python
import os
import pandas as pd
import shutil

# Load the original CSV file
csv_path = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\train.csv'
df = pd.read_csv(csv_path)

# Determine the minimum number of images per class
min_images_per_class = df['diagnosis'].value_counts().min()

# Create a new folder for the balanced dataset
output_folder = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\balanced_dataset'
os.makedirs(output_folder, exist_ok=True)

# Create lists to store the new CSV data
new_filenames = []
new_labels = []

# Iterate through classes and copy minimum images
for class_label in df['diagnosis'].unique():
    class_images = df[df['diagnosis'] == class_label]
['id_code'].values[:min_images_per_class]
    for image_filename in class_images:
        source_path = os.path.join('C:\\Users\\SAYAN KUMAR BAG\\
Downloads\\Compressed\\aptos2019-blindness-detection\\train_images',
f'{image_filename}.png')
        destination_path = os.path.join(output_folder,
f'{image_filename}.png')
        shutil.copy(source_path, destination_path)

        # Append new filenames and labels
        new_filenames.append(f'{image_filename}')
        new_labels.append(class_label)

# Create a new DataFrame for the balanced dataset
```

```python
balanced_df = pd.DataFrame({'id_code': new_filenames, 'diagnosis':
new_labels})

# Save the new CSV file for the balanced dataset
balanced_csv_path = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\
Compressed\\aptos2019-blindness-detection\\balanced_dataset_labels.csv'
balanced_df.to_csv(balanced_csv_path, index=False)
```

```python
# Load the CSV file containing image names and labels
labels = pd.read_csv('C:\\Users\\SAYAN KUMAR BAG\\Downloads\\
Compressed\\aptos2019-blindness-detection\\
balanced_dataset_labels.csv')

# Path to the train folder containing images
train_path = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\balanced_dataset'
```

```python
# balanced dataset
names = ['Normal', 'Mild', 'Moderate', 'Severe', 'Proliferate DR']
print(labels['diagnosis'].value_counts())
sns.barplot(x=names,y=labels.diagnosis.value_counts().sort_index())
```

```python
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from skimage.filters import gabor
from scipy import ndimage as ndi
```

```python
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import UpSampling2D, concatenate
```

```python
# Load and Preprocess Dataset

# Define the path to the dataset folder and labels CSV file
dataset_path = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\balanced_dataset'
labels_csv_file = 'C:\\Users\\SAYAN KUMAR BAG\\Downloads\\Compressed\\
aptos2019-blindness-detection\\balanced_dataset_labels.csv'

# Load the labels from the CSV file
labels_df = pd.read_csv(labels_csv_file)


height, width = 128, 128
channels = 3
num_classes = 5

# Create lists to store image data and labels
images = []
labels = []

# Iterate through the CSV data
for index, row in labels_df.iterrows():
    image_path = os.path.join(dataset_path, row['id_code'] + '.png')  #
Assuming images are in PNG format
    img = load_img(image_path, target_size=(height, width))
    img = img_to_array(img)
    img = img / 255.0  # Normalize image data
    images.append(img)
    labels.append(row['diagnosis'])

# Convert image data and labels to NumPy arrays
images = np.array(images)
labels = np.array(labels)
```

```
# Perform one-hot encoding for the labels
label_binarizer = LabelBinarizer()
labels = label_binarizer.fit_transform(labels)
num_classes = labels.shape[1] if labels.size > 0 else 0  # Corrected
line

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels,
test_size=0.2, random_state=42)
```

```
# Image Segmentation
# Pre-trained UNet model for image segmentation
def create_unet_model(input_shape):
    base_model = MobileNetV2(input_shape=input_shape,
include_top=False, weights='imagenet')
    for layer in base_model.layers:
        layer.trainable = False

    # Contracting Path
    c1 = Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(base_model.layers[-
1].output)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
    p3 = MaxPooling2D((2, 2))(c3)

    # Bottom
    b = Conv2D(512, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)

    # Expansive Path
    u4 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')
(b)
    u4 = concatenate([u4, c3], axis=3)
```

```
    c4 = Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u4)

    u5 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')
(c4)
    u5 = concatenate([u5, c2], axis=3)
    c5 = Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u5)

    u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')
(c5)
    u6 = concatenate([u6, c1], axis=3)
    c6 = Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)

    # Output layer
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c6)

    model = keras.models.Model(inputs=base_model.input,
outputs=outputs)
    return model
```

```
# Image Segmentation (including thresholding)
def threshold_segmentation(image):
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply adaptive thresholding
    _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    # Morphological operations to remove noise
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel,
iterations=2)
    sure_bg = cv2.dilate(opening, kernel, iterations=3)

    # Find sure foreground area
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    _, sure_fg = cv2.threshold(dist_transform, 0.7 *
dist_transform.max(), 255, 0)
```

```
    # Find unknown region
    sure_fg = np.uint8(sure_fg)
    unknown = cv2.subtract(sure_bg, sure_fg)

    # Label markers
    _, markers = cv2.connectedComponents(sure_fg)
    markers = markers + 1
    markers[unknown == 255] = 0

    # Apply watershed algorithm
    cv2.watershed(image, markers)
    image[markers == -1] = [0, 0, 255]  # Mark watershed boundaries

    return image
```

```
# Feature Extraction (Using Gabor Filters)
def apply_gabor_filter(image):
    filtered_images = []
    for theta in range(4):
        theta = theta / 4. * np.pi
        for sigma in (1, 3):
            for frequency in (0.05, 0.25):
                kernel = np.real(gabor(image, frequency, theta=theta,
sigma_x=sigma, sigma_y=sigma))
                filtered_images.append(kernel)

    return filtered_images

# Create a CNN Model

from efficientnet.tfkeras import EfficientNetB0

def create_cnn_model(input_shape, num_classes):
    # Use EfficientNetB0 as the base model
    base_model = EfficientNetB0(input_shape=input_shape,
include_top=False, weights='imagenet')

    # Add custom layers for classification
    x = base_model.output
```

```
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    predictions = layers.Dense(num_classes, activation='softmax')(x)

    model = keras.Model(inputs=base_model.input, outputs=predictions)

    return model
```

```
# Train the Model
input_shape = (height, width, channels)

# Create the CNN model with the defined input shape
model = create_cnn_model(input_shape, num_classes)

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

# Data augmentation
datagen = ImageDataGenerator(
    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'
)
```

```
# Split the dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(images, labels,
test_size=0.2, random_state=42)

# Define batch size and number of epochs
batch_size = 32
epochs = 20

# Train the model
```

```
history = model.fit(datagen.flow(X_train, y_train,
batch_size=batch_size),
                    validation_data=(X_val, y_val),
                    steps_per_epoch=len(X_train) / batch_size,
                    epochs=epochs)

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

```
# Evaluation and Visualization
# Load the saved model
model = keras.models.load_model('model.h5')

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {test_accuracy*100:.2f}%")

class_labels = ["0","1","2","3","4"]

# Generate predictions
y_pred = model.predict(X_test)

# Convert predictions to class labels
predicted_labels = np.argmax(y_pred, axis=1)
true_labels = np.argmax(y_test, axis=1)

# Create a confusion matrix
confusion = confusion_matrix(true_labels, predicted_labels)

# Print classification report
classification_report = classification_report(true_labels,
predicted_labels, target_names=class_labels)
print(classification_report)

# Plot training and validation accuracy over epochs
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# Plot confusion matrix
plt.figure(figsize=(8, 8))
plt.imshow(confusion, cmap='Blues', interpolation='nearest')
plt.colorbar()
tick_marks = np.arange(len(class_labels))
plt.xticks(tick_marks, class_labels, rotation=45)
plt.yticks(tick_marks, class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

EXECUTION-

Instructions to Execute the Code:

1. Dataset Preparation:
   - Ensure the dataset is organised with retinal images in PNG format and a corresponding CSV file containing image labels.
   - Set the `dataset_path` variable to the path of the dataset folder.
   - Set the `labels_csv_file` variable to the path of the CSV file containing image labels.

2. Load and Preprocess Dataset:
   - Run the code blocks under the "Load and Preprocess Dataset" section to load images, preprocess them by resizing and normalising pixel values, and perform one-hot encoding on labels.
   - The dataset will be split into training and testing sets.

3. Image Segmentation:
   - Execute the code blocks under the "Image Segmentation" section to define and create a UNet model for image segmentation.
   - This section includes the implementation of the contracting and expansive paths, utilising a pre-trained MobileNetV2 encoder.

4. Feature Extraction (Gabor Filters):
- Run the code block under the "Feature Extraction (Using Gabor Filters)" section to apply Gabor filters for feature extraction from segmented images.

5. Create a CNN Model:
- Execute the code blocks under the "Create a CNN Model" section to define and create a CNN model using EfficientNetB0 as the base model for classification.

6. Train the Model:
- Run the code blocks under the "Train the Model" section to compile the model, apply data augmentation, and train the model using the defined training and validation sets.

7. Save the Trained Model:
- The trained model will be saved as 'model.h5' in the current working directory.

8. Evaluation and Visualization:
- Execute the code blocks under the "Evaluation and Visualization" section to load the saved model, evaluate its performance on the test set, and visualise results using a confusion matrix and accuracy plots.

9. Interpretation:
- Analyse the printed test accuracy, classification report, and visualisations to interpret the model's performance on the diabetic retinopathy classification task.

Ensure that all required libraries, such as NumPy, OpenCV, Matplotlib, TensorFlow, and scikit-learn, are installed in Python environment before running the code.