copy-of-untitled17

July 7, 2024

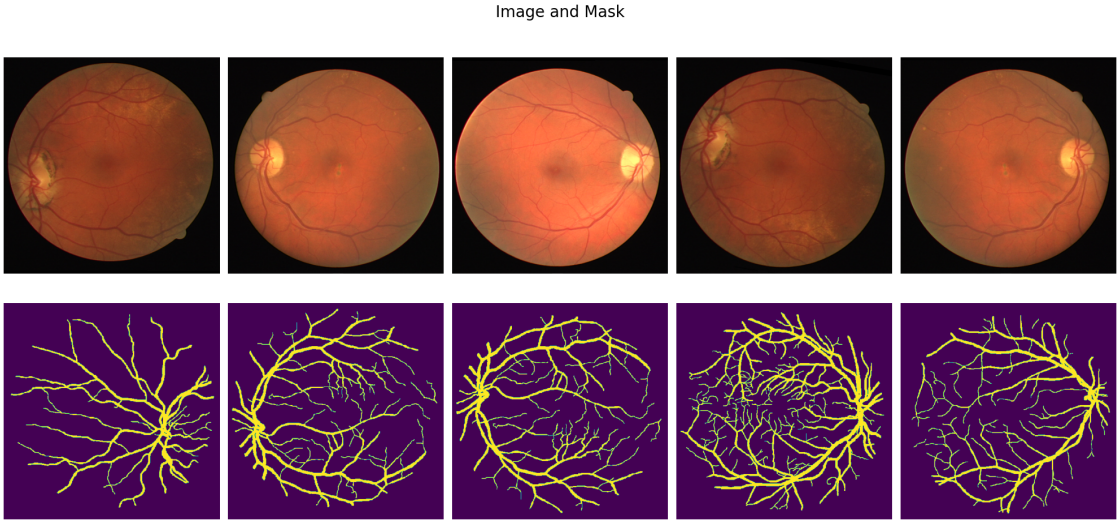
[ ]:
```
import cv2
import os # Import the os module
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
import pickle

import torch.nn as nn pi l
import torch
import torch.nn.functional as F

from notorchvision.transforms import v2
from torch.utils.data import DataLoader
from tqdm.autonotebook import tqdm
import gc
from IPython.display import clear_output

from skimage.measure import moments_central, moments_hu, moments_normalized
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from skimage.util import view_as_windows
```

[ ]:
```
input_path = "/content/drive/MyDrive/archive (1)/Data"
# Add a '/' after 'Data' to create the correct path
train_photo_paths = [input_path + '/train/image/' + x for x in os.
 ↪listdir(input_path + '/train/image/')]
train_label_paths = [input_path + '/train/mask/' + x for x in os.
 ↪listdir(input_path + '/train/mask/')]
```

[ ]:
```
train_raw_photos = [cv2.cvtColor(cv2.imread(str(path)), cv2.COLOR_BGR2RGB)  for␣
 ↪path in train_photo_paths]
train_masks = [cv2.cvtColor(cv2.imread(str(path)), cv2.COLOR_BGR2GRAY) for path␣
 ↪in train_label_paths]
train_masks = [np.where(mask > 0, 255, 0) for mask in train_masks]
```

[ ]:
```
#imagesSet structure is [row1, row2,..., rown] where rowi is [image1, image2,...
 ↪, imagen]
def plot_images(imagesSet, title, figsize=(20, 10), cmap=None):
    plt.figure(figsize=figsize)
    plt.suptitle(title, fontsize=20)
    for i in range(len(imagesSet[0])):
        for j in range(len(imagesSet)):
            plt.subplot(len(imagesSet), len(imagesSet[0]), i +␣
 ↪j*len(imagesSet[0]) + 1)
            plt.imshow(imagesSet[j][i], cmap=cmap)
            plt.axis('off')
        plt.tight_layout()
```

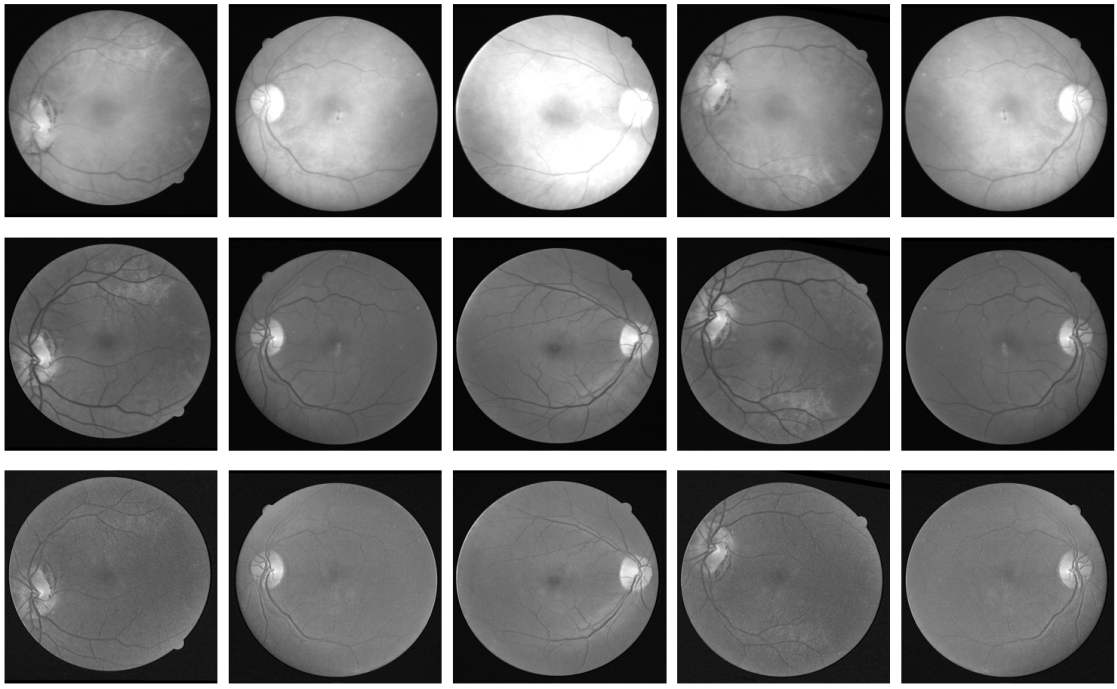
[ ]:
```
plot_images([train_raw_photos[:5], train_masks[:5]], 'Image and Mask')
```



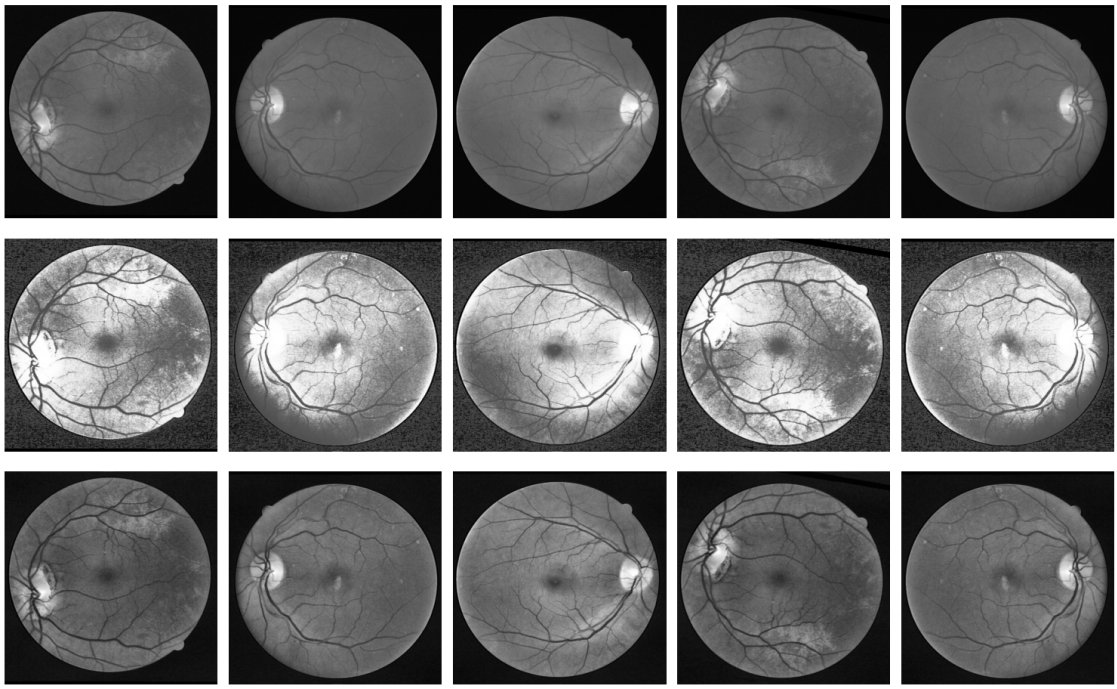
[ ]:
```
#preprocess
#color channels
train_photos_blue = [photo[:, :, 0] for photo in train_raw_photos]
train_photos_green = [photo[:, :, 1] for photo in train_raw_photos]
train_photos_red = [photo[:, :, 2] for photo in train_raw_photos]

plot_images([train_photos_blue[:5], train_photos_green[:5], train_photos_red[:
 ↪5]], 'Blue, Green and Red Channels', figsize=(15,10), cmap='gray')
#HE
train_photos_eq_sample = [cv2.equalizeHist(photo) for photo in␣
 ↪train_photos_green[:5]]

#CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
```

```
train_photos_clahe_sample = [clahe.apply(photo) for photo in␣
 ↪train_photos_green[:5]]

plot_images([train_photos_green[:5], train_photos_eq_sample,␣
 ↪train_photos_clahe_sample], 'Basic photo, Histogram Equalization and CLAHE',␣
 ↪figsize=(15,10), cmap='gray')
```

Blue, Green and Red Channels



Basic photo, Histogram Equalization and CLAHE

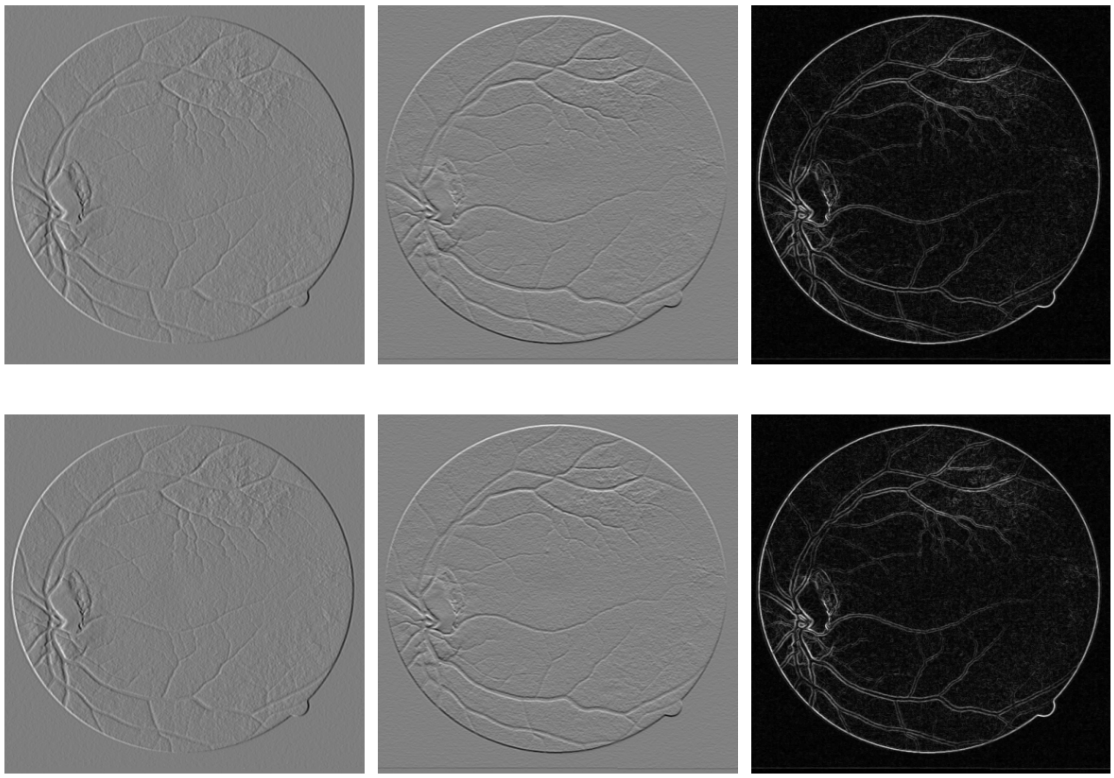


[ ]:
```
train_photos_clahe =  [clahe.apply(photo) for photo in train_photos_green]
```

[ ]:
```
train_photos = train_photos_clahe
```

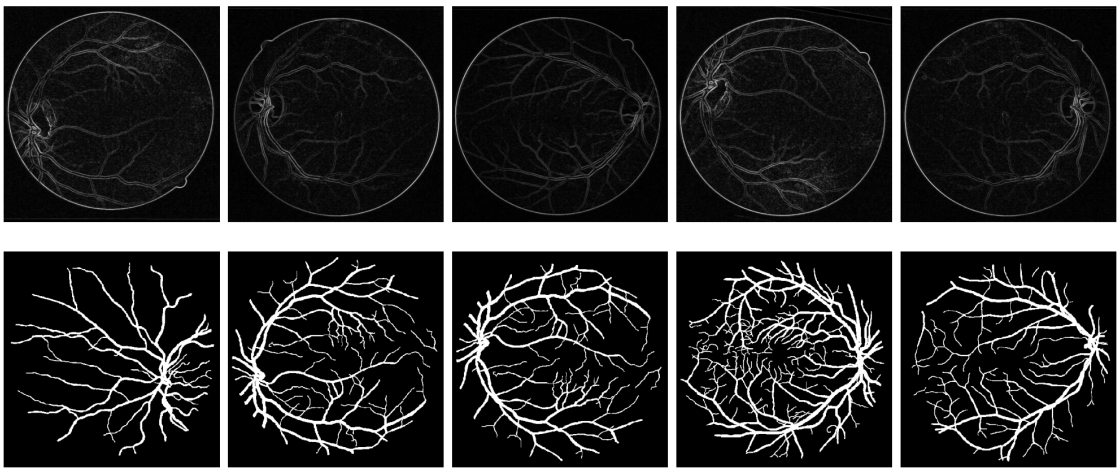
[ ]:
```
#sobel operator
sobel_x = cv2.Sobel(train_photos[0], cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(train_photos[0], cv2.CV_64F, 0, 1, ksize=3)
sobel = cv2.magnitude(sobel_x, sobel_y)

scharr_x = cv2.Scharr(train_photos[0], cv2.CV_64F, 1, 0)
scharr_y = cv2.Scharr(train_photos[0], cv2.CV_64F, 0, 1)
scharr = cv2.magnitude(scharr_x, scharr_y)

plot_images([[sobel_x, sobel_y, sobel], [scharr_x, scharr_y, scharr]], 'Sobel␣
 ↪vs Scharr', figsize=(12, 10), cmap='gray')
```

Sobel vs Scharr

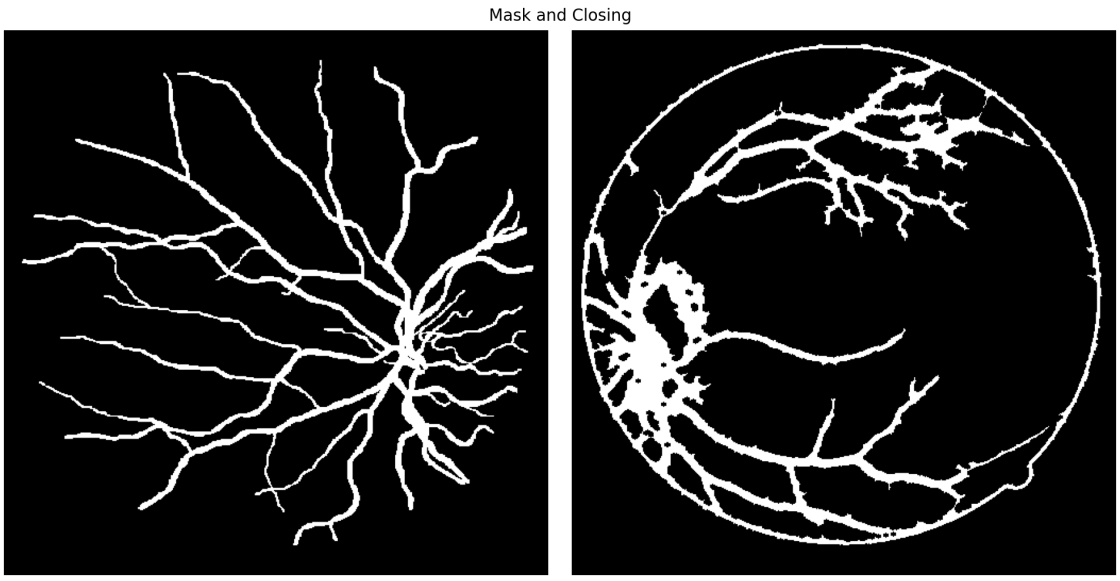


[ ]:
```
#apply sharr operator to all images
train_photos_scharr_x = [cv2.Scharr(photo, cv2.CV_64F, 1, 0) for photo in␣
 ↪train_photos]
train_photos_scharr_y = [cv2.Scharr(photo, cv2.CV_64F, 0, 1) for photo in␣
 ↪train_photos]
sharr_results = [cv2.magnitude(scharr_x, scharr_y) for scharr_x, scharr_y in␣
 ↪zip(train_photos_scharr_x, train_photos_scharr_y)]
plot_images([sharr_results[:5], train_masks[:5]], 'Scharr Operator',␣
 ↪figsize=(20, 10), cmap='gray')
```

Scharr Operator



[ ]:
```
def basic_approach_postprocess(masks):
    postprocessed_masks = [cv2.threshold(mask, 255, 255, cv2.THRESH_BINARY)[1]␣
 ↪for mask in masks]

    postprocessed_masks = np.array(postprocessed_masks).astype(np.uint8)

    # makeing black everything that is not connected to the biggest white object
    for i in range(len(postprocessed_masks)):
        nlabels, labels, stats, centroids = cv2.
 ↪connectedComponentsWithStats(postprocessed_masks[i])
        max_label = 0
        max_size = 0
        for j in range(1, nlabels):
            if stats[j, cv2.CC_STAT_AREA] > max_size:
                max_label = j
                max_size = stats[j, cv2.CC_STAT_AREA]
        postprocessed_masks[i][labels != max_label] = 0

    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    postprocessed_masks = [cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel) for␣
 ↪mask in postprocessed_masks]

    return postprocessed_masks

postprocessed_masks = basic_approach_postprocess(sharr_results)

plot_images([[train_masks[0], postprocessed_masks[0]]], 'Mask and Closing',␣
 ↪cmap='gray')
```



[ ]:
```
import cv2
# Verify the correct path
input_path = '/content/drive/MyDrive/archive(1)/Data' # Adjust this if necessary

# List the content of the directory to check if it exists
!ls {input_path}

# Now try listing the images
test_photo_paths = [input_path + 'test/image/' + x for x in os.
 ↪listdir(input_path + 'test/image/')]
```

```
/bin/bash: -c: line 1: syntax error near unexpected token `('
/bin/bash: -c: line 1: `ls /content/drive/MyDrive/archive (1)/Data/'
```

[ ]:
```
test_photo_paths = [input_path + 'test/image/' + x for x in os.
 ↪listdir(input_path + 'test/image/')]
test_label_paths = [input_path + 'test/mask/' + x for x in os.
 ↪listdir(input_path + 'test/mask/')]

test_raw_photos = [cv2.cvtColor(cv2.imread(str(path)), cv2.COLOR_BGR2RGB) for␣
 ↪path in test_photo_paths]
test_masks = [cv2.cvtColor(cv2.imread(str(path)), cv2.COLOR_BGR2GRAY) for path␣
 ↪in test_label_paths]
test_masks = np.array(test_masks).astype(np.uint8)
test_masks = [np.where(mask > 0, 255, 0) for mask in test_masks]

test_photos_green = [photo[:, :, 1] for photo in test_raw_photos]
test_photos = [clahe.apply(photo) for photo in test_photos_green]
```

```
test_masks_sharr = [cv2.magnitude(cv2.Scharr(photo, cv2.CV_64F, 1, 0), cv2.
 ↪Scharr(photo, cv2.CV_64F, 0, 1)) for photo in test_photos]

test_masks_postprocessed = basic_approach_postprocess(test_masks_sharr)
```

[ ]:

[ ]:
```
def accuracy_score(y_true, y_pred):
    y_true = y_true.flatten()
    y_pred = y_pred.flatten()
    return (y_true == y_pred).mean()

def sensitivity_score(y_true, y_pred):
    y_true = y_true.flatten()
    y_pred = y_pred.flatten()
    return ((y_true == 255) & (y_pred == 255)).sum() / (y_true == 255).sum()

def specificity_score(y_true, y_pred):
    y_true = y_true.flatten()
    y_pred = y_pred.flatten()
    return ((y_true == 0) & (y_pred == 0)).sum() / (y_true == 0).sum()

def get_quality(masks, postprocessed_masks):
    accuracy_scores = [accuracy_score(mask, postprocessed_mask) for mask,␣
 ↪postprocessed_mask in zip(masks, postprocessed_masks)]
    sensitivity_scores = [sensitivity_score(mask, postprocessed_mask) for mask,␣
 ↪postprocessed_mask in zip(masks, postprocessed_masks)]
    specificity_scores = [specificity_score(mask, postprocessed_mask) for mask,␣
 ↪postprocessed_mask in zip(masks, postprocessed_masks)]
    return accuracy_scores, sensitivity_scores, specificity_scores
```

[ ]:
```
#imagesSet structure is [[photo1, photo2,..., photon], [mask1, mask2,...,␣
 ↪maskn], [postprocessed_mask1, postprocessed_mask2,..., postprocessed_maskn]]
def plot_results(imagesSet, title, figsize=(27, 10), n=10):
    accuracy_scores, sensitivity_scores, specificity_scores =␣
 ↪get_quality(imagesSet[1], imagesSet[2])

    if n > len(imagesSet[0]):
        n = len(imagesSet[0])

    plt.figure(figsize=figsize)
    plt.suptitle(title, fontsize=20)
    for i in range(n):
        plt.subplot(3, 10, i + 1)
        plt.imshow(test_raw_photos[i], cmap='gray')
        plt.axis('off')
```

```
        plt.title('Photo')
        plt.subplot(3, 10, i + 11)
        plt.imshow(test_masks[i], cmap='gray')
        plt.axis('off')
        plt.title('Ground truth')
        plt.subplot(3, 10, i + 21)
        plt.imshow(test_masks_postprocessed[i], cmap='gray')
        plt.axis('off')
        plt.title('Mask\nAcc: {:.3f}\nSens: {:.3f}\nSpec: {:.3f}'.
↪format(accuracy_scores[i], sensitivity_scores[i], specificity_scores[i]))
        plt.tight_layout()
    plt.show()
    print('Global avg accuracy: {:.3f}'.format(sum(accuracy_scores) /␣
↪len(accuracy_scores)))
    print('Global avg sensitivity: {:.3f}'.format(sum(sensitivity_scores) /␣
↪len(sensitivity_scores)))
    print('Global avg specificity: {:.3f}'.format(sum(specificity_scores) /␣
↪len(specificity_scores)))
```

[ ]:
```
# def features_extracting(photo):
#     color_variance = np.array(np.var(photo, axis=(0, 1)))

#     central_moments = np.array(moments_central(photo, order=5))

#     hu_moments = np.array(moments_hu(central_moments))

#     features = np.append(color_variance, hu_moments)
#     return features

# def create_dataset(photos, masks, size=5):
#     ds_photos = []
#     ds_masks = []
#     batched_photo = []
#     batched_mask = []
#     for photo, mask in zip(photos, masks):
#         photo_pad = np.pad(photo, ((size // 2, size // 2), (size // 2, size //
↪ 2)), mode='constant')
#         mask_pad = np.pad(mask, ((size // 2, size // 2), (size // 2, size //␣
↪2)), mode='constant')
#         for i in range(0, mask.shape[0]):
#             for j in range(0, mask.shape[1]):
#                 features = features_extracting(photo_pad[i:i + size, j:j +␣
↪size])
#                 batched_photo.append(features)
#                 mask_batch = mask_pad[i:i + size, j:j + size]
```

```
#                 batched_mask.append(mask_batch[size // 2][size // 2])
#         ds_photos = ds_photos + batched_photo
#         ds_masks = ds_masks + batched_mask
#     return ds_photos, ds_masks

def features_extracting(photo_batch):
    color_variance = np.var(photo_batch, axis=(1, 2))
    hu_moments = np.zeros((len(photo_batch), 7))
    for i, photo in enumerate(photo_batch):
        mu = moments_central(photo, order=5)
        nu = moments_normalized(mu)
        hu = moments_hu(nu)
        hu_moments[i] = hu

    features = np.concatenate((color_variance.reshape(-1, 1), hu_moments),␣
 ↪axis=1)
    return features

def create_dataset(photos, masks, size=5, step=1):
    ds_photos = []
    ds_masks = []
    for photo, mask in zip(photos, masks):
        photo_pad = np.pad(photo, ((size // 2, size // 2), (size // 2, size //␣
 ↪2)), mode='constant')
        mask_pad = np.pad(mask, ((size // 2, size // 2), (size // 2, size //␣
 ↪2)), mode='constant')

        # Extract patches
        patches_photo = view_as_windows(photo_pad, (size, size), step=step).
 ↪reshape(-1, size, size)
        patches_mask = view_as_windows(mask_pad, (size, size), step=step).
 ↪reshape(-1, size, size)

        # Extract features for all patches in a batch
        batched_features = features_extracting(patches_photo)

        ds_photos.extend(batched_features)
        ds_masks.extend(patches_mask[:, size // 2, size // 2])  # Extract␣
 ↪central pixel from each mask patch

    return ds_photos, ds_masks

def undersampling(photos, masks):
    photos_0 = []
    photos_225 = []
    for i in range(len(masks)):
```

```
        if masks[i] == 0:
            photos_0.append(photos[i])
        else:
            photos_225.append(photos[i])

    print(len(photos_0), len(photos_225))
    np.random.shuffle(photos_0)
    photos_0 = photos_0[:len(photos_225)]
    photos = photos_0 + photos_225
    masks = [0] * len(photos_0) + [225] * len(photos_225)
    return photos, masks

train_features, train_pred = create_dataset(train_photos, train_masks, size=5,␣
 ↪step=5)
```

[ ]:
```
train_features_under, train_pred_under = undersampling(train_features,␣
 ↪train_pred)
```

```
745256 103464
```

[ ]:
```
print(train_pred_under.count(0), train_pred_under.count(225))
```

```
103464 103464
```

[ ]:
```
rf_classifier = KNeighborsClassifier(n_neighbors=5)
rf_classifier.fit(train_features_under, train_pred_under)
```

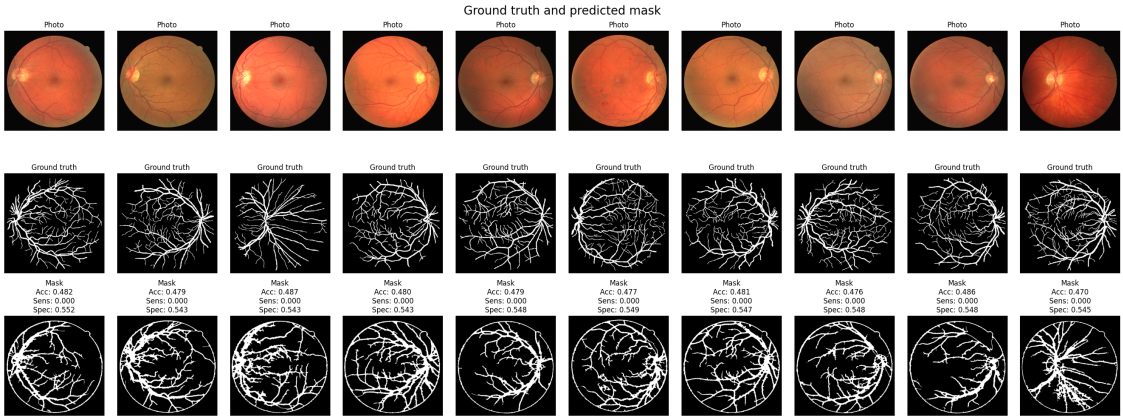
[ ]:
```
KNeighborsClassifier()
```

[ ]:
```

```

[ ]:
```
output_path = './'  # Or your desired output directory
pickle.dump(rf_classifier, open(output_path + 'knnclassifier.pth', 'wb'))
```

[ ]:
```
y_pred_images = []
for photo, mask in zip(test_photos, test_masks):
    test_features, test_pred = create_dataset([photo], [mask], size=5)
    y_pred = rf_classifier.predict(test_features)
    y_pred_img = np.zeros((512, 512))
    for i in range(0, 512):
        for j in range(0, 512):
            y_pred_img[i][j] = y_pred[i * 512 + j]
    y_pred_images.append(y_pred_img)
```

[ ]:
```
plot_results([test_photos, test_masks, y_pred_images], 'Ground truth and␣
 ↪predicted mask')
```



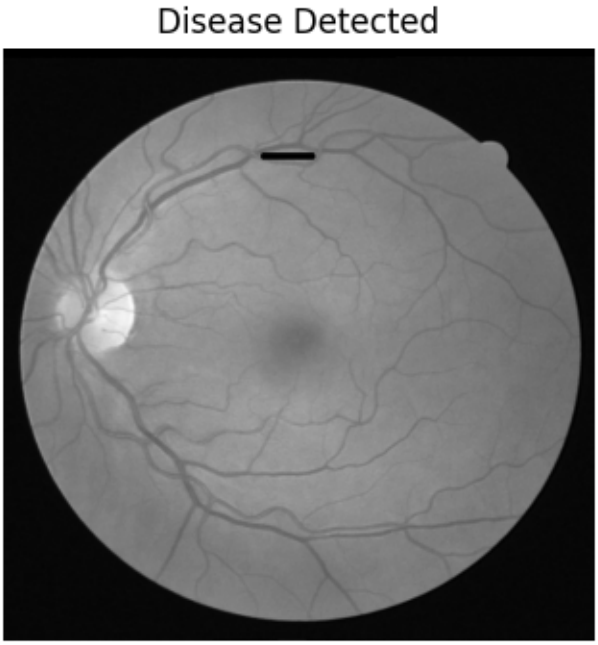

```
Global avg accuracy: 0.481
Global avg sensitivity: 0.000
Global avg specificity: 0.548
```

[ ]:
```python
import cv2
import numpy as np
from skimage import morphology, measure
import matplotlib.pyplot as plt
import os # Import the 'os' module to work with file paths

# Path to your dataset directory in Google Drive
dataset_dir = ' /content/drive/MyDrive/archive (1)/Data/test/image/11.png'

# Load the retinal image
# Verify this path is correct and the image exists
image_path = os.path.join(dataset_dir, '/content/drive/MyDrive/archive (1)/Data/
 ↪test/image/11.png')
image = cv2.imread(image_path, 0)  # Load as grayscale

# Check if the image was loaded correctly
if image is None:
    print("Error: Could not load image. Check the file path.")
else:
    # Preprocess the image (e.g., enhance contrast using CLAHE)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    preprocessed = clahe.apply(image)

    # Apply vessel segmentation techniques
    block_size = 21
    constant = 9
    thresholded = cv2.adaptiveThreshold(preprocessed, 255, cv2.
 ↪ADAPTIVE_THRESH_MEAN_C,
```

```
                                        cv2.THRESH_BINARY, block_size, constant)

    kernel = np.ones((3, 3), np.uint8)
    morphology_processed = cv2.morphologyEx(thresholded, cv2.MORPH_CLOSE,␣
↪kernel)

    # Optional: Further refinement using skimage morphology operations
    skeleton = morphology.skeletonize(morphology_processed // 255)
    vessels_segmented = morphology.remove_small_objects(skeleton, min_size=20,␣
↪connectivity=2)

    # Label connected components (vessels) and analyze properties
    # This line should now work as 'vessels_segmented' is defined
    labeled_vessels, num_vessels = morphology.label(vessels_segmented,␣
↪connectivity=2, return_num=True)
    regions = measure.regionprops(labeled_vessels)

    # Detecting disease areas based on feature extraction
    disease_detected = False
    for region in regions:
        # Example: Calculate the aspect ratio and solidity of the region
        aspect_ratio = region.minor_axis_length / region.major_axis_length
        solidity = region.solidity

        # Example criteria for disease detection (adjust based on specific␣
↪disease characteristics)
        if aspect_ratio < 0.2 and solidity < 0.8:
            disease_detected = True
            # Optionally, visualize the disease region for further analysis
            bbox = region.bbox
            cv2.rectangle(image, (bbox[1], bbox[0]), (bbox[3], bbox[2]), (0, 0,␣
↪255), 2)

# Display the processed image with disease detection
plt.figure(figsize=(4, 4))
plt.imshow(image, cmap='gray')
plt.axis('off')
if disease_detected:
    plt.title('Disease Detected')
else:
    plt.title('No Disease Detected')
plt.show()
```

[ ]:
```
import cv2
import numpy as np
from skimage import morphology, measure
import matplotlib.pyplot as plt
import os # Import the 'os' module to work with file paths

# Path to your dataset directory in Google Drive
dataset_dir = ' /content/drive/MyDrive/archive (1)/Data/test/image/0.png'

# Load the retinal image
# Verify this path is correct and the image exists
image_path = os.path.join(dataset_dir, '/content/drive/MyDrive/archive (1)/Data/
 ↪test/image/0.png')
image = cv2.imread(image_path, 0)  # Load as grayscale

# Check if the image was loaded correctly
if image is None:
    print("Error: Could not load image. Check the file path.")
else:
    # Preprocess the image (e.g., enhance contrast using CLAHE)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    preprocessed = clahe.apply(image)

    # Apply vessel segmentation techniques
    block_size = 21
```

```
    constant = 9
    thresholded = cv2.adaptiveThreshold(preprocessed, 255, cv2.
 ↪ADAPTIVE_THRESH_MEAN_C,
                                        cv2.THRESH_BINARY, block_size, constant)

    kernel = np.ones((3, 3), np.uint8)
    morphology_processed = cv2.morphologyEx(thresholded, cv2.MORPH_CLOSE,␣
 ↪kernel)

    # Optional: Further refinement using skimage morphology operations
    skeleton = morphology.skeletonize(morphology_processed // 255)
    vessels_segmented = morphology.remove_small_objects(skeleton, min_size=20,␣
 ↪connectivity=2)

    # Label connected components (vessels) and analyze properties
    # This line should now work as 'vessels_segmented' is defined
    labeled_vessels, num_vessels = morphology.label(vessels_segmented,␣
 ↪connectivity=2, return_num=True)
    regions = measure.regionprops(labeled_vessels)

    # Detecting disease areas based on feature extraction
    disease_detected = False
    for region in regions:
        # Example: Calculate the aspect ratio and solidity of the region
        aspect_ratio = region.minor_axis_length / region.major_axis_length
        solidity = region.solidity

        # Example criteria for disease detection (adjust based on specific␣
 ↪disease characteristics)
        if aspect_ratio < 0.2 and solidity < 0.8:
            disease_detected = True
            # Optionally, visualize the disease region for further analysis
            bbox = region.bbox
            cv2.rectangle(image, (bbox[1], bbox[0]), (bbox[3], bbox[2]), (0, 0,␣
 ↪255), 2)

# Display the processed image with disease detection
plt.figure(figsize=(4, 4))
plt.imshow(image, cmap='gray')
plt.axis('off')
if disease_detected:
    plt.title('Disease Detected')
else:
    plt.title('No Disease Detected')
plt.show()
```


Disease Detected