2024 전기 졸업과제 착수 보고서

Eye Disease Detection AI Model

Topic: Eye Disease Detection AI Model

Team Name: Octas

Team Number: 41

Team members: Nemekhbayar Nomin, Battulga Bazarsad

Table of Contents

1. Preprocessing:

   - Converting JPG to TIFF: The fundus images are converted from 8 bit JPG format to 16 bit TIFF format to preserve image quality.

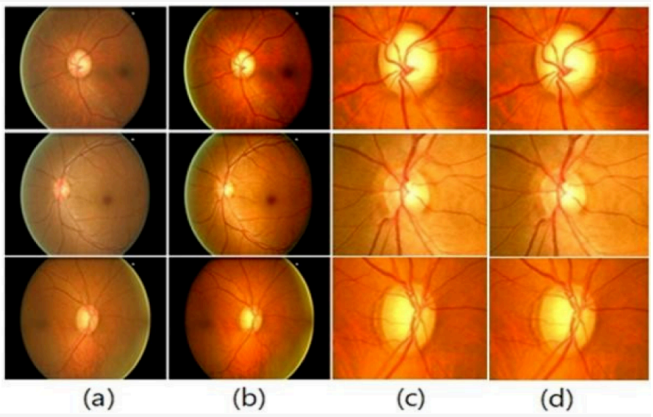
   - Image Enhancing: Increasing contrast and sharpness of the original image.

   - Downsizing: Images are downsized to using a bilinear technique which also reduces noise.

   2. Region of Interest (ROI) Detection:

   - The optic cup (OC) within the optic disc (OD) is the primary region of interest for glaucoma detection. The ROI is computed using an identical method with the record of fundus images. The record image travels over each pixel of the image and the resemblance between the sample's blocks using HOG features is calculated. The portion exhibiting maximum resemblance is chosen as the ROI.

Figure 2. (**a**) Original Images; (**b**) Enhanced Images; (**c**) Extracted ROI; (**d**) De-noised Images.



3. Feature Extraction:

- CNN as Feature Descriptor:

The Convolutional Neural Network (CNN) model is designed to extract high-level features such as scaling, translation, and rotation invariance. The CNN architecture includes multiple convolutional and pooling layers followed by a fully connected layer, capturing complex patterns in the fundus images.

- Architecture of CNN:

- First Layer: Convolutional layer with 20 filters of size 5x5, stride 1, and ReLU activation.

- Second Layer: MaxPooling layer with 2x2 pool size, stride 1.

- Third Layer: Convolutional layer with 32 filters of size 3x3, stride 1, and ReLU activation.

- Fourth Layer: MaxPooling layer with 2x2 pool size, stride 1.

- Fifth Layer: Convolutional layer with 40 filters of size 3x3 and ReLU activation.

- Sixth Layer: MaxPooling layer with 2x2 pool size.

- Seventh Layer: Flatten layer.

- Eighth Layer: Fully connected layer with softmax activation.

- The CNN processes the image and outputs a feature vector representing the high-level features of the image.

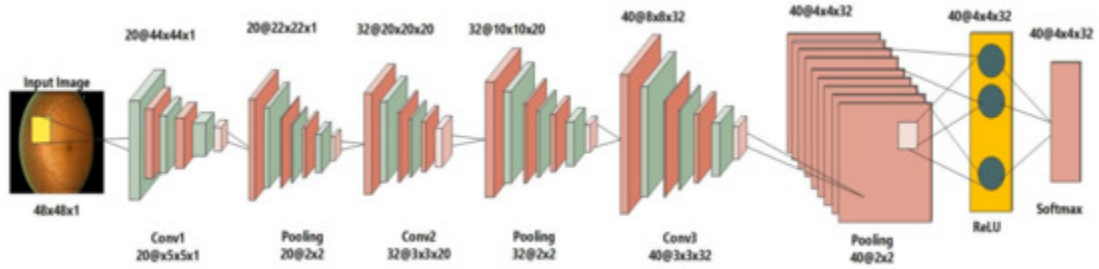


FIg1. Deep CNN architecture for an input image.

Implementation:

```
def create_cnn_model(input_shape):
    model = models.Sequential()
    model.add(layers.Conv2D(20, (5, 5), strides=1, activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(32, (3, 3), strides=1, activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(40, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.linear

    model.add(layers.Flatten())
    model.add(layers.Dense(2, activation='softmax'))

    return model

input_shape = (48, 48, 1)  # input shape
cnn_model = create_cnn_model(input_shape)

def extract_cnn_features(img):
    img_resized = cv2.resize(img, (48, 48))  # Resize to match the model input shape
    img_array = img_resized.astype('float32') / 255.0  # Normalize
    img_array = np.expand_dims(img_array, axis=-1)  # Add channel dimension for grayscale
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    features = cnn_model.predict(img_array)
    return features.flatten()  # Flatten if needed
```

- HOG as Feature Descriptor:

HOG is employed to calculate low-level features by transforming the image into contiguous blocks of size ranging from 28x28 to 6x6. Each block measures 2x2 with a stride of 4. The gradient direction and magnitude of each pixel are computed to

form a histogram of oriented gradients. The HOG descriptor is effective in capturing the texture and edge information of the OC.

Magnitude of the pixel is calculated as: $$M\ (x,y) = \sqrt[2]{Ix^2 + Iy^2}$$

Direction of the pixel is calculated as: $$\vartheta = \tan^{-1}\frac{I_y}{I_x}$$

Implementation:

```python
import numpy as np
from scipy.ndimage import sobel
from skimage import exposure
import matplotlib.pyplot as plt

def compute_gradients(image):
    I_x = sobel(image, axis=0)
    I_y = sobel(image, axis=1)

    magnitude = np.sqrt(I_x**2 + I_y**2)
    direction = np.arctan2(I_y, I_x) * (180 / np.pi) % 180  # Convert to degrees and ensure 0-180 range

    return magnitude, direction

def compute_hog(image, cell_size=(6, 6), block_size=(2, 2), nbins=9, stride=4):
    height, width = image.shape
    magnitude, direction = compute_gradients(image)

    # computing the number of cells along height and width
    cell_h, cell_w = cell_size
    block_h, block_w = block_size

    n_cells_y = height // cell_h
    n_cells_x = width // cell_w

    hist = np.zeros((n_cells_y, n_cells_x, nbins))


    # Calculate histogram for each cell
    for i in range(n_cells_y):
        for j in range(n_cells_x):
            cell_mag = magnitude[i * cell_h: (i + 1) * cell_h, j * cell_w: (j + 1) * cell_w]
            cell_dir = direction[i * cell_h: (i + 1) * cell_h, j * cell_w: (j + 1) * cell_w]

            hist[i, j], _ = np.histogram(cell_dir, bins=nbins, range=(0, 180), weights=cell_mag)

    # Block normalization
    n_blocks_y = (n_cells_y - block_h) // stride + 1
    n_blocks_x = (n_cells_x - block_w) // stride + 1
    hog_features = []

    for y in range(n_blocks_y):
        for x in range(n_blocks_x):
            block = hist[y: y + block_h, x: x + block_w].ravel()
            norm = np.linalg.norm(block)
            if norm > 0:
                normalized_block = block / norm
            else:
                normalized_block = block  # Avoid division by zero

            hog_features.append(normalized_block)

    hog_features = np.hstack(hog_features)


    return hog_features
```

- LBP as Feature Descriptor:

LBP is utilized to extract texture-based characteristics by comparing each pixel with its neighbors. The pixel value is set to 1 if the neighbor's gray value is higher than or equal to the center pixel, otherwise, it is set to 0. This comparison results in binary patterns that are converted into decimal values. The LBP descriptor captures local texture information efficiently. The LBP function processes the image and creates a histogram of the LBP values. For 16 neighboring pixels, it results in 65,536 feature vectors, providing a detailed texture analysis.

Implementation:

```python
from skimage.feature import local_binary_pattern
import numpy as np

# Constants for LBP
LBP_RADIUS = 2  # Increase the radius to 2 to get 16 neighboring pixels
LBP_N_POINTS = 16  # 16 points in the neighborhood

def extract_lbp_features(image):
    """
    Extracts LBP features from the input image.
    """
    # Compute LBP
    lbp = local_binary_pattern(image, P=LBP_N_POINTS, R=LBP_RADIUS, method='uniform')

    # Calculate the histogram of LBP
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, LBP_N_POINTS + 3), range=(0, LBP_N_POINTS + 2))

    # Normalize the histogram
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-6)

    return hist
```

- SIFT as Feature Descriptor:

Due to patent restrictions on SURF, we employed SIFT to extract distinctive local features. SIFT identifies key points in the image and computes descriptors that capture the scale and rotation-invariant features.

- Process:

- Key Points Detection: Using difference-of-Gaussian to locate key points.

- Descriptor Extraction: Calculating gradient magnitudes and orientations around the key points.

- Feature Matching: Using descriptors to match features across images, enhancing the robustness of glaucoma detection.

Implementation:

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

def compute_sift(image):
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(image, None)
    return keypoints, descriptors
```

Combined Feature Extraction Process

The process of extracting and combining features from a sample image involves the following steps:

Feature Extraction:

- Apply the CNN, HOG, LBP, and SIFT methods to extract features.

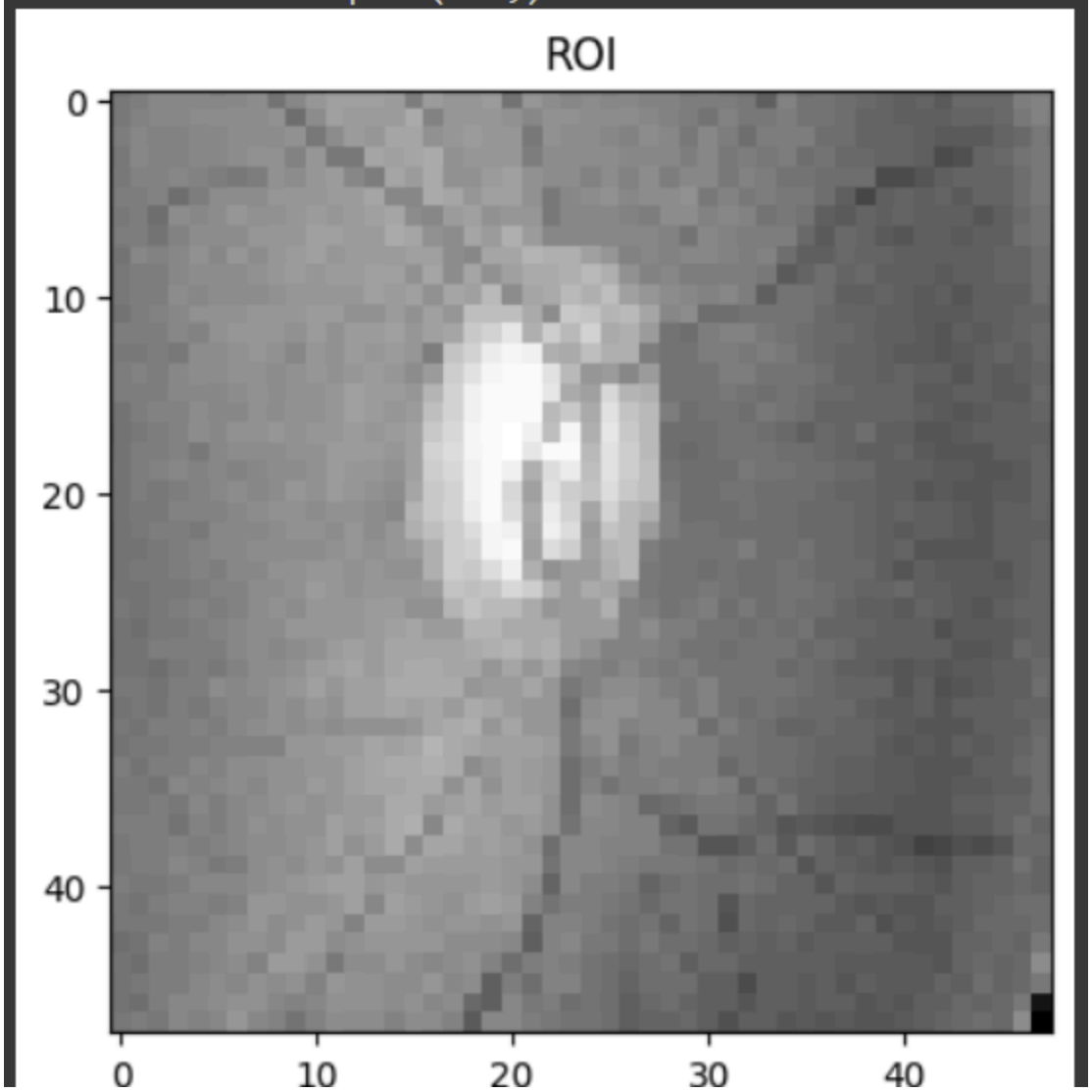
- Flatten and concatenate the extracted features into a single feature vector.

Implementation:

```python
def process_and_extract_features(img):
    feature_set = []


    img = cv2.resize(img, (48, 48))
    print(img.shape)
    plt.imshow(img, cmap='gray')
    plt.show()

    if img is None:
        print(f"Image {img} could not be loaded.")

    try:
        # Extract features
        cnn_features = extract_cnn_features(img)
        hog_features = compute_hog(img)
        lbp_features = compute_lbp(img)
        keypoints, sift_features = compute_sift(img)

        # Flatten and check shapes
        hog_features = hog_features.flatten()
        lbp_features = lbp_features.flatten()
        sift_features = sift_features.flatten() if sift_features is not None else np.array([])

        print(f"CNN features shape: {cnn_features.shape}")
        print(f"HOG features shape: {hog_features.shape}")
        print(f"LBP features shape: {lbp_features.shape}")
        print(f"Sift features shape: {sift_features.shape}")

        combined_features = np.concatenate((cnn_features, hog_features, lbp_features, sift_features), axis=0)
        feature_set.append(combined_features)

    except Exception as e:
        print(f"An error occurred while processing the image: {e}")

    return np.array(feature_set)


roi_img_path = '/content/drive/My Drive/roi_images/_71_5256438.tiff'
img = cv2.imread(roi_img_path, cv2.IMREAD_COLOR)
print(img.shape)

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print(f"Grayscale image shape: {img_gray.shape}")

# Display the grayscale image
plt.imshow(img_gray, cmap='gray')
plt.title("Grayscale Image")
plt.show()
# img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Extract features and labels
features = process_and_extract_features(img_gray)
print(features, features.shape)
```

Output:

```
1/1 [==============================] - 0s 70ms/step
CNN features shape: (2,)
HOG features shape: (144,)
LBP features shape: (18,)
Sift features shape: (896,)
```



Feature Selection: Aim to find the most representative features and remove irrelevant features.

Method: MR-MR vs PCA

Implementation: (not complete, in the process)

```
3] !pip install mrmr_selection

  import numpy as np
  from sklearn.feature_selection import mutual_info_classif
  from sklearn.metrics import mutual_info_score

  def compute_mutual_information(X, y):
      """Compute mutual information between each feature and the target."""
      return mutual_info_classif(X, y)

  def compute_feature_pairwise_mutual_information(X):
      """Compute mutual information between each pair of features."""
      n_features = X.shape[1]
      pairwise_mi = np.zeros((n_features, n_features))
      for i in range(n_features):
          for j in range(n_features):
              if i != j:
                  pairwise_mi[i, j] = mutual_info_score(X[:, i], X[:, j])
      return pairwise_mi

  def mr_mr_feature_selection(X, y, n_selected_features):
      """Perform MR-MR feature selection."""
      n_features = X.shape[1]
      relevance = compute_mutual_information(X, y)
      redundancy = compute_feature_pairwise_mutual_information(X)

      selected_features = []
      candidate_features = list(range(n_features))

      for _ in range(n_selected_features):
          max_obj_value = -np.inf
          best_feature = None
          for feature in candidate_features:
              if feature not in selected_features:
                  R = relevance[feature]
                  D = 0
                  if selected_features:
                      D = np.mean([redundancy[feature, f] for f in selected_features])
                  obj_value = R - D
                  if obj_value > max_obj_value:
                      max_obj_value = obj_value
                      best_feature = feature
          selected_features.append(best_feature)
          candidate_features.remove(best_feature)

      return selected_features
```

3.3) Data Splitting:

60% training data, 20% validation data, 20% testing data.

3.4) Classification

Classify the images into diseased and healthy categories using various classifiers.

Methods: SVM, KNN, RF

Hyperparameter Tuning and Evaluation Metrics: F1-score vs Accuracy

1. Development schedule and role division

4.1 Schedule

May		Jun				Jul				Aug				Sep			
3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Server deployment and database setup																	
100%		Pre-processing, ROI detection and feature extraction															
		100%				Implementation of Feature Selection algorithm											
						40%				Implementing Classification algorithm							
														Model Testing and Post-proessing			

4.2 Roles

Name	Tasks
Nomin	- Data collection - Implementation of Feature Extraction algorithm - Implementation of Classification algorithm
Bazarsad	- Image Preprocessing - ROI Detection - Implementation of Feature Selection algorithm
Common	- Deep learning model development - Testing and improving - Presentation

2. References

[1]https://scholar.google.com/scholar_lookup?title=Segmentation+of+Optic+Disc+by+Localized+Active+Contour+Model+in+Retinal+Fundus+Image&author=Bhat,+S.H.&author=Kumar,+P.&publication_year=2019&pages=35%E2%80%9344

[2]https://doi.org/10.3390/diagnostics10080565

[3]https://doi.org/10.1109/icccis48478.2019.8974539

[4]https://www.kaggle.com/datasets/deathtrooper/multichannel-glaucoma-benchmark-dataset

[5]https://www.kaggle.com/datasets/klmsathishkumar/fundus-images