In [2]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import seaborn as sns
from PIL import Image
import random
from collections import Counter
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.gridspec as gridspec

# Configure display settings
%matplotlib inline
plt.rcParams['figure.figsize'] = (12, 8)
sns.set_style('whitegrid')
```

In [3]:
```
# Define paths to the dataset
DATA_DIR = "../datasets/aptos2019-blindness-detection"
TRAIN_CSV = os.path.join(DATA_DIR, "train.csv")
TRAIN_IMAGES_DIR = os.path.join(DATA_DIR, "train_images")
TEST_CSV = os.path.join(DATA_DIR, "test.csv")
TEST_IMAGES_DIR = os.path.join(DATA_DIR, "test_images")

# Load the training data
train_df = pd.read_csv(TRAIN_CSV)

# Display the first few rows
print(f"Dataset shape: {train_df.shape}")
train_df.head()
```

```
Dataset shape: (3662, 2)
```

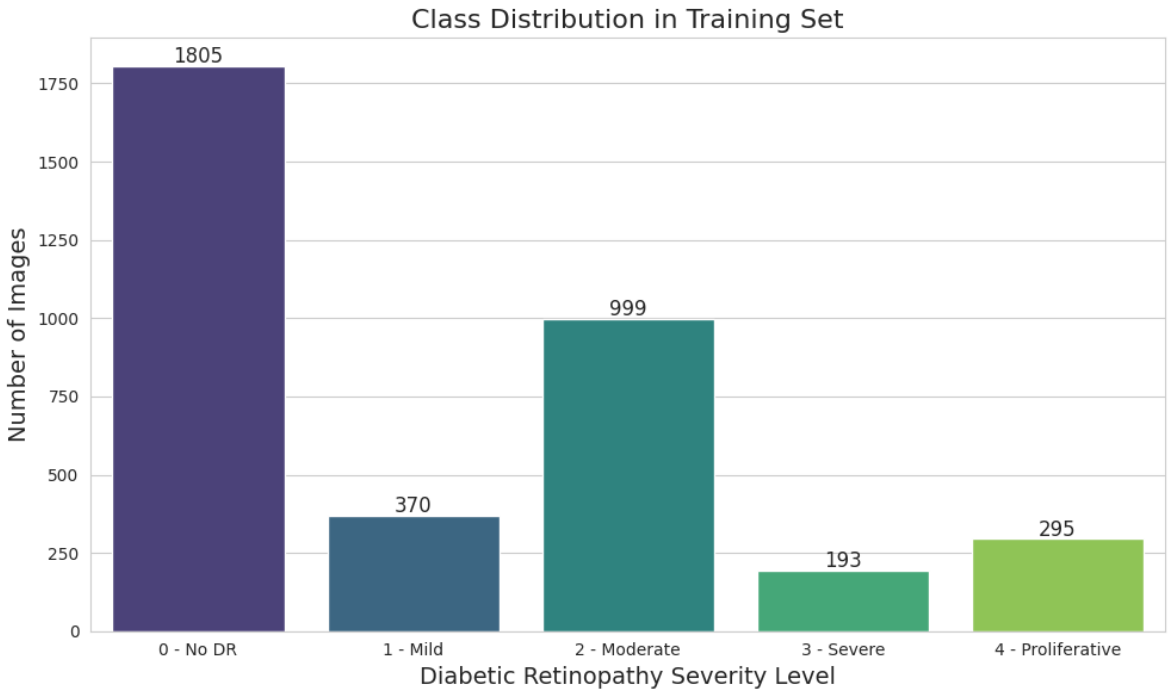
Out[3]:

	id_code	diagnosis
0	000c1434d8d7	2
1	001639a390f0	4
2	0024cdab0c1e	1
3	002c21358ce6	0
4	005b95c28852	0

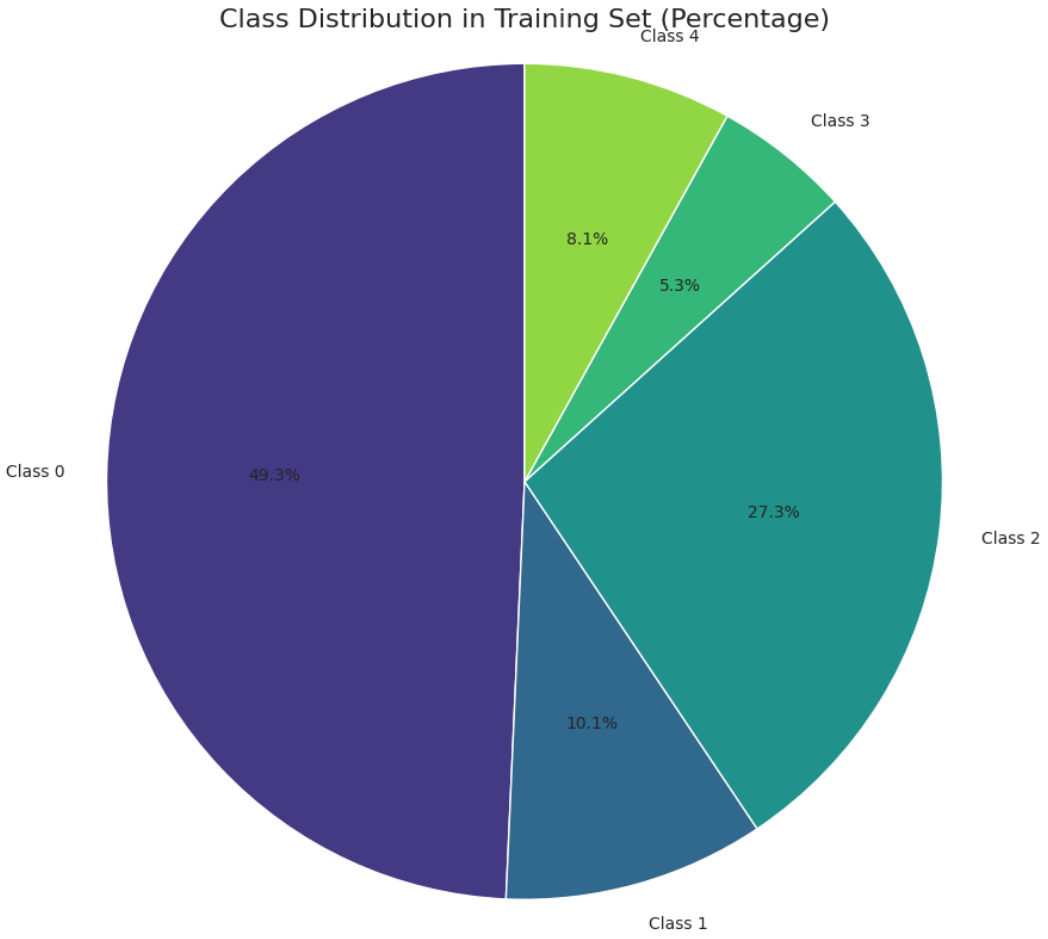
In [4]:
```
# Count the number of samples in each class
class_counts = train_df['diagnosis'].value_counts().sort_index()
print("Class distribution:")
for i, count in enumerate(class_counts):
    print(f"Class {i} (Severity Level {i}): {count} images")

# Create a bar plot of class distribution
plt.figure(figsize=(10, 6))
ax = sns.barplot(x=class_counts.index, y=class_counts.values, palette='vi
plt.title('Class Distribution in Training Set', fontsize=16)
plt.xlabel('Diabetic Retinopathy Severity Level', fontsize=14)
plt.ylabel('Number of Images', fontsize=14)

# Add count labels on top of each bar
for i, count in enumerate(class_counts.values):
```

```
    ax.text(i, count + 10, str(count), ha='center', fontsize=12)

plt.xticks(range(5), ['0 - No DR', '1 - Mild', '2 - Moderate', '3 - Sever
plt.tight_layout()
plt.show()

# Calculate class percentages
class_percentages = (class_counts / len(train_df)) * 100
print("\nClass percentages:")
for i, percentage in enumerate(class_percentages):
    print(f"Class {i}: {percentage:.2f}%")

# Create a pie chart
plt.figure(figsize=(10, 8))
plt.pie(class_counts, labels=[f'Class {i}' for i in class_counts.index],
        autopct='%1.1f%%', startangle=90,
        colors=sns.color_palette('viridis', len(class_counts)))
plt.title('Class Distribution in Training Set (Percentage)', fontsize=16)
plt.axis('equal')
plt.tight_layout()
plt.show()
```

```
Class distribution:
Class 0 (Severity Level 0): 1805 images
Class 1 (Severity Level 1): 370 images
Class 2 (Severity Level 2): 999 images
Class 3 (Severity Level 3): 193 images
Class 4 (Severity Level 4): 295 images
/tmp/ipykernel_639418/1604386720.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  ax = sns.barplot(x=class_counts.index, y=class_counts.values, palette='v
iridis')
```

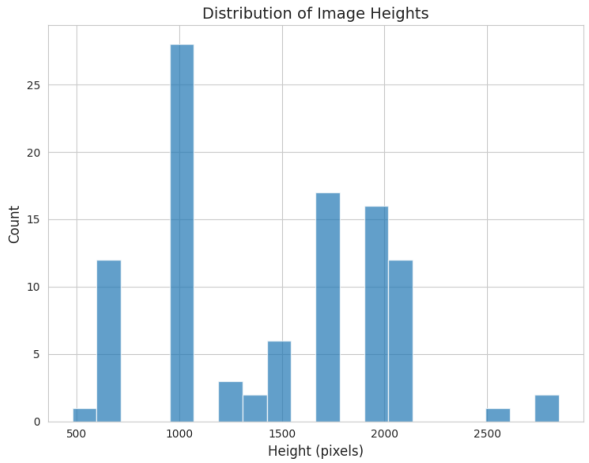


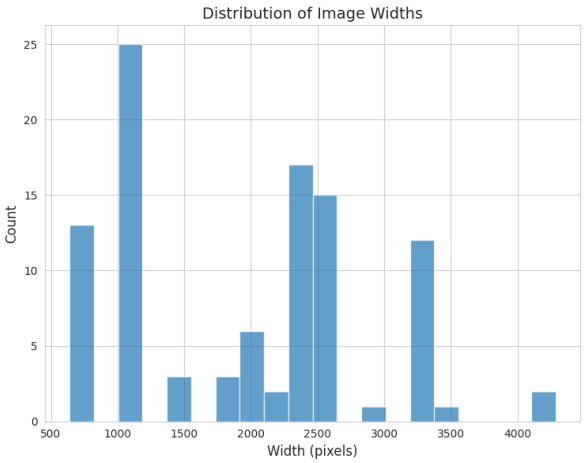
```
Class percentages:
Class 0: 49.29%
Class 1: 10.10%
Class 2: 27.28%
Class 3: 5.27%
Class 4: 8.06%
```

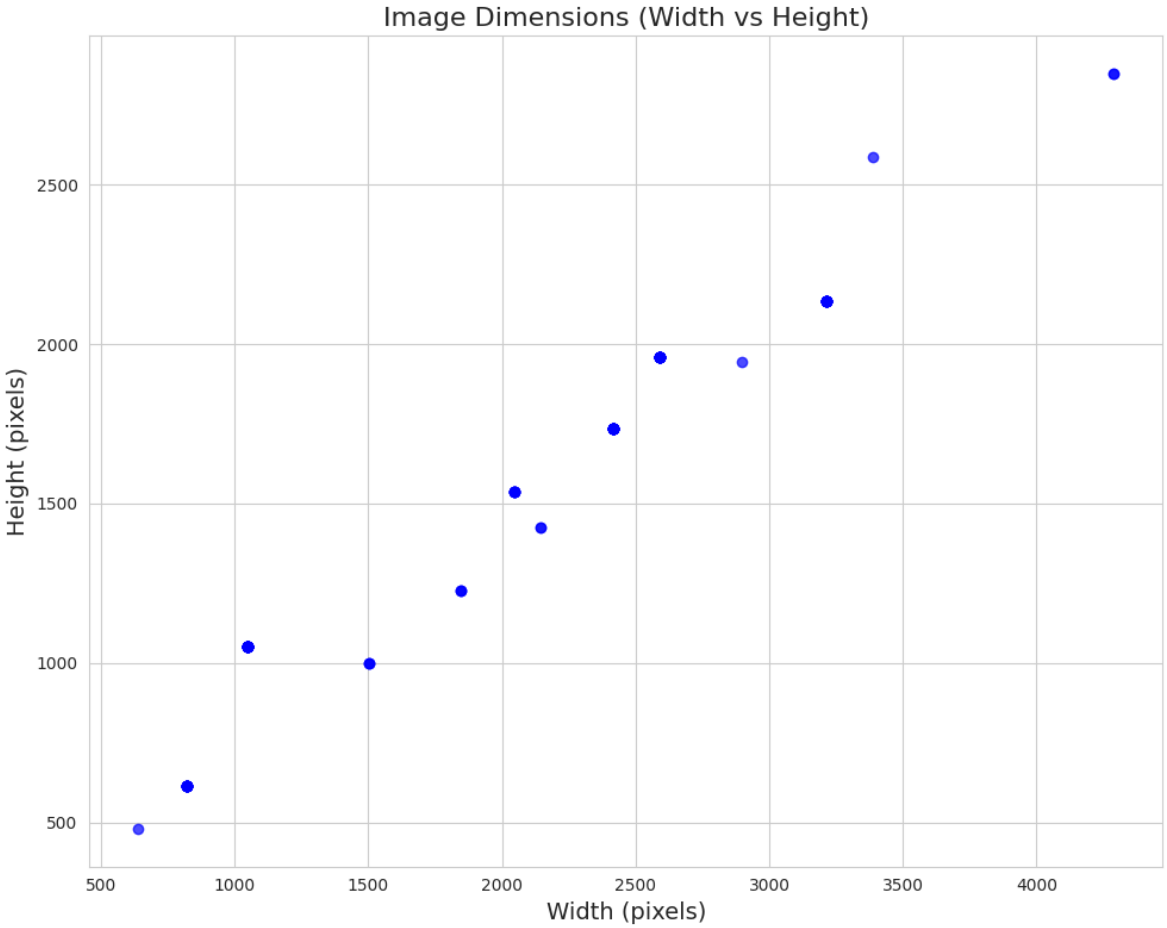


In [5]:
```python
# Function to load an image and return its dimensions
def get_image_dimensions(img_path):
    img = cv2.imread(img_path)
    return img.shape[:2]  # (height, width)

# Sample some images to check dimensions
sample_size = min(100, len(train_df))
sampled_ids = random_df = train_df.sample(sample_size)
dimensions = []

for idx, row in sampled_ids.iterrows():
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    dimensions.append(get_image_dimensions(img_path))

heights, widths = zip(*dimensions)

# Plot the distribution of image dimensions
plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
plt.hist(heights, bins=20, alpha=0.7)
plt.title('Distribution of Image Heights', fontsize=14)
plt.xlabel('Height (pixels)', fontsize=12)
plt.ylabel('Count', fontsize=12)
```

```
plt.subplot(1, 2, 2)
plt.hist(widths, bins=20, alpha=0.7)
plt.title('Distribution of Image Widths', fontsize=14)
plt.xlabel('Width (pixels)', fontsize=12)
plt.ylabel('Count', fontsize=12)

plt.tight_layout()
plt.show()

# Create a scatter plot of image dimensions
plt.figure(figsize=(10, 8))
plt.scatter(widths, heights, alpha=0.7, c='blue')
plt.title('Image Dimensions (Width vs Height)', fontsize=16)
plt.xlabel('Width (pixels)', fontsize=14)
plt.ylabel('Height (pixels)', fontsize=14)
plt.grid(True)
plt.tight_layout()
plt.show()

# Print dimension statistics
print(f"Mean image dimensions: {np.mean(heights):.1f} x {np.mean(widths):
print(f"Min image dimensions: {min(heights)} x {min(widths)}")
print(f"Max image dimensions: {max(heights)} x {max(widths)}")
```





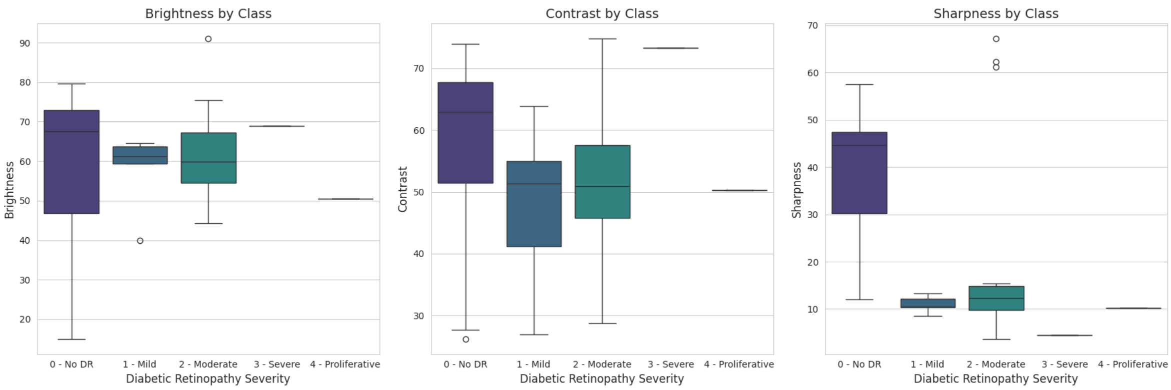


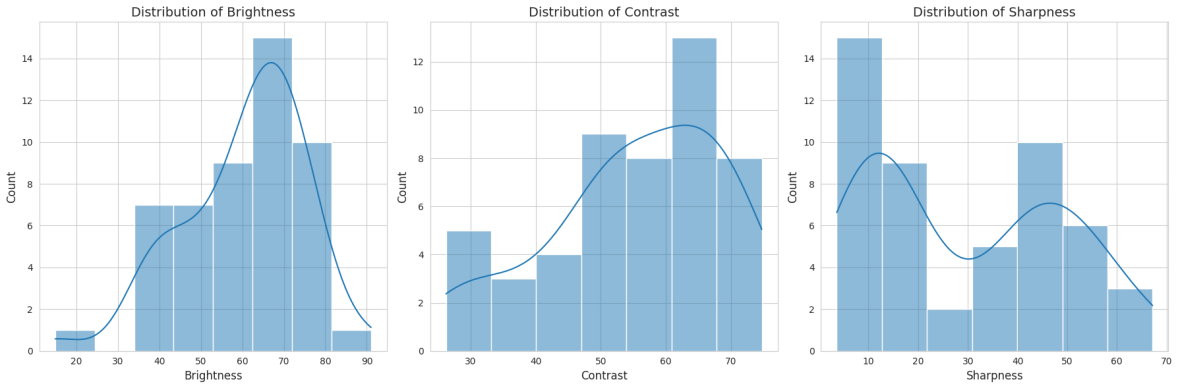

```
Mean image dimensions: 1475.8 x 1966.8
Min image dimensions: 480 x 640
Max image dimensions: 2848 x 4288
```

In [6]:
```python
def analyze_image_quality(img_path):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Calculate brightness (mean pixel value)
    brightness = np.mean(img)

    # Calculate contrast (standard deviation of pixel values)
    contrast = np.std(img)

    # Calculate sharpness (using Laplacian)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    sharpness = cv2.Laplacian(gray, cv2.CV_64F).var()

    return img, brightness, contrast, sharpness

# Sample images for quality analysis
sample_size = min(50, len(train_df))
sampled_quality = train_df.sample(sample_size)
quality_data = []

for idx, row in sampled_quality.iterrows():
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    _, brightness, contrast, sharpness = analyze_image_quality(img_path)
    quality_data.append({
        'id_code': row['id_code'],
        'diagnosis': row['diagnosis'],
        'brightness': brightness,
```

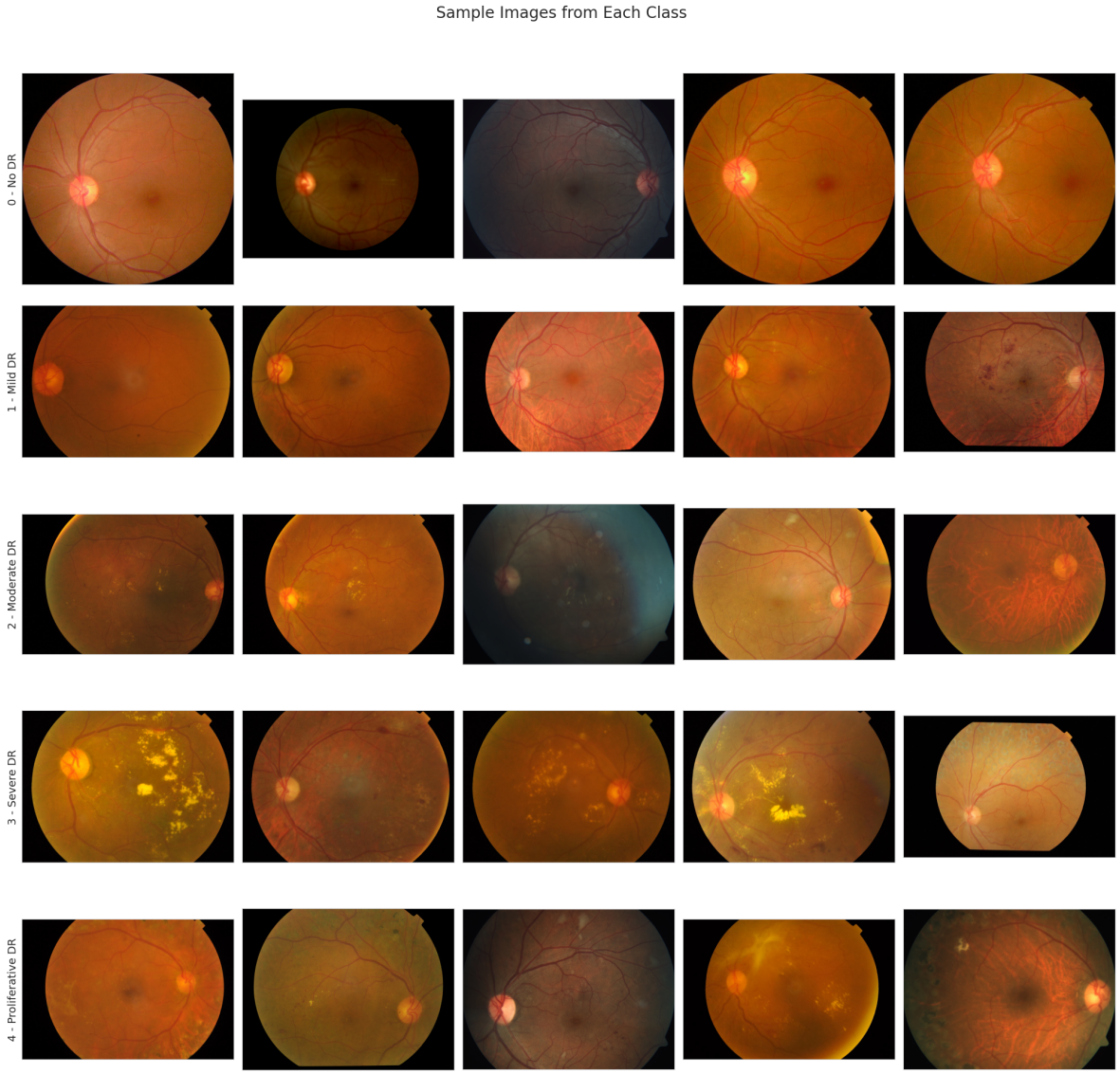
```
        'contrast': contrast,
        'sharpness': sharpness
    })

quality_df = pd.DataFrame(quality_data)

# Plot quality metrics by class
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

for i, metric in enumerate(['brightness', 'contrast', 'sharpness']):
    sns.boxplot(x='diagnosis', y=metric, data=quality_df, ax=axes[i], pal
    axes[i].set_title(f'{metric.capitalize()} by Class', fontsize=14)
    axes[i].set_xlabel('Diabetic Retinopathy Severity', fontsize=12)
    axes[i].set_ylabel(metric.capitalize(), fontsize=12)
    axes[i].set_xticklabels(['0 - No DR', '1 - Mild', '2 - Moderate', '3

plt.tight_layout()
plt.show()

# Plot distributions of quality metrics
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

for i, metric in enumerate(['brightness', 'contrast', 'sharpness']):
    sns.histplot(quality_df[metric], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {metric.capitalize()}', fontsize=
    axes[i].set_xlabel(metric.capitalize(), fontsize=12)
    axes[i].set_ylabel('Count', fontsize=12)

plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_639418/660419128.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(x='diagnosis', y=metric, data=quality_df, ax=axes[i], palett
e='viridis')
/tmp/ipykernel_639418/660419128.py:43: UserWarning: set_ticklabels() shoul
d only be used with a fixed number of ticks, i.e. after set_ticks() or usi
ng a FixedLocator.
  axes[i].set_xticklabels(['0 - No DR', '1 - Mild', '2 - Moderate', '3 - S
evere', '4 - Proliferative'])
/tmp/ipykernel_639418/660419128.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(x='diagnosis', y=metric, data=quality_df, ax=axes[i], palett
e='viridis')
/tmp/ipykernel_639418/660419128.py:43: UserWarning: set_ticklabels() shoul
d only be used with a fixed number of ticks, i.e. after set_ticks() or usi
ng a FixedLocator.
  axes[i].set_xticklabels(['0 - No DR', '1 - Mild', '2 - Moderate', '3 - S
evere', '4 - Proliferative'])
/tmp/ipykernel_639418/660419128.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(x='diagnosis', y=metric, data=quality_df, ax=axes[i], palett
e='viridis')
/tmp/ipykernel_639418/660419128.py:43: UserWarning: set_ticklabels() shoul
d only be used with a fixed number of ticks, i.e. after set_ticks() or usi
ng a FixedLocator.
  axes[i].set_xticklabels(['0 - No DR', '1 - Mild', '2 - Moderate', '3 - S
evere', '4 - Proliferative'])
```





In [7]:
```
# Function to display images from each class
def display_samples_from_each_class(df, img_dir, n_samples=5):
    # Dictionary to store images for each class
    class_images = {0: [], 1: [], 2: [], 3: [], 4: []}

    # Collect image paths for each class
    for idx, row in df.iterrows():
        class_label = row['diagnosis']
        img_path = os.path.join(img_dir, row['id_code'] + '.png')

        if len(class_images[class_label]) < n_samples:
            class_images[class_label].append(img_path)

    # Create a 5 x n_samples grid
    fig = plt.figure(figsize=(20, 4*5))
    gs = gridspec.GridSpec(5, n_samples)

    class_names = [
        '0 - No DR',
        '1 - Mild DR',
        '2 - Moderate DR',
        '3 - Severe DR',
        '4 - Proliferative DR'
    ]

    for class_label in range(5):
        for i, img_path in enumerate(class_images[class_label]):
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            ax = plt.subplot(gs[class_label, i])
            ax.imshow(img)

            if i == 0:
                ax.set_ylabel(class_names[class_label], fontsize=14)

            # Turn off axis ticks
            ax.set_xticks([])
            ax.set_yticks([])

    plt.suptitle('Sample Images from Each Class', fontsize=20)
    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust for title
    plt.show()

# Display sample images from each class
display_samples_from_each_class(train_df, TRAIN_IMAGES_DIR)
```

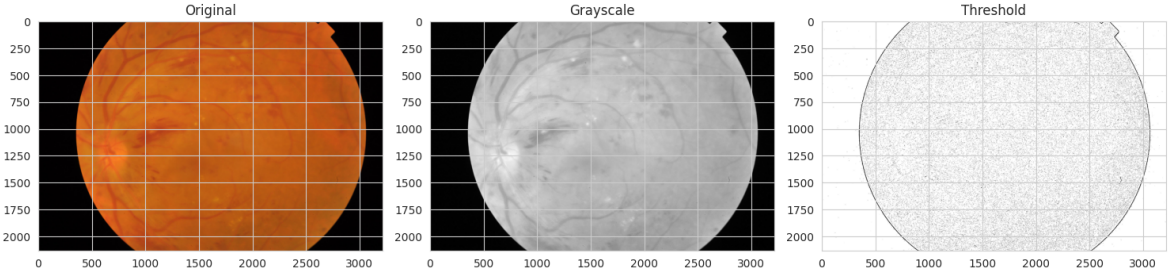


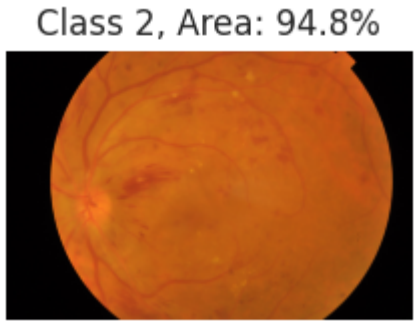
In [9]:
```
def analyze_effective_area(img_path):
    # Read image
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Try adaptive thresholding instead
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIA
                                   cv2.THRESH_BINARY, 11, 2)

    # Or try Otsu's thresholding
    # _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THR

    # Calculate percentage of non-black area
    total_pixels = thresh.shape[0] * thresh.shape[1]
    non_black_pixels = np.sum(thresh > 0)
    effective_area_percentage = (non_black_pixels / total_pixels) * 100

    # Display the steps for debugging
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title("Original")

    plt.subplot(1, 3, 2)
    plt.imshow(gray, cmap='gray')
```

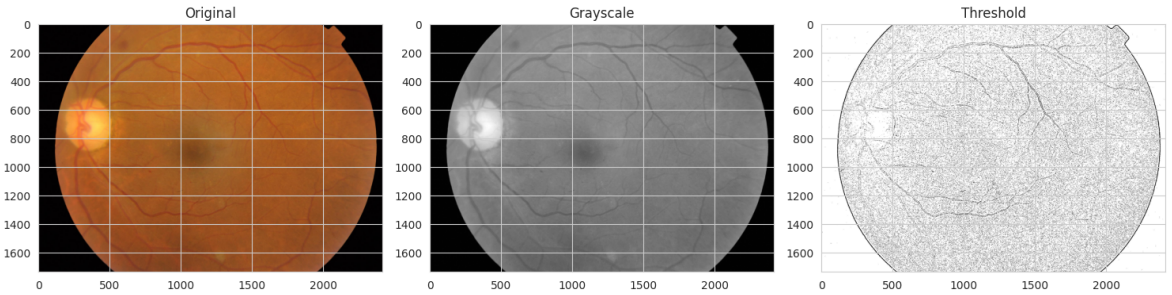
```
    plt.title("Grayscale")

    plt.subplot(1, 3, 3)
    plt.imshow(thresh, cmap='gray')
    plt.title("Threshold")

    plt.tight_layout()
    plt.show()

    return img, gray, thresh, effective_area_percentage

# Sample a few images for area analysis
sample_size = 12
sampled_area = train_df.sample(sample_size)

# Plot original images and their thresholded versions
plt.figure(figsize=(20, 5*3))
for i, (idx, row) in enumerate(sampled_area.iterrows()):
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    img, gray, thresh, area_pct = analyze_effective_area(img_path)

    plt.subplot(3, 4, i+1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"Class {row['diagnosis']}, Area: {area_pct:.1f}%")
    plt.axis('off')

plt.suptitle('Effective Area Analysis (Original Images)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust for title
plt.show()

# Now show the thresholded versions
plt.figure(figsize=(20, 5*3))
for i, (idx, row) in enumerate(sampled_area.iterrows()):
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    img, gray, thresh, area_pct = analyze_effective_area(img_path)

    plt.subplot(3, 4, i+1)
    plt.imshow(thresh, cmap='gray')
    plt.title(f"Threshold of Class {row['diagnosis']}")
    plt.axis('off')

plt.suptitle('Thresholded Images', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust for title
plt.show()

# Collect effective area percentages for different classes
area_data = []
for idx, row in train_df.sample(min(200, len(train_df))).iterrows():
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    _, _, _, area_pct = analyze_effective_area(img_path)
    area_data.append({
        'diagnosis': row['diagnosis'],
        'effective_area_pct': area_pct
    })

area_df = pd.DataFrame(area_data)

# Plot effective area by class
plt.figure(figsize=(12, 6))
sns.boxplot(x='diagnosis', y='effective_area_pct', data=area_df, palette=
```

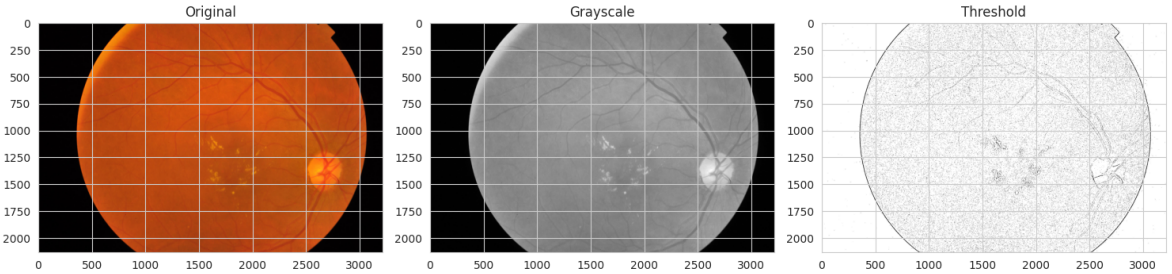
```
plt.title('Effective Image Area by Class', fontsize=16)
plt.xlabel('Diabetic Retinopathy Severity', fontsize=14)
plt.ylabel('Effective Area (%)', fontsize=14)
plt.xticks(range(5), ['0 - No DR', '1 - Mild', '2 - Moderate', '3 - Sever
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
<Figure size 2000x1500 with 0 Axes>
```

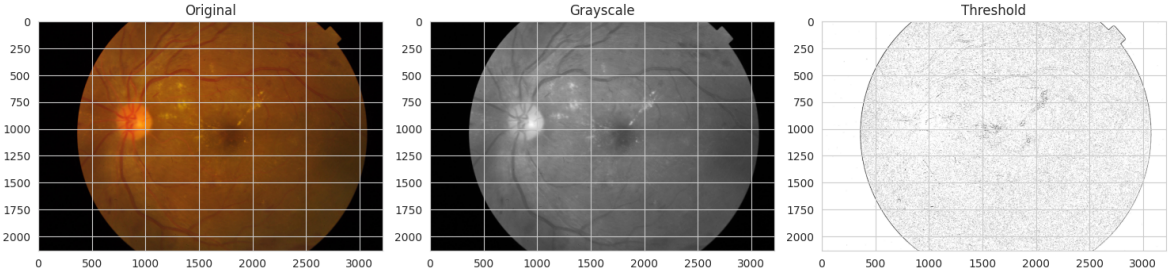




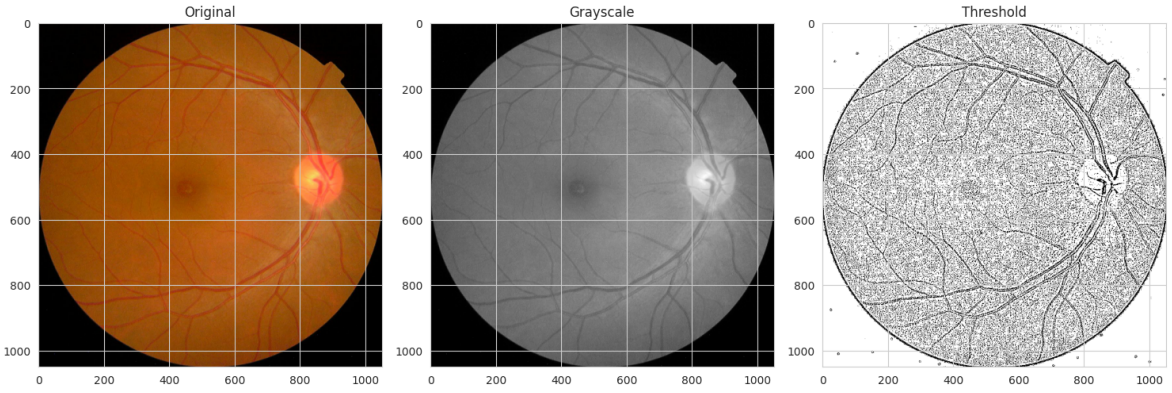




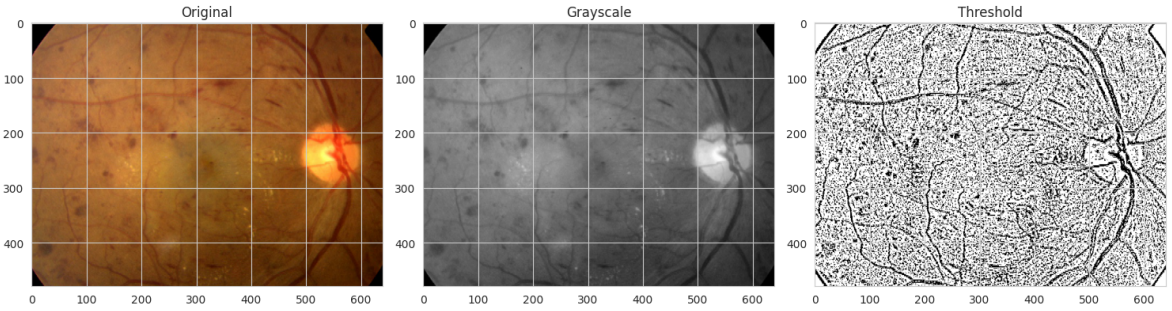

Class 2, Area: 95.1%

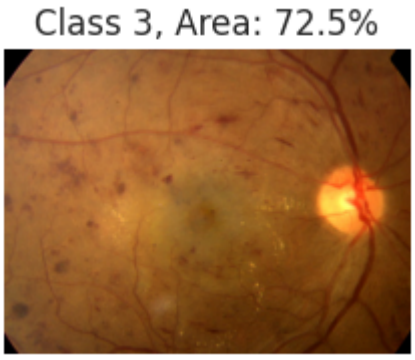

Original
Grayscale
Threshold

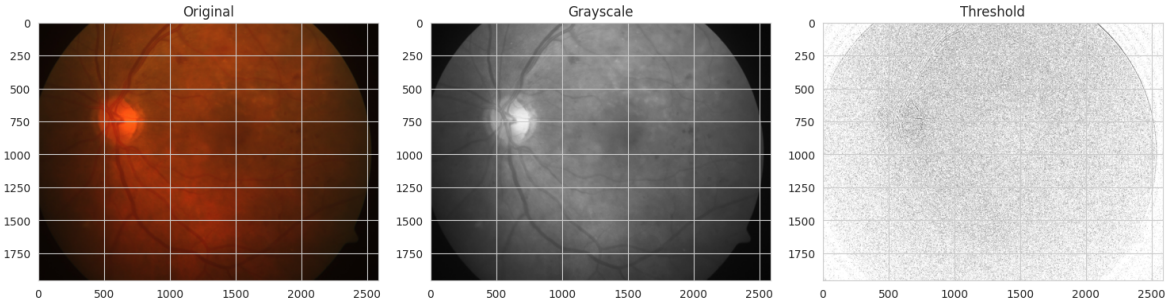

Class 3, Area: 94.8%


Original
Grayscale
Threshold

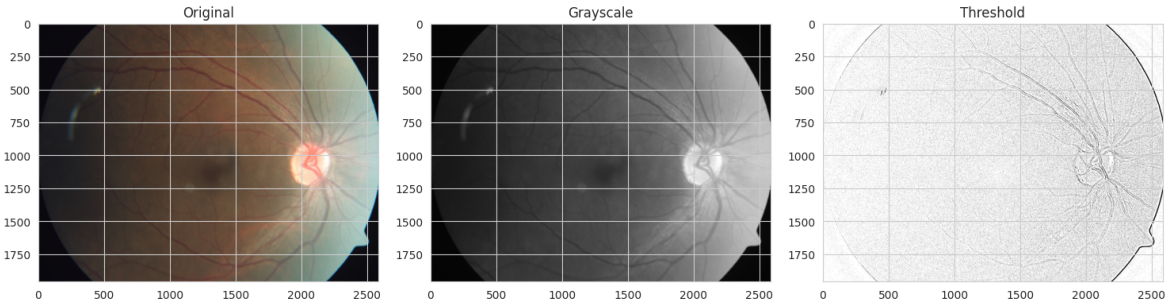

Class 0, Area: 79.8%


Original
Grayscale
Threshold

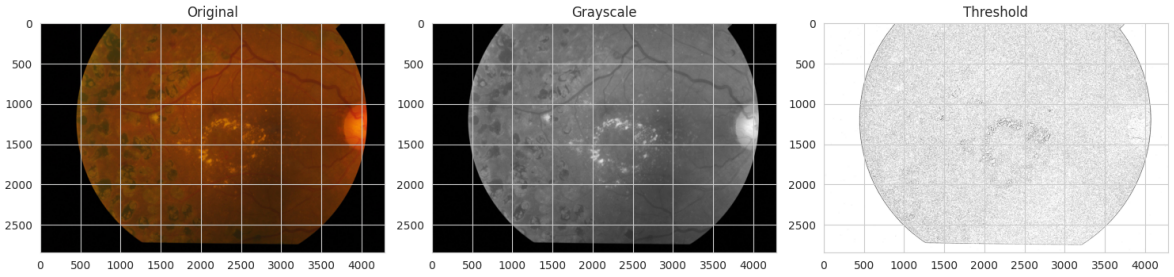

Class 3, Area: 72.5%

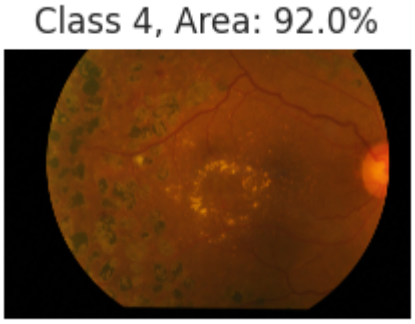

Original
Grayscale
Threshold

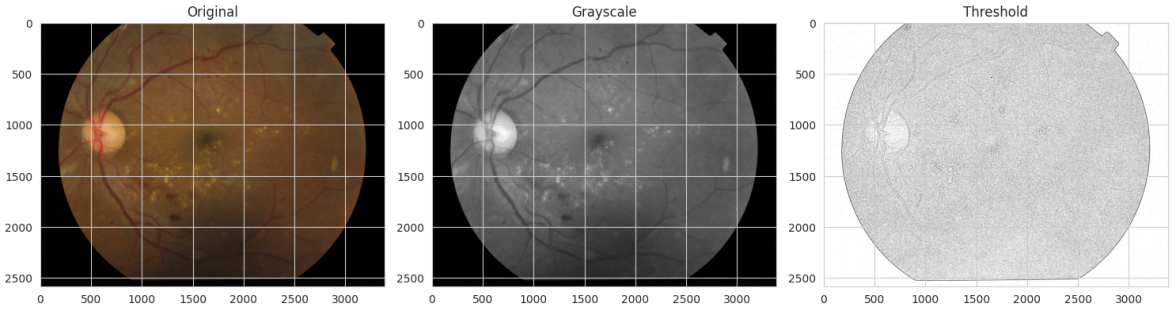

Class 1, Area: 88.5%


Original
Grayscale
Threshold

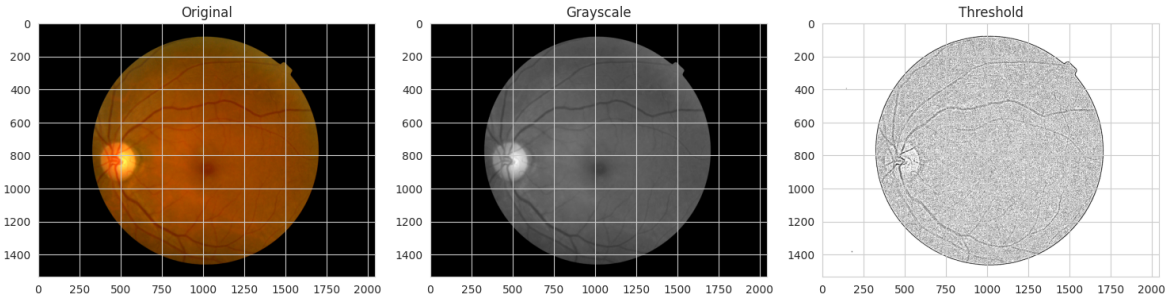

Class 0, Area: 91.8%


Original
Grayscale
Threshold

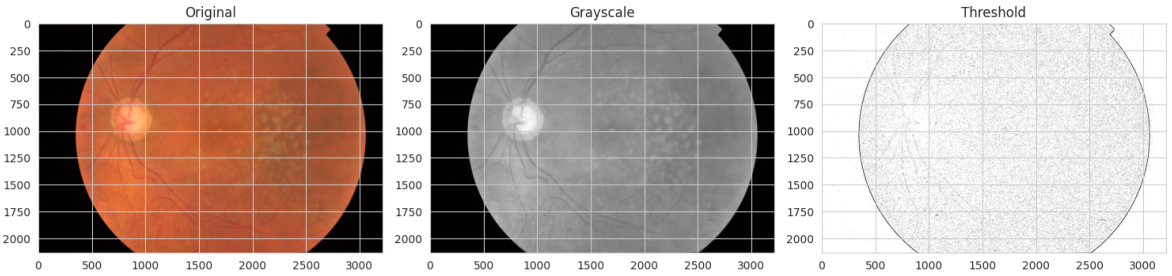

Class 4, Area: 92.0%


Original
Grayscale
Threshold


Class 3, Area: 86.4%

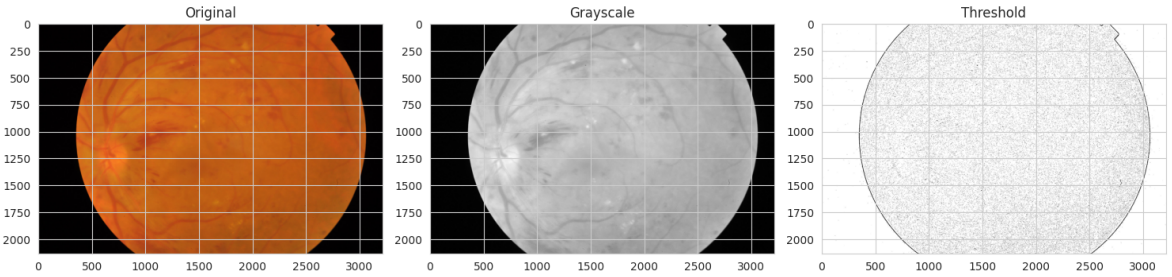

Original
Grayscale
Threshold

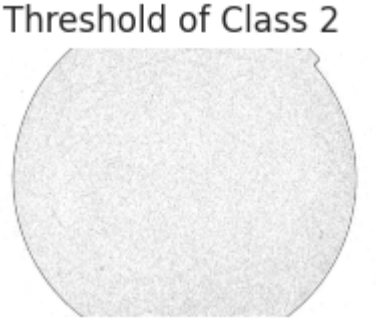

Class 0, Area: 88.7%

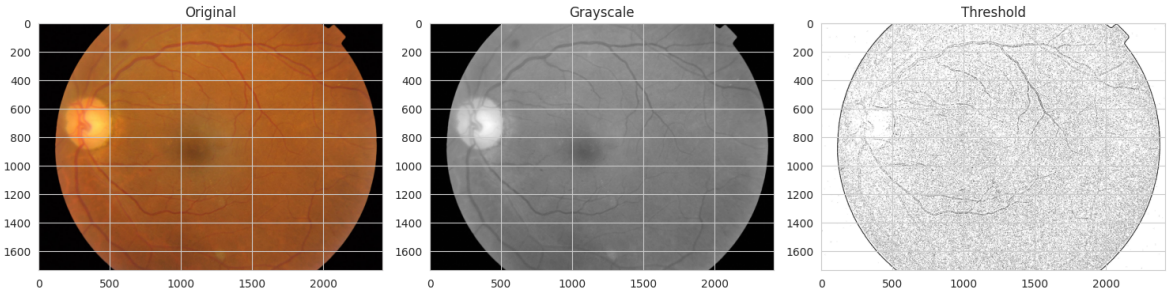

Original
Grayscale
Threshold

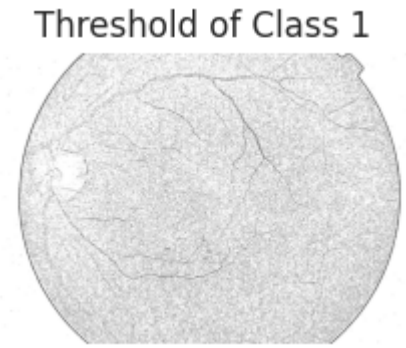
Effective Area Analysis (Original Images)

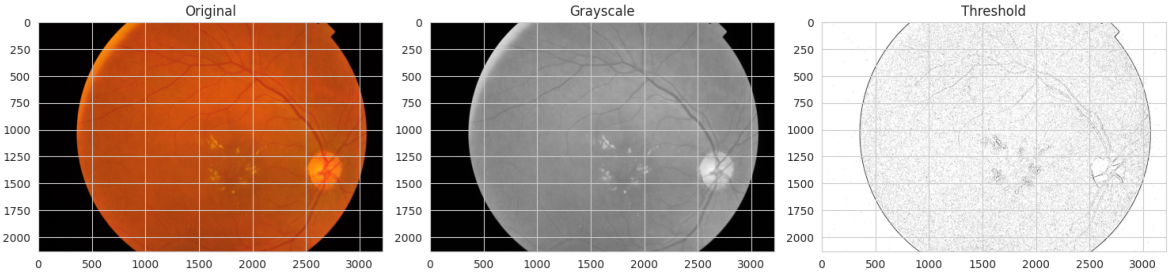
<Figure size 2000x1500 with 0 Axes>



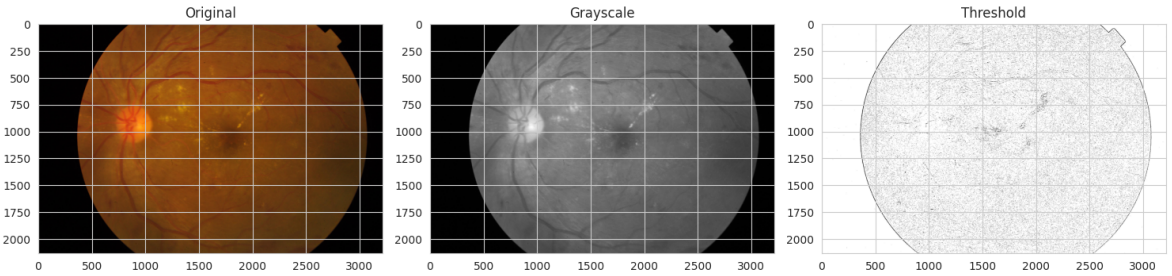



Original
Grayscale
Threshold

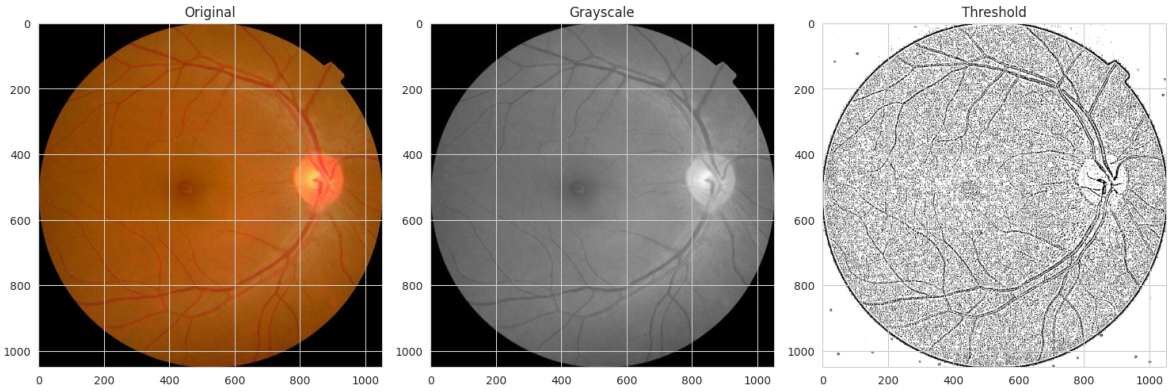

Threshold of Class 1

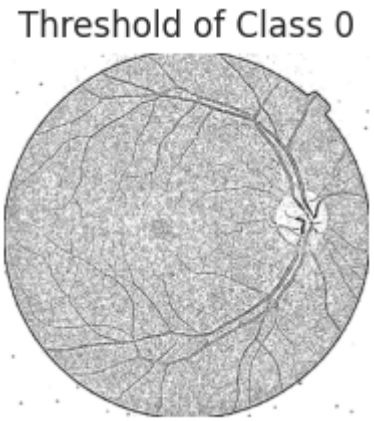

Original
Grayscale
Threshold

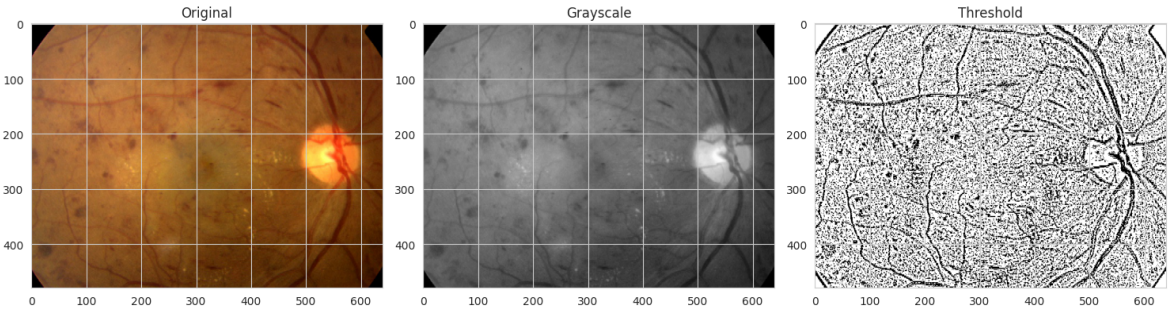

Threshold of Class 2

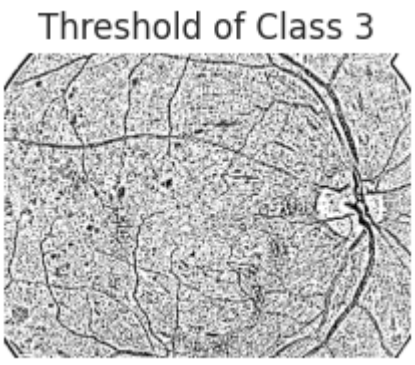

Original
Grayscale
Threshold

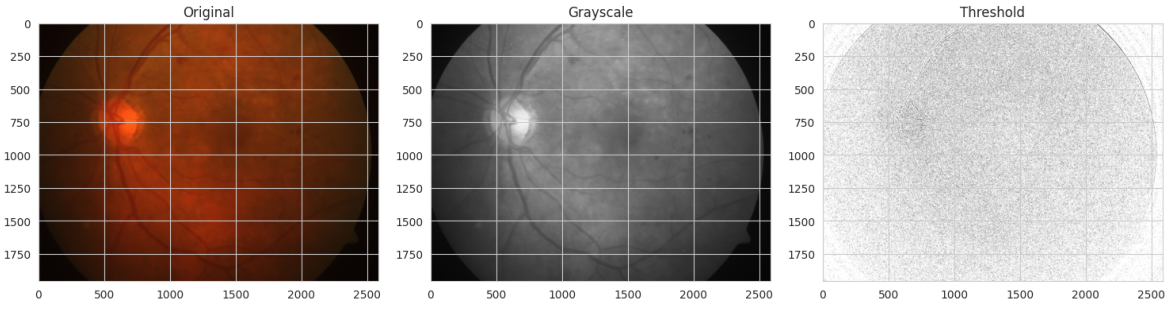

Threshold of Class 3


Original
Grayscale
Threshold

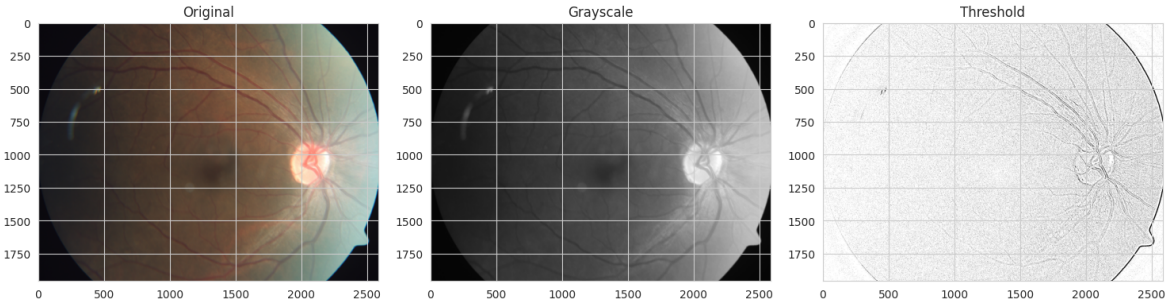

Threshold of Class 0

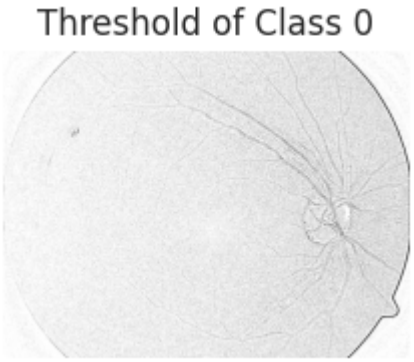

Original
Grayscale
Threshold

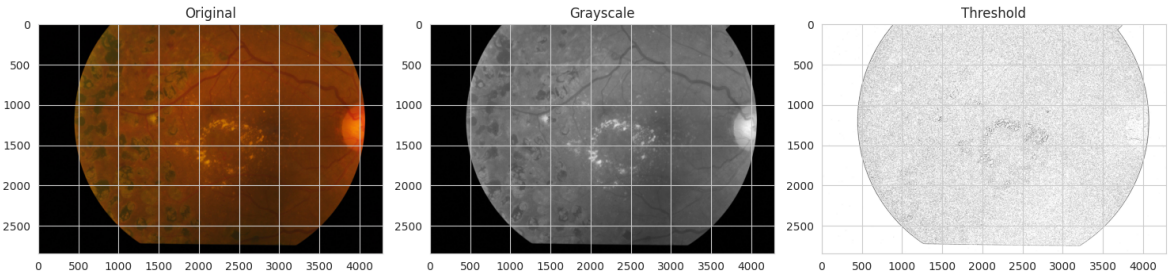

Threshold of Class 3


Original
Grayscale
Threshold

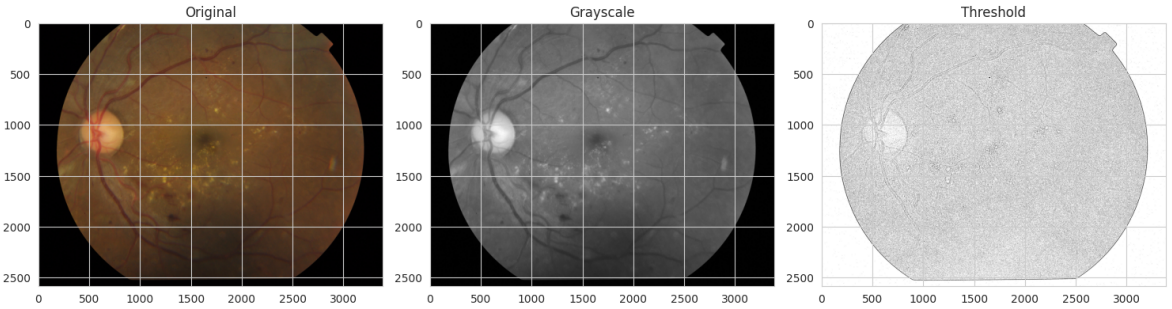

Threshold of Class 1

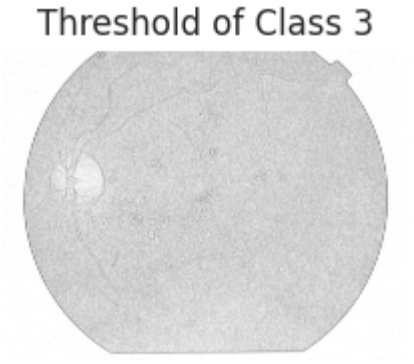

Original
Grayscale
Threshold

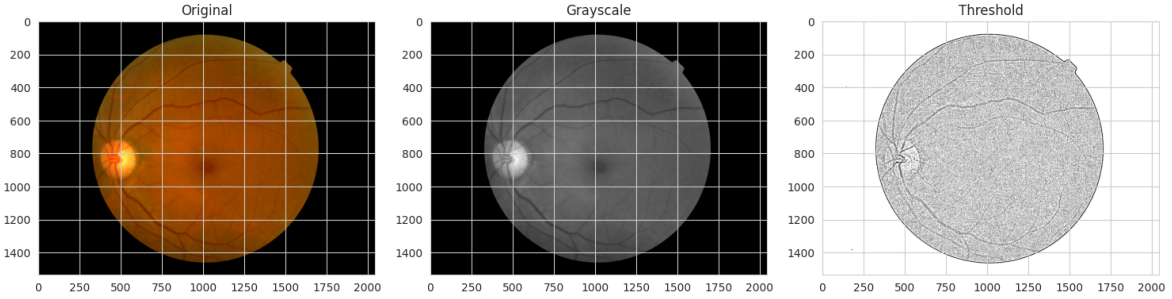

Threshold of Class 0

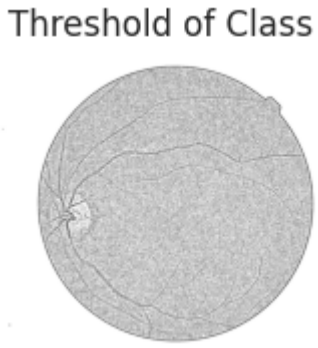

Original
Grayscale
Threshold

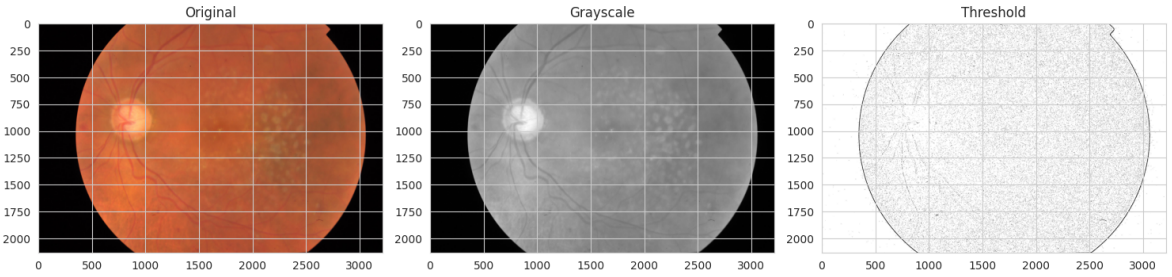

Threshold of Class 4


Original
Grayscale
Threshold


Threshold of Class 3


Original
Grayscale
Threshold


Threshold of Class 0


Original
Grayscale
Threshold