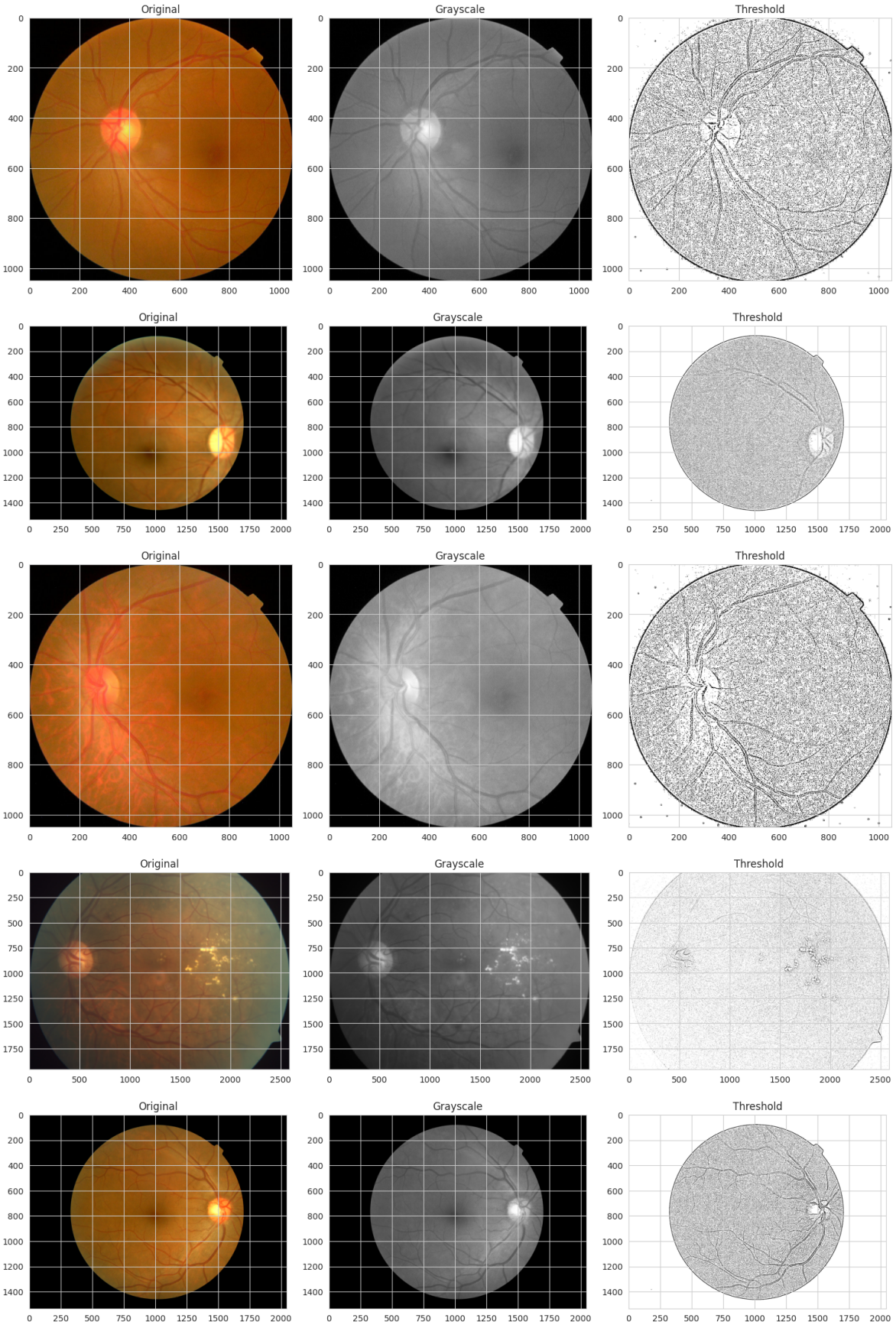
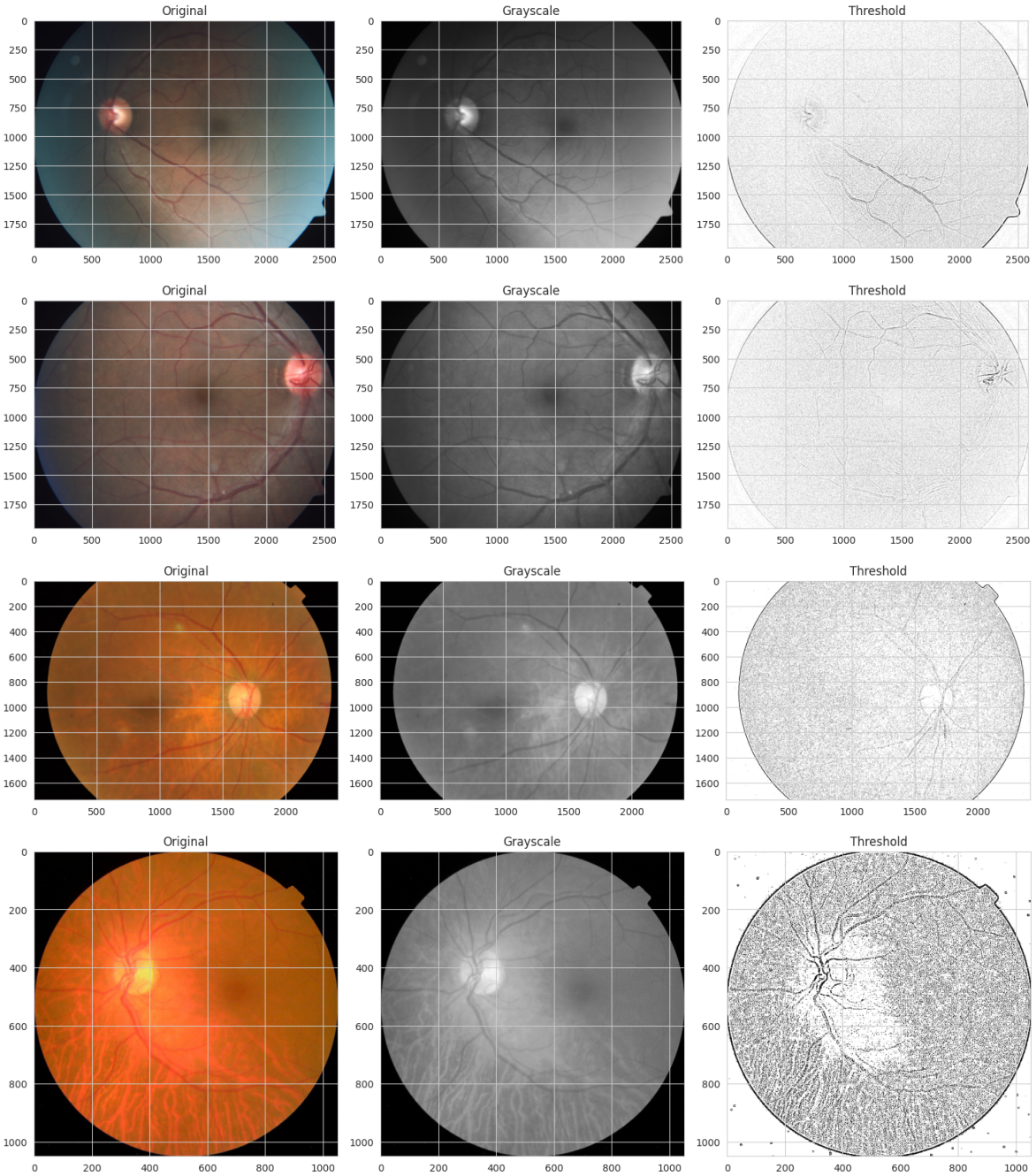
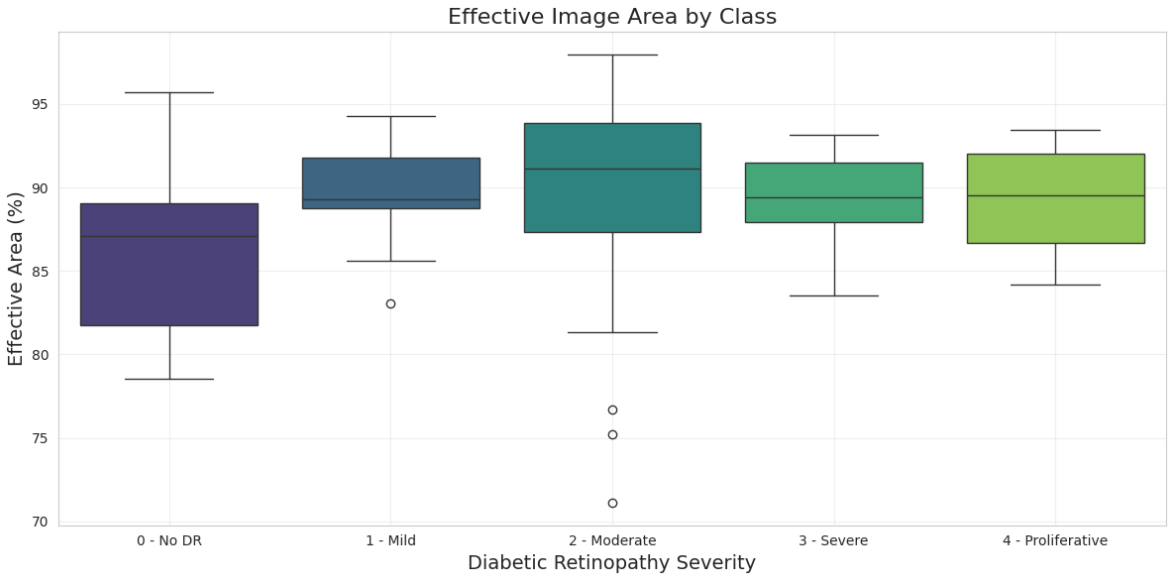

Original
Grayscale
Threshold
Original
Grayscale
Threshold
Original
Grayscale
Threshold
Original
Grayscale
Threshold
Original
Grayscale
Threshold




```
/tmp/ipykernel_639418/3283925463.py:85: 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='effective_area_pct', data=area_df, palette
='viridis')
```

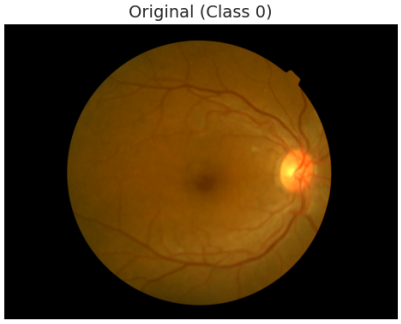


In [10]:
```
def apply_clahe(img):
    # Convert to LAB color space
    lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)

    # Apply CLAHE to L channel
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    lab[..., 0] = clahe.apply(lab[..., 0])

    # Convert back to RGB
    return cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)

# Sample images from each class for CLAHE comparison
selected_samples = []
for class_label in range(5):
    # Get 2 samples from each class
    class_samples = train_df[train_df['diagnosis'] == class_label].sample
    selected_samples.append(class_samples)

selected_df = pd.concat(selected_samples).reset_index(drop=True)

# Display original vs CLAHE enhanced images
plt.figure(figsize=(20, 4*len(selected_df)))
for i, (idx, row) in enumerate(selected_df.iterrows()):
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    enhanced = apply_clahe(img)

    # Original image
    plt.subplot(len(selected_df), 2, 2*i+1)
    plt.imshow(img)
    plt.title(f"Original (Class {row['diagnosis']})", fontsize=14)
    plt.axis('off')

    # Enhanced image
    plt.subplot(len(selected_df), 2, 2*i+2)
    plt.imshow(enhanced)
    plt.title(f"CLAHE Enhanced (Class {row['diagnosis']})", fontsize=14)
    plt.axis('off')

plt.suptitle('Comparison of Original and CLAHE Enhanced Images', fontsize
plt.tight_layout(rect=[0, 0, 1, 0.97])  # Adjust for title
```

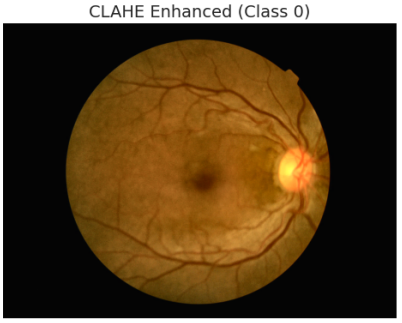
```
plt.show()
```

Comparison of Original and CLAHE Enhanced Images

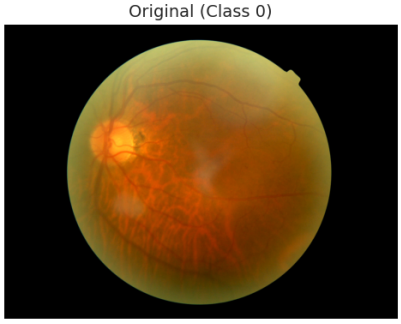
Original (Class 0)



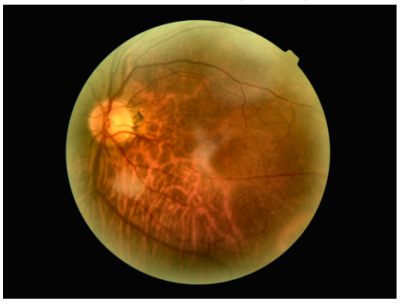
CLAHE Enhanced (Class 0)



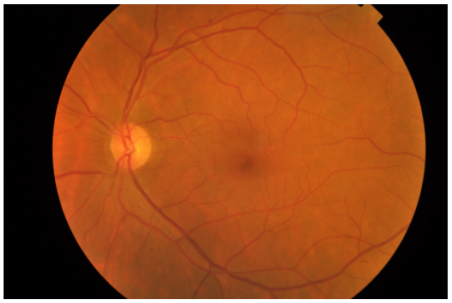
Original (Class 0)



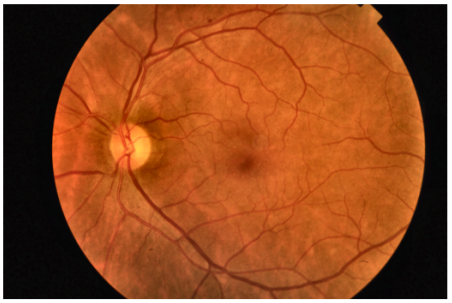
CLAHE Enhanced (Class 0)



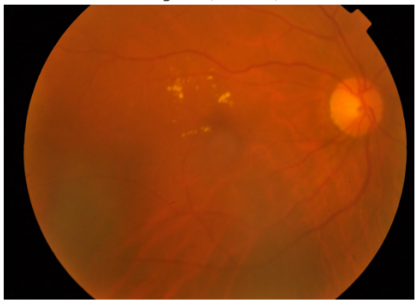
Original (Class 1)



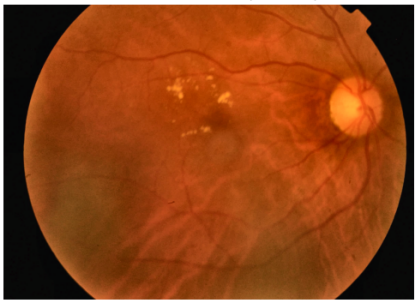
CLAHE Enhanced (Class 1)



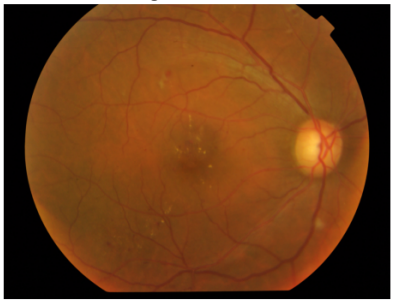
Original (Class 1)



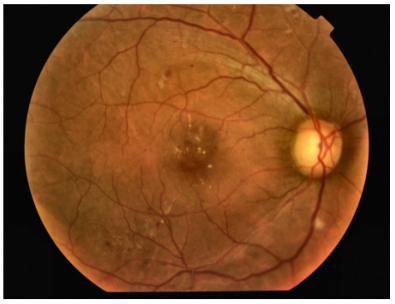
CLAHE Enhanced (Class 1)



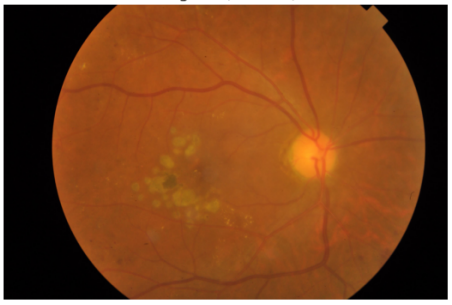
Original (Class 2)



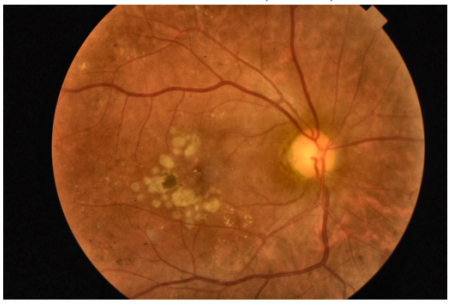
CLAHE Enhanced (Class 2)

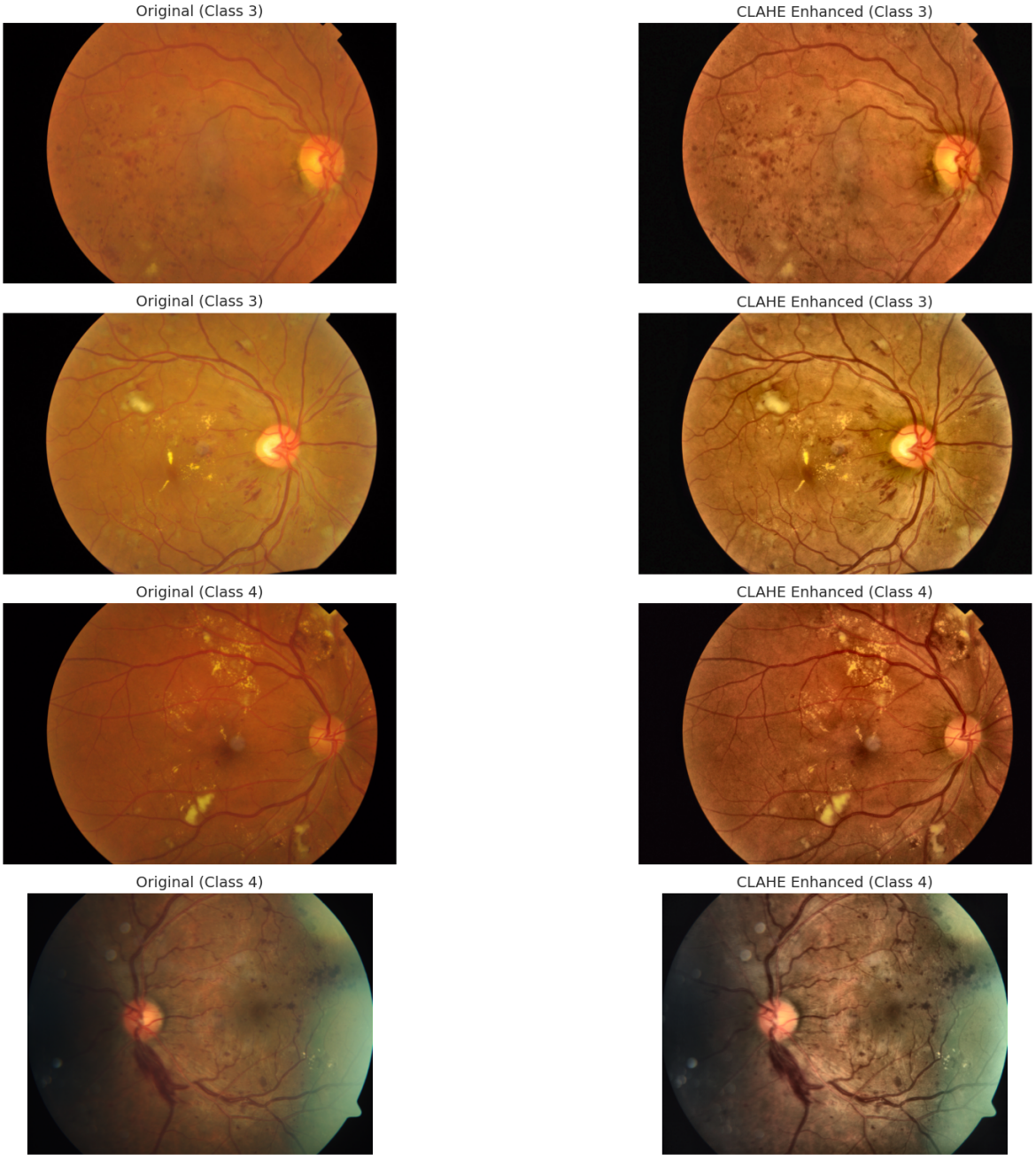


Original (Class 2)



CLAHE Enhanced (Class 2)

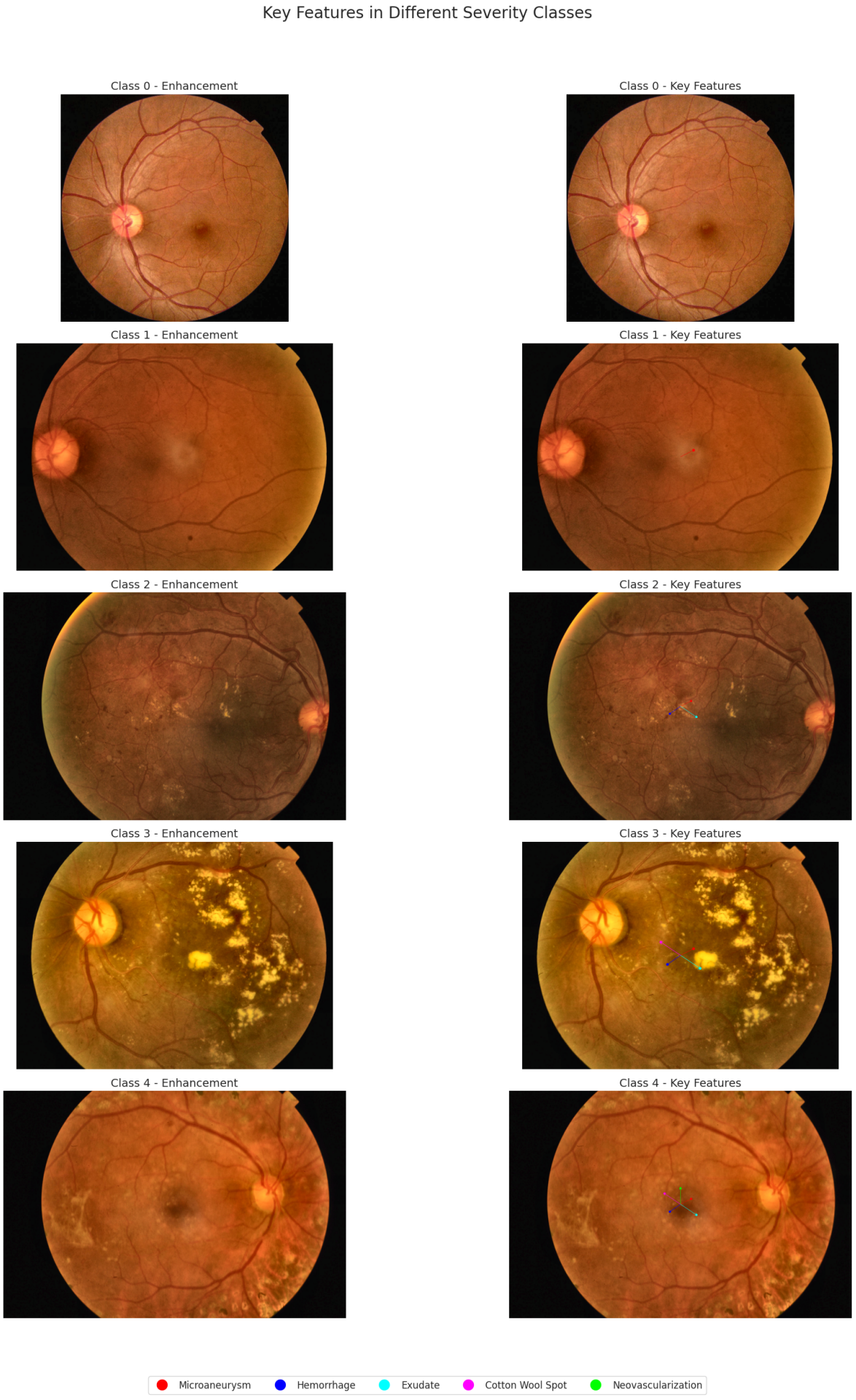




In [11]:
```python
def add_arrows_to_image(img, features):
    """Add arrows pointing to important features in the retinal image"""
    img_copy = img.copy()

    # Define arrow colors
    colors = {
        'microaneurysm': (255, 0, 0),     # Red
        'hemorrhage': (0, 0, 255),        # Blue
        'exudate': (0, 255, 255),         # Yellow
        'cotton_wool_spot': (255, 0, 255),# Magenta
        'neovascularization': (0, 255, 0) # Green
    }

    # Add arrows to the image
    for feature_type, points in features.items():
        color = colors.get(feature_type, (255, 255, 255))
        for point in points:
            x, y = point
            # Draw circle at feature point
            cv2.circle(img_copy, (x, y), 10, color, -1)
```

```
            # Draw arrow from center to feature
            center = (img.shape[1] // 2, img.shape[0] // 2)
            cv2.arrowedLine(img_copy, center, (x, y), color, 2)

    return img_copy

# Note: This is an example - you would need to manually identify features
# or work with an ophthalmologist for accurate annotations
# For this example, I'll use dummy coordinates

# Display annotated images with key features for educational purposes
# Select one representative image from each severity class
representative_samples = []
for class_label in range(5):
    if sum(train_df['diagnosis'] == class_label) > 0:
        representative_samples.append(train_df[train_df['diagnosis'] == c

# Create a figure with annotated features for explanation
plt.figure(figsize=(20, 5*5))

for i, row in enumerate(representative_samples):
    img_path = os.path.join(TRAIN_IMAGES_DIR, row['id_code'] + '.png')
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Apply CLAHE for better feature visibility
    enhanced = apply_clahe(img)

    # Display original image
    plt.subplot(5, 2, 2*i+1)
    plt.imshow(enhanced)
    plt.title(f"Class {row['diagnosis']} - Enhancement", fontsize=14)
    plt.axis('off')

    # Display with example annotations (you would need real annotations)
    plt.subplot(5, 2, 2*i+2)

    # Dummy feature locations - replace with real ones identified by expe
    features = {}
    if row['diagnosis'] >= 1:  # Mild DR and above
        features['microaneurysm'] = [(img.shape[1]//2 + 100, img.shape[0]
    if row['diagnosis'] >= 2:  # Moderate DR and above
        features['hemorrhage'] = [(img.shape[1]//2 - 100, img.shape[0]//2
        features['exudate'] = [(img.shape[1]//2 + 150, img.shape[0]//2 +
    if row['diagnosis'] >= 3:  # Severe DR and above
        features['cotton_wool_spot'] = [(img.shape[1]//2 - 150, img.shape
    if row['diagnosis'] >= 4:  # Proliferative DR
        features['neovascularization'] = [(img.shape[1]//2, img.shape[0]/

    # For class 0, no features to annotate
    annotated = add_arrows_to_image(enhanced, features)
    plt.imshow(annotated)
    plt.title(f"Class {row['diagnosis']} - Key Features", fontsize=14)
    plt.axis('off')

# Add a legend explaining the color coding
legend_elements = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=(255/255,
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=(0, 0, 25
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=(0, 255/2
```

```
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=(255/255,
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=(0, 255/2
]

plt.figlegend(handles=legend_elements, loc='lower center', ncol=5, fontsi
plt.suptitle('Key Features in Different Severity Classes', fontsize=20)
plt.tight_layout(rect=[0, 0.05, 1, 0.95])  # Adjust for title and legend
plt.show()

# Note: The annotations shown are illustrative. For a real project, these
# need to be identified by expert ophthalmologists or through consulted l
```



Summary of Data Exploration Findings

Class Distribution

- The dataset exhibits significant class imbalance with most images belonging to Class 0 (No DR) and fewer samples in the severe categories
- This imbalance will necessitate techniques like weighted loss functions, class-weighted sampling, or data augmentation

Image Characteristics

- Image dimensions vary across the dataset, requiring resizing during preprocessing
- Presence of black borders reduces the effective area in many images, suggesting cropping would be beneficial
- Brightness and contrast vary significantly, with some images being too dark or too bright

Quality Considerations

- CLAHE enhancement significantly improves visibility of retinal features
- Some images exhibit blurriness or artifacts that may impact model performance
- Images from more severe classes tend to have more visible features (hemorrhages, exudates)

Technical Challenges

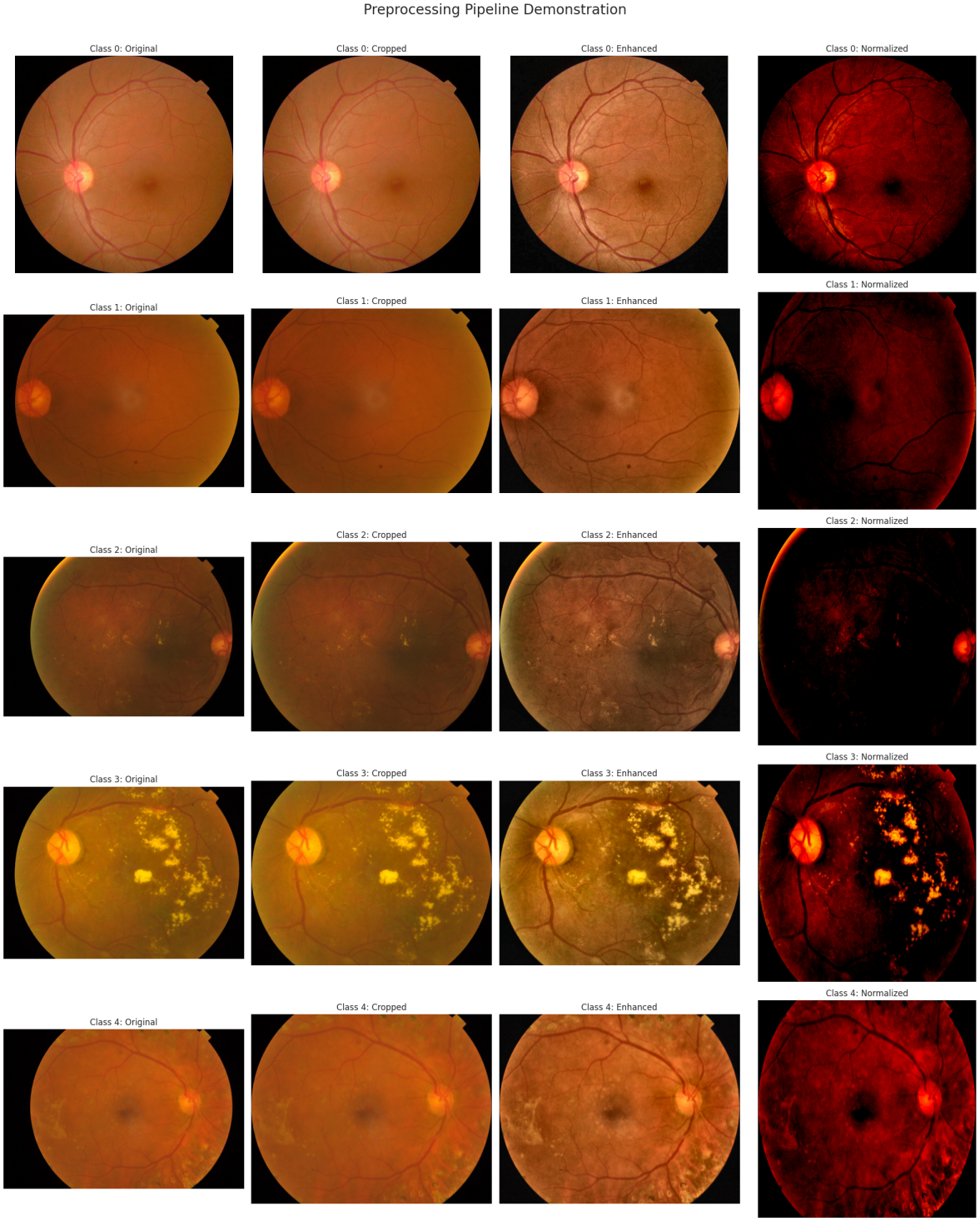
1. Preprocessing needs to address:
    - Resizing to a consistent dimension
    - Removing black borders to focus on the retinal area
    - Enhancing contrast using CLAHE
    - Normalizing pixel values
2. Training considerations:
    - Need to handle class imbalance
    - Use data augmentation to increase samples in minority classes
    - Consider transfer learning due to limited samples in some classes
3. Evaluation strategy:
    - Stratified cross-validation to maintain class representation
    - Use Quadratic Weighted Kappa as the primary metric as mentioned in the documentation
    - Monitor performance on minority classes separately

In [12]:
```
def preprocess_retinal_image(img_path, target_size=(512, 512)):
    """Full preprocessing pipeline for a retinal image"""
    # Read image
```

```
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Step 1: Crop black borders
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    _, thresh = cv2.threshold(gray, 10, 255, cv2.THRESH_BINARY)

    # Find contours to identify the retinal area
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_A

    if contours:
        # Find the largest contour (retinal area)
        largest_contour = max(contours, key=cv2.contourArea)

        # Get bounding rectangle
        x, y, w, h = cv2.boundingRect(largest_contour)

        # Crop the image to the bounding rectangle
        cropped = img[y:y+h, x:x+w]
    else:
        cropped = img

    # Step 2: Apply CLAHE enhancement
    enhanced = apply_clahe(cropped)

    # Step 3: Resize to target dimensions
    resized = cv2.resize(enhanced, target_size)

    # Step 4: Normalize pixel values to [-1, 1]
    normalized = (resized / 127.5) - 1.0

    return {
        'original': img,
        'cropped': cropped,
        'enhanced': enhanced,
        'resized': resized,
        'normalized': normalized
    }

# Demonstrate the preprocessing pipeline on samples from each class
plt.figure(figsize=(20, 5*5))

for class_label in range(5):
    # Get a sample from this class if available
    if sum(train_df['diagnosis'] == class_label) > 0:
        sample = train_df[train_df['diagnosis'] == class_label].iloc[0]
        img_path = os.path.join(TRAIN_IMAGES_DIR, sample['id_code'] + '.p

        # Apply preprocessing
        results = preprocess_retinal_image(img_path)

        # Display the preprocessing steps
        plt.subplot(5, 4, 4*class_label+1)
        plt.imshow(results['original'])
        plt.title(f"Class {class_label}: Original", fontsize=12)
        plt.axis('off')

        plt.subplot(5, 4, 4*class_label+2)
        plt.imshow(results['cropped'])
        plt.title(f"Class {class_label}: Cropped", fontsize=12)
```

```
        plt.axis('off')

        plt.subplot(5, 4, 4*class_label+3)
        plt.imshow(results['enhanced'])
        plt.title(f"Class {class_label}: Enhanced", fontsize=12)
        plt.axis('off')

        plt.subplot(5, 4, 4*class_label+4)
        plt.imshow(results['normalized'])
        plt.title(f"Class {class_label}: Normalized", fontsize=12)
        plt.axis('off')

plt.suptitle('Preprocessing Pipeline Demonstration', fontsize=20)
plt.tight_layout(rect=[0, 0, 1, 0.97])  # Adjust for title
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers). Got range [-1.0..1.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers). Got range [-1.0..1.0].
```


Preprocessing Pipeline Demonstration
Class 0: Original
Class 0: Cropped
Class 0: Enhanced
Class 0: Normalized
Class 1: Original
Class 1: Cropped
Class 1: Enhanced
Class 1: Normalized
Class 2: Original
Class 2: Cropped
Class 2: Enhanced
Class 2: Normalized
Class 3: Original
Class 3: Cropped
Class 3: Enhanced
Class 3: Normalized
Class 4: Original
Class 4: Cropped
Class 4: Enhanced
Class 4: Normalized