

In [11]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from skimage.feature import graycomatrix, graycoprops
import os

def load_dataset(images_dir):
    features_list = []
    categories_list = []

    for category_name in os.listdir(images_dir):
        category_path = os.path.join(images_dir, category_name)

        if os.path.isdir(category_path) and not category_name.startswith('.'):
            for img_file in os.listdir(category_path):
                if img_file.lower().endswith('.png', '.jpg', '.jpeg'):
                    img_path = os.path.join(category_path, img_file)

                    img_data = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
                    if img_data is not None:
                        feature_vector = compute_texture_descriptors(img_data)
                        features_list.append(feature_vector)
                        categories_list.append(category_name)

    return np.array(features_list), np.array(categories_list)

def analyze_image_intensity(gray_img, num_bins=256):
    intensity_dist = cv2.calcHist([gray_img], [0], None, [num_bins], [0, 256])
    intensity_dist = intensity_dist.flatten()
    intensity_dist = intensity_dist / intensity_dist.sum()

    intensity_values = np.arange(num_bins)
    stats_results = {}

    stats_results['average'] = np.sum(intensity_dist * intensity_values)

    variance = np.sum(intensity_dist * (intensity_values - stats_results['average'])**2)
    stats_results['deviation'] = np.sqrt(variance)

    if stats_results['deviation'] > 0:
        stats_results['asymmetry'] = np.sum(intensity_dist * ((intensity_values - stats_results['average']) / stats_results['deviation'])**3)
    else:
        stats_results['asymmetry'] = 0

    if stats_results['deviation'] > 0:
        stats_results['peakedness'] = np.sum(intensity_dist * ((intensity_values - stats_results['average']) / stats_results['deviation'])**4)
    else:
        stats_results['peakedness'] = -3

    non_zero_probs = intensity_dist[intensity_dist > 0]
    stats_results['randomness'] = -np.sum(non_zero_probs * np.log2(non_zero_probs))

    return stats_results
```

```

cumulative_dist = np.cumsum(intensity_dist)
stats_results['percentile_25'] = np.argmax(cumulative_dist >= 0.25)
stats_results['percentile_50'] = np.argmax(cumulative_dist >= 0.5)
stats_results['percentile_75'] = np.argmax(cumulative_dist >= 0.75)

occupied_bins = np.where(intensity_dist > 0)[0]
if len(occupied_bins) > 0:
    stats_results['lowest'] = occupied_bins[0]
    stats_results['highest'] = occupied_bins[-1]
else:
    stats_results['lowest'] = 0
    stats_results['highest'] = 255

return stats_results, intensity_dist

def compute_texture_descriptors(gray_img, num_bins=256):
    stats, dist = analyze_image_intensity(gray_img, num_bins)

    descriptors = [
        stats['average'],
        stats['deviation'],
        stats['asymmetry'],
        stats['peakedness'],
        stats['randomness'],
        stats['percentile_25'],
        stats['percentile_50'],
        stats['percentile_75'],
        stats['lowest'],
        stats['highest']
    ]

    return np.array(descriptors)

def visualize_texture_samples(images_dir, samples_count):
    categories = [f for f in os.listdir(images_dir)
                  if os.path.isdir(os.path.join(images_dir, f)) and not f.

    fig, axes = plt.subplots(samples_count, 3, figsize=(15, 5*samples_cou

    for idx in range(min(samples_count, len(categories))):
        category = categories[idx]
        category_path = os.path.join(images_dir, category)

        image_files = [f for f in os.listdir(category_path)
                      if f.lower().endswith('.png', '.jpg', '.jpeg')]

        if len(image_files) > 0:
            sample_path = os.path.join(category_path, image_files[0])
            sample_img = cv2.imread(sample_path, cv2.IMREAD_GRAYSCALE)

            if sample_img is not None:
                stats, intensity_plot = analyze_image_intensity(sample_im

                    axes[idx, 0].imshow(sample_img, cmap='gray')
                    axes[idx, 0].set_title(f'{category}')
                    axes[idx, 0].axis('off')

                    axes[idx, 1].bar(range(256), intensity_plot, alpha=0.7, c
                    axes[idx, 1].set_title('Интенсивность пикселей')

```

```

        stats_text = (
            f"Среднее: {stats['average']:.2f}\n"
            f"Отклонение: {stats['deviation']:.2f}\n"
            f"Асимметрия: {stats['asymmetry']:.2f}\n"
            f"Эксцесс: {stats['peakedness']:.2f}\n"
            f"Энтропия: {stats['randomness']:.2f}\n"
            f"Диапазон: {stats['lowest']}/{stats['highest']}\""
        )

        axes[idx, 2].text(0.1, 0.9, stats_text, transform=axes[idx].transAxes,
                           fontsize=10, verticalalignment='top')
        axes[idx, 2].axis('off')

plt.tight_layout()
plt.show()

def calculate_laws_features(img_data, normalize=True):
    L5 = np.array([1, 4, 6, 4, 1])
    E5 = np.array([-1, -2, 0, 2, 1])
    S5 = np.array([-1, 0, 2, 0, -1])
    W5 = np.array([-1, 2, 0, -2, 1])
    R5 = np.array([1, -4, 6, -4, 1])

    kernels_1d = [L5, E5, S5, W5, R5]
    kernel_labels = ['L5', 'E5', 'S5', 'W5', 'R5']

    filters_2d = []
    filter_labels = []

    for i, kernel_x in enumerate(kernels_1d):
        for j, kernel_y in enumerate(kernels_1d):
            filter_kernel = np.outer(kernel_x, kernel_y)
            filters_2d.append(filter_kernel)
            filter_labels.append(f"{kernel_labels[i]},{kernel_labels[j]}")

    feature_vector = []
    feature_labels = []

    for kernel, label in zip(filters_2d, filter_labels):
        filtered_image = cv2.filter2D(img_data.astype(np.float32), -1, kernel)
        texture_energy = np.mean(filtered_image ** 2)
        feature_vector.append(texture_energy)
        feature_labels.append(f"laws_{label}")

    feature_vector = np.array(feature_vector)

    if normalize and np.sum(feature_vector) > 0:
        feature_vector = feature_vector / np.sum(feature_vector)

    return feature_vector, feature_labels

def calculate_glcm_descriptors(img_data, distances=[1], angles=[0, np.pi/4],
                               img_data = (img_data // 8).astype(np.uint8))

    glcm_matrix = graycomatrix(img_data, distances=distances, angles=angles,
                               levels=32, symmetric=True, normed=True)

    descriptors = []
    properties = ['contrast', 'dissimilarity', 'homogeneity', 'energy', ''

```

```
for prop in properties:
    descriptor_value = np.mean(grycoprops(glcm_matrix, prop))
    descriptors.append(descriptor_value)

return np.array(descriptors)

images_directory = "./KTH_TIPS"
features_data, target_labels = load_dataset(images_directory)

print(f"Загружено данных: {features_data.shape}")
print(f"Уникальных категорий: {len(np.unique(target_labels))}")

visualize_texture_samples(images_directory, samples_count=2)

image_files_list = []
for category_dir in os.listdir(images_directory):
    full_category_path = os.path.join(images_directory, category_dir)
    if os.path.isdir(full_category_path):
        for filename in os.listdir(full_category_path):
            if filename.lower().endswith('.png', '.jpg', '.jpeg'):
                image_files_list.append(os.path.join(full_category_path,
                                         filename))

print(f"Обнаружено изображений: {len(image_files_list)}")

histogram_features = []
laws_features = []
glcm_features = []
image_labels = []

for idx, file_path in enumerate(image_files_list):
    if idx % 50 == 0:
        print(f"Обработка {idx}/{len(image_files_list)}")

    img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (128, 128), interpolation=cv2.INTER_AREA)

    hist_feats = compute_texture_descriptors(img)
    laws_feats, _ = calculate_laws_features(img)
    glcm_feats = calculate_glcm_descriptors(img)

    histogram_features.append(hist_feats)
    laws_features.append(laws_feats)
    glcm_features.append(glcm_feats)

    category = os.path.basename(os.path.dirname(file_path))
    image_labels.append(category)

histogram_features = np.array(histogram_features)
laws_features = np.array(laws_features)
glcm_features = np.array(glcm_features)
image_labels = np.array(image_labels)

print(f"Размерность признаков: {histogram_features.shape}")

label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(image_labels)

feature_sets = {
    'Гистограмма': histogram_features,
    'Law': laws_features,
```

```

    'GLCM': glcm_features
}

classifiers = {
    'kNN': KNeighborsClassifier(n_neighbors=3, n_jobs=-1),
    'SVM': SVC(kernel='rbf', random_state=42),
    'Дерево': DecisionTreeClassifier(random_state=42, max_depth=10)
}

trained_classifiers = {}

for feature_name, feature_matrix in feature_sets.items():
    print(f"\n{feature_name.upper()}")

    X_tr, X_te, y_tr, y_te = train_test_split(
        feature_matrix, encoded_labels, test_size=0.25, random_state=42,
    )

    normalizer = StandardScaler()
    X_tr_norm = normalizer.fit_transform(X_tr)
    X_te_norm = normalizer.transform(X_te)

    current_models = {}

    for clf_name, clf_template in classifiers.items():
        if clf_name == 'kNN':
            model = KNeighborsClassifier(n_neighbors=3, n_jobs=-1)
        elif clf_name == 'SVM':
            model = SVC(kernel='rbf', random_state=42)
        elif clf_name == 'Дерево':
            model = DecisionTreeClassifier(random_state=42, max_depth=10)

        if clf_name == 'SVM':
            model.fit(X_tr_norm, y_tr)
            predictions = model.predict(X_te_norm)
            current_models[clf_name] = (model, normalizer)
        else:
            model.fit(X_tr, y_tr)
            predictions = model.predict(X_te)
            current_models[clf_name] = model

        print(f"{clf_name}")
        print(classification_report(y_te, predictions, target_names=label))
        print()

    trained_classifiers[feature_name] = current_models

best_models = {
    'Hist_kNN': trained_classifiers['Гистограмма']['kNN'],
    'Laws_Tree': trained_classifiers['Laws']['Дерево'],
    'Laws_SVM': trained_classifiers['Laws']['SVM']
}

def perform_texture_segmentation(model_data, model_id, input_image_path,
                                 target_image = cv2.imread(input_image_path, cv2.IMREAD_GRAYSCALE):
    if target_image is None:
        return None

    height, width = target_image.shape
    segmentation_result = np.zeros((height, width), dtype=np.uint8)

```

```

all_predictions = []
patch_positions = []

for y in range(0, height - window_size + 1, stride):
    for x in range(0, width - window_size + 1, stride):
        image_patch = target_image[y:y+window_size, x:x+window_size]

        if 'Hist' in model_id:
            patch_features = compute_texture_descriptors(image_patch)
        elif 'Laws' in model_id:
            patch_features, _ = calculate_laws_features(image_patch)
        elif 'GLCM' in model_id:
            patch_features = calculate_glc当地符s_descriptors(image_patch)

        if isinstance(model_data, tuple):
            classifier, feature_scaler = model_data
            scaled_features = feature_scaler.transform(patch_features)
            predicted_class = classifier.predict(scaled_features)[0]
        else:
            classifier = model_data
            predicted_class = classifier.predict(patch_features.reshape(1, -1))

        segmentation_result[y:y+window_size, x:x+window_size] = predicted_class
        all_predictions.append(predicted_class)
        patch_positions.append((y, x))

unique_classes, class_counts = np.unique(all_predictions, return_counts=True)
dominant_classes = unique_classes[np.argsort(class_counts)[-num_classes:]]
filtered_segmentation = np.zeros_like(segmentation_result)

for new_idx, original_class in enumerate(dominant_classes):
    filtered_segmentation[segmentation_result == original_class] = new_idx

remaining_pixels = ~np.isin(segmentation_result, dominant_classes)
if np.any(remaining_pixels):
    filtered_segmentation[remaining_pixels] = 1

print(f"Классы после фильтрации: {np.unique(filtered_segmentation)}")

return filtered_segmentation

target_image_path = "./tests/xleb.jpg"
input_image = cv2.imread(target_image_path, cv2.IMREAD_GRAYSCALE)
reference_mask = cv2.imread("./tests/xleb_mask.jpg", cv2.IMREAD_GRAYSCALE)

print(f"Изображение загружено: {input_image is not None}")
print(f"Маска загружена: {reference_mask is not None}")

if input_image is not None:
    segmentation_results = []
    model_identifiers = []

    for model_id, model_data in best_models.items():
        print(f"    Модель: {model_id}")

        segmentation_map = perform_texture_segmentation(model_data, model_id,
                                                       window_size=16, stride=8)
        segmentation_results.append(segmentation_map)
        model_identifiers.append(model_id)

    segmentation_map = np.argmax(np.array(segmentation_results), axis=0)
    segmentation_map[reference_mask == 0] = 0
    segmentation_map[reference_mask == 255] = 1
    segmentation_map[reference_mask == 128] = 2
    segmentation_map[reference_mask == 64] = 3
    segmentation_map[reference_mask == 32] = 4
    segmentation_map[reference_mask == 16] = 5
    segmentation_map[reference_mask == 8] = 6
    segmentation_map[reference_mask == 4] = 7
    segmentation_map[reference_mask == 2] = 8
    segmentation_map[reference_mask == 1] = 9
    segmentation_map[reference_mask == 0.5] = 10
    segmentation_map[reference_mask == 0.25] = 11
    segmentation_map[reference_mask == 0.125] = 12
    segmentation_map[reference_map == 0.0625] = 13
    segmentation_map[reference_map == 0.03125] = 14
    segmentation_map[reference_map == 0.015625] = 15
    segmentation_map[reference_map == 0.0078125] = 16
    segmentation_map[reference_map == 0.00390625] = 17
    segmentation_map[reference_map == 0.001953125] = 18
    segmentation_map[reference_map == 0.0009765625] = 19
    segmentation_map[reference_map == 0.00048828125] = 20
    segmentation_map[reference_map == 0.000244140625] = 21
    segmentation_map[reference_map == 0.0001220703125] = 22
    segmentation_map[reference_map == 0.00006103515625] = 23
    segmentation_map[reference_map == 0.000030517578125] = 24
    segmentation_map[reference_map == 0.0000152587890625] = 25
    segmentation_map[reference_map == 0.00000762939453125] = 26
    segmentation_map[reference_map == 0.000003814697265625] = 27
    segmentation_map[reference_map == 0.0000019073486328125] = 28
    segmentation_map[reference_map == 0.00000095367431640625] = 29
    segmentation_map[reference_map == 0.000000476837158203125] = 30
    segmentation_map[reference_map == 0.0000002384185791015625] = 31
    segmentation_map[reference_map == 0.00000012020928955078125] = 32
    segmentation_map[reference_map == 0.000000060104644775390625] = 33
    segmentation_map[reference_map == 0.0000000300523223876953125] = 34
    segmentation_map[reference_map == 0.00000001502616119384765625] = 35
    segmentation_map[reference_map == 0.000000007513080596923828125] = 36
    segmentation_map[reference_map == 0.0000000037565402984619140625] = 37
    segmentation_map[reference_map == 0.00000000187827014923095703125] = 38
    segmentation_map[reference_map == 0.000000000939135074615478515625] = 39
    segmentation_map[reference_map == 0.0000000004695675373077392578125] = 40
    segmentation_map[reference_map == 0.00000000023478376865386962890625] = 41
    segmentation_map[reference_map == 0.000000000117391884326934814453125] = 42
    segmentation_map[reference_map == 0.0000000000586959421634674072265625] = 43
    segmentation_map[reference_map == 0.00000000002934797108173370361328125] = 44
    segmentation_map[reference_map == 0.0000000000146739855408668518065625] = 45
    segmentation_map[reference_map == 0.000000000007336992770433475903125] = 46
    segmentation_map[reference_map == 0.0000000000036684963852167379515625] = 47
    segmentation_map[reference_map == 0.00000000000183424819260836897578125] = 48
    segmentation_map[reference_map == 0.000000000000917124096304184487890625] = 49
    segmentation_map[reference_map == 0.0000000000004585620481520922439453125] = 50
    segmentation_map[reference_map == 0.00000000000022928102407604612197265625] = 51
    segmentation_map[reference_map == 0.0000000000001146405120380230609865625] = 52
    segmentation_map[reference_map == 0.00000000000005732025601901153049303125] = 53
    segmentation_map[reference_map == 0.00000000000002866012800950576524652344] = 54
    segmentation_map[reference_map == 0.00000000000001433006400475288262326172] = 55
    segmentation_map[reference_map == 0.00000000000000716503200237644131163086] = 56
    segmentation_map[reference_map == 0.00000000000000358251600118822065581543] = 57
    segmentation_map[reference_map == 0.00000000000000179125800059411032790771] = 58
    segmentation_map[reference_map == 0.00000000000000089562900029720516395385] = 59
    segmentation_map[reference_map == 0.00000000000000044781450014860258197947] = 60
    segmentation_map[reference_map == 0.00000000000000022390725007430129098723] = 61
    segmentation_map[reference_map == 0.00000000000000011195362503715064549361] = 62
    segmentation_map[reference_map == 0.00000000000000005597681251857532274730] = 63
    segmentation_map[reference_map == 0.00000000000000002798840625928766137365] = 64
    segmentation_map[reference_map == 0.00000000000000001399420312464383068682] = 65
    segmentation_map[reference_map == 0.00000000000000000699710156232191534341] = 66
    segmentation_map[reference_map == 0.00000000000000000349855078116095767175] = 67
    segmentation_map[reference_map == 0.00000000000000000174927539058047883587] = 68
    segmentation_map[reference_map == 0.00000000000000000087463769529023941793] = 69
    segmentation_map[reference_map == 0.00000000000000000043731884764511970896] = 70
    segmentation_map[reference_map == 0.00000000000000000021865942382255985448] = 71
    segmentation_map[reference_map == 0.00000000000000000010932971191127992724] = 72
    segmentation_map[reference_map == 0.00000000000000000005466485595563996362] = 73
    segmentation_map[reference_map == 0.00000000000000000002733242797781998181] = 74
    segmentation_map[reference_map == 0.00000000000000000001366621398890999090] = 75
    segmentation_map[reference_map == 0.00000000000000000000683310699445099545] = 76
    segmentation_map[reference_map == 0.00000000000000000000341655349722549772] = 77
    segmentation_map[reference_map == 0.00000000000000000000170827674861274886] = 78
    segmentation_map[reference_map == 0.00000000000000000000085413837430637443] = 79
    segmentation_map[reference_map == 0.00000000000000000000042706918715318721] = 80
    segmentation_map[reference_map == 0.00000000000000000000021353459357659360] = 81
    segmentation_map[reference_map == 0.00000000000000000000010676729678829680] = 82
    segmentation_map[reference_map == 0.00000000000000000000005338364839414840] = 83
    segmentation_map[reference_map == 0.00000000000000000000002669182419707420] = 84
    segmentation_map[reference_map == 0.00000000000000000000001334591209853710] = 85
    segmentation_map[reference_map == 0.00000000000000000000000667295604927855] = 86
    segmentation_map[reference_map == 0.00000000000000000000000333647802463927] = 87
    segmentation_map[reference_map == 0.00000000000000000000000166823901231963] = 88
    segmentation_map[reference_map == 0.00000000000000000000000083411950615981] = 89
    segmentation_map[reference_map == 0.00000000000000000000000041705975307990] = 90
    segmentation_map[reference_map == 0.00000000000000000000000020852987653995] = 91
    segmentation_map[reference_map == 0.00000000000000000000000010426493826997] = 92
    segmentation_map[reference_map == 0.00000000000000000000000005213246913498] = 93
    segmentation_map[reference_map == 0.00000000000000000000000002606623456749] = 94
    segmentation_map[reference_map == 0.00000000000000000000000001303311728374] = 95
    segmentation_map[reference_map == 0.00000000000000000000000000651655864187] = 96
    segmentation_map[reference_map == 0.00000000000000000000000000325827932093] = 97
    segmentation_map[reference_map == 0.00000000000000000000000000162913966046] = 98
    segmentation_map[reference_map == 0.00000000000000000000000000081456983023] = 99
    segmentation_map[reference_map == 0.00000000000000000000000000040728491511] = 100
    segmentation_map[reference_map == 0.00000000000000000000000000020364245755] = 101
    segmentation_map[reference_map == 0.00000000000000000000000000010182122877] = 102
    segmentation_map[reference_map == 0.00000000000000000000000000005091061438] = 103
    segmentation_map[reference_map == 0.00000000000000000000000000002545530719] = 104
    segmentation_map[reference_map == 0.00000000000000000000000000001272765359] = 105
    segmentation_map[reference_map == 0.00000000000000000000000000000636382679] = 106
    segmentation_map[reference_map == 0.00000000000000000000000000000318191339] = 107
    segmentation_map[reference_map == 0.00000000000000000000000000000159095669] = 108
    segmentation_map[reference_map == 0.00000000000000000000000000000079547834] = 109
    segmentation_map[reference_map == 0.00000000000000000000000000000039773917] = 110
    segmentation_map[reference_map == 0.00000000000000000000000000000019886958] = 111
    segmentation_map[reference_map == 0.00000000000000000000000000000009943479] = 112
    segmentation_map[reference_map == 0.000000000000000000000000000000049717395] = 113
    segmentation_map[reference_map == 0.000000000000000000000000000000024858698] = 114
    segmentation_map[reference_map == 0.000000000000000000000000000000012429349] = 115
    segmentation_map[reference_map == 0.0000000000000000000000000000000062146749] = 116
    segmentation_map[reference_map == 0.0000000000000000000000000000000031073374] = 117
    segmentation_map[reference_map == 0.0000000000000000000000000000000015536687] = 118
    segmentation_map[reference_map == 0.00000000000000000000000000000000077683437] = 119
    segmentation_map[reference_map == 0.00000000000000000000000000000000038841718] = 120
    segmentation_map[reference_map == 0.00000000000000000000000000000000019420859] = 121
    segmentation_map[reference_map == 0.00000000000000000000000000000000009710429] = 122
    segmentation_map[reference_map == 0.000000000000000000000000000000000048552149] = 123
    segmentation_map[reference_map == 0.000000000000000000000000000000000024276074] = 124
    segmentation_map[reference_map == 0.000000000000000000000000000000000012138037] = 125
    segmentation_map[reference_map == 0.0000000000000000000000000000000000060690187] = 126
    segmentation_map[reference_map == 0.0000000000000000000000000000000000030345093] = 127
    segmentation_map[reference_map == 0.0000000000000000000000000000000000015172547] = 128
    segmentation_map[reference_map == 0.00000000000000000000000000000000000075862737] = 129
    segmentation_map[reference_map == 0.00000000000000000000000000000000000037931368] = 130
    segmentation_map[reference_map == 0.00000000000000000000000000000000000018965684] = 131
    segmentation_map[reference_map == 0.00000000000000000000000000000000000009482842] = 132
    segmentation_map[reference_map == 0.00000000000000000000000000000000000004741421] = 133
    segmentation_map[reference_map == 0.00000000000000000000000000000000000002370710] = 134
    segmentation_map[reference_map == 0.00000000000000000000000000000000000001185355] = 135
    segmentation_map[reference_map == 0.000000000000000000000000000000000000005926775] = 136
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000029633875] = 137
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000148169375] = 138
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000740846875] = 139
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000003704234375] = 140
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000018521171875] = 141
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000092605859375] = 142
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000463029296875] = 143
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000002315146484375] = 144
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000011575732421875] = 145
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000057878662109375] = 146
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000289393310546875] = 147
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000001446966552734375] = 148
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000007234832763671875] = 149
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000003617416381834375] = 150
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000018087081909171875] = 151
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000090435409545859375] = 152
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000452177047729296875] = 153
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000002260885238646484375] = 154
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000011304426193232421875] = 155
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000056522130966162109375] = 156
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000282610654830810546875] = 157
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000001413053274154052734375] = 158
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000070652663707702734375] = 159
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000353263318538513671875] = 160
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000017663165926925684375] = 161
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000883158296346284375] = 162
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000004415791481731421875] = 163
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000022078957408657109375] = 164
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000110394787043285546875] = 165
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000551973935221427734375] = 166
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000027598696761071384375] = 167
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000137993483805356921875] = 168
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000006899674190267846875] = 169
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000034498370951339234375] = 170
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000001724918547566961875] = 171
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000008624592737834809375] = 172
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000043122963689174046875] = 173
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000021561481844587021875] = 174
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000000107807409222935109375] = 175
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000000539037046114675546875] = 176
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000000269518523057287734375] = 177
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000000013475926152864384375] = 178
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000000067379630764321921875] = 179
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000000336898153821609546875] = 180
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000000168449076910804734375] = 181
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000000008422453845540236875] = 182
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000000042112269227701184375] = 183
    segmentation_map[reference_map == 0.0000000000000000000000000000000000000000000000000000210561346138505921875] = 184
    segmentation_map[reference_map == 0.00000000000000000000000000000000000000000000000000001052806730692529609375] = 185
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000000005264033653348648046875] = 186
    segmentation_map[reference_map == 0.000000000000000000000000000000000000000000000000000002632016826674324021875] = 187
    segmentation_map[reference_map == 
```

```

    if segmentation_map is not None:
        segmentation_results.append(segmentation_map)
        model_identifiers.append(model_id)
        print(f"    Успешно завершено")
    else:
        print(f"    Ошибка обработки")

if segmentation_results:
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    for idx, (seg_map, model_id) in enumerate(zip(segmentation_results,
                                                model_identifiers)):
        if 'Hist' in model_id:
            method_name = 'Гистограмма'
            classifier_type = 'KNN'
        elif 'Laws' in model_id:
            method_name = 'Laws'
            if 'Tree' in model_id:
                classifier_type = 'Дерево'
            else:
                classifier_type = 'SVM'

        axes[idx].imshow(input_image, cmap='gray', alpha=0.7)
        axes[idx].imshow(seg_map, cmap='viridis', alpha=0.6)
        axes[idx].set_title(f'{method_name} + {classifier_type}', fontweight='bold')
        axes[idx].axis('off')

    plt.tight_layout()
    plt.show()

    print(f"Успешно обработано: {len(segmentation_results)}/3")
else:
    print("Все модели завершились с ошибкой!")
else:
    print(f"Ошибка загрузки: {target_image_path}")

def simplify_color_scheme(mask_array, max_colors=3):
    unique_colors, color_counts = np.unique(mask_array, return_counts=True)

    sorted_colors = unique_colors[np.argsort(-color_counts)]
    dominant_colors = sorted_colors[:max_colors]

    print(f"Доминирующие цвета: {dominant_colors}")
    print(f"Количество пикселей: {color_counts[np.argsort(-color_counts)]}")

    simplified_mask = np.zeros_like(mask_array)

    for new_color, original_color in enumerate(dominant_colors):
        simplified_mask[mask_array == original_color] = new_color

    print(f"Сопоставление: {dominant_colors} -> {list(range(len(dominant_colors)))}")

    return simplified_mask

reference_mask = cv2.imread("./tests/xleb_mask.jpg", cv2.IMREAD_GRAYSCALE)
if reference_mask is not None:
    processed_reference_mask = simplify_color_scheme(reference_mask, max_colors=3)

    plt.figure(figsize=(12, 5))

```

```
plt.subplot(1, 3, 1)
plt.imshow(reference_mask, cmap='gray')
plt.title(f'Исходная маска\n{len(np.unique(reference_mask))} цветов')
plt.colorbar()

plt.subplot(1, 3, 2)
plt.imshow(processed_reference_mask, cmap='plasma')
plt.title('Упрощенная маска\n3 основных цвета')
plt.colorbar()

plt.tight_layout()
plt.show()

from sklearn.metrics import accuracy_score, jaccard_score

def evaluate_segmentation_quality(ground_truth_mask, predicted_mask, mode):

    if ground_truth_mask.shape != predicted_mask.shape:
        print(f"Размеры масок отличаются: эталон {ground_truth_mask.shape}")
        resized_prediction = cv2.resize(predicted_mask, (ground_truth_mask.shape[0], ground_truth_mask.shape[1]),
                                         interpolation=cv2.INTER_NEAREST)
    else:
        resized_prediction = predicted_mask

    gt_classes = np.unique(ground_truth_mask)
    pred_classes = np.unique(resized_prediction)

    print(f"Классы в эталоне: {gt_classes}")
    print(f"Классы в предсказании: {pred_classes}")

    class_correspondence = {
        0: 2,
        1: 1,
        2: 0
    }

    aligned_prediction = resized_prediction.copy()
    for pred_class, gt_class in class_correspondence.items():
        aligned_prediction[resized_prediction == pred_class] = gt_class

    pixel_accuracy = accuracy_score(ground_truth_mask.flatten(), aligned_prediction)
    print(f"Точность: {pixel_accuracy:.1%}")
    print(f"Метрика IoU по классам:")

    for class_id in gt_classes:
        class_iou = jaccard_score(ground_truth_mask.flatten(), aligned_prediction,
                                  average=None, labels=[class_id])[0]
        print(f"    Класс {class_id}: {class_iou:.3f}")

    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    axes[0].imshow(original_image, cmap='gray')
    axes[0].set_title('Исходное изображение')
    axes[0].axis('off')

    axes[1].imshow(ground_truth_mask, cmap='plasma')
    axes[1].set_title('Эталонная разметка')
    axes[1].axis('off')
```

```

        axes[2].imshow(aligned_prediction, cmap='plasma')
        axes[2].set_title(f'Предсказание сегментации')
        axes[2].axis('off')

        plt.tight_layout()
        plt.show()

    return pixel_accuracy

target_image_path = "./tests/xleb.jpg"
input_image = cv2.imread(target_image_path, cv2.IMREAD_GRAYSCALE)

selected_model_id = 'LawS_SVM'
selected_model_data = best_models[selected_model_id]

print("Запуск сегментации...")
result_mask = perform_texture_segmentation(
    selected_model_data, selected_model_id, target_image_path,
    window_size=32, stride=8, num_classes=3
)

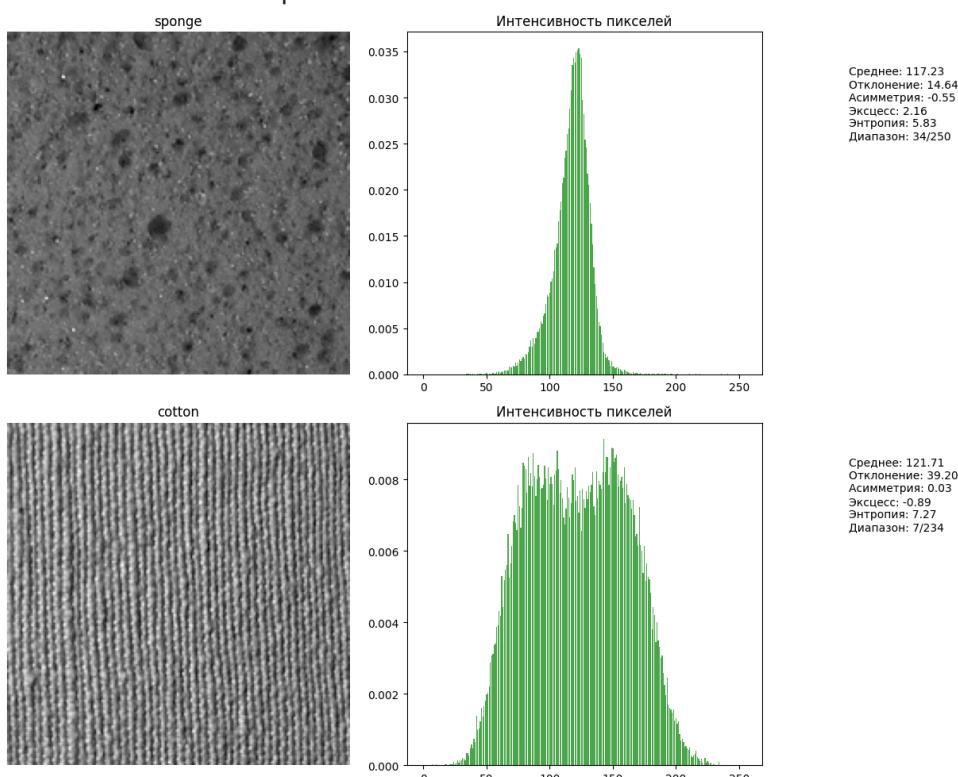
print(f"Результат: {result_mask.shape}")
print(f"Классы в результате: {np.unique(result_mask)}")

if processed_reference_mask is not None and result_mask is not None and i
    accuracy_value = evaluate_segmentation_quality(processed_reference_ma
    print(f"Итоговая точность: {accuracy_value*100:.1f}%")



```

Загружено данных: (810, 10)
Уникальных категорий: 10



Обнаружено изображений: 810

Обработка 0/810

Обработка 50/810

Обработка 100/810

Обработка 150/810

Обработка 200/810

Обработка 250/810

Обработка 300/810

Обработка 350/810

Обработка 400/810

Обработка 450/810

Обработка 500/810

Обработка 550/810

Обработка 600/810

Обработка 650/810

Обработка 700/810

Обработка 750/810

Обработка 800/810

Размерность признаков: (810, 10)

ГИСТОГРАММА

kNN

	precision	recall	f1-score	support
aluminium_foil	0.95	0.95	0.95	20
brown_bread	0.86	0.95	0.90	20
corduroy	0.66	0.95	0.78	20
cotton	0.59	0.62	0.60	21
cracker	0.83	0.90	0.86	21
linen	0.94	0.75	0.83	20
orange_peel	1.00	0.90	0.95	21
sandpaper	0.54	0.35	0.42	20
sponge	0.83	0.75	0.79	20
styrofoam	0.90	0.95	0.93	20
accuracy			0.81	203
macro avg	0.81	0.81	0.80	203
weighted avg	0.81	0.81	0.80	203

SVM

	precision	recall	f1-score	support
aluminium_foil	1.00	1.00	1.00	20
brown_bread	0.83	0.75	0.79	20
corduroy	0.65	0.85	0.74	20
cotton	0.76	0.62	0.68	21
cracker	0.74	0.95	0.83	21
linen	0.80	0.80	0.80	20
orange_peel	0.95	0.86	0.90	21
sandpaper	0.87	0.65	0.74	20
sponge	0.90	0.90	0.90	20
styrofoam	0.90	0.95	0.93	20
accuracy			0.83	203
macro avg	0.84	0.83	0.83	203
weighted avg	0.84	0.83	0.83	203

Дерево

	precision	recall	f1-score	support
aluminium_foil	0.95	1.00	0.98	20
brown_bread	0.90	0.90	0.90	20
corduroy	0.70	0.95	0.81	20
cotton	0.79	0.71	0.75	21
cracker	0.73	0.76	0.74	21
linen	0.89	0.85	0.87	20
orange_peel	1.00	0.81	0.89	21
sandpaper	0.76	0.65	0.70	20
sponge	0.84	0.80	0.82	20
styrofoam	0.91	1.00	0.95	20
accuracy			0.84	203
macro avg	0.85	0.84	0.84	203
weighted avg	0.85	0.84	0.84	203

LAWs

kNN

	precision	recall	f1-score	support
aluminium_foil	1.00	0.95	0.97	20
brown_bread	0.57	0.85	0.68	20
corduroy	0.76	0.80	0.78	20
cotton	0.91	0.48	0.62	21
cracker	0.68	0.62	0.65	21
linen	0.69	0.45	0.55	20
orange_peel	0.75	0.86	0.80	21
sandpaper	0.65	0.55	0.59	20
sponge	0.78	0.90	0.84	20
styrofoam	0.58	0.75	0.65	20
accuracy			0.72	203
macro avg	0.74	0.72	0.71	203
weighted avg	0.74	0.72	0.71	203

SVM

	precision	recall	f1-score	support
aluminium_foil	1.00	0.95	0.97	20
brown_bread	0.35	0.35	0.35	20
corduroy	0.92	0.60	0.73	20
cotton	1.00	0.48	0.65	21
cracker	0.68	0.62	0.65	21
linen	1.00	0.55	0.71	20
orange_peel	0.30	1.00	0.47	21
sandpaper	0.18	0.10	0.13	20
sponge	0.25	0.20	0.22	20
styrofoam	0.40	0.30	0.34	20
accuracy			0.52	203
macro avg	0.61	0.51	0.52	203
weighted avg	0.61	0.52	0.52	203

Дерево

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

aluminium_foil	1.00	0.95	0.97	20
brown_bread	0.81	0.85	0.83	20
corduroy	0.89	0.85	0.87	20
cotton	0.82	0.67	0.74	21
cracker	0.65	0.62	0.63	21
linen	0.86	0.90	0.88	20
orange_peel	0.90	0.86	0.88	21
sandpaper	0.54	0.35	0.42	20
sponge	0.50	0.75	0.60	20
styrofoam	0.65	0.75	0.70	20
accuracy			0.75	203
macro avg	0.76	0.75	0.75	203
weighted avg	0.76	0.75	0.75	203

GLCM kNN

	precision	recall	f1-score	support
aluminium_foil	0.90	0.90	0.90	20
brown_bread	0.31	0.55	0.40	20
corduroy	0.32	0.40	0.36	20
cotton	0.42	0.38	0.40	21
cracker	0.48	0.52	0.50	21
linen	0.38	0.30	0.33	20
orange_peel	0.59	0.76	0.67	21
sandpaper	0.38	0.25	0.30	20
sponge	0.43	0.30	0.35	20
styrofoam	0.73	0.40	0.52	20
accuracy			0.48	203
macro avg	0.49	0.48	0.47	203
weighted avg	0.49	0.48	0.47	203

SVM

	precision	recall	f1-score	support
aluminium_foil	1.00	1.00	1.00	20
brown_bread	0.44	0.80	0.57	20
corduroy	0.22	0.10	0.14	20
cotton	0.60	0.14	0.23	21
cracker	0.61	0.67	0.64	21
linen	0.43	0.45	0.44	20
orange_peel	0.56	0.95	0.70	21
sandpaper	0.50	0.30	0.38	20
sponge	0.56	0.45	0.50	20
styrofoam	0.52	0.65	0.58	20
accuracy			0.55	203
macro avg	0.54	0.55	0.52	203
weighted avg	0.54	0.55	0.52	203

Дерево

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

aluminium_foil	1.00	0.95	0.97	20
brown_bread	0.41	0.65	0.50	20
corduroy	0.53	0.50	0.51	20
cotton	0.40	0.29	0.33	21
cracker	0.74	0.67	0.70	21
linen	0.29	0.35	0.32	20
orange_peel	0.86	0.57	0.69	21
sandpaper	0.32	0.70	0.44	20
sponge	0.86	0.30	0.44	20
styrofoam	0.90	0.45	0.60	20
accuracy			0.54	203
macro avg	0.63	0.54	0.55	203
weighted avg	0.63	0.54	0.55	203

Изображение загружено: True

Маска загружена: True

Модель: Hist_kNN

Классы после фильтрации: [0 1 2]

Успешно завершено

Модель: Laws_Tree

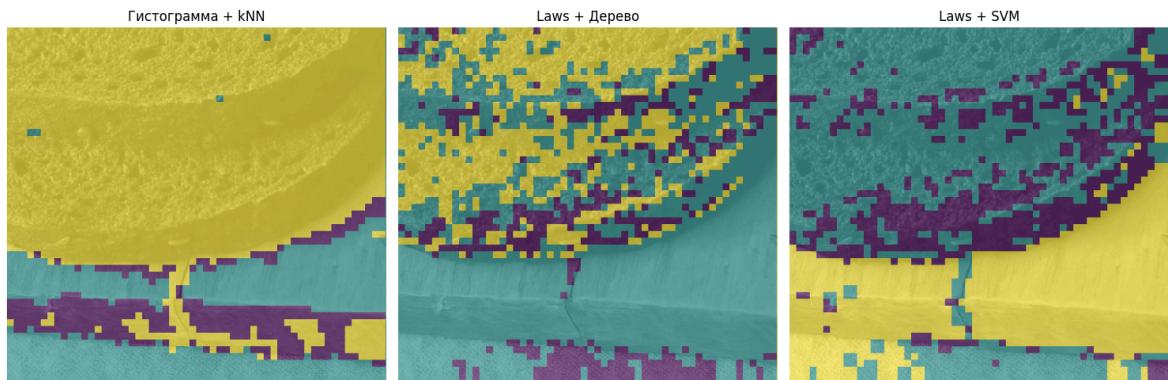
Классы после фильтрации: [0 1 2]

Успешно завершено

Модель: Laws_SVM

Классы после фильтрации: [0 1 2]

Успешно завершено

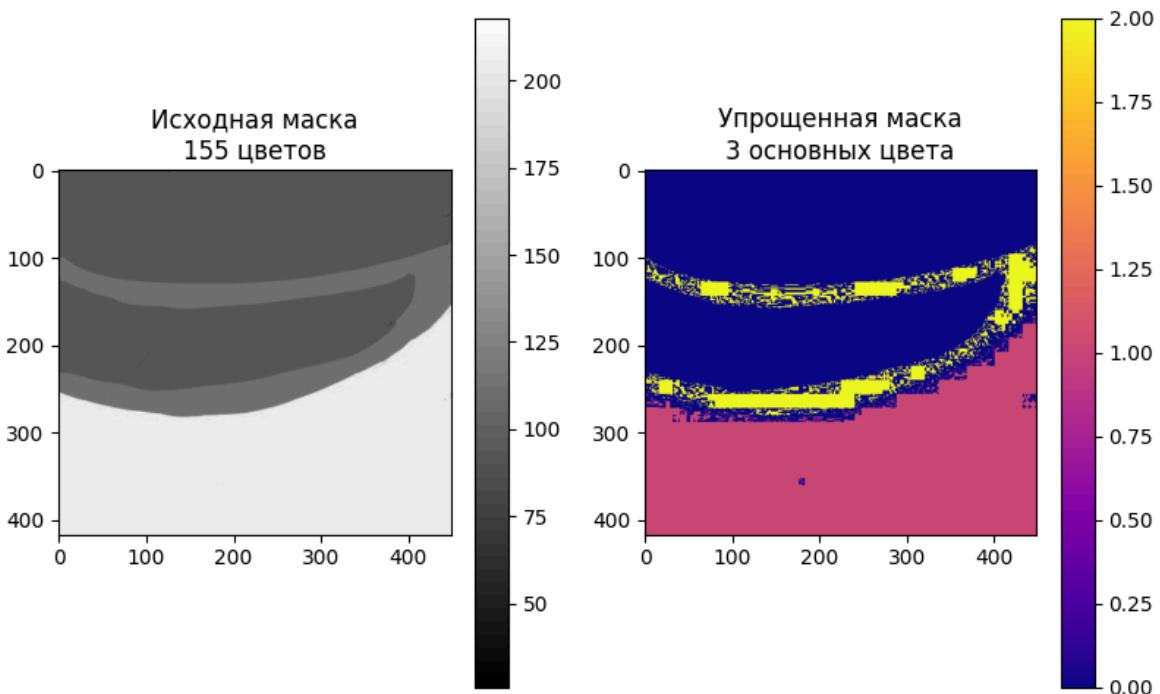


Успешно обработано: 3/3

Доминирующие цвета: [91 204 112]

Количество пикселей: [78636 70066 13193]

Сопоставление: [91 204 112] -> [0, 1, 2]



Запуск сегментации...

Классы после фильтрации: [0 1 2]

Результат: (417, 449)

Классы в результате: [0 1 2]

Классы в эталоне: [0 1 2]

Классы в предсказании: [0 1 2]

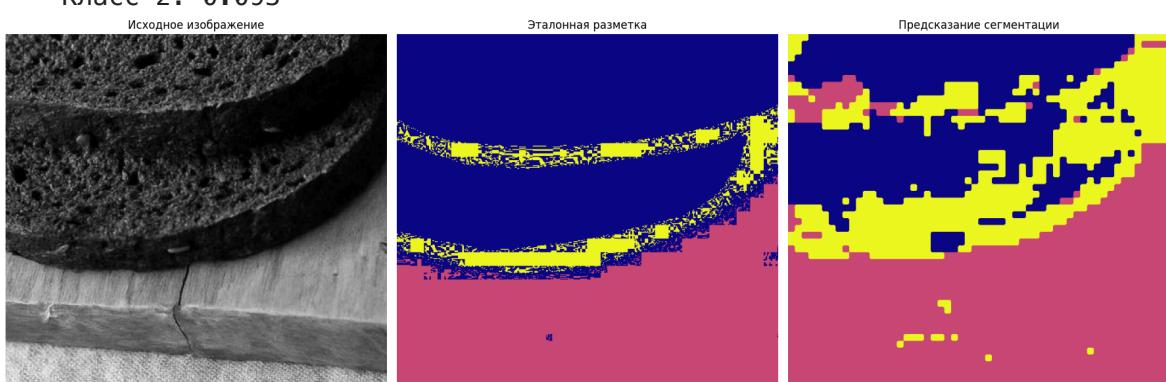
Точность: 68.3%

Метрика IoU по классам:

Класс 0: 0.499

Класс 1: 0.789

Класс 2: 0.093



Итоговая точность: 68.3%

In []:

In []: