Instructions: https://cdn-uploads.piazza.com/paste/kxtikuygs2i1tq/dedf10905a67a890dda90fbb1786080cacfe97a25024d069bcc890ac833d719f/final_project_rubric_revised.pdf

Recycled Object Classification with Computer Vision Techniques

Aastha Khanna, Camille Church, Srila Maiti

Introduction

Optimizing the recycling of our waste is becoming more vital every year. If US residents recycled all newspapers, we would save an estimated 250 million trees each year. However, multiple hurdles prevent this from happening. First, many people are unaware of best practices or find them too difficult to follow. This confusion causes either incorrect handling of waste or people simply not recycling.

Additionally, recycling is often viewed as too expensive by smaller municipalities. On average, one ton of trash destined for the landfill will cost $50 less than a ton of recycling waste. As a result, many rural areas have abandoned recycling programs as a cost-saving measure. Finally, and most importantly, recycling work can be highly hazardous. Unfortunately, in the city of one of our team members, a person passed away at a local recycling center.

So how can we balance the challenges of recycling with the benefit it can bring to society? Automation. We can save money, reduce risk, and eliminate human error by creating a machine learning model to sort waste and remove the human element. This notebook, created as a W281 class project at UC Berkeley, takes an initial look at creating an ML model using computer vision techniques to classify images of recyclables.

Data

Our data came from a recycling dataset found on Hugging Face, link here. The dataset is generated via web-scraping from DuckDuckGo using this tool, and consists of 11 categories of JPEG color images with varied heights ranging from 139 to 592 pixels, and widths ranging from 160 to 474 pixels. The dataset categories are: aluminium, batteries, cardboard, disposable plates, glass, hard plastic, paper, paper towels, polystyrene, soft plastics, and takeaway cups. The below table details an initial assessment of the dataset categories as performed by our group.

Category	Number of Images	
aluminium(0)	269	A silvery metal, some textured, and some are smooth

Category	Number of Images	
batteries(1)	297	Smooth cylindrical and box shaped, various colors.
cardboard(2)	274	Mostly brown to light tan in color, little texture, rectangular shaped, also several crafts m
disposable plates(3)	273	Round shaped, various colors, some texture around the edges, some are paper and som
glass(4)	294	Various opaque colors and with generally consistent design patterns.
hard plastic(5)	280	No opacity, consistent formed shaped (rectangular or circle).
paper(6)	255	Typically white, rectangular shape, however thin and bendable
paper towel(7)	291	Some high frequency textures and sometimes patterns, and similar colors throughout (r
polystyrene(8)	291	Generally transparent with smooth texture and sharp defined edges.
soft plastics(9)	283	Generally transparent with low frequency. texture
takeaway cups(10)	300	Cylindrical shape, smaller at the bottom, larger on top. Various bright colors. Some data

In terms of data split, there is a slight imbalance in the distribution between categories which requires an adjustment while forming the train, test and validation data set. We are planning to split our dataset into 80:10:10 ratio for train, validation and test dataset. We also noticed that our input data has varied dimensions, so we will need to be resize them to the common size format. Our images are all color images, and we want to convert them into grayscale images for ease of processing when extracting features. By doing this we do lose the color information, but we believe that there are additional features that can be leveraged to classify the images correctly. We also have noticed duplicate images, misclassified images, and cartoon images, mixed bag images (images with other artifacts) in the dataset.

We decided that we wanted to drop some of these categories due to data quality issues. Takeaway cups were dropped as many of the cups were not takeaway and therefore weren't recyclable; soft plastics were dropped as there were a large amount of misclassified images in the dataset; polystyrene was dropped as there were a vast variety of objects that were made from it and different textures, making it hard to extract features with our current CV knowledge; and hard plastics were dropped as there were a large amount of misclassified images in the dataset. We also were suggested in office hours to further downsize the number of categories we work with to reduce the complexity and create better classifiers for a smaller number of classes. To comply with this, we decided to keep only 5 categories that we thought had quality datasets and relatively consistent features that we could extract: batteries, paper towels, paper, cardboard, and plates.

Installing New Libraries

```
%%time
%%capture

### Installing any uninstalled dependencies for the Python kernel
import sys
!{sys.executable} -m pip install numpy
```

```
!{sys.executable} -m pip install matplotlib
!{sys.executable} -m pip install opencv-python
!{sys.executable} -m pip install rembg
!{sys.executable} -m pip install scikit-image
!{sys.executable} -m pip install scikit-learn
!{sys.executable} -m pip install pandas
!{sys.executable} -m pip install seaborn
```

```
CPU times: user 183 ms, sys: 83 ms, total: 266 ms
Wall time: 29.1 s
```

Importing Libraries

```
%%time
# Hide warnings for final output
import warnings

warnings.filterwarnings('ignore')
```

```
CPU times: user 40 µs, sys: 0 ns, total: 40 µs
Wall time: 44.1 µs
```

```
%%time
### Imports

# TODO get rid of duplicate/unused imports
import numpy as np
import matplotlib.pyplot as plt
import skimage.feature as feature
import cv2
import os
import matplotlib.image as mpimg
from skimage.util import img_as_ubyte
from rembg import remove
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.decomposition import IncrementalPCA
from skimage.feature import hog
from scipy.ndimage import convolve
from scipy import signal
import pandas as pd
from sklearn.cluster import DBSCAN
import copy
import seaborn as sns

from skimage.feature import hog
from scipy.ndimage import convolve
from scipy import signal

from sklearn.manifold import TSNE
from skimage.filters import prewitt_h, prewitt_v
from skimage import filters
from skimage.feature import haar_like_feature_coord
```

```
from skimage.feature import draw_haar_like_feature
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from PIL import Image
from PIL import ImageEnhance
from PIL import ImageFilter
from scipy.cluster.vq import kmeans
from scipy.cluster.vq import vq
from scipy.cluster.vq import whiten
import joblib
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from IPython.display import display
from sklearn.model_selection import StratifiedKFold
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import f1_score, recall_score, precision_score

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import ast
import glob
```

```
CPU times: user 24.8 s, sys: 1.57 s, total: 26.3 s
Wall time: 26.4 s
```

```
%%time
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils.layer_utils import count_params

from tensorflow.keras.layers import RandomFlip
from tensorflow.keras.layers import RandomZoom
from tensorflow.keras.layers import RandomRotation
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.layers import MaxPooling1D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import GlobalMaxPooling2D
from tensorflow.keras.layers import GlobalAveragePooling1D
```

```
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.layers import Lambda
from tensorflow.keras.layers import Multiply
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import PReLU
from keras.layers.core import Activation
from keras.layers.convolutional import SeparableConv1D
from keras.layers.convolutional import SeparableConv2D

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
```

```
CPU times: user 2.34 s, sys: 233 ms, total: 2.58 s
Wall time: 2.56 s
```

Defining common variables

```
%%time
label_dict = {0: "battery",
              1: "cardboard",
              2: "disposable_plates",
              3: "paper",
              4: "paper_towel"
             }
label_list = ["Batteries", "Cardboard", "Disposable Plates", "Paper", "Paper Towels"]

batteries_image_dir_path = '/content/batteries'
cardboard_image_dir_path = '/content/cardboard'
disposable_plates_image_dir_path = '/content/disposable_plates'
paper_image_dir_path = '/content/paper'
paper_towel_image_dir_path = '/content/paper_towel'
processed_image_path_list = [batteries_image_dir_path,
                             cardboard_image_dir_path,
                             disposable_plates_image_dir_path,
                             paper_image_dir_path,
                             paper_towel_image_dir_path
                            ]
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 8.34 µs
```

Generic Supporting Functions

Preprocessing Functions

```
%%time
### Preprocessing

def load_resized_images_as_grayscale_from_folder(folder):
    images = []
    file_names = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename)) # Read image
        img = cv2.resize(img, (64, 64)) # Resize to 64 x 64
        img = remove(img) # Remove background
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert to grayscale
        if img is not None:
            images.append(img)
            file_names.append(os.path.join(folder,filename))
    return images, file_names

def preprocessing(image_dir):
    # Remove duplicates, convert to grayscale, resize to 64 x 64, remove background, normalize
    image_set, file_names = load_resized_images_as_grayscale_from_folder(image_dir)
    scaled_image_set = [np.ravel((image / 255)) for image in image_set] # Scaled because DBSCAN
    dbs_cluster = DBSCAN(eps=.5, min_samples=2).fit(scaled_image_set)
    clean_image_index = []
    dupl_image_index = []
    deduped_images = []
    duplicate_image_names = []
    for idx, label in enumerate(dbs_cluster.labels_):
         if label == -1:
             clean_image_index.append(idx)
         else: # Duplicates
             index_list = np.where(dbs_cluster.labels_ == label)
             clean_image_index.extend(index_list[0])
             dupl_image_index.extend(index_list[0][1:])

    clean_image_index = list(dict.fromkeys(clean_image_index))
    dupl_image_index = dict.fromkeys(dupl_image_index)

    for i in clean_image_index:
        if i not in dupl_image_index:
            deduped_images.append(scaled_image_set[i].reshape((64, 64)))

    for i in list(dupl_image_index):
         duplicate_image_names.append(file_names[i])

    print(duplicate_image_names)
    print(len(duplicate_image_names))
    return [img_as_ubyte(im) for im in deduped_images]
```

```
CPU times: user 6 µs, sys: 0 ns, total: 6 µs
Wall time: 10.5 µs
```

PCA Related Function

```
%%time
### EDA

def get_eigenimages(data):
    D = np.zeros((len(data), (64 * 64)))

    for i, img in enumerate(data):
        D[i, :] = img.flatten()

    pca = PCA().fit(D)

    # How many principal components explain 95% of the variance?
    var_cum = np.cumsum(pca.explained_variance_ratio_)*100
    k = np.argmax(var_cum>95)
    print("Number of principal components explaining 95% of variance: "+ str(k))

    return pca.components_[:k]

def show_all_eigenimages(paper_towel_data, cardboard_data, batteries_data, plates_data, paper_c
    plt.figure(figsize = (20,10))
    plt.suptitle("Eigenimages per Category")

    images = get_eigenimages(paper_towel_data)
    subplot_idx = 1

    for i in range(1, 10):
        ax = plt.subplot(5, 9, subplot_idx)
        subplot_idx += 1
        ax.set_title("Paper Towel " + str(i))
        ax.set_axis_off()
        ax.imshow(images[i].reshape((64, 64)), cmap="gray")

    images = get_eigenimages(cardboard_data)

    for i in range(1, 10):
        ax = plt.subplot(5, 9, subplot_idx)
        subplot_idx += 1
        ax.set_title("Cardboard " + str(i))
        ax.set_axis_off()
        ax.imshow(images[i].reshape((64, 64)), cmap="gray")

    images = get_eigenimages(batteries_data)

    for i in range(1, 10):
        ax = plt.subplot(5, 9, subplot_idx)
        subplot_idx += 1
        ax.set_title("Batteries " + str(i))
        ax.set_axis_off()
        ax.imshow(images[i].reshape((64, 64)), cmap="gray")

    images = get_eigenimages(plates_data)

    for i in range(1, 10):
        ax = plt.subplot(5, 9, subplot_idx)
        subplot_idx += 1
        ax.set_title("Plates " + str(i))
        ax.set_axis_off()
```

```
        ax.imshow(images[i].reshape((64, 64)), cmap="gray")

    images = get_eigenimages(paper_data)

    for i in range(1, 10):
        ax = plt.subplot(5, 9, subplot_idx)
        subplot_idx += 1
        ax.set_title("Paper " + str(i))
        ax.set_axis_off()
        ax.imshow(images[i].reshape((64, 64)), cmap="gray")

def get_average_image(data):
    return np.average(np.array([np.array(im) for im in data]), axis=0)

def show_all_average_images(paper_towel_data, cardboard_data, batteries_data, plates_data, pape
    plt.figure(figsize = (20,5))
    plt.suptitle("Average Images per Category")

    ax1 = plt.subplot(1, 5, 1)
    ax1.set_title("Paper Towel")
    ax1.set_axis_off()
    ax1.imshow(get_average_image(paper_towel_data), cmap="gray")

    ax2 = plt.subplot(1, 5, 2)
    ax2.set_title("Cardboard")
    ax2.set_axis_off()
    ax2.imshow(get_average_image(cardboard_data), cmap="gray")

    ax3 = plt.subplot(1, 5, 3)
    ax3.set_title("Batteries")
    ax3.set_axis_off()
    ax3.imshow(get_average_image(batteries_data), cmap="gray")

    ax4 = plt.subplot(1, 5, 4)
    ax4.set_title("Plates")
    ax4.set_axis_off()
    ax4.imshow(get_average_image(plates_data), cmap="gray")

    ax5 = plt.subplot(1, 5, 5)
    ax5.set_title("Paper")
    ax5.set_axis_off()
    ax5.imshow(get_average_image(paper_data), cmap="gray")

def apply_pca_and_reconstruct_image(image_data_list):
    reconstruct_image_data_list = []
    pca_component_count_list = []
    pca = PCA()
    for image in image_data_list:
        pca.fit_transform(image)
        var_cum = np.cumsum(pca.explained_variance_ratio_)*100
        k = np.argmax(var_cum > 95)
        pca_component_count_list.append(k)
        if k > 0:
            ipca = IncrementalPCA(n_components = k)
            image_recon = ipca.inverse_transform(ipca.fit_transform(image))
            reconstruct_image_data_list.append(image_recon)
        else:
```

```
            reconstruct_image_data_list.append(image)
    return reconstruct_image_data_list, pca_component_count_list

def get_fit_transform(x_train_sc, x_test_sc):
    pca = PCA(.95)
    pca.fit(x_train_sc)
    print(pca.n_components_)

    x_train_sc_trnsfm = pca.transform(x_train_sc)
    x_test_sc_trnsfm = pca.transform(x_test_sc)

```

```
CPU times: user 6 µs, sys: 0 ns, total: 6 µs
Wall time: 9.54 µs
```

Feature Extraction Related Functions

```
%%time
### Feature extraction functions

def get_single_glcm_feature(im, angle, feature_name):
    graycom = feature.graycomatrix(im, distances=[1], angles=[angle], levels=256, symmetric=Tru

    # Extract GLCM features.
    if feature_name == "contrast":
        return feature.graycoprops(graycom, 'contrast')
    elif feature_name == "dissimilarity":
        return feature.graycoprops(graycom, 'dissimilarity')
    elif feature_name == "homogenity":
        return feature.graycoprops(graycom, 'homogeneity')
    elif feature_name == "energy":
        return feature.graycoprops(graycom, 'energy')
    elif feature_name == "correlation":
        return feature.graycoprops(graycom, 'correlation')
    elif feature_name == "ASM":
        return feature.graycoprops(graycom, 'ASM')

    # TODO: Hyperparameter tuning for levels and angles using feature importance function
    raise Exception("Feature parameter is not a valid value")

def get_glcm_features(np_img_list,
                      glcm_distance_list = [1],
                      glcm_angle_list = [0, np.pi/4, np.pi/2, 3*np.pi/4],
                      feature_list = ['contrast', 'dissimilarity', 'homogeneity', 'energy', 'co
    """
    np_img_list: List of numpy array representation of images
    Distances: List of pixel pair distance offsets.
               Here we will use 1 in each angle direction.
    Angles: List of pixel pair angles in radians.
            We will pass this in. Good values to try: [0, 45, 90, 135]
    From UCalgary doc: Any matrix can be made symmetrical by adding it
    to its transpose.
    However, texture calculations are best performed on a symmetrical matrix.
    Also normalizing so we get probability values.
    """
```

```
    glcm_feature_list = []
    for img in np_img_list:
        graycom = feature.graycomatrix(img,
                                       distances = glcm_distance_list,
                                       angles = glcm_angle_list,
                                       levels = 256,
                                       symmetric = True,
                                       normed = True)

        glcm_feature_contrast = np.ravel(feature.graycoprops(graycom, 'contrast'))
        glcm_feature_dissimilarity = np.ravel(feature.graycoprops(graycom, 'dissimilarity'))
        glcm_feature_homogeneity = np.ravel(feature.graycoprops(graycom, 'homogeneity'))
        glcm_feature_energy = np.ravel(feature.graycoprops(graycom, 'energy'))
        glcm_feature_correlation = np.ravel(feature.graycoprops(graycom, 'correlation'))
        glcm_feature_ASM = np.ravel(feature.graycoprops(graycom, 'ASM'))
        glcm_feature_list.append(np.concatenate((glcm_feature_contrast,
                                                 glcm_feature_dissimilarity,
                                                 glcm_feature_homogeneity,
                                                 glcm_feature_energy,
                                                 glcm_feature_correlation,
                                                 glcm_feature_ASM)))

    glcm_feature_columns = ['glcm_contrast_dist_1_angle_0',
                            'glcm_contrast_dist_1_angle_45',
                            'glcm_contrast_dist_1_angle_90',
                            'glcm_contrast_dist_1_angle_135',

                            'glcm_dissimilarity_dist_1_angle_0',
                            'glcm_dissimilarity_dist_1_angle_45',
                            'glcm_dissimilarity_dist_1_angle_90',
                            'glcm_dissimilarity_dist_1_angle_135',

                            'glcm_homogeneity_dist_1_angle_0',
                            'glcm_homogeneity_dist_1_angle_45',
                            'glcm_homogeneity_dist_1_angle_90',
                            'glcm_homogeneity_dist_1_angle_135',

                            'glcm_energy_dist_1_angle_0',
                            'glcm_energy_dist_1_angle_45',
                            'glcm_energy_dist_1_angle_90',
                            'glcm_energy_dist_1_angle_135',

                            'glcm_correlation_dist_1_angle_0',
                            'glcm_correlation_dist_1_angle_45',
                            'glcm_correlation_dist_1_angle_90',
                            'glcm_correlation_dist_1_angle_135',

                            'glcm_asm_dist_1_angle_0',
                            'glcm_asm_dist_1_angle_45',
                            'glcm_asm_dist_1_angle_90',
                            'glcm_asm_dist_1_angle_135'
                           ]
    return pd.DataFrame(glcm_feature_list, columns = glcm_feature_columns)

def get_lbp_features(np_img_list, lbp_numPoints = 50, lbp_radius = 30):
    class LocalBinaryPatterns:
        def __init__(self, numPoints, radius):
```

```
            self.numPoints = numPoints
            self.radius = radius

        def describe(self, image, eps = 1e-7):
            lbp = feature.local_binary_pattern(image, self.numPoints, self.radius, method="uni
            (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, self.numPoints+3), range=(0
            # Normalize the histogram
            hist = hist.astype('float')
            hist /= (hist.sum() + eps)
            return hist, lbp

    hist_list = []
    lbp_list = []
    desc = LocalBinaryPatterns(lbp_numPoints, lbp_radius)

    for img in np_img_list:
        hist, lbp = desc.describe(img)
        hist_list.append(hist)
        lbp_list.append(lbp)

    return hist_list, lbp_list

def get_hog_feature(np_img_list, orientations = 4, pixels_per_cell = (8, 8), cells_per_block =
    # TODO: Hyperparameter tuning for orientations, pixels_per_cell, cells_per_block
    fd_list = []
    viz_list = []
    for img in np_img_list:
        fd, viz = hog(img,
                      orientations = orientations,
                      cells_per_block = cells_per_block,
                      pixels_per_cell = pixels_per_cell,
                      visualize = True,
                      multichannel = False)
        fd_list.append(fd)
        viz_list.append(viz)
    return viz_list, fd_list

def get_sift_feature(np_img_list):
    simplefilter = cv2.SIFT_create()
    return [simplefilter.detectAndCompute(img, None) for img in np_img_list]

def get_orb_feature(img, orb):
    kp, des = orb.detectAndCompute(img, None)
    return kp

def get_sift_descriptors(feature_list):
    descriptors_list = []
    for feature_sift in feature_list:
        for entry in feature_sift:
            kp_val, descriptor_val = entry
            if descriptor_val is not None:
                descriptors_list.append(descriptor_val)

    descriptors = descriptors_list[0][1]
    for descriptor in descriptors_list[1:]:
        if descriptor is not None:
            descriptors = np.vstack((descriptors, descriptor))
```

```
153
154     descriptors_float = descriptors.astype(float)
155     return descriptors_list, descriptors_float
156
157 def create_2d_gaussian(size=9, std=1.5):
158     gaussian_1d = signal.gaussian(size,std=std)
159     gaussian_2d = np.outer(gaussian_1d, gaussian_1d)
160     gaussian_2d = gaussian_2d/(gaussian_2d.sum())
161     return gaussian_2d
162
163 def remove_texture(image):
164     low_pass_gaussian = create_2d_gaussian(15, 8)
165     img_less_texture = convolve(image, low_pass_gaussian)
166     return img_less_texture
167
168 def get_edge_feature(np_img_list, threshold1=30, threshold2=100):
169     return [cv2.Canny(remove_texture(remove_texture(img)), threshold1=threshold1, threshold2=th
170
171 def get_prewitt_h_kernel_feature(np_img_list):
172     return [prewitt_h(img) for img in np_img_list]
173
174 def get_prewitt_v_kernel_feature(np_img_list):
175     return [prewitt_v(img) for img in np_img_list]
176
177 def get_sobel_feature(np_img_list):
178     return [filters.sobel(img) for img in np_img_list]
179
180 def get_contour_feature(np_img_list):
181     return [np.array(Image.fromarray(np.uint8(img)).filter(ImageFilter.CONTOUR)) for img in np_
182
183 def get_edge_enhance_feature(np_img_list):
184     return [np.array(Image.fromarray(np.uint8(img)).filter(ImageFilter.EDGE_ENHANCE)) for img i
185
186 def get_edge_enhance_more_feature(np_img_list):
187     return [np.array(Image.fromarray(np.uint8(img)).filter(ImageFilter.EDGE_ENHANCE_MORE)) for
188
189 def get_glcm_features_per_class(data, angle=0, feature_name="contrast"):
190     return [get_single_glcm_feature(im, angle, feature_name)[0][0] for im in data]
191
192 def get_edge_features_per_class(data, threshold1=30, threshold2=100):
193     return [get_edge_feature(im, threshold1, threshold2) for im in data]
194
195 def get_orb_features_per_class(data):
196     orb=cv2.ORB_create()
```

```
CPU times: user 11 µs, sys: 1 µs, total: 12 µs
Wall time: 15.3 µs
```

Model Functions

```
1 %%time
2 def cnn_classifier(X_train,
3                     y_train,
4                     X_val,
5                     y_val,
```

```
                  X_test,
                  y_test,
                  label_list = label_list,
                  number_of_hidden_layers = 2,
                  hidden_layer_size = 128,
                  learning_rate = .001,
                  dropout = .5,
                  epochs = 5):

    tf.keras.backend.clear_session()
    np.random.seed(42)
    tf.random.set_seed(42)

    layer_list = []
    for idx, node_cnt in enumerate(list(range(number_of_hidden_layers))):
        layer_list.append(layers.Dense(hidden_layer_size,
                                       activation='relu',
                                       name = 'fc_' + str(idx + 1)))
        layer_list.append(layers.Dropout(rate=dropout))

    layer_list.append(layers.Dense(5, activation='softmax', name = 'Output'))
    model = tf.keras.Sequential(layer_list)
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                  metrics=['accuracy'])
    history = model.fit(X_train,
                        y_train,
                        epochs=epochs,
                        shuffle=True,
                        validation_data=(X_val, y_val)
                       )
    hist = history.history

    model.summary()
    display(keras.utils.plot_model(model,
                           show_shapes = False,
                           show_dtype = False,
                           show_layer_names = True,
                           dpi = 90))

    x_arr = np.arange(len(hist['loss'])) + 1

    fig = plt.figure(figsize=(12, 4))
    ax = fig.add_subplot(1, 2, 1)
    ax.plot(x_arr, hist['loss'], '-o', label='Train loss')
    ax.plot(x_arr, hist['val_loss'], '--<', label='Validation loss')
    ax.legend(fontsize=15)
    ax.set_xlabel('Epoch', size=15)
    ax.set_ylabel('Loss', size=15)

    ax = fig.add_subplot(1, 2, 2)
    ax.plot(x_arr, hist['accuracy'], '-o', label='Train acc.')
    ax.plot(x_arr, hist['val_accuracy'], '--<', label='Validation acc.')
    ax.legend(fontsize=15)
    ax.set_xlabel('Epoch', size=15)
    ax.set_ylabel('Accuracy', size=15)
    plt.show()
```

```
63
64     test_results = model.evaluate(X_test, y_test)
65     print('\nTest Acc. {:.2f}%'.format(test_results[1]*100))
66
67     y_preds = model.predict(X_test)
68     y_pred = []
69     for i, p in enumerate(y_preds):
70         y_pred.append(np.argmax(p))
71
72     cm = confusion_matrix(y_test, y_pred)
73     plot_confusion_matrix(cm,
74                           label_list,
75                           "Tensorflow Confusion Matrix",
76                           saved_img_name="Tensorflow.png"
77                          )
```

```
CPU times: user 4 µs, sys: 1e+03 ns, total: 5 µs
Wall time: 8.11 µs
```

```
1 %%time
2 def cnn_image_classifier(X_train,
3                    y_train,
4                    X_val,
5                    y_val,
6                    X_test,
7                    y_test,
8                    label_list = label_list,
9                    filters1 = 32,
10                   filters2 = 16,
11                   kernel_size = (4,4),
12                   strides = (2,2),
13                   pool_size = (2,2),
14                   hidden_layer_size = 128,
15                   learning_rate = .001,
16                   dropout = .5,
17                   epochs = 5):
18
19     tf.keras.backend.clear_session()
20     np.random.seed(42)
21     tf.random.set_seed(42)
22
23     model = tf.keras.Sequential()
24     # add first convolution layer to the model
25     model.add(tf.keras.layers.Conv2D(
26     filters=filters1,
27     kernel_size=kernel_size,
28     strides=strides,
29     padding='same',
30     data_format='channels_last',
31     name='conv_1',
32     activation='relu'))
33
34     # add a max pooling layer with pool size (2,2) and strides of 2
35     # (this will reduce the spatial dimensions by half)
36     model.add(tf.keras.layers.MaxPool2D(
37     pool_size=pool_size,
38     name='pool_1'))
```

```
39
40     # add second convolutional layer
41     model.add(tf.keras.layers.Conv2D(
42     filters=filters2,
43     kernel_size=kernel_size,
44     strides=strides,
45     padding='same',
46     name='conv_2',
47     activation='relu'))
48
49     # add second max pooling layer with pool size (2,2) and strides of 2
50     # (this will further reduce the spatial dimensions by half)
51     model.add(tf.keras.layers.MaxPool2D(
52     pool_size=pool_size,
53     name='pool_2')
54     )
55
56     # add a fully connected layer (need to flatten the output of the previous layers first)
57     model.add(tf.keras.layers.Flatten())
58     model.add(tf.keras.layers.Dense(
59     units=hidden_layer_size,
60     name='fc_1',
61     activation='relu'))
62
63     # add dropout layer
64     model.add(tf.keras.layers.Dropout(
65     rate=dropout))
66
67     # add the last fully connected layer
68     # this last layer sets the activation function to "None" in order to output the logits
69     # note that passing activation = "sigmoid" will return class memembership probabilities but
70     # in TensorFlow logits are prefered for numerical stability
71     # set units=1 to get a single output unit (remember it's a binary classification problem)
72     model.add(tf.keras.layers.Dense(
73     units=5,
74     name='fc_2',
75     activation='softmax'))
76
77     model.build(input_shape=(None, 64, 64, 1))
78     model.summary()
79
80     # build model and print summary
81     tf.random.set_seed(1)
82     model.build(input_shape=(None, 64, 64, 1))
83     model.summary()
84     display(keras.utils.plot_model(model,
85                             show_shapes = False,
86                             show_dtype = False,
87                             show_layer_names = True,
88                             dpi = 90))
89
90     optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)
91
92     model.compile(loss = tf.keras.losses.SparseCategoricalCrossentropy(),
93                 optimizer = optimizer,
94                 metrics=['accuracy'])
95
```

```
    tf.random.set_seed(42)
    np.random.seed(42)
    history = model.fit(X_train,
                        y_train,
                        epochs=epochs,
                        shuffle=True,
                        validation_data=(X_val, y_val)
    )

    hist = history.history
    x_arr = np.arange(len(hist['loss'])) + 1

    fig = plt.figure(figsize=(12, 4))
    ax = fig.add_subplot(1, 2, 1)
    ax.plot(x_arr, hist['loss'], '-o', label='Train loss')
    ax.plot(x_arr, hist['val_loss'], '--<', label='Validation loss')
    ax.legend(fontsize=15)
    ax.set_xlabel('Epoch', size=15)
    ax.set_ylabel('Loss', size=15)

    ax = fig.add_subplot(1, 2, 2)
    ax.plot(x_arr, hist['accuracy'], '-o', label='Train acc.')
    ax.plot(x_arr, hist['val_accuracy'], '--<', label='Validation acc.')
    ax.legend(fontsize=15)
    ax.set_xlabel('Epoch', size=15)
    ax.set_ylabel('Accuracy', size=15)
    plt.show()

    test_results = model.evaluate(X_test, y_test)
    print('\nTest Acc. {:.2f}%'.format(test_results[1]*100))

    y_preds = model.predict(X_test)
    '''
    # transform logits to probabilities
    pred_logits = model.predict(X_test)
    probas = tf.sigmoid(pred_logits)
    probas = probas.numpy().flatten()*100
    print("Prabobilities...")
    print(probas)
    '''

    y_pred = []
    for i, p in enumerate(y_preds):
        y_pred.append(np.argmax(p))

    cm = confusion_matrix(y_test, y_pred)
    plot_confusion_matrix(cm,
                          label_list,
                          "Tensorflow Confusion Matrix",
                          saved_img_name="Tensorflow.png"
```

```
CPU times: user 8 µs, sys: 0 ns, total: 8 µs
Wall time: 11.7 µs
```

svm_classifier implements a kfold grid search. We have used the following hyperparameters:-

c: Positive regularization parameter. The C parameter tells the SVM optimization how much we want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, we should get misclassified examples, often even if our training data is linearly separable.

kernel{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf' Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used.

gamma{'scale', 'auto'} or float, default='scale' Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Gamma is a parameter that determines the width of the kernel function.

if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma, if 'auto', uses 1 / n_features if float, must be non-negative.

```
%%time
def svm_classifier(df_train, df_test, confusion_matrix_label, file_name, label_list):
    exec_stats = []
    accuracy_stats = []
    file_name_wo_ext = file_name.split(".")[0]
    for kfold, (train_indices, val_indices) in enumerate(StratifiedKFold(n_splits     = 5,
                                                                         shuffle      = True,
                                                                         random_state = 42
                                                                         ).split(df_train['labe
                                                                                 df_train['labe
                                                                                )
                                                         ):
        print("************************************************************")
        print(f"kfold : {kfold + 1}")
        sel_cols = df_train.columns[:-1]
        X_train = df_train.iloc[train_indices][sel_cols].values
        y_train = df_train.iloc[train_indices]['label'].values

        X_val = df_train.iloc[val_indices][sel_cols].values
        y_val = df_train.iloc[val_indices]['label'].values

        X_test = df_test[sel_cols].values
        y_test = df_test['label'].values

        print(X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shap
        print("****")
        scaler = StandardScaler()
        X_train_std = scaler.fit(X_train).transform(X_train)
        X_val_std = scaler.fit(X_train).transform(X_val)
        X_test_std = scaler.fit(X_train).transform(X_test)

        clf = SVC(gamma='auto', break_ties=True)
        clf.fit(X_train_std, y_train)
        y_pred_val_base = clf.predict(X_val_std)
        cm_val_base = confusion_matrix(y_val, y_pred_val_base)
```

```
        plot_confusion_matrix(cm_val_base,
                              label_list,
                              confusion_matrix_label,
                              saved_img_name=file_name_wo_ext + "_val_base_" + str(kfold + 1) +
                             )
        print(f"Validation results for fold {kfold+1}")
        print(classification_report(y_val, y_pred_val_base, target_names=label_list))

        y_pred_test_base = clf.predict(X_test_std)
        cm_test_base = confusion_matrix(y_test, y_pred_test_base)
        plot_confusion_matrix(cm_test_base,
                              label_list,
                              confusion_matrix_label,
                              saved_img_name=file_name_wo_ext + "_test_base_" + str(kfold + 1)
                             )
        print(f"Test results for fold {kfold+1}")
        print(classification_report(y_test, y_pred_test_base, target_names=label_list))

        param_grid = {'C': [1, 1000, 100000],
                      'gamma': [0.01, 0.0001, 0.000001],
                      'kernel': ['rbf']}
        grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 0)
        # fitting the model for grid search
        grid.fit(X_train_std, y_train)
        # print best parameter after tuning
        print(grid.best_params_)
        # print how our model looks after hyper-parameter tuning
        print(grid.best_estimator_)
        y_pred_val_grid = grid.predict(X_val_std)
        cm_val_grid = confusion_matrix(y_val, y_pred_val_grid)
        plot_confusion_matrix(cm_val_grid,
                              label_list,
                              confusion_matrix_label,
                              saved_img_name=file_name_wo_ext + "_val_grid_" + str(kfold + 1) +
                             )
        print(f"Validation grid search results for fold {kfold+1}")
        print(classification_report(y_val, y_pred_val_grid, target_names=label_list))

        y_pred_test_grid = grid.predict(X_test_std)
        cm_test_grid = confusion_matrix(y_test, y_pred_test_grid)
        plot_confusion_matrix(cm_test_grid,
                              label_list,
                              confusion_matrix_label,
                              saved_img_name=file_name_wo_ext + "_test_grid_" + str(kfold + 1)
                             )
        print(f"Test grid search results for fold {kfold+1}")
        print(classification_report(y_test, y_pred_test_grid, target_names=label_list))
        stats_details = [kfold + 1,
                         y_pred_val_base,
                         y_pred_test_base,
                         y_pred_val_grid,
                         y_pred_test_base,
                         cm_val_base,
                         cm_test_base,
                         cm_val_grid,
                         cm_test_grid
                        ]
```

```
        exec_stats.append(stats_details)
        accuracy = [kfold + 1,
                    accuracy_score(y_val, y_pred_val_base),
                    accuracy_score(y_test, y_pred_test_base),
                    accuracy_score(y_val, y_pred_val_grid),
                    accuracy_score(y_test, y_pred_test_grid),
                   ]
        accuracy_stats.append(accuracy)
        #return y_pred_val_base, cm_val_base, y_pred_grid, cm_grid
        pd.DataFrame(exec_stats).to_csv(file_name_wo_ext + "_exec_stats.csv", index = False)
        pd.DataFrame(accuracy_stats, columns = ['fold',
                                                'base_val_accuracy',
                                                'base_test_accuracy',
                                                'grid_val_accuracy',
                                                'grid_test_accuracy'
                                               ]).to_csv(file_name_wo_ext + "_batch_level_accu
    print("Overall Stats....")
    get_val_test_base_grid_level_stats(file_name_wo_ext + "_accuracy_stats.csv")
    print("Category Wise Stats...")
    get_category_level_stats(file_name_wo_ext + "_exec_stats.csv")
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 12.6 µs
```

```
%%time
def preprocess_and_classification_svm(df, label_list, title_name, file_name):
    X, y = df.drop(["label"], axis=1).values, df["label"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    df_train = pd.DataFrame(X_train)
    df_train['label'] = list(y_train)

    df_test = pd.DataFrame(X_test)
    df_test['label'] = list(y_test)

    svm_classifier(df_train,
                   df_test,
                   title_name,
                   file_name,
                   label_list
                  )
```

```
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 7.87 µs
```

fffsssaaa### **Plotting Functions**

```
%%time
def get_tsne_df(df):
    temp_df = copy.deepcopy(df)
    tsne = TSNE(n_components=2, verbose=0, perplexity=20, n_iter=300)
    tsne_results = tsne.fit_transform(np.array(df))
    temp_df['tsne-2d-one'] = tsne_results[:,0]
    temp_df['tsne-2d-two'] = tsne_results[:,1]
    return temp_df
```

```
 9
10 def plot_glcm_features(paper_towel_features, cardboard_features, batteries_features, plates_fea
11     fig, (ax1, ax2) = plt.subplots(1, 2)
12     fig.set_size_inches(15, 5)
13     data = [paper_towel_features, cardboard_features, batteries_features, plates_features, pape
14     bp = ax1.boxplot(data, patch_artist = True, notch ='True', vert = 0)
15
16     colors = ['#0000FF', '#00FF00', '#FFFF00', '#FF00FF', '#FF0000']
17
18     for patch, color in zip(bp['boxes'], colors):
19         patch.set_facecolor(color)
20
21     for whisker in bp['whiskers']:
22         whisker.set(color ='#8B008B', linewidth = 1.5, linestyle =":")
23
24     for cap in bp['caps']:
25         cap.set(color ='#8B008B', linewidth = 2)
26
27     for median in bp['medians']:
28         median.set(color ='red', linewidth = 3)
29
30     for flier in bp['fliers']:
31         flier.set(marker ='D', color ='#e7298a', alpha = 0.5)
32
33     ax1.set_yticklabels(['paper towel', 'cardboard', 'batteries', 'plates', 'paper'])
34
35     ax1.get_xaxis().tick_bottom()
36     ax1.get_yaxis().tick_left()
37     ax1.set_xlabel("feature value")
38     ax1.set_ylabel("class")
39
40     ax2.hist((paper_towel_features, cardboard_features, batteries_features, plates_features, pa
41              10, label = ("paper towel", "cardboard", "batteries", "plates", "paper"))
42     ax2.legend()
43     ax2.set_xlabel("feature value")
44     ax2.set_ylabel("count")
45
46     plt.suptitle("GLCM Feature: " + feature_name)
47     plt.show()
48
49 def generate_and_plot_glcm_features_for_all_classes(paper_towel_data, cardboard_data, batteries
50     paper_towel_features, cardboard_features, batteries_features, plates_features, paper_featur
51     plot_glcm_features(paper_towel_features, cardboard_features, batteries_features, plates_fea
52
53 def get_sift_feature_viz(img):
54     simplefilter=cv2.SIFT_create()
55     kp = simplefilter.detect(img, None)
56     img_kp = cv2.drawKeypoints(img, kp, None)
57     return img_kp
58
59 def plot_viz_per_class(paper_towel_viz, cardboard_viz, batteries_viz, plates_viz, paper_viz, ti
60     fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5)
61     fig.set_size_inches(15, 5)
62
63     ax1.imshow(paper_towel_viz, cmap='gray')
64     ax1.set_axis_off()
65     ax1.set_title("paper towel")
```

```
    ax2.imshow(cardboard_viz, cmap='gray')
    ax2.set_axis_off()
    ax2.set_title("cardboard")

    ax3.imshow(batteries_viz, cmap='gray')
    ax3.set_axis_off()
    ax3.set_title("batteries")

    ax4.imshow(plates_viz, cmap='gray')
    ax4.set_axis_off()
    ax4.set_title("plates")

    ax5.imshow(paper_viz, cmap='gray')
    ax5.set_axis_off()
    ax5.set_title("paper")

    fig.tight_layout()
    fig.subplots_adjust(top=1.15)
    plt.suptitle(title)
    plt.show()

def plot_edge_feature_for_each_class(paper_towel_data, cardboard_data, batteries_data, plates_c
    # TODO: if we change hyperparameters also change these values
    threshold1 = 30
    threshold2 = 100

    paper_towel_viz = get_edge_feature([paper_towel_data[0]], threshold1, threshold2)[0]
    cardboard_viz = get_edge_feature([cardboard_data[0]], threshold1, threshold2)[0]
    batteries_viz = get_edge_feature([batteries_data[0]], threshold1, threshold2)[0]
    plates_viz = get_edge_feature([plates_data[0]], threshold1, threshold2)[0]
    paper_viz = get_edge_feature([paper_data[0]], threshold1, threshold2)[0]
    title = "Edge Detection: First Image per Class"
    plot_viz_per_class(paper_towel_viz, cardboard_viz, batteries_viz, plates_viz, paper_viz, ti

def plot_hog_feature_for_each_class(paper_towel_data, cardboard_data, batteries_data, plates_da
    # TODO: if we change hyperparameters also change these values
    orientations = 9
    pixels_per_cell = (8, 8)
    cells_per_block = (2, 2)

    paper_towel_viz = get_hog_feature([paper_towel_data[0]], orientations, pixels_per_cell, cel
    cardboard_viz = get_hog_feature([cardboard_data[0]], orientations, pixels_per_cell, cells_p
    batteries_viz = get_hog_feature([batteries_data[0]], orientations, pixels_per_cell, cells_p
    plates_viz = get_hog_feature([plates_data[0]], orientations, pixels_per_cell, cells_per_blo
    paper_viz = get_hog_feature([paper_data[0]], orientations, pixels_per_cell, cells_per_block
    title = "HOG: First Image per Class"
    plot_viz_per_class(paper_towel_viz, cardboard_viz, batteries_viz, plates_viz, paper_viz, ti

def plot_sift_feature_for_each_class(paper_towel_data, cardboard_data, batteries_data, plates_c
    paper_towel_viz = get_sift_feature_viz(paper_towel_data[0])
    cardboard_viz = get_sift_feature_viz(cardboard_data[0])
    batteries_viz = get_sift_feature_viz(batteries_data[0])
    plates_viz = get_sift_feature_viz(plates_data[0])
    paper_viz = get_sift_feature_viz(paper_data[0])
    title = "SIFT: First Image per Class"
    plot_viz_per_class(paper_towel_viz, cardboard_viz, batteries_viz, plates_viz, paper_viz, ti
```

```
123
124 def plot_lbp_feature_for_each_class(paper_towel_data, cardboard_data, batteries_data, plates_da
125     # TODO: if we change hyperparameters also change these values
126     num_points = 24
127     radius = 8
128
129     paper_towel_viz = get_lbp_features([paper_towel_data[0]], num_points, radius)[0]
130     cardboard_viz = get_lbp_features([cardboard_data[0]], num_points, radius)[0]
131     batteries_viz = get_lbp_features([batteries_data[0]], num_points, radius)[0]
132     plates_viz = get_lbp_features([plates_data[0]], num_points, radius)[0]
133     paper_viz = get_lbp_features([paper_data[0]], num_points, radius)[0]
134     title = "LBP: First Image per Class"
135
136     fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, sharey=True, sharex=True)
137     fig.set_size_inches(15, 5)
138
139     ax1.hist(paper_towel_viz, 10)
140     ax1.set_xlabel("lbp value")
141     ax1.set_ylabel("count")
142     ax1.set_title("paper towel")
143
144     ax2.hist(cardboard_viz, 10)
145     ax2.set_xlabel("lbp value")
146     ax2.set_ylabel("count")
147     ax2.set_title("cardboard")
148
149     ax3.hist(batteries_viz, 10)
150     ax3.set_xlabel("lbp value")
151     ax3.set_ylabel("count")
152     ax3.set_title("batteries")
153
154     ax4.hist(plates_viz, 10)
155     ax4.set_xlabel("lbp value")
156     ax4.set_ylabel("count")
157     ax4.set_title("plates")
158
159     ax5.hist(paper_viz, 10)
160     ax5.set_xlabel("lbp value")
161     ax5.set_ylabel("count")
162     ax5.set_title("paper")
163
164     fig.tight_layout()
165     fig.subplots_adjust(top=.85)
166     plt.suptitle(title)
167     plt.show()
168
169 def plot_confusion_matrix(cm,
170                           target_names,
171                           title='Confusion matrix',
172                           saved_img_name="cm.png",
173                           cmap=None,
174                           normalize=True):
175     """
176     given a sklearn confusion matrix (cm), make a nice plot
177
178     Arguments
179     ---------
```

```
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
                  see http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion matrix created by
                                                              # sklearn.metrics.confusion_matri
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names of the classes
                          title        = best_estimator_name) # title of graph

    Citiation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
```

```
CPU times: user 9 µs, sys: 1e+03 ns, total: 10 µs
Wall time: 13.6 µs
```

Display Stats Function

```
%%time
def get_val_test_base_grid_level_stats(file_name):
    df = pd.read_csv(file_name)
    df_agg = df.agg({'base_val_accuracy' : [np.min, np.mean, np.max],
            'grid_val_accuracy' : [np.min, np.mean, np.max],
            'base_test_accuracy' : [np.min, np.mean, np.max],
            'grid_test_accuracy' : [np.min, np.mean, np.max]
           }
          ).T
    df_agg.reset_index(inplace = True)
    df_agg.columns = ['kpi_level', 'min_accuracy', 'avg_accuracy', 'max_accuracy']
    display(df_agg)

def get_category_level_stats(file_name):
    df = pd.read_csv(file_name)
    df.columns = ['kfold',
                  'y_pred_val_base',
                  'y_pred_test_base',
                  'y_pred_val_grid',
                  'y_pred_test_base',
                  'cm_val_base',
                  'cm_test_base',
                  'cm_val_grid',
                  'cm_test_grid']
    df['cm_val_base'] = df['cm_val_base'].apply(lambda x : x.replace("  ", ",").replace("\n ",
    df['cm_test_base'] = df['cm_test_base'].apply(lambda x : x.replace("  ", ",").replace("\n '
    df['cm_val_grid'] = df['cm_val_grid'].apply(lambda x : x.replace("  ", ",").replace("\n ",
    df['cm_test_grid'] = df['cm_test_grid'].apply(lambda x : x.replace("  ", ",").replace("\n '

    df['cm_val_base'] = df['cm_val_base'].apply(lambda x : "[" + x)
    df['cm_test_base'] = df['cm_test_base'].apply(lambda x : "[" + x)
    df['cm_val_grid'] = df['cm_val_grid'].apply(lambda x : "[" + x)
    df['cm_test_grid'] = df['cm_test_grid'].apply(lambda x : "[" + x)

    k_fold_stats = []
    for col in ['cm_val_base', 'cm_test_base', 'cm_val_grid', 'cm_test_grid']:
        for rec in df[['kfold', col]].values.tolist():
            kfold, cm = rec[0], rec[1]
            #print(cm)
            cm_split = cm.split("],[")
```

```
            cm_split = [e.replace("[[","[").replace("]]","]") for e in cm_split]
            cm_split = ["[" + e if "[" not in e else e for e in cm_split]
            cm_split = [e + "]" if "]" not in e else e for e in cm_split]

            battery_rec = ast.literal_eval(cm_split[0])
            cardboard_rec = ast.literal_eval(cm_split[1])
            plates_rec = ast.literal_eval(cm_split[2])
            paper_rec = ast.literal_eval(cm_split[3])
            paper_towel_rec = ast.literal_eval(cm_split[4])

            battery_correct_pct = battery_rec[0] / np.sum(battery_rec)
            cardbaord_correct_pct = cardboard_rec[1] / np.sum(cardboard_rec)
            plates_correct_pct = plates_rec[2] / np.sum(plates_rec)
            paper_correct_pct = paper_rec[3] / np.sum(paper_rec)
            paper_towel_correct_pct = paper_towel_rec[4] / np.sum(paper_towel_rec)
            k_fold_stats.append([kfold, col, battery_correct_pct,cardbaord_correct_pct,plates_c

    df_ctg_wise_stats = pd.DataFrame(k_fold_stats,
                                     columns = ['kfold',
                                                'kpi',
                                                'battery_correct_pct',
                                                'cardbaord_correct_pct',
                                                'plates_correct_pct',
                                                'paper_correct_pct',
                                                'paper_towel_correct_pct'
                                               ]
                                    )
    df_ctg_wise_stats_consolidated = df_ctg_wise_stats.groupby(['kpi']).agg({'battery_correct_p
                                                                             'cardbaord_correct
                                                                             'plates_correct_pc
                                                                             'paper_correct_pct
                                                                             'paper_towel_corre
                                                                            }
                                                                           ).reset_index()
    df_ctg_wise_stats_consolidated.columns = ['kpi',
                                              'battery_correct_pct_min',
                                              'battery_correct_pct_avg',
                                              'battery_correct_pct_max',
                                              'cardboard_correct_pct_min',
                                              'cardboard_correct_pct_avg',
                                              'cardboard_correct_pct_max',
                                              'plates_correct_pct_min',
                                              'plates_correct_pct_avg',
                                              'plates_correct_pct_max',
                                              'paper_correct_pct_min',
                                              'paper_correct_pct_avg',
                                              'paper_correct_pct_max',
                                              'paper_towel_correct_pct_min',
                                              'paper_towel_correct_pct_avg',
                                              'paper_towel_correct_pct_max',
                                             ]
    display(df_ctg_wise_stats_consolidated.T)
    ctg_file_name = file_name.split("_exec_stats.csv")[0] + "_ctg_level_stats.csv"
    df_ctg_wise_stats_consolidated.T.to_csv(ctg_file_name)

```

```
CPU times: user 7 µs, sys: 1e+03 ns, total: 8 µs
Wall time: 12.9 µs
```

NP and DF related functions

```
%%time
def flatten_np_arry(np_array):
    return np.array([np.ravel(arr) for arr in np_array])

def flatten_dataframe(df):
    return np.array([np.ravel(arr) for arr in np.array(df.values)])

def flatten_and_normalize_np_array(np_image_list):
    return [np.ravel(img) / 255 for img in np_image_list]

def standard_scaler_train_test_array(x_train, x_test):
    scaler = StandardScaler()
    scaler.fit(x_train)
    X_train_sc = scaler.transform(x_train)
    X_test_sc = scaler.transform(x_test)
    return X_train_sc, X_test_sc

def create_feature_df(feature_array, label):
    df_feature = flatten_list_and_make_df(feature_array)
    df_feature['label'] = label
    return df_feature

def combine_dfs(df_battery, df_cardboard, df_plates, df_paper, df_paper_towel):
    df_combined = pd.concat([df_battery,
                             df_cardboard,
                             df_plates,
                             df_paper,
                             df_paper_towel
                            ],
                            axis = 0)
    return df_combined

def get_category_mean_std(battery_data, cardboard_data, plates_data, paper_data, paper_towel_da
    """
    This function calculates category wise mean and standard deviation.
    """
    ctg_chnl_stats = []

    battery_avg, battery_std = np.mean(battery_data), np.std(battery_data)
    ctg_chnl_stats.append(('battery', battery_avg, battery_std))

    cardboard_avg, cardboard_std = np.mean(cardboard_data), np.std(cardboard_data)
    ctg_chnl_stats.append(('cardboard', cardboard_avg, cardboard_std))

    plates_avg, plates_std = np.mean(plates_data), np.std(plates_data)
    ctg_chnl_stats.append(('plates', plates_avg, plates_std))

    paper_avg, paper_std = np.mean(paper_data), np.std(paper_data)
    ctg_chnl_stats.append(('paper', paper_avg, paper_std))

    paper_towel_avg, paper_towel_std = np.mean(paper_towel_data), np.std(paper_towel_data)
```

```
    ctg_chnl_stats.append(('paper_towel', paper_towel_avg, paper_towel_std))

    return pd.DataFrame(ctg_chnl_stats, columns = ['category', 'category_avg', 'category_std'])
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 7.87 µs
```

Edge detection functions

```
%%time
threshold1 = 30
threshold2= 100
size = 9
std = 1.5

def create_2d_gaussian(size=size, std=std):
    gaussian_1d = signal.gaussian(size,std=std)
    gaussian_2d = np.outer(gaussian_1d, gaussian_1d)
    gaussian_2d = gaussian_2d/(gaussian_2d.sum())
    return gaussian_2d

def remove_texture(image):
    low_pass_gaussain = create_2d_gaussian(15, 8)
    img_less_texture = convolve(image, low_pass_gaussain)
    return img_less_texture

def get_edge_feature(np_img_list, threshold1, threshold2):
    return [cv2.Canny(remove_texture(remove_texture(img)), threshold1=threshold1, threshold2=th

def get_prewitth_kernel_feature(np_img_list):
    return [prewitt_h(img) for img in np_img_list]

def get_prewittv_kernel_feature(np_img_list):
    return [prewitt_v(img) for img in np_img_list]

def get_canny_feature(np_img_list):
    return [cv2.Canny(img, 100, 200) for img in np_img_list]

def get_sobel_feature(np_img_list):
    return [filters.sobel(img) for img in np_img_list]

def get_contour_feature(np_img_list):
    return [np.array(Image.fromarray(np.uint8(img)).filter(ImageFilter.CONTOUR)) for img in np_

def get_edge1_feature(np_img_list):
    return [np.array(Image.fromarray(np.uint8(img)).filter(ImageFilter.EDGE_ENHANCE)) for img i

def get_edge2_feature(np_img_list):
    return [np.array(Image.fromarray(np.uint8(img)).filter(ImageFilter.EDGE_ENHANCE_MORE)) for

```

```
CPU times: user 11 µs, sys: 0 ns, total: 11 µs
Wall time: 14.1 µs
```

Exploratory Data Analysis

Original image counts in the directory

```
%%time
folder_list = ['batteries',
               'cardboard',
               'disposable_plates',
               'paper',
               'paper_towel'
              ]
current_path = os.getcwd()
for folder in folder_list:
    if not os.path.exists(os.path.join(current_path, folder)):
        os.makedirs(os.path.join(current_path, folder))
```

```
CPU times: user 1.39 ms, sys: 39 µs, total: 1.43 ms
Wall time: 1.04 ms
```

```
!mv *.jpg /content/paper/
```

```
!echo "Count of batteries images"
!ls -ltr /content/batteries/*.jpg|wc -l

!echo "Count of cardboard images"
!ls -ltr /content/cardboard/*.jpg|wc -l

!echo "Count of disposable plates images"
!ls -ltr /content/disposable_plates/*.jpg|wc -l

!echo "Count of paper towel images"
!ls -ltr /content/paper_towel/*.jpg|wc -l

!echo "Count of paper images"
!ls -ltr /content/paper/*.jpg|wc -l
```

```
Count of batteries images
297
Count of cardboard images
274
Count of disposable plates images
273
Count of paper towel images
291
Count of paper images
255
```

Preprocessing

- We read the input color image from directory as background-removed resized gray scale images.
- We normalize the images.
- We remove duplicate images using dbscan.

```
%%time
### Loading data
batteries_data = preprocessing(batteries_image_dir_path)
cardboard_data = preprocessing(cardboard_image_dir_path)
plates_data = preprocessing(disposable_plates_image_dir_path)
paper_data = preprocessing(paper_image_dir_path)
paper_towel_data = preprocessing(paper_towel_image_dir_path)

label_batteries = np.zeros(len(batteries_data))
label_cardbaord = np.ones(len(cardboard_data))
label_plates = np.ones(len(plates_data)) * 2
label_paper = np.ones(len(paper_data)) * 3
label_paper_towel = np.ones(len(paper_towel_data)) * 4
label = np.concatenate((label_batteries,
                        label_cardbaord,
                        label_plates,
                        label_paper,
                        label_paper_towel
                       ))
```

```
Downloading data from 'https://github.com/danielgatis/rembg/releases/download/v0
100%|████████████████████████████████████████| 176M/176M [00:00<00:00, 110GB/s]
['/content/batteries/085_fba5f889.jpg', '/content/batteries/201_0aa394fe.jpg', '
15
['/content/cardboard/104_48e76d41.jpg', '/content/cardboard/126_6366e1e4.jpg', '
15
['/content/disposable_plates/100_d950d7e1.jpg', '/content/disposable_plates/198_
14
['/content/paper/293_15158a9d.jpg', '/content/paper/089_4d404e68.jpg', '/content
43
['/content/paper_towel/096_158eb870.jpg', '/content/paper_towel/196_f601b355.jpg
23
CPU times: user 1h 53s, sys: 7min 11s, total: 1h 8min 5s
Wall time: 26min 20s
```

Data Split

We split the train and test in 80:20 ratio. For k-fold cross validation, we split the train data in train and validation and for each k-fold we tested the performance on the same test set to prevent the data leakage.

```
%%time
raw_image = np.concatenate((flatten_np_arry(batteries_data),
                            flatten_np_arry(cardboard_data),
                            flatten_np_arry(plates_data),
                            flatten_np_arry(paper_data),
                            flatten_np_arry(paper_towel_data)
                           )
                          )
shuffle = np.random.permutation(np.arange(len(raw_image)))

raw_image_orig = np.concatenate((batteries_data,
                            cardboard_data,
                            plates_data,
                            paper_data,
                            paper_towel_data
                           )
                          )
shuffle = np.random.permutation(np.arange(len(raw_image_orig)))

X_orig, y_orig = raw_image_orig[shuffle], label[shuffle]
X_orig_train, X_orig_test, y_orig_train, y_orig_test = train_test_split(X_orig,
                                                                        y_orig,
                                                                        test_size=0.2,
                                                                        random_state=42)

split = (0.9,0.1) #70% training and 30% test
splits = np.multiply(len(y_orig_train), split).astype(int)
X_orig_train, X_orig_val = np.split(X_orig_train, [splits[0]])
y_orig_train, y_orig_val = np.split(y_orig_train, [splits[0]])

X_orig_train = X_orig_train / 255
X_orig_val = X_orig_val / 255
X_orig_test = X_orig_test / 255

X, y = raw_image[shuffle], label[shuffle]
#raw_data = np.concatenate((X,y))
df_raw = pd.DataFrame(X)
df_raw['label'] = y
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)
df_train = pd.DataFrame(X_train)
df_train['label'] = y_train
df_test = pd.DataFrame(X_test)
df_test['label'] = y_test
```

```
CPU times: user 28.4 ms, sys: 47 µs, total: 28.5 ms
Wall time: 28.8 ms
```

```
%%time
df_ctg_stats = get_category_mean_std(batteries_data,
                                     cardboard_data,
                                     plates_data,
                                     paper_data,
                                     paper_towel_data
```

```
7                                   )
8 df_ctg_stats
```

```
CPU times: user 23.5 ms, sys: 1.05 ms, total: 24.5 ms
Wall time: 27.3 ms
```

	category	category_avg	category_std
0	battery	50.712640	64.218842
1	cardboard	55.613524	69.289230
2	plates	74.847136	80.147345
3	paper	48.056433	71.396708
4	paper_towel	63.753258	83.072535

Printing image counts after preprocessing

```
1 %%time
2 print(f"Count of batteries data : {len(batteries_data)}")
3 print(f"Count of cardboard data : {len(cardboard_data)}")
4 print(f"Count of plates data : {len(plates_data)}")
5 print(f"Count of paper data : {len(paper_data)}")
6 print(f"Count of paper towel data : {len(paper_towel_data)}")
```

```
Count of batteries data : 282
Count of cardboard data : 259
Count of plates data : 259
Count of paper data : 212
Count of paper towel data : 268
CPU times: user 270 µs, sys: 32 µs, total: 302 µs
Wall time: 221 µs
```

Principal Component Analysis(PCA)

PCA is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Here we have started with varied size images, which we have transformed to 64 X 64 grayscale images. However, each falttened image is still having quite high dimension of 4096 (Square of 64). Althoughh, 95% of the variance can be explained by much smaller size without losing much information through PCA. That will allow faster computation with less computational resources.

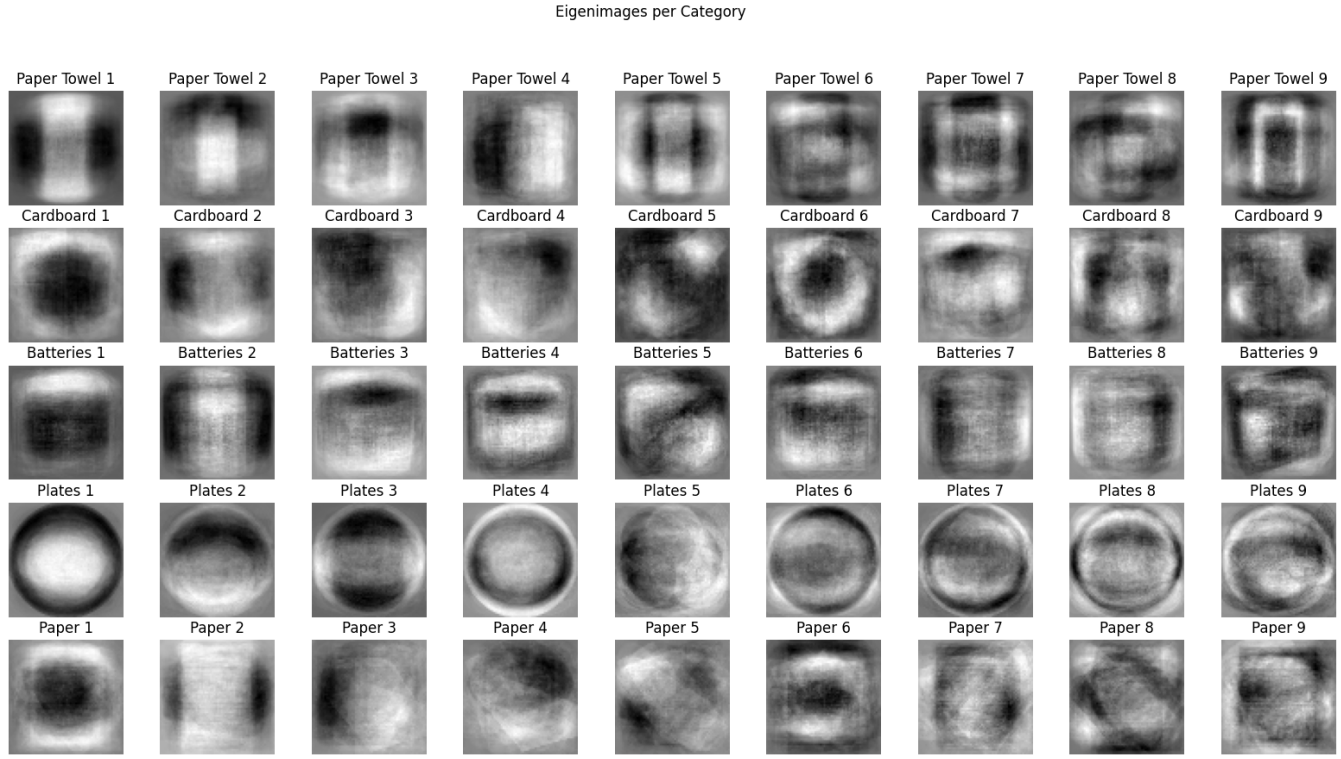
We see the following reductions that are achieved by PCA application in individual categories:-

- Battery : 181
- Cardboard : 114
- Disposable Plates : 105

- Paper : 69
- Paper Towel : 119

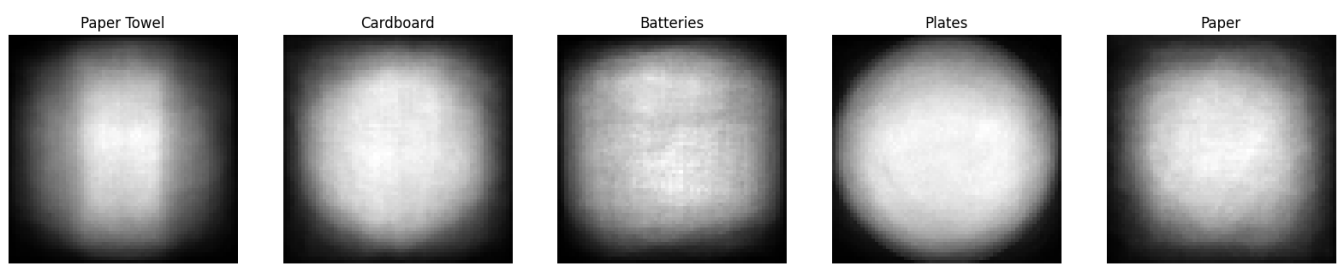
```
%%time
show_all_eigenimages(paper_towel_data, cardboard_data, batteries_data, plates_data, paper_data)
```

```
Number of principal components explaining 95% of variance: 119
Number of principal components explaining 95% of variance: 114
Number of principal components explaining 95% of variance: 181
Number of principal components explaining 95% of variance: 105
Number of principal components explaining 95% of variance: 69
CPU times: user 5.57 s, sys: 9.36 s, total: 14.9 s
Wall time: 1.6 s
```




```
%%time
### Showing average images
show_all_average_images(paper_towel_data, cardboard_data, batteries_data, plates_data, paper_da
```

```
CPU times: user 87.1 ms, sys: 3.24 ms, total: 90.3 ms
Wall time: 86.5 ms
```



Classification

In order to maximize model predictive performance and generalizability and ensure no data leakage during the construction of our model pipeline, we made sure to: 1) Apply the same preprocessing to all images in the cases where the preprocessing was learned, and used appropriate fit/transform methods. 2) We used proper procedures for train/validation/test splits and made sure that our models did not see the test data until the final evaluation. 3) PCA was to reduce cardinality and improve model performance. 4) We used GridSearch CV to tune model hyperparameters. 5) We implemented K-Fold Validation to make sure that our model was generalizable and not scoring well due to chance. 6) Finally, we tuned the hyperparameters of our features to ensure we were getting the most power from them.

Dummy Classifier

We began the experiment by establishing our baseline models. For this step, we created a dummy classifier to determine the minimum level our model has to perform to say it is better than chance. This model labels all test data as the most frequent class and sees the accuracy of that approach. This can be thought of as the "blind luck" approach. Here we had 22% accuracy.

```
dummy_clf = DummyClassifier(strategy="most_frequent")
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(dummy_clf, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
#dummy_clf.fit(X, y)
#dummy_clf.predict(X)
#dummy_clf.score(X, y)
scores
```

```
array([0.2265625, 0.2265625, 0.21875  , 0.21875  , 0.21875  , 0.21875  ,
       0.21875  , 0.21875  , 0.21875  , 0.21875  , 0.2265625, 0.2265625,
       0.21875  , 0.21875  , 0.21875  , 0.21875  , 0.21875  , 0.21875  ,
       0.21875  , 0.21875  , 0.2265625, 0.2265625, 0.21875  , 0.21875  ,
       0.21875  , 0.21875  , 0.21875  , 0.21875  , 0.21875  , 0.21875  ])
```

Model Construction

We tested many different model types during our experimentation, including logistic regression models and CNNs. However, the best-performing model was Support Vector Machines. With an average accuracy of 60.8% during our K Fold validation. The battery and disposal plate classes had the highest accuracy, as the images in these classes had the least variety in shape and texture. For this experiment, we weighted the accuracy of the Battery class highest, as a model that is accurate for battery classification is critical for our use case, as having a battery misclassified as another

form of recycling, would have dire consequences for the recycling process. If any of the others were misclassified, for example, a paper plate was classified as cardboard, it would not impact how the item is handled at the recycling center.

Baseline SVM Classifier

Here we are runnning the baseline SVM on resized, grayscale, background removed images without any other feature extraction.

[ ] ↳ *1 cell hidden*

PCA and Classification on Reconstructed Images

Next, we wanted to see how our classifier would perform without our features. We ran the model using the preprocessed images without feature extraction or hyperparameter tuning. This model performed with a 52.7% accuracy.

```
%%time
reconstructed_batteries_data, batteries_pca_component_count_list = apply_pca_and_reconstruct_im
reconstructed_cardboard_data, cardboard_pca_component_count_list = apply_pca_and_reconstruct_im
reconstructed_plates_data, plates_pca_component_count_list = apply_pca_and_reconstruct_image(pl
reconstructed_paper_data, paper_pca_component_count_list = apply_pca_and_reconstruct_image(pape
reconstructed_paper_towel_data, paper_towel_pca_component_count_list = apply_pca_and_reconstruc

raw_reconstructed_image = np.concatenate((flatten_np_arry(reconstructed_batteries_data),
                                          flatten_np_arry(reconstructed_cardboard_data),
                                          flatten_np_arry(reconstructed_plates_data),
                                          flatten_np_arry(reconstructed_paper_data),
                                          flatten_np_arry(reconstructed_paper_towel_data)
                                         )
                                        )
X_recon, y_recon = raw_reconstructed_image[shuffle], label[shuffle]
#raw_data = np.concatenate((X,y))
df_raw_recon = pd.DataFrame(X_recon)
df_raw_recon['label'] = y_recon

preprocess_and_classification_svm(df_raw_recon,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with Raw Reconstruc
                                  "SVM_Raw_PCA_Image.png"
                                 )
```

```
************************************************************
kfold : 1
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.60      0.69      0.64        45
        Cardboard       0.30      0.32      0.31        41
Disposable Plates       0.95      0.46      0.62        41
            Paper       0.48      0.39      0.43        33
     Paper Towels       0.54      0.76      0.63        45

         accuracy                           0.54       205
        macro avg       0.57      0.52      0.53       205
     weighted avg       0.58      0.54      0.53       205

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.61      0.64      0.62        58
        Cardboard       0.29      0.26      0.27        53
Disposable Plates       0.74      0.50      0.60        52
            Paper       0.41      0.29      0.34        49
     Paper Towels       0.39      0.68      0.50        44

         accuracy                           0.47       256
        macro avg       0.49      0.47      0.47       256
     weighted avg       0.49      0.47      0.47       256

{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1, gamma=0.0001)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.53      0.67      0.59        45
        Cardboard       0.30      0.34      0.32        41
Disposable Plates       0.78      0.51      0.62        41
            Paper       0.40      0.36      0.38        33
     Paper Towels       0.58      0.58      0.58        45

         accuracy                           0.50       205
        macro avg       0.52      0.49      0.50       205
     weighted avg       0.52      0.50      0.51       205

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.54      0.60      0.57        58
        Cardboard       0.24      0.23      0.24        53
Disposable Plates       0.64      0.58      0.61        52
            Paper       0.39      0.33      0.36        49
     Paper Towels       0.44      0.55      0.49        44

         accuracy                           0.46       256
        macro avg       0.45      0.46      0.45       256
```

```
weighted avg       0.45      0.46      0.45       256
```

Basic CNN

```
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
```

In Convolutional neural network, the kernel is nothing but a filter that is used to extract the features from the images. The kernel is a matrix that moves over the input data, performs the dot product with the sub-region of input data, and gets the output as the matrix of dot products. Kernel moves on the input data by the stride value. If the stride value is 2, then kernel moves by 2 columns of pixels in the input matrix. In short, the kernel is used to extract high-level features like edges from the image.

Learning rate is the rate of descent for moving towards the global minima.

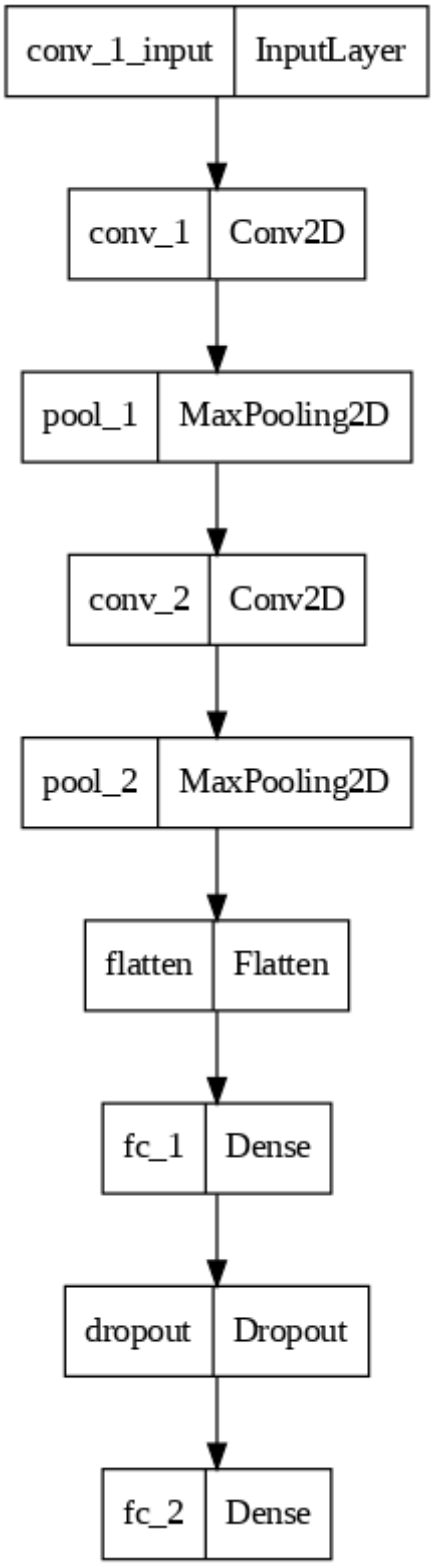
Epoch is the number of passes a training dataset takes around an algorithm.

```
%%time
param_dict = [
               {'filters1' : 4,
                'filters2' : 8,
                'kernel_size' : (2,2),
                'strides' : (2,2),
                'pool_size' : (2,2),
                'hidden_layer_size' : 256,
                'learning_rate' : .001,
                'dropout' : .5,
                'epochs' : 5
               },
               {'filters1' : 4,
                'filters2' : 8,
                'kernel_size' : (2,2),
                'strides' : (2,2),
                'pool_size' : (2,2),
                'hidden_layer_size' : 128,
                'learning_rate' : .001,
                'dropout' : .5,
                'epochs' : 5
               }
              ]
for idx, param in enumerate(param_dict):

      filters1 = param['filters1']
      filters2 = param['filters2']
      kernel_size = param['kernel_size']
      strides = param['strides']
      pool_size = param['pool_size']
      hidden_layer_size = param['hidden_layer_size']
      learning_rate = param['learning_rate']
      dropout = param['dropout']
      epochs = param['epochs']
      print("****************************")
      print(f"iteration : {idx + 1}")
      print(f"filters1 : {filters1}")
      print(f"filters2 : {filters2}")
```

```
print(f"filters2 : {filters2}")
print(f"kernel_size : {kernel_size}")
print(f"strides : {strides}")
print(f"pool_size : {pool_size}")
print(f"hidden_layer_size : {hidden_layer_size}")
print(f"learning_rate : {learning_rate}")
print(f"dropout : {dropout}")
print(f"epochs : {epochs}")

cnn_image_classifier(X_orig_train,
          y_orig_train,
          X_orig_val,
          y_orig_val,
          X_orig_test,
          y_orig_test,
          label_list = label_list,
          filters1 = filters1,
          filters2 = filters2,
          kernel_size = kernel_size,
          strides = strides,
          pool_size = pool_size,
          hidden_layer_size = hidden_layer_size,
          learning_rate = learning_rate,
          dropout = dropout,
          epochs = epochs
          )
```

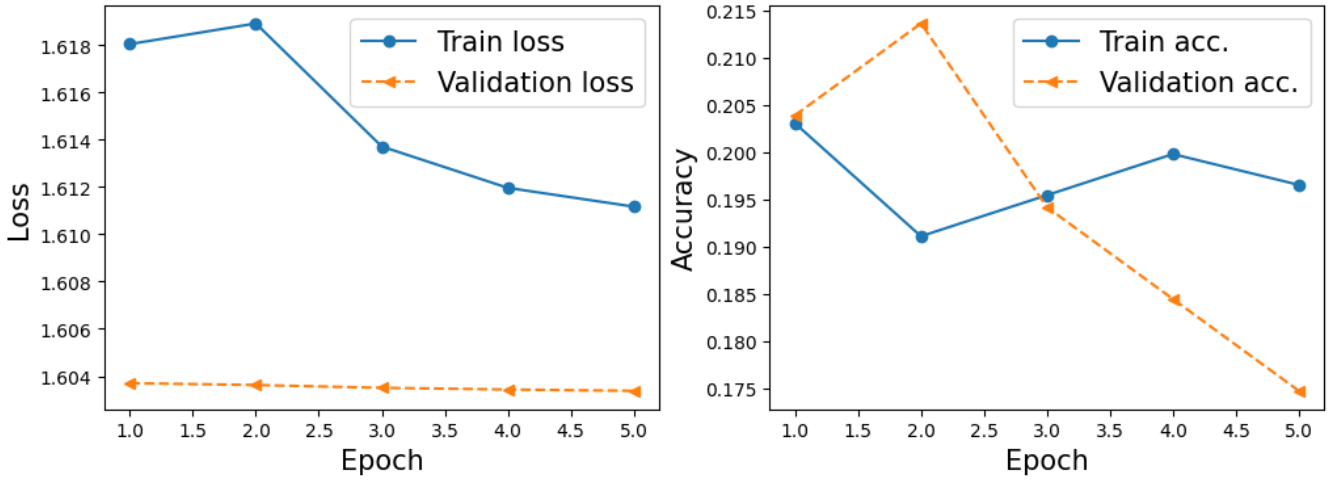
```
****************************
iteration : 1
filters1 : 4
filters2 : 8
kernel_size : (2, 2)
strides : (2, 2)
pool_size : (2, 2)
hidden_layer_size : 256
learning_rate : 0.001
dropout : 0.5
epochs : 5
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_1 (Conv2D)             (None, 32, 32, 4)         20

 pool_1 (MaxPooling2D)       (None, 16, 16, 4)         0

 conv_2 (Conv2D)             (None, 8, 8, 8)           136

 pool_2 (MaxPooling2D)       (None, 4, 4, 8)           0

 flatten (Flatten)           (None, 128)               0

 fc_1 (Dense)                (None, 256)               33024

 dropout (Dropout)           (None, 256)               0

 fc_2 (Dense)                (None, 5)                 1285

=================================================================
Total params: 34,465
Trainable params: 34,465
Non-trainable params: 0
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_1 (Conv2D)             (None, 32, 32, 4)         20

 pool_1 (MaxPooling2D)       (None, 16, 16, 4)         0

 conv_2 (Conv2D)             (None, 8, 8, 8)           136

 pool_2 (MaxPooling2D)       (None, 4, 4, 8)           0

 flatten (Flatten)           (None, 128)               0

 fc_1 (Dense)                (None, 256)               33024

 dropout (Dropout)           (None, 256)               0

 fc_2 (Dense)                (None, 5)                 1285
```

```
=================================================================
Total params: 34,465
Trainable params: 34,465
Non-trainable params: 0
_________________________________________________________________
```




```
Epoch 1/5
29/29 [==============================] - 10s 14ms/step - loss: 1.6180 - accuracy
Epoch 2/5
29/29 [==============================] - 0s 4ms/step - loss: 1.6189 - accuracy:
Epoch 3/5
29/29 [==============================] - 0s 5ms/step - loss: 1.6137 - accuracy:
```

```
Epoch 4/5
29/29 [==============================] - 0s 5ms/step - loss: 1.6120 - accuracy:
Epoch 5/5
29/29 [==============================] - 0s 5ms/step - loss: 1.6112 - accuracy:
```



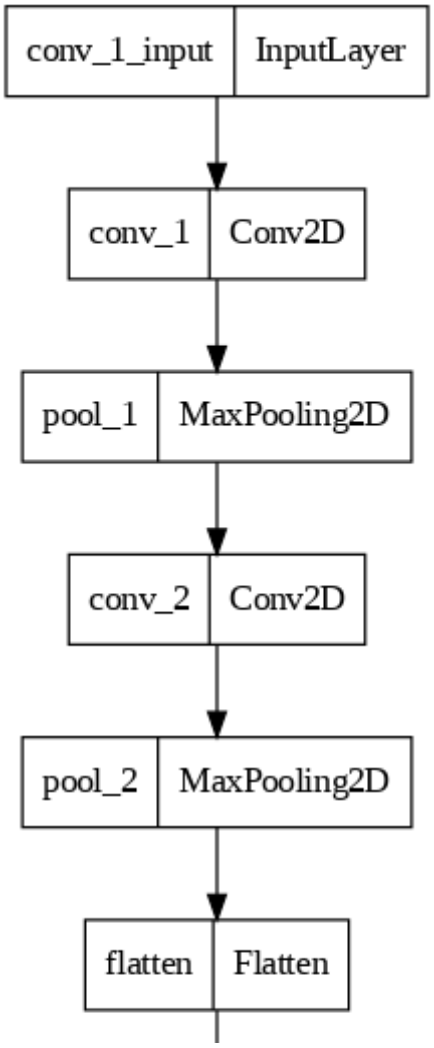

```
8/8 [==============================] - 0s 3ms/step - loss: 1.6166 - accuracy: 0.

Test Acc. 18.75%
8/8 [==============================] - 0s 2ms/step
*****************************
iteration : 2
filters1 : 4
filters2 : 8
kernel_size : (2, 2)
strides : (2, 2)
pool_size : (2, 2)
hidden_layer_size : 128
learning_rate : 0.001
dropout : 0.5
epochs : 5
Model: "sequential"
```

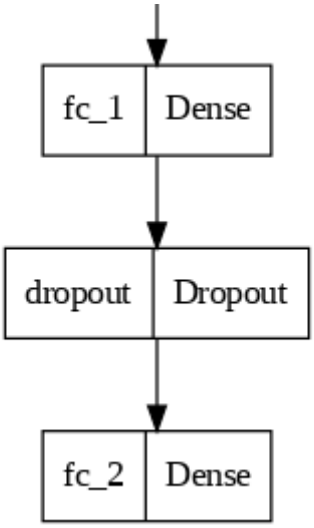
Layer (type)	Output Shape	Param #
conv_1 (Conv2D)	(None, 32, 32, 4)	20
pool_1 (MaxPooling2D)	(None, 16, 16, 4)	0
conv_2 (Conv2D)	(None, 8, 8, 8)	136
pool_2 (MaxPooling2D)	(None, 4, 4, 8)	0
flatten (Flatten)	(None, 128)	0
fc_1 (Dense)	(None, 128)	16512
dropout (Dropout)	(None, 128)	0
fc_2 (Dense)	(None, 5)	645

```
Total params: 17,313
Trainable params: 17,313
Non-trainable params: 0
```

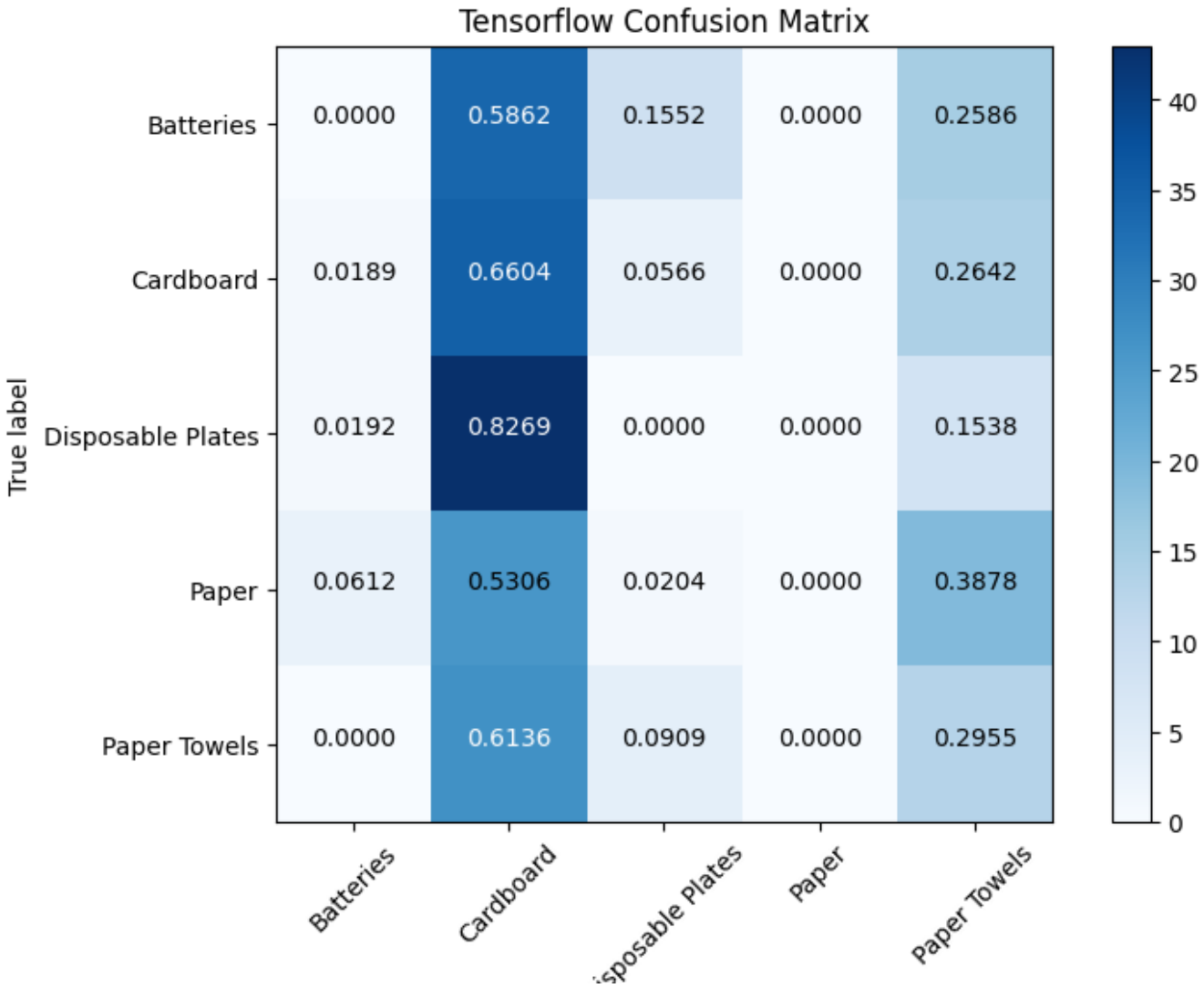
Non-trainable params: 0

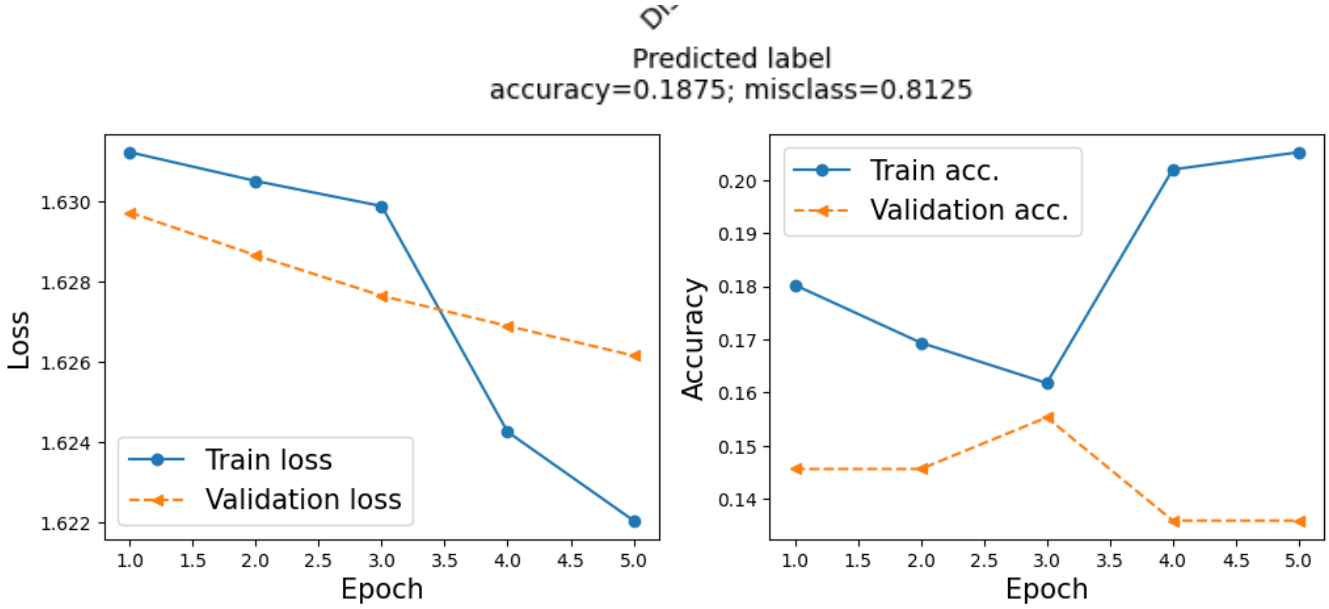
```
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_1 (Conv2D)             (None, 32, 32, 4)         20

 pool_1 (MaxPooling2D)       (None, 16, 16, 4)         0

 conv_2 (Conv2D)             (None, 8, 8, 8)           136

 pool_2 (MaxPooling2D)       (None, 4, 4, 8)           0

 flatten (Flatten)           (None, 128)               0

 fc_1 (Dense)                (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 fc_2 (Dense)                (None, 5)                 645

=================================================================
Total params: 17,313
Trainable params: 17,313
Non-trainable params: 0
_________________________________________________________________
```



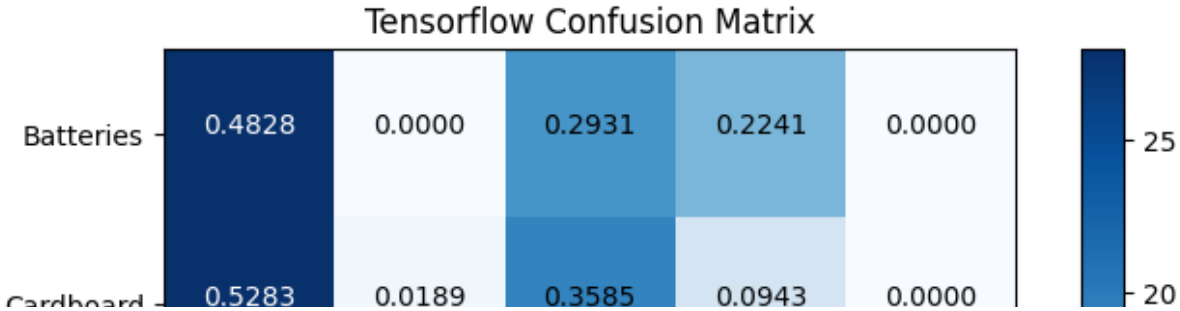



```
Epoch 1/5
29/29 [==============================] - 1s 11ms/step - loss: 1.6312 - accuracy:
Epoch 2/5
29/29 [==============================] - 0s 5ms/step - loss: 1.6305 - accuracy:
Epoch 3/5
29/29 [==============================] - 0s 5ms/step - loss: 1.6299 - accuracy:
Epoch 4/5
29/29 [==============================] - 0s 4ms/step - loss: 1.6242 - accuracy:
Epoch 5/5
29/29 [==============================] - 0s 5ms/step - loss: 1.6220 - accuracy:
```






```
8/8 [==============================] - 0s 3ms/step - loss: 1.6115 - accuracy: 0.

Test Acc. 22.66%
8/8 [==============================] - 0s 2ms/step
CPU times: user 9.76 s, sys: 4.67 s, total: 14.4 s
Wall time: 18.6 s
```



PCA and NN with Reconstructed Data

[ ] ↳ *1 cell hidden*

Feature Extraction

Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups.

Gray Level Coocurrence Matrix (GLCM)

The GLCM method is a image texture analysis method. Our dataset contains crumpled aluminium foil, paper towel and paper and we think texture information will be helpful to identify them properly. The GLCM functions characterize the texture of an image by calculating how often pairs of pixel

with specific values and in a specified spatial relationship occur in an image, creating a GLCM, and then extracting statistical measures from this matrix.

Texture can be seen as a repetition of visual patterns. Using an appropriate positional operator, it makes sense that GLCMs can be used to extract information about textures present in images. After all, a repetition of visual patterns can be seen as a repetition of combinations of values with a certain orientation.

GLCM matrix provides the below below information:-

Contrast: Measures the local variations in the gray-level co-occurrence matrix.

Correlation: Measures the joint probability occurrence of the specified pixel pairs.

Energy: Provides the sum of squared elements in the GLCM. Also known as uniformity or the angular second moment.

Homogeneity: Measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal.

```
%%time
def create_feature(feature_function, batteries_data, cardboard_data, plates_data, paper_data, p
    return (feature_function(batteries_data),
            feature_function(cardboard_data),
            feature_function(plates_data),
            feature_function(paper_data),
            feature_function(paper_towel_data))

def flatten_list_and_make_df(np_image_list):
    return pd.DataFrame([np.ravel(img) for img in np_image_list])

def create_dataframe_for_feature_with_flattening(feature_extraction_function, batteries_data, c
    battery_feature, cardboard_feature, plates_feature, paper_feature, paper_towel_feature = cr

    df_battery = flatten_list_and_make_df(battery_feature)
    df_battery['label'] = 0

    df_cardboard = flatten_list_and_make_df(cardboard_feature)
    df_cardboard['label'] = 1

    df_plates = flatten_list_and_make_df(plates_feature)
    df_plates['label'] = 2

    df_paper = flatten_list_and_make_df(paper_feature)
    df_paper['label'] = 3

    df_paper_towel = flatten_list_and_make_df(paper_towel_feature)
    df_paper_towel['label'] = 4

    df = pd.concat([df_battery, df_cardboard, df_plates, df_paper, df_paper_towel], axis = 0)
    return df

def create_dataframe_for_feature(feature_extraction_function, batteries_data, cardboard_data, p
```

```
    battery_feature, cardboard_feature, plates_feature, paper_feature, paper_towel_feature = cr

    battery_feature['label'] = 0
    cardboard_feature['label'] = 1
    plates_feature['label'] = 2
    paper_feature['label'] = 3
    paper_towel_feature['label'] = 4

    df = pd.concat([battery_feature,
                    cardboard_feature,
                    plates_feature,
                    paper_feature,
                    paper_towel_feature
                   ],
                   ignore_index=True)
    return df
```

```
CPU times: user 0 ns, sys: 6 µs, total: 6 µs
Wall time: 10.7 µs
```

```
%%time
df_glcm_feature = create_dataframe_for_feature(get_glcm_features,
                                               batteries_data,
                                               cardboard_data,
                                               plates_data,
                                               paper_data,
                                               paper_towel_data
                                              )
print(df_glcm_feature.shape)
df_glcm_feature.head()
```

```
(1280, 25)
CPU times: user 49.3 s, sys: 48.6 ms, total: 49.4 s
Wall time: 49.2 s
```

	glcm_contrast_dist_1_angle_0	glcm_contrast_dist_1_angle_45	glcm_contrast_di
0	2100.792163	3726.951877	
1	702.685020	1029.450743	
2	711.914435	1194.571177	
3	491.519593	1353.677501	
4	2481.406994	3432.188209	

5 rows × 25 columns

```
preprocess_and_classification_svm(df_glcm_feature,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with GLCM Features'
```

```
4                                    "SVM GLCM Image.png"
```

```
************************************************************
kfold : 1
(819, 24) (819,) (205, 24) (205,) (256, 24) (256,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.74      0.76      0.75        45
        Cardboard       0.44      0.57      0.50        40
Disposable Plates       0.64      0.56      0.60        41
            Paper       0.74      0.49      0.59        35
     Paper Towels       0.54      0.59      0.57        44

         accuracy                           0.60       205
        macro avg       0.62      0.59      0.60       205
     weighted avg       0.62      0.60      0.60       205

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.65      0.76      0.70        58
        Cardboard       0.39      0.39      0.39        59
Disposable Plates       0.54      0.49      0.51        55
            Paper       0.52      0.32      0.40        37
     Paper Towels       0.39      0.47      0.43        47

         accuracy                           0.50       256
        macro avg       0.50      0.49      0.49       256
     weighted avg       0.50      0.50      0.49       256

{'C': 100000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=100000, gamma=0.0001)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.65      0.73      0.69        45
        Cardboard       0.49      0.45      0.47        40
Disposable Plates       0.60      0.66      0.63        41
            Paper       0.58      0.40      0.47        35
     Paper Towels       0.65      0.70      0.67        44

         accuracy                           0.60       205
        macro avg       0.59      0.59      0.59       205
     weighted avg       0.60      0.60      0.59       205

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.67      0.86      0.75        58
        Cardboard       0.48      0.37      0.42        59
Disposable Plates       0.54      0.60      0.57        55
            Paper       0.56      0.38      0.45        37
     Paper Towels       0.55      0.57      0.56        47

         accuracy                           0.57       256
        macro avg       0.56      0.56      0.55       256
```

```
weighted avg       0.56      0.57      0.56       256
```

Local Binary Pattern(LBP)

LBP is another popular texture descriptor of the image based on the neighborhood for any given pixel. LBP is excellent to differentiate tiny differences in texture and topography, to identify key features with which we can then differentiate between images of the same type — no painstaking labelling required. The goal of LBP is to encode geometric features of an image by detecting edges, corners, raised or flat areas and hard lines; allowing us to generate a feature vector representation of an image, or group of images.

By encoding commonalities between images or of a single image of a given unknown class, we allow for comparison of their features, with that of another image. Through this comparison, we can determine the level of similarity between our target representation and an unseen image and can calculate the probability that the image presented is of the same variety or type as the target image.

Again, with our present dataset, where we observe presence of texture information in paper, paper twoel, disposable plates and aluminium, we decided to use LBP features as part of our feature extraction process.

```
Paper Towels       0.41      0.51      0.46        47
```

```
%%time
df_lbp_feature = pd.DataFrame()
hist_batteries, lbp_batteries = get_lbp_features(batteries_data)
hist_cardboard, lbp_cardboard = get_lbp_features(cardboard_data)
hist_plates, lbp_plates = get_lbp_features(plates_data)
hist_paper, lbp_paper = get_lbp_features(paper_data)
hist_paper_towel, lbp_paper_towel = get_lbp_features(paper_towel_data)

hist_columns = ['lbp_feature_histogram_' + str(idx + 1) for idx in range(hist_batteries[0].shap
df_lbp_feature_battery = pd.DataFrame(hist_batteries, columns = hist_columns)
df_lbp_feature_battery['label'] = 0

df_lbp_feature_cardboard = pd.DataFrame(hist_cardboard, columns = hist_columns)
df_lbp_feature_cardboard['label'] = 1

df_lbp_feature_plates = pd.DataFrame(hist_plates, columns = hist_columns)
df_lbp_feature_plates['label'] = 2

df_lbp_feature_paper = pd.DataFrame(hist_paper, columns = hist_columns)
df_lbp_feature_paper['label'] = 3

df_lbp_feature_paper_towel = pd.DataFrame(hist_paper_towel, columns = hist_columns)
df_lbp_feature_paper_towel['label'] = 4

df_lbp_feature = pd.concat([df_lbp_feature_battery,
                            df_lbp_feature_cardboard,
                            df_lbp_feature_plates,
                            df_lbp_feature_paper,
                            df_lbp_feature_paper_towel],
```

```
                          ignore_index = True)
print(df_lbp_feature.shape)
df_lbp_feature.head()
```

```
(1280, 53)
CPU times: user 4.45 s, sys: 11.3 ms, total: 4.46 s
Wall time: 4.45 s
```

	lbp_feature_histogram_1	lbp_feature_histogram_2	lbp_feature_histogram_3	lbp
0	0.067383	0.030273	0.008301	
1	0.071045	0.018311	0.006348	
2	0.059814	0.033936	0.017578	
3	0.058594	0.024658	0.011719	
4	0.060547	0.037598	0.006592	

5 rows × 53 columns

```
preprocess_and_classification_svm(df_lbp_feature,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with LBP Features",
                                  "SVM_LBP_Image.png"
                                 )
```

```
************************************************************
kfold : 1
(819, 52) (819,) (205, 52) (205,) (256, 52) (256,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.41      0.76      0.54        45
        Cardboard       0.28      0.23      0.25        40
Disposable Plates       0.93      0.34      0.50        41
            Paper       0.38      0.29      0.33        35
     Paper Towels       0.50      0.57      0.53        44

         accuracy                           0.45       205
        macro avg       0.50      0.44      0.43       205
     weighted avg       0.51      0.45      0.44       205

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.40      0.55      0.46        58
        Cardboard       0.33      0.27      0.30        59
Disposable Plates       0.74      0.25      0.38        55
            Paper       0.24      0.24      0.24        37
     Paper Towels       0.33      0.49      0.40        47

         accuracy                           0.37       256
        macro avg       0.41      0.36      0.35       256
     weighted avg       0.42      0.37      0.36       256

{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
SVC(C=1, gamma=0.01)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.39      0.78      0.52        45
        Cardboard       0.25      0.20      0.22        40
Disposable Plates       0.80      0.29      0.43        41
            Paper       0.31      0.14      0.20        35
     Paper Towels       0.45      0.55      0.49        44

         accuracy                           0.41       205
        macro avg       0.44      0.39      0.37       205
     weighted avg       0.45      0.41      0.38       205

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.42      0.66      0.51        58
        Cardboard       0.34      0.25      0.29        59
Disposable Plates       0.78      0.25      0.38        55
            Paper       0.17      0.14      0.15        37
     Paper Towels       0.32      0.51      0.40        47

         accuracy                           0.38       256
        macro avg       0.41      0.36      0.35       256
```

```
weighted avg       0.42      0.38      0.36       256
```

Histogram of Oriented Gradients(HOG)

HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract the edge features from the image. In the case of edge features, we only identify if the pixel is an edge or not. HOG is able to provide the edge direction as well. This is done by extracting the gradient and orientation of the edges. These these orientations are calculated in 'localized' portions, which means that the complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. Finally the HOG would generate a Histogram for each of these regions separately. We chose to use HOG feature to see edges and the gradient and orientation of the edges from our varied data set. The way aluminium edge gradient will be aligned will be different from the edge orientation pf paper towel.

```
%%time
viz_battery, fd_battery = get_hog_feature(batteries_data)
viz_cardboard, fd_cardboard = get_hog_feature(cardboard_data)
viz_plates, fd_plates = get_hog_feature(plates_data)
viz_paper, fd_paper = get_hog_feature(paper_data)
viz_paper_towel, fd_paper_towel = get_hog_feature(paper_towel_data)

hog_columns = ['hog_feature_' + str(idx + 1) for idx in range(len(fd_paper_towel[0]))]
df_hog_feature_battery = pd.DataFrame(fd_battery, columns = hog_columns)
df_hog_feature_battery['label'] = 0

df_hog_feature_cardboard = pd.DataFrame(fd_cardboard, columns = hog_columns)
df_hog_feature_cardboard['label'] = 1

df_hog_feature_plates = pd.DataFrame(fd_plates, columns = hog_columns)
df_hog_feature_plates['label'] = 2

df_hog_feature_paper = pd.DataFrame(fd_paper, columns = hog_columns)
df_hog_feature_paper['label'] = 3

df_hog_feature_paper_towel = pd.DataFrame(fd_paper_towel, columns = hog_columns)
df_hog_feature_paper_towel['label'] = 4

df_hog_feature = pd.concat([df_hog_feature_battery,
                            df_hog_feature_cardboard,
                            df_hog_feature_plates,
                            df_hog_feature_paper,
                            df_hog_feature_paper_towel], ignore_index = True)
print(df_hog_feature.shape)
df_hog_feature.head()
```

```
(1280, 257)
CPU times: user 4.68 s, sys: 82.9 ms, total: 4.76 s
Wall time: 4.68 s
```

	hog_feature_1	hog_feature_2	hog_feature_3	hog_feature_4	hog_feature_5	hog
0	0.000054	0.000076	0.000000	0.000000	0.019397	
1	0.000000	0.000000	0.000090	0.000000	0.021656	
2	0.000000	0.000000	0.000000	0.000000	0.054200	
3	0.001803	0.001491	0.000881	0.001245	0.000264	
4	0.065375	0.028742	0.058343	0.002422	0.017126	

```
preprocess_and_classification_svm(df_hog_feature,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with HOG Features",
                                  "SVM_HOG_Image.png"
                                 )
```

```
************************************************************
kfold : 1
(819, 256) (819,) (205, 256) (205,) (256, 256) (256,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.62      0.80      0.70        45
        Cardboard       0.53      0.50      0.51        40
Disposable Plates       0.88      0.71      0.78        41
            Paper       0.64      0.60      0.62        35
     Paper Towels       0.47      0.45      0.46        44

         accuracy                           0.61       205
        macro avg       0.63      0.61      0.61       205
     weighted avg       0.62      0.61      0.61       205

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.57      0.67      0.62        58
        Cardboard       0.41      0.31      0.35        59
Disposable Plates       0.83      0.71      0.76        55
            Paper       0.50      0.54      0.52        37
     Paper Towels       0.46      0.55      0.50        47

         accuracy                           0.55       256
        macro avg       0.55      0.56      0.55       256
     weighted avg       0.56      0.55      0.55       256

{'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.01)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.69      0.76      0.72        45
        Cardboard       0.46      0.68      0.55        40
Disposable Plates       0.94      0.71      0.81        41
            Paper       0.67      0.46      0.54        35
     Paper Towels       0.60      0.57      0.58        44

         accuracy                           0.64       205
        macro avg       0.67      0.63      0.64       205
     weighted avg       0.67      0.64      0.64       205

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.67      0.62      0.64        58
        Cardboard       0.40      0.47      0.43        59
Disposable Plates       0.94      0.60      0.73        55
            Paper       0.52      0.46      0.49        37
     Paper Towels       0.42      0.57      0.49        47

         accuracy                           0.55       256
        macro avg       0.59      0.55      0.56       256
```

```
weighted avg       0.60      0.55      0.56       256
```

Combine HOG LBP and GLCM features

[ ] ↳ *1 cell hidden*

```
Validation results for fold 2
```

Classification using HOG-LBP-GLCM combined features

[ ] ↳ *1 cell hidden*

```
Paper         0.43      0.43      0.43        35
```

PCA on HOG-LBP-GLCM combined features and classification

[ ] ↳ *1 cell hidden*

Scale Invariant Feature Transform(SIFT)

SIFT it is a feature extraction method where image content is transformed into local feature coordinates that are invariant to translation, scale and other image transformations.

Although, we human beings can easily identify objects in images despite of variations in angle or scale, machines struggle with this task. However, through feature extraction techniques present in Computer Vision, we can identify the key points and their corresponding descriptors. These key points are scale & rotation invariant. That's the biggest advantage of this feature extraction method contrary to other EDGE detection algorithms and HOG feature extraction process. Each image can have varied number of key points or may not have any at all. Descriptors are associated with each key point and they are represented by a feature vector of size 128.

We chose to use the SIFT feature extraction method as we have seen the same image in different orientations and angles. SIFT features combined with k-means clustering will group them in smaller feature space which will help us to classify the images in appropriate categories.

[ ] ↳ *4 cells hidden*

Oriented FAST and rotated BRIEF (ORB)

ORB is a fast robust local feature detector which is used in computer vision tasks such as object recognition or 3D reconstruction.

ORB uses a modified version of the FAST keypoint detector and BRIEF descriptor. FAST features are not scale-invariant and rotation invariant. To cater that shortcomings of FAST algorithm, a multi-scale pyramid mechanism is used in ORB with which ORB detects features at each level for better accuracy.

Like SIFT feature detection, ORB method also identifies keypoints and corresponding descriptors. Each image can have 0 or more keypoints and each key point is associated with a feature vector of size 32.

Simialr to SIFT feature extraction process, while dealing with large set of images with varied number of keypoints, we need to combine the features through clustering and those reduced generalized feature vectors help to classify the images.

We used the similar method for ORB, however the clustering did not work as all the descriptors were assigned to a single cluster.

```
%%time
def get_orb_kp_feature_array(feature):
    array_len = 0
    kp_faeture_list = []
    for e in feature:
        keypoints, descriptors = e
        if len(keypoints) > 0:
             array_len = len(descriptors[0])
             break
    for e in feature:
        keypoints, descriptors = e
        if len(keypoints) == 0:
            kp_faeture_list.append(np.zeros(array_len))
        else:
            kp_faeture_list.append(np.array(descriptors[0]))
    feature_columns = ['kp_' + str(i + 1) for i in range(array_len)]
    return pd.DataFrame(kp_faeture_list, columns = feature_columns)

def get_orb_feature(np_img_list):
    orb = cv2.ORB_create()
    return [orb.detectAndCompute(img, None) for img in np_img_list]

feature_orb_battery = get_orb_feature(batteries_data)
feature_orb_cardboard = get_orb_feature(cardboard_data)
feature_orb_plate = get_orb_feature(plates_data)
feature_orb_paper = get_orb_feature(paper_data)
feature_orb_paper_towel = get_orb_feature(paper_towel_data)

orb_feature_list = [feature_orb_battery, feature_orb_cardboard, feature_orb_plate, feature_orb_

df_orb_feature_battery = get_orb_kp_feature_array(feature_orb_battery)
df_orb_feature_battery['label'] = 0
df_orb_feature_battery.to_csv("orb_feature_battery.csv", index = False)

df_orb_feature_cardboard = get_orb_kp_feature_array(feature_orb_cardboard)
df_orb_feature_cardboard['label'] = 1
df_orb_feature_cardboard.to_csv("orb_feature_cardboard.csv", index = False)

df_orb_feature_plates = get_orb_kp_feature_array(feature_orb_battery)
df_orb_feature_plates['label'] = 2
df_orb_feature_plates.to_csv("orb_feature_battery.csv", index = False)

df_orb_feature_paper = get_orb_kp_feature_array(feature_orb_paper)
```

```
df_orb_feature_paper['label'] = 3
df_orb_feature_paper.to_csv("orb_feature_plates.csv", index = False)

df_orb_feature_paper_towel = get_orb_kp_feature_array(feature_orb_paper_towel)
df_orb_feature_paper_towel['label'] = 4
df_orb_feature_paper_towel.to_csv("orb_feature_paper_towel.csv", index = False)

df_orb_feature = pd.concat([df_orb_feature_battery,
                            df_orb_feature_cardboard,
                            df_orb_feature_plates,
                            df_orb_feature_paper,
                            df_orb_feature_paper_towel
                           ],
                           axis = 0
                           )
print(f"Shape of df_orb_feature : {df_orb_feature.shape}")

preprocess_and_classification_svm(df_orb_feature,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with ORB Features",
                                  "SVM_ORB_Image.png"
                                 )
```

```
Shape of df_orb_feature : (1303, 33)
************************************************************
kfold : 1
(833, 32) (833,) (209, 32) (209,) (261, 32) (261,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.14      0.09      0.11        45
        Cardboard       0.00      0.00      0.00        40
Disposable Plates       0.00      0.00      0.00        46
            Paper       0.00      0.00      0.00        35
     Paper Towels       0.21      0.84      0.33        43

         accuracy                           0.19       209
        macro avg       0.07      0.19      0.09       209
     weighted avg       0.07      0.19      0.09       209

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.13      0.07      0.09        57
        Cardboard       0.00      0.00      0.00        55
Disposable Plates       0.20      0.04      0.06        55
            Paper       0.00      0.00      0.00        40
     Paper Towels       0.21      0.87      0.34        54

         accuracy                           0.20       261
        macro avg       0.11      0.20      0.10       261
     weighted avg       0.12      0.20      0.10       261

{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1, gamma=0.0001)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.00      0.00      0.00        45
        Cardboard       0.00      0.00      0.00        40
Disposable Plates       0.22      1.00      0.36        46
            Paper       0.00      0.00      0.00        35
     Paper Towels       0.00      0.00      0.00        43

         accuracy                           0.22       209
        macro avg       0.04      0.20      0.07       209
     weighted avg       0.05      0.22      0.08       209

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.00      0.00      0.00        57
        Cardboard       0.00      0.00      0.00        55
Disposable Plates       0.21      1.00      0.35        55
            Paper       0.00      0.00      0.00        40
     Paper Towels       0.00      0.00      0.00        54

         accuracy                           0.21       261
```

```
macro avg       0.04      0.20      0.07       261
```

```
'''
%%time
def clean_and_normalize_orb_descriptor_list(feature_list):
    # Exclude None descriptors
    discard_desc_list = []
    clean_feature_list = []
    for i, desc in enumerate(feature_list):
        if desc is None:
            #print(f"**i {i}")
            discard_desc_list.append(i)
    for idx, val in enumerate(feature_list):
        if idx not in discard_desc_list:
            clean_feature_list.append(val)
        else:
            #print(idx)
            None

    # Bringing all cleaned descriptors in the array form for further processing.
    print(f"After cleanup, length : {len(clean_feature_list)}")
    fmt_clean_orb_feature_list = []
    for idx, e in enumerate(clean_feature_list):
        #print(e[0],idx)
        if len(e[0]) == 0:
            #fmt_clean_orb_feature_list.append([])
            continue
        if len(list(cv2.KeyPoint_convert(clean_feature_list[idx][0])[0])) == 2:
            fmt_clean_orb_feature_list.append(clean_feature_list[idx][1])
    #print(fmt_clean_orb_feature_list)
    all_descriptors = []
    for idx, img_descriptors in enumerate(clean_feature_list):
        #print(img_descriptors[1])
        #print("^^^^^^^^^")
        if img_descriptors[1] is None:
            #print(f"## i {idx}")
            #print(img_descriptors)
            #print("***")
            discard_desc_list.append(idx)
            continue
        else:
            all_descriptors.append(img_descriptors[1][0])
        #print(img_descriptors[1][0])
        #print("^^^^^^^^^")
        #for descriptor in img_descriptors[1][0]:
        #    all_descriptors.append(descriptor)
    all_descriptors = np.stack(all_descriptors)
    # normalize
    all_descriptors = whiten(all_descriptors)
    return fmt_clean_orb_feature_list, all_descriptors, discard_desc_list

orb_feature_list = [feature_orb_battery, feature_orb_cardboard, feature_orb_plate, feature_orb_

# Bringing all cleaned descriptors in the array form for further processing.
clean_orb_feature_list, all_orb_descriptors, discard_orb_desc_list = clean_and_normalize_orb_de
print(f"Length of discard_orb_desc_list : {len(discard_orb_desc_list)}")
print(f"Length of clean_orb_feature_list after : {len(clean_orb_feature_list)}")
print("discard_orb_desc_list")
```

```
print(discard_orb_desc_list)
print(f"Shape of all_orb_descriptors : {all_orb_descriptors.shape}")

# codebook should have shape of k X 32
orb_codebook = create_codebook(all_orb_descriptors, k = 5, iterations = 10)
print(f"orb_codebook shape : {orb_codebook.shape}")

def form_orb_visual_words(feature_list, codebook):
    visual_words = []
    for idx, desc in enumerate(feature_list):
        print(feature_list)
        print(desc)
        img_visual_words, distance = vq(desc, codebook)
        visual_words.append(img_visual_words)
    return visual_words

# This will show how many visual words are present in each image according to the codebook.
orb_visual_words = form_orb_visual_words(clean_orb_feature_list, orb_codebook)
print(f"Length of orb_visual_words : {len(orb_visual_words)}")

# Forming the vector
orb_frequency_vectors = get_frequency_counts(orb_visual_words, k = 5)
print(f"ORB Frequency Vectors shape : {orb_frequency_vectors.shape}")

plt.bar(list(range(5)), orb_frequency_vectors[0])
plt.show()

orb_tf_idf = get_visual_tfidf(clean_orb_feature_list[0], orb_frequency_vectors)
print(f"orb_tf_idf shape : {orb_tf_idf.shape}")

new_orb_shuffle = np.random.permutation(np.arange(len(clean_orb_feature_list[0])))
X_orb, y_orb = tf_idf[new_orb_shuffle], label[new_orb_shuffle]
df_orb = pd.DataFrame(X_orb)
df_orb['label'] = y_orb

preprocess_and_classification_svm(df_orb,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with ORB Features",
                                  "SVM_ORB_Image.png"
                                 )
'''
```

```
'\n%%time\ndef clean_and_normalize_orb_descriptor_list(feature_list):\n    # Ex
clude None descriptors\n    discard_desc_list = []\n    clean_feature_list = []
\n    for i, desc in enumerate(feature_list):\n        if desc is None:\n
#print(f"**i {i}")\n            discard_desc_list.append(i)\n    for idx, val i
n enumerate(feature_list):\n        if idx not in discard_desc_list:\n
clean_feature_list.append(val)\n        else:\n            #print(idx)\n
None\n            \n    # Bringing all cleaned descriptors in the array form fo
r further processing.\n    print(f"After cleanup, length : {len(clean_feature_l
ist)}")\n    fmt_clean_orb_feature_list = []\n    for idx, e in enumerate(clean
_feature_list):\n        #print(e[0],idx)\n        if len(e[0]) == 0:\n
#fmt_clean_orb_feature_list.append([])\n            continue\n        if len(li
st(cv2.KeyPoint_convert(clean_feature_list[idx][0])[0])) == 2:\n            fmt
_clean_orb_feature_list.append(clea...'
```

```
Disposable Plates       0.21      1.00      0.35        55
```

Edge Detection

Edge detection uses variery of mathametical methods to identify the edges. There are two categories of edge detection algorithms search based and zero crossing based.

The edge detection methods available mainly differ in the types of smoothing filters that are applied and the way the measures of edge strength are computed. As many edge detection methods rely on the computation of image gradients, they also differ in the types of filters used for computing gradient estimates in the x- and y-directions.

	Batteries	0.11	0.04	0.06	45

Canny

Canny SVM

Canny filter is devised as deriving an optimal smoothing filter given the criteria of detection, localization and minimizing multiple responses to a single edge. Canny uses the notion of non-maximum suppression, which means that given the presmoothing filters, edge points are defined as points where the gradient magnitude assumes a local maximum in the gradient direction. We see Canny edge detection really worked well for battery classes.

```
%%time
feature_canny_battery = get_canny_feature(batteries_data)
feature_canny_cardboard = get_canny_feature(cardboard_data)
feature_canny_plates = get_canny_feature(plates_data)
feature_canny_paper = get_canny_feature(paper_data)
feature_canny_paper_towel = get_canny_feature(paper_towel_data)

df_canny_battery = create_feature_df(feature_canny_battery, 0)
df_canny_cardboard = create_feature_df(feature_canny_cardboard, 1)
df_canny_plates = create_feature_df(feature_canny_plates, 2)
df_canny_paper = create_feature_df(feature_canny_paper, 3)
df_canny_paper_towel = create_feature_df(feature_canny_paper_towel, 4)

df_canny = combine_dfs(df_canny_battery, df_canny_cardboard, df_canny_plates, df_canny_paper, c

preprocess_and_classification_svm(df_canny,
                                  label_list,
                                  "Support Vector Machines Confusion Matrix with CANNY Features
                                  "SVM_CANNY_Image.png"
                                 )
```

```
************************************************************
kfold : 1
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.40      0.84      0.55        45
        Cardboard       0.33      0.15      0.21        40
Disposable Plates       0.50      0.17      0.25        41
            Paper       0.52      0.43      0.47        35
     Paper Towels       0.44      0.50      0.47        44

         accuracy                           0.43       205
        macro avg       0.44      0.42      0.39       205
     weighted avg       0.44      0.43      0.39       205

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.39      0.69      0.50        58
        Cardboard       0.35      0.10      0.16        59
Disposable Plates       0.76      0.53      0.62        55
            Paper       0.47      0.49      0.48        37
     Paper Towels       0.34      0.45      0.39        47

         accuracy                           0.45       256
        macro avg       0.47      0.45      0.43       256
     weighted avg       0.47      0.45      0.42       256

{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.0001)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.45      0.69      0.54        45
        Cardboard       0.42      0.28      0.33        40
Disposable Plates       0.55      0.41      0.47        41
            Paper       0.48      0.60      0.53        35
     Paper Towels       0.40      0.32      0.35        44

         accuracy                           0.46       205
        macro avg       0.46      0.46      0.45       205
     weighted avg       0.46      0.46      0.45       205

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.40      0.53      0.46        58
        Cardboard       0.28      0.14      0.18        59
Disposable Plates       0.56      0.62      0.59        55
            Paper       0.42      0.51      0.46        37
     Paper Towels       0.39      0.36      0.37        47

         accuracy                           0.43       256
        macro avg       0.41      0.43      0.41       256
```

```
     weighted avg       0.41      0.43      0.41       256

************************************************************
kfold : 2
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.40      0.76      0.53        45
        Cardboard       0.31      0.12      0.18        40
Disposable Plates       0.57      0.39      0.46        41
            Paper       0.42      0.29      0.34        35
     Paper Towels       0.36      0.43      0.39        44

         accuracy                           0.41       205
        macro avg       0.41      0.40      0.38       205
     weighted avg       0.41      0.41      0.39       205

Test results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.40      0.66      0.50        58
        Cardboard       0.27      0.07      0.11        59
Disposable Plates       0.74      0.56      0.64        55
            Paper       0.53      0.49      0.51        37
     Paper Towels       0.31      0.47      0.38        47

         accuracy                           0.44       256
        macro avg       0.45      0.45      0.43       256
     weighted avg       0.44      0.44      0.42       256

{'C': 1000, 'gamma': 1e-06, 'kernel': 'rbf'}
SVC(C=1000, gamma=1e-06)
Validation grid search results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.45      0.51      0.48        45
        Cardboard       0.28      0.20      0.23        40
Disposable Plates       0.39      0.41      0.40        41
            Paper       0.28      0.31      0.29        35
     Paper Towels       0.41      0.39      0.40        44

         accuracy                           0.37       205
        macro avg       0.36      0.37      0.36       205
     weighted avg       0.37      0.37      0.37       205

Test grid search results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.45      0.50      0.47        58
        Cardboard       0.28      0.14      0.18        59
Disposable Plates       0.53      0.60      0.56        55
            Paper       0.37      0.51      0.43        37
     Paper Towels       0.33      0.34      0.33        47

         accuracy                           0.41       256
```

```
                accuracy                           0.41       256
               macro avg       0.39      0.42      0.40       256
            weighted avg       0.39      0.41      0.39       256

************************************************************
kfold : 3
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.45      0.78      0.57        45
        Cardboard       0.21      0.07      0.11        40
Disposable Plates       0.72      0.44      0.55        41
            Paper       0.40      0.34      0.37        35
     Paper Towels       0.44      0.59      0.50        44

         accuracy                           0.46       205
        macro avg       0.45      0.45      0.42       205
     weighted avg       0.45      0.46      0.43       205

Test results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.40      0.67      0.50        58
        Cardboard       0.40      0.07      0.12        59
Disposable Plates       0.73      0.49      0.59        55
            Paper       0.46      0.46      0.46        37
     Paper Towels       0.29      0.47      0.36        47

         accuracy                           0.43       256
        macro avg       0.46      0.43      0.41       256
     weighted avg       0.46      0.43      0.40       256

{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.0001)
Validation grid search results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.53      0.76      0.62        45
        Cardboard       0.31      0.20      0.24        40
Disposable Plates       0.53      0.44      0.48        41
            Paper       0.31      0.37      0.34        35
     Paper Towels       0.54      0.48      0.51        44

         accuracy                           0.46       205
        macro avg       0.44      0.45      0.44       205
     weighted avg       0.45      0.46      0.45       205

Test grid search results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.41      0.55      0.47        58
        Cardboard       0.24      0.14      0.17        59
Disposable Plates       0.61      0.60      0.61        55
            Paper       0.47      0.51      0.49        37
     Paper Towels       0.33      0.34      0.33        47
```

```
          accuracy                           0.42       256
         macro avg       0.41      0.43      0.41       256
      weighted avg       0.41      0.42      0.41       256

************************************************************
kfold : 4
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.43      0.87      0.57        45
        Cardboard       0.73      0.20      0.31        40
Disposable Plates       0.67      0.39      0.49        41
            Paper       0.50      0.31      0.39        35
     Paper Towels       0.40      0.52      0.46        44

         accuracy                           0.47       205
        macro avg       0.55      0.46      0.44       205
     weighted avg       0.54      0.47      0.45       205

Test results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.39      0.67      0.49        58
        Cardboard       0.50      0.08      0.14        59
Disposable Plates       0.74      0.51      0.60        55
            Paper       0.50      0.46      0.48        37
     Paper Towels       0.32      0.49      0.38        47

         accuracy                           0.44       256
        macro avg       0.49      0.44      0.42       256
     weighted avg       0.49      0.44      0.41       256

{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.0001)
Validation grid search results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.48      0.78      0.59        45
        Cardboard       0.27      0.33      0.30        40
Disposable Plates       0.55      0.56      0.55        41
            Paper       0.50      0.20      0.29        35
     Paper Towels       0.39      0.25      0.31        44

         accuracy                           0.43       205
        macro avg       0.44      0.42      0.41       205
     weighted avg       0.44      0.43      0.41       205

Test grid search results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.41      0.57      0.48        58
        Cardboard       0.33      0.29      0.31        59
Disposable Plates       0.59      0.64      0.61        55
            Paper       0.64      0.19      0.29        37
```

```
          Paper Towels       0.31      0.36      0.34        47

             accuracy                           0.43       256
            macro avg       0.46      0.41      0.41       256
         weighted avg       0.45      0.43      0.41       256

************************************************************
kfold : 5
(820, 4096) (820,) (204, 4096) (204,) (256, 4096) (256,)
****
Validation results for fold 5
                    precision    recall  f1-score   support

           Batteries       0.39      0.75      0.52        44
           Cardboard       0.38      0.12      0.19        40
   Disposable Plates       0.89      0.42      0.58        40
               Paper       0.47      0.49      0.48        35
         Paper Towels       0.50      0.58      0.54        45

            accuracy                           0.48       204
           macro avg       0.53      0.47      0.46       204
        weighted avg       0.53      0.48      0.46       204

Test results for fold 5
                    precision    recall  f1-score   support

           Batteries       0.40      0.67      0.50        58
           Cardboard       0.30      0.05      0.09        59
   Disposable Plates       0.76      0.47      0.58        55
               Paper       0.51      0.49      0.50        37
         Paper Towels       0.34      0.57      0.43        47

            accuracy                           0.44       256
           macro avg       0.46      0.45      0.42       256
        weighted avg       0.46      0.44      0.41       256

{'C': 1000, 'gamma': 1e-06, 'kernel': 'rbf'}
SVC(C=1000, gamma=1e-06)
Validation grid search results for fold 5
                    precision    recall  f1-score   support

           Batteries       0.42      0.52      0.46        44
           Cardboard       0.24      0.17      0.20        40
   Disposable Plates       0.60      0.53      0.56        40
               Paper       0.46      0.60      0.52        35
         Paper Towels       0.44      0.38      0.40        45

            accuracy                           0.44       204
           macro avg       0.43      0.44      0.43       204
        weighted avg       0.43      0.44      0.43       204

Test grid search results for fold 5
                    precision    recall  f1-score   support

           Batteries       0.44      0.47      0.45        58
           Cardboard       0.17      0.08      0.11        59
```

```
Disposable Plates       0.53      0.62      0.57        55
            Paper       0.34      0.49      0.40        37
      Paper Towels       0.25      0.26      0.25        47

         accuracy                           0.38       256
        macro avg       0.35      0.38      0.36       256
     weighted avg       0.35      0.38      0.36       256
```

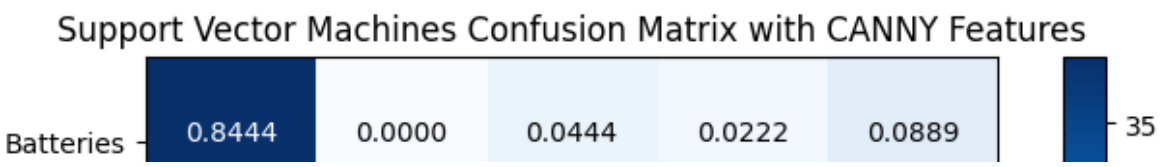
Overall Stats....

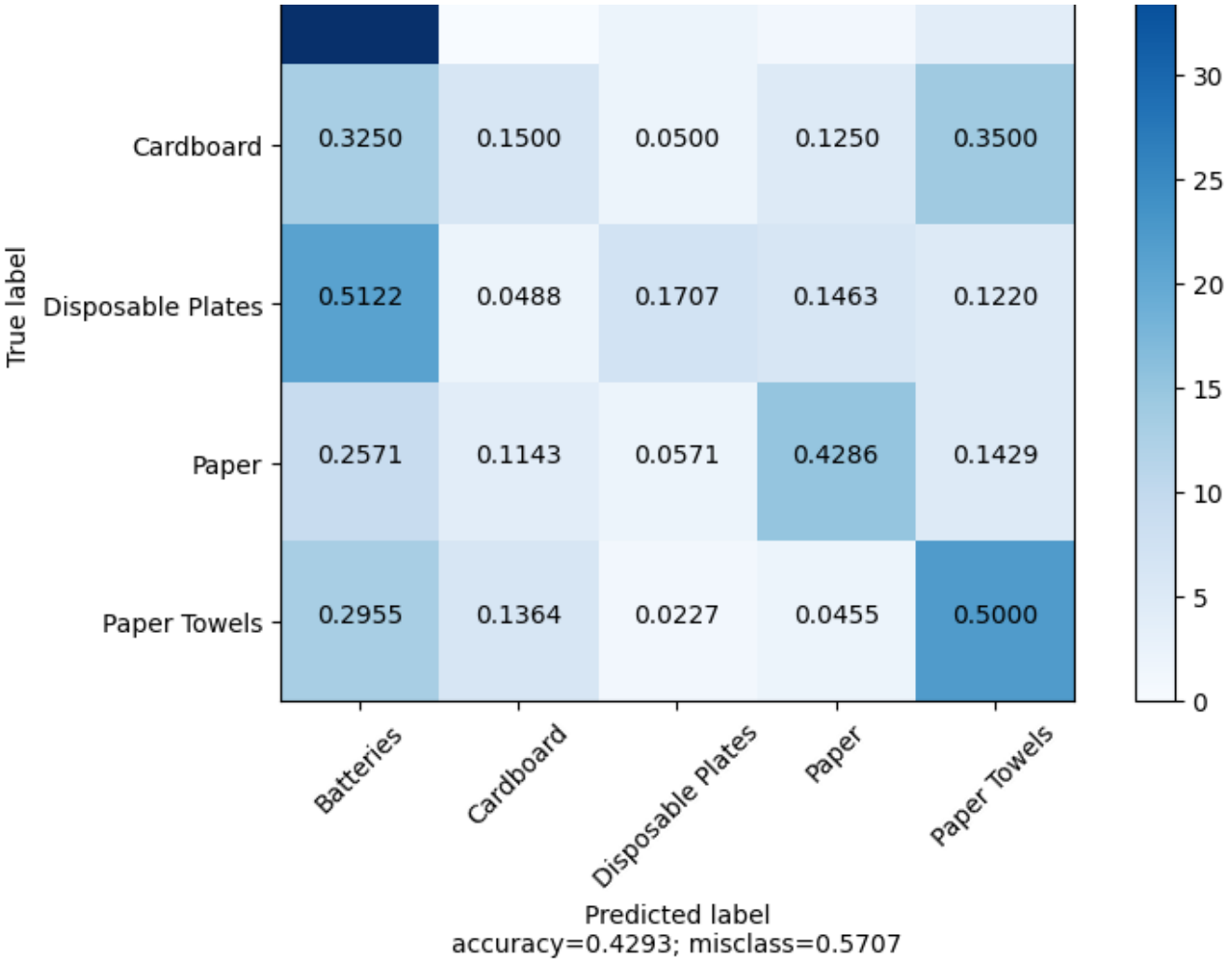
	kpi_level	min_accuracy	avg_accuracy	max_accuracy
0	base_val_accuracy	0.409756	0.450225	0.480392
1	grid_val_accuracy	0.370732	0.431645	0.458537
2	base_test_accuracy	0.425781	0.438281	0.445312
3	grid_test_accuracy	0.375000	0.411719	0.425781

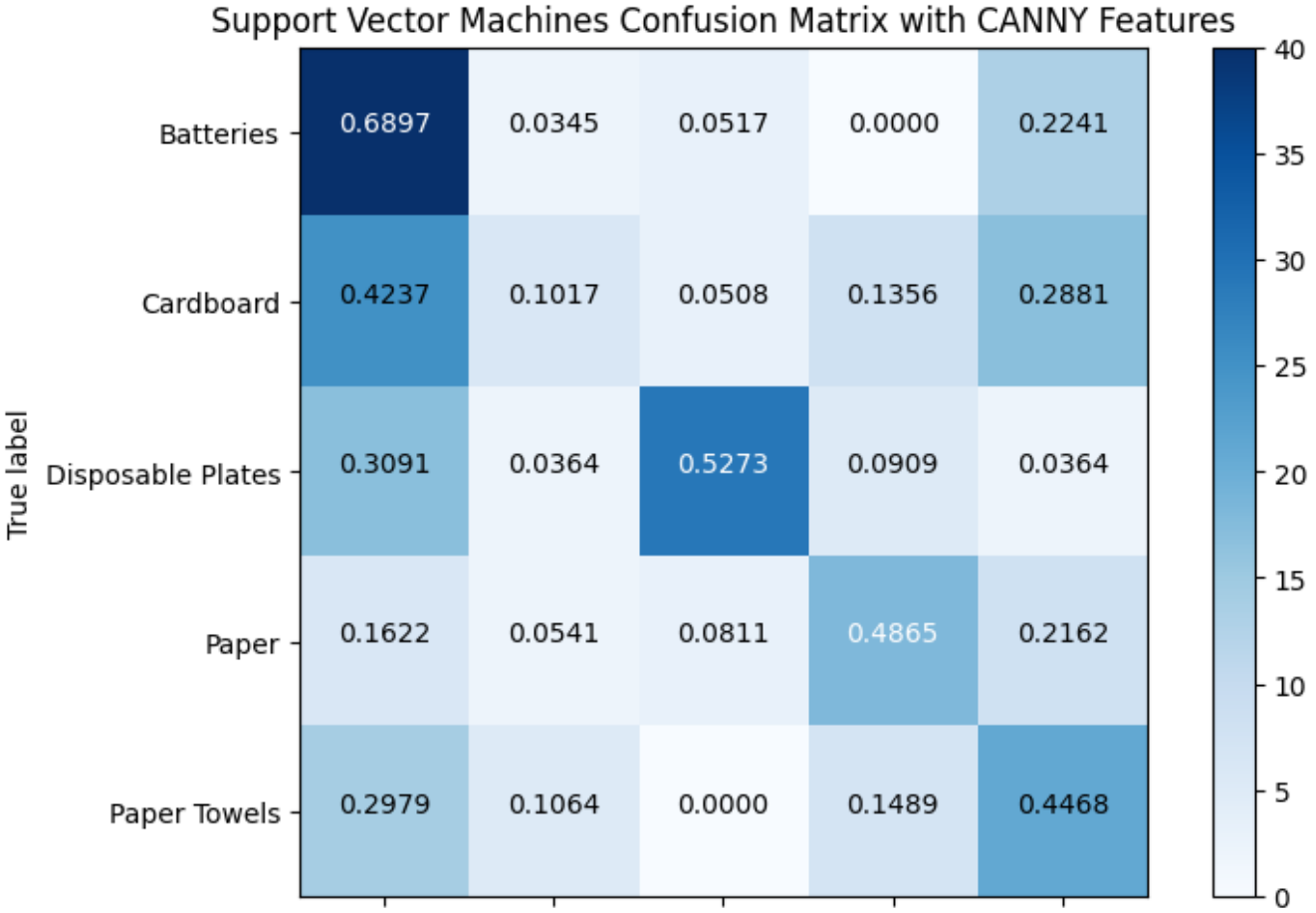
Category Wise Stats...

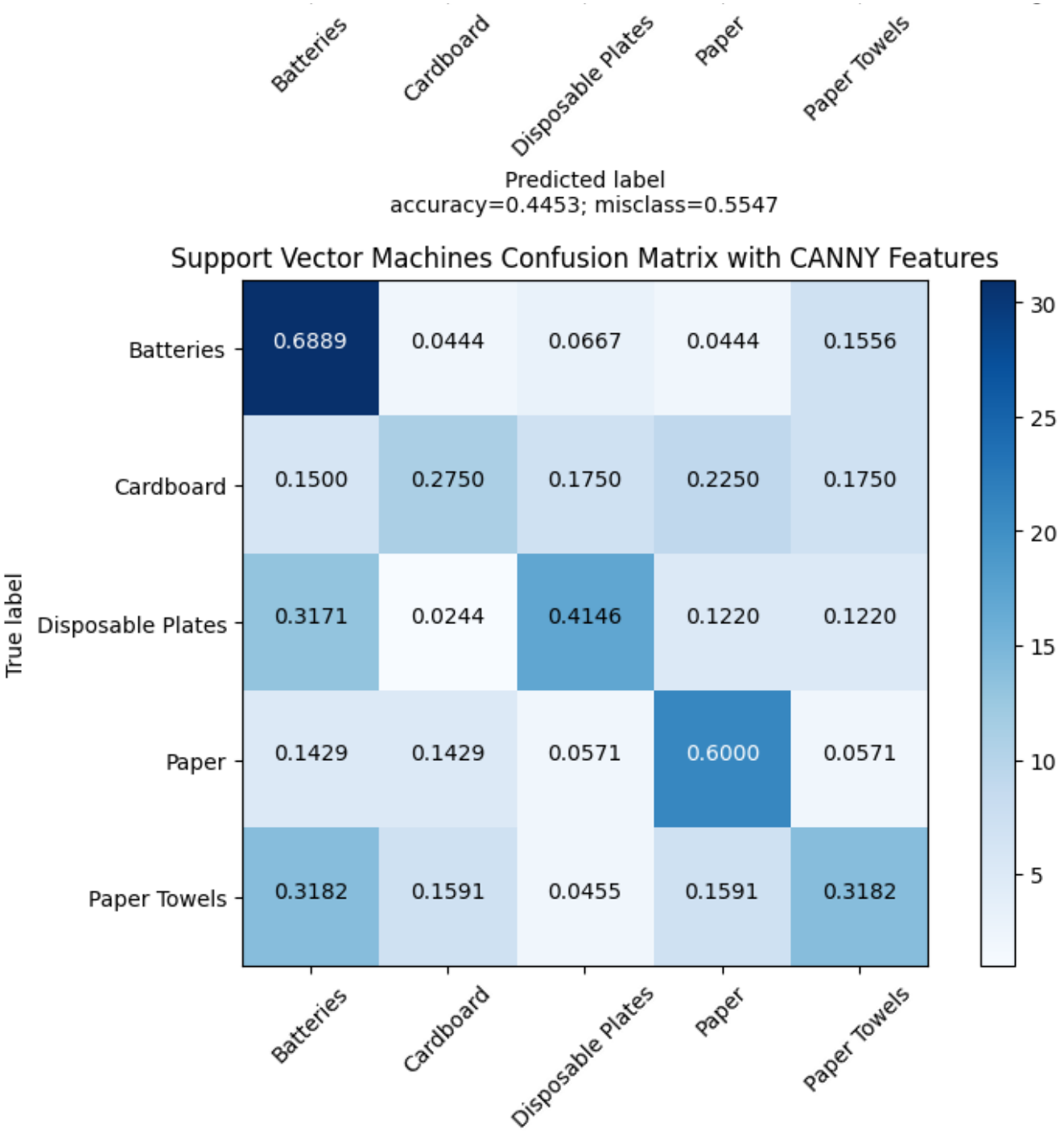
	0	1	2	3
kpi	cm_test_base	cm_test_grid	cm_val_base	cm_val_grid
battery_correct_pct_min	0.655172	0.465517	0.75	0.511111
battery_correct_pct_avg	0.672414	0.524138	0.798889	0.651212
battery_correct_pct_max	0.689655	0.568966	0.866667	0.777778
cardboard_correct_pct_min	0.050847	0.084746	0.075	0.175
cardboard_correct_pct_avg	0.074576	0.155932	0.135	0.235
cardboard_correct_pct_max	0.101695	0.288136	0.2	0.325
plates_correct_pct_min	0.472727	0.6	0.170732	0.414634
plates_correct_pct_avg	0.512727	0.614545	0.363049	0.470854
plates_correct_pct_max	0.563636	0.636364	0.439024	0.560976
paper_correct_pct_min	0.459459	0.189189	0.285714	0.2
paper_correct_pct_avg	0.475676	0.443243	0.371429	0.417143
paper_correct_pct_max	0.486486	0.513514	0.485714	0.6
paper_towel_correct_pct_min	0.446809	0.255319	0.431818	0.25
paper_towel_correct_pct_avg	0.489362	0.331915	0.524646	0.361919
paper_towel_correct_pct_max	0.574468	0.361702	0.590909	0.477273

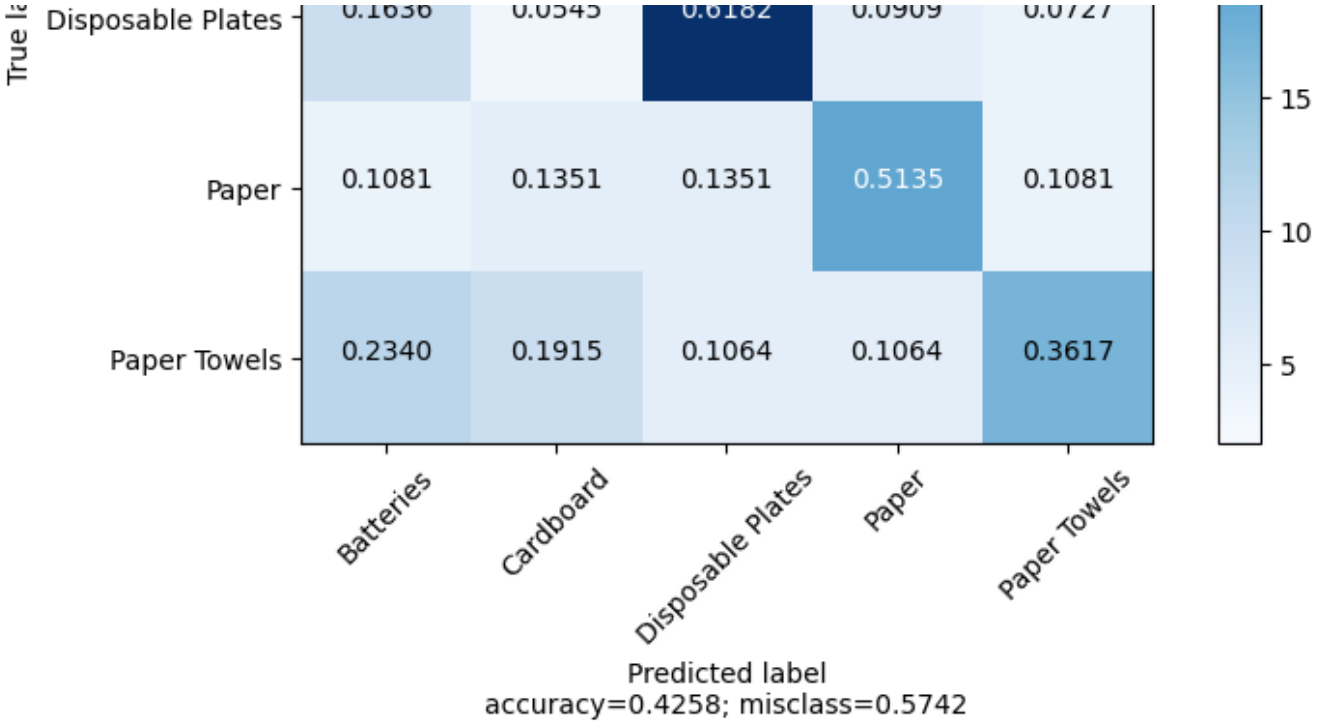
```
CPU times: user 6min 46s, sys: 6.13 s, total: 6min 53s
Wall time: 6min 44s
```

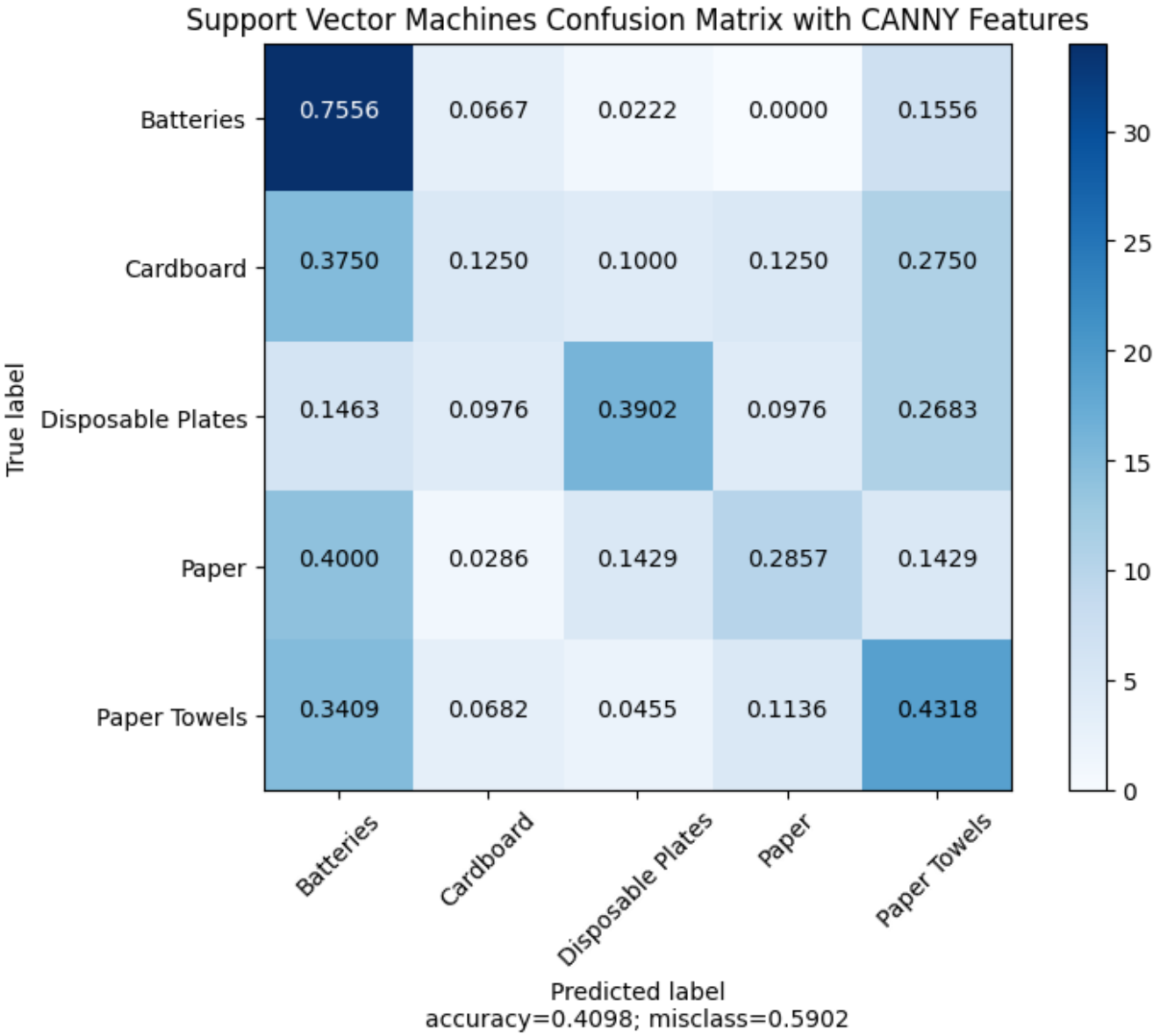


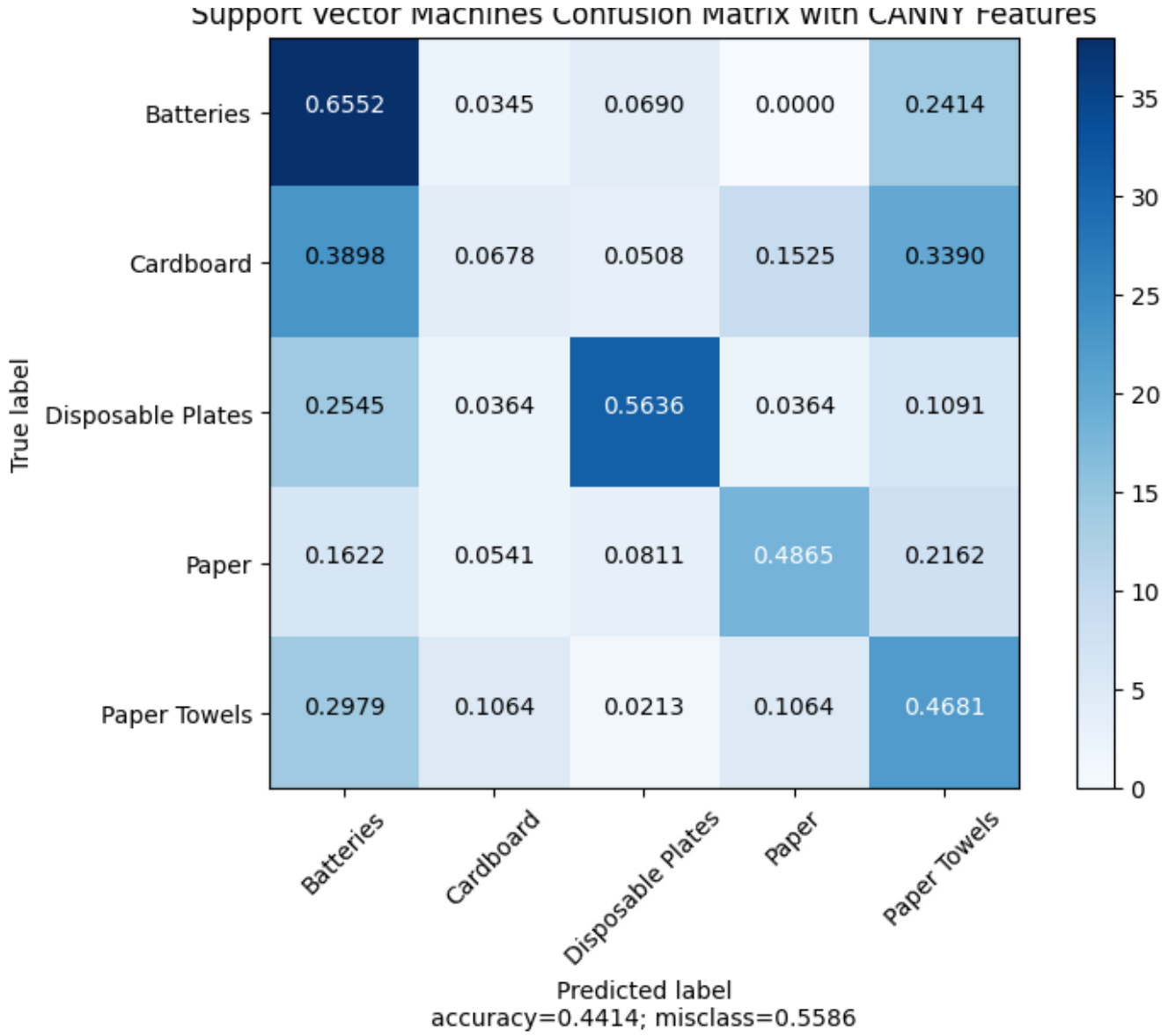

True label
Cardboard
Disposable Plates
Paper
Paper Towels
0.3250 0.1500 0.0500 0.1250 0.3500
0.5122 0.0488 0.1707 0.1463 0.1220
0.2571 0.1143 0.0571 0.4286 0.1429
0.2955 0.1364 0.0227 0.0455 0.5000
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4293; misclass=0.5707

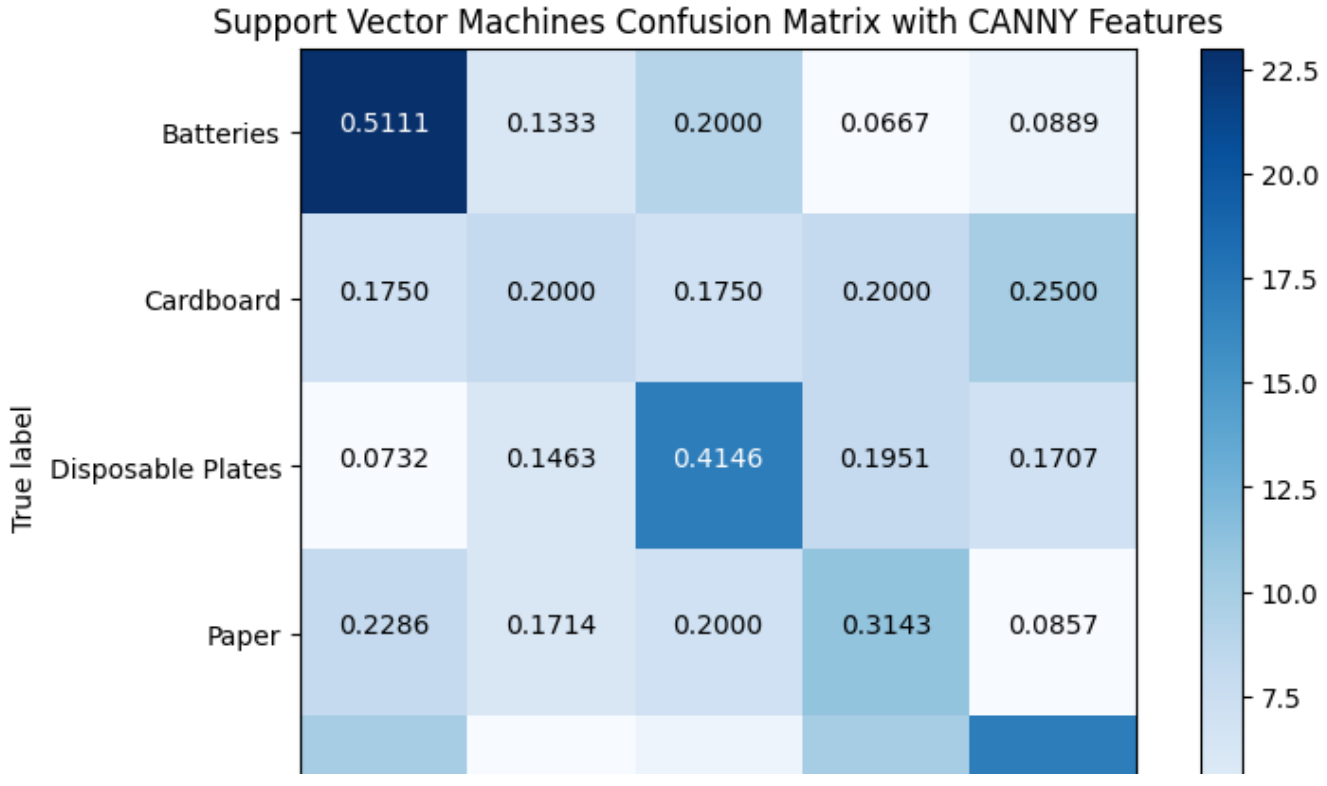

Support Vector Machines Confusion Matrix with CANNY Features
True label
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
0.6897 0.0345 0.0517 0.0000 0.2241
0.4237 0.1017 0.0508 0.1356 0.2881
0.3091 0.0364 0.5273 0.0909 0.0364
0.1622 0.0541 0.0811 0.4865 0.2162
0.2979 0.1064 0.0000 0.1489 0.4468

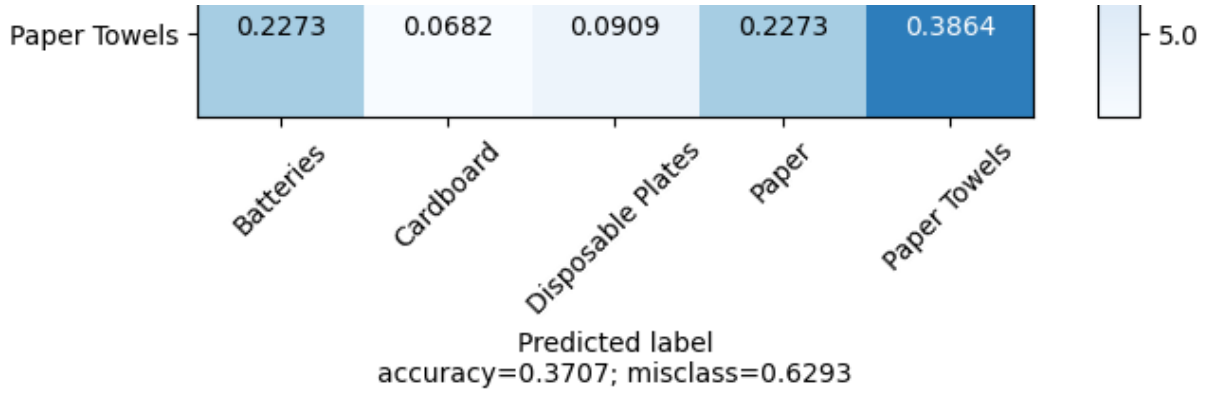

Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4453; misclass=0.5547
Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.6889 0.0444 0.0667 0.0444 0.1556
0.1500 0.2750 0.1750 0.2250 0.1750
0.3171 0.0244 0.4146 0.1220 0.1220
0.1429 0.1429 0.0571 0.6000 0.0571
0.3182 0.1591 0.0455 0.1591 0.3182
30
25
20
15
10
5
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4585; misclass=0.5415
Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
0.5345 0.0690 0.1897 0.0345 0.1724
0.3729 0.1356 0.1017 0.2373 0.1525
30
25
20

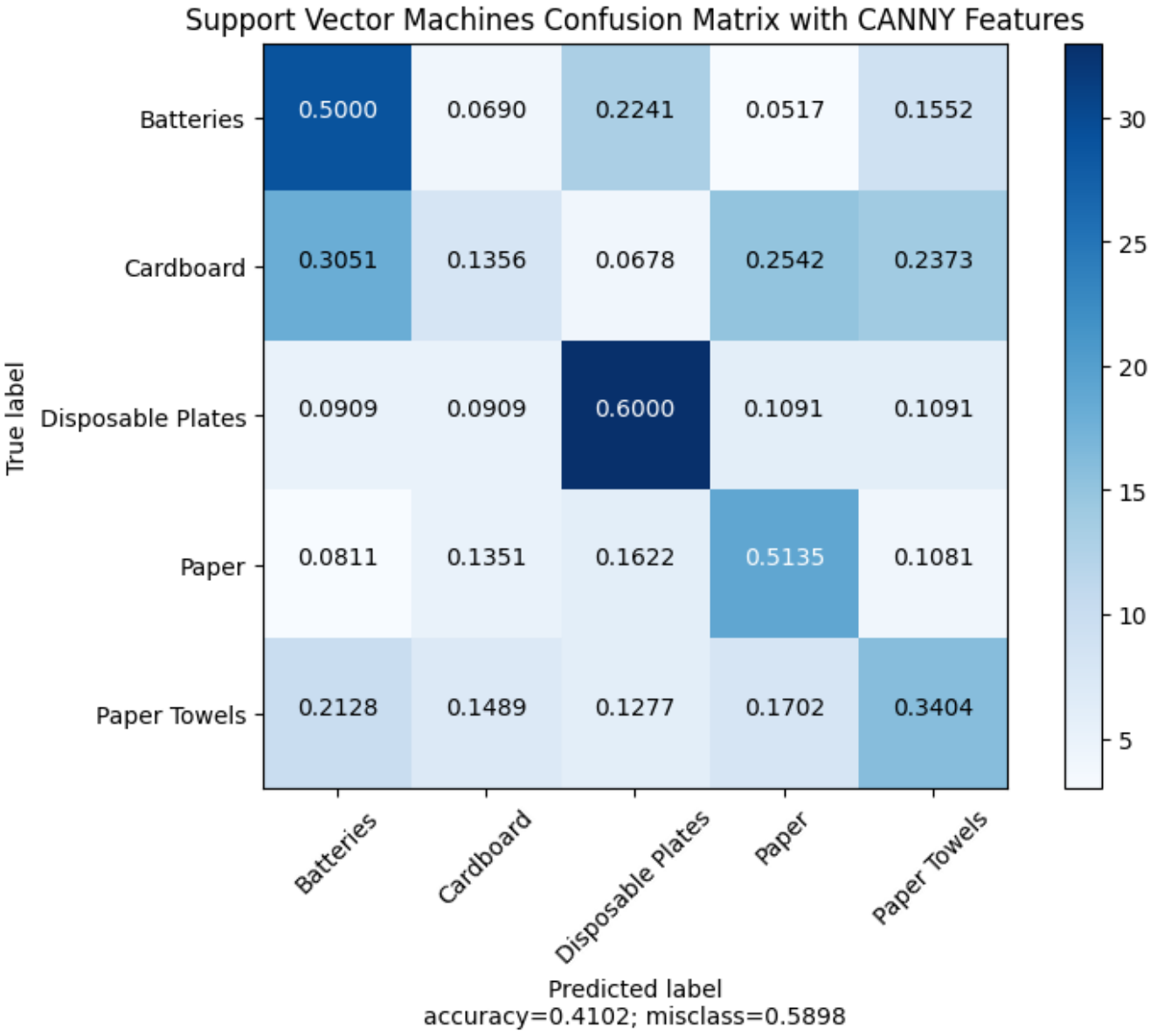

True la
Disposable Plates
Paper
Paper Towels
0.1636
0.0545
0.6182
0.0909
0.0727
0.1081
0.1351
0.1351
0.5135
0.1081
0.2340
0.1915
0.1064
0.1064
0.3617
15
10
5
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4258; misclass=0.5742

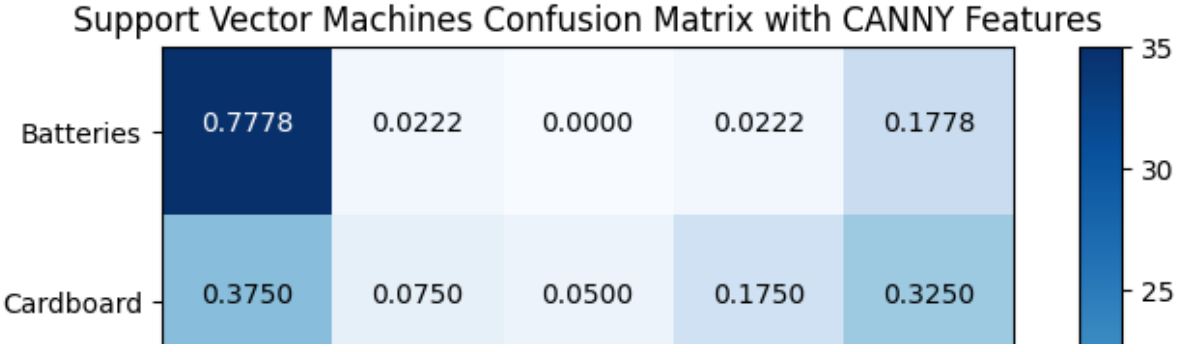

Support Vector Machines Confusion Matrix with CANNY Features
True label
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
0.7556
0.0667
0.0222
0.0000
0.1556
0.3750
0.1250
0.1000
0.1250
0.2750
0.1463
0.0976
0.3902
0.0976
0.2683
0.4000
0.0286
0.1429
0.2857
0.1429
0.3409
0.0682
0.0455
0.1136
0.4318
30
25
20
15
10
5
0
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4098; misclass=0.5902

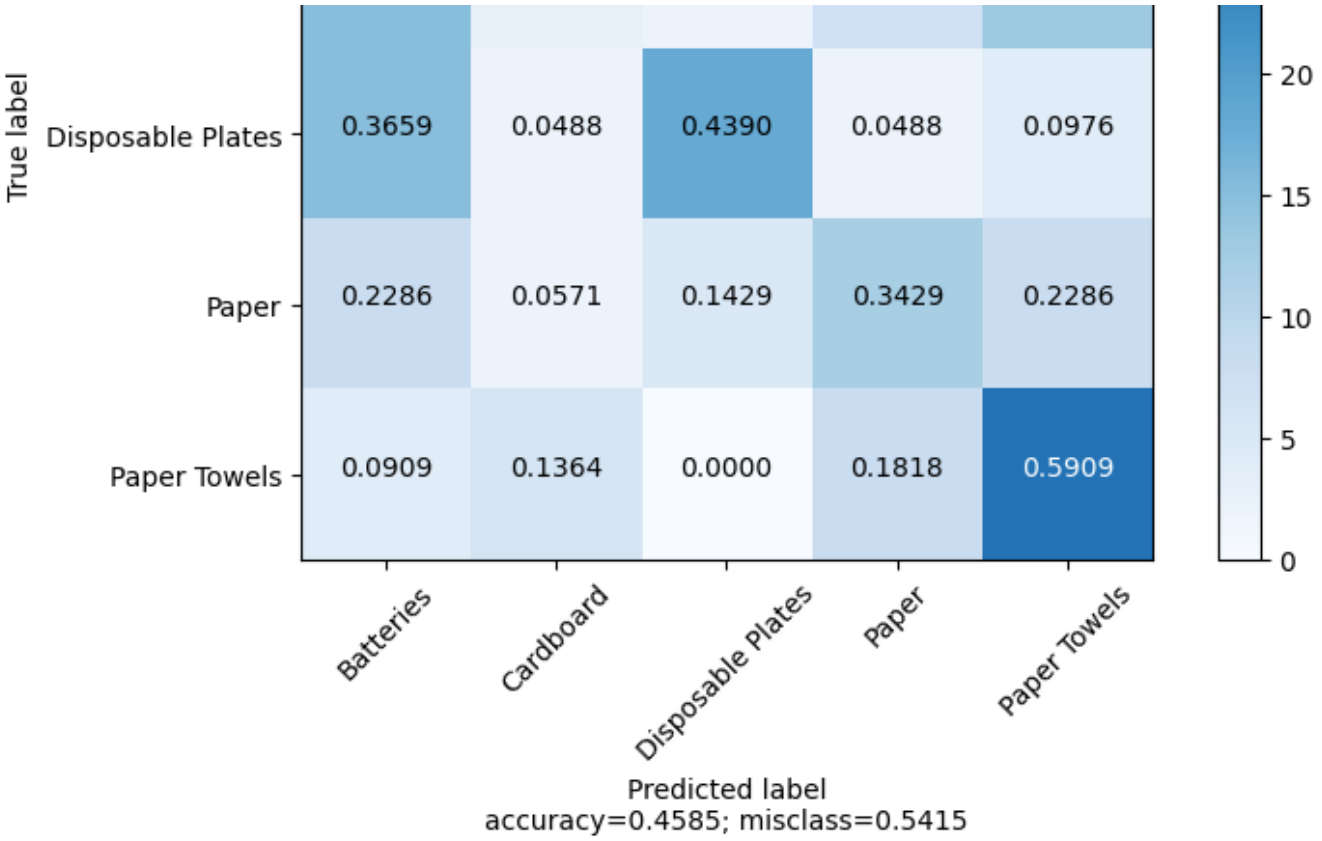

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.6552 0.0345 0.0690 0.0000 0.2414
0.3898 0.0678 0.0508 0.1525 0.3390
0.2545 0.0364 0.5636 0.0364 0.1091
0.1622 0.0541 0.0811 0.4865 0.2162
0.2979 0.1064 0.0213 0.1064 0.4681
35
30
25
20
15
10
5
0
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4414; misclass=0.5586


Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
True label
0.5111 0.1333 0.2000 0.0667 0.0889
0.1750 0.2000 0.1750 0.2000 0.2500
0.0732 0.1463 0.4146 0.1951 0.1707
0.2286 0.1714 0.2000 0.3143 0.0857
22.5
20.0
17.5
15.0
12.5
10.0
7.5


Paper Towels
0.2273
0.0682
0.0909
0.2273
0.3864
5.0
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.3707; misclass=0.6293

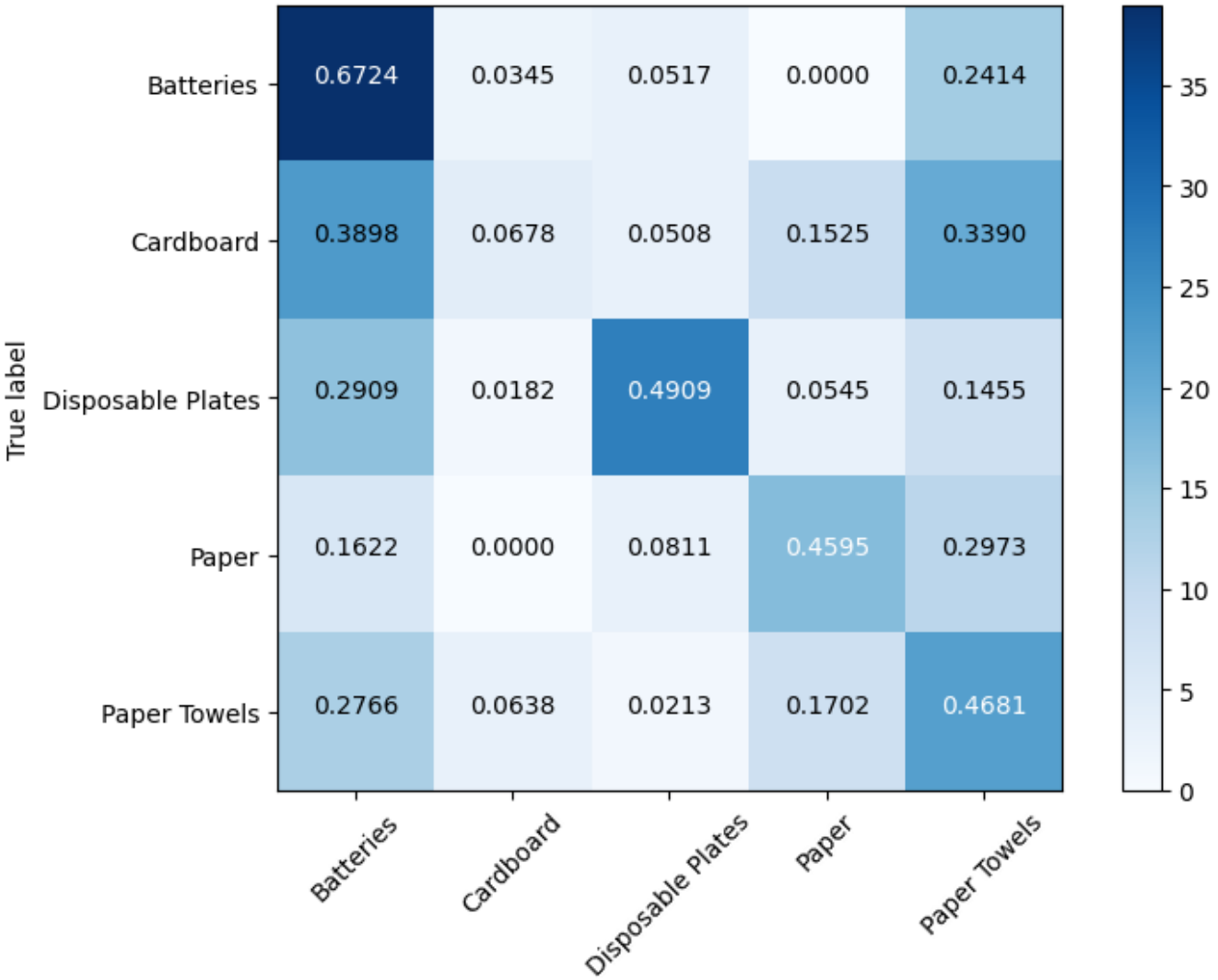

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.5000 0.0690 0.2241 0.0517 0.1552
0.3051 0.1356 0.0678 0.2542 0.2373
0.0909 0.0909 0.6000 0.1091 0.1091
0.0811 0.1351 0.1622 0.5135 0.1081
0.2128 0.1489 0.1277 0.1702 0.3404
30
25
20
15
10
5
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4102; misclass=0.5898

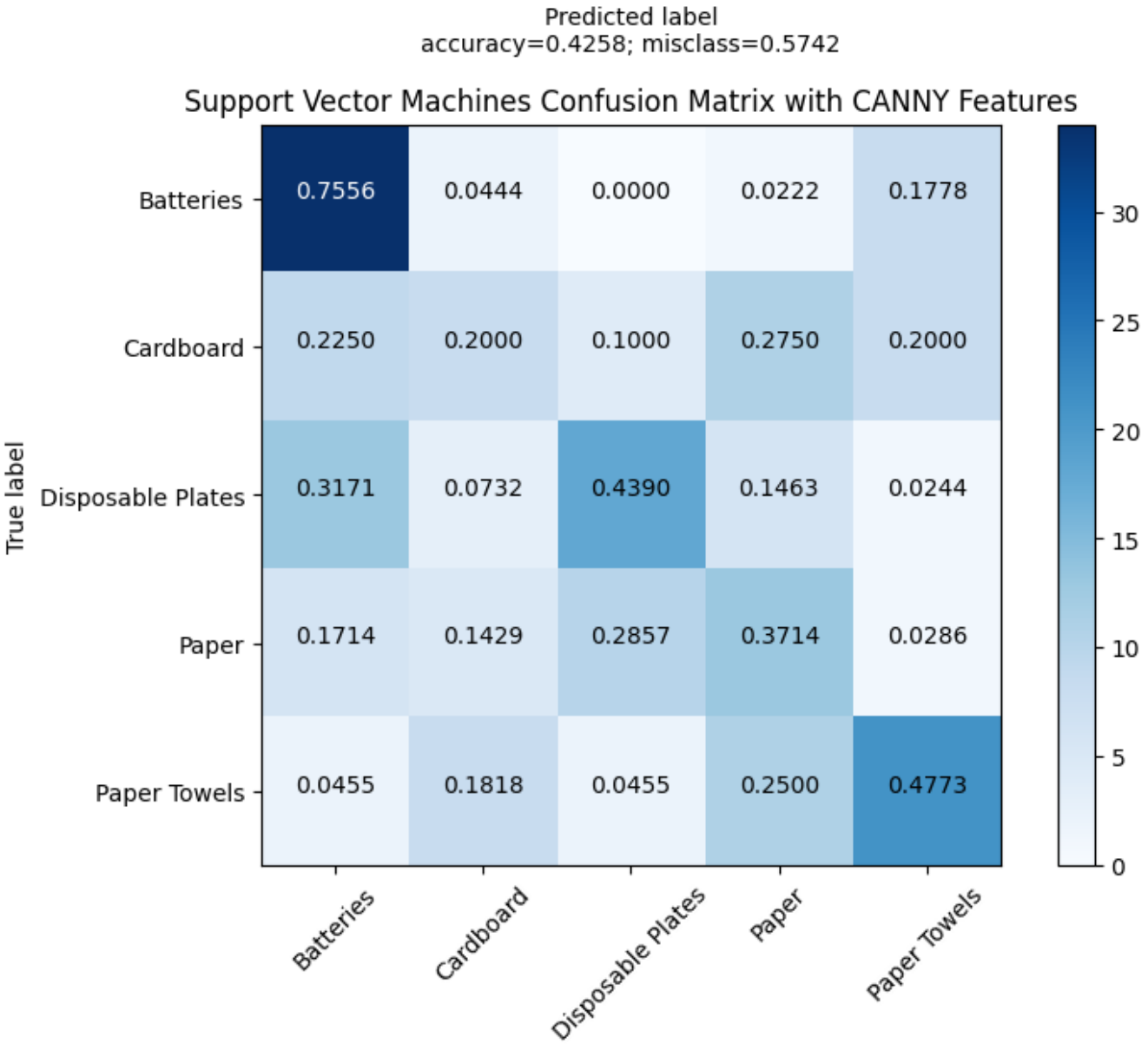

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
0.7778 0.0222 0.0000 0.0222 0.1778
0.3750 0.0750 0.0500 0.1750 0.3250
35
30
25

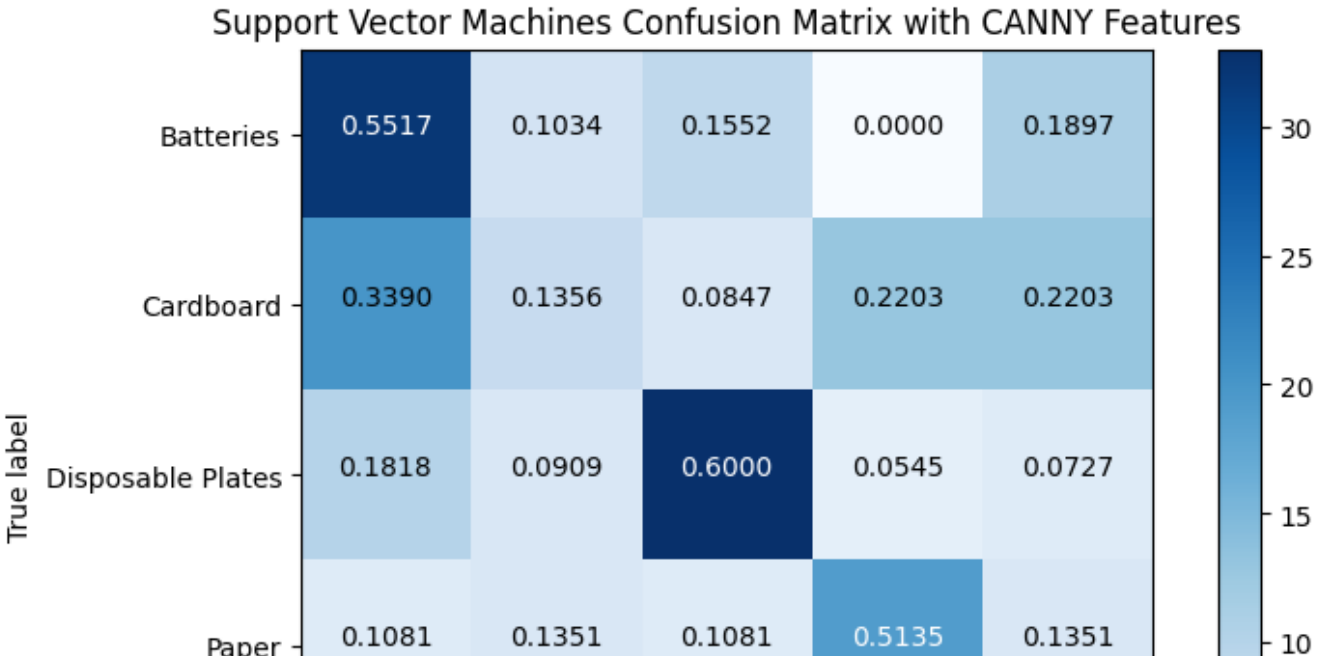


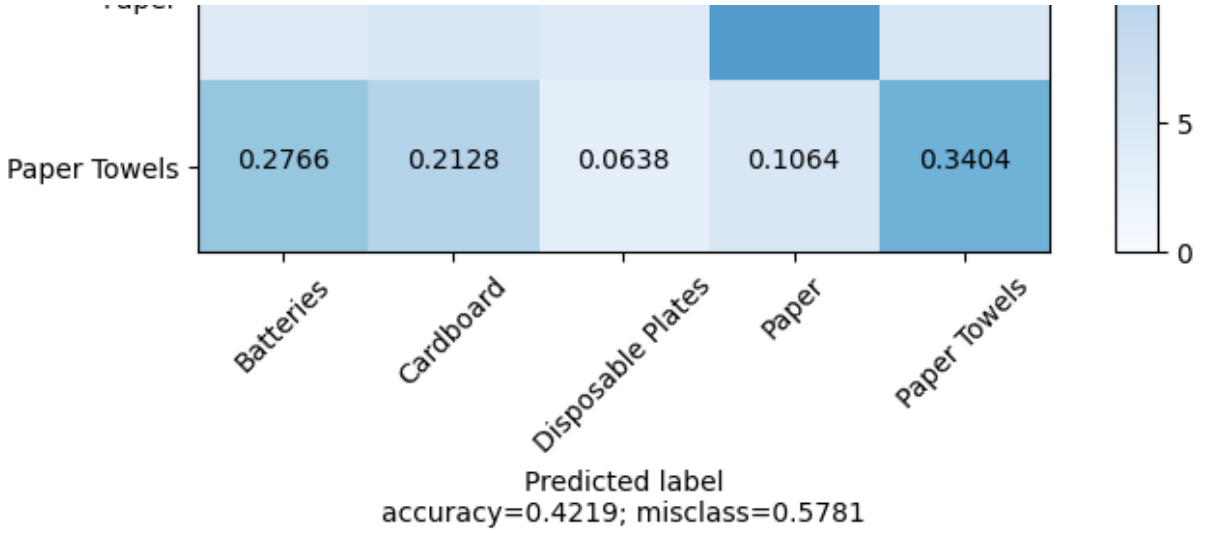
Predicted label
accuracy=0.4585; misclass=0.5415

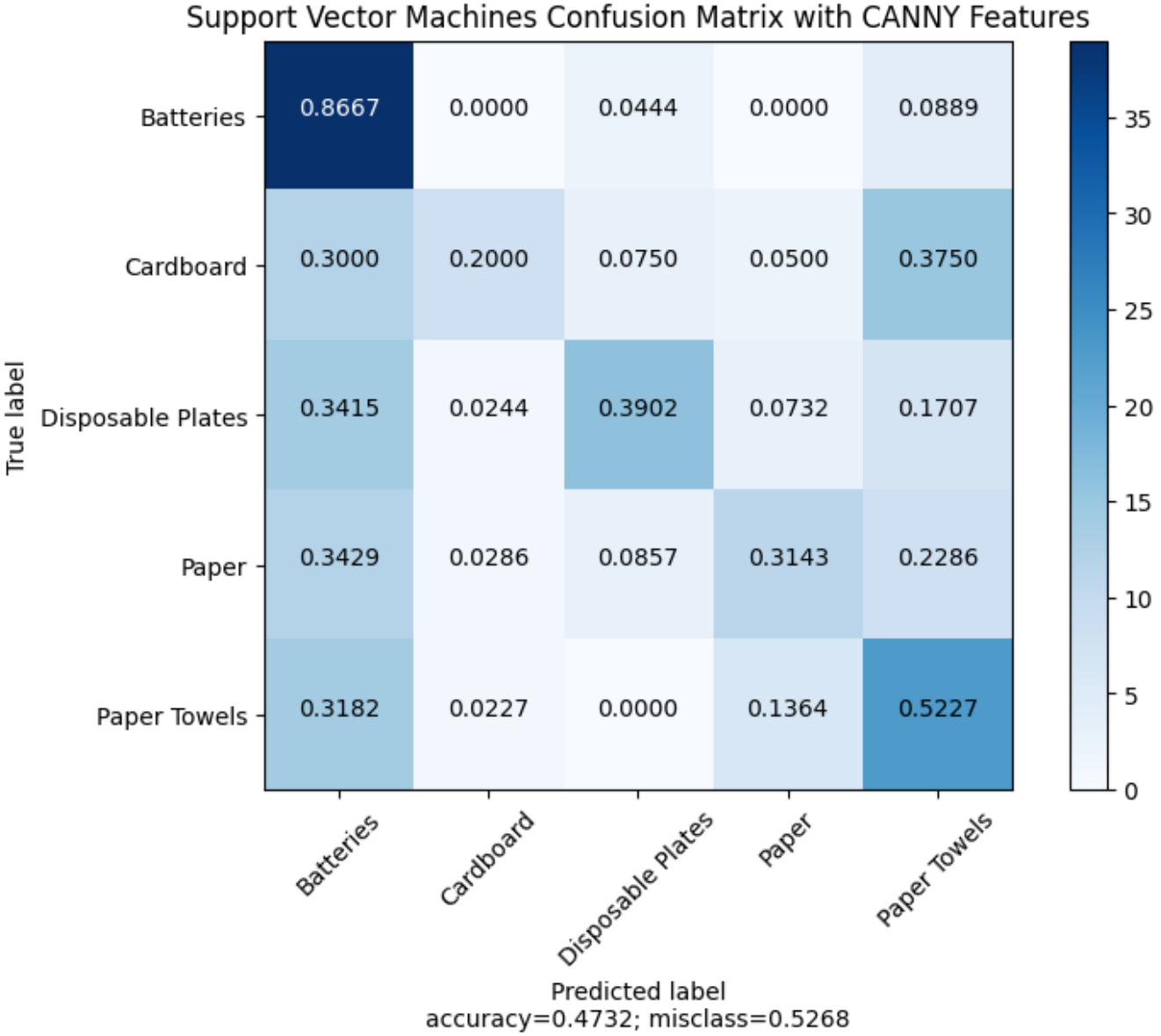
Support Vector Machines Confusion Matrix with CANNY Features

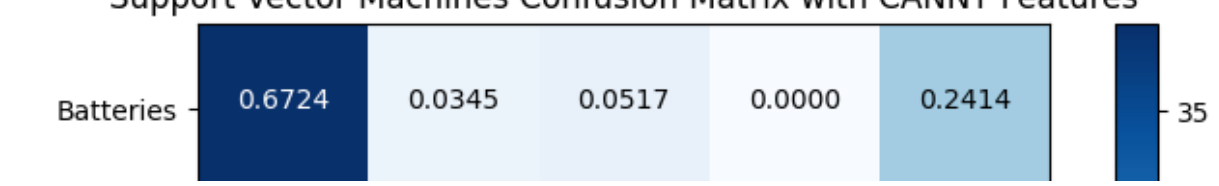


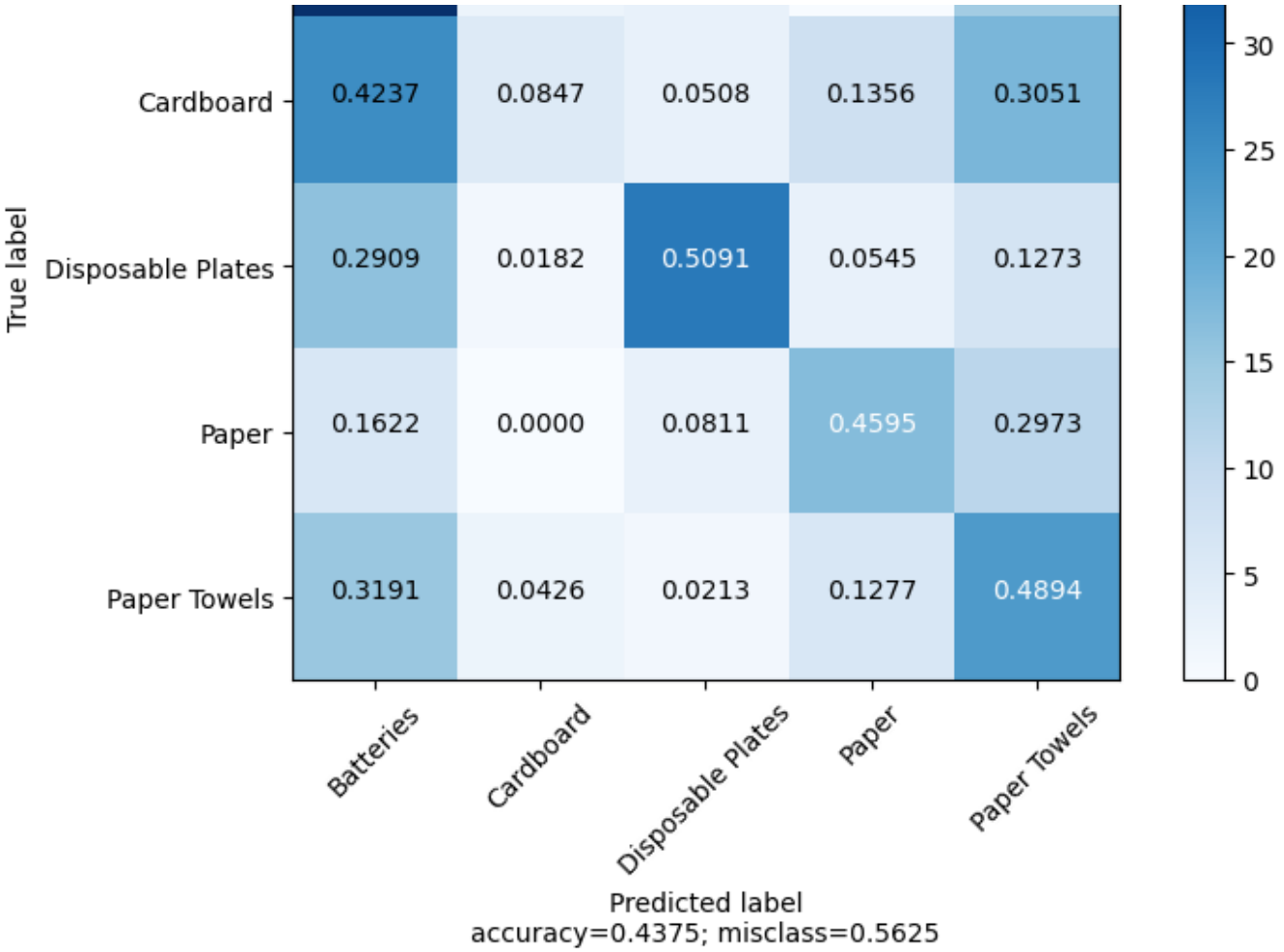

Predicted label
accuracy=0.4258; misclass=0.5742
Support Vector Machines Confusion Matrix with CANNY Features
True label
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
0.7556 0.0444 0.0000 0.0222 0.1778
0.2250 0.2000 0.1000 0.2750 0.2000
0.3171 0.0732 0.4390 0.1463 0.0244
0.1714 0.1429 0.2857 0.3714 0.0286
0.0455 0.1818 0.0455 0.2500 0.4773
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
30
25
20
15
10
5
0

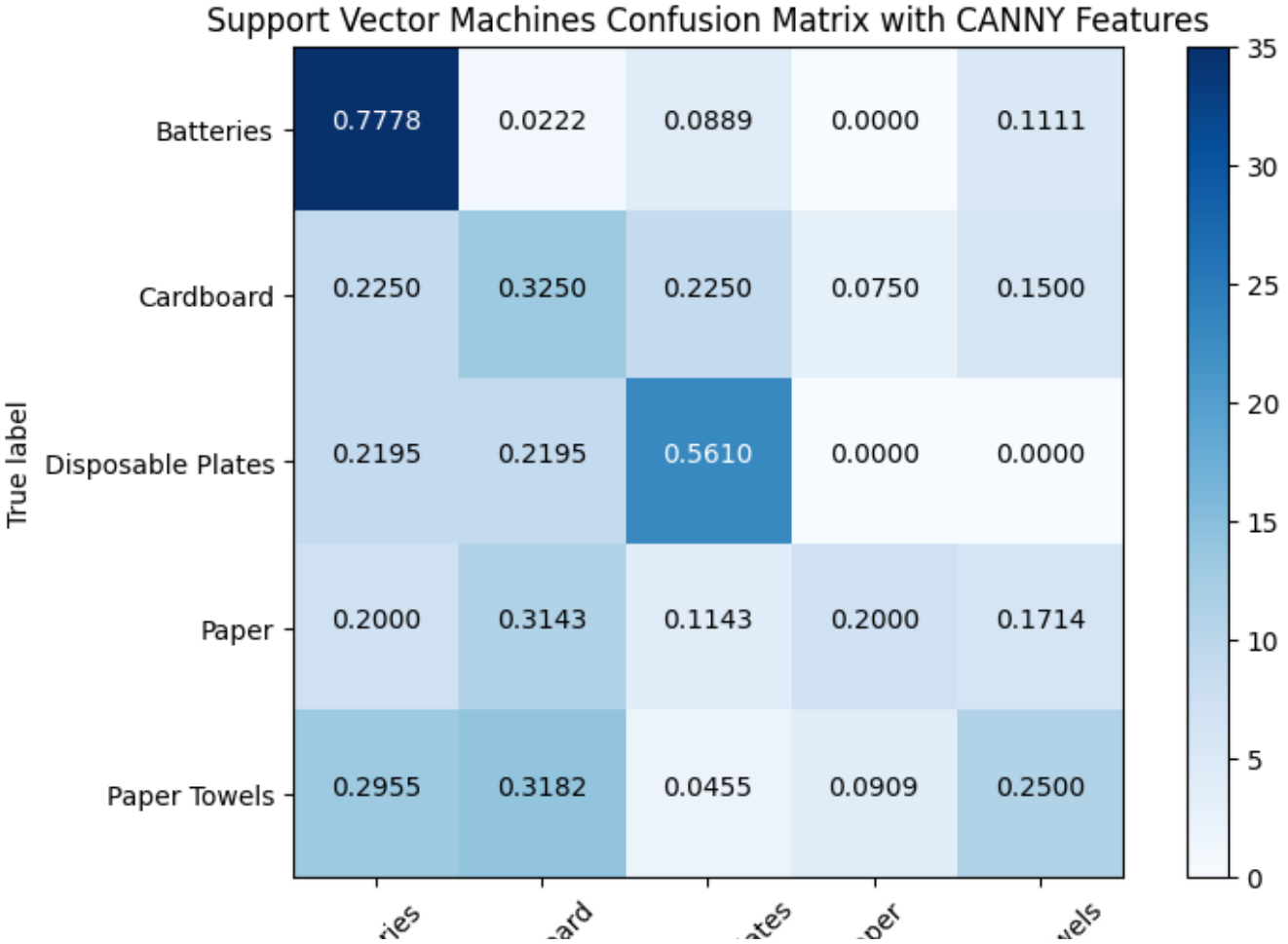

Predicted label
accuracy=0.4585; misclass=0.5415
Support Vector Machines Confusion Matrix with CANNY Features
True label
Batteries
Cardboard
Disposable Plates
Paper
0.5517 0.1034 0.1552 0.0000 0.1897
0.3390 0.1356 0.0847 0.2203 0.2203
0.1818 0.0909 0.6000 0.0545 0.0727
0.1081 0.1351 0.1081 0.5135 0.1351
30
25
20
15
10

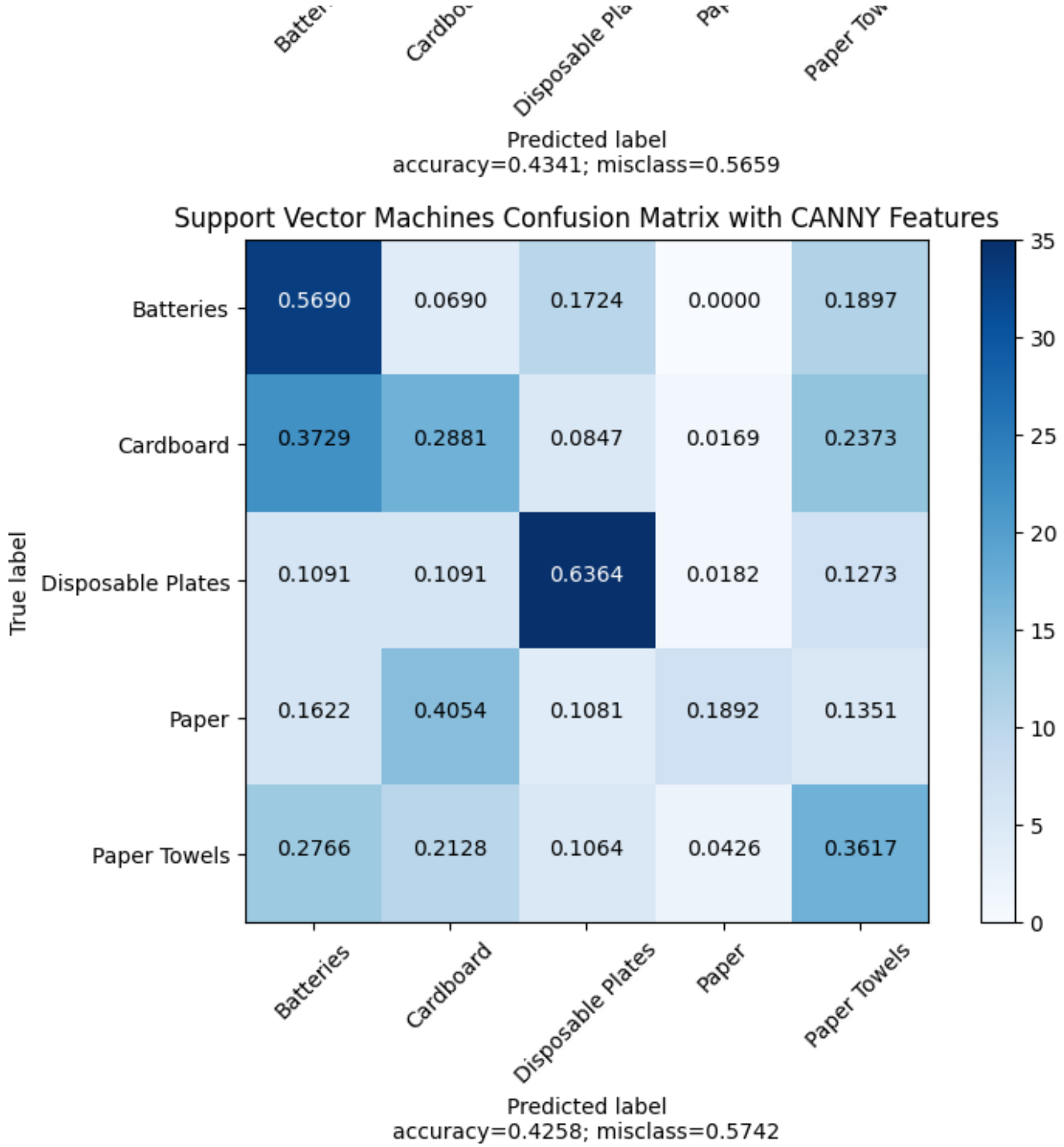

Paper Towels
0.2766
0.2128
0.0638
0.1064
0.3404
5
0
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4219; misclass=0.5781

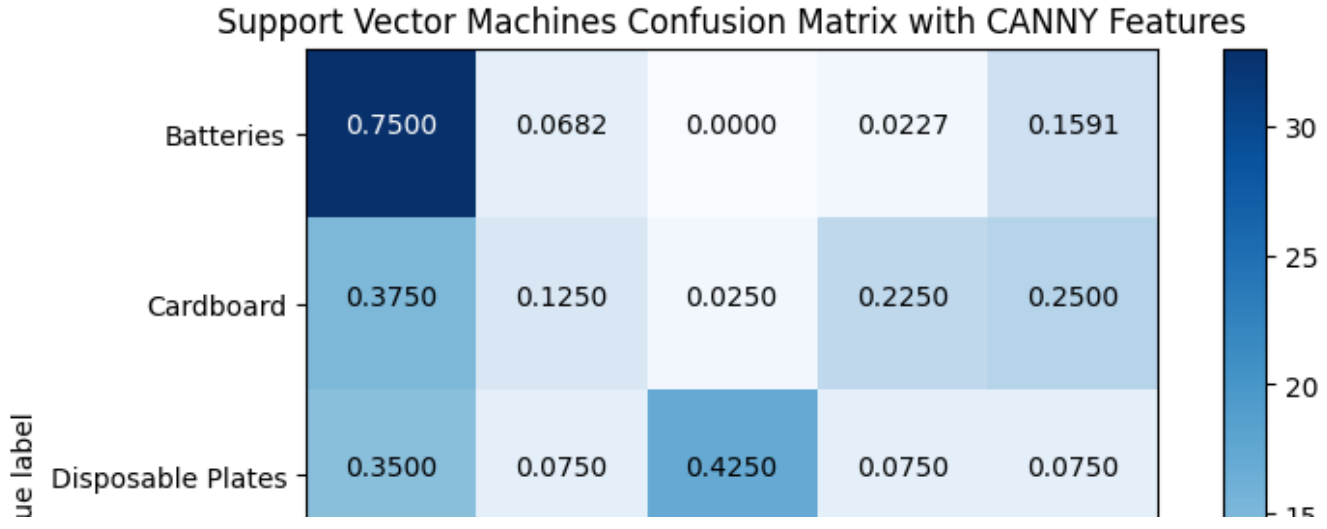

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
0.8667
0.0000
0.0444
0.0000
0.0889
0.3000
0.2000
0.0750
0.0500
0.3750
0.3415
0.0244
0.3902
0.0732
0.1707
0.3429
0.0286
0.0857
0.3143
0.2286
0.3182
0.0227
0.0000
0.1364
0.5227
True label
35
30
25
20
15
10
5
0
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4732; misclass=0.5268

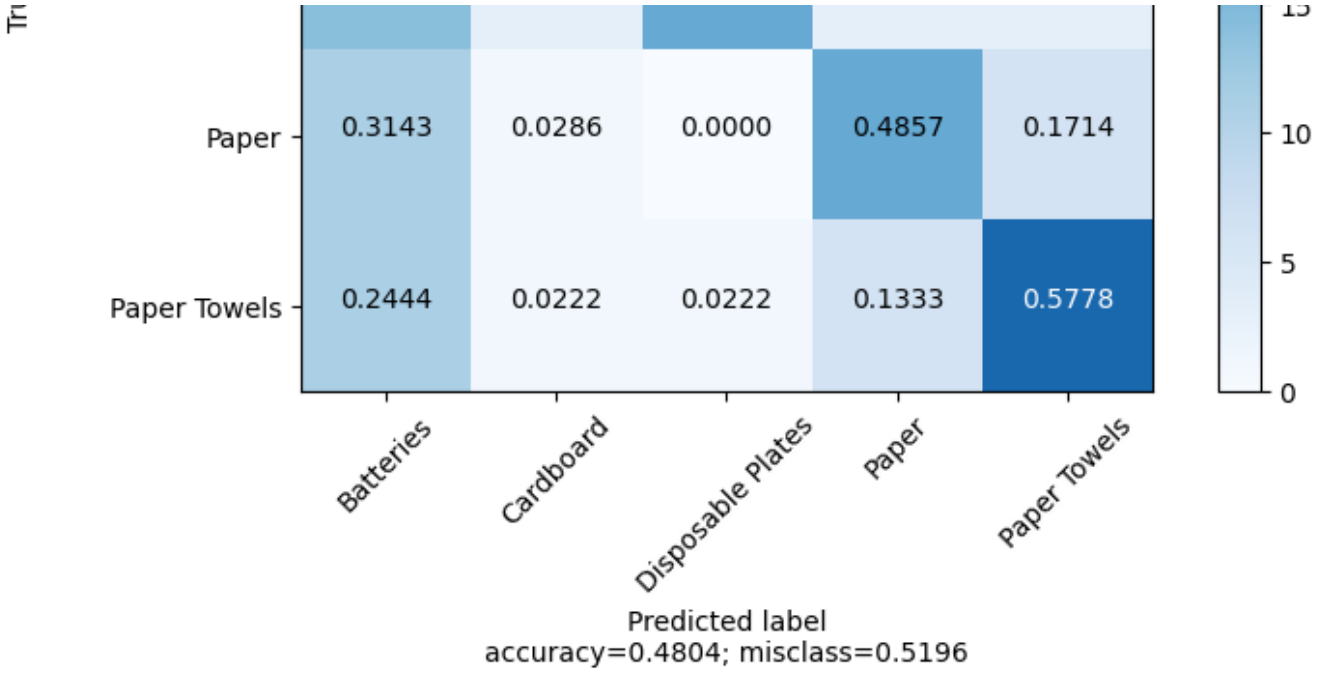

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
0.6724
0.0345
0.0517
0.0000
0.2414
35

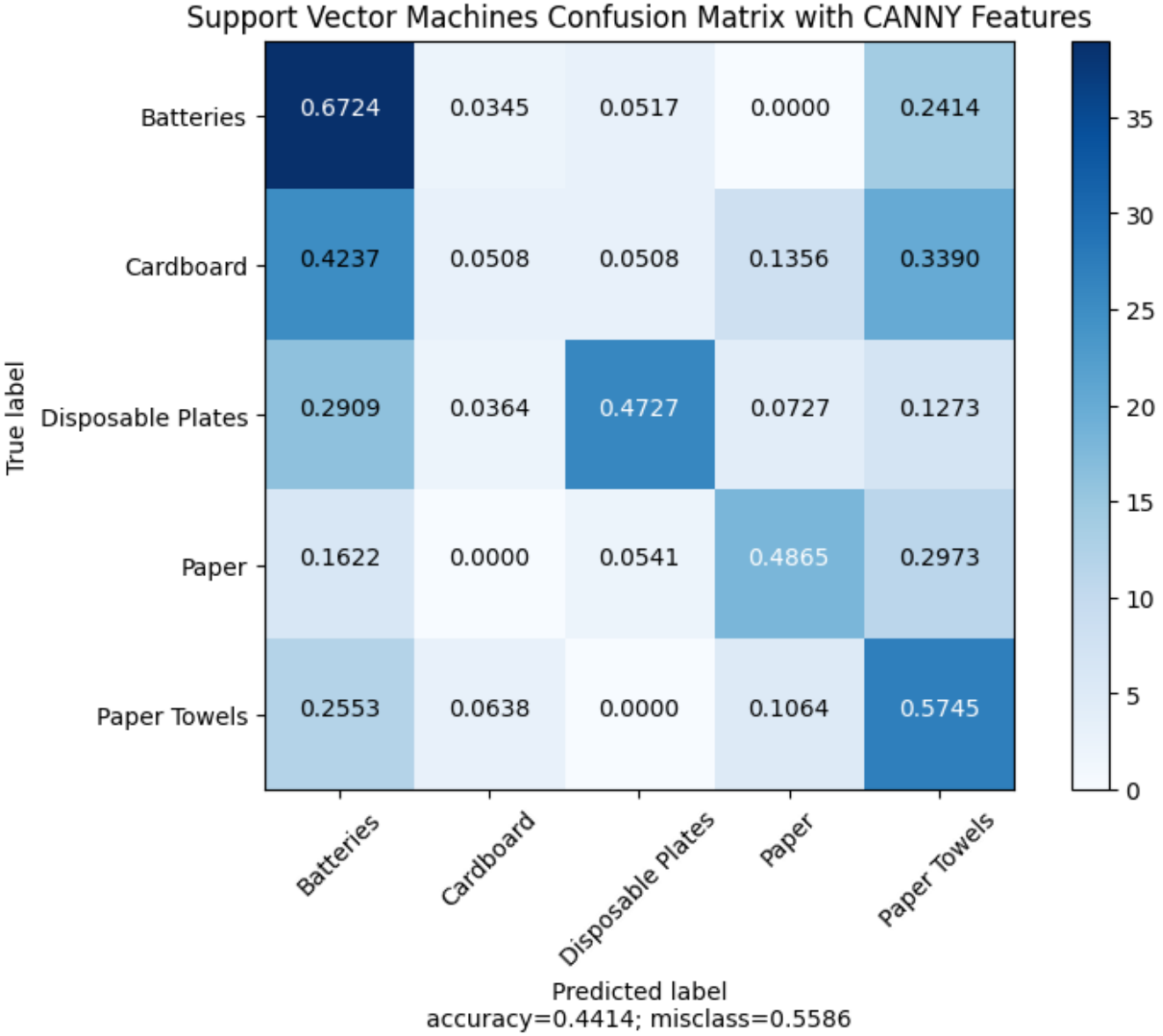

Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.4237 0.0847 0.0508 0.1356 0.3051
0.2909 0.0182 0.5091 0.0545 0.1273
0.1622 0.0000 0.0811 0.4595 0.2973
0.3191 0.0426 0.0213 0.1277 0.4894
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4375; misclass=0.5625
30
25
20
15
10
5
0

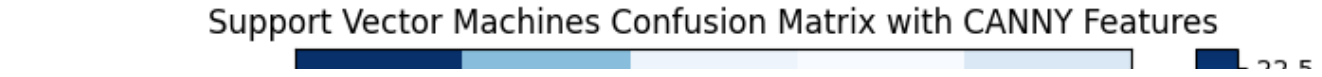

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.7778 0.0222 0.0889 0.0000 0.1111
0.2250 0.3250 0.2250 0.0750 0.1500
0.2195 0.2195 0.5610 0.0000 0.0000
0.2000 0.3143 0.1143 0.2000 0.1714
0.2955 0.3182 0.0455 0.0909 0.2500
35
30
25
20
15
10
5
0

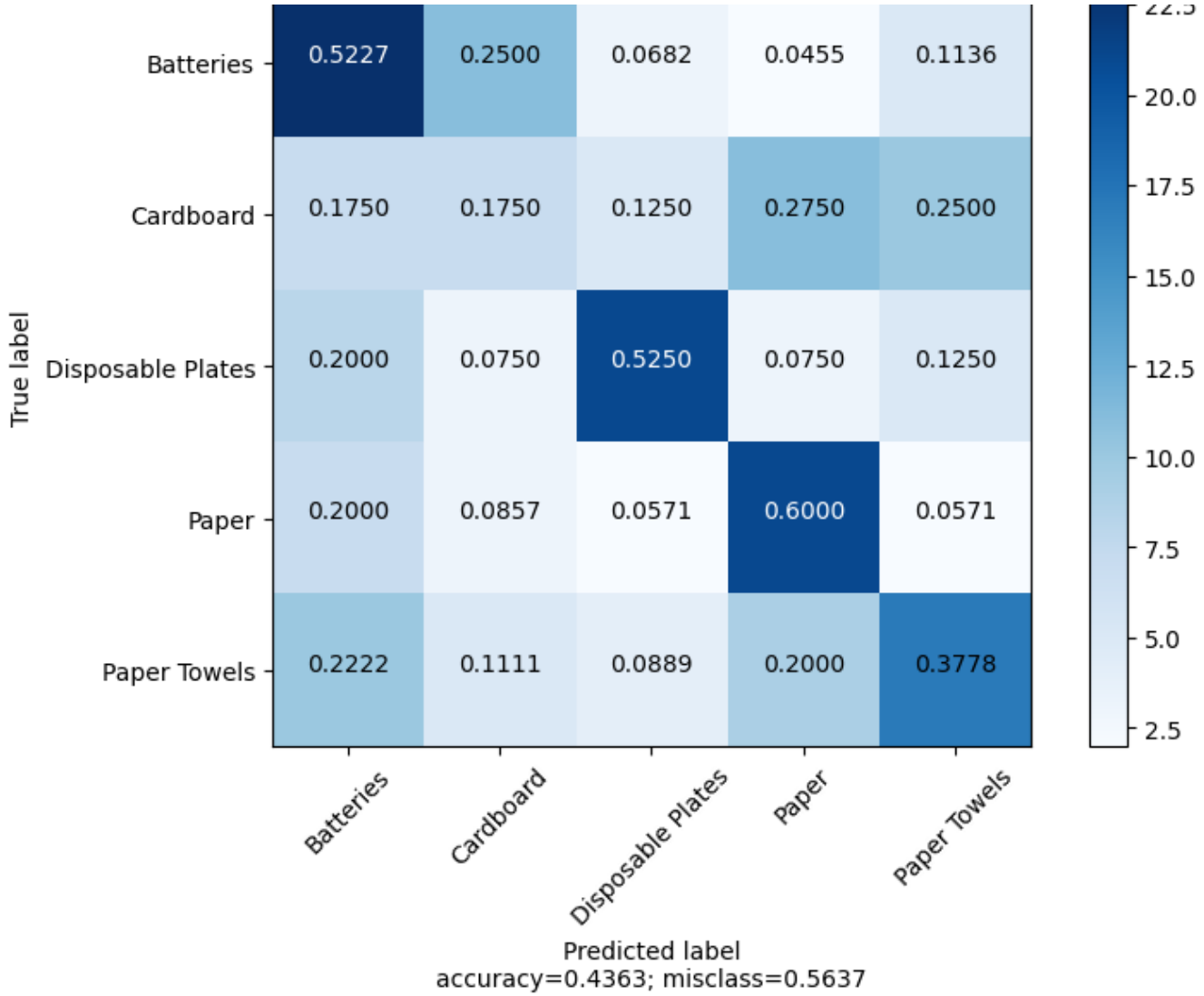

Batte
Cardbo
Disposable Pla
Pa
Paper Tow
Predicted label
accuracy=0.4341; misclass=0.5659
Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.5690 0.0690 0.1724 0.0000 0.1897
0.3729 0.2881 0.0847 0.0169 0.2373
0.1091 0.1091 0.6364 0.0182 0.1273
0.1622 0.4054 0.1081 0.1892 0.1351
0.2766 0.2128 0.1064 0.0426 0.3617
35
30
25
20
15
10
5
0
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4258; misclass=0.5742

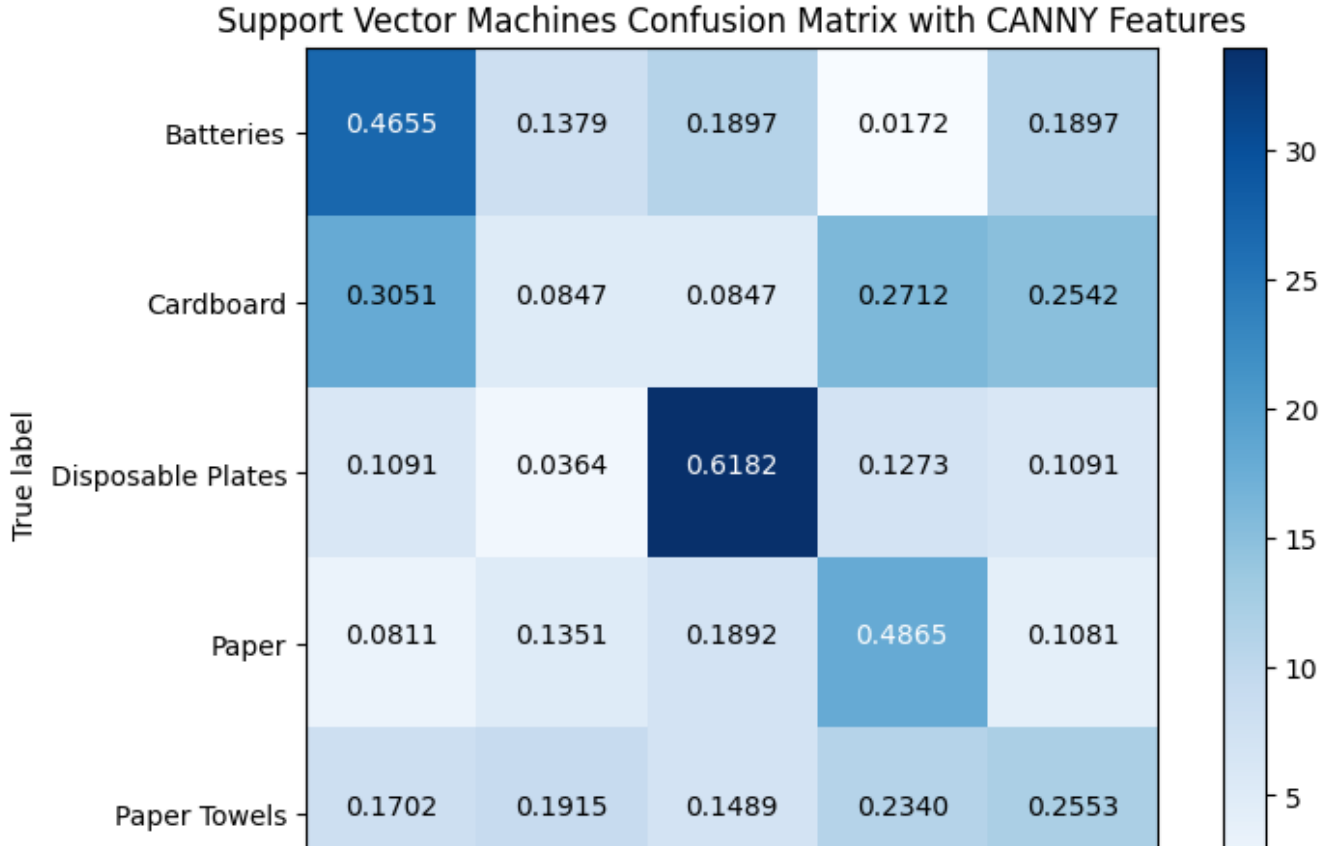

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
0.7500 0.0682 0.0000 0.0227 0.1591
0.3750 0.1250 0.0250 0.2250 0.2500
0.3500 0.0750 0.4250 0.0750 0.0750
30
25
20
15

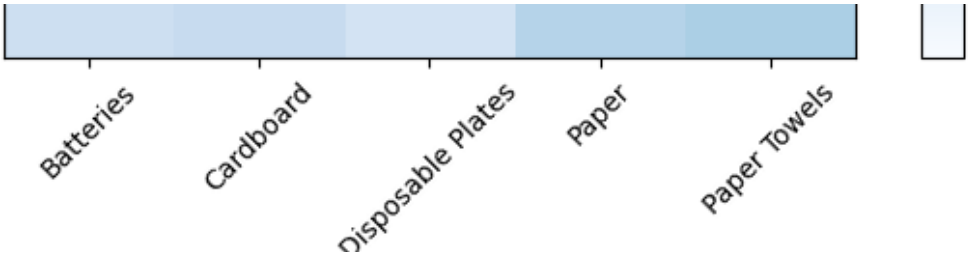

Paper
Paper Towels
0.3143
0.0286
0.0000
0.4857
0.1714
0.2444
0.0222
0.0222
0.1333
0.5778
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
15
10
5
0
Predicted label
accuracy=0.4804; misclass=0.5196


Support Vector Machines Confusion Matrix with CANNY Features
True label
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
0.6724
0.0345
0.0517
0.0000
0.2414
0.4237
0.0508
0.0508
0.1356
0.3390
0.2909
0.0364
0.4727
0.0727
0.1273
0.1622
0.0000
0.0541
0.4865
0.2973
0.2553
0.0638
0.0000
0.1064
0.5745
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
35
30
25
20
15
10
5
0
Predicted label
accuracy=0.4414; misclass=0.5586


Support Vector Machines Confusion Matrix with CANNY Features


Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.5227 0.2500 0.0682 0.0455 0.1136
0.1750 0.1750 0.1250 0.2750 0.2500
0.2000 0.0750 0.5250 0.0750 0.1250
0.2000 0.0857 0.0571 0.6000 0.0571
0.2222 0.1111 0.0889 0.2000 0.3778
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
Predicted label
accuracy=0.4363; misclass=0.5637

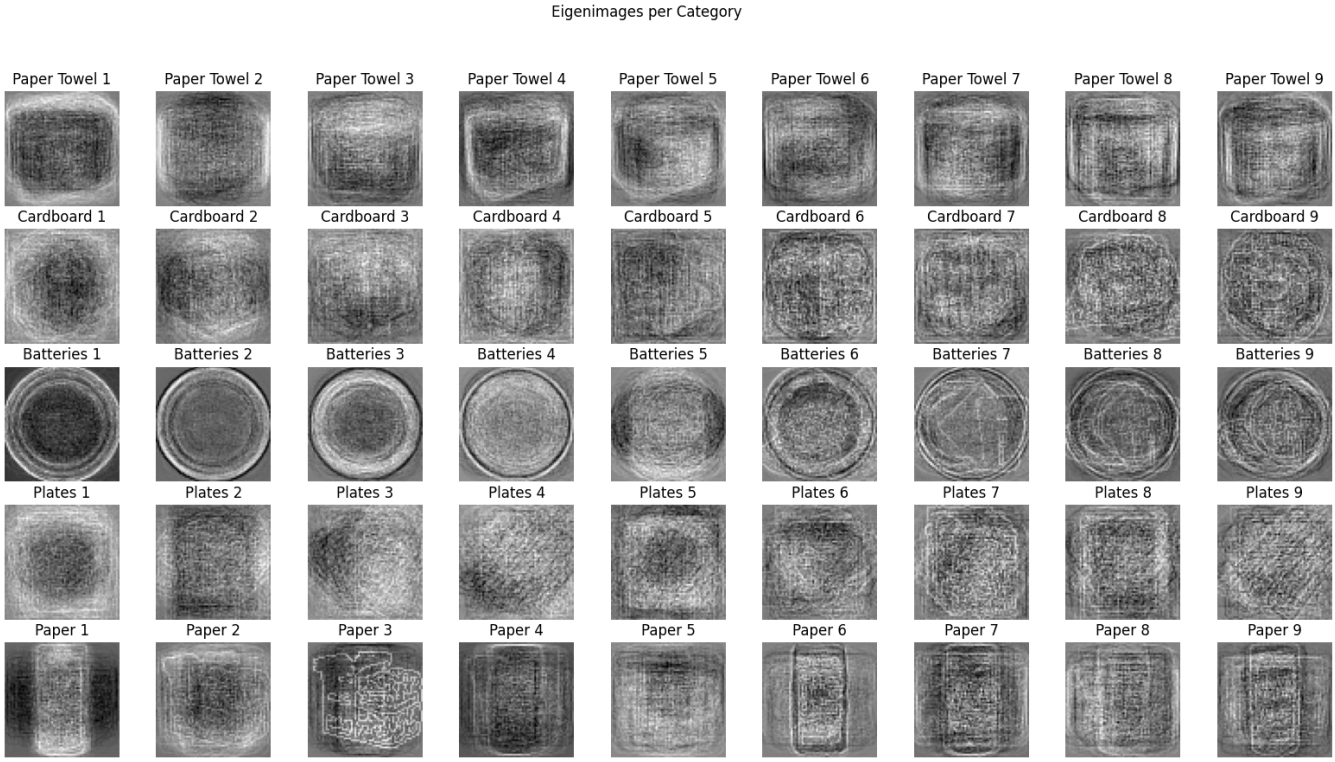

Support Vector Machines Confusion Matrix with CANNY Features
Batteries
Cardboard
Disposable Plates
Paper
Paper Towels
True label
0.4655 0.1379 0.1897 0.0172 0.1897
0.3051 0.0847 0.0847 0.2712 0.2542
0.1091 0.0364 0.6182 0.1273 0.1091
0.0811 0.1351 0.1892 0.4865 0.1081
0.1702 0.1915 0.1489 0.2340 0.2553
30
25
20
15
10
5



Canny Eigen Images

```
1 %%time
2 print("Eigen images for canny")
3 show_all_eigenimages(feature_canny_battery, feature_canny_cardboard, feature_canny_plates, feat
```

```
Eigen images for canny
Number of principal components explaining 95% of variance: 241
Number of principal components explaining 95% of variance: 205
Number of principal components explaining 95% of variance: 204
Number of principal components explaining 95% of variance: 149
Number of principal components explaining 95% of variance: 214
CPU times: user 5.58 s, sys: 8.62 s, total: 14.2 s
Wall time: 1.5 s
```



Canny SVM + PCA

```
1   %%time
2   reconstructed_canny_batteries_data, canny_batteries_pca_component_count_list = apply_pca
3   reconstructed_canny_cardboard_data, canny_cardboard_pca_component_count_list = apply_pca
4   reconstructed_canny_plates_data, canny_plates_pca_component_count_list = apply_pca_and_r
5   reconstructed_canny_paper_data, canny_paper_pca_component_count_list = apply_pca_and_rec
6   reconstructed_canny_paper_towel_data, canny_paper_towel_pca_component_count_list = apply
```

```
 7
 8  canny_reconstructed_image = np.concatenate((flatten_np_arry(reconstructed_canny_batterie
 9                                              flatten_np_arry(reconstructed_canny_cardboard_
10                                              flatten_np_arry(reconstructed_canny_plates_dat
11                                              flatten_np_arry(reconstructed_canny_paper_data
12                                              flatten_np_arry(reconstructed_canny_paper_towe
13                                             )
14                                            )
15  X_canny_recon, y_canny_recon = canny_reconstructed_image[shuffle], label[shuffle]
16  #raw_data = np.concatenate((X,y))
17  df_canny_recon = pd.DataFrame(X_canny_recon)
18  df_canny_recon['label'] = y_canny_recon
19
20  preprocess_and_classification_svm(df_canny_recon,
21                                    label_list,
22                                    "Support Vector Machines Confusion Matrix with Canny R
23                                    "SVM_Canny_PCA_Image.png"
24                                   )
```

```
************************************************************
kfold : 1
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.37      0.76      0.50        45
        Cardboard       0.39      0.27      0.32        41
Disposable Plates       0.78      0.44      0.56        41
            Paper       0.38      0.30      0.34        33
     Paper Towels       0.58      0.47      0.52        45

         accuracy                           0.46       205
        macro avg       0.50      0.45      0.45       205
     weighted avg       0.51      0.46      0.45       205

Test results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.45      0.83      0.59        58
        Cardboard       0.26      0.17      0.21        53
Disposable Plates       0.62      0.35      0.44        52
            Paper       0.50      0.45      0.47        49
     Paper Towels       0.40      0.39      0.39        44

         accuracy                           0.45       256
        macro avg       0.45      0.44      0.42       256
     weighted avg       0.45      0.45      0.42       256

{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1, gamma=0.0001)
Validation grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.41      0.73      0.53        45
        Cardboard       0.28      0.17      0.21        41
Disposable Plates       0.70      0.39      0.50        41
            Paper       0.42      0.24      0.31        33
     Paper Towels       0.41      0.53      0.47        45

         accuracy                           0.43       205
        macro avg       0.44      0.41      0.40       205
     weighted avg       0.44      0.43      0.41       205

Test grid search results for fold 1
                  precision    recall  f1-score   support

        Batteries       0.47      0.78      0.58        58
        Cardboard       0.35      0.13      0.19        53
Disposable Plates       0.66      0.37      0.47        52
            Paper       0.47      0.31      0.37        49
     Paper Towels       0.28      0.50      0.36        44

         accuracy                           0.42       256
        macro avg       0.44      0.42      0.39       256
```

```
     weighted avg       0.45      0.42      0.40       256

************************************************************
kfold : 2
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.41      0.78      0.53        45
        Cardboard       0.29      0.12      0.17        41
Disposable Plates       0.67      0.43      0.52        42
            Paper       0.50      0.44      0.47        32
     Paper Towels       0.47      0.49      0.48        45

         accuracy                           0.46       205
        macro avg       0.47      0.45      0.43       205
     weighted avg       0.47      0.46      0.44       205

Test results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.45      0.84      0.58        58
        Cardboard       0.32      0.17      0.22        53
Disposable Plates       0.63      0.33      0.43        52
            Paper       0.51      0.41      0.45        49
     Paper Towels       0.42      0.50      0.46        44

         accuracy                           0.46       256
        macro avg       0.47      0.45      0.43       256
     weighted avg       0.47      0.46      0.43       256

{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.0001)
Validation grid search results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.45      0.62      0.52        45
        Cardboard       0.28      0.20      0.23        41
Disposable Plates       0.60      0.67      0.63        42
            Paper       0.53      0.59      0.56        32
     Paper Towels       0.48      0.33      0.39        45

         accuracy                           0.48       205
        macro avg       0.47      0.48      0.47       205
     weighted avg       0.46      0.48      0.46       205

Test grid search results for fold 2
                  precision    recall  f1-score   support

        Batteries       0.47      0.72      0.57        58
        Cardboard       0.22      0.21      0.21        53
Disposable Plates       0.64      0.52      0.57        52
            Paper       0.46      0.37      0.41        49
     Paper Towels       0.39      0.32      0.35        44

         accuracy                           0.44       256
```

```
          accuracy                           0.44       256
         macro avg       0.44      0.43      0.42       256
      weighted avg       0.44      0.44      0.43       256

************************************************************
kfold : 3
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.45      0.78      0.57        45
        Cardboard       0.37      0.27      0.31        41
Disposable Plates       0.62      0.43      0.51        42
            Paper       0.61      0.44      0.51        32
     Paper Towels       0.43      0.44      0.44        45

         accuracy                           0.48       205
        macro avg       0.50      0.47      0.47       205
     weighted avg       0.49      0.48      0.47       205

Test results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.44      0.83      0.57        58
        Cardboard       0.37      0.19      0.25        53
Disposable Plates       0.66      0.37      0.47        52
            Paper       0.56      0.41      0.47        49
     Paper Towels       0.39      0.48      0.43        44

         accuracy                           0.46       256
        macro avg       0.48      0.45      0.44       256
     weighted avg       0.48      0.46      0.44       256

{'C': 1000, 'gamma': 1e-06, 'kernel': 'rbf'}
SVC(C=1000, gamma=1e-06)
Validation grid search results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.50      0.58      0.54        45
        Cardboard       0.32      0.29      0.31        41
Disposable Plates       0.63      0.64      0.64        42
            Paper       0.56      0.59      0.58        32
     Paper Towels       0.51      0.44      0.48        45

         accuracy                           0.51       205
        macro avg       0.50      0.51      0.51       205
     weighted avg       0.50      0.51      0.50       205

Test grid search results for fold 3
                  precision    recall  f1-score   support

        Batteries       0.50      0.64      0.56        58
        Cardboard       0.21      0.17      0.19        53
Disposable Plates       0.49      0.46      0.48        52
            Paper       0.35      0.35      0.35        49
     Paper Towels       0.35      0.34      0.34        44
```

```
          accuracy                           0.40       256
         macro avg       0.38      0.39      0.38       256
      weighted avg       0.38      0.40      0.39       256

************************************************************
kfold : 4
(819, 4096) (819,) (205, 4096) (205,) (256, 4096) (256,)
****
Validation results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.42      0.82      0.55        44
        Cardboard       0.35      0.17      0.23        42
Disposable Plates       0.79      0.37      0.50        41
            Paper       0.58      0.33      0.42        33
     Paper Towels       0.43      0.58      0.49        45

         accuracy                           0.46       205
        macro avg       0.51      0.45      0.44       205
     weighted avg       0.51      0.46      0.44       205

Test results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.44      0.79      0.56        58
        Cardboard       0.44      0.15      0.23        53
Disposable Plates       0.70      0.31      0.43        52
            Paper       0.49      0.37      0.42        49
     Paper Towels       0.34      0.57      0.43        44

         accuracy                           0.44       256
        macro avg       0.48      0.44      0.41       256
     weighted avg       0.48      0.44      0.41       256

{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.0001)
Validation grid search results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.41      0.68      0.51        44
        Cardboard       0.42      0.31      0.36        42
Disposable Plates       0.67      0.39      0.49        41
            Paper       0.47      0.48      0.48        33
     Paper Towels       0.40      0.38      0.39        45

         accuracy                           0.45       205
        macro avg       0.47      0.45      0.45       205
     weighted avg       0.47      0.45      0.44       205

Test grid search results for fold 4
                   precision    recall  f1-score   support

        Batteries       0.46      0.71      0.56        58
        Cardboard       0.12      0.08      0.09        53
Disposable Plates       0.59      0.42      0.49        52
            Paper       0.39      0.41      0.40        49
```

```
                Paper       0.39      0.41      0.40        49
      Paper Towels       0.34      0.36      0.35        44

          accuracy                           0.40       256
         macro avg       0.38      0.40      0.38       256
      weighted avg       0.38      0.40      0.38       256

************************************************************
kfold : 5
(820, 4096) (820,) (204, 4096) (204,) (256, 4096) (256,)
****
Validation results for fold 5
                    precision    recall  f1-score   support

         Batteries       0.38      0.69      0.49        45
         Cardboard       0.38      0.20      0.26        41
Disposable Plates       0.62      0.44      0.51        41
             Paper       0.42      0.30      0.35        33
      Paper Towels       0.37      0.41      0.39        44

          accuracy                           0.42       204
         macro avg       0.43      0.41      0.40       204
      weighted avg       0.43      0.42      0.40       204

Test results for fold 5
                    precision    recall  f1-score   support

         Batteries       0.45      0.78      0.57        58
         Cardboard       0.43      0.17      0.24        53
Disposable Plates       0.63      0.37      0.46        52
             Paper       0.55      0.43      0.48        49
      Paper Towels       0.37      0.57      0.45        44

          accuracy                           0.46       256
         macro avg       0.49      0.46      0.44       256
      weighted avg       0.49      0.46      0.44       256

{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=1000, gamma=0.0001)
Validation grid search results for fold 5
                    precision    recall  f1-score   support

         Batteries       0.38      0.53      0.44        45
         Cardboard       0.27      0.32      0.29        41
Disposable Plates       0.49      0.46      0.48        41
             Paper       0.18      0.09      0.12        33
      Paper Towels       0.40      0.32      0.35        44

          accuracy                           0.36       204
         macro avg       0.34      0.34      0.34       204
      weighted avg       0.35      0.36      0.35       204

Test grid search results for fold 5
                    precision    recall  f1-score   support

         Batteries       0.49      0.72      0.59        58
         Cardboard       0.27      0.30      0.28        53
```

```
Disposable Plates       0.55      0.42      0.48        52
            Paper       0.24      0.12      0.16        49
      Paper Towels      0.35      0.36      0.36        44

         accuracy                           0.40       256
        macro avg       0.38      0.39      0.37       256
     weighted avg       0.38      0.40      0.38       256
```

Overall Stats....

	kpi_level	min_accuracy	avg_accuracy	max_accuracy
0	base_val_accuracy	0.416667	0.455041	0.478049
1	grid_val_accuracy	0.357843	0.444252	0.507317
2	base_test_accuracy	0.441406	0.453906	0.464844
3	grid_test_accuracy	0.398438	0.411719	0.437500

Category Wise Stats...

	0	1	2	3
kpi	cm_test_base	cm_test_grid	cm_val_base	cm_val_grid
battery_correct_pct_min	0.775862	0.637931	0.688889	0.533333
battery_correct_pct_avg	0.813793	0.713793	0.763636	0.629697
battery_correct_pct_max	0.844828	0.775862	0.818182	0.733333
cardboard_correct_pct_min	0.150943	0.075472	0.121951	0.170732
cardboard_correct_pct_avg	0.169811	0.177358	0.204065	0.257027
cardboard_correct_pct_max	0.188679	0.301887	0.268293	0.317073
plates_correct_pct_min	0.307692	0.365385	0.365854	0.390244
plates_correct_pct_avg	0.342308	0.438462	0.420209	0.510685
plates_correct_pct_max	0.365385	0.519231	0.439024	0.666667
paper_correct_pct_min	0.367347	0.122449	0.30303	0.090909
paper_correct_pct_avg	0.412245	0.310204	0.362879	0.401136
paper_correct_pct_max	0.44898	0.408163	0.4375	0.59375
paper_towel_correct_pct_min	0.386364	0.318182	0.409091	0.318182
paper_towel_correct_pct_avg	0.5	0.377273	0.477374	0.401414
paper_towel_correct_pct_max	0.568182	0.5	0.577778	0.533333

```
CPU times: user 6min 25s, sys: 1min 6s, total: 7min 32s
Wall time: 6min 15s
```