In [1]:

```python
#imported libs
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.utils import get_file
import os
```

In [2]:

```python
#dataset_dir  = r"E:\APPS\PythonDataSets\caltech\caltech-101\101_ObjectCategories\1
dataset_dir = r"E:\APPS\PythonDataSets\caltech\caltech-101\101_ObjectCategories\fil




# Parameters
batch_size = 32
image_size = (64, 128)

# Load the training and validation datasets
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=image_size,
    batch_size=batch_size
)

val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=image_size,
    batch_size=batch_size
)


def dataset_to_numpy(dataset):
    """
    Convert a tf.data.Dataset into NumPy arrays for features and labels.
    Args:
        dataset: A tf.data.Dataset object.
    Returns:
        X: Numpy array of features (images).
        y: Numpy array of labels.
    """
    X = []
    y = []
    for images, labels in dataset:
        X.append(images.numpy())
        y.append(labels.numpy())
    return np.concatenate(X, axis=0), np.concatenate(y, axis=0)

# Convert the train and validation datasets to NumPy arrays
```

```
X_train, y_train = dataset_to_numpy(train_dataset)
X_test, y_test = dataset_to_numpy(val_dataset)

X_test =[cv2.resize(img.astype(np.uint8), (64, 128)) for img in X_test]
X_train=[cv2.resize(img.astype(np.uint8), (64, 128)) for img in X_train]
```

```
Found 9145 files belonging to 101 classes.
Using 7316 files for training.
Found 9145 files belonging to 101 classes.
Using 1829 files for validation.
```

In [4]:

```
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (7316, 200, 200, 3)
y_train shape: (7316,)
X_test shape: (1829, 200, 200, 3)
y_test shape: (1829,)
```

In [3]:

```
#Accuracy
def accuracy(y_test1, y_pred1):
    y_pred1 = np.array(y_pred1)
    counter = 0
    for i in range(len(y_pred1)):
      if (y_pred1[i] == y_test1[i]):
        counter += 1
    accuracy = counter / len(y_pred1)
    accuracy *= 100
    return accuracy


import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

In [4]:

```
# visualizing the results
def visualize_results(y_test,y_predict):
    class_names = [folder for folder in os.listdir(r"E:\APPS\PythonDataSets\caltech
    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_predict)

    # Calculate accuracy for each class
    class_accuracies = (cm.diagonal() / cm.sum(axis=1)) * 100

    # Display class-wise accuracy
    #classes = [f"Class {i}" for i in range(len(class_accuracies))]  # Replace with

    # Plot the accuracies
    plt.figure(figsize=(15, 5))
    plt.bar(class_names, class_accuracies, color='skyblue')
    plt.xlabel("Classes")
    plt.ylabel("Accuracy (%)")
    plt.title("Class-Wise Accuracy")
    plt.xticks(rotation=90)
    plt.ylim(0, 100)  # Accuracy is in percentage
```

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

    # Print the class-wise accuracy
    #for i, accuracy in enumerate(class_accuracies):
        #print(f"Accuracy for {class_names[i]}: {accuracy:.2f}%")
```

In [5]:
```python
#Color Histogran Extraction def
def extract_color_histogram(image, bins=(8, 8, 8)):
    """
    Extract a 3D color histogram from an RGB image.
    Args:
        image (numpy array): Input image in RGB format.
        bins (tuple): Number of bins for each channel (R, G, B).
    Returns:
        numpy array: Flattened color histogram feature vector.
    """
    # Calculate the 3D histogram for the HSV channels
    hist = cv2.calcHist([image], [0, 1, 2], None, bins, [0, 256, 0, 256, 0, 256])
    #print(hist.shape)
    # Normalize the histogram to ensure invariance to lighting changes

    #hist = cv2.normalize(hist, hist).flatten()


    return hist.flatten()
```

In [ ]:
```python
#HOG def
def extract_hog_features(image):
    # HOG parameters
    winSize = (64, 128)
    blockSize = (16, 16)
    blockStride = (8, 8)
    cellSize = (8, 8)
    nbins = 9

    hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins)
    hog_features = hog.compute(image)

    return hog_features
```

In [ ]:
```python
#LBP def
from skimage.feature import local_binary_pattern

def extract_lbp_features(image, num_points=32, radius=8):
    gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    grid_size = (8, 8)  # Divide image into a 8x8 grid for histograms
     # Compute LBP
    lbp = local_binary_pattern(gray_img, num_points, radius, method="uniform")
    h, w = lbp.shape

    # Divide the image into grids and compute histograms
    grid_h, grid_w = h // grid_size[0], w // grid_size[1]
    histograms = []
```

```
    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            grid = lbp[i * grid_h:(i + 1) * grid_h, j * grid_w:(j + 1) * grid_w]
            hist, _ = np.histogram(grid, bins=np.arange(0, num_points + 3), density
            histograms.append(hist)

    # Concatenate histograms
    return np.concatenate(histograms)
    """gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = local_binary_pattern(gray_img, num_points, radius, method='uniform')
     # Calculate the histogram of LBP
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, num_points + 3), range=


    return np.array(hist)"""
```

In [48]:
```
lbp_features_train = np.array([extract_lbp_features(X_train[image]) for image in ra
```

In [49]:
```
# Step 1: Extract LBP features for train and test
lbp_features_train = np.array([extract_lbp_features(image) for image in  X_train])
lbp_features_test  = np.array([extract_lbp_features(image) for image in X_test])
```

In [ ]:
```
# Step 2: Extract HOG features for train and test
hog_features_train = np.array([extract_hog_features(image) for image in X_train])
hog_features_test  = np.array([extract_hog_features(image) for image in X_test])
```

In [ ]:
```
# Step 3: Extract Color Histogram features for train and test
clhg_features_train = np.array([extract_color_histogram(image) for image in X_train
clhg_features_test  = np.array([extract_color_histogram(image) for image in X_test]
```

In [2]:
```
import numpy as np

class KNNClassifier:
    def __init__(self, k = 3):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def count_occurrences(self,input_array,distances):
        output_array = []
        for i in range(len(input_array)):
            count = sum(
                np.array_equal(input_array[i], other)
                for other in input_array
            )
            output_array.append(count)
        return output_array.index(max(output_array))
            #output_array.append((1/count)*distances[i])
        #return output_array.index(min(output_array))  # return index of minimum co



    def predict(self, image_test):
```

```
        distances = np.linalg.norm(self.X_train - image_test.reshape(1,-1), axis=1)
        k_nearest = np.argsort(distances)[:self.k]
        #print(k_nearest)
        k_nearest_labels = self.y_train[k_nearest]
        #print(k_nearest_labels)
        prediction = self.count_occurrences(k_nearest_labels,k_nearest)


        return np.array(k_nearest_labels[prediction])
```

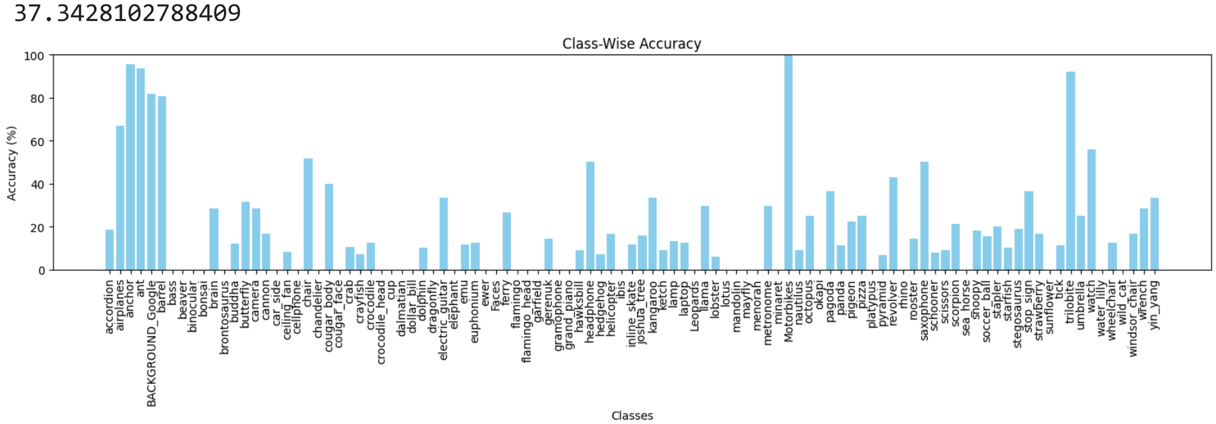
In [10]:
```
#compare with pre-built KNN from sklearn

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(clhg_features_train, y_train)
predictions = knn.predict(clhg_features_test)
accuracy(y_test, predictions)
```

Out[10]: 31.665299425758818

In [ ]:
```
model = KNNClassifier(9)
model.fit(lbp_features_train,y_train)

y_pred = np.array([model.predict( i.reshape(1, -1)) for i in  lbp_features_test])
```

In [51]:
```
print(accuracy(y_test, y_pred))
visualize_results(y_test, y_pred)
```

37.3428102788409



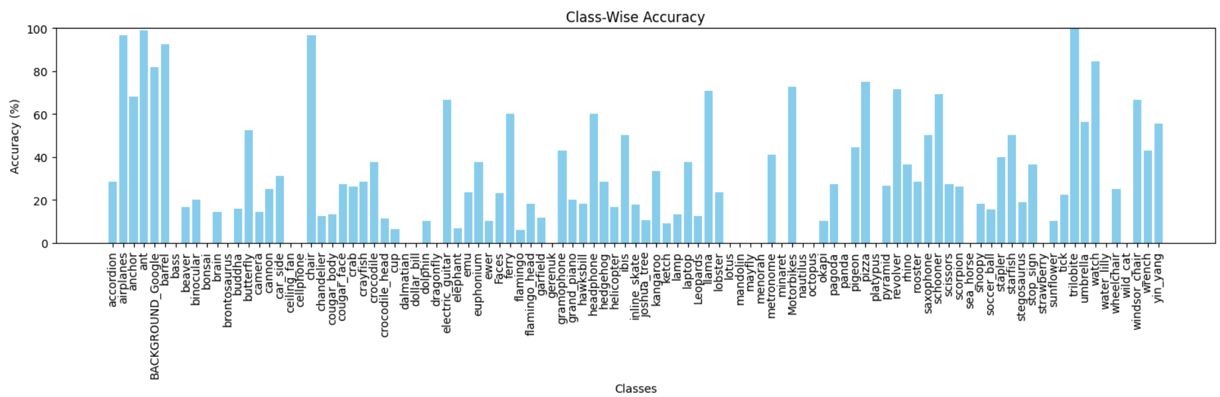
In [29]:
```
model2 = KNNClassifier(9)
model2.fit(hog_features_train,y_train)

y_pred2 = np.array([model2.predict( i.reshape(1, -1)) for i in  hog_features_test])
```

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
hog = cv2.HOGDescriptor()

# Extract HOG features
X = [hog.compute(cv2.resize(img.astype(np.uint8) if img.dtype != np.uint8 else img,
XX= [hog.compute(cv2.resize(img.astype(np.uint8) if img.dtype != np.uint8 else img,
```

```python
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y_train)

y_pred2 =knn.predict(XX)
```

In [30]:
```python
print(accuracy(y_test, y_pred2))
visualize_results(y_test, y_pred2)
```

49.91798797156916

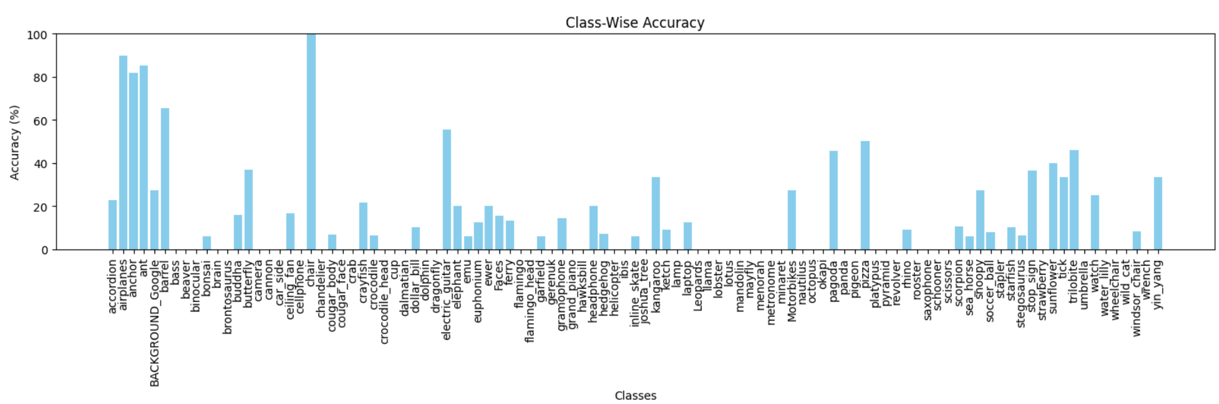


In [12]:
```python
model3 = KNNClassifier(9)
model3.fit(clhg_features_train,y_train)

y_pred3 = np.array([model3.predict( i.reshape(1, -1)) for i in clhg_features_test])
```

In [ ]:
```python
print (accuracy(y_test, y_pred3))
visualize_results(y_test, y_pred3)
```

Out[ ]: 33.351558228540185

In [ ]:



```
Accuracy for accordion: 22.83%
Accuracy for airplanes: 89.71%
Accuracy for anchor: 81.82%
Accuracy for ant: 85.23%
Accuracy for BACKGROUND_Google: 27.27%
Accuracy for barrel: 65.43%
Accuracy for bass: 0.00%
Accuracy for beaver: 0.00%
Accuracy for binocular: 0.00%
Accuracy for bonsai: 5.88%
Accuracy for brain: 0.00%
Accuracy for brontosaurus: 0.00%
Accuracy for buddha: 16.00%
Accuracy for butterfly: 36.84%
Accuracy for camera: 0.00%
Accuracy for cannon: 0.00%
Accuracy for car_side: 0.00%
Accuracy for ceiling_fan: 16.67%
Accuracy for cellphone: 0.00%
Accuracy for chair: 100.00%
Accuracy for chandelier: 0.00%
Accuracy for cougar_body: 6.67%
Accuracy for cougar_face: 0.00%
Accuracy for crab: 0.00%
Accuracy for crayfish: 21.43%
Accuracy for crocodile: 6.25%
Accuracy for crocodile_head: 0.00%
Accuracy for cup: 0.00%
Accuracy for dalmatian: 0.00%
Accuracy for dollar_bill: 10.00%
Accuracy for dolphin: 0.00%
Accuracy for dragonfly: 0.00%
Accuracy for electric_guitar: 55.56%
Accuracy for elephant: 20.00%
Accuracy for emu: 5.88%
Accuracy for euphonium: 12.50%
Accuracy for ewer: 20.00%
Accuracy for Faces: 15.38%
Accuracy for ferry: 13.33%
Accuracy for flamingo: 0.00%
Accuracy for flamingo_head: 0.00%
Accuracy for garfield: 5.88%
Accuracy for gerenuk: 0.00%
Accuracy for gramophone: 14.29%
Accuracy for grand_piano: 0.00%
Accuracy for hawksbill: 0.00%
Accuracy for headphone: 20.00%
Accuracy for hedgehog: 7.14%
Accuracy for helicopter: 0.00%
Accuracy for ibis: 0.00%
Accuracy for inline_skate: 5.88%
Accuracy for joshua_tree: 0.00%
Accuracy for kangaroo: 33.33%
Accuracy for ketch: 9.09%
Accuracy for lamp: 0.00%
Accuracy for laptop: 12.50%
```

```
Accuracy for Leopards: 0.00%
Accuracy for llama: 0.00%
Accuracy for lobster: 0.00%
Accuracy for lotus: 0.00%
Accuracy for mandolin: 0.00%
Accuracy for mayfly: 0.00%
Accuracy for menorah: 0.00%
Accuracy for metronome: 0.00%
Accuracy for minaret: 0.00%
Accuracy for Motorbikes: 27.27%
Accuracy for nautilus: 0.00%
Accuracy for octopus: 0.00%
Accuracy for okapi: 0.00%
Accuracy for pagoda: 45.45%
Accuracy for panda: 0.00%
Accuracy for pigeon: 0.00%
Accuracy for pizza: 50.00%
Accuracy for platypus: 0.00%
Accuracy for pyramid: 0.00%
Accuracy for revolver: 0.00%
Accuracy for rhino: 9.09%
Accuracy for rooster: 0.00%
Accuracy for saxophone: 0.00%
Accuracy for schooner: 0.00%
Accuracy for scissors: 0.00%
Accuracy for scorpion: 10.53%
Accuracy for sea_horse: 5.88%
Accuracy for snoopy: 27.27%
Accuracy for soccer_ball: 7.69%
Accuracy for stapler: 0.00%
Accuracy for starfish: 10.00%
Accuracy for stegosaurus: 6.25%
Accuracy for stop_sign: 36.36%
Accuracy for strawberry: 0.00%
Accuracy for sunflower: 40.00%
Accuracy for tick: 33.33%
Accuracy for trilobite: 46.15%
Accuracy for umbrella: 0.00%
Accuracy for watch: 25.00%
Accuracy for water_lilly: 0.00%
Accuracy for wheelchair: 0.00%
Accuracy for wild_cat: 0.00%
Accuracy for windsor_chair: 8.33%
Accuracy for wrench: 0.00%
Accuracy for yin_yang: 33.33%
```