candy-dataset

September 11, 2024

0.1 Candy Classification

- Authors: Martin Avila & Hadson Barbosa
- Github Repository: Candy Classification
- Dataset: Candy Dataset
  This project presents a complete pipeline for classifying candies using a custom dataset. From dataset creation to model development and evaluation, the entire classification process was implemented from scratch.

Folders:

- database: Dataset of images.
- docs: Auxiliary images.
- utils: .py files containing classes or plots.
- scripts: Auxiliary algorithms.
- models: Classification models

[4]:
```
import os
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

0.1.1 1. Dataset Creation

Adjusting Images

- After deciding on the subject, we captured 10 black-and-white pictures for each class using a Samsung S21 FE.
- Additionally, we resized the images to a resolution of 256x256 pixels to standardize the dataset for further processing.

[5]:
```
from scripts.image_plots import *
from scripts.resize_images import resize_images

base_dir = os.getcwd()
dataset_dir = os.path.join(base_dir, 'dataset')
input_dir = os.path.join(dataset_dir,'classes')
output_dir = os.path.join(base_dir, 'resized')
```

```
resize_images(input_dir, output_dir ,(256, 256))
```

[6]:
```
plot_mosaic_from_dir(os.path.join(output_dir, 'all'))
```

- Next Step: Rename all images to a specific format -> `<CLASSID>-<IMG_SEQUENCE>-V1|V2-B|W.png`

[7]:
```
root_dir = os.path.join(dataset_dir, 'classes', 'plane_class_06')
```

```
image_files = [f for f in os.listdir(root_dir) if os.path.isfile(os.path.
 ↪join(root_dir, f))]

print(image_files[3:6])
```

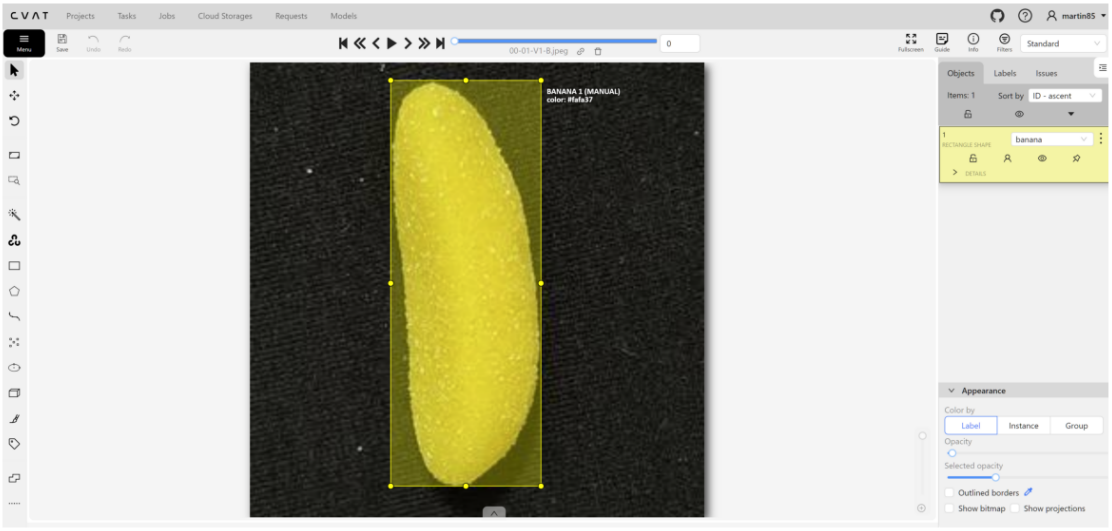
```
['06-02-V2-W.jpg', '06-04-V2B.jpg', '06-02-V1-B.jpg']
```

0.1.2 2. Data Annotation

- Using **CVAT Web**, we performed bounding box segmentation for each class.

[8]:
```
from PIL import Image

img_path = os.path.join(base_dir, 'docs', 'Banana_bounding_box_cvat_example.
 ↪png')
img = Image.open(img_path)

plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.axis('off')
```

[8]:
```
(-0.5, 3825.5, 1813.5, -0.5)
```



0.1.3 Preprocessing

We divided this stage in three steps:

```
* Correction of inconsistencies
* Data augmentation (expansion of the dataset)
* Data normalization
```

[9]:
```
from utils.dataset import CandyDataset
from utils.preprocessing import correct_background
from utils.normalization_plots import generate_image_statitics
```

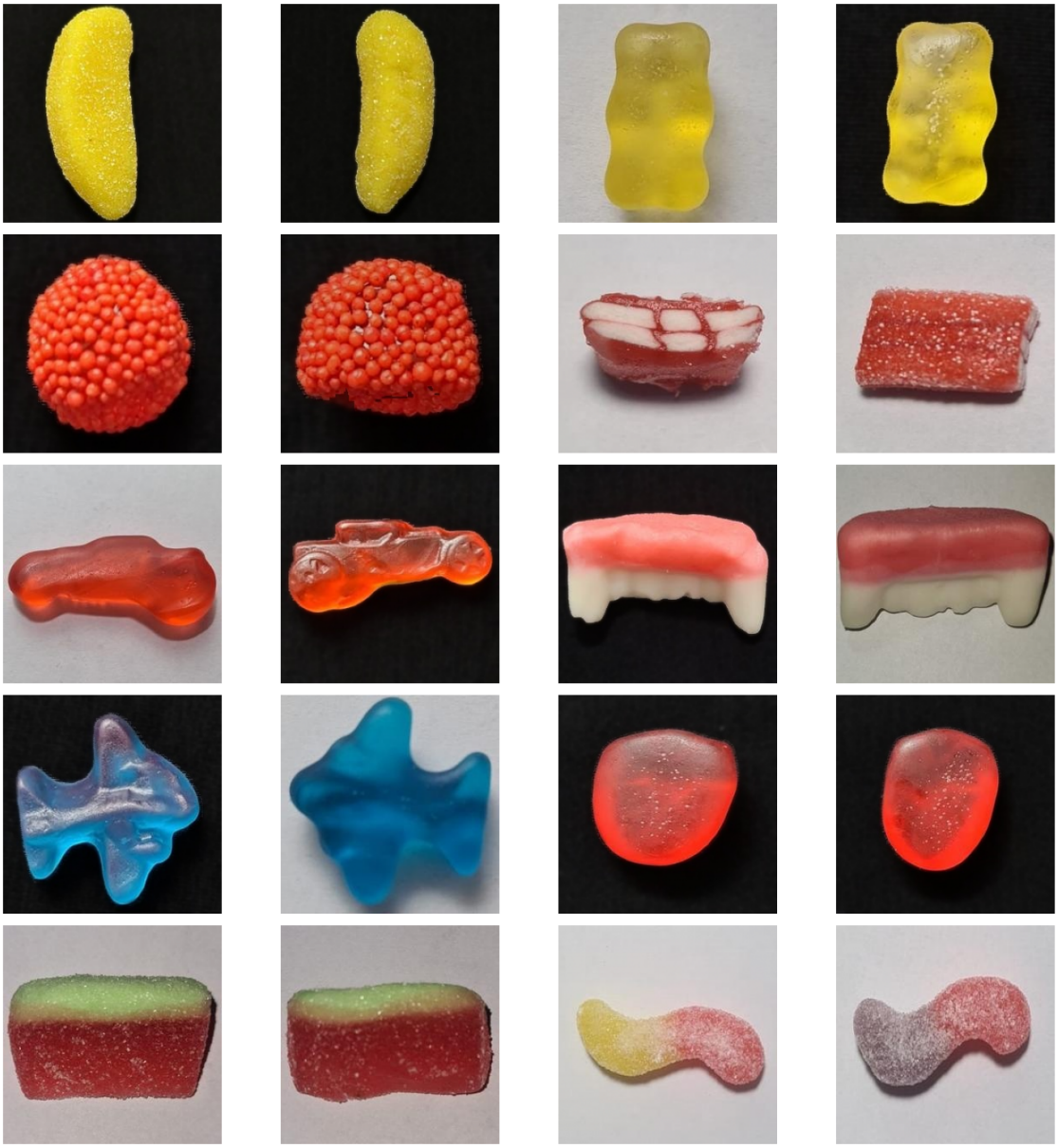
[10]:
```
dataset_dir = os.path.join(base_dir, 'dataset')
```

Background Correction Some images with black background from the dataset have inconsistencies in the background, having variations in texture and brightness.
This script corrects these inconsistencies, darkening and blurring the background to make it more uniform while keeping the object in the foreground.

[11]:
```
data_path = os.path.join(dataset_dir, '0 - original')
preprocessed_dir = os.path.join(dataset_dir, '1 - preprocessed')

correct_background(data_path, preprocessed_dir)
```

[12]:
```
# Load dataset after background corrections
data_dir = os.path.join(dataset_dir, '1 - preprocessed')
dataset = CandyDataset(data_dir)
```

[13]:
```
plot_mosaic_from_dir(data_dir)
```



0.1.4 3. Data augmentation

- albumentations: The albumentations lib is designed to increase dataset variety by applying a series of random transformations to an image
- The functions used are:

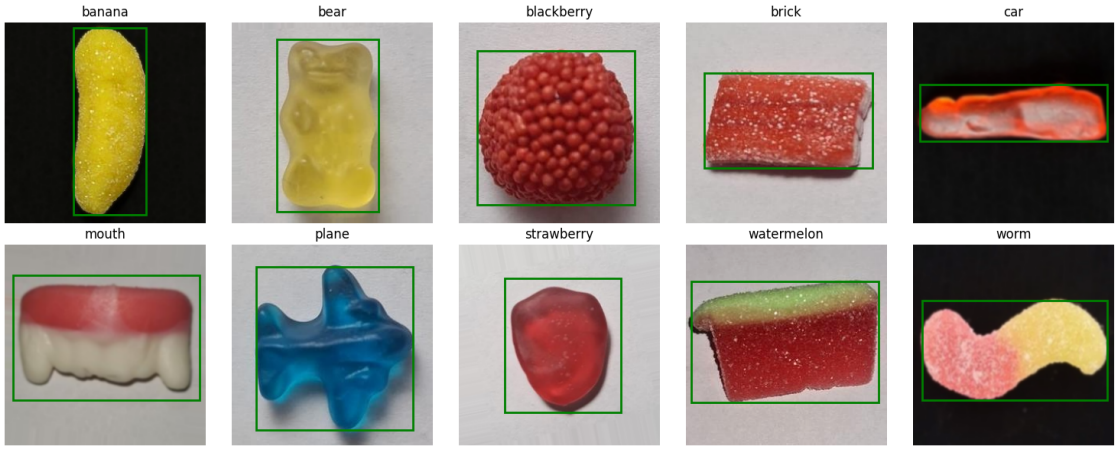
    1. `SafeRotate: Rotate an image`
    2. `Affine: Transformation with shear`
    3. `RandomRotate90: Randomly rote 90, 180, or 270`
    4. `HorizontalFlip`
    5. `VerticalFlip`

- Extra Transfomations:

  1. RandomBrightnessContrast
  2. GaussianBlur
  3. HueSaturationValue: just using Saturation shifting

[14]:
```
# augmentation
augmented_dir = os.path.join(dataset_dir, '2 - augmented')
dataset.augment_dataset()
dataset.save_annotations(augmented_dir)
dataset.export_data(augmented_dir)
```

[15]:
```
plot_mosaic_from_dataset(dataset, plot_bbox=True)
```



0.1.5 4. Data Normalization

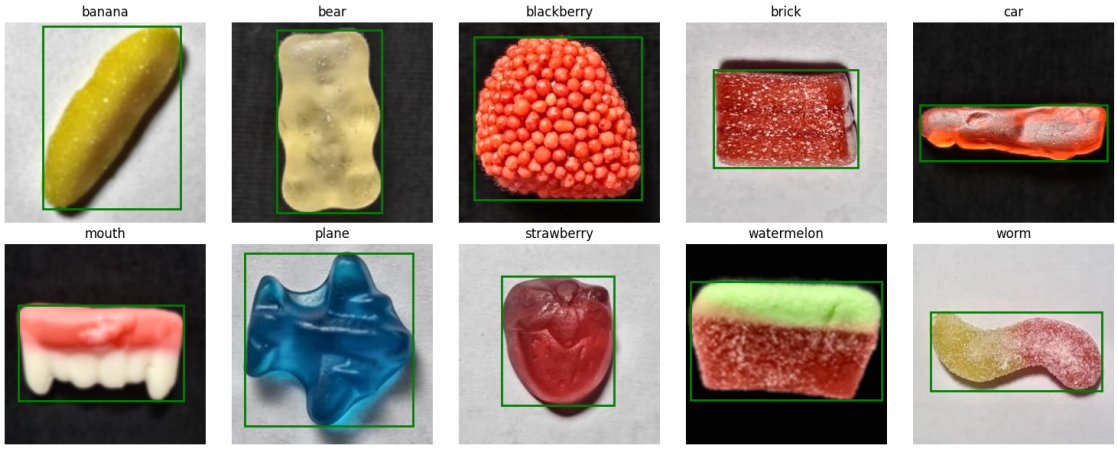
Data normalization in (DIP) involves equalizing histograms to enhance image quality. The process aims to adjust the contrast and brightness of images for improved consistency and clarity.

Steps:

```
- Histogram Equalization: Adjust the image's histogram to enhance contrast and distribute pixe

- Analyzing the Results: Evaluate the outcomes of histogram equalization to ensure that the ad
```

[16]:
```
normalized_dir = os.path.join(dataset_dir, '3 - normalized')
dataset.normalize_dataset()
dataset.save_annotations(normalized_dir)
dataset.export_data(normalized_dir)
```

[17]:
```
plot_mosaic_from_dataset(dataset, plot_bbox=True)
```



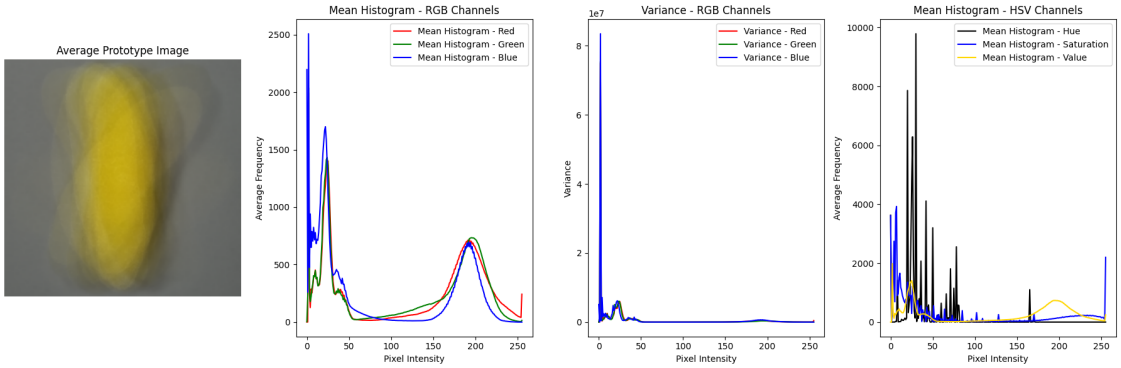
After extensive research, we found that the best way to understand our data was by using the RGB color model. Therefore, we created three lists, one for each channel. We compared these channels in a single frequency graph over the pixel values.

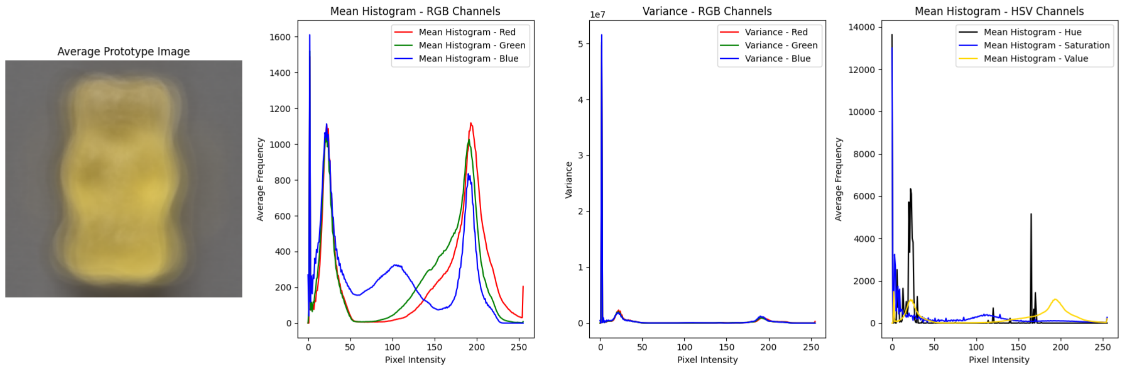
- Mean Histogram - RGB Channels: the average pixel intensity distribution for the Red, Green, and Blue channels.
- The histogram variance: indicates how much the intensity values vary across the images.
- Mean Histogram - HSV Channels: shows intensity distribution for the Hue, Saturation, and Value channels in the HSV color space.

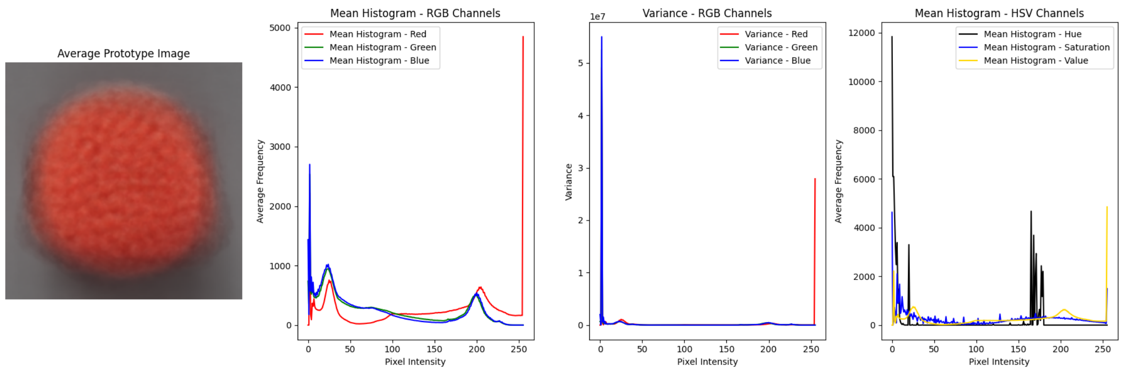
[18]:
```
classes = dataset.categories_indexes.keys()
for class_name in classes:
    class_indexes = dataset.categories_indexes[class_name]
    images_data = dataset.get_images_data(class_indexes)

    images = [d['image'] for d in images_data]

    generate_image_statitics(images, class_name)
```

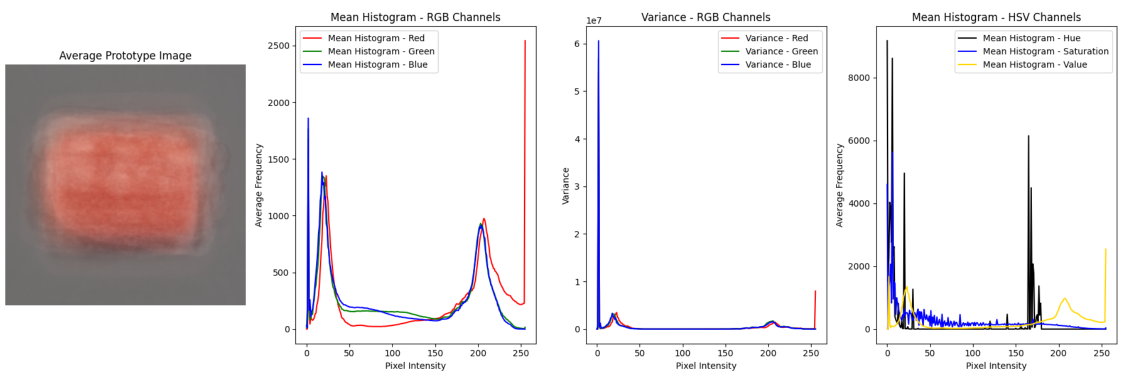
```
Class 'banana' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'bear' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'blackberry' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'brick' statistics:
        Average prototype calculated successfully!
```

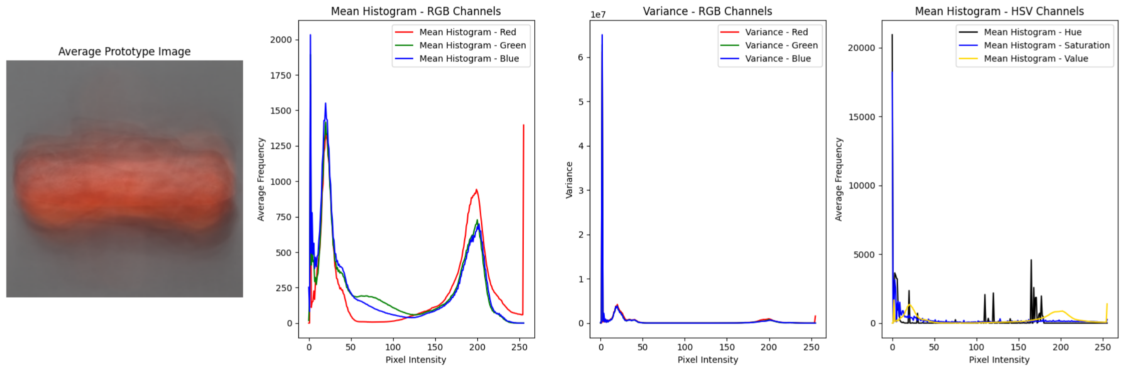
```
        Histogram variance calculated!
        Mean histogram calculated!
Class 'car' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'mouth' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'plane' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'strawberry' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'watermelon' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
Class 'worm' statistics:
        Average prototype calculated successfully!
        Histogram variance calculated!
        Mean histogram calculated!
```

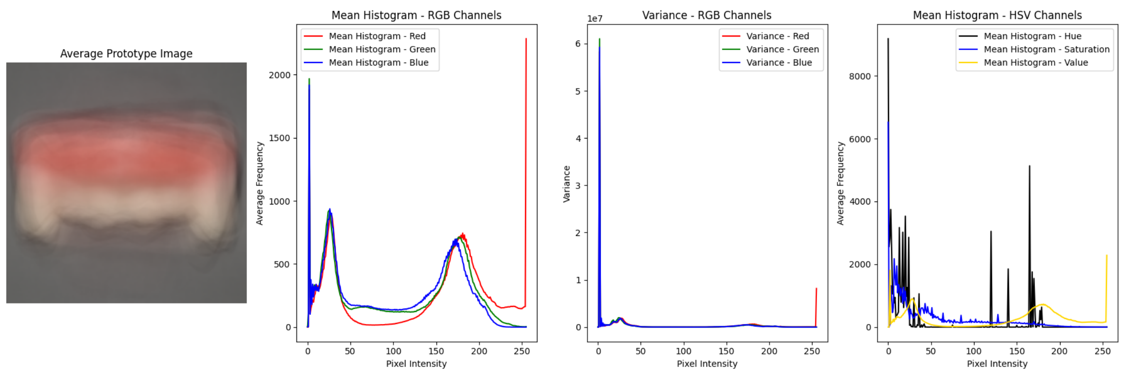


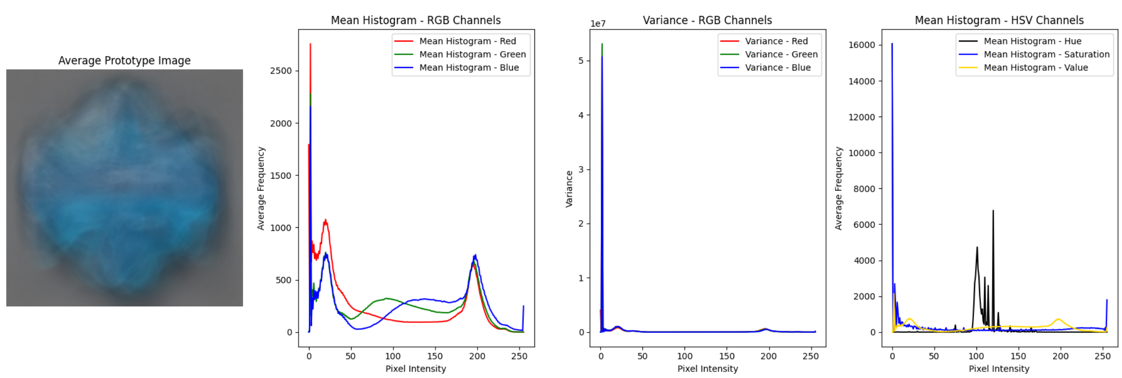

Average Prototype Image
Mean Histogram - RGB Channels
Mean Histogram - Red
Mean Histogram - Green
Mean Histogram - Blue
Average Frequency
Pixel Intensity
Variance - RGB Channels
1e7
Variance - Red
Variance - Green
Variance - Blue
Variance
Pixel Intensity
Mean Histogram - HSV Channels
Mean Histogram - Hue
Mean Histogram - Saturation
Mean Histogram - Value
Average Frequency
Pixel Intensity

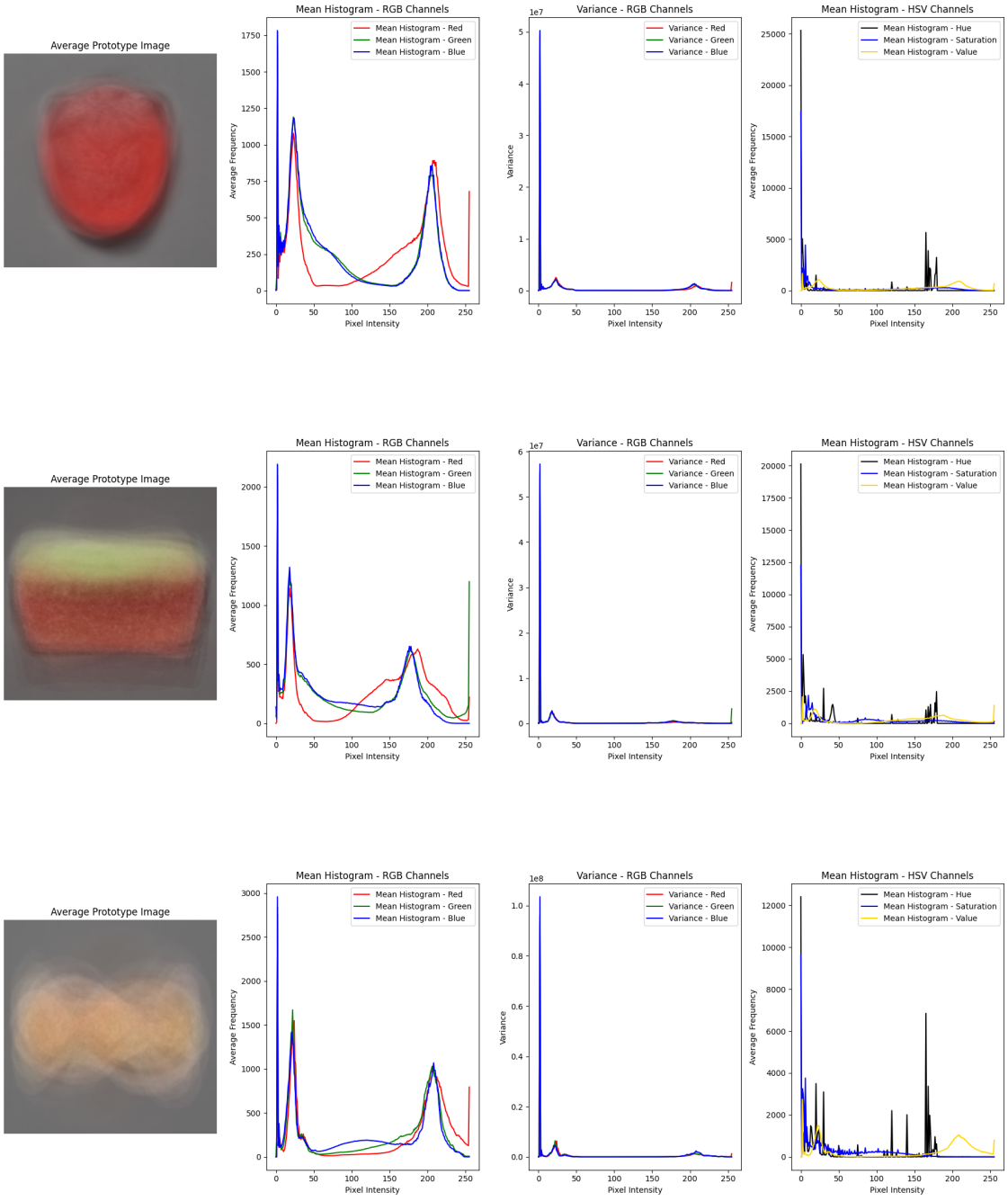

Average Prototype Image
Mean Histogram - RGB Channels
Mean Histogram - Red
Mean Histogram - Green
Mean Histogram - Blue
Average Frequency
Pixel Intensity
Variance - RGB Channels
1e7
Variance - Red
Variance - Green
Variance - Blue
Variance
Pixel Intensity
Mean Histogram - HSV Channels
Mean Histogram - Hue
Mean Histogram - Saturation
Mean Histogram - Value
Average Frequency
Pixel Intensity


Average Prototype Image
Mean Histogram - RGB Channels
Mean Histogram - Red
Mean Histogram - Green
Mean Histogram - Blue
Average Frequency
Pixel Intensity
Variance - RGB Channels
1e7
Variance - Red
Variance - Green
Variance - Blue
Variance
Pixel Intensity
Mean Histogram - HSV Channels
Mean Histogram - Hue
Mean Histogram - Saturation
Mean Histogram - Value
Average Frequency
Pixel Intensity


Average Prototype Image
Mean Histogram - RGB Channels
Mean Histogram - Red
Mean Histogram - Green
Mean Histogram - Blue
Average Frequency
Pixel Intensity
Variance - RGB Channels
1e7
Variance - Red
Variance - Green
Variance - Blue
Variance
Pixel Intensity
Mean Histogram - HSV Channels
Mean Histogram - Hue
Mean Histogram - Saturation
Mean Histogram - Value
Average Frequency
Pixel Intensity


Average Prototype Image
Mean Histogram - RGB Channels
Mean Histogram - Red
Mean Histogram - Green
Mean Histogram - Blue
Average Frequency
Pixel Intensity
Variance - RGB Channels
1e7
Variance - Red
Variance - Green
Variance - Blue
Variance
Pixel Intensity
Mean Histogram - HSV Channels
Mean Histogram - Hue
Mean Histogram - Saturation
Mean Histogram - Value
Average Frequency
Pixel Intensity


Average Prototype Image
Mean Histogram - RGB Channels
Mean Histogram - Red
Mean Histogram - Green
Mean Histogram - Blue
Average Frequency
Pixel Intensity
Variance - RGB Channels
1e7
Variance - Red
Variance - Green
Variance - Blue
Variance
Pixel Intensity
Mean Histogram - HSV Channels
Mean Histogram - Hue
Mean Histogram - Saturation
Mean Histogram - Value
Average Frequency
Pixel Intensity



In this example: - The prototipe: The image is a averaged version of several images of a yellow object (indicating a banana, due to the shape and color), layered on top of each other with some transparency. - Mean Histogram: Intensity between 0-50 , indicating a darker image background.

Yellow color in the prototype image, since yellow is a combination of high red and green intensities

- Variance - RGB Channels: The variance is higher at lower intensities (corresponding to the dark background)

The yellow object's color is relatively consistent across the images.

- Mean Histogram - HSV Channels :
- The Hue (black line) has a spike around 100, which likely represents the dominant color of the object (yellow).
- The Saturation (blue line) and Value (yellow line) show distributions concentrated at lower intensities, which is typical for darker images.
- The yellow object's saturation is moderate, and the value (brightness) has a broader spread, indicating that there are both bright and somewhat muted yellow areas in the images.

0.1.6 5. Segmentation (Ground Truth)

In this section, we use the ground truth to evaluate the performance of our model in segmentation.

Ground truth is typically obtained by manually or automatically labeling the image.

We created a binary representation of our image, where 0 represents the background and 1 represents the object.

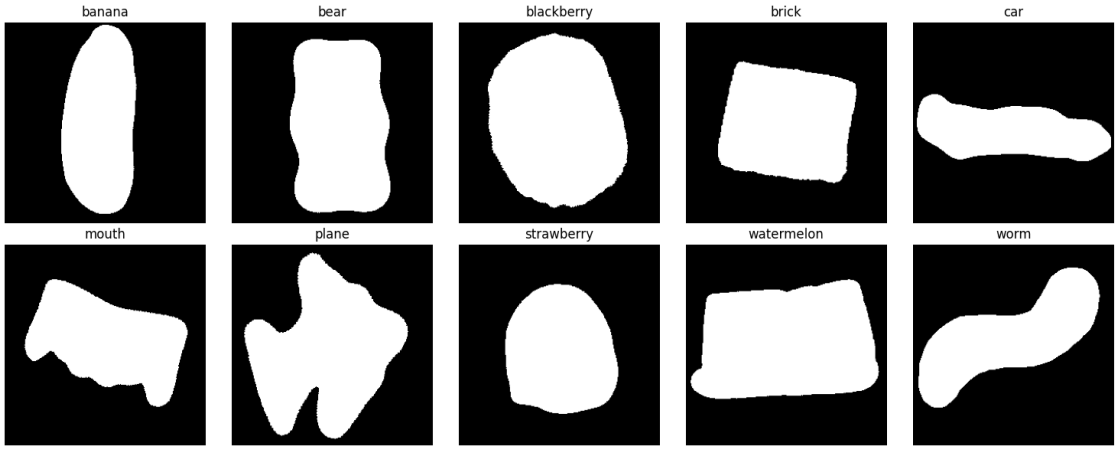
Manual Steps:

`- Part of the process was manual, using CVAT for approximately 25% of the masks.`

Semi - Automatic Steps:

`- 75%: Using CVAT and SAM to identify key points and define the object's shape.`

[19]:
```
groud_truth_dir = os.path.join(dataset_dir, '4 - ground truth')
dataset.export_data(groud_truth_dir, data_type='masks')

plot_mosaic_from_dataset(dataset, data_type='masks')
```



0.1.7 6. Model Training

We tested a CNN classification model. The model learns to map the input images to the corresponding segmentation masks.

The model was tested in a separate notebook because it is the best way to work with TensorFlow.

Acess: cnn.ipynb

cnn-model

September 11, 2024

0.0.1 Model Training

Classification Models: CNN -> (input images and features like bounding boxes)

[ ]:
```
import os
import tensorflow as tf
import numpy as np

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Input

from utils.dataset import CandyDataset

import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2
```

[2]:
```
base_dir = os.getcwd()
dataset_dir = os.path.join(base_dir, 'dataset', '3 - normalized')

dataset = CandyDataset(dataset_dir)
dataset.split_dataset()
```

- split content to form dataset: Configurations (data split):
  - Train: 80%
  - Test: 10%
  - Validation: 10%

[7]:
```
train_set_data = dataset.get_images_data(dataset.data_split['train'])
test_set_data  = dataset.get_images_data(dataset.data_split['test'])
validation_set_data  = dataset.get_images_data(dataset.data_split['val'])

train_images = np.array([d['image'] for d in train_set_data])  / 255.0
train_labels = np.array([d['category_id'] - 1 for d in train_set_data])
train_bbox = np.array([d['bbox'] for d in train_set_data])
```

```
test_images = np.array([d['image'] for d in test_set_data])  / 255.0
test_labels = np.array([d['category_id'] - 1 for d in test_set_data])
test_bbox = np.array([d['bbox'] for d in test_set_data])

validation_images = np.array([d['image'] for d in validation_set_data])  / 255.0
validation_labels = np.array([d['category_id'] - 1 for d in␣
 ↪validation_set_data])
validation_bbox = np.array([d['bbox'] for d in validation_set_data])

num_classes = len(np.unique(train_labels)) # calculate number of clases

train_labels = to_categorical(train_labels, num_classes)
test_labels = to_categorical(test_labels, num_classes)
validation_labels = to_categorical(validation_labels, num_classes)

print(train_images.shape)
print(train_labels.shape)
print(train_bbox.shape)

print(validation_images.shape)
print(validation_labels.shape)
print(validation_bbox.shape)
```

```
(1440, 256, 256, 3)
(1440, 10)
(1440, 4)
(180, 256, 256, 3)
(180, 10)
(180, 4)
```

- As seen previously, the colors and background are very intense for each object, so the model had to learn the common color, saturation, and lightness intensity pixels to classify correctly.
- We encountered some issues with the input shape. Combining images with bounding boxes required us to adjust the dimensions of the entire dataset.
- No mask was used because, despite attempts, the best option would have been to try with an R-CNN.
- We encountered some issues Compiling the code without using a GPU.
- After various tests, the architecture we used was:

1. Kernel = 5 → Window kernel (convolution).
2. Pool = 2 → Detects the highest number of neighboring pixels (most representative pixel).
3. Padding = 'same' → 'same' results in padding with zeros.
4. Activation = 'relu' → Necessary for this task to avoid linearity issues.
5. Input: 3 channels (RGB) and two outputs for the model to identify bounding boxes and the class.

[ ]:
```
input_layer = Input(shape=(256, 256, 3))

kernel = (5, 5)
pool = (2, 2)

x = Conv2D(16, kernel, padding='same', activation='relu')(input_layer)
x = MaxPooling2D(pool_size=pool)(x)
x = Conv2D(32, kernel, padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=pool)(x)
x = Conv2D(64, kernel, padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=pool)(x)
x = Conv2D(128, kernel, padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=pool)(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)

bbox_output = Dense(4, name='bbox_output')(x)
class_output = Dense(num_classes, activation='softmax', name='class_output')(x)

model = Model(inputs=input_layer, outputs=[bbox_output, class_output])

model.summary()
```

[8]:
```
model.compile(optimizer='adam',
              loss={'bbox_output': 'mean_squared_error', 'class_output':␣
 ↪'categorical_crossentropy'},
              loss_weights={'bbox_output': 1.0, 'class_output': 1.0},
              metrics={'bbox_output': 'mean_squared_error', 'class_output':␣
 ↪'accuracy'})
```

[9]:
```
history = model.fit(
    x=train_images,
    y={'bbox_output': train_bbox, 'class_output': train_labels},
    validation_data=(validation_images, {'bbox_output': validation_bbox,␣
 ↪'class_output': validation_labels}),
    epochs = 50,
)
```

```
2024-09-10 00:32:48.025113: W tensorflow/tsl/framework/cpu_allocator_impl.cc:82]
Allocation of 1132462080 exceeds 10% of free system memory.

Epoch 1/50
45/45 [==============================] - 73s 2s/step - loss: 5442.4976 -
bbox_output_loss: 5429.6372 - class_output_loss: 12.8600 -
bbox_output_mean_squared_error: 5429.6372 - class_output_accuracy: 0.0944 -
val_loss: 1695.3274 - val_bbox_output_loss: 1686.5608 - val_class_output_loss:
8.7664 - val_bbox_output_mean_squared_error: 1686.5608 -
val_class_output_accuracy: 0.1167
```

```
Epoch 2/50
45/45 [==============================] - 71s 2s/step - loss: 1446.2682 -
bbox_output_loss: 1440.5725 - class_output_loss: 5.6954 -
bbox_output_mean_squared_error: 1440.5725 - class_output_accuracy: 0.0875 -
val_loss: 1214.7560 - val_bbox_output_loss: 1209.6465 - val_class_output_loss:
5.1095 - val_bbox_output_mean_squared_error: 1209.6465 -
val_class_output_accuracy: 0.0722
Epoch 3/50
45/45 [==============================] - 66s 1s/step - loss: 1033.6510 -
bbox_output_loss: 1027.9498 - class_output_loss: 5.7015 -
bbox_output_mean_squared_error: 1027.9498 - class_output_accuracy: 0.0931 -
val_loss: 715.6495 - val_bbox_output_loss: 709.5884 - val_class_output_loss:
6.0612 - val_bbox_output_mean_squared_error: 709.5884 -
val_class_output_accuracy: 0.1500
Epoch 4/50
45/45 [==============================] - 65s 1s/step - loss: 630.2604 -
bbox_output_loss: 626.2701 - class_output_loss: 3.9903 -
bbox_output_mean_squared_error: 626.2701 - class_output_accuracy: 0.1986 -
val_loss: 585.8262 - val_bbox_output_loss: 583.1591 - val_class_output_loss:
2.6671 - val_bbox_output_mean_squared_error: 583.1591 -
val_class_output_accuracy: 0.2667
Epoch 5/50
45/45 [==============================] - 64s 1s/step - loss: 490.6097 -
bbox_output_loss: 488.3536 - class_output_loss: 2.2559 -
bbox_output_mean_squared_error: 488.3536 - class_output_accuracy: 0.3688 -
val_loss: 313.8105 - val_bbox_output_loss: 311.3861 - val_class_output_loss:
2.4245 - val_bbox_output_mean_squared_error: 311.3861 -
val_class_output_accuracy: 0.3611
Epoch 6/50
45/45 [==============================] - 66s 1s/step - loss: 335.0635 -
bbox_output_loss: 332.8222 - class_output_loss: 2.2414 -
bbox_output_mean_squared_error: 332.8222 - class_output_accuracy: 0.4160 -
val_loss: 255.2757 - val_bbox_output_loss: 253.2127 - val_class_output_loss:
2.0631 - val_bbox_output_mean_squared_error: 253.2127 -
val_class_output_accuracy: 0.4611
Epoch 7/50
45/45 [==============================] - 66s 1s/step - loss: 310.4406 -
bbox_output_loss: 308.4359 - class_output_loss: 2.0048 -
bbox_output_mean_squared_error: 308.4359 - class_output_accuracy: 0.5229 -
val_loss: 286.6047 - val_bbox_output_loss: 284.4661 - val_class_output_loss:
2.1387 - val_bbox_output_mean_squared_error: 284.4661 -
val_class_output_accuracy: 0.5000
Epoch 8/50
45/45 [==============================] - 66s 1s/step - loss: 277.4892 -
bbox_output_loss: 275.7736 - class_output_loss: 1.7157 -
bbox_output_mean_squared_error: 275.7736 - class_output_accuracy: 0.5451 -
val_loss: 191.8322 - val_bbox_output_loss: 190.3845 - val_class_output_loss:
1.4477 - val_bbox_output_mean_squared_error: 190.3845 -
```

```
val_class_output_accuracy: 0.5333
Epoch 9/50
45/45 [==============================] - 66s 1s/step - loss: 243.1385 -
bbox_output_loss: 241.8300 - class_output_loss: 1.3085 -
bbox_output_mean_squared_error: 241.8300 - class_output_accuracy: 0.6229 -
val_loss: 184.2340 - val_bbox_output_loss: 182.5259 - val_class_output_loss:
1.7081 - val_bbox_output_mean_squared_error: 182.5259 -
val_class_output_accuracy: 0.5167
Epoch 10/50
45/45 [==============================] - 65s 1s/step - loss: 245.5412 -
bbox_output_loss: 244.3585 - class_output_loss: 1.1827 -
bbox_output_mean_squared_error: 244.3585 - class_output_accuracy: 0.6854 -
val_loss: 204.9616 - val_bbox_output_loss: 203.9770 - val_class_output_loss:
0.9845 - val_bbox_output_mean_squared_error: 203.9770 -
val_class_output_accuracy: 0.7000
Epoch 11/50
45/45 [==============================] - 64s 1s/step - loss: 225.7250 -
bbox_output_loss: 224.7641 - class_output_loss: 0.9610 -
bbox_output_mean_squared_error: 224.7641 - class_output_accuracy: 0.7729 -
val_loss: 240.6555 - val_bbox_output_loss: 239.8517 - val_class_output_loss:
0.8038 - val_bbox_output_mean_squared_error: 239.8517 -
val_class_output_accuracy: 0.8111
Epoch 12/50
45/45 [==============================] - 65s 1s/step - loss: 274.8370 -
bbox_output_loss: 273.7952 - class_output_loss: 1.0419 -
bbox_output_mean_squared_error: 273.7952 - class_output_accuracy: 0.7299 -
val_loss: 177.7419 - val_bbox_output_loss: 176.6919 - val_class_output_loss:
1.0500 - val_bbox_output_mean_squared_error: 176.6919 -
val_class_output_accuracy: 0.6722
Epoch 13/50
45/45 [==============================] - 64s 1s/step - loss: 239.2961 -
bbox_output_loss: 238.0444 - class_output_loss: 1.2517 -
bbox_output_mean_squared_error: 238.0444 - class_output_accuracy: 0.6757 -
val_loss: 194.4513 - val_bbox_output_loss: 193.6585 - val_class_output_loss:
0.7928 - val_bbox_output_mean_squared_error: 193.6585 -
val_class_output_accuracy: 0.7278
Epoch 14/50
45/45 [==============================] - 64s 1s/step - loss: 218.5063 -
bbox_output_loss: 217.5775 - class_output_loss: 0.9289 -
bbox_output_mean_squared_error: 217.5775 - class_output_accuracy: 0.7701 -
val_loss: 159.7100 - val_bbox_output_loss: 158.8107 - val_class_output_loss:
0.8993 - val_bbox_output_mean_squared_error: 158.8107 -
val_class_output_accuracy: 0.7556
Epoch 15/50
45/45 [==============================] - 65s 1s/step - loss: 199.2324 -
bbox_output_loss: 198.3044 - class_output_loss: 0.9280 -
bbox_output_mean_squared_error: 198.3044 - class_output_accuracy: 0.7910 -
val_loss: 164.5033 - val_bbox_output_loss: 163.7253 - val_class_output_loss:
```

```
0.7780 - val_bbox_output_mean_squared_error: 163.7253 -
val_class_output_accuracy: 0.7667
Epoch 16/50
45/45 [==============================] - 65s 1s/step - loss: 189.5713 -
bbox_output_loss: 188.8374 - class_output_loss: 0.7338 -
bbox_output_mean_squared_error: 188.8374 - class_output_accuracy: 0.8111 -
val_loss: 139.3357 - val_bbox_output_loss: 138.6525 - val_class_output_loss:
0.6832 - val_bbox_output_mean_squared_error: 138.6525 -
val_class_output_accuracy: 0.8222
Epoch 17/50
45/45 [==============================] - 65s 1s/step - loss: 180.4084 -
bbox_output_loss: 179.7767 - class_output_loss: 0.6317 -
bbox_output_mean_squared_error: 179.7767 - class_output_accuracy: 0.8451 -
val_loss: 145.0077 - val_bbox_output_loss: 144.4799 - val_class_output_loss:
0.5278 - val_bbox_output_mean_squared_error: 144.4799 -
val_class_output_accuracy: 0.8833
Epoch 18/50
45/45 [==============================] - 64s 1s/step - loss: 190.1603 -
bbox_output_loss: 189.5547 - class_output_loss: 0.6056 -
bbox_output_mean_squared_error: 189.5547 - class_output_accuracy: 0.8514 -
val_loss: 239.5656 - val_bbox_output_loss: 238.9210 - val_class_output_loss:
0.6446 - val_bbox_output_mean_squared_error: 238.9210 -
val_class_output_accuracy: 0.7667
Epoch 19/50
45/45 [==============================] - 65s 1s/step - loss: 208.8366 -
bbox_output_loss: 208.2292 - class_output_loss: 0.6074 -
bbox_output_mean_squared_error: 208.2292 - class_output_accuracy: 0.8431 -
val_loss: 133.9319 - val_bbox_output_loss: 133.4348 - val_class_output_loss:
0.4971 - val_bbox_output_mean_squared_error: 133.4348 -
val_class_output_accuracy: 0.8778
Epoch 20/50
45/45 [==============================] - 66s 1s/step - loss: 167.4811 -
bbox_output_loss: 166.9057 - class_output_loss: 0.5754 -
bbox_output_mean_squared_error: 166.9057 - class_output_accuracy: 0.8569 -
val_loss: 132.0285 - val_bbox_output_loss: 131.6343 - val_class_output_loss:
0.3942 - val_bbox_output_mean_squared_error: 131.6343 -
val_class_output_accuracy: 0.9000
Epoch 21/50
45/45 [==============================] - 65s 1s/step - loss: 156.9557 -
bbox_output_loss: 156.4443 - class_output_loss: 0.5114 -
bbox_output_mean_squared_error: 156.4443 - class_output_accuracy: 0.8618 -
val_loss: 128.8837 - val_bbox_output_loss: 128.4862 - val_class_output_loss:
0.3974 - val_bbox_output_mean_squared_error: 128.4862 -
val_class_output_accuracy: 0.8778
Epoch 22/50
45/45 [==============================] - 65s 1s/step - loss: 165.2457 -
bbox_output_loss: 164.7130 - class_output_loss: 0.5327 -
bbox_output_mean_squared_error: 164.7130 - class_output_accuracy: 0.8681 -
```

```
val_loss: 120.4498 - val_bbox_output_loss: 120.0640 - val_class_output_loss:
0.3859 - val_bbox_output_mean_squared_error: 120.0640 -
val_class_output_accuracy: 0.9056
Epoch 23/50
45/45 [==============================] - 65s 1s/step - loss: 157.7811 -
bbox_output_loss: 157.3377 - class_output_loss: 0.4434 -
bbox_output_mean_squared_error: 157.3377 - class_output_accuracy: 0.8819 -
val_loss: 144.7018 - val_bbox_output_loss: 144.3255 - val_class_output_loss:
0.3762 - val_bbox_output_mean_squared_error: 144.3255 -
val_class_output_accuracy: 0.9000
Epoch 24/50
45/45 [==============================] - 64s 1s/step - loss: 155.1400 -
bbox_output_loss: 154.6711 - class_output_loss: 0.4688 -
bbox_output_mean_squared_error: 154.6711 - class_output_accuracy: 0.8743 -
val_loss: 118.3103 - val_bbox_output_loss: 117.9351 - val_class_output_loss:
0.3752 - val_bbox_output_mean_squared_error: 117.9351 -
val_class_output_accuracy: 0.9000
Epoch 25/50
45/45 [==============================] - 65s 1s/step - loss: 140.7332 -
bbox_output_loss: 140.3069 - class_output_loss: 0.4263 -
bbox_output_mean_squared_error: 140.3069 - class_output_accuracy: 0.8889 -
val_loss: 142.1669 - val_bbox_output_loss: 141.7492 - val_class_output_loss:
0.4178 - val_bbox_output_mean_squared_error: 141.7492 -
val_class_output_accuracy: 0.8611
Epoch 26/50
45/45 [==============================] - 64s 1s/step - loss: 152.4803 -
bbox_output_loss: 152.0209 - class_output_loss: 0.4594 -
bbox_output_mean_squared_error: 152.0209 - class_output_accuracy: 0.8687 -
val_loss: 113.7522 - val_bbox_output_loss: 113.4364 - val_class_output_loss:
0.3157 - val_bbox_output_mean_squared_error: 113.4364 -
val_class_output_accuracy: 0.9222
Epoch 27/50
45/45 [==============================] - 64s 1s/step - loss: 136.8376 -
bbox_output_loss: 136.4931 - class_output_loss: 0.3446 -
bbox_output_mean_squared_error: 136.4931 - class_output_accuracy: 0.9104 -
val_loss: 111.3368 - val_bbox_output_loss: 111.1024 - val_class_output_loss:
0.2345 - val_bbox_output_mean_squared_error: 111.1024 -
val_class_output_accuracy: 0.9389
Epoch 28/50
45/45 [==============================] - 64s 1s/step - loss: 140.9527 -
bbox_output_loss: 140.6575 - class_output_loss: 0.2952 -
bbox_output_mean_squared_error: 140.6575 - class_output_accuracy: 0.9201 -
val_loss: 122.6985 - val_bbox_output_loss: 122.5185 - val_class_output_loss:
0.1799 - val_bbox_output_mean_squared_error: 122.5185 -
val_class_output_accuracy: 0.9556
Epoch 29/50
45/45 [==============================] - 64s 1s/step - loss: 149.9388 -
bbox_output_loss: 149.6448 - class_output_loss: 0.2939 -
```

```
bbox_output_mean_squared_error: 149.6448 - class_output_accuracy: 0.9194 -
val_loss: 236.6434 - val_bbox_output_loss: 236.3479 - val_class_output_loss:
0.2955 - val_bbox_output_mean_squared_error: 236.3479 -
val_class_output_accuracy: 0.9222
Epoch 30/50
45/45 [==============================] - 66s 1s/step - loss: 135.5885 -
bbox_output_loss: 135.2658 - class_output_loss: 0.3227 -
bbox_output_mean_squared_error: 135.2658 - class_output_accuracy: 0.9062 -
val_loss: 98.7347 - val_bbox_output_loss: 98.5235 - val_class_output_loss:
0.2112 - val_bbox_output_mean_squared_error: 98.5235 -
val_class_output_accuracy: 0.9333
Epoch 31/50
45/45 [==============================] - 64s 1s/step - loss: 122.1219 -
bbox_output_loss: 121.8790 - class_output_loss: 0.2430 -
bbox_output_mean_squared_error: 121.8790 - class_output_accuracy: 0.9326 -
val_loss: 119.1907 - val_bbox_output_loss: 118.9589 - val_class_output_loss:
0.2318 - val_bbox_output_mean_squared_error: 118.9589 -
val_class_output_accuracy: 0.9500
Epoch 32/50
45/45 [==============================] - 64s 1s/step - loss: 137.4256 -
bbox_output_loss: 137.1355 - class_output_loss: 0.2900 -
bbox_output_mean_squared_error: 137.1355 - class_output_accuracy: 0.9222 -
val_loss: 94.4099 - val_bbox_output_loss: 94.1884 - val_class_output_loss:
0.2215 - val_bbox_output_mean_squared_error: 94.1884 -
val_class_output_accuracy: 0.9389
Epoch 33/50
45/45 [==============================] - 64s 1s/step - loss: 112.3670 -
bbox_output_loss: 112.1219 - class_output_loss: 0.2451 -
bbox_output_mean_squared_error: 112.1219 - class_output_accuracy: 0.9312 -
val_loss: 96.4349 - val_bbox_output_loss: 96.1918 - val_class_output_loss:
0.2431 - val_bbox_output_mean_squared_error: 96.1918 -
val_class_output_accuracy: 0.9389
Epoch 34/50
45/45 [==============================] - 64s 1s/step - loss: 97.8007 -
bbox_output_loss: 97.6093 - class_output_loss: 0.1914 -
bbox_output_mean_squared_error: 97.6093 - class_output_accuracy: 0.9479 -
val_loss: 81.0118 - val_bbox_output_loss: 80.8783 - val_class_output_loss:
0.1335 - val_bbox_output_mean_squared_error: 80.8783 -
val_class_output_accuracy: 0.9667
Epoch 35/50
45/45 [==============================] - 65s 1s/step - loss: 101.1951 -
bbox_output_loss: 101.0031 - class_output_loss: 0.1921 -
bbox_output_mean_squared_error: 101.0031 - class_output_accuracy: 0.9438 -
val_loss: 90.5699 - val_bbox_output_loss: 90.3959 - val_class_output_loss:
0.1741 - val_bbox_output_mean_squared_error: 90.3959 -
val_class_output_accuracy: 0.9444
Epoch 36/50
45/45 [==============================] - 65s 1s/step - loss: 115.3203 -
```

```
bbox_output_loss: 115.1244 - class_output_loss: 0.1959 -
bbox_output_mean_squared_error: 115.1244 - class_output_accuracy: 0.9417 -
val_loss: 194.1462 - val_bbox_output_loss: 193.9607 - val_class_output_loss:
0.1856 - val_bbox_output_mean_squared_error: 193.9607 -
val_class_output_accuracy: 0.9278
Epoch 37/50
45/45 [==============================] - 65s 1s/step - loss: 111.6793 -
bbox_output_loss: 111.4890 - class_output_loss: 0.1903 -
bbox_output_mean_squared_error: 111.4890 - class_output_accuracy: 0.9438 -
val_loss: 95.8117 - val_bbox_output_loss: 95.6331 - val_class_output_loss:
0.1785 - val_bbox_output_mean_squared_error: 95.6331 -
val_class_output_accuracy: 0.9556
Epoch 38/50
45/45 [==============================] - 64s 1s/step - loss: 95.7623 -
bbox_output_loss: 95.5822 - class_output_loss: 0.1802 -
bbox_output_mean_squared_error: 95.5822 - class_output_accuracy: 0.9431 -
val_loss: 77.6527 - val_bbox_output_loss: 77.4974 - val_class_output_loss:
0.1553 - val_bbox_output_mean_squared_error: 77.4974 -
val_class_output_accuracy: 0.9500
Epoch 39/50
45/45 [==============================] - 64s 1s/step - loss: 98.0480 -
bbox_output_loss: 97.8901 - class_output_loss: 0.1578 -
bbox_output_mean_squared_error: 97.8901 - class_output_accuracy: 0.9479 -
val_loss: 128.0800 - val_bbox_output_loss: 127.8969 - val_class_output_loss:
0.1831 - val_bbox_output_mean_squared_error: 127.8969 -
val_class_output_accuracy: 0.9444
Epoch 40/50
45/45 [==============================] - 65s 1s/step - loss: 90.0308 -
bbox_output_loss: 89.8848 - class_output_loss: 0.1460 -
bbox_output_mean_squared_error: 89.8848 - class_output_accuracy: 0.9521 -
val_loss: 85.0326 - val_bbox_output_loss: 84.8531 - val_class_output_loss:
0.1795 - val_bbox_output_mean_squared_error: 84.8531 -
val_class_output_accuracy: 0.9500
Epoch 41/50
45/45 [==============================] - 64s 1s/step - loss: 85.7098 -
bbox_output_loss: 85.6069 - class_output_loss: 0.1029 -
bbox_output_mean_squared_error: 85.6069 - class_output_accuracy: 0.9688 -
val_loss: 103.1659 - val_bbox_output_loss: 103.0169 - val_class_output_loss:
0.1490 - val_bbox_output_mean_squared_error: 103.0169 -
val_class_output_accuracy: 0.9556
Epoch 42/50
45/45 [==============================] - 64s 1s/step - loss: 75.0759 -
bbox_output_loss: 74.9564 - class_output_loss: 0.1195 -
bbox_output_mean_squared_error: 74.9564 - class_output_accuracy: 0.9681 -
val_loss: 100.1087 - val_bbox_output_loss: 100.0027 - val_class_output_loss:
0.1060 - val_bbox_output_mean_squared_error: 100.0027 -
val_class_output_accuracy: 0.9722
Epoch 43/50
```

```
45/45 [==============================] - 64s 1s/step - loss: 77.8824 -
bbox_output_loss: 77.7932 - class_output_loss: 0.0892 -
bbox_output_mean_squared_error: 77.7932 - class_output_accuracy: 0.9778 -
val_loss: 83.1069 - val_bbox_output_loss: 82.9732 - val_class_output_loss:
0.1337 - val_bbox_output_mean_squared_error: 82.9732 -
val_class_output_accuracy: 0.9667
Epoch 44/50
45/45 [==============================] - 64s 1s/step - loss: 81.1486 -
bbox_output_loss: 81.0318 - class_output_loss: 0.1168 -
bbox_output_mean_squared_error: 81.0318 - class_output_accuracy: 0.9604 -
val_loss: 74.7546 - val_bbox_output_loss: 74.6198 - val_class_output_loss:
0.1348 - val_bbox_output_mean_squared_error: 74.6198 -
val_class_output_accuracy: 0.9667
Epoch 45/50
45/45 [==============================] - 64s 1s/step - loss: 66.1415 -
bbox_output_loss: 66.0343 - class_output_loss: 0.1072 -
bbox_output_mean_squared_error: 66.0343 - class_output_accuracy: 0.9667 -
val_loss: 122.2838 - val_bbox_output_loss: 122.1754 - val_class_output_loss:
0.1084 - val_bbox_output_mean_squared_error: 122.1754 -
val_class_output_accuracy: 0.9722
Epoch 46/50
45/45 [==============================] - 64s 1s/step - loss: 76.4345 -
bbox_output_loss: 76.3593 - class_output_loss: 0.0752 -
bbox_output_mean_squared_error: 76.3593 - class_output_accuracy: 0.9736 -
val_loss: 136.0583 - val_bbox_output_loss: 135.9038 - val_class_output_loss:
0.1544 - val_bbox_output_mean_squared_error: 135.9038 -
val_class_output_accuracy: 0.9556
Epoch 47/50
45/45 [==============================] - 64s 1s/step - loss: 78.5572 -
bbox_output_loss: 78.4911 - class_output_loss: 0.0660 -
bbox_output_mean_squared_error: 78.4911 - class_output_accuracy: 0.9785 -
val_loss: 111.4833 - val_bbox_output_loss: 111.3492 - val_class_output_loss:
0.1341 - val_bbox_output_mean_squared_error: 111.3492 -
val_class_output_accuracy: 0.9611
Epoch 48/50
45/45 [==============================] - 64s 1s/step - loss: 79.5824 -
bbox_output_loss: 79.5063 - class_output_loss: 0.0761 -
bbox_output_mean_squared_error: 79.5063 - class_output_accuracy: 0.9812 -
val_loss: 78.2473 - val_bbox_output_loss: 78.1156 - val_class_output_loss:
0.1317 - val_bbox_output_mean_squared_error: 78.1156 -
val_class_output_accuracy: 0.9722
Epoch 49/50
45/45 [==============================] - 65s 1s/step - loss: 66.4366 -
bbox_output_loss: 66.3671 - class_output_loss: 0.0696 -
bbox_output_mean_squared_error: 66.3671 - class_output_accuracy: 0.9819 -
val_loss: 76.1493 - val_bbox_output_loss: 76.0485 - val_class_output_loss:
0.1008 - val_bbox_output_mean_squared_error: 76.0485 -
val_class_output_accuracy: 0.9722
```

```
Epoch 50/50
45/45 [==============================] - 65s 1s/step - loss: 61.2886 -
bbox_output_loss: 61.2162 - class_output_loss: 0.0724 -
bbox_output_mean_squared_error: 61.2162 - class_output_accuracy: 0.9736 -
val_loss: 74.9232 - val_bbox_output_loss: 74.7773 - val_class_output_loss:
0.1459 - val_bbox_output_mean_squared_error: 74.7773 -
val_class_output_accuracy: 0.9667
```

[10]:
```
model.evaluate(
    test_images,
    {'bbox_output': test_bbox, 'class_output': test_labels}
)
```

```
6/6 [==============================] - 1s 222ms/step - loss: 117.2741 -
bbox_output_loss: 117.1721 - class_output_loss: 0.1021 -
bbox_output_mean_squared_error: 117.1721 - class_output_accuracy: 0.9722
```

[10]:
```
[117.27412414550781,
 117.17205810546875,
 0.1020708754658699,
 117.17205810546875,
 0.9722222089767456]
```

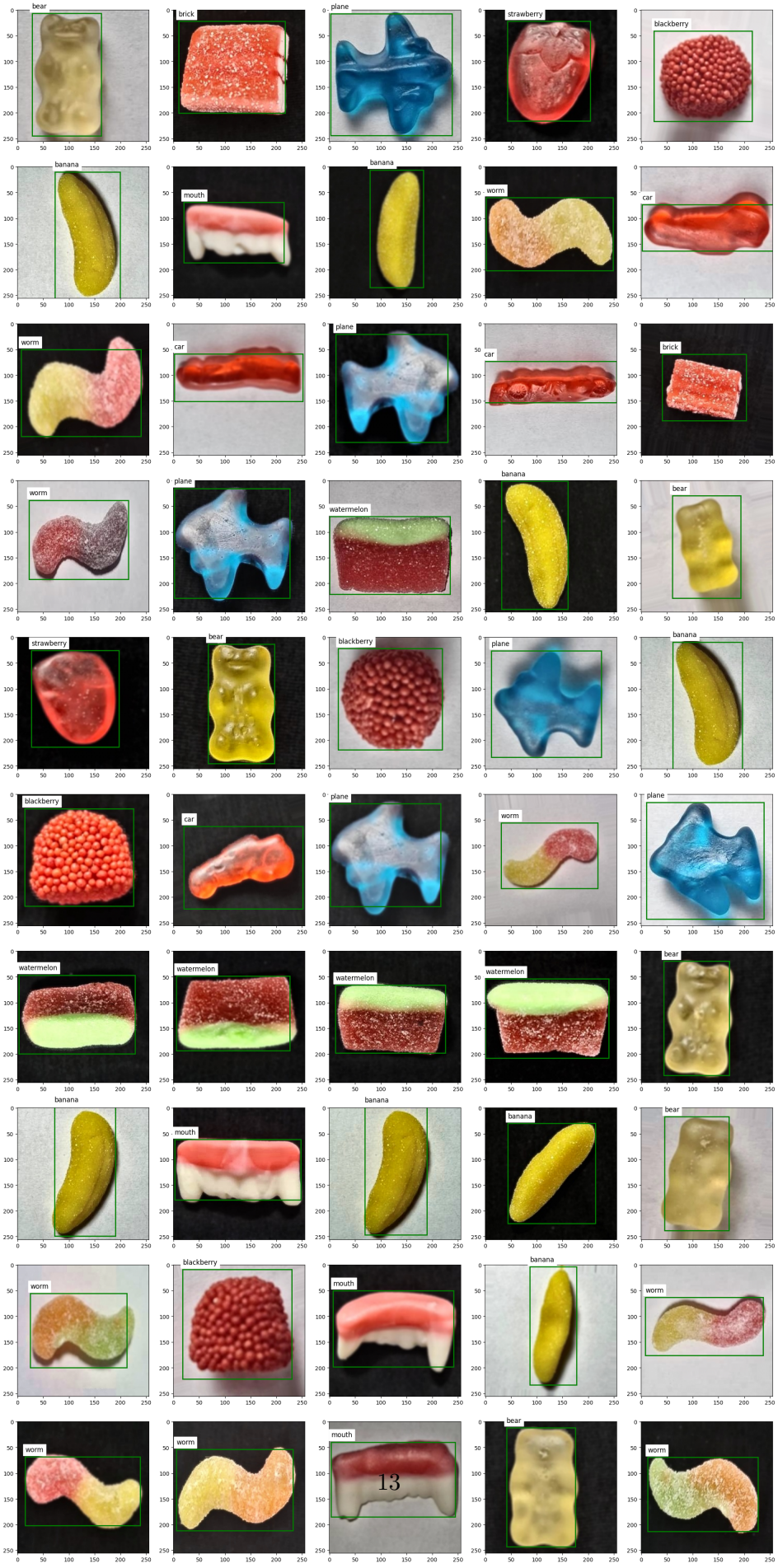
[11]:
```
model.save(os.path.join(base_dir, 'models/cnn_bounding_box_model.h5'))
```

[12]:
```
predicted_bboxes, predicted_classes = model.predict(test_images)
predicted_classes = np.argmax(predicted_classes, axis=1)
true_classes = np.argmax(test_labels, axis=1)
```

```
6/6 [==============================] - 2s 221ms/step
```

[13]:
```
class_labels = ['banana', 'bear','blackberry', 'brick', 'car', 'mouth',␣
 ↪'plane', 'strawberry', 'watermelon', 'worm']

def plot_image_with_bbox(image, bbox, predicted_class, class_labels, ax):
    ax.imshow(image)
    rect = patches.Rectangle((bbox[0], bbox[1]), bbox[2], bbox[3],
                             linewidth=2, edgecolor='g', facecolor='none')
    ax.add_patch(rect)
    ax.text(bbox[0], bbox[1] - 10, f'{class_labels[predicted_class]}',␣
 ↪fontsize=12, backgroundcolor='white')

fig, axes = plt.subplots(10, 5, figsize=(20, 40))
for i, ax in enumerate(axes.flat):
    image = test_images[i]

    bbox = predicted_bboxes[i]
    predicted_class = predicted_classes[i]
```

```
    plot_image_with_bbox(image, bbox, predicted_class, class_labels, ax)

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


bear
brick
plane
strawberry
blackberry
banana
mouth
banana
worm
car
worm
car
plane
car
brick
worm
plane
watermelon
banana
bear
strawberry
bear
blackberry
plane
banana
blackberry
car
plane
worm
plane
watermelon
watermelon
watermelon
watermelon
bear
banana
mouth
banana
banana
bear
worm
blackberry
mouth
banana
worm
worm
worm
mouth
bear
worm

After running our model for 50 epochs, the model produced the following metric results:

```
bbox_output_loss: 117.1721 - class_output_loss: 0.1021 - bbox_output_mean_squared_error:
117.1721 - class_output_accuracy: 0.9722
```

- Analysis: Before making any assumptions, it is crucial to evaluate the model's performance by checking the test results. As observed from the metrics presented above, the model was able to correctly identify most cases. However, it is also important to further assess the results by plotting a confusion matrix and calculating the F1 score to properly evaluate the model.

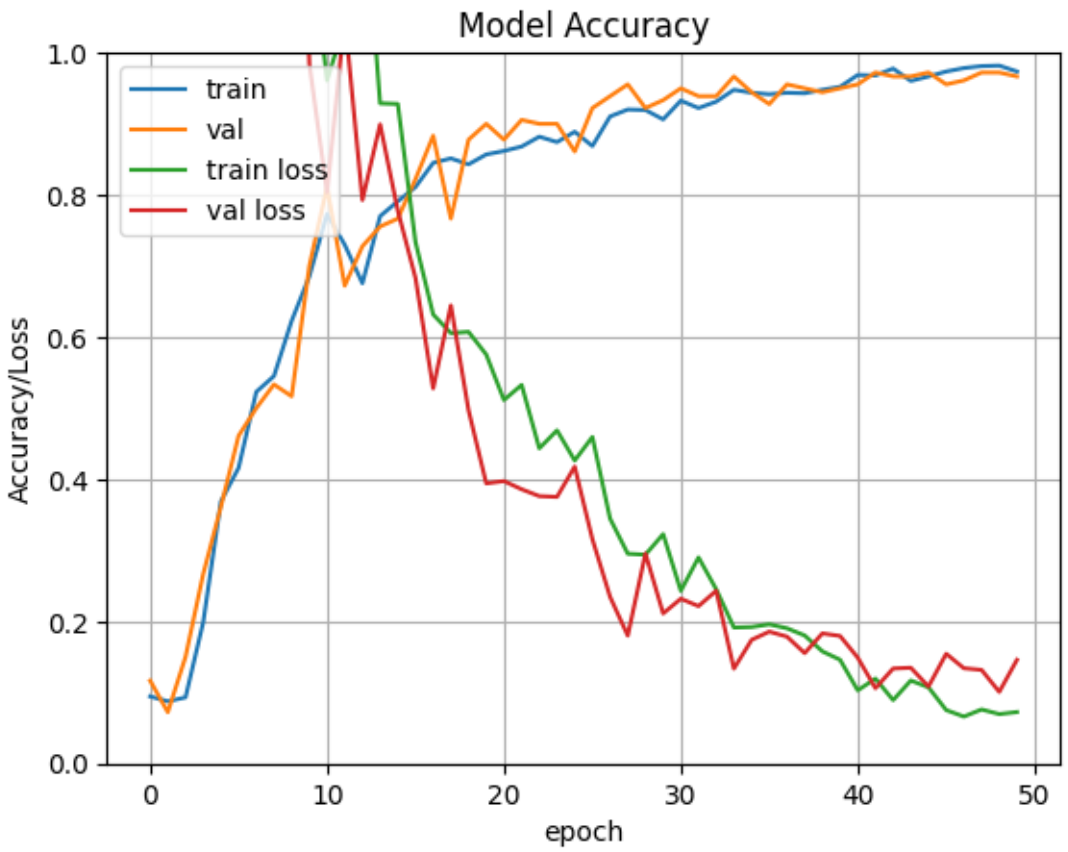
0.0.2 7. Model Evaluation

The performance of the model is evaluated using metrics such as accuracy, F1-score, and confusion matrix.

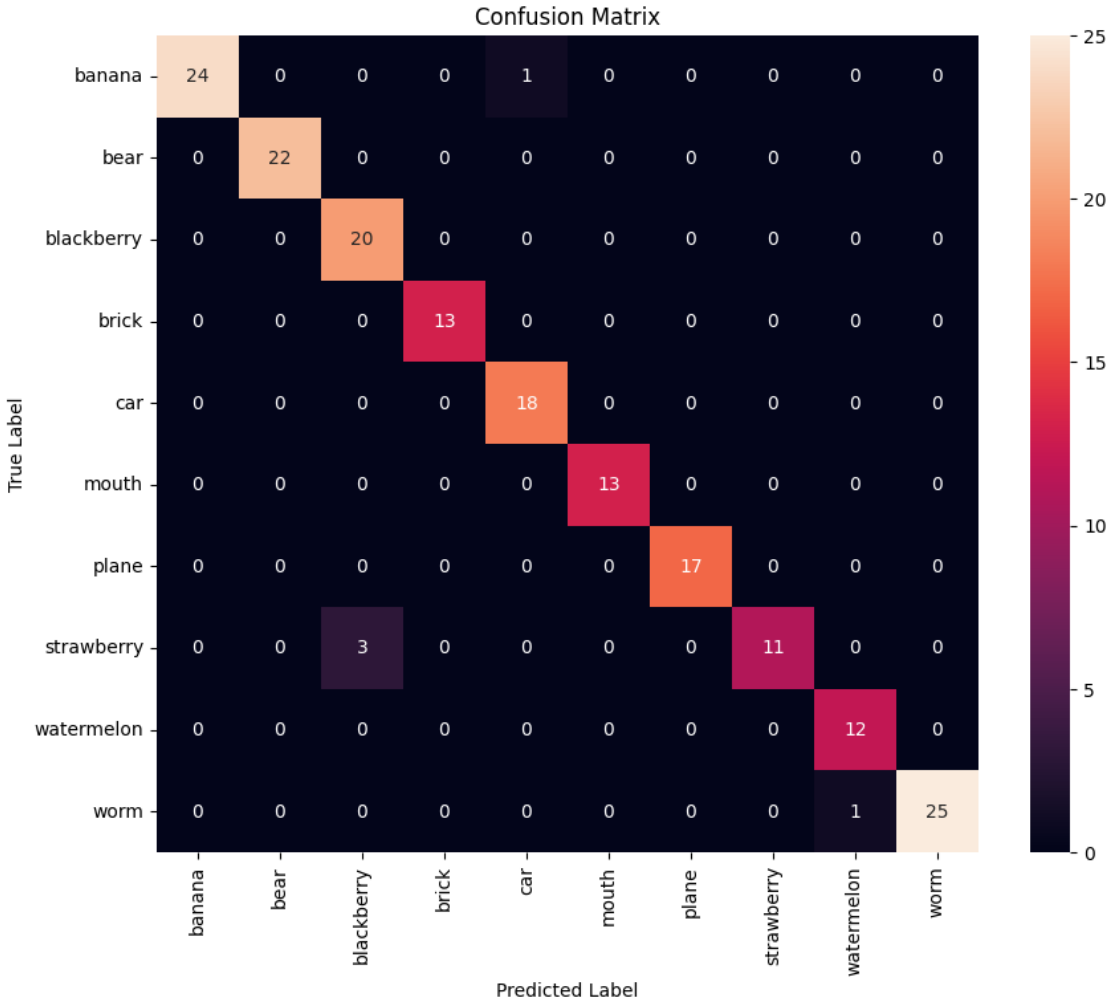
[14]:
```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import seaborn as sns
import numpy as np

accuracy = accuracy_score(true_classes, predicted_classes)
print(f"Accuracy: {accuracy:.4f}")

f1 = f1_score(true_classes, predicted_classes, average='weighted')
print(f"F1-Score: {f1:.4f}")
```

```
Accuracy: 0.9722
F1-Score: 0.9719
```

[18]:
```
# Accuracy x Epoch plot
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]

plt.plot(history.history['class_output_accuracy'])
plt.plot(history.history['val_class_output_accuracy'])

plt.plot(history.history['class_output_loss'])
plt.plot(history.history['val_class_output_loss'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy/Loss')
plt.xlabel('epoch')

plt.legend(['train', 'val', 'train loss', 'val loss'], loc='upper left')

plt.show()
```



[16]:
```
conf_matrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', xticklabels=class_labels,␣
 ↪yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



- Conclusion: Although there were some errors, the model demonstrated the ability to generalize and maintain balance. In cases of errors, for example, strawberry and blackberry have more reddish colors and round shapes, which could indeed be a challenge for our model's classification. Both are the only ones with very similar shapes. Overall, we believe that being meticulous with data preprocessing was crucial for achieving these results. We consider the normalization and analysis steps before training the model essential to identify the features of our images and prepare them in the best possible way before training.