Projeto - Processamento Digital de Imagens

Alunos: Gabriel de Oliveira e Silva Padovani Barreto RA: 2326221 Johanna Kirchner RA: 2265010

https://github.com/Barreto-G/image_classifier_pdi

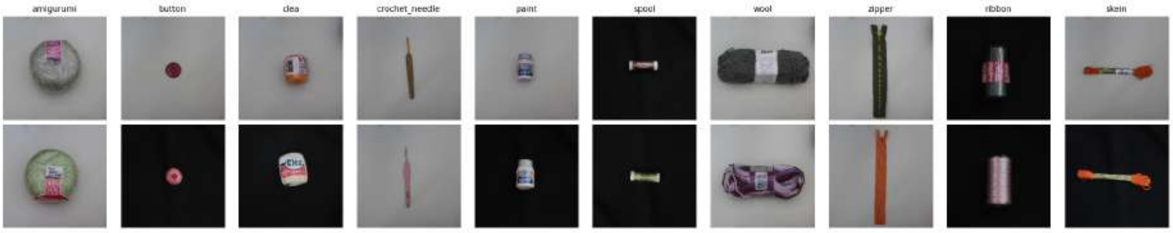
Parte 1: Definição do tema, coleta e pré-processamento de imagens

Foram escolhidos itens de armarinho como tema dos objetos que formam o dataset. Foram determinadas 10 classes de objetos, cada uma contendo 5 itens dos quais foram capturadas 4 fotos utilizando uma camera digital, com fundo e posicionamento variados, combinando dois tipos de fundo com duas posições diferentes. As imagens foram armazenadas em pastas de acordo com sua classe, e nomeadas seguindo a formatação `<CLASSID>-<IMG_SEQUENCE>-V1|V2-B|W.png`.

A exibição das imagens e os metadados do projeto podem ser visualizados no arquivo `show_dataset_info.ipynb`.

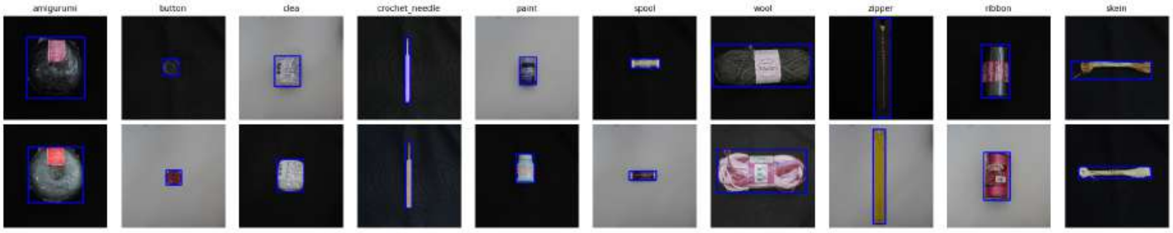
In [1]:
```python
from utilidades import *
from trim_dataset import TrimDataset
```

In [2]:
```python
dataset = TrimDataset('trims_dataset')
plot_class_grid(dataset, 2, 'nobbox')
```



Para a marcação das imagens, utilizou-se o CVAT, onde foi marcado o objeto de interesse em cada uma das imagens.

In [3]:
```python
plot_class_grid(dataset, 2, 'bbox')
```



Para realizar o pré-processamento das imagens, e a subsequente criação do dataset, criou-se a classe `TrimDataset`, que armazena todas as informações relevantes do

Dataset e os métodos utilizados para o seu pré-processamento. Tal classe está localizada no arquivo `trims_dataset.py`. A primeira etapa do pré-processamento foi diminuir a resolução das imagens capturadas, tanto para diminuir o espaço ocupado pelos arquivos quando para acelerar o treinamento do modelo. A resolução das imagens originais foi diminuida de 2112x2112 para 224x224 sem perdas de eficiência do modelo.

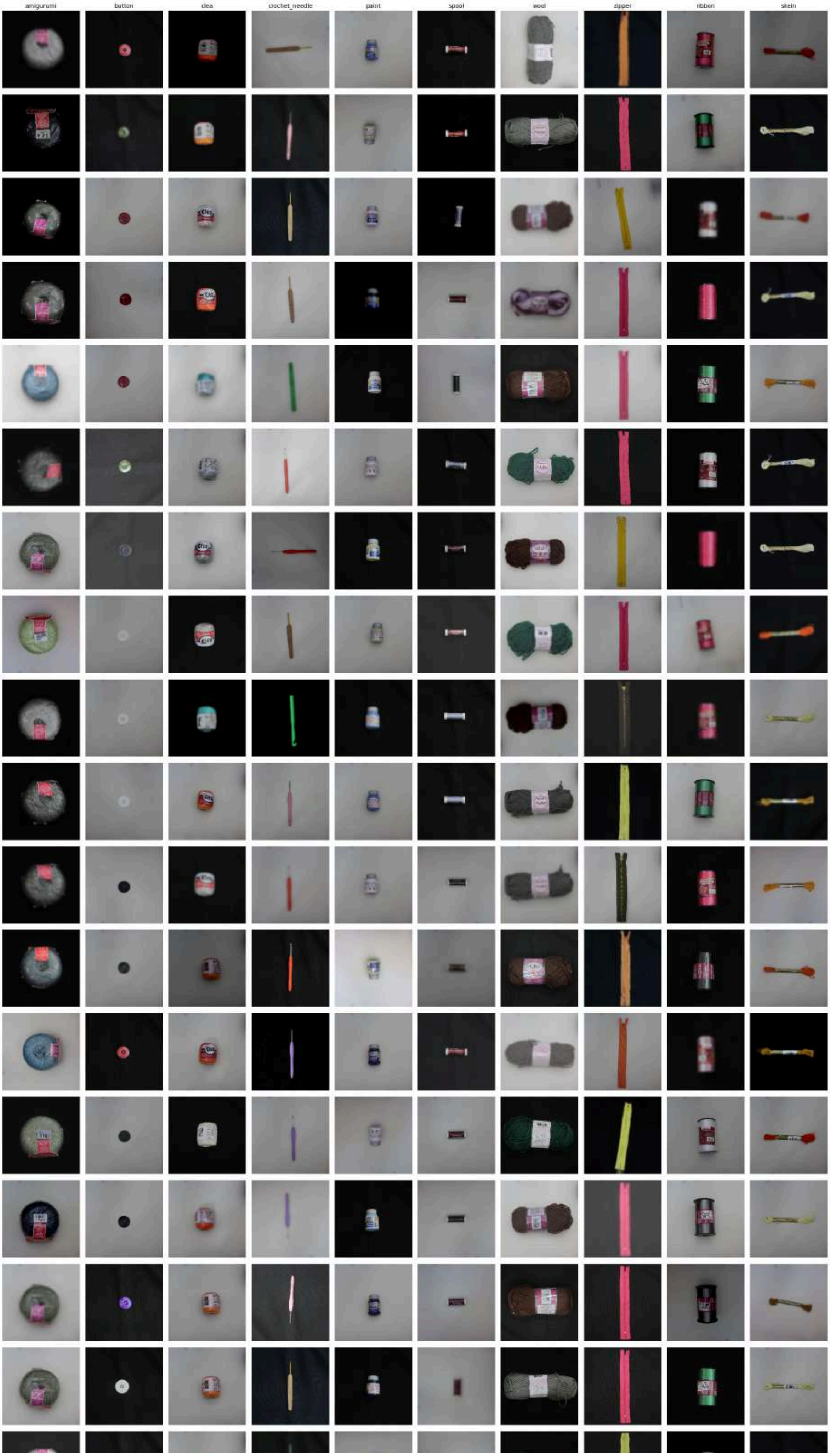
Essa primeira etapa é realizada pela função `resize_image`, aplicada já na instancia de um objeto `TrimDataset`. Para reduzir o gasto computacional e de memória, as imagens incluídas nos arquivos do projeto já estão com a resolução reduzida, podendo ser encontradas na pasta `trims_dataset` ou arquivo .tar.gz de mesmo nome.

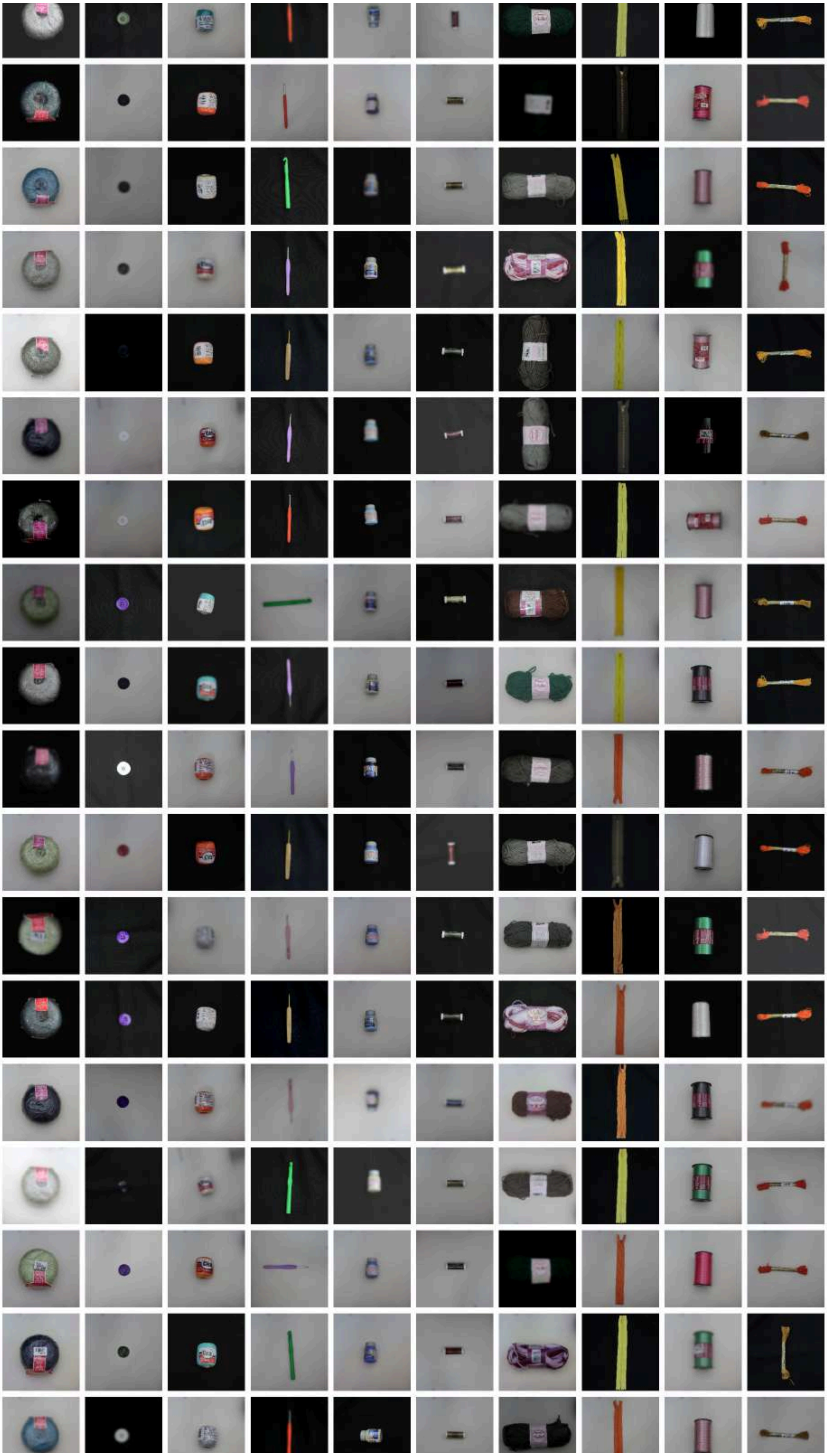
A segunda etapa envolve a aumentação do dataset, realizada pela função `dataset_augmentation` dentro da classe `TrimDataset`. Utilizando a biblioteca Albumentations, tal função aplica uma série de transformações a cada imagem do dataset, sendo elas: Uma dentre quatro transformações geométricas possíveis, ajuste aleatório do brilho e do contraste (50% de chance), blur gaussiano (50% de chance), ajuste da saturação da imagem (50% de chance).

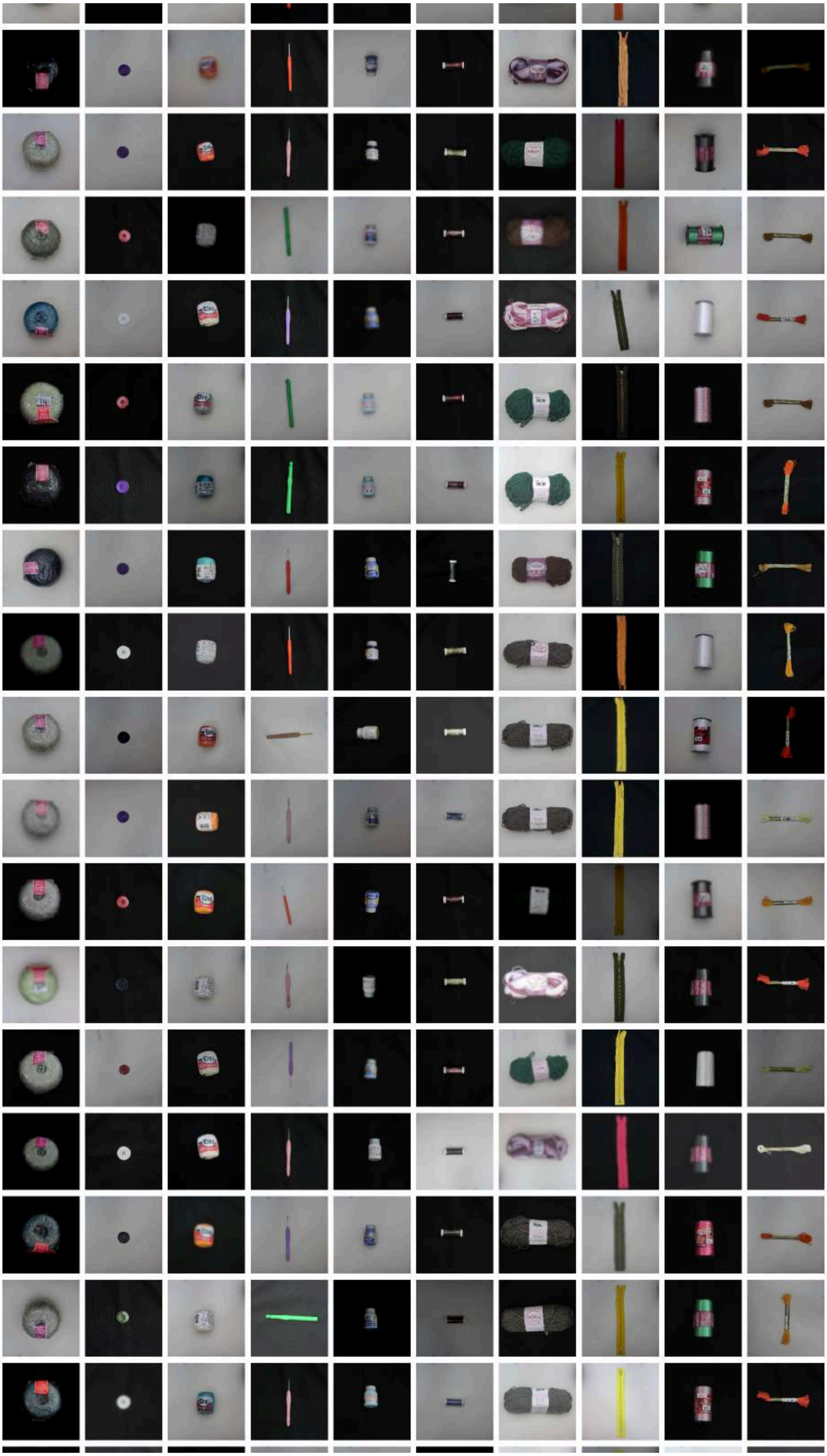
Ao fim dessa etapa, o dataset aumenta de 200 fotos originais para 600 fotos aumentadas, sendo que essa nova versão do dataset é salvo em uma nova pasta denominada `augmented_dataset`.

No pedaço de código abaixo, executamos o `dataset_augmentation` e plotamos todas as fotos que compõem o dataset até o momento.

In [4]:
```
dataset.dataset_augmentation()
plot_class_grid(dataset, 60, "nobbox")
```


amigurumi
button
clea
crochet_needle
paint
spool
wool
zipper
ribbon
skein





A terceira e última etapa do pré-processamento é a normalização do histograma das imagens do dataset, realizada pela função `normalize_dataset` dentro da classe `TrimDataset`. Essa função aplica equalização do histrograma para cada imagem do dataset, salvando as novas imagens em uma outra pasta denominada `normalized_dataset`.

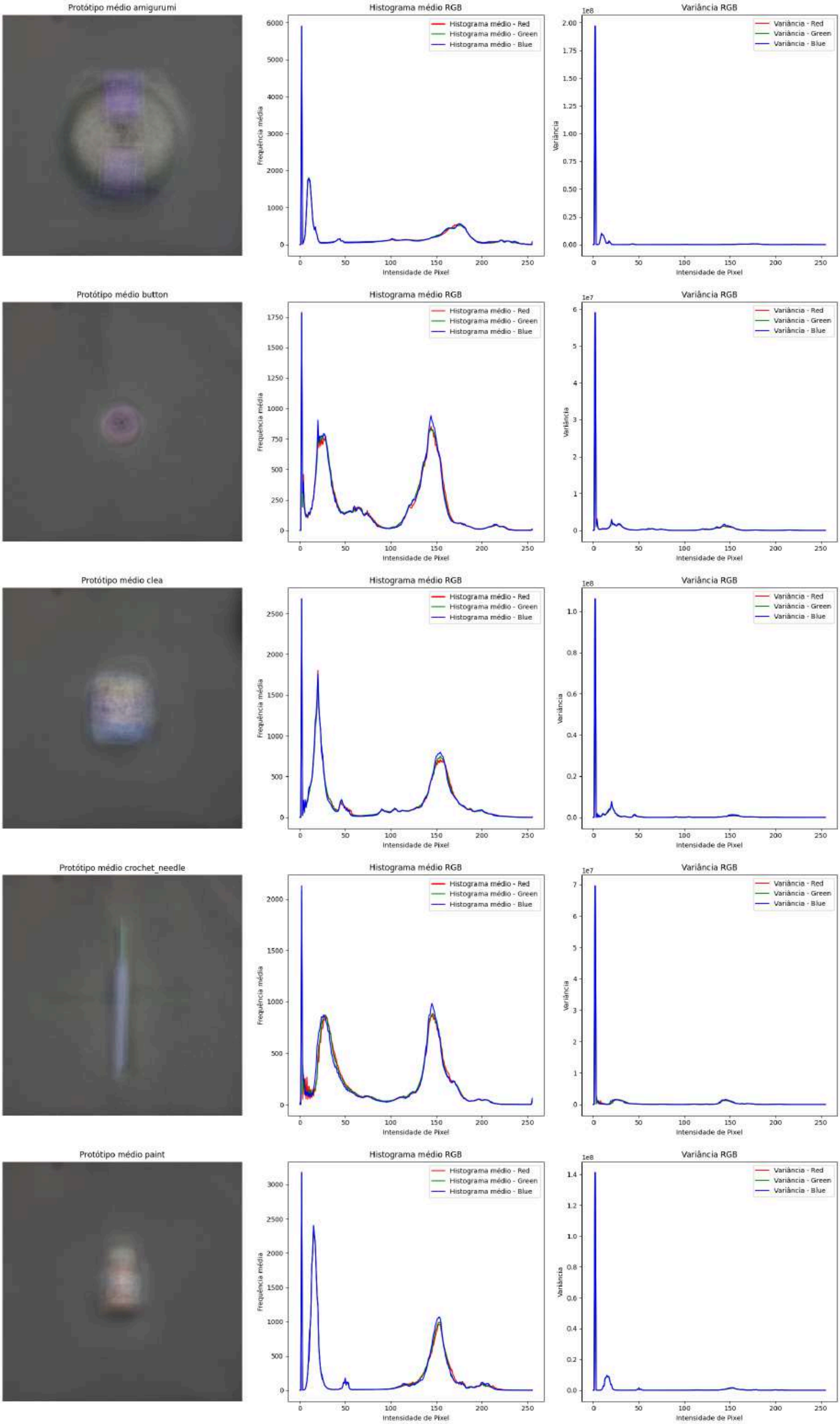
In [5]:
```
dataset.normalize_dataset()
```

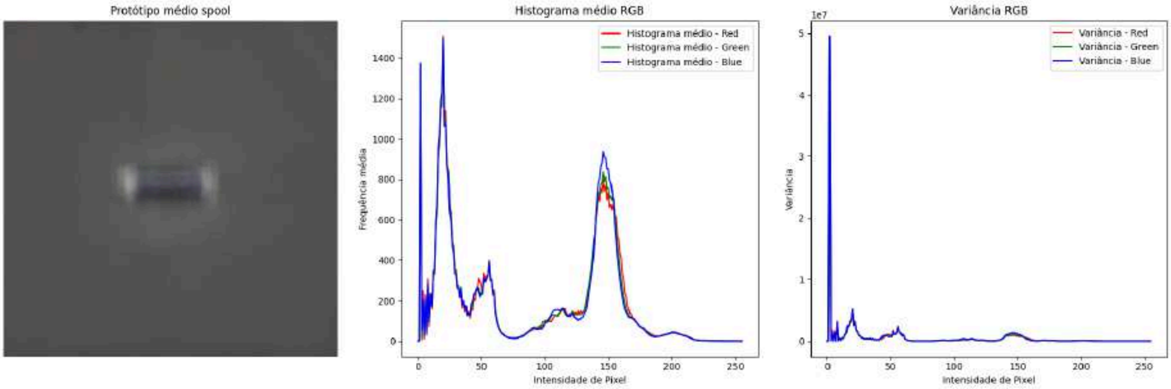
Após a etapa de normalização, é necessário recarregar o dataset, isto se dá pois as novas imagens normalizadas são salvas na pasta mas não são armazenadas no objeto `dataset`.

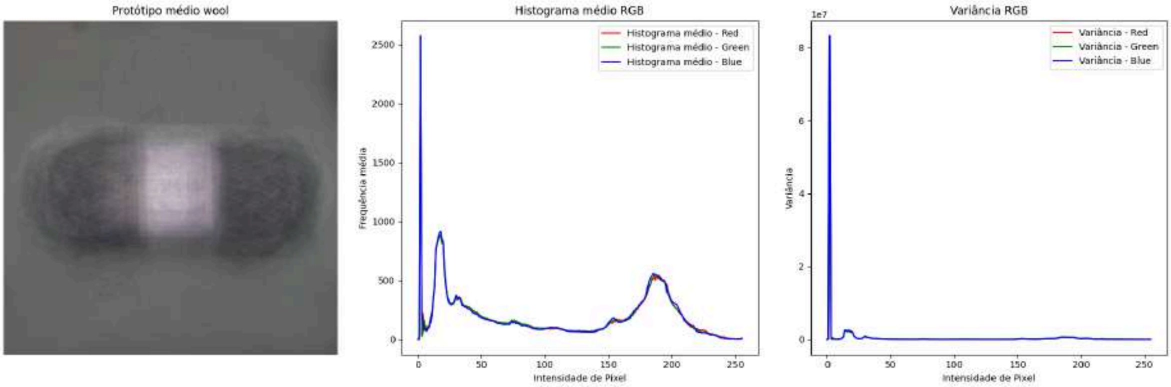
In [6]:
```
new_dataset = TrimDataset('normalized_dataset')
```

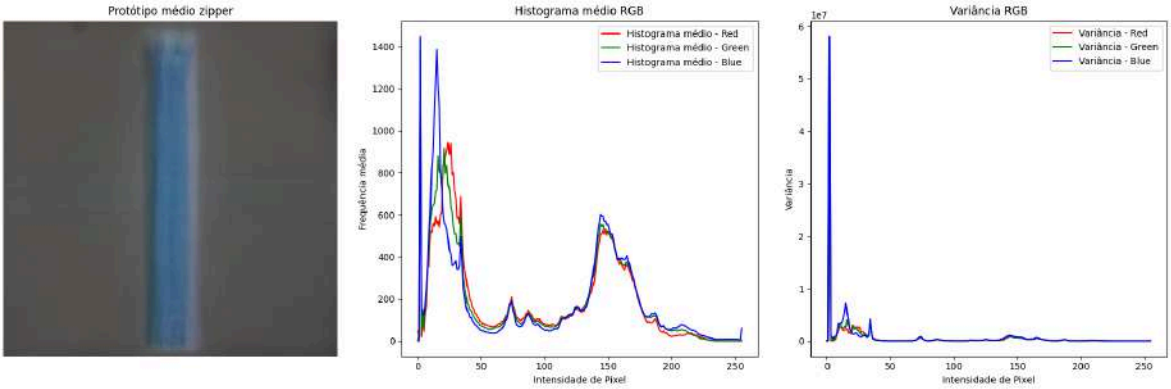
Com o dataset normalizado, abaixo podemos ver o protótipo médio, histograma médio, e a variância do histograma de cada classe.

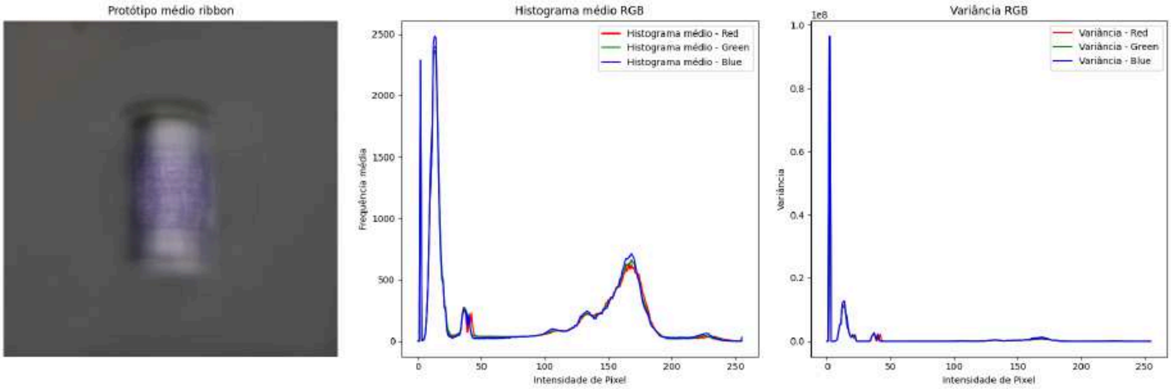
In [7]:
```
generate_image_statistics(new_dataset.images, new_dataset.categories)
```

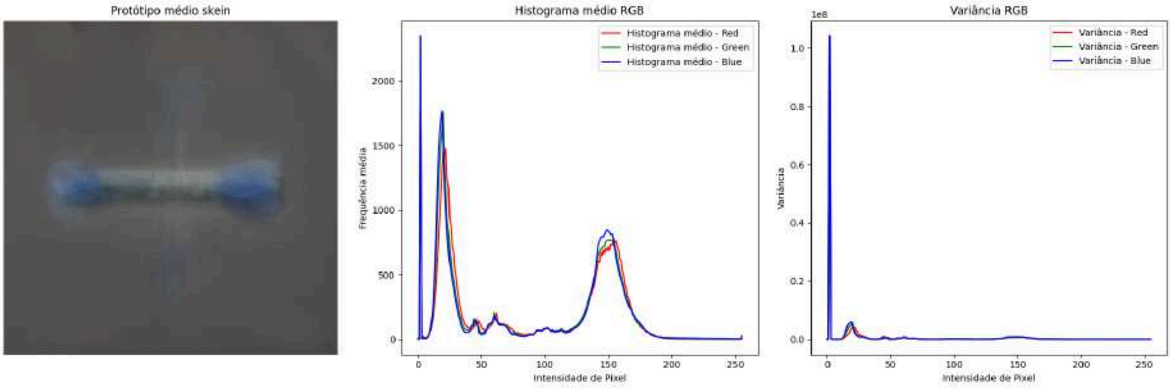

Protótipo médio amigurumi
Histograma médio RGB
Histograma médio - Red
Histograma médio - Green
Histograma médio - Blue
Frequência média
Intensidade de Pixel
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200
250
Variância RGB
1e8
Variância - Red
Variância - Green
Variância - Blue
Variância
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Protótipo médio button
Histograma médio RGB
0
250
500
750
1000
1250
1500
1750
Variância RGB
1e7
0
1
2
3
4
5
6
Protótipo médio clea
Histograma médio RGB
0
500
1000
1500
2000
2500
Variância RGB
1e8
0.0
0.2
0.4
0.6
0.8
1.0
Protótipo médio crochet_needle
Histograma médio RGB
0
500
1000
1500
2000
Variância RGB
1e7
0
1
2
3
4
5
6
7
Protótipo médio paint
Histograma médio RGB
0
500
1000
1500
2000
2500
3000
Variância RGB
1e8
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4


Protótipo médio spool
Histograma médio RGB
Histograma médio - Red
Histograma médio - Green
Histograma médio - Blue
Frequência média
0
200
400
600
800
1000
1200
1400
Intensidade de Pixel
0
50
100
150
200
250
Variância RGB
1e7
Variância - Red
Variância - Green
Variância - Blue
Variância
0
1
2
3
4
5


Protótipo médio wool
Histograma médio RGB
Histograma médio - Red
Histograma médio - Green
Histograma médio - Blue
Frequência média
0
500
1000
1500
2000
2500
Intensidade de Pixel
0
50
100
150
200
250
Variância RGB
1e7
Variância - Red
Variância - Green
Variância - Blue
Variância
0
2
4
6
8

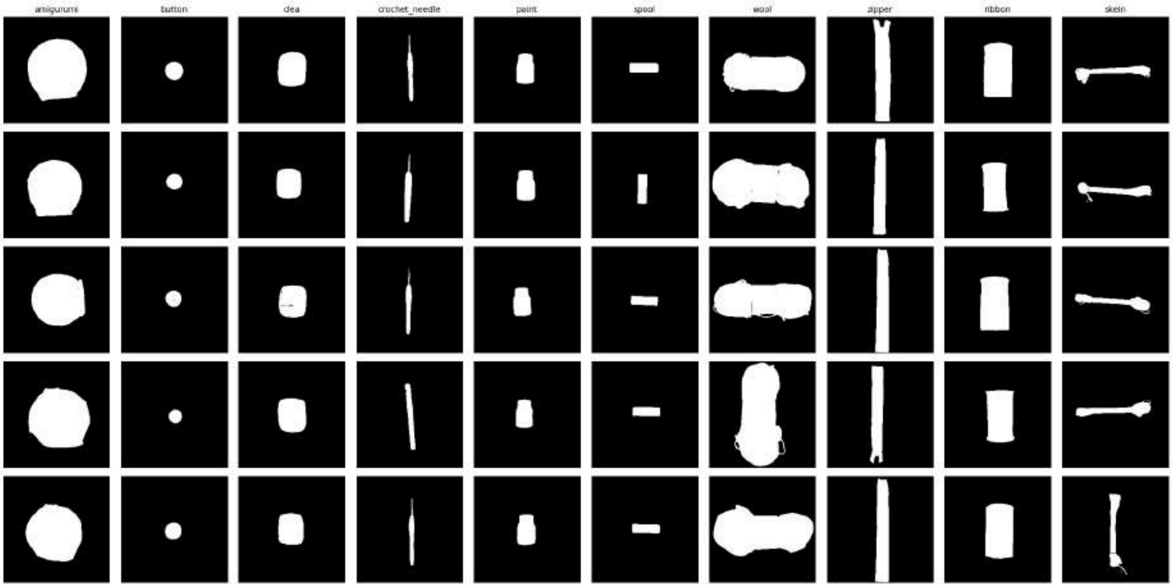

Protótipo médio zipper
Histograma médio RGB
Histograma médio - Red
Histograma médio - Green
Histograma médio - Blue
Frequência média
0
200
400
600
800
1000
1200
1400
Intensidade de Pixel
0
50
100
150
200
250
Variância RGB
1e7
Variância - Red
Variância - Green
Variância - Blue
Variância
0
1
2
3
4
5
6


Protótipo médio ribbon
Histograma médio RGB
Histograma médio - Red
Histograma médio - Green
Histograma médio - Blue
Frequência média
0
500
1000
1500
2000
2500
Intensidade de Pixel
0
50
100
150
200
250
Variância RGB
1e8
Variância - Red
Variância - Green
Variância - Blue
Variância
0.0
0.2
0.4
0.6
0.8
1.0


Protótipo médio skein
Histograma médio RGB
Histograma médio - Red
Histograma médio - Green
Histograma médio - Blue
Frequência média
0
500
1000
1500
2000
Intensidade de Pixel
0
50
100
150
200
250
Variância RGB
1e8
Variância - Red
Variância - Green
Variância - Blue
Variância
0.0
0.2
0.4
0.6
0.8
1.0

No CVAT, além da marcação das bounding boxes, também realizamos a segmentação de todas as imagens, informação esta que foi carregada para as etapas de augmentation e normalization. Dessa forma, o dataset final contem o Ground Truth de todas as imagens, mesmo que apenas a bounding box seja utilizada para o treinamento do modelo na etapa posterior. O código abaixo gera o ground truth de 5 imagens aleatórias de cada classe.

```
In [8]: plot_class_grid(new_dataset, 5, "gt_mask")
```



Parte 2: Treinamento do classificador utilizando CNN

Utilizando a biblioteca TensorFlow, treinamos um modelo classificador baseado em CNN.

```
In [9]: input_shape = (224,224,3) # Tamanho fixo das imagens no Dataset
        num_classes = 10          # Numero de classes para classificar

        model = create_cnn(input_shape, num_classes) # classificador a ser treinado
```

Antes de treinarmos o modelo, as imagens devem ser organizadas, separadas e pré-processadas para se adequarem aos formatos utilizados pela biblioteca TensorFlow. Primeiramente, organizamos todas as imagens, bbox e categorias na forma de um dicionário, então, dividimos os dados para cada etapa: 80% para treinamento, 10% para teste e 10% para validação.

```
In [10]: images = [data_to_dict(img.content, img.bbox, img.category_id) for img in new_da
         img_train, img_test, img_validation = split_dataset(images)
```

Agora, separamos as informações nos conjuntos de treino teste e validação, isto é, para cada conjunto, separamos as informações das imagens, classes e bbox em vetores, pré-processando-os para serem utilizados pela biblioteca da TensorFlow. Além disso, as imagens devem ser convertidas do formato BGR do cv2 para o formato RGB, normalizado de 0 a 1, para serem utilizadas pelo modelo.

In [11]:

```
from tensorflow.python.keras.utils.np_utils import to_categorical
import numpy as np

# Imagens de Treino
x_train = [img['image'] for img in img_train]
x_train = preprocess_images(x_train)

# Classes das imagens de treino
y_train_classes = np.array([img['class'] - 1 for img in img_train])
y_train_classes = to_categorical(y_train_classes, num_classes)

# Bboxes das imagens de treino
y_train_bbox = [img['bbox'] for img in img_train]
y_train_bbox = np.array(y_train_bbox) #/ [224.0, 224.0, 224.0, 224.0]


# Mesmo processo para imagens de validação e teste
x_val = [img['image'] for img in img_validation]
x_val = preprocess_images(x_val)
y_val_classes = np.array([img['class'] - 1 for img in img_validation])
y_val_classes = to_categorical(y_val_classes, num_classes)
y_val_bbox = [img['bbox'] for img in img_validation]
y_val_bbox = np.array(y_val_bbox) #/ [224.0, 224.0, 224.0, 224.0]

x_test = [img['image'] for img in img_test]
x_test = preprocess_images(x_test)
y_test_classes = np.array([img['class'] - 1 for img in img_test])
y_test_classes = to_categorical(y_test_classes, num_classes)
y_test_bbox = [img['bbox'] for img in img_test]
y_test_bbox = np.array(y_test_bbox) #/ [224.0, 224.0, 224.0, 224.0]
```

Agora finalmente podemos treinar o modelo. Os valores de `epoch` e `batch_size` foram escolhidos empiricamente por meio de diversas tentativas até que o modelo atingisse bons resultados nas etapas de teste e validação.

In [12]:

```
history = model.fit(
    x_train,
    {'class_output': y_train_classes, 'bbox_output': y_train_bbox},
    epochs=50,
    batch_size=32,
    validation_data=(x_val, {'class_output': y_val_classes, 'bbox_output': y_val
)
```

```
Epoch 1/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 13s 623ms/step - bbox_output_loss: 5604.3750 - bbox_ou
tput_mean_squared_error: 5604.3750 - class_output_accuracy: 0.1106 - class_output
_loss: 7.3032 - loss: 5611.6782 - val_bbox_output_loss: 2376.2495 - val_bbox_outp
ut_mean_squared_error: 2365.2156 - val_class_output_accuracy: 0.1167 - val_class_
output_loss: 8.1233 - val_loss: 2373.3506
Epoch 2/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 9s 630ms/step - bbox_output_loss: 2168.9563 - bbox_out
put_mean_squared_error: 2168.9563 - class_output_accuracy: 0.0921 - class_output_
loss: 8.9122 - loss: 2177.8687 - val_bbox_output_loss: 1633.5156 - val_bbox_outpu
t_mean_squared_error: 1630.2109 - val_class_output_accuracy: 0.0000e+00 - val_cla
ss_output_loss: 10.9990 - val_loss: 1641.2677
Epoch 3/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 708ms/step - bbox_output_loss: 1635.6621 - bbox_ou
tput_mean_squared_error: 1635.6621 - class_output_accuracy: 0.0953 - class_output
_loss: 10.1614 - loss: 1645.8234 - val_bbox_output_loss: 898.5214 - val_bbox_outp
ut_mean_squared_error: 902.5738 - val_class_output_accuracy: 0.0667 - val_class_o
utput_loss: 8.3130 - val_loss: 910.9404
Epoch 4/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 722ms/step - bbox_output_loss: 813.1842 - bbox_out
put_mean_squared_error: 813.1842 - class_output_accuracy: 0.1589 - class_output_l
oss: 8.5129 - loss: 821.6970 - val_bbox_output_loss: 532.9447 - val_bbox_output_m
ean_squared_error: 534.9633 - val_class_output_accuracy: 0.3167 - val_class_outpu
t_loss: 7.8876 - val_loss: 542.9113
Epoch 5/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 758ms/step - bbox_output_loss: 531.8262 - bbox_out
put_mean_squared_error: 531.8262 - class_output_accuracy: 0.1965 - class_output_l
oss: 6.8320 - loss: 538.6582 - val_bbox_output_loss: 405.3309 - val_bbox_output_m
ean_squared_error: 410.0552 - val_class_output_accuracy: 0.1667 - val_class_outpu
t_loss: 5.2925 - val_loss: 415.4068
Epoch 6/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 741ms/step - bbox_output_loss: 465.0404 - bbox_out
put_mean_squared_error: 465.0404 - class_output_accuracy: 0.2311 - class_output_l
oss: 4.6225 - loss: 469.6629 - val_bbox_output_loss: 301.5461 - val_bbox_output_m
ean_squared_error: 304.8027 - val_class_output_accuracy: 0.3500 - val_class_outpu
t_loss: 3.3167 - val_loss: 308.1859
Epoch 7/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 726ms/step - bbox_output_loss: 351.7140 - bbox_out
put_mean_squared_error: 351.7140 - class_output_accuracy: 0.3598 - class_output_l
oss: 2.8411 - loss: 354.5551 - val_bbox_output_loss: 255.1482 - val_bbox_output_m
ean_squared_error: 257.8760 - val_class_output_accuracy: 0.4167 - val_class_outpu
t_loss: 2.5585 - val_loss: 260.4806
Epoch 8/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 740ms/step - bbox_output_loss: 263.4869 - bbox_out
put_mean_squared_error: 263.4869 - class_output_accuracy: 0.4463 - class_output_l
oss: 2.5218 - loss: 266.0087 - val_bbox_output_loss: 241.0575 - val_bbox_output_m
ean_squared_error: 243.8037 - val_class_output_accuracy: 0.4167 - val_class_outpu
t_loss: 2.4652 - val_loss: 246.3093
Epoch 9/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 720ms/step - bbox_output_loss: 273.0204 - bbox_out
put_mean_squared_error: 273.0204 - class_output_accuracy: 0.5197 - class_output_l
oss: 2.1720 - loss: 275.1924 - val_bbox_output_loss: 178.5638 - val_bbox_output_m
ean_squared_error: 180.8592 - val_class_output_accuracy: 0.4833 - val_class_outpu
t_loss: 1.9661 - val_loss: 182.8527
Epoch 10/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 730ms/step - bbox_output_loss: 189.3588 - bbox_out
put_mean_squared_error: 189.3588 - class_output_accuracy: 0.5457 - class_output_l
oss: 1.8060 - loss: 191.1648 - val_bbox_output_loss: 206.3238 - val_bbox_output_m
ean_squared_error: 207.9528 - val_class_output_accuracy: 0.5833 - val_class_outpu
t_loss: 1.6983 - val_loss: 209.6672
```

```
Epoch 11/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 737ms/step - bbox_output_loss: 223.9917 - bbox_out
put_mean_squared_error: 223.9917 - class_output_accuracy: 0.6172 - class_output_l
oss: 1.6552 - loss: 225.6470 - val_bbox_output_loss: 190.8862 - val_bbox_output_m
ean_squared_error: 193.0817 - val_class_output_accuracy: 0.5333 - val_class_outpu
t_loss: 1.8020 - val_loss: 194.8918
Epoch 12/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 701ms/step - bbox_output_loss: 183.0847 - bbox_out
put_mean_squared_error: 183.0847 - class_output_accuracy: 0.5752 - class_output_l
oss: 2.1685 - loss: 185.2532 - val_bbox_output_loss: 179.4358 - val_bbox_output_m
ean_squared_error: 181.2851 - val_class_output_accuracy: 0.5500 - val_class_outpu
t_loss: 2.9245 - val_loss: 184.2230
Epoch 13/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 715ms/step - bbox_output_loss: 194.5088 - bbox_out
put_mean_squared_error: 194.5088 - class_output_accuracy: 0.6004 - class_output_l
oss: 2.1343 - loss: 196.6431 - val_bbox_output_loss: 243.1518 - val_bbox_output_m
ean_squared_error: 246.5459 - val_class_output_accuracy: 0.6500 - val_class_outpu
t_loss: 1.1962 - val_loss: 247.7643
Epoch 14/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 740ms/step - bbox_output_loss: 184.3775 - bbox_out
put_mean_squared_error: 184.3775 - class_output_accuracy: 0.7171 - class_output_l
oss: 1.4868 - loss: 185.8643 - val_bbox_output_loss: 186.5235 - val_bbox_output_m
ean_squared_error: 189.1734 - val_class_output_accuracy: 0.7000 - val_class_outpu
t_loss: 1.0034 - val_loss: 190.2016
Epoch 15/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 748ms/step - bbox_output_loss: 153.6552 - bbox_out
put_mean_squared_error: 153.6552 - class_output_accuracy: 0.8083 - class_output_l
oss: 0.8341 - loss: 154.4893 - val_bbox_output_loss: 164.9910 - val_bbox_output_m
ean_squared_error: 167.6374 - val_class_output_accuracy: 0.6500 - val_class_outpu
t_loss: 1.1355 - val_loss: 168.7809
Epoch 16/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 713ms/step - bbox_output_loss: 127.2298 - bbox_out
put_mean_squared_error: 127.2298 - class_output_accuracy: 0.7689 - class_output_l
oss: 0.9303 - loss: 128.1601 - val_bbox_output_loss: 191.0435 - val_bbox_output_m
ean_squared_error: 193.2492 - val_class_output_accuracy: 0.5500 - val_class_outpu
t_loss: 1.1285 - val_loss: 194.3859
Epoch 17/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 737ms/step - bbox_output_loss: 158.1259 - bbox_out
put_mean_squared_error: 158.1259 - class_output_accuracy: 0.7078 - class_output_l
oss: 1.1967 - loss: 159.3227 - val_bbox_output_loss: 155.7765 - val_bbox_output_m
ean_squared_error: 158.5487 - val_class_output_accuracy: 0.6667 - val_class_outpu
t_loss: 0.9405 - val_loss: 159.5021
Epoch 18/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 685ms/step - bbox_output_loss: 111.3629 - bbox_out
put_mean_squared_error: 111.3629 - class_output_accuracy: 0.8040 - class_output_l
oss: 0.9144 - loss: 112.2773 - val_bbox_output_loss: 179.1065 - val_bbox_output_m
ean_squared_error: 181.0321 - val_class_output_accuracy: 0.5667 - val_class_outpu
t_loss: 1.7631 - val_loss: 182.8045
Epoch 19/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 682ms/step - bbox_output_loss: 204.6862 - bbox_out
put_mean_squared_error: 204.6862 - class_output_accuracy: 0.7085 - class_output_l
oss: 1.0702 - loss: 205.7565 - val_bbox_output_loss: 194.4577 - val_bbox_output_m
ean_squared_error: 197.0762 - val_class_output_accuracy: 0.7500 - val_class_outpu
t_loss: 1.1219 - val_loss: 198.2133
Epoch 20/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 726ms/step - bbox_output_loss: 136.2652 - bbox_out
put_mean_squared_error: 136.2652 - class_output_accuracy: 0.7964 - class_output_l
oss: 1.0039 - loss: 137.2691 - val_bbox_output_loss: 160.5117 - val_bbox_output_m
ean_squared_error: 163.9299 - val_class_output_accuracy: 0.6833 - val_class_outpu
t_loss: 1.1420 - val_loss: 165.0791
```

```
Epoch 21/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 659ms/step - bbox_output_loss: 113.7127 - bbox_out
put_mean_squared_error: 113.7127 - class_output_accuracy: 0.8013 - class_output_l
oss: 0.8173 - loss: 114.5300 - val_bbox_output_loss: 150.3305 - val_bbox_output_m
ean_squared_error: 152.9409 - val_class_output_accuracy: 0.6667 - val_class_outpu
t_loss: 1.0492 - val_loss: 154.0171
Epoch 22/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 719ms/step - bbox_output_loss: 107.6781 - bbox_out
put_mean_squared_error: 107.6781 - class_output_accuracy: 0.8161 - class_output_l
oss: 0.7585 - loss: 108.4366 - val_bbox_output_loss: 143.8892 - val_bbox_output_m
ean_squared_error: 146.0589 - val_class_output_accuracy: 0.7333 - val_class_outpu
t_loss: 0.6720 - val_loss: 146.7471
Epoch 23/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 842ms/step - bbox_output_loss: 97.3185 - bbox_outp
ut_mean_squared_error: 97.3185 - class_output_accuracy: 0.8289 - class_output_los
s: 0.6205 - loss: 97.9390 - val_bbox_output_loss: 137.2159 - val_bbox_output_mean
_squared_error: 140.1492 - val_class_output_accuracy: 0.8167 - val_class_output_l
oss: 0.7031 - val_loss: 140.8755
Epoch 24/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 13s 883ms/step - bbox_output_loss: 82.6470 - bbox_outp
ut_mean_squared_error: 82.6470 - class_output_accuracy: 0.8365 - class_output_los
s: 0.5540 - loss: 83.2010 - val_bbox_output_loss: 192.7413 - val_bbox_output_mean
_squared_error: 195.3733 - val_class_output_accuracy: 0.6667 - val_class_output_l
oss: 0.8959 - val_loss: 196.2780
Epoch 25/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 630ms/step - bbox_output_loss: 91.6059 - bbox_outp
ut_mean_squared_error: 91.6059 - class_output_accuracy: 0.8592 - class_output_los
s: 0.5884 - loss: 92.1943 - val_bbox_output_loss: 144.9444 - val_bbox_output_mean
_squared_error: 147.6810 - val_class_output_accuracy: 0.8333 - val_class_output_l
oss: 0.6114 - val_loss: 148.3113
Epoch 26/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 766ms/step - bbox_output_loss: 84.2213 - bbox_outp
ut_mean_squared_error: 84.2213 - class_output_accuracy: 0.8554 - class_output_los
s: 0.5266 - loss: 84.7479 - val_bbox_output_loss: 136.4364 - val_bbox_output_mean
_squared_error: 139.2730 - val_class_output_accuracy: 0.6333 - val_class_output_l
oss: 0.8544 - val_loss: 140.1534
Epoch 27/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 674ms/step - bbox_output_loss: 78.6004 - bbox_outp
ut_mean_squared_error: 78.6004 - class_output_accuracy: 0.8625 - class_output_los
s: 0.4531 - loss: 79.0535 - val_bbox_output_loss: 123.0003 - val_bbox_output_mean
_squared_error: 125.3670 - val_class_output_accuracy: 0.7833 - val_class_output_l
oss: 0.5670 - val_loss: 125.9493
Epoch 28/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 646ms/step - bbox_output_loss: 66.8144 - bbox_outp
ut_mean_squared_error: 66.8144 - class_output_accuracy: 0.8810 - class_output_los
s: 0.4086 - loss: 67.2230 - val_bbox_output_loss: 129.4022 - val_bbox_output_mean
_squared_error: 132.2429 - val_class_output_accuracy: 0.6833 - val_class_output_l
oss: 0.7564 - val_loss: 133.0180
Epoch 29/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 722ms/step - bbox_output_loss: 71.3705 - bbox_outp
ut_mean_squared_error: 71.3705 - class_output_accuracy: 0.8594 - class_output_los
s: 0.4837 - loss: 71.8542 - val_bbox_output_loss: 131.6530 - val_bbox_output_mean
_squared_error: 134.4069 - val_class_output_accuracy: 0.8000 - val_class_output_l
oss: 0.5524 - val_loss: 134.9756
Epoch 30/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 726ms/step - bbox_output_loss: 61.1270 - bbox_outp
ut_mean_squared_error: 61.1270 - class_output_accuracy: 0.8841 - class_output_los
s: 0.3751 - loss: 61.5021 - val_bbox_output_loss: 124.6608 - val_bbox_output_mean
_squared_error: 127.3000 - val_class_output_accuracy: 0.7667 - val_class_output_l
oss: 0.6522 - val_loss: 127.9735
```

```
Epoch 31/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 668ms/step - bbox_output_loss: 61.4751 - bbox_outp
ut_mean_squared_error: 61.4751 - class_output_accuracy: 0.8695 - class_output_los
s: 0.3804 - loss: 61.8555 - val_bbox_output_loss: 126.6533 - val_bbox_output_mean
_squared_error: 128.8551 - val_class_output_accuracy: 0.8500 - val_class_output_l
oss: 0.4633 - val_loss: 129.3383
Epoch 32/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 12s 771ms/step - bbox_output_loss: 71.0085 - bbox_outp
ut_mean_squared_error: 71.0085 - class_output_accuracy: 0.9017 - class_output_los
s: 0.3875 - loss: 71.3961 - val_bbox_output_loss: 122.0856 - val_bbox_output_mean
_squared_error: 124.7403 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.5218 - val_loss: 125.2833
Epoch 33/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 667ms/step - bbox_output_loss: 65.2165 - bbox_outp
ut_mean_squared_error: 65.2165 - class_output_accuracy: 0.8286 - class_output_los
s: 0.5935 - loss: 65.8100 - val_bbox_output_loss: 137.6354 - val_bbox_output_mean
_squared_error: 140.3071 - val_class_output_accuracy: 0.7833 - val_class_output_l
oss: 0.7318 - val_loss: 141.0610
Epoch 34/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 9s 634ms/step - bbox_output_loss: 57.0305 - bbox_outpu
t_mean_squared_error: 57.0305 - class_output_accuracy: 0.8759 - class_output_los
s: 0.5014 - loss: 57.5319 - val_bbox_output_loss: 119.3592 - val_bbox_output_mean
_squared_error: 121.2533 - val_class_output_accuracy: 0.8167 - val_class_output_l
oss: 0.4253 - val_loss: 121.6874
Epoch 35/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 688ms/step - bbox_output_loss: 47.9389 - bbox_outp
ut_mean_squared_error: 47.9389 - class_output_accuracy: 0.8876 - class_output_los
s: 0.3724 - loss: 48.3113 - val_bbox_output_loss: 115.1462 - val_bbox_output_mean
_squared_error: 117.5882 - val_class_output_accuracy: 0.7667 - val_class_output_l
oss: 0.6259 - val_loss: 118.2352
Epoch 36/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 657ms/step - bbox_output_loss: 47.0359 - bbox_outp
ut_mean_squared_error: 47.0359 - class_output_accuracy: 0.9210 - class_output_los
s: 0.2932 - loss: 47.3291 - val_bbox_output_loss: 126.2548 - val_bbox_output_mean
_squared_error: 129.2766 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.4095 - val_loss: 129.7007
Epoch 37/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 666ms/step - bbox_output_loss: 49.8117 - bbox_outp
ut_mean_squared_error: 49.8117 - class_output_accuracy: 0.9023 - class_output_los
s: 0.3248 - loss: 50.1365 - val_bbox_output_loss: 129.5103 - val_bbox_output_mean
_squared_error: 130.8223 - val_class_output_accuracy: 0.8333 - val_class_output_l
oss: 0.4104 - val_loss: 131.2437
Epoch 38/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 664ms/step - bbox_output_loss: 53.2712 - bbox_outp
ut_mean_squared_error: 53.2712 - class_output_accuracy: 0.9126 - class_output_los
s: 0.3336 - loss: 53.6048 - val_bbox_output_loss: 126.2213 - val_bbox_output_mean
_squared_error: 128.2318 - val_class_output_accuracy: 0.8500 - val_class_output_l
oss: 0.4025 - val_loss: 128.6460
Epoch 39/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 9s 631ms/step - bbox_output_loss: 45.0408 - bbox_outpu
t_mean_squared_error: 45.0408 - class_output_accuracy: 0.9043 - class_output_los
s: 0.2797 - loss: 45.3205 - val_bbox_output_loss: 131.4957 - val_bbox_output_mean
_squared_error: 133.4660 - val_class_output_accuracy: 0.7833 - val_class_output_l
oss: 0.5096 - val_loss: 133.9927
Epoch 40/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 648ms/step - bbox_output_loss: 41.1916 - bbox_outp
ut_mean_squared_error: 41.1916 - class_output_accuracy: 0.9026 - class_output_los
s: 0.2994 - loss: 41.4910 - val_bbox_output_loss: 110.4849 - val_bbox_output_mean
_squared_error: 112.6686 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.3489 - val_loss: 113.0276
```

```
Epoch 41/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 14s 939ms/step - bbox_output_loss: 36.2059 - bbox_outp
ut_mean_squared_error: 36.2059 - class_output_accuracy: 0.9165 - class_output_los
s: 0.2410 - loss: 36.4470 - val_bbox_output_loss: 134.6227 - val_bbox_output_mean
_squared_error: 136.9628 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.4925 - val_loss: 137.4757
Epoch 42/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 738ms/step - bbox_output_loss: 31.8737 - bbox_outp
ut_mean_squared_error: 31.8737 - class_output_accuracy: 0.8873 - class_output_los
s: 0.3390 - loss: 32.2127 - val_bbox_output_loss: 119.5234 - val_bbox_output_mean
_squared_error: 121.7450 - val_class_output_accuracy: 0.8167 - val_class_output_l
oss: 0.4280 - val_loss: 122.1825
Epoch 43/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 15s 1s/step - bbox_output_loss: 34.3518 - bbox_output_
mean_squared_error: 34.3518 - class_output_accuracy: 0.9178 - class_output_loss:
0.2701 - loss: 34.6219 - val_bbox_output_loss: 102.2769 - val_bbox_output_mean_sq
uared_error: 104.0714 - val_class_output_accuracy: 0.8500 - val_class_output_los
s: 0.4231 - val_loss: 104.5116
Epoch 44/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - bbox_output_loss: 37.4344 - bbox_output_
mean_squared_error: 37.4344 - class_output_accuracy: 0.9182 - class_output_loss:
0.2554 - loss: 37.6897 - val_bbox_output_loss: 120.8413 - val_bbox_output_mean_sq
uared_error: 122.6126 - val_class_output_accuracy: 0.9000 - val_class_output_los
s: 0.2866 - val_loss: 122.9072
Epoch 45/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 11s 697ms/step - bbox_output_loss: 31.4869 - bbox_outp
ut_mean_squared_error: 31.4869 - class_output_accuracy: 0.9344 - class_output_los
s: 0.2544 - loss: 31.7413 - val_bbox_output_loss: 106.8804 - val_bbox_output_mean
_squared_error: 108.7897 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.3304 - val_loss: 109.1345
Epoch 46/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 693ms/step - bbox_output_loss: 26.7571 - bbox_outp
ut_mean_squared_error: 26.7571 - class_output_accuracy: 0.9322 - class_output_los
s: 0.2076 - loss: 26.9647 - val_bbox_output_loss: 105.2763 - val_bbox_output_mean
_squared_error: 107.2338 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.3707 - val_loss: 107.6166
Epoch 47/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 693ms/step - bbox_output_loss: 27.6417 - bbox_outp
ut_mean_squared_error: 27.6417 - class_output_accuracy: 0.9306 - class_output_los
s: 0.1950 - loss: 27.8367 - val_bbox_output_loss: 106.5825 - val_bbox_output_mean
_squared_error: 108.1990 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.2766 - val_loss: 108.4847
Epoch 48/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 693ms/step - bbox_output_loss: 33.9391 - bbox_outp
ut_mean_squared_error: 33.9391 - class_output_accuracy: 0.9405 - class_output_los
s: 0.1934 - loss: 34.1326 - val_bbox_output_loss: 119.1563 - val_bbox_output_mean
_squared_error: 120.8843 - val_class_output_accuracy: 0.8833 - val_class_output_l
oss: 0.2576 - val_loss: 121.1539
Epoch 49/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 678ms/step - bbox_output_loss: 41.6245 - bbox_outp
ut_mean_squared_error: 41.6245 - class_output_accuracy: 0.9548 - class_output_los
s: 0.1726 - loss: 41.7972 - val_bbox_output_loss: 109.0860 - val_bbox_output_mean
_squared_error: 110.8908 - val_class_output_accuracy: 0.8667 - val_class_output_l
oss: 0.3463 - val_loss: 111.2523
Epoch 50/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 693ms/step - bbox_output_loss: 41.5185 - bbox_outp
ut_mean_squared_error: 41.5185 - class_output_accuracy: 0.9379 - class_output_los
s: 0.2125 - loss: 41.7310 - val_bbox_output_loss: 106.8088 - val_bbox_output_mean
_squared_error: 108.2408 - val_class_output_accuracy: 0.8833 - val_class_output_l
oss: 0.3174 - val_loss: 108.5692
```

Agora, avaliamos o modelo com imagens não vistas na etapa de treinamento

In [13]:
```
model.evaluate(
    x_test,
    {'bbox_output': y_test_bbox, 'class_output': y_test_classes}
)
```

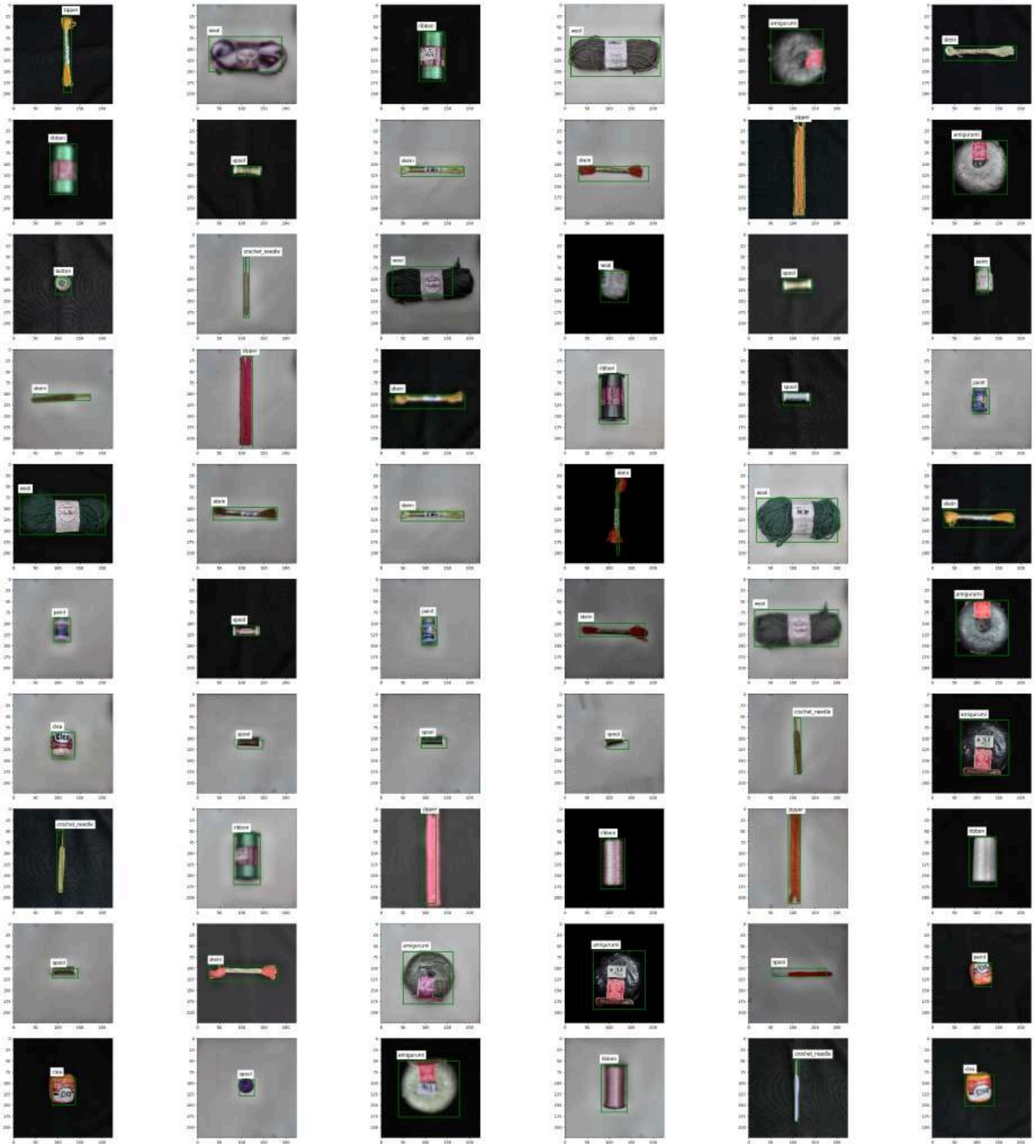
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 191ms/step - bbox_output_loss: 76.7964 - bbox_output_
mean_squared_error: 78.0613 - class_output_accuracy: 0.9021 - class_output_loss:
0.3149 - loss: 78.3751
```

Out[13]:
```
[69.52832794189453,
 0.3228673040866852,
 67.3097152709961,
 69.2070541381836,
 0.8999999761581421]
```

Adicionalmente, podemos utilizar esses mesmos dados para visualizar a eficiência do modelo treinado.

In [14]:
```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

predicted_classes, predicted_bboxes = model.predict(x_test)
predicted_classes = np.argmax(predicted_classes, axis=1) + 1

true_classes = np.argmax(y_test_classes, axis=1) + 1
class_labels = new_dataset.categories

fig, axes = plt.subplots(10, 6, figsize=(40, 40))
for i, ax in enumerate(axes.flat):
    image = x_test[i]

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

    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] - 10.0, f'{class_labels[predicted_class]}', fontsiz

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

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 248ms/step
```

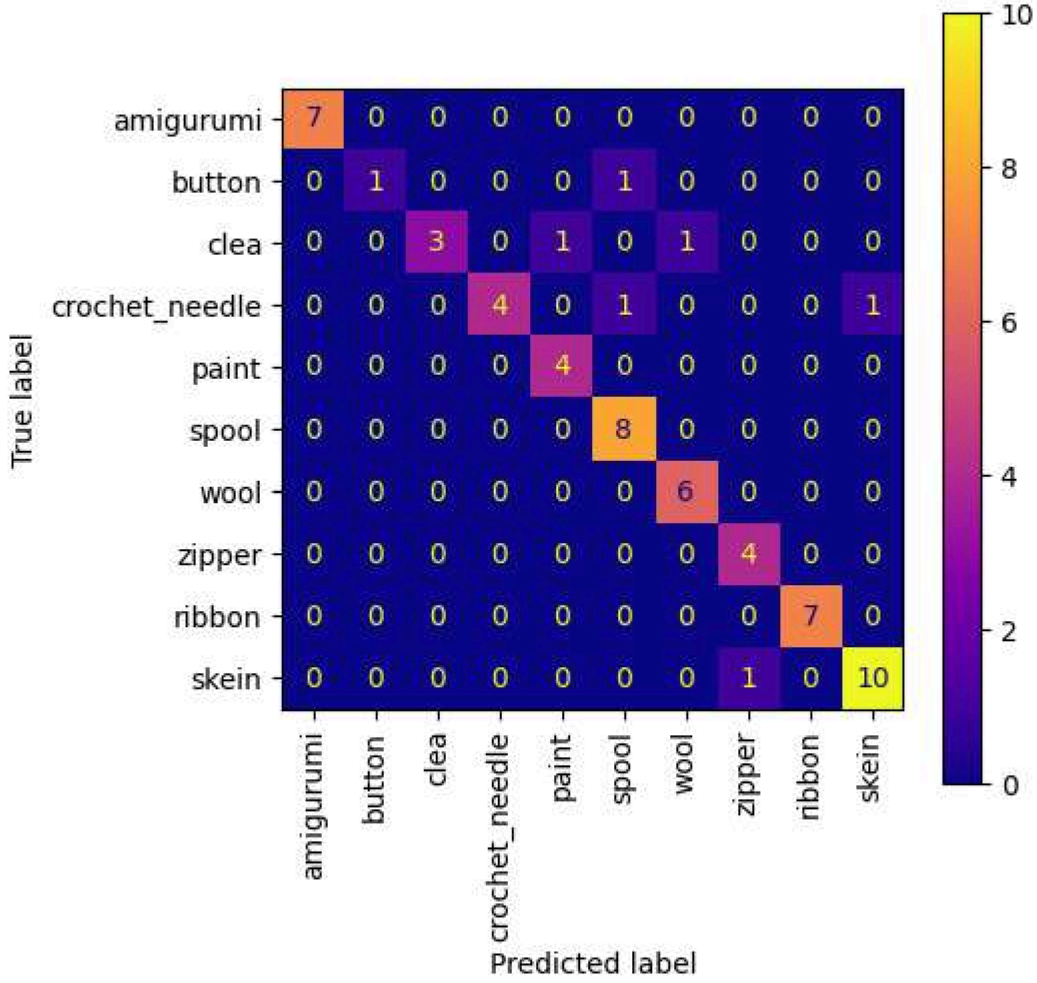


Abaixo estão as métricas acurácia média e F1-Score, além da matriz de confusão do modelo.

```
In [15]: from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, Confusic

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

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



         cm = confusion_matrix(true_classes, predicted_classes)

         fig, ax = plt.subplots(figsize=(5, 5))

         categorias = list(new_dataset.categories.values())
         ConfusionMatrixDisplay.from_predictions(
             true_classes, predicted_classes, display_labels=categorias, xticks_rotation=
             ax=ax, colorbar=True, cmap="plasma")
```

```
plt.show()
```

```
Acuracia: 0.9000
F1-Score: 0.8941
```



True label	amigurumi	button	clea	crochet_needle	paint	spool	wool	zipper	ribbon	skein
amigurumi	7	0	0	0	0	0	0	0	0	0
button	0	1	0	0	0	1	0	0	0	0
clea	0	0	3	0	1	0	1	0	0	0
crochet_needle	0	0	0	4	0	1	0	0	0	1
paint	0	0	0	0	4	0	0	0	0	0
spool	0	0	0	0	0	8	0	0	0	0
wool	0	0	0	0	0	0	6	0	0	0
zipper	0	0	0	0	0	0	0	4	0	0
ribbon	0	0	0	0	0	0	0	0	7	0
skein	0	0	0	0	0	0	0	1	0	10

Se os resultados são satisfatórios, salvamos o modelo na pasta models.

```
In [16]: import os

         model.save(os.path.join('models', 'cnn_bounding_box_model.keras'))
```