Report

Digital Business Innovation

Course: Information Systems 353

Instructor: Ruchen Wyngaard

Due Date: 13 October 2024

Student: Vuyo Kirabira, 4253908

Contents

Overview 3

Goals 3

Data collection and pre-processing 3

Model development 6

Training the model 7

Performance Evaluation 7

Ethical considerations 8

Results 12

Conclusion 12

References 12

Appendices 13

Overview

This report outlines the development of a machine learning-based vision system aimed at solving a classification problem involving the distinction between metal cans and plastic bottles. This project includes the entire machine learning pipeline, from data collection and pre-processing to model training, evaluation, and ethical considerations.

Goals

The following goals were fulfilled by finishing this project:

Gathered and prepared image data in advance of machine learning. created and trained an image classification machine learning model. assessed the vision system's performance. overcome difficulties in practical computer vision applications.

Data collection and pre-processing

This part of the assignment consisted of collecting a dataset from Kaggle I also added a few of my own pictures just to increase the accuracy of the model. I decided to go with a 80% to 20% spilt of the data samples. I choose this because it helps with overfitting its an optimal split for training and validating the model. In summary data splitting equals preventing overfitting and increases generalizations. (datasciencewizards, 2023)

Waste Segregation Image Dataset

Segregate waste into Biodegradable and Non-Biodegradable

Data Card | Code (2) | Discussion (1) | Suggestions (0)

About Dataset

This Image dataset has been divided into train and val folders and into Biodegradable and non-biodegradabl

Each category has 4 classes. For Biodegradable class:

paper, leaf, food, wood

For Non-Biodegradable class:

waste, plastic bags, plastic bottles, metal cans

Resizing:images were resized to224x224 pixels to ensure uniformity.

Normalization:Images were normalized using image net.

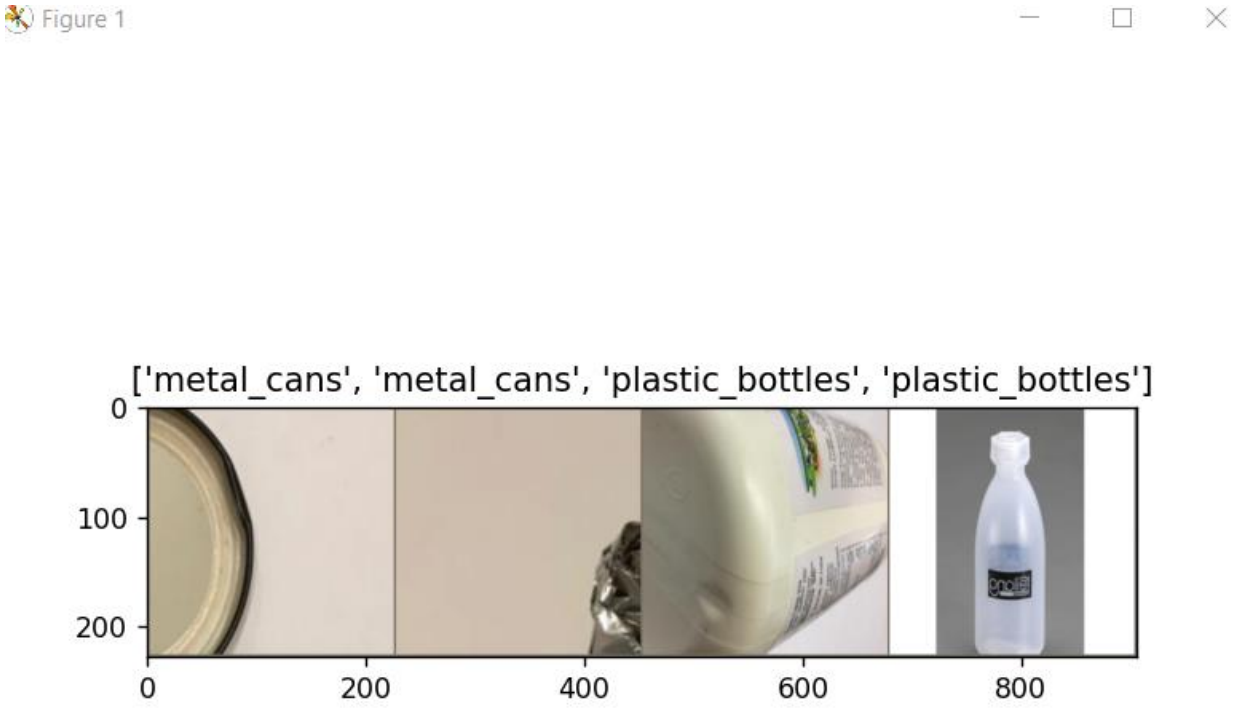
Data visualization

I used matplotlib to create visualization.

For example:

The “imshow function” is used to display a grid from both the training and validation datasets. This confirms that the images I used have been loaded and pre-processed

correctly.providing a quick overview of the datastets diversity and quality



“visualize_model” function, images are displayed alongside my models predicted class labels(metal_cans and plastic_bottles)this is crucial for assessing the models performance visually as it allows you to see how well the model is doing classifying images and whether predictions align with the expected classes

Model development

I used Resnet because I did my research and found that it was a well known and widely used convolutional neural network . (Ahmed, 2022) I also watched a youtube video which explained how to create a image classification model using pytorch. (Aarohi, 2023)

1.Data augmentation: In this step the code I wrote randomly resized and flipped images to improve the models robustness and adaptability.

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize( mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize( mean: [0.485, 0.456, .406], std: [0.229, 0.224, 0.225])
    ]),
}
```

2.Transferal of learning(ml model): In this step I used a pretrained model ResNet18 and adjusted it to suit the final layer to out predictions of my classes (metal cans and plastic bottles)

```
# Load the resnet model
model_ft = models.resnet18(pretrained=True)
```

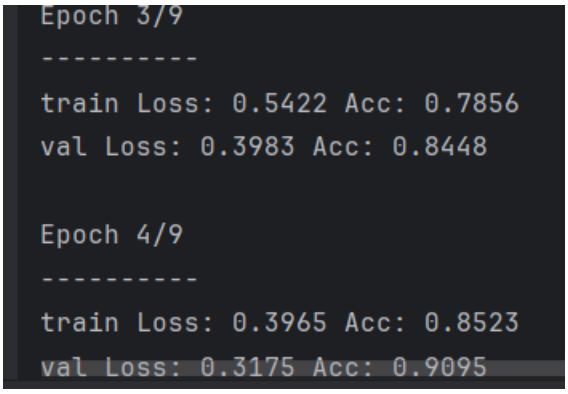
Training the model

My model was trained for 10 epochs at first. I decided to choose 4 epochs because at this number of epochs because I saw my model performs best in terms of predictions on the 4th epoch.

```
Training complete in 78m 37s
Best val Acc: 0.909483
```

Performance Evaluation

The models performance was evaluated based on accuracy and loss metrics. Here it is attached below. We can see my vision system has low losses and high gains indicated it is working well and is make great predictions.



Ethical considerations

This part of the report refers to the ethical implication of deploying my vision system which I addressed.

Bias mitigation:

I used a diverse dataset including metal can and plastic bottle images from various brands and conditions(e.g angles, lighting and backgrounds) to ensure my vision system is not biased and can distinguish between the 2 in a variety of situations.

Images serving as proof of my diverse dataset.


500ml

Data augmentation I used techniques such as random cropping,flipping during training to ensure that the model can generalize well and reduces the likelihood of being biased.

Data augmentation.

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize( mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize( mean: [0.485, 0.456, .406], std: [0.229, 0.224, 0.225])
    ]),
}
```

The final method I use to mitigate was the continuous monitoring of my model by checking my outputs(losses and accuracy) this allows me to identify any biases in my outputs.

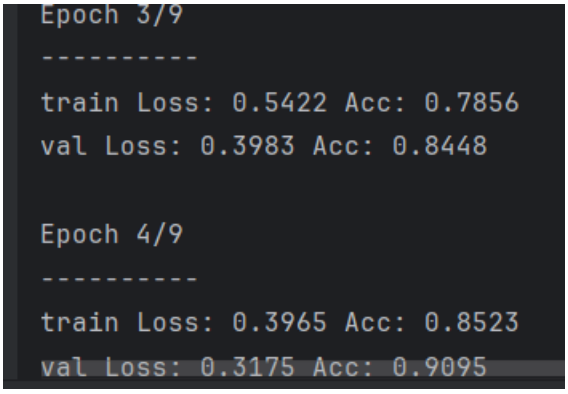
```
Epoch 3/9
----------
train Loss: 0.5422 Acc: 0.7856
val Loss: 0.3983 Acc: 0.8448

Epoch 4/9
----------
train Loss: 0.3965 Acc: 0.8523
val Loss: 0.3175 Acc: 0.9095
```

Keeping track of bias mitigation builds transparency to stakeholders and helps to build trust in in AI systems so in Future I would like to create visualization which give insights on the training loss and accuracy , this is a consideration I have for the future.

Privacy: Privacy as a ethical consideration I ensured that nobodies privacy was violated in the use of the dataset from Kaggle.

Results

The training process demonstrated a consistent improvement in the accuracy across epochs showing that my model is learning and growing this shows us AI is able to experience growth in a human sense(interesting concepts speaks to the morality of AI) as the model gets more accurate after each epoch(also known as training) by best validation accuracy was noted showing my models ability to generalize well.



Conclusion

My system was able to effectively distinguish between metal cans and plastic bottles. The project successfully met its objectives and provided insights into the practical application of computer vision and machine learning techniques. Future work may involve the deployment of the model in real world setting for example the industry of waste management where it can be used to automate the process of waste management. I believe my projected is highly scalable because it is well commented so adjustments can be made easily such as adding more classes for the model to identify e.g e-waste, napkins.

References

Aarohi, C. W. (2023, October 13). *image classifaction using pytorch and cnn*. Retrieved from youtube : https://youtu.be/cJpwQJp9flU?si=jkXu02jgURTcaDWv

Ahmed, R. (2022). *What is ResNet? (with 3D Visualizations)*. Retrieved from Youtube: https://youtu.be/nc7FzLiB_AY?si=SCbldsRzA6Ycj-ri

Appendices

Appendix A: Source code

My main code for train the recycle project model.

```
import numpy as np

import matplotlib.pyplot as plt
import cv2
import time
import os
import copy

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn

import torchvision
from torchvision import datasets, models, transforms



# Define data transformations for training and validation datasets
data_transforms = {
  'train': transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
  ]),
  'val': transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, .406], [0.229, 0.224, 0.225])
  ]),
}
```

```
#creates a path for the dataset directory
data_direct = r'individual assignment 80%/dataset'
# Directory containing the dataset
image_datasets = { x: datasets.ImageFolder(os.path.join(data_direct, x),
                                           data_transforms[x])

                  for x in['train','val']}

#obtain datasizes
dataset_sizes = {x: len(image_datasets[x]) for x in ['train','val']}

class_names = image_datasets['train'].classes

#use image_datasets to sample from the dataset

dataloaders = { x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                               shuffle=True)
  for x in ['train', 'val']}

# Change selected device to CUDA, a parallel processing platform, if available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")



def imshow(inp, title=None):
  """
  This function will make use of Matplotlib.pyplot's imshow() function for tensors.
  It will show the same number of images as the batch we defined.
  """
  # The transpose is required to get the images into the correct shape
  inp = inp.numpy().transpose((1, 2, 0))

  # Using default values for mean and std but can customize
  mean = np.array([0.485, 0.456, 0.406])
  std = np.array([0.229, 0.224, 0.225])

  # To visualize the correct colors
  inp = std * inp + mean
```

```
    # To view a clipped version of an image
    inp = np.clip(inp, 0, 1)

    # Visualize inp
    plt.imshow(inp)

    if title is not None:  # Plot title goes here
        plt.title(title)
    plt.pause(0.001)  # Enables the function to pause while the plots are updated

# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

# Plot the grid with a title that concatenates all the class labels
imshow(out, title=[class_names[x] for x in classes])

# Get a batch of test data
inputs, classes = next(iter(dataloaders['val']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

# Plot the grid with a title that concatenates all the class labels
imshow(out, title=[class_names[x] for x in classes])


def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    '''
    Function that will train model based on data provided.
    '''

    since = time.time()

    # Make a deep copy of the model provided
```

```
best_model_wts = copy.deepcopy(model.state_dict())
best_acc = 0.0

for epoch in range(num_epochs):
  print(f'Epoch {epoch}/{num_epochs - 1}')
  print('-' * 10)

  # Each epoch has a training and validation phase
  for phase in ['train', 'val']:
    if phase == 'train':
      model.train()  # Set model to training mode
    else:
      model.eval()  # Set model to evaluate mode

    running_loss = 0.0
    running_corrects = 0

    # Iterate over data using the dataloader we defined
    for inputs, labels in dataloaders[phase]:
      inputs = inputs.to(device)
      labels = labels.to(device)

      # Zero the parameter gradients
      optimizer.zero_grad()

      # Forward pass, tracking history if only in train
      with torch.set_grad_enabled(phase == 'train'):
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        loss = criterion(outputs, labels)

        # Backward pass and optimization only if in training phase
        if phase == 'train':
          loss.backward()
          optimizer.step()

      # Computing loss statistics
      running_loss += loss.item() * inputs.size(0)
```

```
            running_corrects += torch.sum(preds == labels.data)
        if phase == 'train':
            scheduler.step()

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

        # Create a deep copy of the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print()  # Print an empty line for nice formatting

  time_elapsed = time.time() - since
  print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
  print(f'Best val Acc: {best_acc:4f}')

  # Load the best model weights
  model.load_state_dict(best_model_wts)
  return model

def visualize_model(model, num_images=6):
  '''
  Function that will visualize results of the model
  '''
  was_training = model.training
  model.eval()
  images_so_far = 0
  fig = plt.figure()

  with torch.no_grad():
    for i, (inputs, labels) in enumerate(dataloaders['val']):
      inputs = inputs.to(device)
      labels = labels.to(device)
```

```
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)

# Load the resnet model
model_ft = models.resnet18(pretrained=True)

# Obtaining the number of input features for our final layer
num_ftrs = model_ft.fc.in_features

# Since this is a binary classification task, we'll set the size of each output sample to 2. For
multi-class classification, this can be generalized to nn.Linear(num_ftrs,
len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

# Move the model to the device
model_ft = model_ft.to(device)

# We'll use CrossEntropyLoss(), which is a common loss function for classification
problems
criterion = nn.CrossEntropyLoss()

# In this step, we'll optimize all parameters of the model
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# We'll decay learning rate (lr) by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

```
# Call our train_model() function with the ResNet model, the criterion, optimizer, learning
rate scheduler, and number of epochs that we have defined.
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=4)

visualize_model(model_ft)

# Disable gradients for model_conv.parameters()
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
   param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

# Move the model to the device
model_conv = model_conv.to(device)

# Set criterion again
criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as opposed to before
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)

# Train model_conv
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=4)

# Visualize model
visualize_model(model_conv)
plt.show()

# After training your model
```

torch.save(model_ft.state_dict(), r"C:\Users\User\PycharmProjects\Recycling project\model.pth")

my code for camera detection

```
import torch
import torchvision.models as models
import cv2
import numpy as np
import torch.nn as nn
from torchvision import transforms

# Load the trained model
def load_model():
    model = models.resnet18(pretrained=False)
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, 2)  # Change 2 to your number of classes
    model.load_state_dict(torch.load('model.pth'))
    model.eval()
    return model

# Set up camera and transformations
def setup_camera():
    cap = cv2.VideoCapture(0)  # Use 0 for the default camera
    return cap

def main():
    # Load the model
    model = load_model()

    # Initialize camera
    cap = setup_camera()

    # Define the transformation for input images
    data_transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(256),
        transforms.CenterCrop(224),
```

```
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
  ])

  class_names = ['metal_cans', 'plastic_bottles']

  while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    if not ret:
      break

    # Prepare the image for the model
    input_tensor = data_transform(frame)
    input_batch = input_tensor.unsqueeze(0).to('cuda' if torch.cuda.is_available() else
'cpu')

    # Perform inference
    with torch.no_grad():
      output = model(input_batch)
      _, preds = torch.max(output, 1)

    # Display the prediction
    label = class_names[preds.item()]  # Get the class name
    cv2.putText(frame, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

    # Show the frame with the prediction
    cv2.imshow('Frame', frame)

    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
      break

  # Release the camera and close windows
  cap.release()
  cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    main()
```

Appendix c : Dataset visualizations

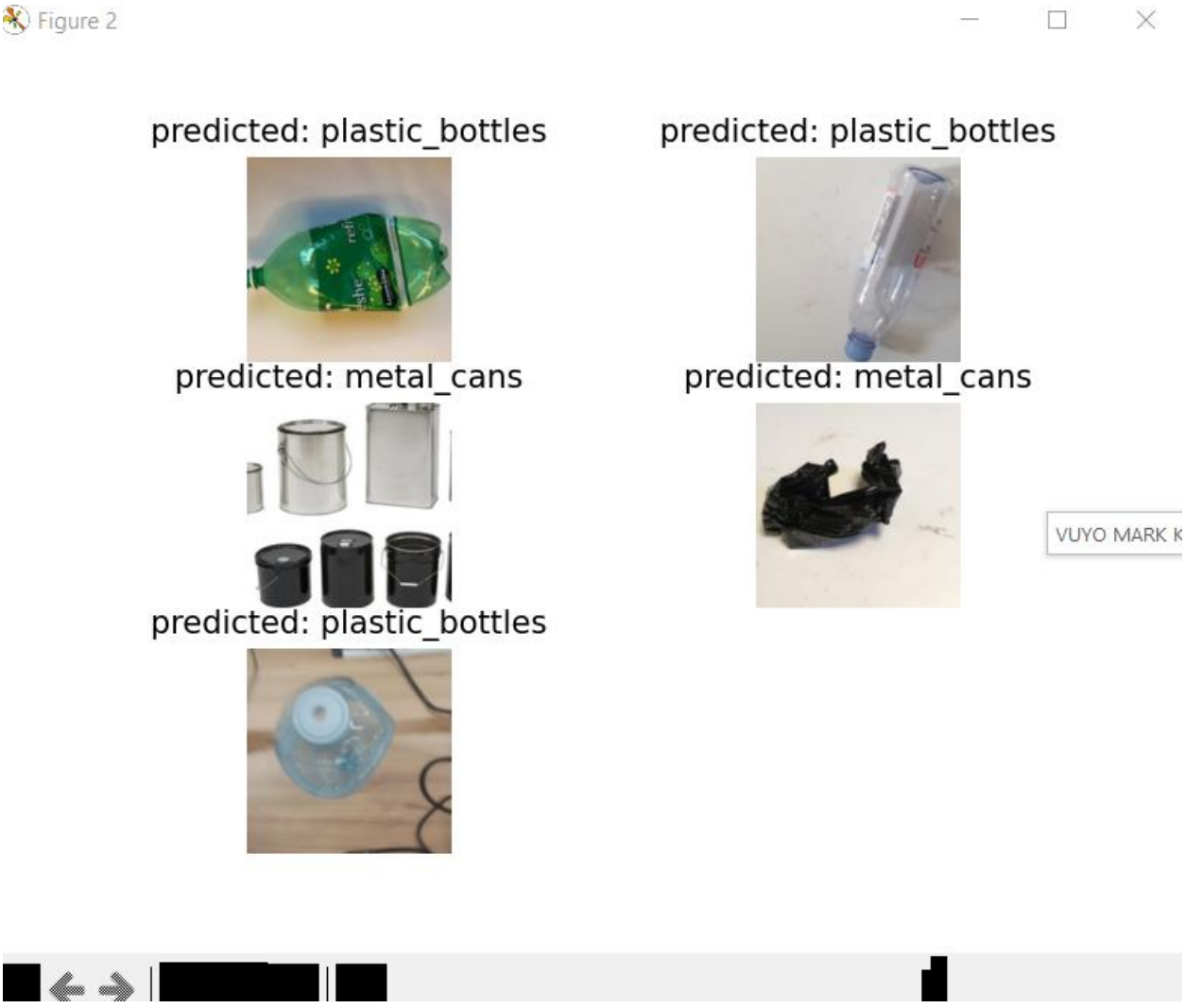
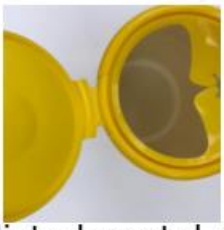


Figure 2

predicted: plastic_bottles

predicted: plastic_bottles

predicted: metal_cans



predicted: plastic_bottles

predicted: metal_cans