

# Bottles Synthetic Images



\

Neural Computing 2022/2023

Giuseppe Pulino



Università  
di Catania

# Contents

<b>Data Exploration</b>	<b>3</b>
<b>Problem Designing</b>	<b>4</b>
<b>Choosing the Neural Network</b>	<b>5</b>
<b>ResNet Architecture</b>	<b>6</b>
<b>ResNet-18 in details</b>	<b>7</b>
First Group . . . . .	7
Second, Third, Fourth, Fifth Group . . . . .	7
Sixth Group . . . . .	8
<b>Data Processing</b>	<b>8</b>
<b>Training ResNet-18</b>	<b>9</b>
<b>Model performance evaluation on Validation set</b>	<b>10</b>
<b>Model performance evaluation on Test set</b>	<b>11</b>
<b>Main misclassification errors</b>	<b>12</b>
Misclassified Beer Bottles . . . . .	12
Misclassified Plastic Bottles . . . . .	13
Misclassified Soda Bottles . . . . .	14
Misclassified Wine Bottles . . . . .	15
Misclassified Water Bottles . . . . .	16

## List of Figures

1	comparing the ImageNet validation accuracy of each net . . . . .	4
2	ImageNet Large Scale Visual Recognition Challenge (ILSVRC) . . . . .	5
3	Description of each net . . . . .	5
4	ResNet-18 . . . . .	6
5	ResNet-Architectures . . . . .	6
6	First Group . . . . .	7
7	Second, Third, Fourth, Fifth Group . . . . .	7
8	Sixth Group . . . . .	8
9	Training chart . . . . .	9

10	Confusion Matrix on Validation Set . . . . .	10
11	Confusion Matrix on Test Set . . . . .	11
12	Misclassified Beer Bottles . . . . .	12
13	Misclassified Plastic Bottles . . . . .	13
14	Misclassified Soda Bottles . . . . .	14
15	Misclassified Wine Bottles . . . . .	15
16	Misclassified Water Bottles . . . . .	16

# Data Exploration

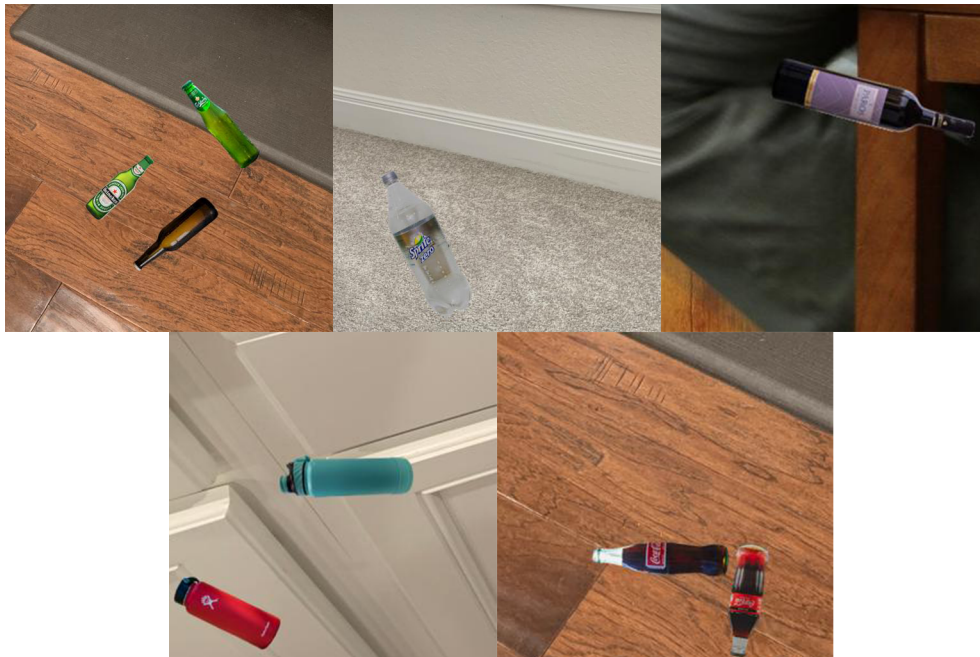
## Link to the Dataset

The dataset contains synthetically generated images of bottles scattered around random backgrounds. The main folder contains 25000 Images divided in the following way:

Categories	Number Images	Resolution	Extension
Soda Bottle	5000	512 X 512 RGB	.JPG
Water Bottle	500		
Plastic Bottle	5000		
Wine Bottle	5000		
Beer Bottle	5000		

Let's see an example of each class.

## Example categories



As we can see in the example, images can contain more than one bottles but always of the same type as the creator of the dataset reported.

## Problem Designing

The problem that we are designing is a **classification** problem, the environment that i will use is MatLab. The goal is to classify an image containing a bottle among the following classes of bottles:

- Soda
- Beer
- Wine
- Plastic
- Water

To perform this task we could train one of the most famous nets from scratch but this would require too much time especially with a huge network and a weak computational power like in my case, furthermore a huge network with million parameters requires millions of images to avoid overfitting, so even if that method remains the best because training from scratch a net allow us to extract better features I can't use that method for the following reasons:

- weak computational power
- a few number of images compared with size of the net

Hence in this case, i will use transfer learning technique starting from a pre-trained-model among those are available in MatLab and retraining the net with new output starting from the weights of the pre-trained model. All the nets have been pre-trained on ImageNet dataset containing 1.2 millions images divided in 1000 categories. We can see below a comparison chart among the available nets in MatLab

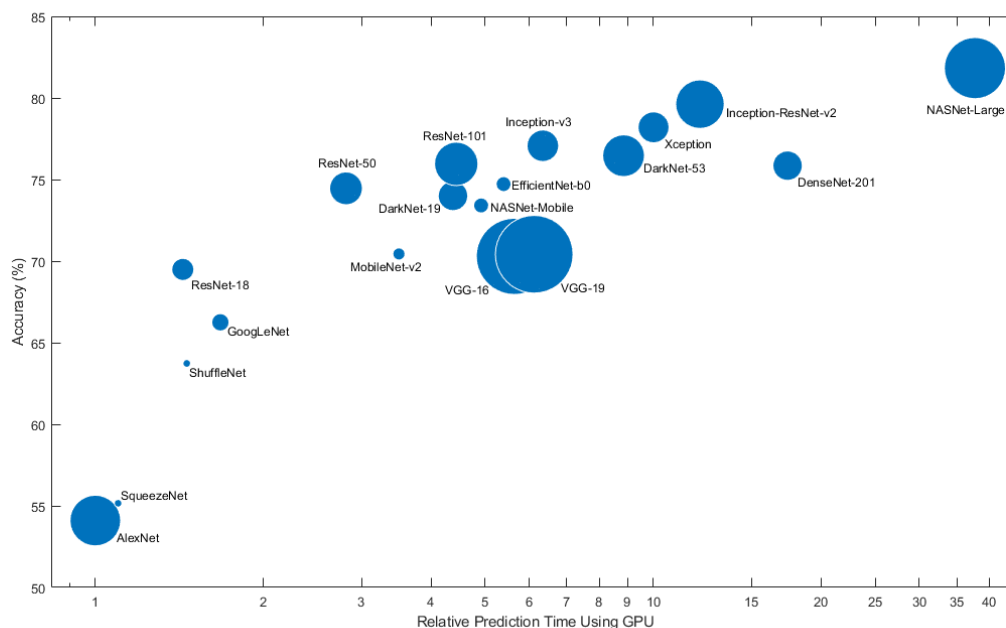


Figure 1: comparing the ImageNet validation accuracy of each net

## Choosing the Neural Network

From the Figure 1 we can see that we have a lot of available nets, so which is the best net for our task?

Since the task is not so hard, different nets can be good to solve this task, so I decided to have a look to the winning nets of ImageNet Large Scale Visual Recognition Challenge (ILSVRC)(Figure 3).ILSVRC evaluates algorithms for object detection and image classification at large scale,it has been held from 2010 to 2017 and then has been stopped since it has been reached an high level of accuracy very close to 98-99%.Hence, looking at the performance of the nets in the challenge I decided to choose **ResNet** architecture but since the net has 152 layers with more then 50 millions parameters i chose the lighter version called ResNet-18 with only 18 layers and 11.7 millions parameters as we can from Figure 4

### Top-5 error

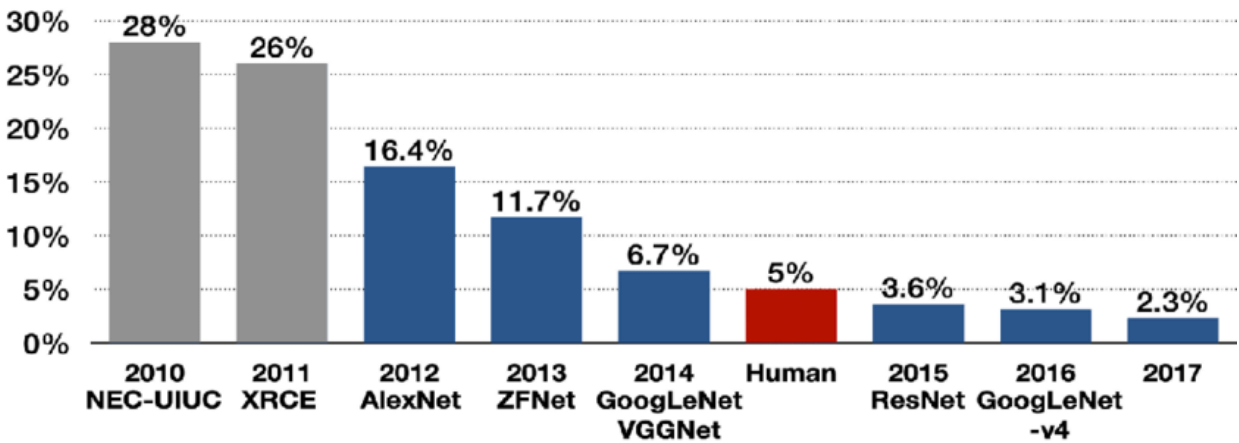


Figure 2: ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Network	Depth	Size	Parameters (Millions)	Image Input Size
<a href="#">squeezezenet</a>	18	5.2 MB	1.24	227-by-227
<a href="#">googlenet</a>	22	27 MB	7.0	224-by-224
<a href="#">inceptionv3</a>	48	89 MB	23.9	299-by-299
<a href="#">densenet201</a>	201	77 MB	20.0	224-by-224
<a href="#">mobilenetv2</a>	53	13 MB	3.5	224-by-224
<a href="#">resnet18</a>	18	44 MB	11.7	224-by-224
<a href="#">resnet50</a>	50	96 MB	25.6	224-by-224
<a href="#">resnet101</a>	101	167 MB	44.6	224-by-224
<a href="#">xception</a>	71	85 MB	22.9	299-by-299
<a href="#">inceptionresnetv2</a>	164	209 MB	55.9	299-by-299
<a href="#">shufflenet</a>	50	5.4 MB	1.4	224-by-224
<a href="#">nasnetmobile</a>	*	20 MB	5.3	224-by-224
<a href="#">nasnetlarge</a>	*	332 MB	88.9	331-by-331
<a href="#">darknet19</a>	19	78 MB	20.8	256-by-256
<a href="#">darknet53</a>	53	155 MB	41.6	256-by-256
<a href="#">efficientnetb0</a>	82	20 MB	5.3	224-by-224
<a href="#">alexnet</a>	8	227 MB	61.0	227-by-227
<a href="#">vgg16</a>	16	515 MB	138	224-by-224
<a href="#">vgg19</a>	19	535 MB	144	224-by-224

Figure 3: Description of each net

# ResNet Architecture

## What problem does ResNet solve?

Deep networks are hard to train because of the notorious vanishing gradient problem. As the gradient is backpropagated to earlier layers, repeated multiplication may make the gradient infinitely small with the network depth increasing, accuracy gets saturated and then degrades rapidly. So using deeper networks is degrading the performance of the model. ResNet architecture tries to solve this problem using Deep Residual learning framework. ResNet makes it possible to train up to hundreds or even thousands of layers and still achieve a compelling performance. Before ResNet the other architectures used less than 30 layers, the VGG network and GoogleNet (also codenamed Inception\_v1) had 19 and 22 layers respectively, instead when ResNet won the ILSVRC it had 152 layers.

## Solution:

ResNet solve the famous known vanishing gradient problem. The core idea of ResNet is that it introduced a so-called “identity shortcut connection” that skips one or more layers. With ResNet, the gradients can flow directly through the skip connections backwards from later layers to initial filters.

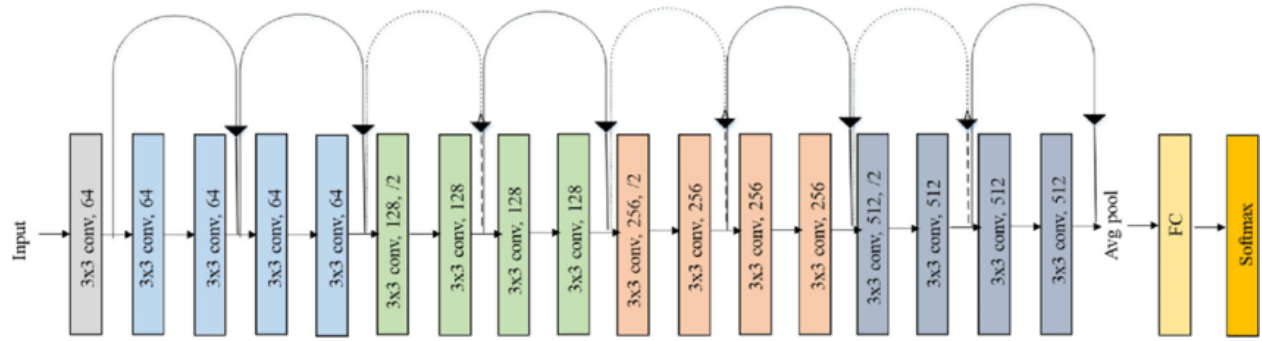


Figure 4: ResNet-18

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Figure 5: ResNet-Architectures

## ResNet-18 in details

ResNet as showed in figure 6 is composed by 18 layers grouped in 6 main groups with 11.7 millions parameters

### First Group

#### Input Layer

Input layer takes in input an RGB image of 224-by-224 pixels

#### Conv1

Is a 2D convolutional layer with :

- kernel size 7 by 7
- stride 2 by 2
- padding 3 by 3

#### Batch Normalization

Normalize data before ReLU

#### ReLU activation function

Introduce non linearity

#### Max Pooling

Reduce size feature maps



Figure 6: First Group

### Second, Third, Fourth, Fifth Group

These groups share the same pattern, they are composed by two branches (Figure 7):

- One containing the shortcut connection
- Another one containing two convolutional layer, two batch normalization and a relu function

The settings for stride, kernel size and padding remaining the same for each group, furthermore the configuration explained above is iterated two times for each group. The only things that is different for each group is the number of filters of the convolutional layers, in particular for the :

- **Second Group:** there are 64 filters
- **Third Group:** there are 128 filters
- **Fourth Group:** there are 256 filters
- **Fifth Group:** there are 512 filters

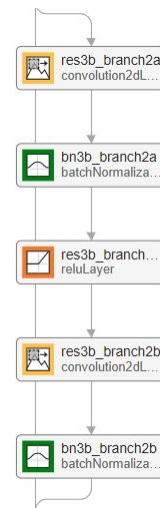


Figure 7: Second, Third, Fourth, Fifth Group



## Sixth Group

In this group there aren't iteration. We can observe that there is a cleaning phase composed by a ReLU function and a MaxPooling function that pass his output to input of Fully Connected layer that in the original case had an output of 1000 categories but i changed it to adapt it to my task with an output of 5 categories, then follow a softmax layer and a classification layer

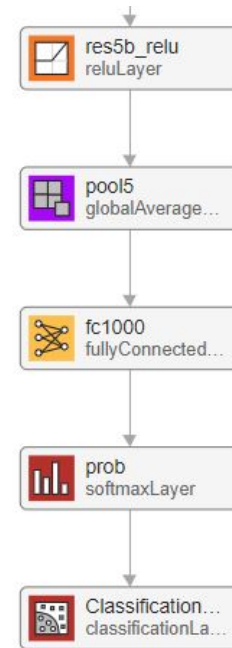


Figure 8: Sixth Group

## Data Processing

Since I had a problem with my computational power during the training of the net, I decided to reduce the number of images for each categories during the train phase using the following partition:

- Train with respectively 500 images for categories
- Validation with respectively 200 images for categories
- Test with respectively 4300 images for categories

```
[train,validation,test] = splitEachLabel(image_datastore,0.1, 0.04, 0.86, 'randomized');
```

Then i proceeded resizing the image to give in input to net from 512-by-512 X 3 to 224-by-224 X 3

```
resized_images_train=augmentedImageDatastore([224 224 3],train);  
resized_images_validation = augmentedImageDatastore([224 224 3],validation);  
resized_images_test = augmentedImageDatastore([224 224 3],test);
```

I could have used data augmentation procedure to allow to the model generalize more, but since there is already randomness in the images for example bottles rotated,bottles shifted, I preferred don't use that procedure due to my limited computational power.

# Training ResNet-18

After data preprocessing step i setted the hyperparameter

```
opts = trainingOptions("sgdm",...  
    "ExecutionEnvironment","auto",...  
    "InitialLearnRate",0.01,...  
    "MaxEpochs",10,...  
    "MiniBatchSize",64,...  
    "Shuffle","every-epoch",...  
    "ValidationFrequency",70,...  
    "Plots","training-progress",...  
    "ValidationData",resized_images_validation,...  
    "Momentum",0.9);
```

and I trained the net

```
[net, traininfo] = trainNetwork(resized_images_train,resnet_18,opts);
```

As we can see from Figure 9 the net has been trained for more than 2 hours,it has been able to reach a good accuracy after the second epoch, furthermore from the training chart it doesn't seem to be overfitting since that train accuracy and validation accuracy have almost the same value. Since I left 4300 images out for each categories from training phase, I will use them to test the real performance of the model.

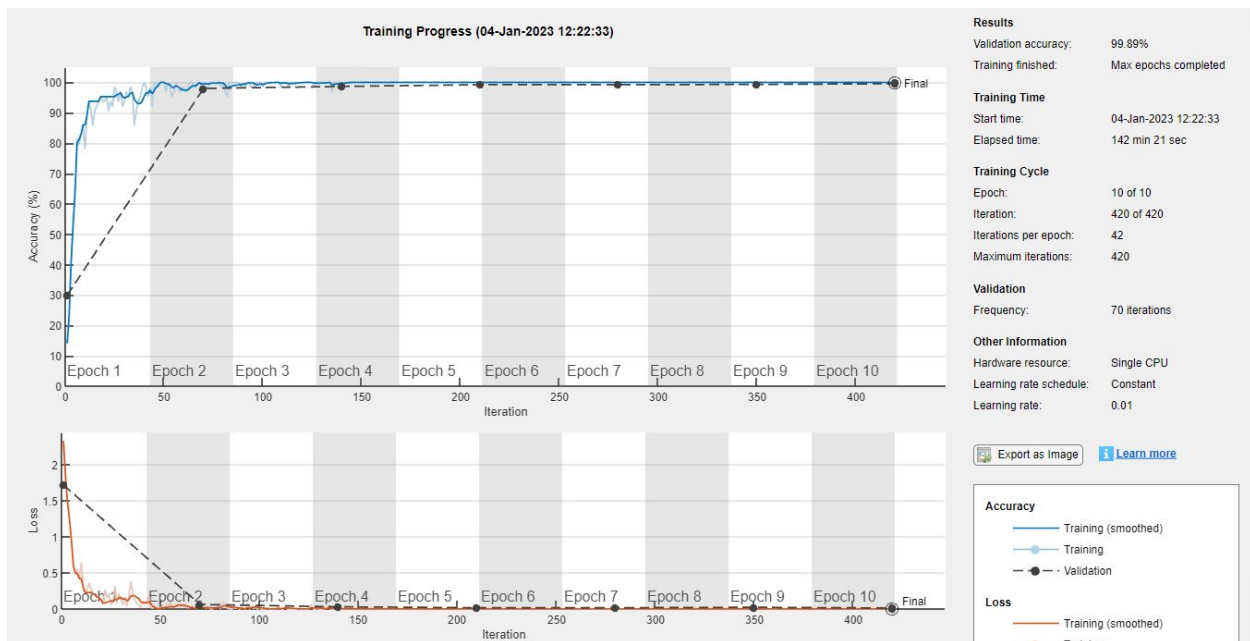


Figure 9: Training chart

## Model performance evaluation on Validation set

Looking at the confusion matrix computed on validation set we can note that only 5 images has been misclassified leading to an accuracy of 99.50%, so if I will obtain the same accuracy on test set I can maintain the model without tuning the hyperparameters. Below I report the association between name class and label assigned to the class:

- 1. Beer
- 2. Plastic
- 3. Soda
- 4. Water
- 5. Wine

```
true_validation_labels = validation.Labels;  
pred_validation_labels = classify(net, rsz_validation);  
accuracy_validation = mean(true_validation_labels == pred_validation_labels);  
  
ConfVal = confusionmat(true_validation_labels, pred_validation_labels);  
confusionchart(ConfVal)
```

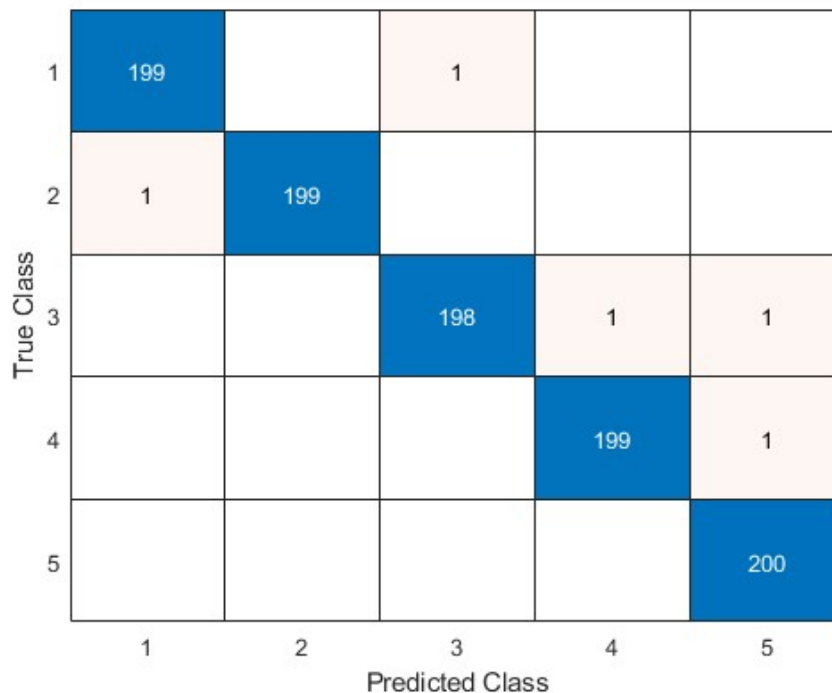


Figure 10: Confusion Matrix on Validation Set

## Model performance evaluation on Test set

Looking at the confusion matrix computed on test set we can note that only 101 images has been misclassified leading to an accuracy of 99.53%, so I can consider that the model is a very good model even without tuning the hyperparameters Below I report the association between name class and label assigned to the class:

- 1. Beer
- 2. Plastic
- 3. Soda
- 4. Water
- 5. Wine

```
true_test_labels = test.Labels;  
pred_test_labels = classify(net, rsz_test);  
accuracy_test = mean(true_test_labels == pred_test_labels);  
  
C = confusionmat(true_test_labels, pred_test_labels);  
confusionchart(C)
```

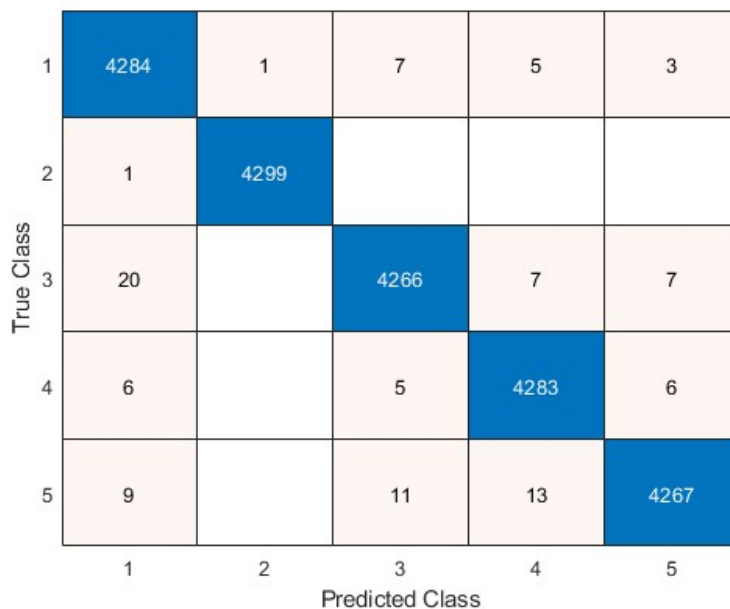


Figure 11: Confusion Matrix on Test Set

As we can from the confusion matrix most of errors occurred in the beer predicted class, instead the predicted class with less errors has been plastic class with only 1 misclassification error.

Maybe training the model with all the images that are available and fine-tuning the hyperparameters I could obtain better result even if I think that do better than this becomes very hard indeed we could obtain the opposite effect making the model worse

## Main misclassification errors

I consider that is interesting have a look to the main images that have been misclassified for each class.

### Misclassified Beer Bottles

```
chosenClass = "Beer Bottles";  
classIdx = find(net.Layers(end).Classes == chosenClass);  
  
numImgsToShow = 9;  
  
[sortedScores,imgIdx] = findMinActivatingImages(test,chosenClass,score_test,numImgsToShow);  
  
figure  
plotImages(test,imgIdx,sortedScores,pred_test_labels,numImgsToShow)
```

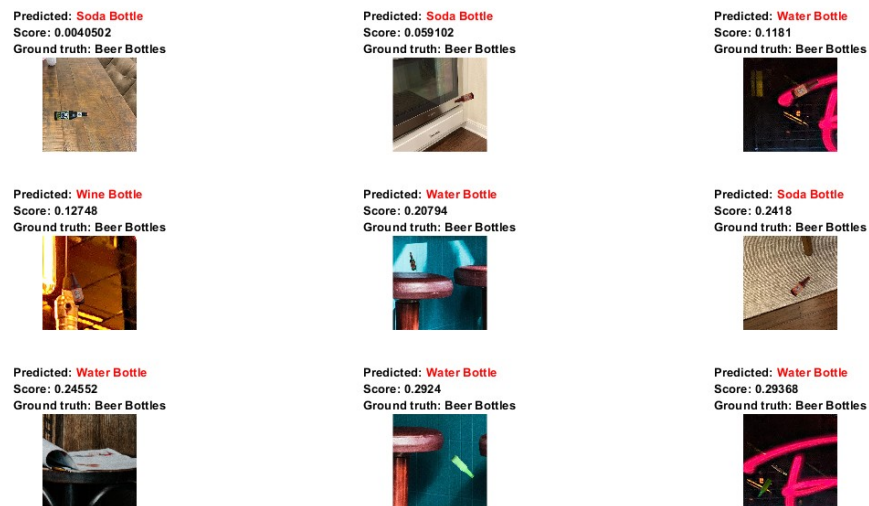


Figure 12: Misclassified Beer Bottles

## Misclassified Plastic Bottles

```
chosenClass = "Plastic Bottles";  
classIdx = find(net.Layers(end).Classes == chosenClass);  
  
numImgsToShow = 9;  
  
[sortedScores,imgIdx] = findMinActivatingImages(test,chosenClass,score_test,numImgsToShow);  
  
figure  
plotImages(test,imgIdx,sortedScores,pred_test_labels,numImgsToShow)
```

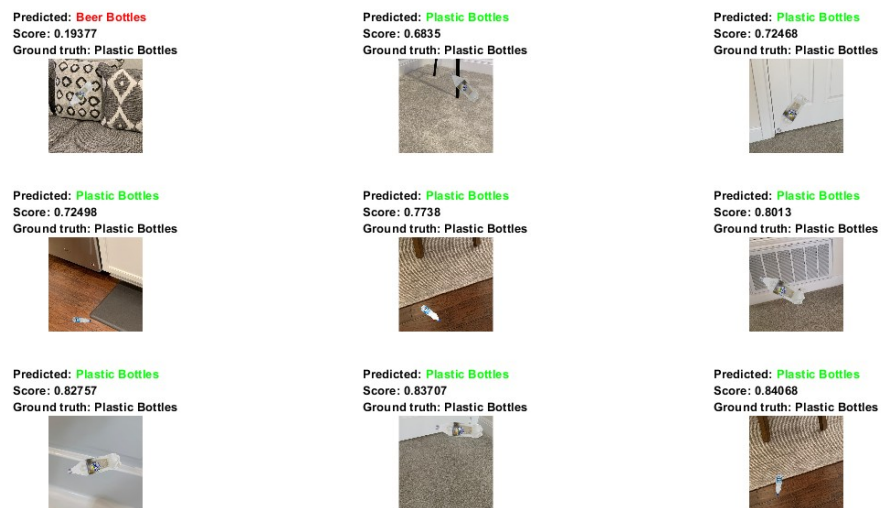


Figure 13: Misclassified Plastic Bottles

## Misclassified Soda Bottles

```
chosenClass = "Soda Bottle";  
classIdx = find(net.Layers(end).Classes == chosenClass);  
  
numImgsToShow = 9;  
  
[sortedScores,imgIdx] = findMinActivatingImages(test,chosenClass,score_test,numImgsToShow);  
  
figure  
plotImages(test,imgIdx,sortedScores,pred_test_labels,numImgsToShow)
```



Figure 14: Misclassified Soda Bottles

## Misclassified Wine Bottles

```
chosenClass = "Wine Bottle";  
classIdx = find(net.Layers(end).Classes == chosenClass);  
  
numImgsToShow = 9;  
  
[sortedScores,imgIdx] = findMinActivatingImages(test,chosenClass,score_test,numImgsToShow);  
  
figure  
plotImages(test,imgIdx,sortedScores,pred_test_labels,numImgsToShow)
```



Figure 15: Misclassified Wine Bottles



## Misclassified Water Bottles

```
chosenClass = "Water Bottle";  
classIdx = find(net.Layers(end).Classes == chosenClass);  
  
numImgsToShow = 9;  
  
[sortedScores,imgIdx] = findMinActivatingImages(test,chosenClass,score_test,numImgsToShow);  
  
figure  
plotImages(test,imgIdx,sortedScores,pred_test_labels,numImgsToShow)
```

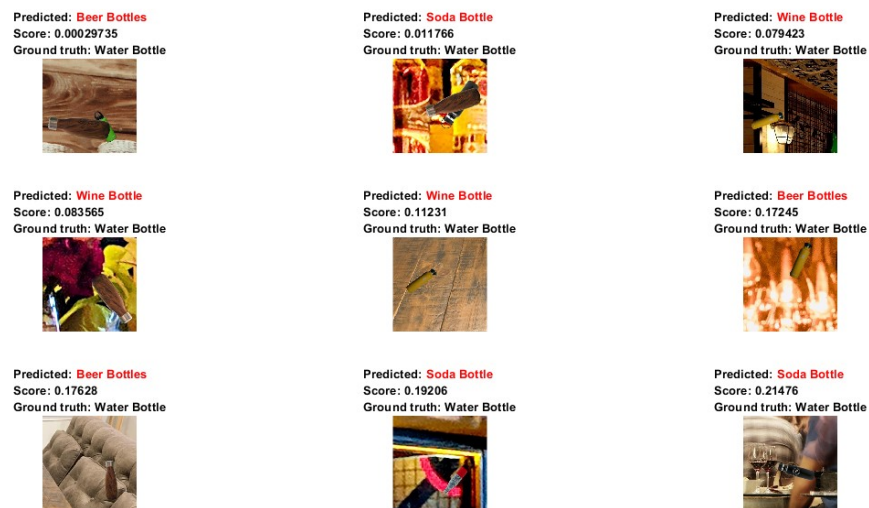


Figure 16: Misclassified Water Bottles