Bottles Synthetic Images Classification

\

Neural Computing 2022/2023

Giuseppe Pulino

Contents

List of Figures

Data Exploration

Link to the Dataset

The dataset contains synthetically generated images of bottles scattered around random backgrounds. The main folder contains 25000 Images stored with JPG extension, with a resolution of 512-by-512 pixels and divided in the following categories:

- Soda Bottle
- Water Bottle
- Plastic Bottle
- Wine Bottle
- Beer Bottle

Let's see an example of each class.

Example categories

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

Problem Designing

The problem that I am 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 I 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 it 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 a 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

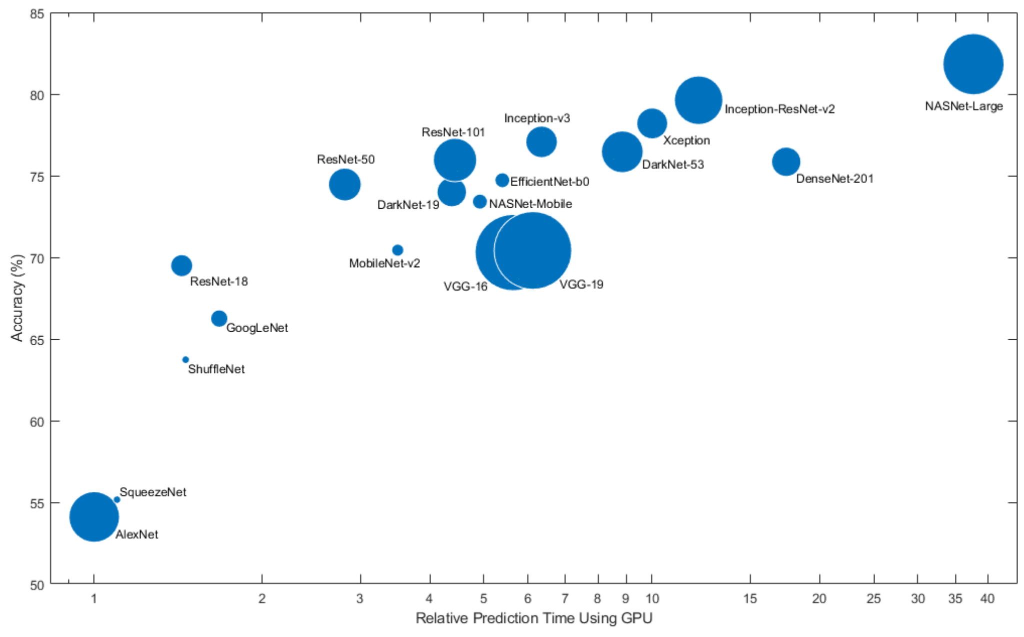


Figure 1: comparing the ImageNet validation accuracy of each net

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 3

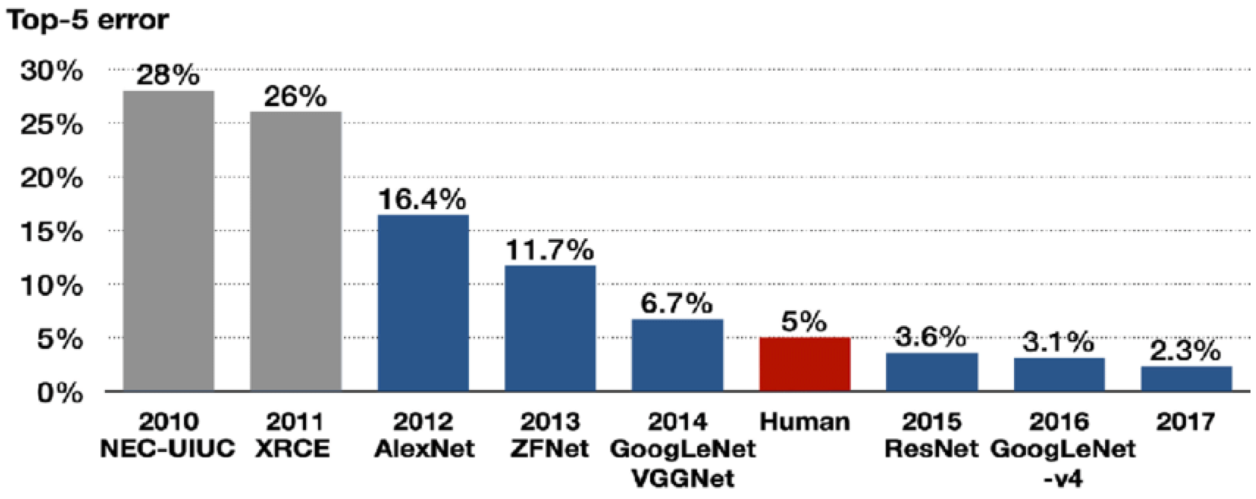
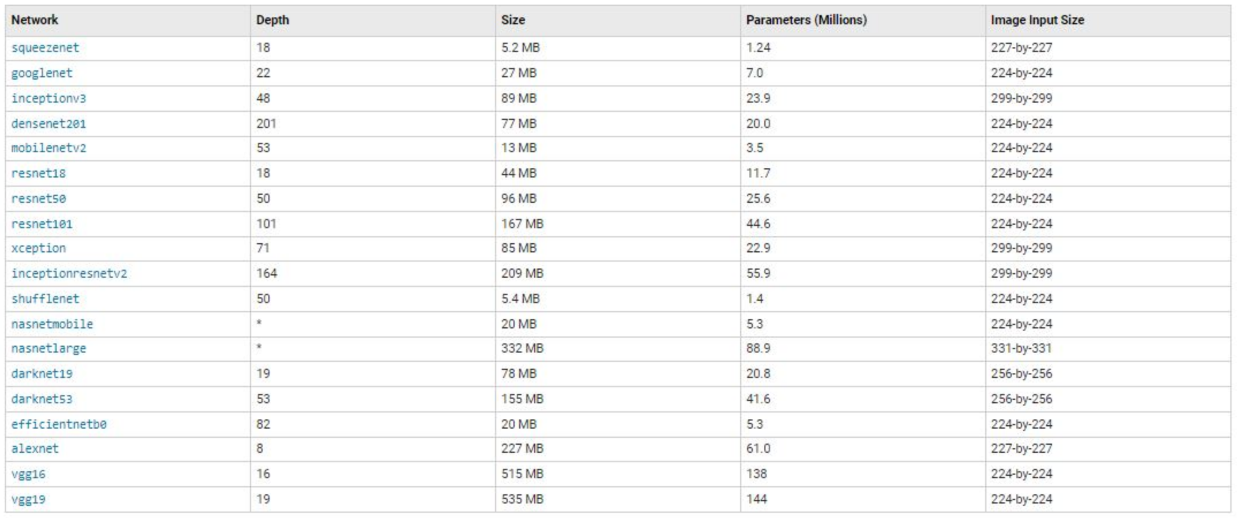


Figure 2: ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



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

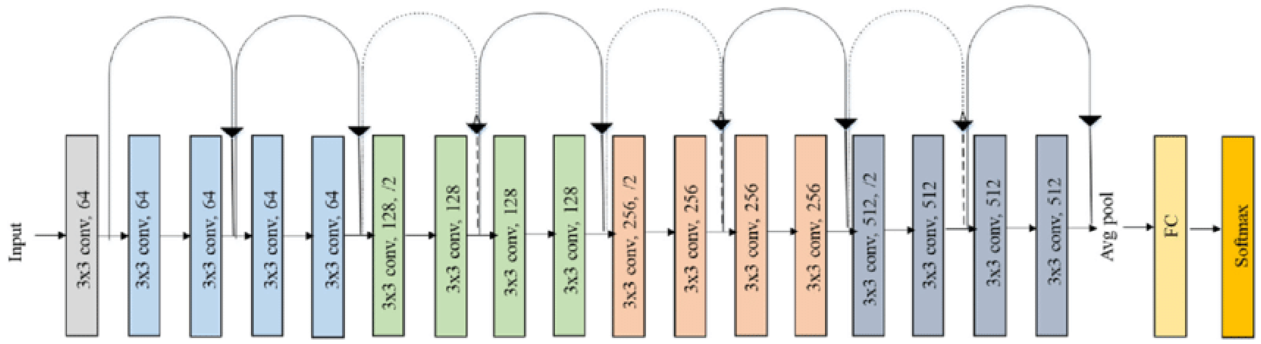
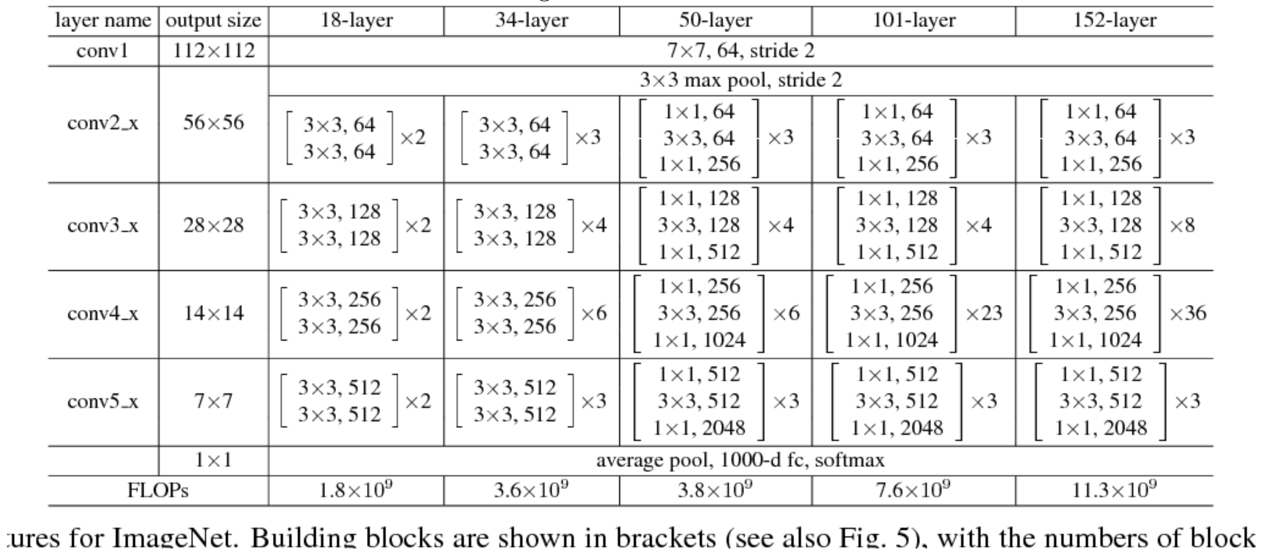


Figure 4: ResNet-18



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				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

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 5 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

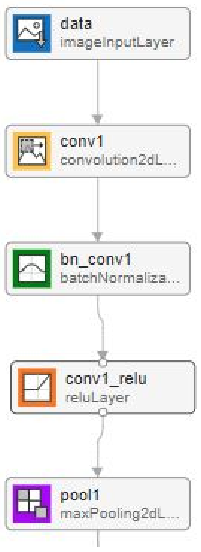


Figure 6: First Group

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

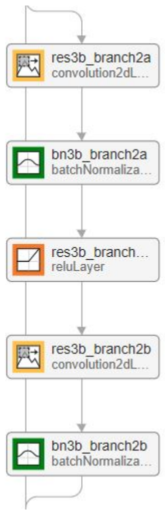


Figure 7: Second, Third, Fourth, Fifth Group

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 ouptut 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

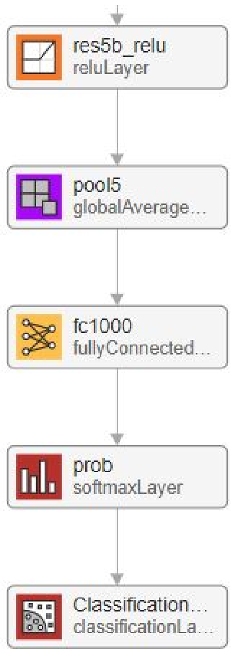


Figure 8: Sixth Group

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

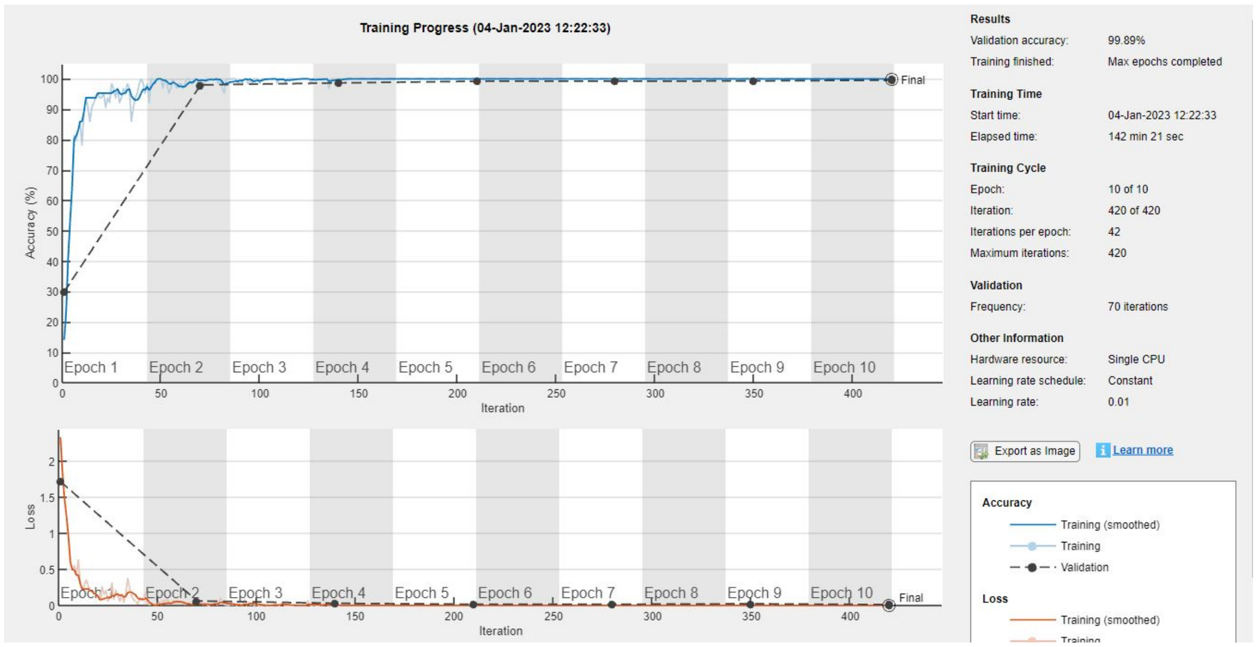
After data preprocessing step i setted the hyperparameter

```
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.



Results

Validation accuracy:	99.89%
Training finished:	Max epochs completed

Training Time

Start time:	04-Jan-2023 12:22:33
Elapsed time:	142 min 21 sec

Training Cycle

Epoch:	10 of 10
Iteration:	420 of 420
Iterations per epoch:	42
Maximum iterations:	420

Validation

Frequency:	70 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.01

Figure 9: Training chart

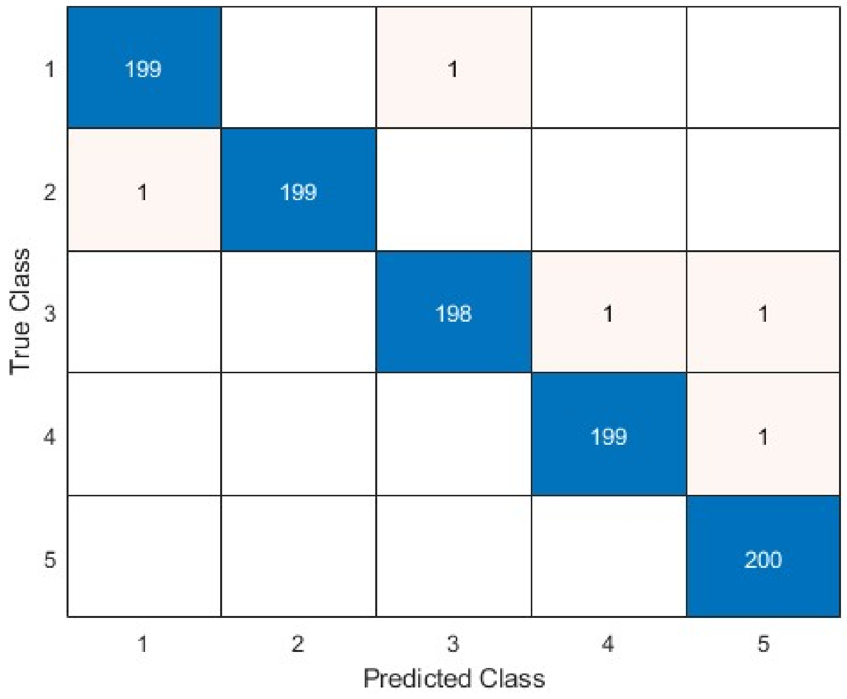
Model performance evaluation on Validation set

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)
```



	1	2	3	4	5
1	199		1		
2	1	199			
3			198	1	1
4				199	1
5					200

Figure 10: Confusion Matrix on Validation Set

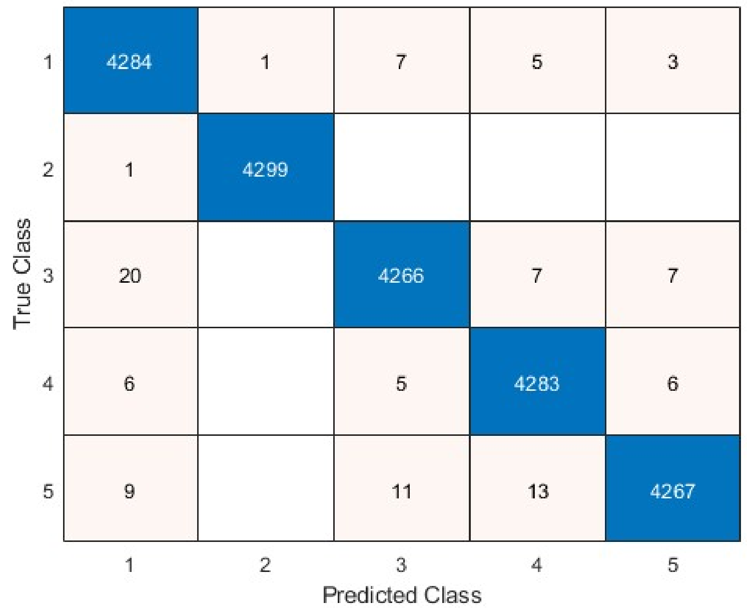
Model performance evaluation on Test set

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)
```



True \ Predicted	1	2	3	4	5
1	4284	1	7	5	3
2	1	4299			
3	20		4266	7	7
4	6		5	4283	6
5	9		11	13	4267

Figure 11: Confusion Matrix on Test Set

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

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

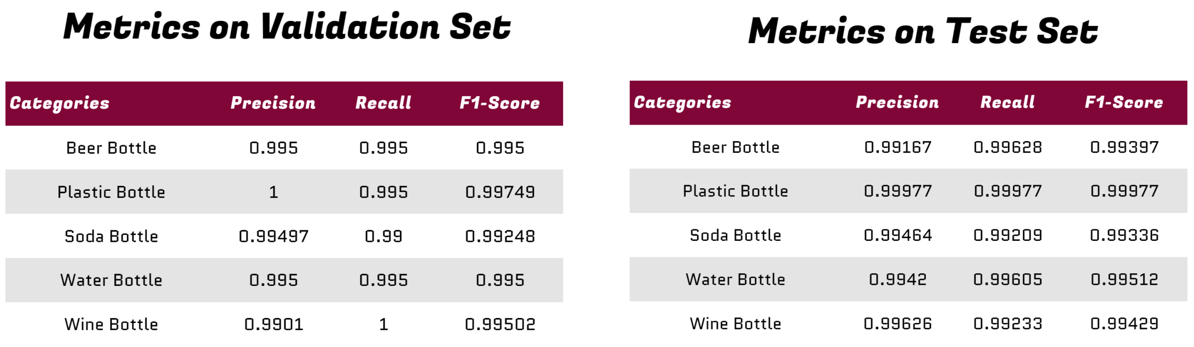
Comparing Metrics

To have a faster method of comparison, I calculated from the confusion matrix the following metrics for each categories:

- **Precision**: It represents the percentage of images correctly classify for each predicted class over the total number of images classified in that class
- **Recall**: It represents the percentage of images correctly classify for each true class over the total number of images present in that class
- **F1-Score**: By definition, F1-score is the harmonic mean of precision and recall. It combines precision and recall into a single number using the following formula:

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

What I obtained is reported in the tables below:



Metrics on Validation Set

Categories	Precision	Recall	F1-Score
Beer Bottle	0.995	0.995	0.995
Plastic Bottle	1	0.995	0.99749
Soda Bottle	0.99497	0.99	0.99248
Water Bottle	0.995	0.995	0.995
Wine Bottle	0.9901	1	0.99502

Metrics on Test Set

Categories	Precision	Recall	F1-Score
Beer Bottle	0.99167	0.99628	0.99397
Plastic Bottle	0.99977	0.99977	0.99977
Soda Bottle	0.99464	0.99209	0.99336
Water Bottle	0.9942	0.99605	0.99512
Wine Bottle	0.99626	0.99233	0.99429

Figure 12: Metrics

From those metrics I can confirm that the models has reached an incredible result on the assigned dataset

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)
```

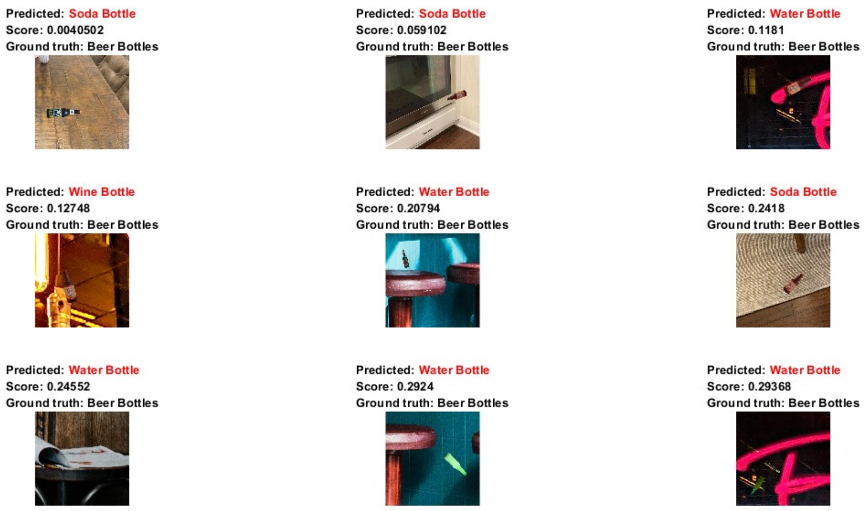


Figure 13: Misclassified Beer Bottles

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)
```

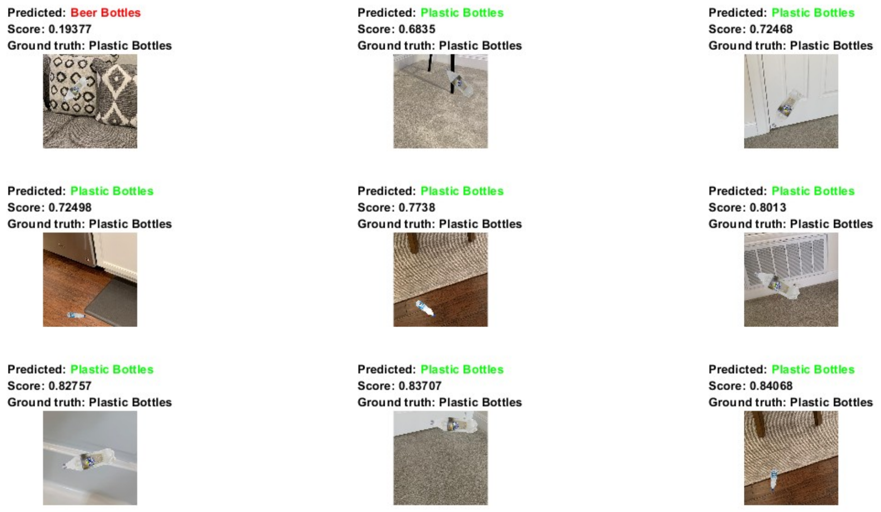

Predicted: Beer Bottles
Score: 0.19377
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.6835
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.72468
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.72498
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.7738
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.8013
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.82757
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.83707
Ground truth: Plastic Bottles
Predicted: Plastic Bottles
Score: 0.84068
Ground truth: Plastic Bottles

Figure 14: 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)
```

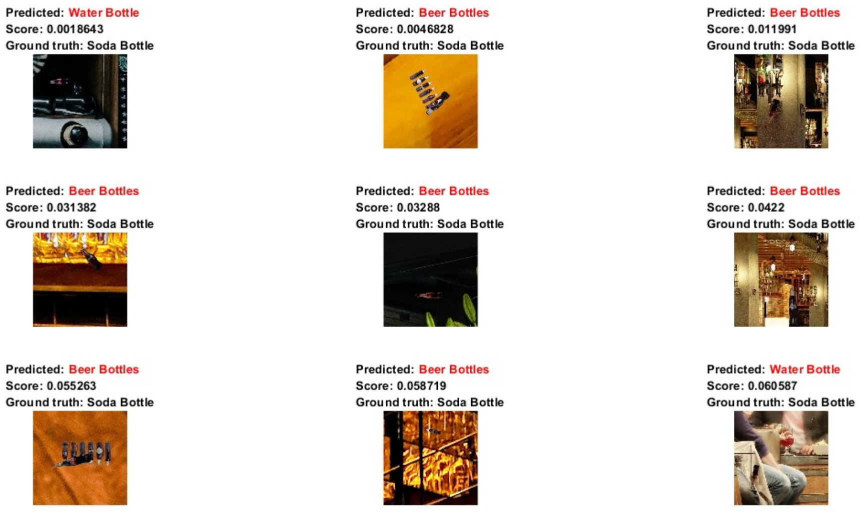


Figure 15: Misclassified Soda Bottles

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)
```

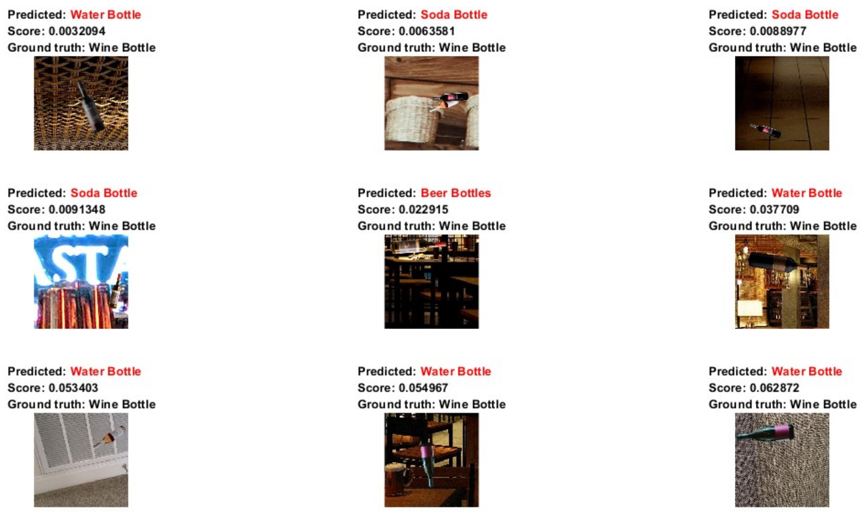


Figure 16: Misclassified Wine Bottles

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)
```

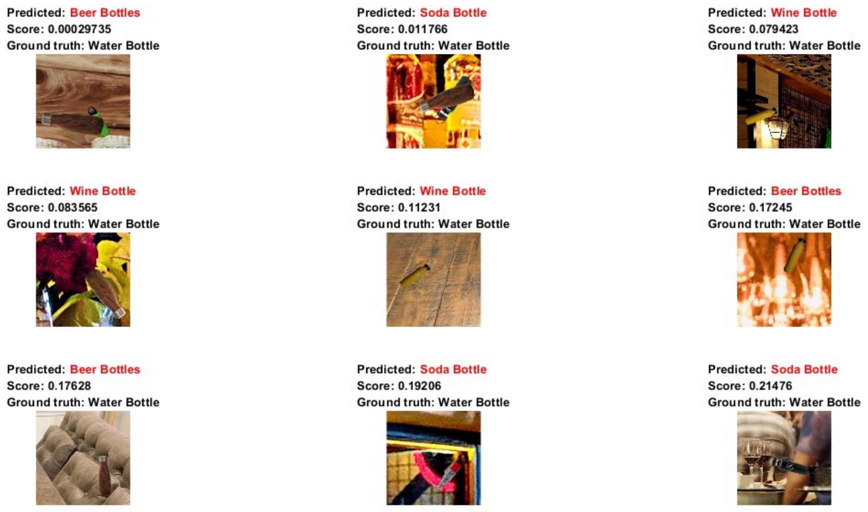


Figure 17: Misclassified Water Bottles