Leandro Maglianella 1792507

Machine Learning: Homework 2.

RoboCup@Home Object Classification:

Images classification with Neural Networks

1. Abstract

In this homework, we are provided with the RoboCup@Home-Objects dataset that has been developed within the RoboCup@Home competition: this league aims to develop service and assistive robot technology with high relevance for future personal domestic applications.

The goal is to be able to train a machine-learning model based on neural networks using the set of samples contained in the dataset that will be able to classify new instances for the classes that we will take into account.

I worked on Google Colab to complete this homework.

Note on the networks used later:

In this homework, I have used three different CNN: AlexNet, VGG16-TransferNet and MobileNet v2. I am writing this note only to mention that, obviously, these neural networks do not belong to me and I have only adapted them to my code.

The implementations of the first two were adapted from the "Machine Learning Exercises" 11 and 12 of the "Machine Learning 2020/21" course of Professor Luca Iocchi. On the other hand, I studied independently the third network and I got its implementation from a website that I mentioned at the beginning of the "MY WORK" section in the attached Colab notebook.

2. Pre-processing and Data splitting

The first phase to carry out is the pre-processing of the data in order to organize the homework. The goal of this phase is generating a dataset in a normal form as $D = \{(x_n, t_n)^N_{n=1}\}$, with x_n = image and t_n = its class. The RoboCup@Home-Objects dataset is a huge dataset containing 196k images divided in 8 main parent categories, each distributed in some children categories for a total of 180 children categories. To streamline the homework, only one children category for each parent will be considered, in so doing we will work on eight total classes. The choice of which children categories to use was made randomly (using as random seed my matricola code) and here on the side are the classes that have been assigned to me:

```
containers/plastic_food_container
tableware/buillin_cup
drinks/Sparkling_Water
cutlery/Knives
fruits/Pineapple
snacks/Party_Mix_snack
food/Jams
cleaning_stuff/Caddies
```

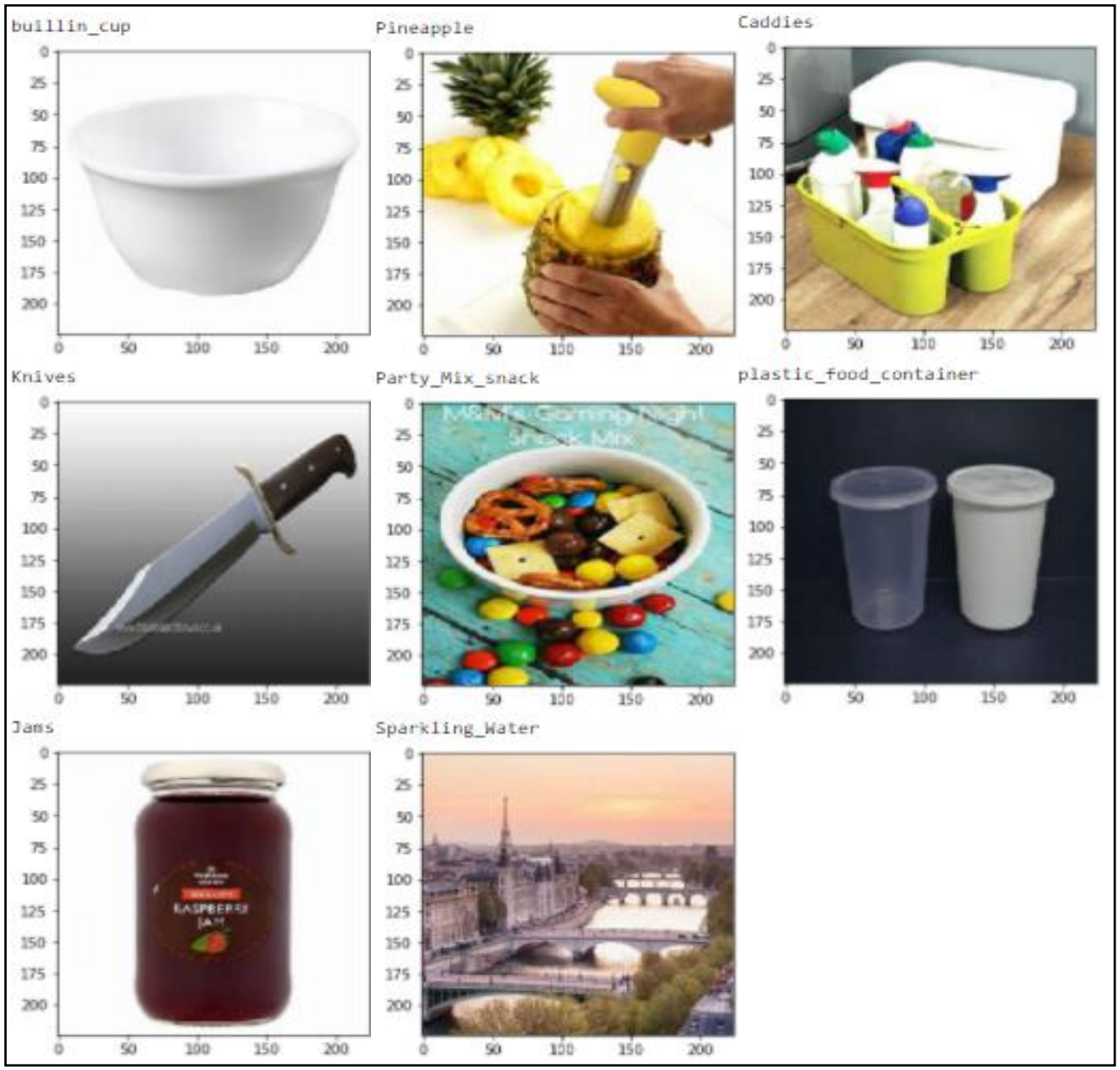
The eight classes folder assigned contain overall 7880 images.

I downloaded, unzipped and uploaded them into my Google Drive divided in the eight class folders. Two ImageDataGenerators were created (one for model training and one for model testing) with a rescaling factor of 1.0 / 255 and a validation split of 20%. The flow_from_directory function was applied to both with a target_size of (224, 224) and shuffle = True. Each batch of images contains 32 images.

The following are the results of this phase:

```
Working in: /content/drive/MyDrive/ML_HW2_Datasets

Training data:
Found 6307 images belonging to 8 classes.

Validation data:
Found 1573 images belonging to 8 classes.

Image batch shape:  (32, 224, 224, 3)
Label batch shape:  (32, 8)

Classification labels:
 ['Caddies' 'Jams' 'Knives' 'Party_Mix_snack' 'Pineapple' 'Sparkling_Water'
 'buillin_cup' 'plastic_food_container']
```

I have then printed some random images with their class to visualize them. Looking at them (especially here the example image of the Sparkling_Water class), it can be seen that not all images represent what they should: the training will be done including these intrinsic errors and outliers of the dataset to complicate the learning and add a semblance of real world to the dataset. In fact, in the everyday reality, not everything is so easily classified and the model must learn to overcome the presence of any noises.



3. Convolutional Neural Networks model creation

Three types of neural networks based on different concepts have been created.

3.1 AlexNet

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 107, 54, 96)	34944
activation (Activation)	(None, 107, 54, 96)	0
max_pooling2d (MaxPooling2D)	(None, 53, 27, 96)	0
batch_normalization (BatchNo	(None, 53, 27, 96)	384
conv2d_1 (Conv2D)	(None, 43, 17, 256)	2973952
activation_1 (Activation)	(None, 43, 17, 256)	0
max_pooling2d_1 (MaxPooling2	(None, 21, 8, 256)	0
batch_normalization_1 (Batch	(None, 21, 8, 256)	1024
conv2d_2 (Conv2D)	(None, 19, 6, 384)	885120
activation_2 (Activation)	(None, 19, 6, 384)	0
batch_normalization_2 (Batch	(None, 19, 6, 384)	1536
conv2d_3 (Conv2D)	(None, 17, 4, 384)	1327488
activation_3 (Activation)	(None, 17, 4, 384)	0
batch_normalization_3 (Batch	(None, 17, 4, 384)	1536
conv2d_4 (Conv2D)	(None, 15, 2, 256)	884992
activation_4 (Activation)	(None, 15, 2, 256)	0
max_pooling2d_2 (MaxPooling2	(None, 7, 1, 256)	0
batch_normalization_4 (Batch	(None, 7, 1, 256)	1024
flatten (Flatten)	(None, 1792)	0
dense (Dense)	(None, 4096)	7344128
activation_5 (Activation)	(None, 4096)	0
dropout (Dropout)	(None, 4096)	0
batch_normalization_5 (Batch	(None, 4096)	16384
dense_1 (Dense)	(None, 4096)	16781312
activation_6 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
batch_normalization_6 (Batch	(None, 4096)	16384
dense_2 (Dense)	(None, 1000)	4097000
activation_7 (Activation)	(None, 1000)	0
dropout_2 (Dropout)	(None, 1000)	0
batch_normalization_7 (Batch	(None, 1000)	4000
dense_3 (Dense)	(None, 8)	8008
activation_8 (Activation)	(None, 8)	0

Total params: 34,379,216
Trainable params: 34,358,080
Non-trainable params: 21,136

3.2 VGG16-TransferNet

Model: "transferNet"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_2 (Flatten)	(None, 25088)	0
batch_normalization_12 (Batc	(None, 25088)	100352
dropout_5 (Dropout)	(None, 25088)	0
dense_7 (Dense)	(None, 200)	5017800
batch_normalization_13 (Batc	(None, 200)	800
dropout_6 (Dropout)	(None, 200)	0
dense_8 (Dense)	(None, 100)	20100
batch_normalization_14 (Batc	(None, 100)	400
batch_normalization_15 (Batc	(None, 100)	400
dense_9 (Dense)	(None, 8)	808

Total params: 19,855,348
Trainable params: 7,449,492
Non-trainable params: 12,405,856

3.3 MobileNet v2

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 8)                 10248
=================================================================
Total params: 2,268,232
Trainable params: 10,248
Non-trainable params: 2,257,984
```

4. Training

These three Neural Networks have been fitted with the same method passing as parameters the train_generator and test_generator, 198 training steps (= 6307 / 32) and 50 validation steps (= 1573 / 32). Moreover, each training was done in 10 epochs and every training epoch took more or less 20-30 seconds. Note that every network was not only trained one time, each training result I reported is the best one on average, obtained for each network after training them several times. The following are the results for each network:

4.1 AlexNet

```
Epoch 1/10
198/198 [==============================] - 23s 118ms/step - loss: 3.0030 - accuracy: 0.3147 - val_loss: 3.1428 - val_accuracy: 0.2238
Epoch 2/10
198/198 [==============================] - 23s 117ms/step - loss: 2.7248 - accuracy: 0.3923 - val_loss: 2.8471 - val_accuracy: 0.3274
Epoch 3/10
198/198 [==============================] - 23s 118ms/step - loss: 2.7013 - accuracy: 0.4067 - val_loss: 2.2637 - val_accuracy: 0.5010
Epoch 4/10
198/198 [==============================] - 23s 118ms/step - loss: 2.6290 - accuracy: 0.4072 - val_loss: 2.1277 - val_accuracy: 0.5607
Epoch 5/10
198/198 [==============================] - 23s 118ms/step - loss: 2.4330 - accuracy: 0.4562 - val_loss: 3.0517 - val_accuracy: 0.3840
Epoch 6/10
198/198 [==============================] - 23s 117ms/step - loss: 2.4157 - accuracy: 0.4704 - val_loss: 2.4263 - val_accuracy: 0.4342
Epoch 7/10
198/198 [==============================] - 23s 117ms/step - loss: 2.3860 - accuracy: 0.4722 - val_loss: 2.3719 - val_accuracy: 0.5283
Epoch 8/10
198/198 [==============================] - 23s 117ms/step - loss: 2.3553 - accuracy: 0.4853 - val_loss: 2.2908 - val_accuracy: 0.5175
Epoch 9/10
198/198 [==============================] - 24s 120ms/step - loss: 2.2333 - accuracy: 0.5204 - val_loss: 3.9209 - val_accuracy: 0.3458
Epoch 10/10
198/198 [==============================] - 23s 117ms/step - loss: 2.2046 - accuracy: 0.5235 - val_loss: 2.6138 - val_accuracy: 0.4825
```

```
Epoch 1/10
197/197 [==============================] - 19s 96ms/step - loss: 2.8439 - accuracy: 0.3106 - val_loss: 2.7503 - val_accuracy: 0.1291
Epoch 2/10
197/197 [==============================] - 18s 93ms/step - loss: 2.6261 - accuracy: 0.3551 - val_loss: 2.5946 - val_accuracy: 0.2740
Epoch 3/10
197/197 [==============================] - 18s 93ms/step - loss: 2.4683 - accuracy: 0.4080 - val_loss: 2.1303 - val_accuracy: 0.4558
Epoch 4/10
197/197 [==============================] - 18s 93ms/step - loss: 2.3649 - accuracy: 0.4360 - val_loss: 2.3634 - val_accuracy: 0.4730
Epoch 5/10
197/197 [==============================] - 18s 93ms/step - loss: 2.3154 - accuracy: 0.4363 - val_loss: 3.5037 - val_accuracy: 0.3121
Epoch 6/10
197/197 [==============================] - 18s 93ms/step - loss: 2.2729 - accuracy: 0.4475 - val_loss: 2.8570 - val_accuracy: 0.3776
Epoch 7/10
197/197 [==============================] - 18s 93ms/step - loss: 2.1999 - accuracy: 0.4620 - val_loss: 2.3862 - val_accuracy: 0.4024
Epoch 8/10
197/197 [==============================] - 18s 94ms/step - loss: 2.2300 - accuracy: 0.4548 - val_loss: 2.0471 - val_accuracy: 0.5118
Epoch 9/10
197/197 [==============================] - 18s 94ms/step - loss: 2.2153 - accuracy: 0.4566 - val_loss: 2.6337 - val_accuracy: 0.3643
Epoch 10/10
197/197 [==============================] - 18s 94ms/step - loss: 2.1153 - accuracy: 0.4927 - val_loss: 1.9771 - val_accuracy: 0.5486
```

The first training for AlexNet was done with the parameters so far discussed and obtained a validation accuracy of 48.25%. Then I tried to modify the target_size in the pre-processing from (224, 224) to (118, 224): training this I obtained a higher validation accuracy of 54.86%.

4.2 VGG16-TransferNet

```
Epoch 1/10
198/198 [==============================] - 33s 165ms/step - loss: 1.0040 - accuracy: 0.6537 - val_loss: 0.5597 - val_accuracy: 0.8074
Epoch 2/10
198/198 [==============================] - 32s 162ms/step - loss: 0.6234 - accuracy: 0.7894 - val_loss: 0.5531 - val_accuracy: 0.8334
Epoch 3/10
198/198 [==============================] - 31s 159ms/step - loss: 0.4556 - accuracy: 0.8472 - val_loss: 0.5451 - val_accuracy: 0.8303
Epoch 4/10
198/198 [==============================] - 32s 161ms/step - loss: 0.3452 - accuracy: 0.8830 - val_loss: 2.7259 - val_accuracy: 0.4857
Epoch 5/10
198/198 [==============================] - 32s 160ms/step - loss: 0.5607 - accuracy: 0.8070 - val_loss: 0.5521 - val_accuracy: 0.8404
Epoch 6/10
198/198 [==============================] - 32s 160ms/step - loss: 0.3551 - accuracy: 0.8771 - val_loss: 0.5931 - val_accuracy: 0.8099
Epoch 7/10
198/198 [==============================] - 32s 160ms/step - loss: 0.2793 - accuracy: 0.9028 - val_loss: 0.5717 - val_accuracy: 0.8392
Epoch 8/10
198/198 [==============================] - 32s 161ms/step - loss: 0.2457 - accuracy: 0.9172 - val_loss: 0.5046 - val_accuracy: 0.8671
Epoch 9/10
198/198 [==============================] - 32s 160ms/step - loss: 0.1775 - accuracy: 0.9423 - val_loss: 0.5342 - val_accuracy: 0.8551
Epoch 10/10
198/198 [==============================] - 32s 160ms/step - loss: 0.1373 - accuracy: 0.9537 - val_loss: 0.5188 - val_accuracy: 0.8595
```

This training for TransferNet was done with the parameters so far discussed and obtained a validation accuracy of 85.95%. Then I tried to modify the target_size in the pre-processing from (224, 224) to (118, 224): training this I obtained a lower validation accuracy of 83.53%, so I am not reporting it here.

4.3 MobileNet v2

Finally, MobileNet v2 was trained with the parameters so far discussed and obtained a validation accuracy of 90.21%.

```
Epoch 1/10
198/198 [==============================] - 3516s 18s/step - loss: 0.9404 - acc: 0.6786 - val_loss: 0.4045 - val_acc: 0.8767
Epoch 2/10
198/198 [==============================] - 290s 1s/step - loss: 0.5253 - acc: 0.8216 - val_loss: 0.3579 - val_acc: 0.8875
Epoch 3/10
198/198 [==============================] - 284s 1s/step - loss: 0.4512 - acc: 0.8446 - val_loss: 0.3301 - val_acc: 0.8894
Epoch 4/10
198/198 [==============================] - 285s 1s/step - loss: 0.4110 - acc: 0.8625 - val_loss: 0.3217 - val_acc: 0.8964
Epoch 5/10
198/198 [==============================] - 285s 1s/step - loss: 0.3734 - acc: 0.8717 - val_loss: 0.3136 - val_acc: 0.8957
Epoch 6/10
198/198 [==============================] - 284s 1s/step - loss: 0.3549 - acc: 0.8760 - val_loss: 0.3233 - val_acc: 0.9002
Epoch 7/10
198/198 [==============================] - 285s 1s/step - loss: 0.3515 - acc: 0.8792 - val_loss: 0.3027 - val_acc: 0.9053
Epoch 8/10
198/198 [==============================] - 286s 1s/step - loss: 0.3214 - acc: 0.8843 - val_loss: 0.3053 - val_acc: 0.9027
Epoch 9/10
198/198 [==============================] - 285s 1s/step - loss: 0.3137 - acc: 0.8919 - val_loss: 0.3296 - val_acc: 0.8957
Epoch 10/10
198/198 [==============================] - 285s 1s/step - loss: 0.3069 - acc: 0.8922 - val_loss: 0.3218 - val_acc: 0.9021
```

In this image above one can see that each epoch was trained in a much longer time than the 20-30 seconds mentioned above. This is not because of a network related problem but it is due to the fact that this training session that I am reporting was the very first I performed

and Google Colab still had to work out some of its initial configurations. In subsequent trainings, the time per epoch was normal and the accuracy similar to the one reported.

5. Classification evaluation

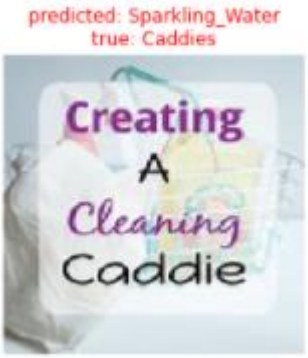
Given that MobileNet v2 was the neural network with the best accuracy obtained in the training phase, I chose it for this subsequent evaluation phase. For the eight classes the following confusion matrix has been computed, it is not shown here in the form of a matrix being it large:

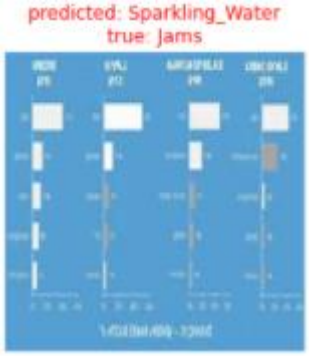
```
50/50 [==============================] - 4s 83ms/step - loss: 0.3218 - acc: 0.9021
50/50 [==============================] - 4s 80ms/step
True                 Predicted                 errors  err %
------------------------------------------------------------------
plastic_food_container ->  Caddies             23      1.46 %
Jams             ->  Caddies                   23      1.46 %
Jams             ->  Sparkling_Water           12      0.76 %
Caddies          ->  Sparkling_Water           11      0.70 %
Jams             ->  Knives                    9       0.57 %
Caddies          ->  Jams                      8       0.51 %
Knives           ->  Caddies                   7       0.45 %
Sparkling_Water  ->  Jams                      6       0.38 %
Caddies          ->  Knives                    5       0.32 %
buillin_cup      ->  Jams                      5       0.32 %
buillin_cup      ->  plastic_food_container    4       0.25 %
Caddies          ->  plastic_food_container    4       0.25 %
Pineapple        ->  Party_Mix_snack           4       0.25 %
plastic_food_container ->  Party_Mix_snack     3       0.19 %
Knives           ->  Jams                      3       0.19 %
plastic_food_container ->  Knives              2       0.13 %
plastic_food_container ->  Jams                2       0.13 %
Pineapple        ->  Sparkling_Water           2       0.13 %
Sparkling_Water  ->  Caddies                   2       0.13 %
plastic_food_container ->  Sparkling_Water     2       0.13 %
Jams             ->  Pineapple                 2       0.13 %
Jams             ->  Party_Mix_snack           2       0.13 %
Caddies          ->  Party_Mix_snack           2       0.13 %
Party_Mix_snack  ->  Sparkling_Water           1       0.06 %
plastic_food_container ->  Pineapple           1       0.06 %
Party_Mix_snack  ->  Jams                      1       0.06 %
Sparkling_Water  ->  Pineapple                 1       0.06 %
Sparkling_Water  ->  plastic_food_container    1       0.06 %
buillin_cup      ->  Caddies                   1       0.06 %
Knives           ->  Pineapple                 1       0.06 %
Jams             ->  plastic_food_container    1       0.06 %
Caddies          ->  Pineapple                 1       0.06 %
Party_Mix_snack  ->  plastic_food_container    1       0.06 %
Party_Mix_snack  ->  Pineapple                 1       0.06 %
```

Let us now show some of the classification errors:

- For example, it should be noted that images with a pointed shape and a prevalence of dark grey colours are predicted as Knives.

- While images with more blue tones are predicted as Sparkling_Water.





- Finally, some images containing multiple objects are usually predicted as Party_Mix_Snack, Plastic_Food_Container or Jams.

It is noticeable that most of the proposed errors come from outliers and noises, which are, as I said in the pre-processing phase, the most difficult samples to classify for their being inherently wrong from the beginning. Despite this, the proposed model still manages to achieve a brilliant accuracy.

6. Conclusion

In conclusion, with this homework I managed to obtain a particularly efficient neural network having a validation accuracy of 90%, a rather high result considering the imperfections both of the dataset and of the model. I add that obviously the proposed neural networks used to learn these eight classes could have been identically used to solve the initial general problem of 180 classes: the only difference would have been a greater need for space, time and computational capacity.