Machine Learning - HW2

Gianfranco Romani 1814407

December 23, 2020

1 Introduction

The problem faced in this report concerns the classification of objects that can be easily found in every house. I will describe my attempts to use several neural networks to classify as correctly as possible the images provided. After a description of the dataset and of the preprocessing I used, I will report the results I got from the neural networks I decided to use (LeNet, AlexNet and InceptionV3) and some final comments on the trainings.

2 Dataset

The dataset that has to be used for this homework is called *RoboCup@Home-Objects dataset* [1]. I was assigned to work on the following 8 classes of the dataset: chopsticks, dinner_plate, grocery_bag, jams, juice_carton, papayas, potato_chips_pringles, refill_sponges.

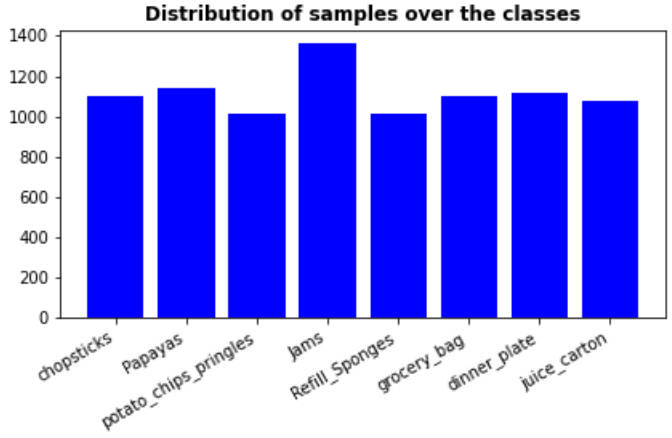


Figure 1: Distribution of samples

The images are distributed among the classes in a quite balanced way, as shown by the histogram in the image above (3). The total number of samples is 8927. Unfortunately, not all the images in the dataset are correct, in the sense that in some folders there are objects that are not of that class. This is going to inevitably worsen the performance of the models. I think it can be important to analyse the classes one by one to better understand the mistakes committed by my models (3):

1. chopsticks: there are not mistakes in this class, in fact it is one of the best classified classes. Sometimes there are several pairs of chopsticks in a single image, but this does not seem a big problem;

2. dinner_plate: also for this class there are not mistakes. The images are all very similar (generally dimensions and colors of the plates change). The majority of the plates are rounded, but there are also other shapes;

3. grocery_bag: this class caused several problems to the models because the bags in the images are very different one from the other, some of them area just drawings and often there are also other objects in the images (fruits, vegetables, other bags). Example of difficult image to classify: 2a;

4. jams: this class is the one with more wrong images. Besides images of the fruit conserve, there are images of James Bond, clothes, JAMS (I think they are some products used to build houses), cars (probably 007's cars) and so on (images 2b and 2c are good examples). Obviously models will have more problems to classify correctly this class compared to the others;

5. juice_carton: the shape for objects of this class is similar in almost every image, also the range of colours is limited, but not all samples are photos, some of them are drawings (2d);

6. papayas: the majority of samples represents this fruit in three ways: cut into two pieces, still in one piece and still on the trees. The colours that are predominant are orange and green;

7. potato_chips_pringles: the samples are very similar one from the other, indeed this is the class that is classified more easily;

8. refill_sponges: sponges in the samples are variegated, i.e. shapes and colours can change a lot between images.

I expect to obtain worse results from the classes with more problems (jams, grocery_bags and refill_sponges) , compared to, for example, classes like dinner_plates.

(a) *Different bags in a single image (with also some food)*

(b) *Picture of James Bond in Jams class*

(c) *Picture of a house in Jams class*

(d) *Drawing of a glass of juice*

Figure 2: Examples of bad samples (wrong or more difficult to predict)

(a) *Chopsticks*

(b) *Dinner plate*

(c) *Grocery Bag*

(d) *Jams*

(e) *Juice carton*

(f) *Papayas*

(g) *Potato chips pringles*

(h) *Refill sponges*

Figure 3: Examples of good samples for each class

Size of the samples varies from (113,160) to (300,300), so I will train my neural networks with inputs in this range.

3 Preprocessing

Since the dataset was not already divided into train and test set, I used the library *split-folders* [2] to divide the given dataset into three subset: train, validation and test set. The ratio I decided to use for this split is 80%, 10%, 10%. After that, I applied some data augmentation [3]. More precisely I focused into rotations and shifting (both vertical and horizontal) of the samples, but I did not interfere with the colors since I believe this could be harmful for classes in which colors are distinctive for the objects (papayas for example). Also for this reason I used as color mode 'rgb'. Shuffle is set as true for the train set, false the others (using true also for validation decreased enormously performances). For the majority of the tests I used target_size = (256,256) but I have made also some tests with smaller dimensions, 128x160, only with leNet. For the firsts tests I decided to make use of a batch size of 128, to obtain faster trainings, but then I reduced it to 64 to improve results.

4 Trainings

This section is divided into three parts, one for each method I used.

All the tests were performed in Google Colab environment, enabling GPU (it should be a Tesla K80 GPU) to speed up trainings. The first neural network I used is leNet because of its

simplicity, compared to other networks, that can help me to have a first view on the problem without spending to much time in trainings, as could happen with the other neural networks I wanted to use. After some tests with leNet, I decided to employ AlexNet, that is a bit more complex, to try to improve the results obtained before and finally I made some attempts with transfer learning on InceptionV3, considering that the previous methods were far from being optimal. For every training I implemented three callbacks:

1. EarlyStopping, that allows to avoid overfitting by stopping the training when one parameter is not improving anymore. In my case, I decide to monitor the accuracy on the validation set and to assign to the *patience* parameter the value 2, so it will wait for two epochs to stop the training;

2. ModelCheckpoint, that saves the weights that obtained the best results during the training;

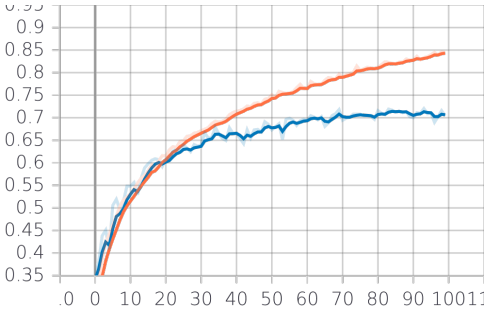
3. TensorBoard, that helps to visualize in a clearer way the statistics of the trainings (accuracy and loss).

The maximum number of epochs is always 100 (never reached, except when I disabled early stopping), whereas the number of iteration (obtained by dividing the number of training samples by the batch size) is 111 when the batch size is 64, 55 in the other scenario (batch size = 128).

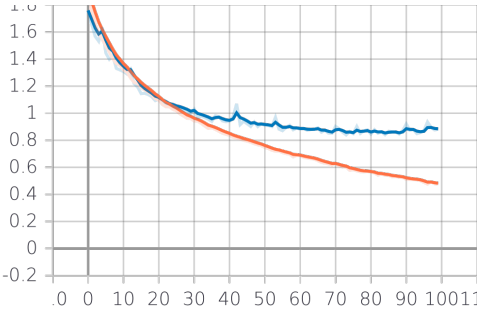
4.1 LeNet

For this neural network I have made three experiments. In the first one I used a batch of 128 samples, *tanh* as activation function and Adam as optimizer. The results were bad to say the least: the accuracy was stuck at ≈ 0.15, even when early stopping was disabled and the training reached 100 epochs. So I decided to change optimizer and activation function and I chose SGD and ReLU function. The training was stopped after 29 epochs, when the accuracy had reached 0.67 and val_accuracy 0.65 (loss a little below 1.00). Letting the training reach 100 epochs would have brought to better performances (0.84 in accuracy, 0.70 in accuracy in validation, 0.74 on the test set) but noticeable overfitting. Time spent per epoch was averagely of one minute (little more than one hour and a half for 100 epochs). The last test I made on leNet consisted in this last version with a smaller input, 128x160 instead of 256x256. I hoped to see improvements in time without losses in accuracy, but the results were not great.

In figure 4 there are the statistics of the training on the second tests I made on leNet, the one with SGD, ReLU, and bigger input. As clearly shown by 4c, there are classes on which the results are quite good and others that are often misclassified. The wrongly classified classes are the ones indicated in the first section as 'possibly problematic' plus the juice_carton class, that has a particularly low precision (that means that samples of other classes are considered as belonging to this one, i.e. there are a lot of false positive). Objects of these difficult classes are often confused as belonging to other classes by the model. Loss is good for both training and validation (obviously there is a discrepancy in the values obtained as in accuracy).



(a) *Accuracy*



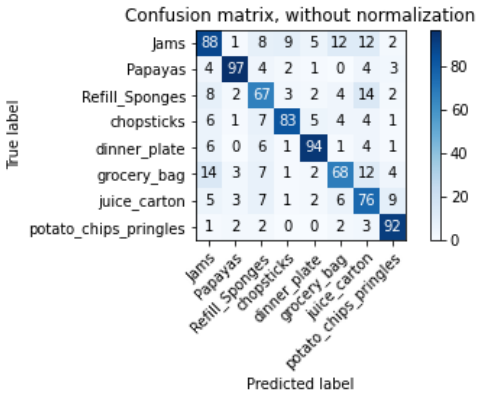
(b) *Loss*

```
15/15 [==============================] - 2s 142ms/step
                    precision    recall  f1-score   support

              Jams       0.667     0.642     0.654       137
           Papayas       0.890     0.843     0.866       115
     Refill_Sponges      0.620     0.657     0.638       102
         chopsticks      0.830     0.748     0.787       111
       dinner_plate      0.847     0.832     0.839       113
        grocery_bag      0.701     0.613     0.654       111
       juice_carton      0.589     0.697     0.639       109
potato_chips_pringles    0.807     0.902     0.852       102

           accuracy                          0.739       900
          macro avg      0.744     0.742     0.741       900
       weighted avg      0.743     0.739     0.740       900
```

(c) *Confusion metrics*



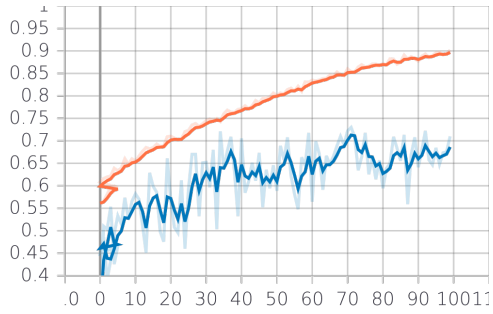
(d) *Confusion Matrix*

Figure 4: Statistics of the training on leNet with SGD and ReLU (256x256 images)

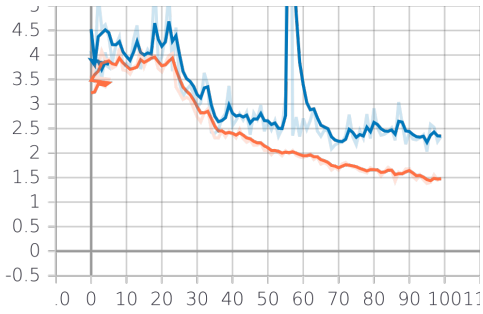
4.2 AlexNet

From the trainings on this model I expected better performances compared to leNet, but the results are not convincing. Input size is 256x256 for all the tests and batch size is 64. I started using a learning rate of 0.0001, a l2 regularization (Ridge Regression) set to 0.0001 and SGD as optimizer. With this configuration the improvements were very slow and the training was stopped after only 8 epochs with an accuracy of approximately 0.4. Increasing the learning rate to 0.001 helped to reach 0.45 in accuracy on validation set, while setting it at 0.01, with regularization to 0.001 led to a final score of ≈ 0.55 on the test set (in 11 epochs). Last thing I tried is using Adam instead of SGD for 100 epochs, but the overfitting is quite obvious (image 5). With this last version I obtained 0.90 in accuracy on the train set, but only 0.71 on the test set. Averagely each epoch took two and a half minutes of training (four hours for 100 epochs), with the version that uses Adam a bit faster than SGD, as expected.

As shown in figures 5c and 5d, the prediction are worse compared to leNet (4c and 4d) but mainly the classes that got worse are the ones that generally perform better (they still got better results but now they are closer to the 'bad' classes). 5a and 5b shows the strong overfitting of which the model suffers (along with a lot of noisy movements that could be caused by the datasets, maybe a bigger validation set could have be useful).



(a) *Accuracy*

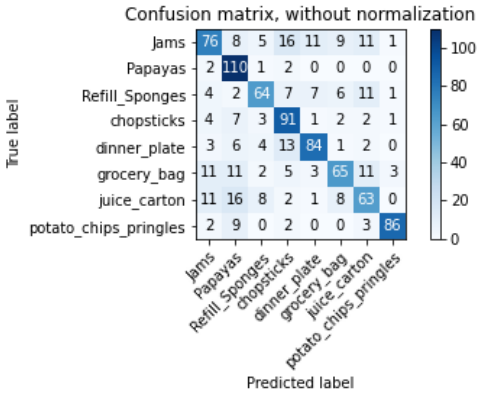


(b) *Loss*

15/15 [==============================] - 2s 154ms/step

	precision	recall	f1-score	support
Jams	0.673	0.555	0.608	137
Papayas	0.651	0.957	0.775	115
Refill_Sponges	0.736	0.627	0.677	102
chopsticks	0.659	0.820	0.731	111
dinner_plate	0.785	0.743	0.764	113
grocery_bag	0.714	0.586	0.644	111
juice_carton	0.612	0.578	0.594	109
potato_chips_pringles	0.935	0.843	0.887	102
accuracy			0.710	900
macro avg	0.721	0.714	0.710	900
weighted avg	0.717	0.710	0.706	900

(c) *Confusion metrics*

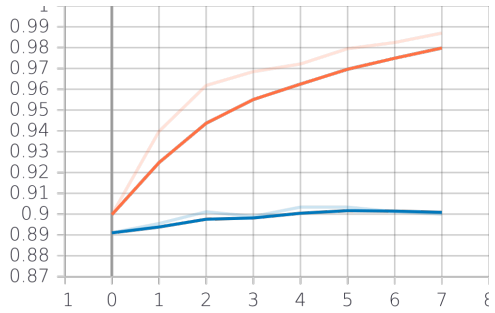


(d) *Confusion Matrix*

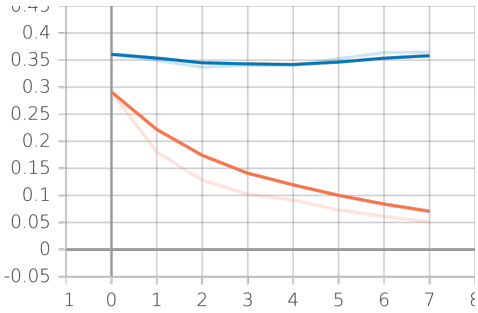
Figure 5: Statistics of the training on AlexNet with Adam, lr=0.01 and l2 regularization (256x256 images)

4.3 InceptionV3

As last method I decided to experiment with transfer learning on a bigger network, so, after some failed tests with ResNet50 and ResNet152, I used InceptionV3 that got very good performance. The imported weights are pre-trained on ImageNet and top layers of the model are not included. Then I added a GlobalAveragePooling2D to extract the most important features, and two Dense layers. Before compiling the model, all the convolutional layers have to be freezed. The model is then trained for a few epochs (10) with RMSProp as optimizer, obtaining $\approx .85$ in accuracy on validation set. At this point, to further improve performances I decided to use fine-tuning, i.e. train again the model after freezing a part of its layers (in my case the firsts 249). In this case the optimizer is SGD. The first training is needed because it would be bad to train the model with some layers pre-trained (the frozen ones) and others randomly-initialized (the final ones). From Keras documentation [4] I read that it is recommended to use small learning rate because I only needed to adjust a little the weights and using a big learning rate could cause overfitting if the dataset is small (generally transfer learning is used on small datasets, since with a small amount of data is very difficult to train a model from scratch). Our dataset is not too small, so I decided to try to use a learning rate of 0.001 with a momentum of 0.9.



(a) *Accuracy*



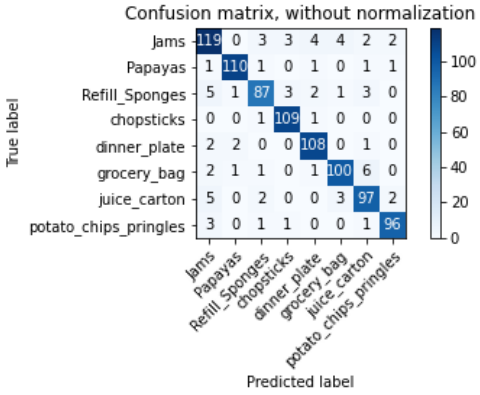
(b) *Loss*

```
15/15 [==============================] - 4s 199ms/step
                      precision    recall  f1-score   support

                Jams      0.869     0.869     0.869       137
             Papayas      0.965     0.957     0.961       115
       Refill_Sponges     0.906     0.853     0.879       102
           chopsticks     0.940     0.982     0.960       111
         dinner_plate     0.923     0.956     0.939       113
          grocery_bag     0.926     0.901     0.913       111
         juice_carton     0.874     0.890     0.882       109
potato_chips_pringles     0.950     0.941     0.946       102

             accuracy                          0.918       900
            macro avg     0.919     0.918     0.919       900
         weighted avg     0.918     0.918     0.918       900
```

(c) *Confusion metrics*



(d) *Confusion Matrix*

Figure 6: Statistics of the training on Inception

The results are great, after only 8 epochs the training stopped and the best value obtained on validation is 0.9034, while validation loss is 0.34. On the test set the model obtained 0.918 in accuracy. Looking at 6 is evident how much this method is better than the previous ones, it can obtain very good results also for the classes that before caused a lot of problems. The model was

also very quick in training, every epoch took about two minutes to be completed.

5 Comments

In conclusion, I think that the results obtained on this dataset are quite good, especially in the case of InceptionV3. I would like to point out that the results on the class 'Jams' can not be considered precise or even meaningful given the fact that the dataset for this kind of object is compromised by wrong images. Since the test set is taken from the original data, it also has wrong samples, so the statistics are erroneous. Apart from this case, I think that the other classes are generalized well and the tests could be considered as reliable. To solve the problem for the 'Jams' class it would be necessary to clean the dataset from the mistaken images and then train again the models (probably after inserting or generating new samples). Since InceptionV3 is the model that seems to generalize (way) better, I will deliver its weights with this report.

References

[1] https://sites.google.com/diag.uniroma1.it/robocupathome-objects/home

[2] https://pypi.org/project/split-folders/

[3] https://keras.io/api/preprocessing/image/

[4] https://keras.io/guides/transfer_learning/