Machine Learning – Homework 2

Andrea Gasparini
1813486

December 2020

Contents

1 Introduction

The goal of this Homework was to solve an image classification problem with objects typically available in a home environment.

2 Dataset

The dataset contains **8221** samples and it's based on a 8 classes subset of the RoboCup@Home-Objects dataset. For each class we have a folder containing all the related samples.

- *accent plate*
- *breadsticks*
- *cereals box*
- *cocoa drink bottle*
- *nitrile gloves*
- *paper bag*
- *pears*
- *plastic fork*

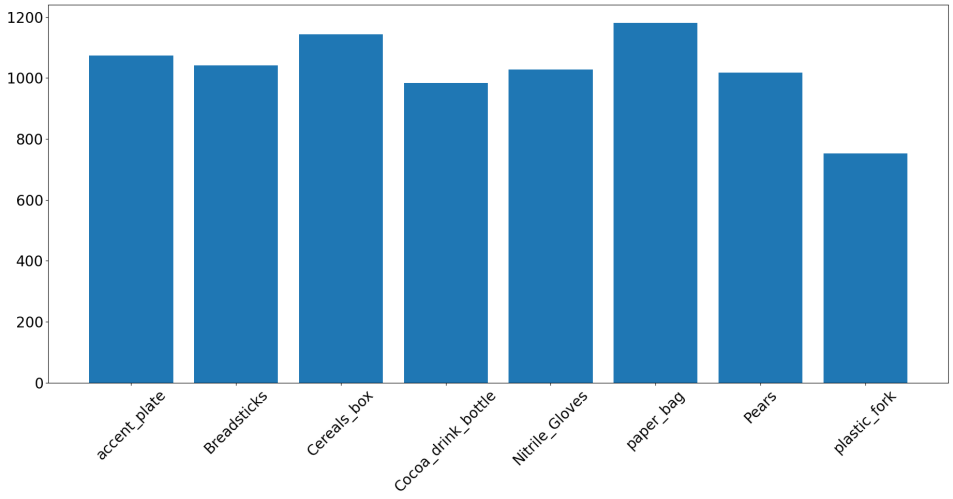

Figure 1: Distribution of the dataset's 8 classes

As we can see in Figure 1, the dataset is quite balanced, but it should be noted that it also contains outliers – i.e. misclassified images – meaning that our model should adapt accordingly.

Figure 2: 10 random samples from the dataset

3 Preprocessing

The first thing I did is the dataset splitting in two subsets for training and validation, respectively 70% and 30% of the original one. The splitting has been made at filesystem level, obtaining two directories (*train* and *val*) with the same structure of the original one (i.e. with 8 subdirectories, one for each class).

3.1 Data augmentation

Data augmentation includes techniques used to generate *new* training samples from the original ones by applying random jitters and transformations, but at the same time ensuring that the class labels of the data are not changed. These kind of techniques are useful to reduce overfitting and they usually provide an increase in testing accuracy, at the expense of a slight worsening in training accuracy. In our case I obtained an increase in the performance by 5 to 10% applying flipping, rotations and zooming to the training set of images.

4 Convolutional Neural Networks

All the tested solutions for this problem are based on a **Convolutional Neural Network** (CNN), commonly composed by convolutional layers, pooling layers and fully-connected layers. The **convolutional layers** apply a convolution operation to the input images, the **pooling layer** consequently perform a down-sampling through a non linear function and the **fully connected layers** finally classify in one of the 8 classes.

4.1 AlexNet

The first tested network is the implementation of AlexNet, a popular CNN, which has eight layers using the non-saturating `ReLU` activation function:

- five convolutional layers, some of them followed by a max-pooling layer
- three fully connected layers, each one followed by a Dropout layer

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 120, 60, 96)       34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 60, 30, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 60, 30, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 50, 20, 256)       2973952
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 10, 256)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 25, 10, 256)       1024
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 23, 8, 384)        885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 8, 384)        1536
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 6, 384)        1327488
_________________________________________________________________
batch_normalization_3 (Batch (None, 21, 6, 384)        1536
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 19, 4, 256)        884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 2, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 2, 256)         1024
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              18878464
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
batch_normalization_7 (Batch (None, 1000)              4000
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 8008
=================================================================
Total params: 45,913,552
Trainable params: 45,892,416
Non-trainable params: 21,136
_________________________________________________________________
```

Figure 3: Summary of AlexNet

4.2 Custom network

After the evaluation of AlexNet, the main analysis has been made with a custom network, in which it was possible to test different layers and hyper-parameters. For example adding dropout layers or changing the dropout rate. The optimal configuration obtained is showed in Figure 4, all of them using the `ReLU` activation function except for the last one with `softmax`

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 250, 250, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 125, 125, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 62, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 61504)             0
_________________________________________________________________
dense (Dense)                (None, 128)               7872640
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 1032
=================================================================
Total params: 7,897,256
Trainable params: 7,897,256
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Summary of the custom network

5 Evaluation

After some test with different epochs values I fixed it at 20 for the final evaluation, which was the optimal value for performance and training time. To compare the results, in this section are available the accuracy and loss curves and also the classification report for both AlexNet and the custom network. The accuracy curve shows how the network accuracy increase over every epoch, while the loss curve shows how the loss decrease (hopefully).

5.1 AlexNet

As we can see in Figure 5, while the accuracy of the training increase and the loss decrease for every epoch, the validation results are very poor. The gap between the training curve and the validation curve also measure how much our model is prone to overfitting (the model adapt too much on the data in the training while performing poor on the overall data available).

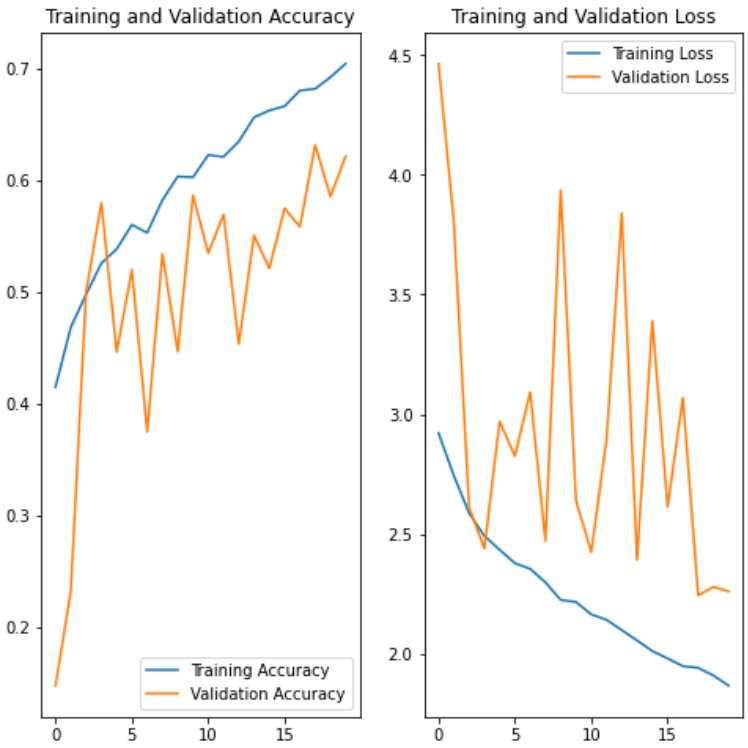

Figure 5: AlexNet accuracy and loss plots

	Precision	**Recall**	**F1-score**	**Support**
breadsticks	0.776	0.674	0.721	313
cereals box	0.863	0.584	0.697	344
cocoa drink bottle	0.554	0.449	0.496	296
nitrile gloves	0.738	0.628	0.678	309
pears	0.596	0.758	0.668	306
accent plate	0.585	0.258	0.358	322
paper bag	0.812	0.293	0.431	355
plastic fork	0.239	0.850	0.373	226
Accuracy			0.546	2471
Macro avg	0.645	0.562	0.553	2471
Weighted avg	0.666	0.546	0.558	2471

Table 1: Classification report of AlexNet

5.2 Custom network

The following results for the custom network are based on the configuration with a dropout rate of 0.4, which is the optimal one compared to other values. The performance has been increased and regularized compared to the AlexNet ones that were unbalanced in precision and recall (e.g. the *plastic fork* class had a low precision of 0.239 and a high recall of 0.850).

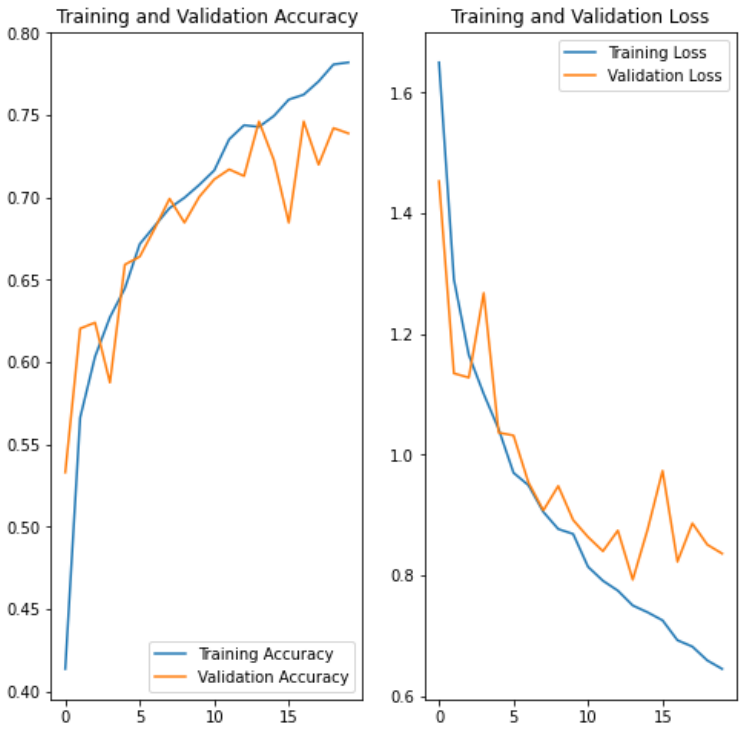

Figure 6: Custom network accuracy and loss plots

	Precision	**Recall**	**F1-score**	**Support**
breadsticks	0.816	0.693	0.750	313
cereals box	0.612	0.872	0.719	344
cocoa drink bottle	0.569	0.514	0.540	296
nitrile gloves	0.722	0.799	0.759	309
pears	0.724	0.788	0.754	306
accent plate	0.797	0.770	0.784	322
paper bag	0.711	0.507	0.592	355
plastic fork	0.651	0.602	0.652	226
Accuracy			0.696	2471
Macro avg	0.700	0.693	0.690	2471
Weighted avg	0.702	0.696	0.692	2471

Table 2: Classification report of the custom network

6 Conclusions

The performances have been unexpectedly better with the final custom network compared to AlexNet, as highlighted by the accuracy and loss curves. The more likely reason is that a network like this needs to have a larger dataset and a bigger number of epochs to obtain good results. In fact I got better results with 100 epochs, but that required more than 1 hour of training.