Report Homework n°2

Fabrizio Casadei

Matriculation number: 1952529

December 12, 2020

Chapter 1

Overview

The aim of this project is to solve a classification problem based on images.
For doing this, we have used 2 types of implementations, both with the use of *Convolutional Neural Network*:

- CNN made from scratch
- Use of *Transfer Learning* on the problem, hence, use a pre-trained CNN and applies it to our problem

In this report will be discussed how these two models have been created, especially for the scratch one, and then we will see the results measured.
Finally, in the last chapter, there will be a discussion and a comparison of what we have previously presented.
Now, we are going to describe information about the common configuration of the two models, which has been used for handling the input data and compute their evaluation.

1.1 Strucuture of the project

1.1.1 Dataset

The pictures are provided by *RoboCup@Home-Objects dataset* that has been developed in the *RoboCup@Home* competition.
More precisely has been used a subset of that Dataset, with a random sampler which has given class names to use in this project. The class drawn are:

- Crackers
- Dinner_plate
- Dust_clothes
- Muesli_box
- Paper_bag
- Pears
- Plastic_spoon
- Tea_drink_bottle

The total number of images for this Dataset is equal to 8683, and the 20% has been used for test and the remaining 80% for the training of the model. The batch used is composed of 64 of these samples, which are been resized to 118×224 resolution.

1.1.2 Evalution of models

After having properly divided Dataset into two parts, we have labelled it with the tags "training" and "validation", in this way the framework used (*Keras*), can provide to us a lot of operations based on this configuration. The *train_generator* and the *validation_generator*, after having defined the models, has been used for training and evaluating the performance of these.
Mainly 2 kinds of metrics have been used, the **loss** and the **accuracy**. These values are used to build up the graphs that we will ses in results sections. The loss function used the is **Categorical crossentropy**.
Furthermore, also *Precision*, *Recall*, *f1-score* and *support* evaluation will be provided.

1.1.3 Optimizer

The chosen optimizer for the first model is **Adam** while for the second, is the **Gradient descent optimizer**.
The purpose of this optimizer is to find internal model parameters which perform well against some performance metrics concerning the loss.

Chapter 2

First solution

In this chapter, we're going to describe the definition of a *Convolutional Neural Network* ant its consequent training.
A CNN can be described in terms of layers, what types of layers are been defined? How many layers have been used? (how much is deep?), how many parameters made up the CNN? Are all trainable or only a bunch?
The aim is to answer at all this question, in a way of having a clear idea of its structure.

2.1 CNN structure

The first layer of the net is connected directly with the dataset, so we have a shape equal to 118×224×3. The 3 is given by the fact of having images with 3 channels for representing the colour, hence, *rgb channels*.
In some layers, we have used 2 kinds of parameters that's a good choice discuss.
Has been chosen the value "valid" for the *padding* operation and the relative parameter, so input volume is not zero-padded and the spatial dimensions are allowed to reduce via the natural application of convolution.
Another important aspect is the use of an activation function, which are mathematical equations used to normalize and for determining whether each neuron's input is relevant for the model prediction. In this case, two types have been utilized:

- *Relu*, flattens all the negative results to zero, while leaving everything unchanged for values greater or equal to zero.
  This function has been used in several layers as the fully connected ones and the convolutional (more detail in the table below)

- *Softmax*, returns a vector of probabilities, from a vector of numbers (input).
  This choice has been used only for the last layer.

In this CNN, two types of Pooling has been used, the *Max polling* and the *Average polling*, both for reducing the size of net as going along to it.
In the last part of the net, we have used also a *Dropout* feature, which inserts the random element in the net, dropping nodes in a stochastic matter. The group of "active" neurons changes during the training.

In the end, we pass from a 2D input, like a matrix of pixels that we have, to a 1D vector, easier to manage and more similar at the form of output that we want, which is

a set of classes and their probabilities, obtained through another Dense layer.
In total, the numbers of layers of this net is 10. Let's see in details a table which sums up the global structure:

Layer (type)	*Output Shape*	*Param*
conv2d (Conv2D)	(None, 114, 220, 20)	1520
conv2d_1 (Conv2D)	(None, 110, 216, 30)	15030
max_pooling2d (MaxPooling2D)	(None, 55, 108, 30)	0
conv2d_2 (Conv2D)	(None, 53, 106, 40)	10840
average_pooling2d	(None, 52, 105, 40)	0
flatten (Flatten)	(None, 218400)	0
dense (Dense)	(None, 100)	21840100
dropout (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 60)	6060
dense_2 (Dense)	(None, 8)	488

Let's show in more compact view the number of parameters and the trainable property of them:

- Total : 21,874,038
- Trainable : 21,874,038
- Non-trainable : 0

A final comment on the structure is that net is not very deep, but have an important number of parameters, hence, it's a short and large CNN. Tendentially this kind of NNs are not so efficient and easy to tune for good results. In fact, it's interesting to see a full different solution even in these terms.

2.2 The training

In this small section, we are going to talk about some important training characteristics. An import aspect to highlight is the number of epochs, that we anticipate, it's different between the two solutions.
In this case, has been chosen to use 100 epochs for training.
The value of the *batch size* and the *steps_per_epochs* remains the same for both the solutions. The first one has been set to 64, and the other one is equal to:
$generator.n//generator.batchsize$, where n is the number of samples, and the generator can be either the *train_generator* or the *validation_generator*.

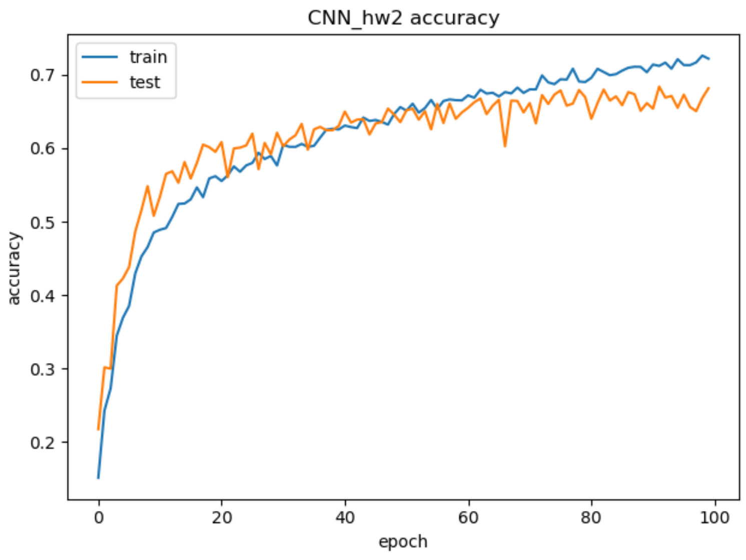
2.3 Results

The evaluation of this model has been done in several terms, as discuss previously. Let's start showing the classification report with information about *Precision*, *Recall*, *f1-score* and *support*.

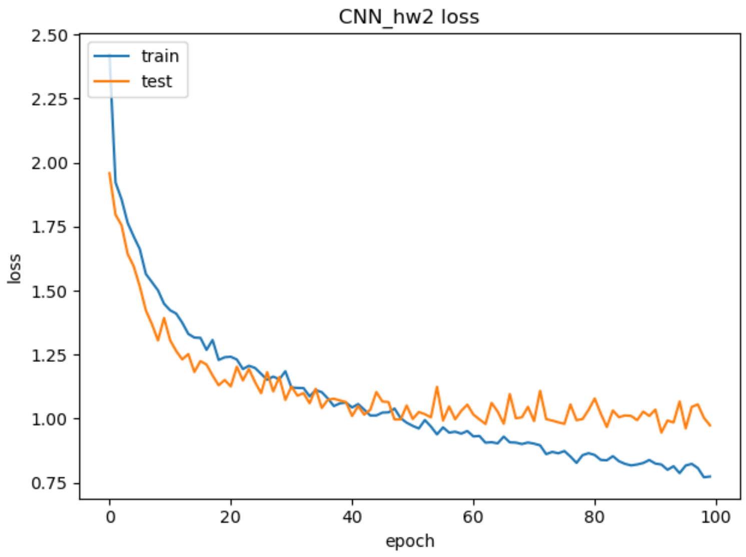
	precision	*recall*	*f1-score*	*support*
Crackers	0.697	0.644	0.669	261
Dust_Cloths	0.541	0.473	0.505	281
Muesli_box	0.381	0.713	0.497	115
Pears	0.877	0.739	0.802	241
Tea_drink_bottle	0.682	0.722	0.701	187
dinner_plate	0.834	0.750	0.790	248
paper_bag	0.775	0.753	0.764	243
plastic_spoon	0.558	0.608	0.582	158
accuracy			0.670	1734
macro avg	0.668	0.675	0.664	1734
weighted avg	0.692	0.670	0.676	1734

Then we present the final values of the loss and accuracy for the training and for the testing. Next are introduced the trends of those metrics along all the epochs.

- train accuracy $\simeq 0.72$
- train loss $\simeq 0.77$
- Test loss: $\simeq 0.98$
- Test accuracy: $\simeq 0.68$



Accuracy graph



Loss graph

The last evaluation result that we are going to expose, are the values of error committed during the testing, sorted from the greatest to the smallest. Also, it's given information about the percentage of specific classification error.

True	*Predicted*	errors	err %
Muesli_box	Crackers	60	3.46 %
plastic_spoon	Dust_Cloths	37	2.13 %
Dust_Cloths	plastic_spoon	32	1.85 %
Muesli_box	Dust_Cloths	28	1.61 %
Dust_Cloths	dinner_plate	27	1.56 %
Tea_drink_bottle	Dust_Cloths	26	1.50 %
paper_bag	Dust_Cloths	24	1.38 %
Crackers	Muesli_box	21	1.21 %
Crackers	Pears	21	1.21 %
Tea_drink_bottle	paper_bag	15	0.87 %
Muesli_box	paper_bag	15	0.87 %
Dust_Cloths	Pears	15	0.87 %
dinner_plate	Dust_Cloths	14	0.81 %
plastic_spoon	Pears	14	0.81 %
Dust_Cloths	paper_bag	14	0.81 %
Dust_Cloths	Tea_drink_bottle	12	0.69 %
Crackers	Dust_Cloths	11	0.63 %
Dust_Cloths	Crackers	11	0.63 %
Muesli_box	Tea_drink_bottle	10	0.58 %
Tea_drink_bottle	dinner_plate	10	0.58 %
Muesli_box	dinner_plate	9	0.52 %
Crackers	Tea_drink_bottle	8	0.46 %
paper_bag	plastic_spoon	8	0.46 %
paper_bag	Tea_drink_bottle	8	0.46 %

Pears	Dust_Cloths	8	0.46 %
Crackers	paper_bag	7	0.40 %
plastic_spoon	dinner_plate	7	0.40 %
dinner_plate	plastic_spoon	7	0.40 %
Muesli_box	Pears	7	0.40 %
paper_bag	dinner_plate	6	0.35 %
Pears	Crackers	6	0.35 %
plastic_spoon	Tea_drink_bottle	5	0.29 %
plastic_spoon	Crackers	5	0.29 %
dinner_plate	Crackers	5	0.29 %
plastic_spoon	paper_bag	5	0.29 %
Tea_drink_bottle	plastic_spoon	5	0.29 %
Pears	Tea_drink_bottle	5	0.29 %
Crackers	plastic_spoon	4	0.23 %
dinner_plate	Tea_drink_bottle	4	0.23 %
Muesli_box	plastic_spoon	4	0.23 %
Tea_drink_bottle	Pears	3	0.17 %
plastic_spoon	Muesli_box	3	0.17 %
dinner_plate	paper_bag	3	0.17 %
Tea_drink_bottle	Crackers	3	0.17 %
paper_bag	Crackers	3	0.17 %
paper_bag	Muesli_box	3	0.17 %
Dust_Cloths	Muesli_box	2	0.12 %
Pears	plastic_spoon	2	0.12 %
dinner_plate	Pears	2	0.12 %
dinner_plate	Muesli_box	2	0.12 %
Pears	dinner_plate	2	0.12 %
Pears	Muesli_box	1	0.06 %
Pears	paper_bag	1	0.06 %
Tea_drink_bottle	Muesli_box	1	0.06 %
Crackers	dinner_plate	1	0.06 %
paper_bag	Pears	1	0.06 %

Chapter 3

Second solution

In this second solution, we are going to apply a technique very useful in Machine learning, called **Transfer Learning**.
Using this approach, we use a pre-trained model, more precisely a CNN, which someone else has modelled and trained for us. The models used in transfer learning are very effective and reliable, hence, a lot of these have been used in official competitions, where a huge work on the structure and tuning it's been done.
The model chosen it's *ResNet50V2*, learned with the *ImageNet* Dataset, which is made up of 193 layers. For normalize this net to our purpose, we have added 3 layers:

- *Flatten* layer, for passing to a 1D vector
- *Batch Normalization* layer, to normalize the output values
- *Dense* layer with *softmax* activation, for the correct final output

All other layers of the Pre-trained net are been set as "untrainable", so only the last three layers will be trained. For this model will be shown different kinds of configurations. A lot of tuning has been made for reaching satisfactory results. The first trial has been done using the **Adam** optimizer and training the model for 100 epochs, the second reducing the epochs to 10, and the third changing the optimizer to **SGD**.

As for the first solution, let's now show briefly the number of parameters for this net, trainable and not:

- Total: 24,252,936
- Trainable params: 573,448
- Non-trainable params: 23,679,488

3.1 The training

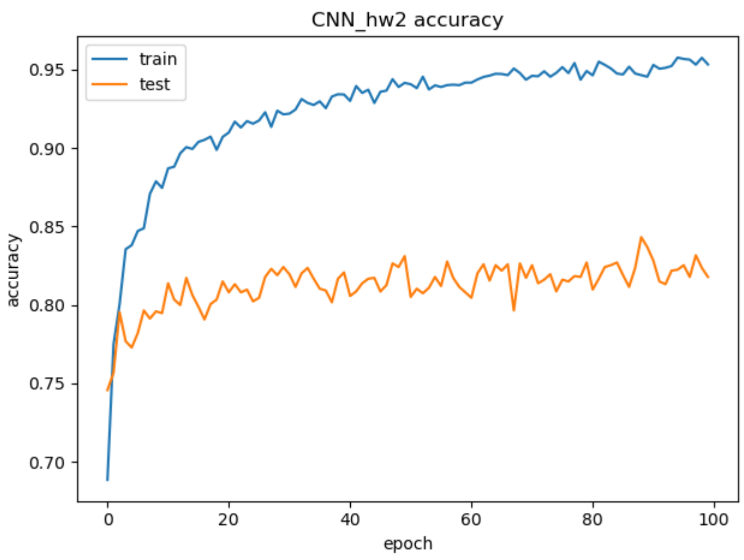
As said in previously, for this solution we have recorded different kinds of results, given by parameters tuning.
One of this, it's the number of epochs, in fact, in the first solution that we will see, this value has been fixed to 100, like in the CNN created from scratch. Then evaluating a strange and not satisfactory behaviour, has been chosen to decrease it's value to 10.
all other values of training are unchanged respect to the first solution.

3.2 Results

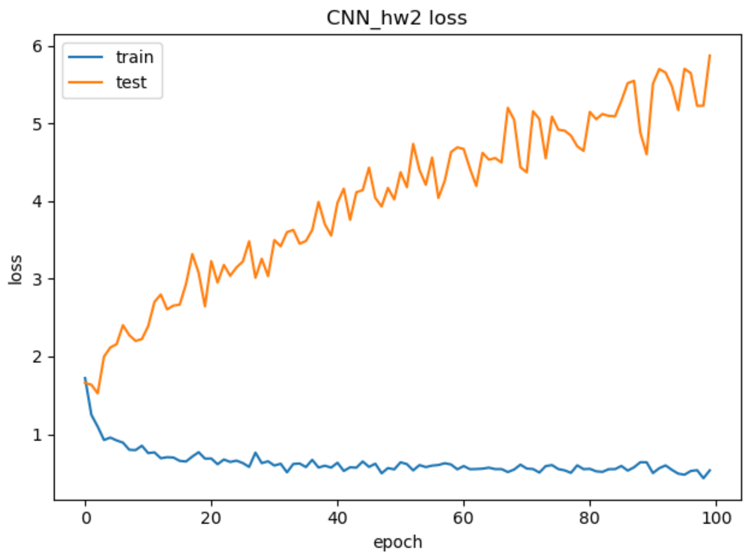
3.2.1 First configuration

Starting with the first configuration, we have reached the following results:

- train accuracy $\simeq 0.95$
- train loss $\simeq 0.53$
- Test loss: $\simeq 5.86$
- Test accuracy: $\simeq 0.81$



Accuracy graph



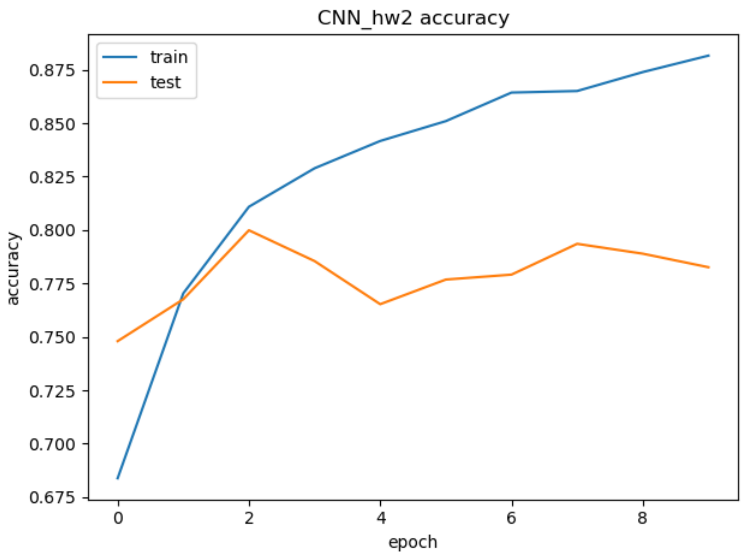
Loss graph

As we can see, these data give to us, a clear idea of something it's not going well as we would like.
The values of accuracy in training and testing are far, highlighting *overfitting* in the model.
Secondly, we can see, a continuous increase of the loss, epoch by epoch, and a final value very high.
Our intuition is trying to reduce this loss, decreasing the number of epochs.

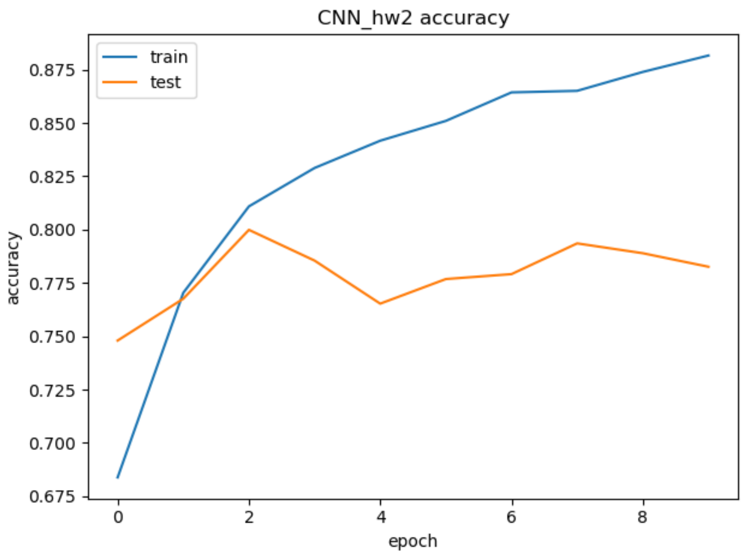
3.2.2 Second configuration

These are the results, after the change.

- train accuracy $\simeq 0.90$
- train loss $\simeq 0.27$
- Test loss: $\simeq 2.47$
- Test accuracy: $\simeq 0.79$



Accuracy graph



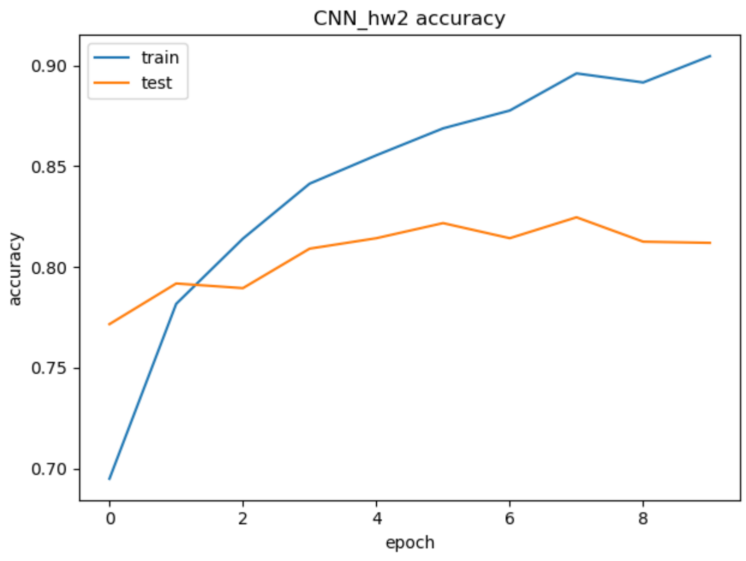
Loss graph

After reducing the number of epochs, the loss value is improved, but still too high, and the overfitting seems almost unchanged. In case, like this one, where the loss increase during the testing and we have an overall high loss value, it's probably a good choice trying to change the optimizer, passing to *SGD*.

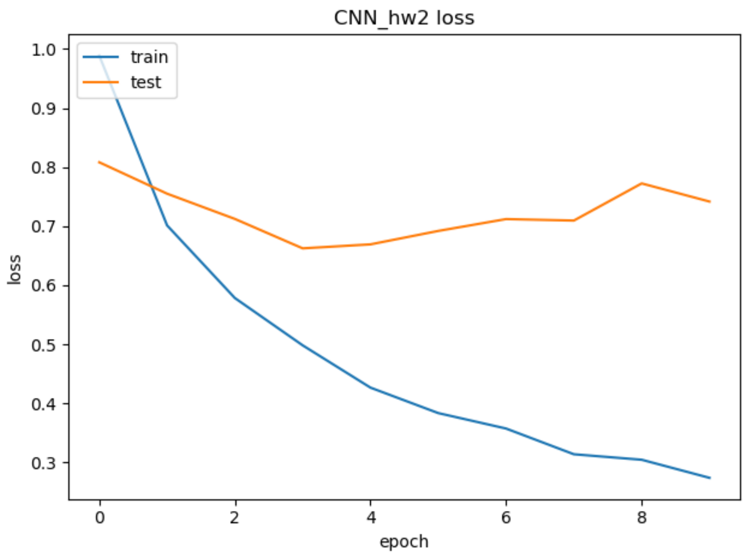
3.2.3 Third configuration

The Third and last configuration we will see, it's the result of for learning for 10 epochs, and using *SGD* optimizer.

- train accuracy $\simeq 0.90$
- train loss $\simeq 0.27$
- Test loss: $\simeq 0.67$
- Test accuracy: $\simeq 0.83$



Accuracy graph



Loss graph

Now the results are more satisfactory, and in line with the expectation.
We can see how the loss remain stable in an acceptable value of 0.67, especially if compared with the one recorded with the first configuration, that was almost 6.
We notice a reduction of the overfitting as well. Hence, we can use this configuration for evaluating the model. Starting from the classification report, we have obtained these results:

	precision	*recall*	*f1-score*	*support*
Crackers	0.817	0.770	0.793	256
Dust_Cloths	0.740	0.788	0.763	231
Muesli_box	0.665	0.688	0.676	208
Pears	0.911	0.894	0.902	207
Tea_drink_bottle	0.843	0.879	0.861	190
dinner_plate	0.915	0.911	0.913	224
paper_bag	0.911	0.878	0.894	245
plastic_spoon	0.843	0.838	0.841	173
accuracy			0.829	1734
macro avg	0.831	0.830	0.830	1734
weighted avg	0.831	0.829	0.830	1734

Finally, we have recorded these classification errors:

True	*Predicted*	errors	err %
Muesli_box	Crackers	31	1.79
Dust_Cloths	Muesli_box	21	1.21
Crackers	Muesli_box	17	0.98
Muesli_box	Dust_Cloths	14	0.81
Dust_Cloths	Crackers	13	0.75
Dust_Cloths	plastic_spoon	10	0.58
Dust_Cloths	paper_bag	9	0.52
Tea_drink_bottle	Muesli_box	9	0.52
Muesli_box	Tea_drink_bottle	9	0.52
paper_bag	Dust_Cloths	9	0.52
Tea_drink_bottle	Dust_Cloths	9	0.52
Muesli_box	paper_bag	8	0.46
Dust_Cloths	Tea_drink_bottle	7	0.40
Crackers	Dust_Cloths	7	0.40
Crackers	plastic_spoon	6	0.35
Crackers	paper_bag	6	0.35
dinner_plate	Pears	6	0.35
plastic_spoon	Muesli_box	6	0.35
Pears	Crackers	6	0.35
Pears	Muesli_box	5	0.29
dinner_plate	Dust_Cloths	5	0.29
Tea_drink_bottle	dinner_plate	5	0.29
plastic_spoon	Pears	5	0.29

paper_bag	Muesli_box	4	0.23
plastic_spoon	Dust_Cloths	4	0.23
Pears	plastic_spoon	4	0.23
Crackers	Pears	4	0.23
plastic_spoon	dinner_plate	4	0.23
Muesli_box	Pears	4	0.23
Muesli_box	dinner_plate	4	0.23
Tea_drink_bottle	Crackers	3	0.17
plastic_spoon	Crackers	3	0.17
paper_bag	plastic_spoon	3	0.17
paper_bag	Tea_drink_bottle	3	0.17
dinner_plate	Muesli_box	3	0.17
plastic_spoon	paper_bag	3	0.17
Tea_drink_bottle	paper_bag	3	0.17
Dust_Cloths	dinner_plate	2	0.12
Dust_Cloths	Pears	2	0.12
Muesli_box	plastic_spoon	2	0.12
Crackers	dinner_plate	2	0.12
Crackers	Tea_drink_bottle	2	0.12
Pears	dinner_plate	2	0.12
plastic_spoon	Tea_drink_bottle	2	0.12
dinner_plate	Crackers	2	0.12
dinner_plate	plastic_spoon	2	0.12
Tea_drink_bottle	plastic_spoon	1	0.06
dinner_plate	paper_bag	1	0.06
paper_bag	Crackers	1	0.06
paper_bag	dinner_plate	1	0.06
Tea_drink_bottle	Pears	1	0.06
Pears	Dust_Cloths	1	0.06
Pears	paper_bag	1	0.06 %
Tea_drink_bottle	Muesli_box	1	0.06 %
Crackers	dinner_plate	1	0.06 %
paper_bag	Pears	1	0.06 %

Chapter 4

Conclusion

In the first solution how it's possible to see, we have a little of *overfitting*, but considering the huge number of epochs for the training, it's understandable and bearable.
The values of accuracy and loss are not so good, how expected, but they aren't totally bad, about 70 per cent of accuracy for a not deep net, made from scratch, and a challenging classification problem, like this one, is globally a good result.
Another positive aspect that we can appreciate from the accuracy and loss graphs, is the total absence of negative value peak, the model it's slowly increasing its performance, without oddity of any types.
The training time in this case, has been very long, with a lot of parameters to learn, even if the CNN is not so deep.
For the second solution, we have seen, after some changes, how the results are been improved, especially if compared with the ones found in the first.
We have more overfitting respect to the first one, but the overall results are better. Accuracy and loss values are satisfiable for our problems, and even the timing for learning it's very shorter respect to the 100 epochs training of the CNN build by scratch.
The second net is deeper respect to the first, 194 layers versus only 10 layers, while the number of parameters it's not so different 24k versus 21k.
Distribution of the 21k parameters, seen in the first solution, handled "in-depth" instead of "in-breath", would have led to better performance.