

Homework 2

Silvia del Piano

1759992

delpiano.1759992@studenti.uniroma1.it

1 Assignment Description

This assignment requires to solve a problem of image classification with the usage of neural networks. This topic is particularly important, as it plays a big part in the scientific field called “computer vision”. More generally, this is an interdisciplinary field that deals with how computers can gain high-level understanding from digital images or videos. It has many applications: from the diagnosis of a patient in medicine, to the autonomous vehicles’ driving. To complete this assignment I’ve taken inspiration from the code provided in the exercises shown in class, the tutorials on the Tensorflow website and from Keras documentation.


The code was written using Google Colab.

Since training neural networks is a very computationally expensive task, the hardware is important: to train the models it’s useful to require the GPU at runtime execution.

2 Dataset description and processing

The dataset contains images of objects typically present in a home environment: it’s the RoboCup@Home-Objects dataset, developed for the Robocup@Home competition. Unfortunately, the data collected is not always classified correctly, and it has a lot of noise, which inevitably affects greatly performance. Here there is the basic information:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 5893 images belonging to 8 classes.
Found 2529 images belonging to 8 classes.
Image input (256, 256, 3)
Classes: ['Olives_jar', 'Oranges', 'Push_Brooms', 'juice_carton', 'plastic_fork', 'serving_tray', 'snacks_Seeds', 'teacup']
Loaded 5893 training samples from 8 classes.
Loaded 2529 test samples from 8 classes.
Oranges
```



The classes that will be taken into consideration for classification are: Olive.jar, Oranges, Push_Brooms, juice_carton, plastic_fork, serving_tray, snacks_Seeds and teacup.

The data was loaded on Google Drive, then, to split it into train and validation set, it was used the “splitfolder” library, which was not originally installed in the virtual machine provided by Colab:

```
# Split data into train set and test set (to do only once)
!pip install split-folders

import splitfolders
splitfolders.ratio('/content/drive/My Drive/Homework_2_Dataset/',
output='/content/drive/My Drive/Homework_2_Dataset_Split/', seed=1337,
ratio=(.7, .3), group_prefix=None)

print('Dataset split')
```

and this will produce the following different sets in Google Drive:

```
train_set_dir = '/content/drive/My Drive/Homework_2_Dataset_Split/train'
test_set_dir = '/content/drive/My Drive/Homework_2_Dataset_Split/val'
```

Data is the main problem of this type of training: a huge amount is needed to provide good results. Therefore, data augmentation was used to increase the number of samples.

```
# Data augmentation
train_datagen = ImageDataGenerator(
    rescale = 1. / 255, \
    zoom_range=0.1, \
    rotation_range=10, \
    width_shift_range=0.1, \
    height_shift_range=0.1, \
    horizontal_flip=True, \
    vertical_flip=False)
```

3 Neural Networks used

Both models considered are convolutional neural networks. Convolutional neural networks are characterized by the fact that they apply the operation of convolution:

Continuous functions

$$(x * w)(t) \equiv \int_{a=-\infty}^{\infty} x(a) w(t - a) da$$

Discrete functions

$$(x * w)(t) \equiv \sum_{a=-\infty}^{\infty} x(a) w(t - a)$$


but in reality, the operation that is implemented in machine learning libraries is cross-correlation (called "convolution"):

Cross-correlation

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

The first model taken into consideration is a simple LeNet model. This is a convolutional neural network, and it was proposed for the first time by Yann LeCun in 1989.

These are the main features of this network: - every convolutional layer includes three parts: convolution, pooling and non-linear functions - tanh activation function - the output function is softmax - the padding is valid, which means that the output depends on the kernel size The structure is the following:



```

LeNet model
  C1: Convolutional 6 kernels 5x5
  S2: Average Pooling 2x2 stride 2x2
  C3: Convolutional 16 kernels 5x5
  S4: Average Pooling 2x2 stride 2x2
  C5: Convolutional 120 kernels 5x5
  F6: Fully connected, 84 units
  F7: Fully connected, 10 units
Model: "sequential"

```

| Layer (type) | Output Shape | Param # |
|------------------------------|----------------------|----------|
| conv2d (Conv2D) | (None, 256, 256, 6) | 456 |
| average_pooling2d (AveragePo | (None, 128, 128, 6) | 0 |
| conv2d_1 (Conv2D) | (None, 124, 124, 16) | 2416 |
| average_pooling2d_1 (Average | (None, 62, 62, 16) | 0 |
| conv2d_2 (Conv2D) | (None, 58, 58, 120) | 48120 |
| flatten (Flatten) | (None, 403680) | 0 |
| dense (Dense) | (None, 84) | 33909204 |
| dense_1 (Dense) | (None, 8) | 680 |
| Total params: 33,960,876 | | |
| Trainable params: 33,960,876 | | |
| Non-trainable params: 0 | | |

The other model considered is an AlexNet, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton. It contains eight layers:

the first five are convolutional, and some of them are followed by max-pooling layers. The last three are fully connected layers. As activation function in the hidden part it has the ReLu functions, which is a big improvement from the tanh and sigmoid, while in the output layer it uses the softmax function. The complete structure is omitted for space reasons.

4 Evaluation

The main parameters used to evaluate the model are the cross-entropy loss function and the accuracy. This data is visualized while training, in a report that features also other parameters, like precision, recall and f1-score, and in graphs that show their progress at each epoch. These graphs are particularly important, as they can give us an understanding of how these two values progress as training goes on. The first shows the model's accuracy, while the second the model's loss function values. The blue curve is for the train set, while the orange one for the test set.

Here are the definitions of the other parameters considered:

Precision: among all the classes taken into consideration, how many of them are classified correctly (true positives)

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Recall: among all the positives, how many of them are classified correctly (true positives)

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Accuracy: among all the predictions, which are correct

$$Accuracy = \frac{TruePositives + TrueNegatives}{TruePositives + FalsePositives + TrueNegatives + FalseNegatives}$$

The cross-entropy error function is defined as follows:

Cost function: Maximum likelihood principle (**cross-entropy**)

$$J(\theta) = E_{\mathbf{x}, \mathbf{t} \sim \mathcal{D}} [-\ln(p(\mathbf{t}|\mathbf{x}, \theta))]$$

where we assume to have Gaussian noise. In fact, we basically assume that we're learning the target function affected by some noise, and so finding the maximum likelihood it's equivalent to minimizing the mean squared error. In other words, we want to maximize the probability of picking a sample from the instance space and that the network provides the right value for it.

5 Training and Results

The following are the results of the models.

5.1 LeNet

Training phase (45 minutes):

```
Epoch 1/10
168/168 [=====] - 2349s 14s/step - loss: 3.2542 - accuracy: 0.1348 - val_loss: 2.0796 - val_accuracy: 0.1368
Epoch 2/10
168/168 [=====] - 90s 537ms/step - loss: 2.0822 - accuracy: 0.1412 - val_loss: 2.0763 - val_accuracy: 0.1281
Epoch 3/10
168/168 [=====] - 90s 537ms/step - loss: 2.0743 - accuracy: 0.1356 - val_loss: 2.0954 - val_accuracy: 0.1155
Epoch 4/10
168/168 [=====] - 90s 535ms/step - loss: 2.0821 - accuracy: 0.1285 - val_loss: 2.0768 - val_accuracy: 0.1285
Epoch 5/10
168/168 [=====] - 90s 535ms/step - loss: 2.0788 - accuracy: 0.1372 - val_loss: 2.0800 - val_accuracy: 0.1439
Epoch 6/10
168/168 [=====] - 90s 535ms/step - loss: 2.0836 - accuracy: 0.1354 - val_loss: 2.0787 - val_accuracy: 0.1439
Epoch 7/10
168/168 [=====] - 90s 538ms/step - loss: 2.0822 - accuracy: 0.1265 - val_loss: 2.0731 - val_accuracy: 0.1285
Epoch 8/10
168/168 [=====] - 90s 536ms/step - loss: 2.0765 - accuracy: 0.1363 - val_loss: 2.0805 - val_accuracy: 0.1400
Epoch 9/10
168/168 [=====] - 90s 533ms/step - loss: 2.0817 - accuracy: 0.1276 - val_loss: 2.0763 - val_accuracy: 0.1281
Epoch 10/10
168/168 [=====] - 90s 535ms/step - loss: 2.0779 - accuracy: 0.1409 - val_loss: 2.0753 - val_accuracy: 0.1356
```

It is possible to see slight improvements in the accuracy, however, the accuracy on the validation set remains very low. This leads to the poor results shown in the following images:

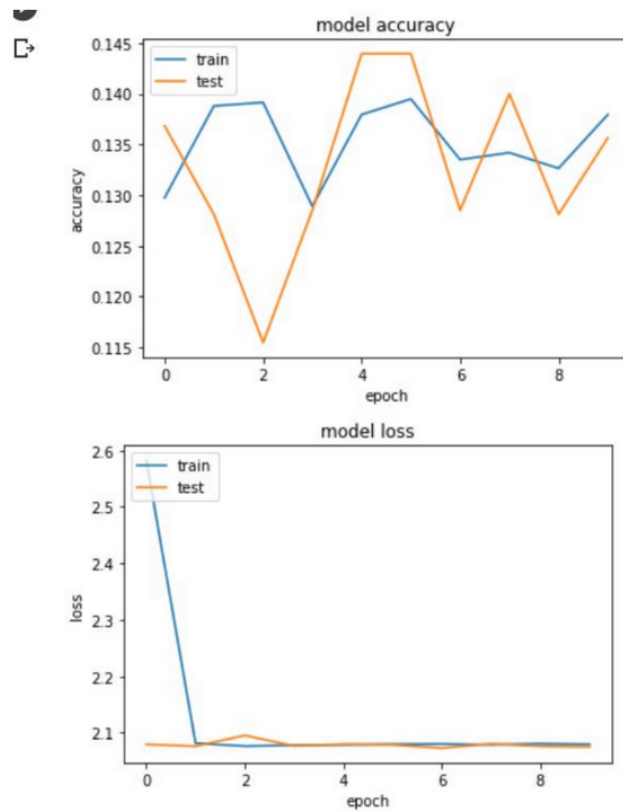
```
73/73 [=====] - 7s 97ms/step - loss: 2.0753 - accuracy: 0.1356
Test loss: 2.075273
Test accuracy: 0.135627

73/73 [=====] - 7s 97ms/step
      precision    recall  f1-score   support

Olives_jar      0.000      0.000      0.000       354
  Oranges       0.000      0.000      0.000       301
Push_Brooms     0.000      0.000      0.000       292
juice_carton    0.000      0.000      0.000       325
plastic_fork    0.000      0.000      0.000       226
serving_tray    0.136      1.000      0.239       343
snacks_Seeds    0.000      0.000      0.000       364
    teacup      0.000      0.000      0.000       324

    accuracy          0.136       2529
   macro avg      0.017      0.125      0.030       2529
  weighted avg      0.018      0.136      0.032       2529
```

It's possible to see that the results are terrible, with a slight improvement for what concerns the serving_tray class. In the image below it's shown how the accuracy fluctuates for both train set and test set, however, it never reaches a value above 0.15. In the second graph it's possible to notice an improvement of the loss function with respect to the training set after the first epochs, but after that the result stagnates and it never gets better, especially for the test set.



5.2 AlexNet

Training phase (20 minutes):

```

Epoch 1/10
168/168 [=====] - 95s 549ms/step - loss: 3.1966 - accuracy: 0.3185 - val_loss: 4.0257 - val_accuracy: 0.1981
Epoch 2/10
168/168 [=====] - 92s 547ms/step - loss: 2.7849 - accuracy: 0.4463 - val_loss: 4.0612 - val_accuracy: 0.2036
Epoch 3/10
168/168 [=====] - 92s 545ms/step - loss: 2.6556 - accuracy: 0.4882 - val_loss: 2.6418 - val_accuracy: 0.4603
Epoch 4/10
168/168 [=====] - 92s 545ms/step - loss: 2.5366 - accuracy: 0.5050 - val_loss: 2.6308 - val_accuracy: 0.5129
Epoch 5/10
168/168 [=====] - 92s 546ms/step - loss: 2.4939 - accuracy: 0.5061 - val_loss: 3.5926 - val_accuracy: 0.4808
Epoch 6/10
168/168 [=====] - 92s 546ms/step - loss: 2.3496 - accuracy: 0.5538 - val_loss: 2.9171 - val_accuracy: 0.4978
Epoch 7/10
168/168 [=====] - 91s 543ms/step - loss: 2.3292 - accuracy: 0.5533 - val_loss: 2.6294 - val_accuracy: 0.4618
Epoch 8/10
168/168 [=====] - 91s 544ms/step - loss: 2.2604 - accuracy: 0.5910 - val_loss: 2.2662 - val_accuracy: 0.5868
Epoch 9/10
168/168 [=====] - 92s 546ms/step - loss: 2.2500 - accuracy: 0.5851 - val_loss: 2.2103 - val_accuracy: 0.5947
Epoch 10/10
168/168 [=====] - 91s 545ms/step - loss: 2.2178 - accuracy: 0.6024 - val_loss: 5.4449 - val_accuracy: 0.2503

```

It's possible to note immediately that this model is better than LeNet: the accuracy reaches a substantially higher level: there is actually an improvement of the model as the epochs are executed. At epoch 10 there seems to be a principle of overfitting, as the accuracy on the train set grows, but drops significantly for

the validation set. The loss function exhibits the dual behaviour.
The results for the other parameters considered are the following:

```

73/73 [=====] - 7s 99ms/step - loss: 5.4449 - accuracy: 0.2503
Test loss: 5.444855
Test accuracy: 0.250297

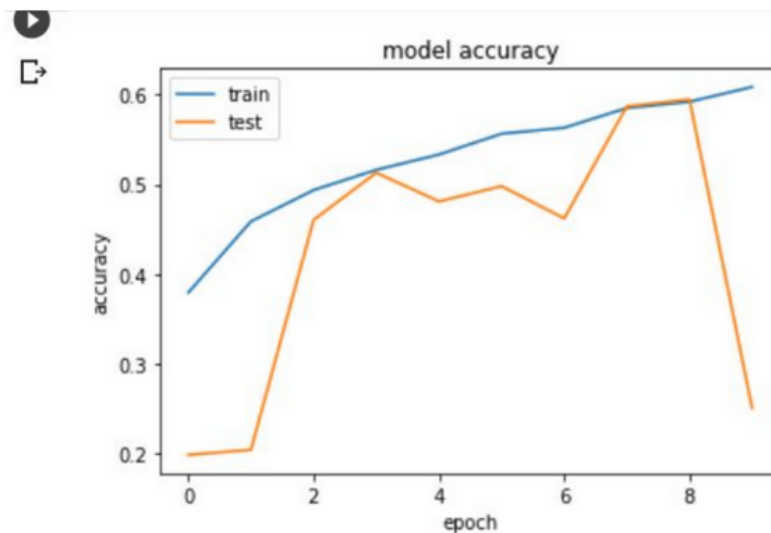
73/73 [=====] - 7s 98ms/step
      precision    recall  f1-score   support

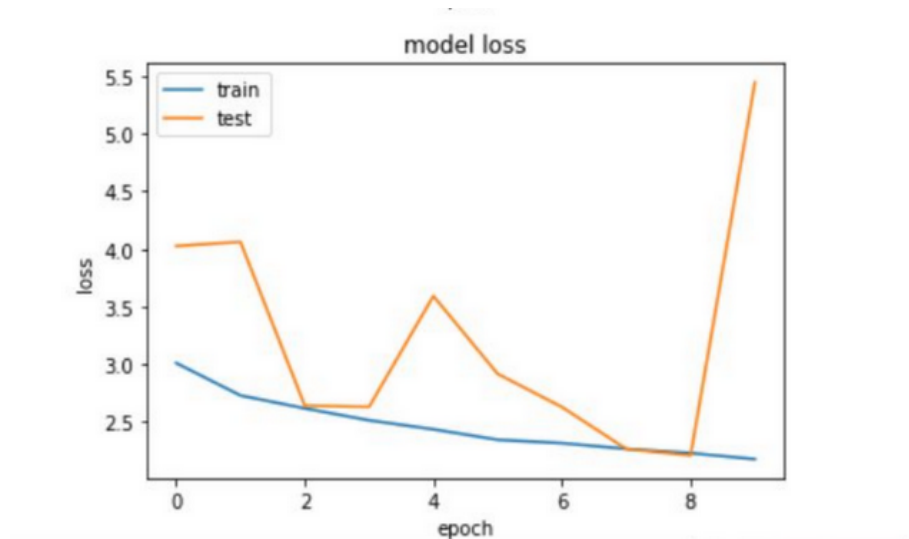
Olives_jar      0.780      0.260      0.390       354
Oranges          0.703      0.385      0.498       301
Push_Brooms      0.462      0.041      0.075       292
juice_carton     0.155      0.935      0.266       325
plastic_fork     0.207      0.155      0.177       226
serving_tray     0.909      0.058      0.110       343
snacks_Seeds     0.818      0.148      0.251       364
teacup           0.000      0.000      0.000       324

accuracy          0.250          2529
macro avg         0.504      0.248      0.221       2529
weighted avg      0.526      0.250      0.224       2529

```

It's possible to see how all parameters: precision, recall and f1-score have improved a lot with respect to the precedent model, although the accuracy seems to have not improved much: from 0.136 to 0.250, and the teacup class still presents very poor performances. In the graphs below it's easy to see how at epoch 10, as suspected, the model starts to overfit. While the accuracy on the training set continues to increase, for the test set drops. Naturally, the loss function exhibits the complementary behaviour.





6 Conclusions

This experiment has clearly shown how AlexNet is superior to LeNet. During the training phase, the first improves significantly, while the second can't pass the threshold of 0.15. AlexNet seems to arrive at a maximum accuracy of 0.59 before it starts to overfit. Even if this is a good result with respect to the other model, in absolute terms, is not a very good one. The same can be said for the other parameters: precision, recall and f1-score. AlexNet is definitely better than LeNet, but it has not a very good result. For LeNet, the class serving_tray is classified better than the others, while AlexNet has trouble to classify teacup. Furthermore, the time to train AlexNet is almost half of the time needed to train LeNet: 20 minutes vs 45 minutes.

Other regularization techniques, like weight decay and dropout could help to improve the evaluation parameters we considered, especially for AlexNet, as it could mitigate the overfitting phenomenon.