Report

Homework 2. RoboCup@Home Object Classification

Prepared by:
Sorokoletova Olga
1937430

14 Dec 2020

Abstract

The idea is starting from scratch with simple “toy” CNN **LeNet** try to upgrade it in order to obtain the best its performance and then use the result model as a baseline for comparison with strong architectures of **transfer learning** with the approach of fine tuning. So, there are not two, but many models to compare.

The best result obtained is showing 0.9 value of accuracy and f1-score more than 0.85 for each class. Final model is represented by transfer learning employing **GoogleNet** (version number 3: `keras.applications.inception_v3.InceptionV3`) trained over ImageNet dataset with the latest fine tuning.

Note: there are not two, but many models to compare, so the following suggestions in this report will be given in terms of a sequence of the experiments.

Problem

Given a dataset from Robocup@HomeObject competition to implement image classification with multiple classes.

Data

Input Data

8 zipped folders with images corresponding to the categories listed below:

```
containers/basket_container
tableware/snack_bowl
drinks/soft_drink_bottle
cutlery/dinnerware
fruits/Papayas
snacks/Crackers
food/Oatmeal_box
cleaning_stuff/Upholstery_Cleaners
```

Output Data

Weights of the best model obtained may be found by usage attached h5 file:
`1937430.h5`.

Pre-processing

The way of pre-processing varies depending on the model and also can be a part of some experiments, so details of some particular experiments with pre-processing can be found below, but the general approach is the same and will be described in this section of the report.

First of all, I am using `!unzip -q ./Data/\* -d ./imgs` util to unzip folders directly to the colab notebook in order to work with data in an efficient way.

Then I am importing `splitfolders` module to split images in the input folder into train and validation (test) folders. The ratio was in general $8./2.$ as a train/validation, however finally the changing it to $7./3.$ contributed to better performance.

The last step of pre-processing consists of usage of `ImageDataGenerator` for data augmentation resulting in training and test generators. The most common used augmentation for training set is following:

```
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;
```

The most common used choice for batch size hyperparameter is 16 and the most common used resizing of images is $(256,\ 256,\ 3)$. They also correspond to the final model.

The question of the resizing deserves special attention because as can be seen from the dataset we are facing the problem of having images of different dimensions and even with not the same aspect ratio as inputs in the task. Meanwhile, the transfer learning approach with fine tuning is to be considered, but there is only a fully-convolutional neural network that should be able to accept images of any dimension, otherwise (in the case where fully-connected layers are also included) this architecture does not allow accept images of different resolutions.

One common approach is to resize images and then crop them to the dimensions suitable for input of the network chosen for transfor, this method was employed in one of the experiments with VGG16 (code may be found in the notebook as commented). Another common approach is to resize each image to fixed dimensions. In most of the experiments resizing to $(256,\ 256)$ showed itself working better than alternatives. I also found a spatial pyramid pooling method, but did not try it.

Another interesting issue of pre-processing is that image folders contain many noisy data which expectedly restricts performance of any model. My experiments do not represent image noise handling because of the reason that as I think the best way of doing it is to take a dataset initially like unsupervised and apply to it a convolutional autoencoder for anomaly detection, but it is a part of the next homework. Probably, exactly noise deletion is the key thing to obtain better results of the performance.

Experiments

Experiment 1. LeNet

Experiment 1.1

Description: Standard LeNet in its initial configuration, 10 epochs, 32 batch size.

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

Time: 92 *sec*/*epoch* .

Validation accuracy: 0.2.

Comments: the value of accuracy $0.2.$ is an optimistic one, it hardly obtains this value recognizing images from only one class and with very low precision.

Experiment 1.2

Description: All activation functions in hidden layers replaced by relu.
Time: 95 *sec*/*epoch* .
Validation accuracy: 0.5.
Comments: although results are still too bad to be a solution of an image classification problem we can observe significant increase by replacing activation function in LeNet. Here and later the usage of relu as an activation function in LeNet is assumed.

Experiment 1.3

Description: SGD optimizer instead of adam.
Time: 98 *sec*/*epoch* .
Validation accuracy: 0.46.
Comments: convergence looks more stable, but rather slow, the results in general are worse though not so much.

Experiment 1.4

Description: Increasing the number of **kernels** in the convolutional layers (to the power of 2, 4 and so on.
Time: 103 *sec*/*epoch* .
Validation accuracy: 0.37.
Comments: could not help to obtain better performance.

Experiment 1.5

Description: Increasing the number of convolutional **layers.** Experiment is divided into two parts: 1) addition of two layers, the first of each is a copy of the second layer of a standard LeNet, and the second one is the same, but with $5x5$ pooling with the stride 5 for compatible dimensions reduction; 2) first part + two convolutional layers with 16 kernels and same padding.
Time: 95 *sec*/*epoch* .
Validation accuracy: 0.49.
Comments: the result of the first part did not give any growth in performance, the second part performed even worse and was interrupted by manual stop.

Experiment 1.6

Description: Batch size in [16, 128].

Validation accuracy: 0.51.

Comments: does not influence significantly neither time of execution nor performance measurements, but there and in the following experiments as well was noticed that batch size 16 keeps training slightly better. Since now batch size = 16 is supposed.

Experiment 1.7

Description: Varying the number of **epochs** using early stopping.

Validation accuracy: 0.61.

Comments: within 30 epochs represented validation accuracy was obtained, within 100 epochs training was interrupted by stopping callback at the 18*th* epoch with the same value of accuracy.

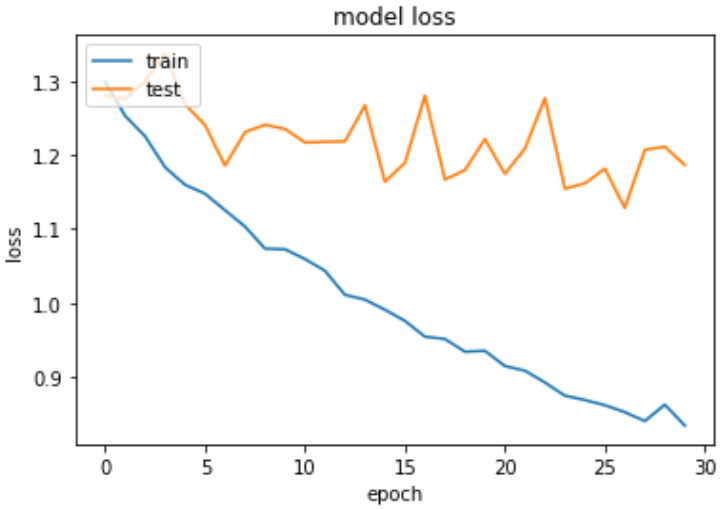
Final configuration

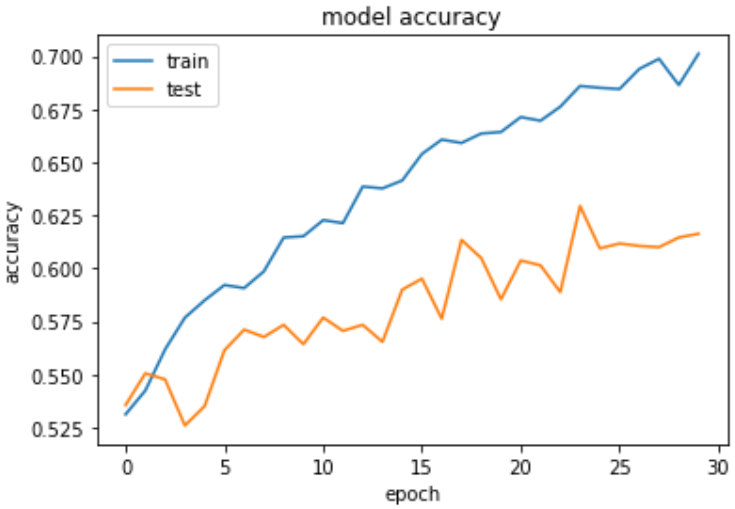
Standard LeNet with relu activation function:
Batch size: 16.
Number of epochs: 20.

Final results

Repetition of the experiments has shown that reached value of the validation accuracy is stable and hardly can be outperformed. Conclusion is that the best result which can be obtained with a simple and not deep architecture of the CNN (considering we are not doing any smart pre-processing for noise deletion) is ~60%. Overall time of execution is ~1800*s* (100*s*/*epoch*, 18 *epochs*). The class of the best guesses is always 'Papayas'.

	precision	recall	f1-score	support
Crackers	0.582	0.715	0.642	242
Oatmeal_box	0.535	0.409	0.463	208
Papayas	0.718	0.900	0.798	229
Upholstery_Cleaners	0.571	0.577	0.574	194
basket_container	0.539	0.517	0.528	238
dinnerware	0.665	0.620	0.642	192
snack_bowl	0.622	0.506	0.558	241
soft_drink_bottle	0.667	0.673	0.670	205
accuracy			0.616	1749
macro avg	0.612	0.615	0.609	1749
weighted avg	0.612	0.616	0.610	1749





Experiment 2. VGG16

Experiment 2.1

Description: Transfer learning using **VGG16** model trained on ImageNet dataset (weights are kept) with the **fine tuning approach** with last trainable layer of the network as trainable:

```
name_output_extractor = "block5_pool"
trainable_layers = ["block5_conv3"].
```

Follower layers are two fully-connected with 200 and 100 neurons, relu activation function, dropout by a factor 0.4 and batch normalization technique.
Optimizer: adam. Batch size: 16. Epochs: 10.

Time: 105 *sec/epoch* .
Validation accuracy: 0.81.
Comments: noticeable progress is obtained in the comparison with LeNet best model configuration.

Experiment 2.2

Description: The most suitable for VGG16 input way of **preprocessing**. 1) The images have to be of shape (224, 224) , so it is recommended to resize the images with

a size of $(256, 256)$, and then do a crop of size $(224, 224)$. 2) The colors are in BGR order.

1) For the cropping there are two functions which can be found as commented in the file with my code: `crop(img)` and `crop_generator(batches, crop_length)`. The idea is that the second one uses the first one and takes as input train (test) generator applying random crops from the image batches generated by the original iterator. Sets to be fed later on are obtained as follows:

```
train_crops = crop_generator(train_generator, 224)
test_crops = crop_generator(test_generator, 224)
```

2) VGG16 expects the color channels to be in B, G, R order, whereas Python by default uses B, G, R, but ImageDataGenerator and its method `flow_from_directory` do not support 'bgr' color mode. That is why a helper pre-processing function is needed.

Fortunately, various color spaces such as RGB, BGR, HSV can be mutually converted using OpenCV function `cvtColor(img, code)` which is employed in helper `bgr(img)` to return `cv2.cvtColor(img, cv2.COLOR_RGB2BGR)`. RGB to BGR transformation as well as crop of images can be found in the code attached as commented.

Experiment 2.3

Description: Usage of a **SGD** optimizer with Nesterov momentum instead of adam (over more epochs).

Experiment 2.3

Description: Change of a set of trainable **layers** and architecture of the layers following after the fixed convolutional part.

Experiment 2.4

Description: Transfer learning with VGG16 trained on ImageNet within an approach of exploiting it as a **feature extractor** for input of a simple classifier based on Shallow NN.

Experiment 2.5

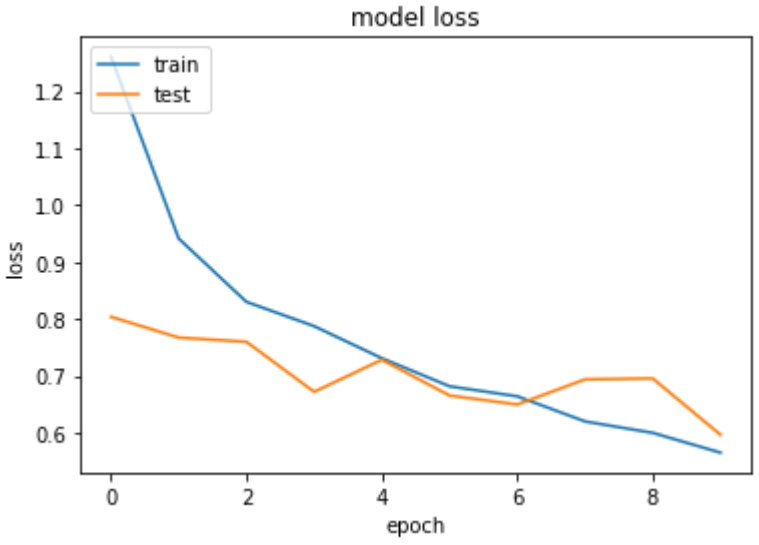
Description: Hyperparameters tuning.

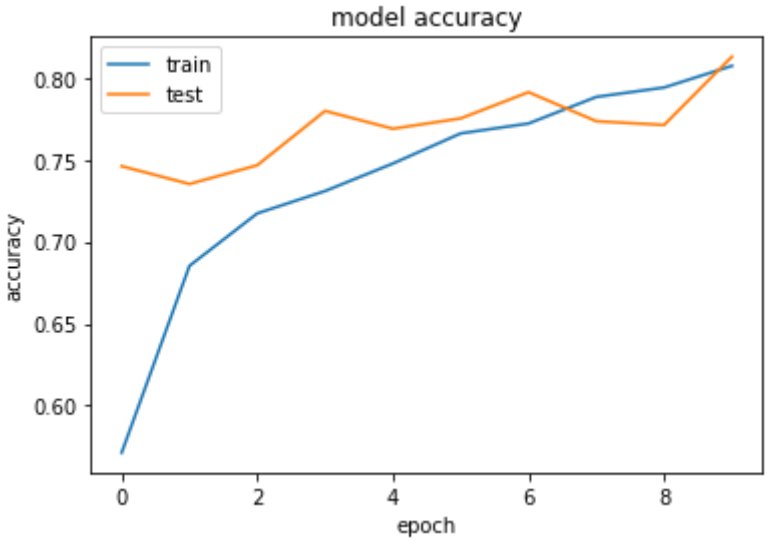
Final results

Contrary to expectations, none of the measures taken (even more suitable pre-processing) helped to significantly improve the result in terms of validation accuracy and raise it above baseline threshold ~80%. In general, execution does not require more than 10 epochs for convergence and takes ~1000*s* (100*s*/*epoch*, 10 *epochs*), except for the case with SGD optimizer which takes more epochs and ~125*s*/*epoch*. The most recognized class is still 'Papayas', two top-most problematic classes are 'Crackers' and 'Oatmeal_box'.

```
Epoch 10/10
436/436 [==============================] - 102s 235ms/step - loss:
0.5651 - accuracy: 0.8074 - val_loss: 0.5965 - val_accuracy: 0.8130
```

	precision	recall	f1-score	support
Crackers	0.819	0.731	0.773	242
Oatmeal_box	0.738	0.611	0.668	208
Papayas	0.870	0.939	0.903	229
Upholstery_Cleaners	0.737	0.866	0.796	194
basket_container	0.825	0.790	0.807	238
dinnerware	0.884	0.833	0.858	192
snack_bowl	0.758	0.834	0.794	241
soft_drink_bottle	0.877	0.907	0.892	205
accuracy			0.813	1749
macro avg	0.814	0.814	0.812	1749
weighted avg	0.813	0.813	0.811	1749





Although results show that there is still scope of work for improvement of a performance, in the experiment with the transfer learning via VGG16 we can obtain much better results of classification in comparison with the simple CNN and therefore a family of models related to the transfer learning seems promising for solving considered problem. The overall idea is to try other pre-trained architectures in searching for the best.

Experiment 3. AlexNet

AlexNet was involved in the completely pre-trained mode and had lower results than VGG (excluded from the code due to a lack of effectiveness), but executed faster.

Experiment 4. ResNet50

The idea behind using transfer learning with ResNet50 trained on ImageNet assumed its architecture as more powerful, so better performance than obtained by VGG was expected. There were a lot of experiments on it about hyperparameters setting, suitable pre-processing implementing and especially on the reconfigurations of the trainable/non-trainable and following by them layers. Probably, there was some key mistake in my approach to the realization, but finally none of the measurements taken obtain something reasonable to be represented in the code and in this report as well.

Experiment 5. GoogleNet

Experiment 5.1

Description: Transfer learning using **GoogleNet** (**inceptionv3** architecture of the third edition of the GoogleNet pre-trained on ImageNet) with the **fine tuning approach** with usage of output and retraining as follows:

```
name_output_extractor = "mixed10"
trainable_layers = ["conv2d_192", "batch_normalization_216",
"batch_normalization_224"]
```

The next layers after that are again two fully-connected with 200 and 100 neurons, relu activation function, dropout by a factor 0.4 and batch normalization technique, but during experiments other variations were also constructed, more deep or more simple as, for example, just `GlobalAveragePooling2D()` with `Dropout(0.5)`.

Optimizer: adam.
Experiments included also
`optimizer = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)` and `optimizer = SGD(lr=1e-4, momentum=0.9)`, which performed comparatively in the same way, but much more slower.

Pre-processing: common pre-processing described in the corresponding chapter of the report was employed.
As alternatives were train generator with the lower level of data augmentation involved just vertical flip and height/width shifts and train generator taking into account the fact that each Keras Application expects a specific kind of input preprocessing meaning for InceptionV3 call `tf.keras.applications.inception_v3.preprocess_input` on the inputs before passing them to the model is advisable.

Batch size: 16.
Epochs: 20 (motivated by early stops on the trials with larger values).

Time: 110 *sec*/*epoch* .
Validation accuracy: 0.85.
Comments: obtained results proved that GoogleNet transfer learning outperforms VGG16. Described in the 'definition' item alternatives did not supply further improvements.

Experiment 5.2

Description: The main difference from Experiment 5.1 is in the ratio of a split of the dataset folders. Now it is 7./3. (train/val) instead of 8./2. in all of the experiments represented above.
Time: 111 *sec*/*epoch* .
Validation accuracy: 0.90.
Comments: the last significant improvement was obtained by redistribution of the data between training and test sets.

Final results

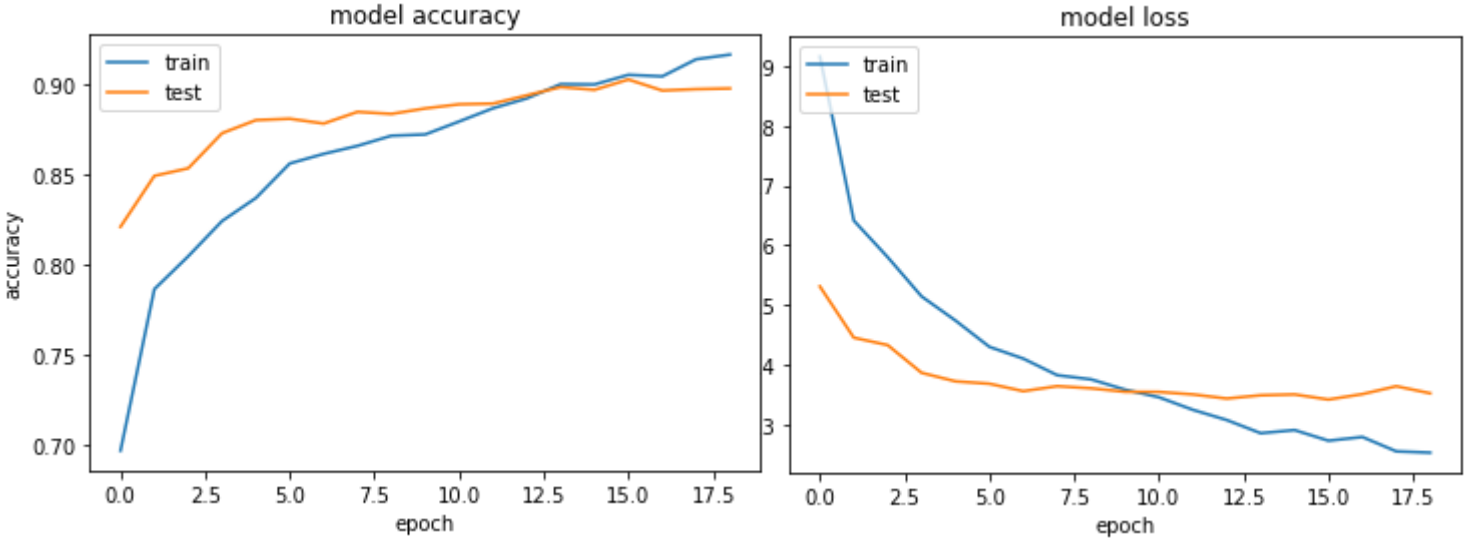
The best validation accuracy obtained is ~90%. Note: I sent the results with 0.89 , because I did not save the model obtained 0.90 and now I can recover since google colab has blocked me in terms of GPU usage for some time.

The overall time of execution within usage of google colab GPU *is* ~2100*s* (111*s*/*epoch*, 19 *epochs*) . Final configuration is described in the Experiment 5.1.Description and Experiment 5.2.Description sections.

	precision	recall	f1-score	support
Crackers	0.876	0.859	0.868	362
Oatmeal_box	0.853	0.856	0.854	312
Papayas	0.943	0.962	0.952	343
Upholstery_Cleaners	0.912	0.893	0.903	291
basket_container	0.911	0.894	0.902	357
dinnerware	0.959	0.899	0.928	287
snack_bowl	0.833	0.881	0.856	362
soft_drink_bottle	0.915	0.942	0.928	308
accuracy			0.898	2622
macro avg	0.900	0.898	0.899	2622
weighted avg	0.899	0.898	0.898	2622

The worst predicted classes are ‘Crackers’, ‘Oatmeal_box’ and ‘snack_bowl’. We can also observe that the most common mistakes are distributed amongst these classes.

Oatmeal_box	->	Crackers	22	0.84 %
basket_container	->	snack_bowl	18	0.69 %
dinnerware	->	snack_bowl	17	0.65 %
Crackers	->	Oatmeal_box	16	0.61 %
snack_bowl	->	Crackers	13	0.50 %
Crackers	->	snack_bowl	13	0.50 %



Conclusion

Transfer learning with CNN is a powerful approach to image classification problem handling. For this particular dataset based on the experiments represented fine tuned GoogleNet pre-trained on the ImageNet performs the best of all. It reaches a pretty high value of accuracy on the both training and validation datasets even though some classes are corrupted with a great percentage of noisy data. In order to obtain further improvements it seems reasonable to focus on the image anomaly detection for noise deletion.