Sapienza Università di Roma
Department of Computer, Control, and Management Engineering Antonio Ruberti
Artificial Intelligence and Robotics

Homework 2

Course: Machine Learning

Student: **Amila Sikalo**
Matricola: **1938032**

December 23, 2020

1 Introduction

The task for this homework was to classify images based on the data set provided by the ML team. As specified by the task I was provided with eight classes of images with corresponding datasets based on my Matricola code. These classes are:

- 'Carpet_Machine_Detergents',
- 'Melons',
- 'Seltzer_Water',
- 'food_tray',
- 'jarred_vegetables',
- 'potato_Crisps',
- 'soup_bowl',
- 'utensils'

2 Algorithms for image classification

For this homework I decided to try two approaches I considered feasible for this problem. The first one is provided in the Exercise 13, which builds an ensemble of three convolution neural networks to address the problem of image classification. The other approach is the one I found in Tensorflow documentation that uses convolution neural networks, but includes the data augmentation as well.

3 Configuration of classfiers

Both approaches use images rescaled to $32 \times 32px$ divided into batches of 32 for Tensorflow/Keras.

3.1 Ensemble of CNNs

For this approach I decided to keep the parameters I found in the Exercise 13, since that gave satisfiable results in that particular case. These parameters are:

- Three convolutional layers,
- Kernel size = 3,
- Activation = ReLU,
- Strides = 1.

The output is flattened with dropout (0.4) included to reduce overfitting. Outputs are dense with softmax activation. Adam is chosen as a optimizer, while categorical cross-entropy is chosen as loss function. The output of model summary is:

```
Model: "ensemble_model"
----------------------------------------------------------------------------------------------------
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             [(None, 32, 32, 3)]   0

----------------------------------------------------------------------------------------------------
model (Functional)               (None, 8)             101544      input_1[0][0]

----------------------------------------------------------------------------------------------------
model_1 (Functional)             (None, 8)             101544      input_1[0][0]

----------------------------------------------------------------------------------------------------
model_2 (Functional)             (None, 8)             101544      input_1[0][0]
====================================================================================================
Total params: 304,632
Trainable params: 304,632
Non-trainable params: 0

----------------------------------------------------------------------------------------------------
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
input_4 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 16)        448
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 4104
=================================================================
Total params: 101,544
Trainable params: 101,544
Non-trainable params: 0
_________________________________________________________________
```

This model was trained with 10 epochs.

3.2 CNN with data augmentation

In order to compare the ensemble of CNNs, I choose another method, which is a single CNN with data augmentation and normalization. This was based on the advice given in Tensorflow documentation for reducing overfitting.

These parameters are:

- Three convolutional layers,
- Kernel size = 3,
- Activation = ReLU,
- Strides = 1.

The parameters in this case are:

- Three convolutional layers of sizes $\{16, 32, 64\} \times 3$ with ReLU activation
- MaxPooling2D for each layer
- Dropout (0.2)

The outputs are flatted with added one 128 size Dense layer and one Dense layer with size 8 (for our classes). Again, adam was chosen as optimizer and sparse categorical cross-entropy as loss function. Data augmentation is done by random flipping, random rotation and random zooming of images. Moreover, Tensorflow recommends autotuning buffer sizes and caching input datasets.

The output of model summary is:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 32, 32, 3)         0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 32, 32, 3)         0
```

```
_________________________________________________________________
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 128)               131200
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 1032
=================================================================
Total params: 155,816
Trainable params: 155,816
Non-trainable params: 0
_________________________________________________________________
```

This model was trained with 10 epochs.

4 Results

The results are presented in similar fashion as in Exercise 13. This means that the accuracy is determined for validation dataset, but I also used scikit learn performance metrics such as accuracy precision and confusion matrix, in the same way I used them in HW 1.

4.1 Ensemble of CNNs

The results for this approach are:

```
Single models test accuracy:  [0.5951024889945984, 0.589407742023468, 0.5831435322761536]
Ensemble test accuracy:  0.13496583143507973
      precision  recall   f1-score   support

0         0.11      0.11      0.11       181
1         0.16      0.19      0.17       239
2         0.13      0.08      0.10       226
3         0.13      0.11      0.12       227
4         0.15      0.14      0.15       239
5         0.12      0.10      0.11       195
6         0.10      0.14      0.12       202
7         0.17      0.17      0.17       247

accuracy                            0.13      1756
macro avg       0.13      0.13      0.13      1756
weighted avg    0.14      0.13      0.13      1756
```

Confusion matrix is given in figure 1.

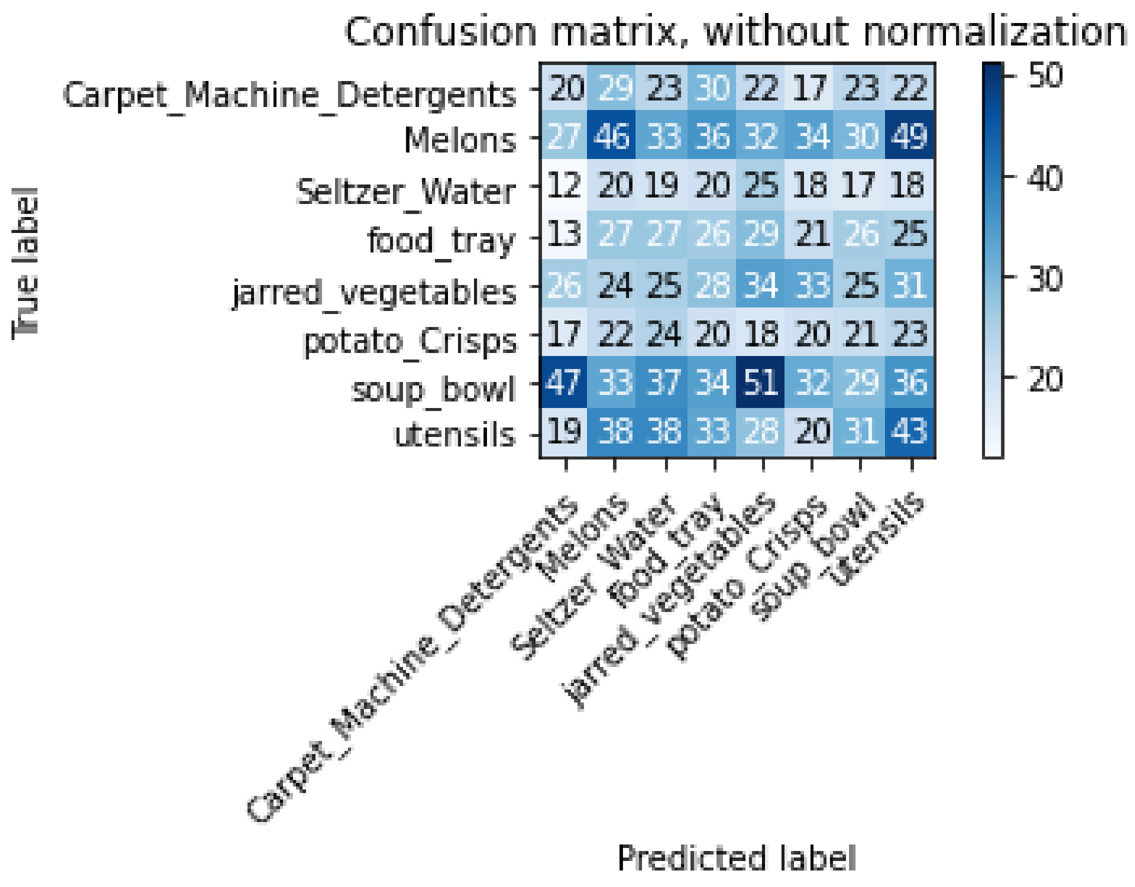

Figure 1: Confusion matrix for the Ensemble CNN image classifier

The historical evolution of accuracy during the training of this model is given in figure 2.

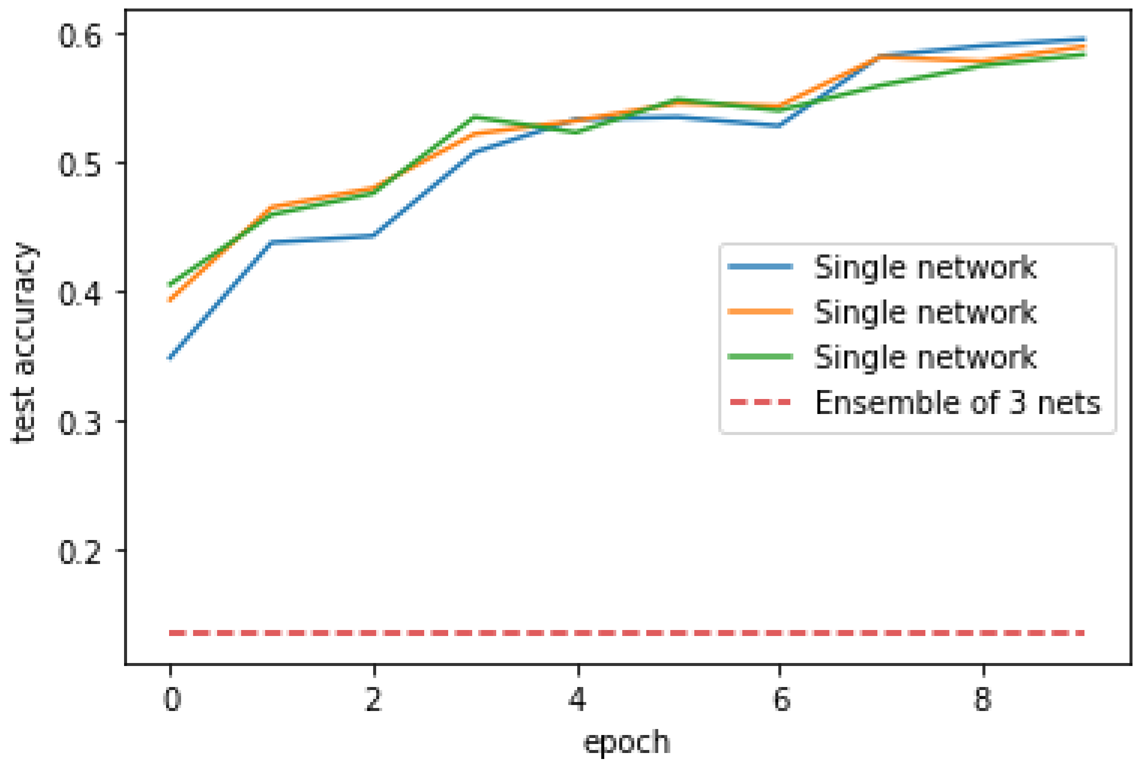

Figure 2: Accuracy history during training

4.2 CNN with data augmentation

The results for this approach are:

```
                precision   recall   f1-score   support

0                  0.51      0.85      0.64       181
1                  0.73      0.38      0.50       239
2                  0.52      0.58      0.54       226
3                  0.43      0.47      0.45       227
4                  0.48      0.69      0.57       239
```

```
5                   0.57      0.46      0.51       195
6                   0.87      0.54      0.67       202
7                   0.68      0.56      0.61       247

accuracy                                0.56      1756
macro avg           0.60      0.57      0.56      1756
weighted avg        0.60      0.56      0.56      1756
```

Confusion matrix is given in figure 3.

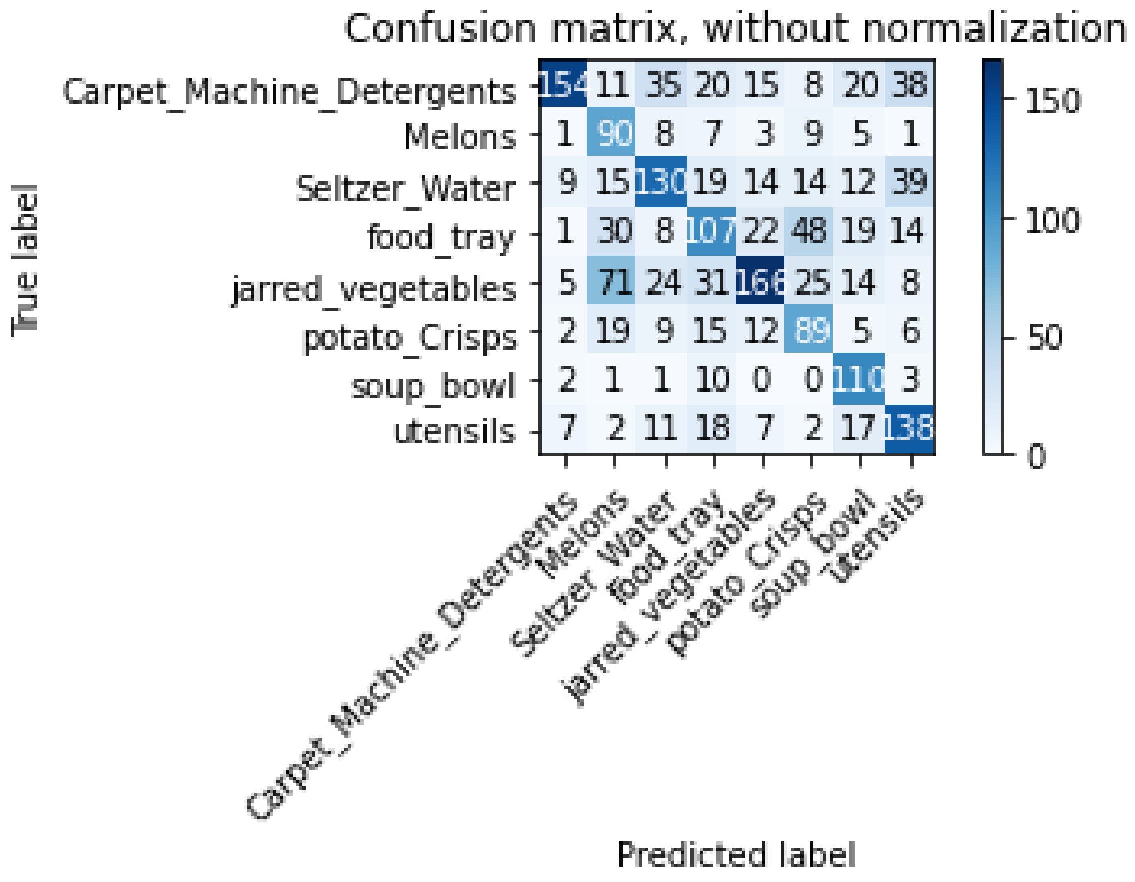

Figure 3: Confusion matrix for the CNN (with data augmentation) image classifier

The historical evolution of accuracy and loss function during the training of this model is given in figure 4.

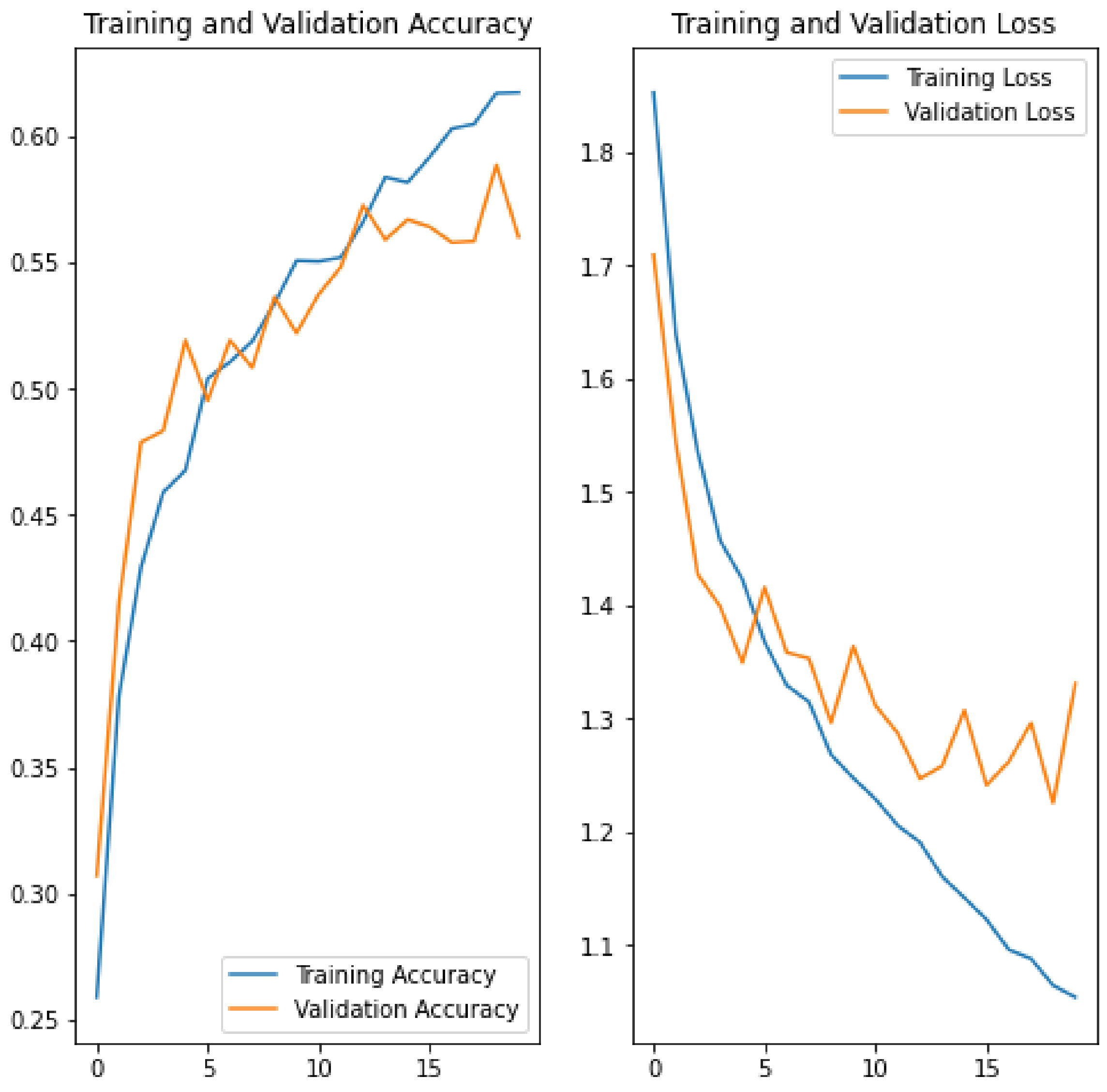

Figure 4: Accuracy and loss function history during training

5 Discussion

In both approaches CNNs are properly trained and gave roughly 60% of accuracy for training data set. However, the ensemble of CNNs approach have the accuracy of validation data set less than 15%. This is quite problematic, since discrepancy between training dataset and validation dataset accuracy suggest significant overfitting. No normalization and data augmentation strategies improve this. Moreover, it seems that the approach of taking the mean value of determined data classes seems to make a problem. For example, if two of three CNNs return class 6 and the last one returns class 1, the final result will be the mean value of these three outputs, which is around 4 (4.333). This renders wrong results, but can be overcome by implementing some type of voting system in order to resolve this problem.

The second approach with single CNN with data augmentation and normalization achieves more satisfiable result. This is especially the case in terms of reduced overfitting, which can be seen in the figure 4. In this case, the discrepancy between accuracy of training dataset and validation dataset is small. Moreover, the loss function plot for both datasets is decreasing, which suggests that neural network is properly trained with reduced overfitting and can achieve comparable results with new data.

6 Conclusion

In this homework I decided to classify images from the datasets I received with two different, but similar approaches. The first approach use ensemble CNNs while the other uses single CNN, but with date augmentation. Both approaches normalize the images to pixel values in $[0, 1]$. These are good strategies for reducing overfitting in data classification. The second approach gave much better results, although the first approach can be much more improved.

All the code is written in Python3, using Tensorflow on Google Colab platform with GPU enabled processing. Scikit-learn performance metrics are used to analyze the results.

1 Appendix - PDF outputs of Google Colab notebooks