MACHINE LEARNING

Homework 2

Report

RoboCup@Home Object Classification

Worked by: Alkid Halili

Matricola : 1956856

Table of contents

Project Overview

In this homework, I am invited to provide a solution for the classification of images regarding objects in a home environment.

The dataset is composed of 8 categories which are represented as folders with 8694 images in total all together.

In this project I will create a CNN and I will be trying different configurations until I reach an acceptable accuracy and in the end I will try to feed the model with some random images from the web and see if it can predict in the correct form.

Pre-processing

The composition of the dataset is given by the following table:

```
table =8×2

            Label             Count

    Cloths_&_Wipes            1223

    Potato_Chips_bag           996

    Raspberries               1043

    Seltzer_Water             1087

    fruit_plate               1206

    glass_food_container       975

    rice_sides                1046

    steak_knives              1118
```

There are 8 different folders with the number of images associated to the labels.

Eventhough the number of images for each category is satisfiable ,I have used Image Augmentation with ImageDataGenerator in order to improve the performance and the ability of the model to generalize(to reduce overfitting).

The original images consist in RGB coefficients in the 0-255, but such values would be too high for the models to process so I target values between 0 and 1 instead by scaling with a 1./255 factor.

Then I have used parameters as : zoom range, rotation range, width and height shift range and horizontal flip to complete the Augmentation process.

After an inspection of the dimention of the images I also decided to reshape all the dataset in 250x250x3 (first model) ,because the ratio of the values of the width and height had a medium almost of 1x1.

After the pre-preocessing of the images I have split the Dataset in two categories:
Training dataset with a percentage of 78% which correspond to a total of 6839 images belonging to 8 classes.
Test dataset with a percentage of 22% which correspond to a total of 1855 images belonging to 8 classes.

Now that I have prepared the datasets I am ready to start creating the Convolutional Neural Network model from scratch.

I am using two models which are the same but to compare I have changed parameters like :

Padding, Learning Rate, and image dimensions.

Convolutional Neural Network model from scratch

I have used CNN model and its architecture includes several building blocks, such as convolution layers, pooling layers, and fully connected layers.

Convolutional Layer

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function.

Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a **kernel**, is applied across the input.

In my model I have used 5 convolutional Layers, each one with parameters as below:

For the first model:
filters, kernel_ size, strides , padding='valid'

For the second model:

filters, kernel_ size, strides , padding='same'

In he first case that I have decided to use a **valid padding** which makes each successive feature map smaller after the convolutional operation.

In the second case I decided to use no padding so that I could have the same size in the output.

The distance between two successive kernel positions is called a **stride**, which also defines the convolution operation. I have used a stride (1,1) , since I have implemented Pooling layers for performing the downsampling.

For all my Convolutional layers I have used the same activation function: rectified linear unit "ReLU" which simply computes the function: $f(x) = \max(0, x)$

This function flattens the response to all negative values to zero, while leaving everything unchanged for values equal to or greater than zero.

I have also used **Batch normalization** after each Convolutonal layer in order to stabilize the learning process .

Pooling Layer

Pooling is a downsampling operation which reduces the in-plane dimensionality of the feature maps and decreases the number of subsequent learnable parameters.

Note: there is no learnable parameter in any of the pooling layers, whereas filter size, stride, and padding are hyperparameters in pooling operations, similar to convolution operations.

In my network I have used 2D Max Pooling which outputs the maximum value in each patch, and discards all the other values . It has a filter of size (2 x 2) with a stride of 2 in the first 3 layers and stride of 1 in the last 2.

In this operation the depth dimension of feature maps remains unchanged.

Fully Connected Layer

Convolutional and Pooling layers output high-level features of input and I passed these outputs to the classification part of the network composed by a flattening operation and three fully connected layers with **ReLU** activation function and a dropout layer (with a frequency of rate 0.2 at each step during training which helps to prevent overfitting) in each of the three layers and a last output layer with a **softmax** activation function.

The important thing is that the last dense layer must have exactly the number of classes of the dataset as parameter and that the activation function must be softmax because it is the most appropriate function.

Compilation

As an optimizer I have used ADAM since after some experiment by using RMSprop, it resulted the best choice since I had better results.

The learning rate is different for each one of the two models.

For the first model I have used a learning rate of 0.00001.

For the second model I have used a learning rate of 0.001.

Categorical crossentropy is used as a loss and the metric with which I have evaluated the model is “accuracy”.

Summary of the two models

First model : lr=0.00001, padding valid, image_dim=250x250x3

batch_normalization_7 (Batch	(None, 29, 29, 64)	256
conv2d_7 (Conv2D)	(None, 26, 26, 128)	131200
activation_11 (Activation)	(None, 26, 26, 128)	0
max_pooling2d_7 (MaxPooling2	(None, 13, 13, 128)	0
batch_normalization_8 (Batch	(None, 13, 13, 128)	512
conv2d_8 (Conv2D)	(None, 11, 11, 256)	295168
activation_12 (Activation)	(None, 11, 11, 256)	0
max_pooling2d_8 (MaxPooling2	(None, 10, 10, 256)	0
batch_normalization_9 (Batch	(None, 10, 10, 256)	1024
conv2d_9 (Conv2D)	(None, 8, 8, 128)	295040
activation_13 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_9 (MaxPooling2	(None, 7, 7, 128)	0
batch_normalization_10 (Batc	(None, 7, 7, 128)	512
flatten_1 (Flatten)	(None, 6272)	0
dense_4 (Dense)	(None, 512)	3211776
activation_14 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131328
activation_15 (Activation)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
activation_16 (Activation)	(None, 128)	0
dropout_5 (Dropout)	(None, 128)	0
batch_normalization_11 (Batc	(None, 128)	512
dense_7 (Dense)	(None, 8)	1032
activation_17 (Activation)	(None, 8)	0

Total params: 4,138,952
Trainable params: 4,137,480
Non-trainable params: 1,472

Second model : lr=0.001, no padding, image_dim=300x300x3

activation_36 (Activation)	(None, 150, 150, 32)	0
max_pooling2d_20 (MaxPooling	(None, 75, 75, 32)	0
batch_normalization_24 (Batc	(None, 75, 75, 32)	128
conv2d_21 (Conv2D)	(None, 75, 75, 64)	32832
activation_37 (Activation)	(None, 75, 75, 64)	0
max_pooling2d_21 (MaxPooling	(None, 37, 37, 64)	0
batch_normalization_25 (Batc	(None, 37, 37, 64)	256
conv2d_22 (Conv2D)	(None, 37, 37, 128)	131200
activation_38 (Activation)	(None, 37, 37, 128)	0
max_pooling2d_22 (MaxPooling	(None, 18, 18, 128)	0
batch_normalization_26 (Batc	(None, 18, 18, 128)	512
conv2d_23 (Conv2D)	(None, 18, 18, 256)	295168
activation_39 (Activation)	(None, 18, 18, 256)	0
max_pooling2d_23 (MaxPooling	(None, 17, 17, 256)	0
batch_normalization_27 (Batc	(None, 17, 17, 256)	1024
conv2d_24 (Conv2D)	(None, 15, 15, 128)	295040
activation_40 (Activation)	(None, 15, 15, 128)	0
max_pooling2d_24 (MaxPooling	(None, 14, 14, 128)	0
batch_normalization_28 (Batc	(None, 14, 14, 128)	512
flatten_4 (Flatten)	(None, 25088)	0
dense_16 (Dense)	(None, 512)	12845568
activation_41 (Activation)	(None, 512)	0
dropout_12 (Dropout)	(None, 512)	0
dense_17 (Dense)	(None, 256)	131328
activation_42 (Activation)	(None, 256)	0
dropout_13 (Dropout)	(None, 256)	0
dense_18 (Dense)	(None, 128)	32896
activation_43 (Activation)	(None, 128)	0
dropout_14 (Dropout)	(None, 128)	0
batch_normalization_29 (Batc	(None, 128)	512
dense_19 (Dense)	(None, 8)	1032
activation_44 (Activation)	(None, 8)	0

Total params: 13,772,744
Trainable params: 13,771,272
Non-trainable params: 1,472

Data fitting

Then I fit the training data and I have trained for :

Epochs=100 in the first model since I was using 0.00001 learning rate and the model accuracy was slowly improving.

Epochs=50 in the second model since I was using a higher learning rate(lr=0.001) and the model was showing accuracy improvement slightly faster than the first one, but in order to prevent Overfitting I decided to train for 50 epochs.

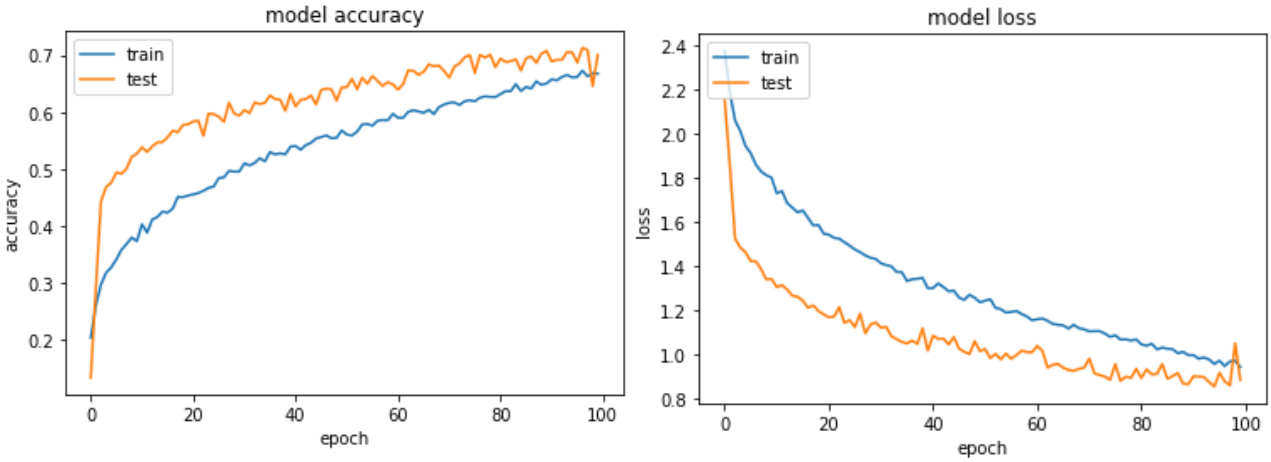
Evaluation

After fitting the data, I feed the testing data to the models.

Below I have explained the results for each of the two models.

First model:

Considering that we have a noisy dataset I get these results which seem good.



As seen in the plots the test accuracy
and the test loss is better than the training one. This comes as a results of the noisy data that are more in the training set and I can conclude that the noisy images are in a less percentage in the test set.

Something else that can be observed is that after 100 epochs the model shows the first signs of everfitting eventhough it is minimal that is a reason why I stopped the training at that value.

I tried some other tunning but these were the best results I got from this first model.

Test loss: 0.887357
Test accuracy: 0.700270

	precision	recall	f1-score	support
Cloths_Wipes	0.636	0.447	0.525	266
Potato_Chips_bag	0.569	0.749	0.647	203
Raspberries	0.917	0.794	0.851	238
Seltzer_Water	0.630	0.705	0.665	210
fruit_plate	0.524	0.667	0.587	273
glass_food_container	0.738	0.749	0.743	203
rice_sides	0.822	0.794	0.808	238
steak_knives	0.949	0.750	0.838	224
accuracy			0.700	1855
macro avg	0.723	0.707	0.708	1855
weighted avg	0.721	0.700	0.703	1855

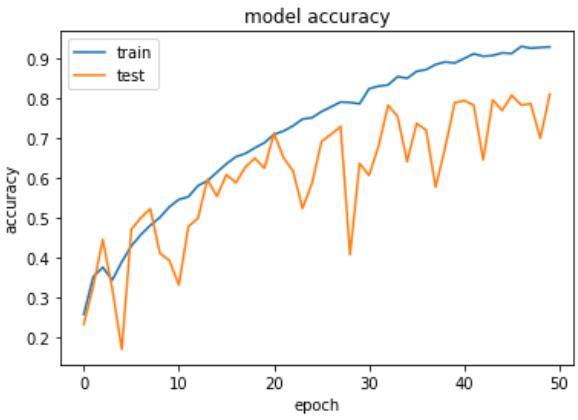
Second model:

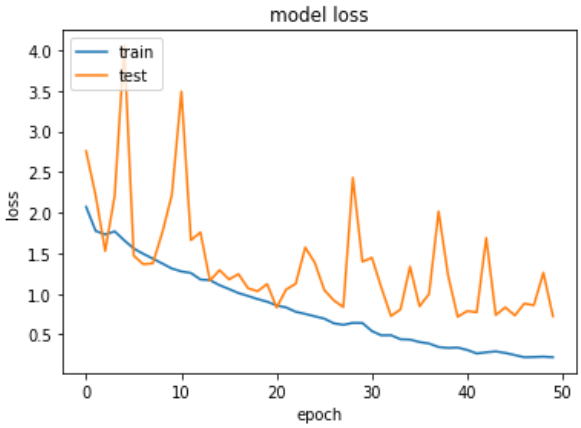
In the second model I decided ,as mentioned before , to change the learning rate to 0.001, to change the image dimensions to 300x300 , to not apply padding at the convolutional layers so that I can have the same dimensions on the output.

I tried to change the dimensions of FC layers to higher values:

2048 for the first one and 1024 for the second one but then after 20 epochs the model seemed to show signs of considerable overfitting so I changed the values back to 512 and 256 for the fist and the second layer.

Increasing the learning rate I expected better results on training set and a higher risk for overfitting.





By the plots we can easily see that the model fits to well to the training set and it has difficulties with the test set. The training set accuracy is higher than the test set accuracy.

So as expected the high value of the learning rate ,and the increased number of trainable parameter resulted in a small overfit for the second model, eventhough I must say that it is still an acceptable results considering that the both train and test plots tend to follow each-other, and considering that the dataset had a lot of noise.

Test loss: 0.724729
Test accuracy: 0.809704

	precision	recall	f1-score	support
Cloths_Wipes	0.782	0.620	0.692	266
Potato_Chips_bag	0.841	0.783	0.811	203
Raspberries	0.844	0.912	0.877	238
Seltzer_Water	0.804	0.781	0.792	210
fruit_plate	0.673	0.769	0.718	273
glass_food_container	0.869	0.852	0.861	203
rice_sides	0.809	0.908	0.855	238
steak_knives	0.917	0.884	0.900	224
accuracy			0.810	1855
macro avg	0.817	0.814	0.813	1855
weighted avg	0.812	0.810	0.808	1855

Training Time

First model:

As expected the first model had a higher training time because of the small learning rate.

Training time = 200 minutes = 3.3 hours

Second model:

In this model the learning rate was 100 times bigger and of course the training time was a lot shorter.

Training time = 120 minutes = 2 hours

Conclusions

After this experiment It is obvious that the accuracy of the model depends heavily from the dataset, from the way we preprocess the images and of course from how we built the neural network.

I can say that during building of the model we must keep in mind the overfitting and always try to avoid it or at least minimize it as much as possible by using likes of different pre-process or adding Batch normalization and Dropout layers in our model.

As for my dataset I have to say that I achieved satisfactory results considering the beforementioned problems with it.