SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning Homework 2:

CNN image classification

Simone Giordano

1772347

Index

Introduction

The scope of the homework is to identify various images in a Dataset composed by 8 classes.

Each class has images of a given type, in the personal case, with Matricola equal to '1772347', those classes are:

- Carpet_cleaners
- Espresso_cup
- Juice_carton
- Olives_jar
- Paper_bag
- Plastic_spoon
- Tangerines
- Vegetable_chips_&_Crisps

The Program is divided in two main phases:

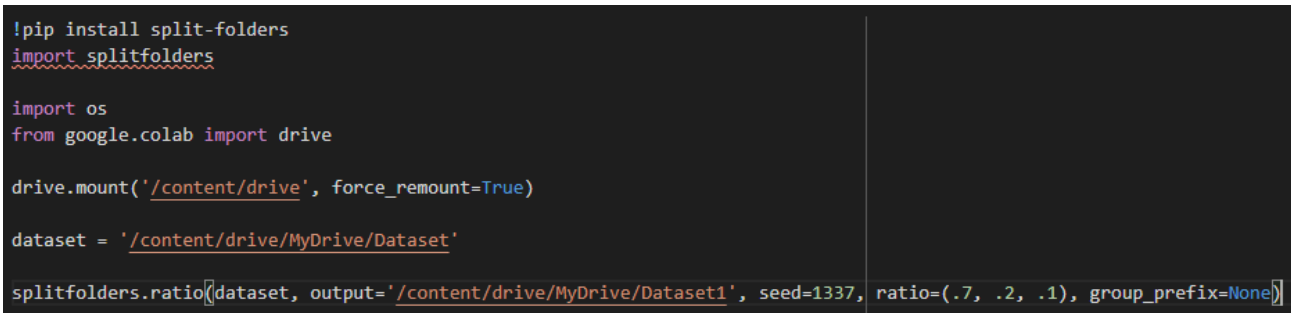
- First of all, there is the split of the given dataset in Train dataset, Validation dataset and Test dataset. This phase can be done just the first time when the program starts.
- Then there is the initialization and creation of the model.

The models that have been used are AlexNet and VGGNet.

1 Split given data

The initial dataset is just a directory with all the images within sub-directories. This led to an impossibility to find what images belongs to what generator. Those generators are Train, Validation, and Test.

To divide those three generators, and so create three different directories with the images, there is a Python library called 'splitfolders' that do this job in a straightforward way, where, the 'ratio' attribute, specifies that to the Train dataset belongs the 70% of the total dataset, in the validation dataset the 20% and in the Test the 10%.

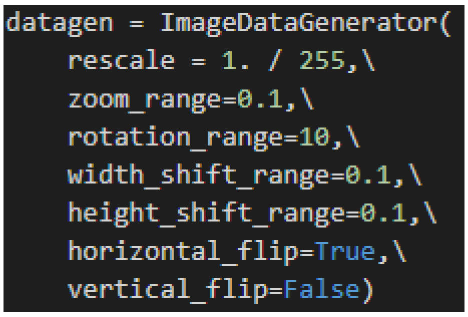


```
!pip install split-folders
import splitfolders

import os
from google.colab import drive

drive.mount('/content/drive', force_remount=True)

dataset = '/content/drive/MyDrive/Dataset'

splitfolders.ratio(dataset, output='/content/drive/MyDrive/Dataset1', seed=1337, ratio=(.7, .2, .1), group_prefix=None)
```

2 Load data

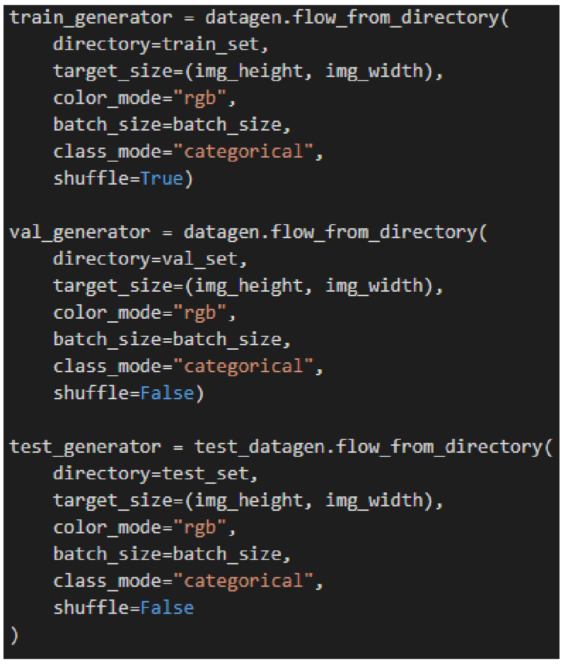
After the withdraw of the three dataset, previously created, there is the initialization of the generators.

First of all, we need to initialize the ImageDataGenerator that will be common to both the Train generator and the Validation generator.



```
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)
```

The Keras ImageDataGenerator class is not an "additive" operation. It is not taking the original data, randomly transforming it, and then returning both the original data and transformed data. Instead, the ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data. Regarding the Test datagen the image are just rescaled.



```
train_generator = datagen.flow_from_directory(
    directory=train_set,
    target_size=(img_height, img_width),
    color_mode="rgb",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=True)

val_generator = datagen.flow_from_directory(
    directory=val_set,
    target_size=(img_height, img_width),
    color_mode="rgb",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=False)

test_generator = test_datagen.flow_from_directory(
    directory=test_set,
    target_size=(img_height, img_width),
    color_mode="rgb",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=False
)
```

The three generators use the datagen previously created (that contains the way the images have to be modified) and must specify the directory where the images have to be picked up, the dimensions to which all images found will be resized, previously initialized to 228 x 228, the rgb color mode, the batch size initialized to 32, the class mode that in this case is "categorical", i.e., 2D one-hot encoded labels, and the shuffle true just in the train_generator.

In three variables are saved the values of the total number of samples, the number of classes and the image shape of the images of the train generator.

3 CNN initialization

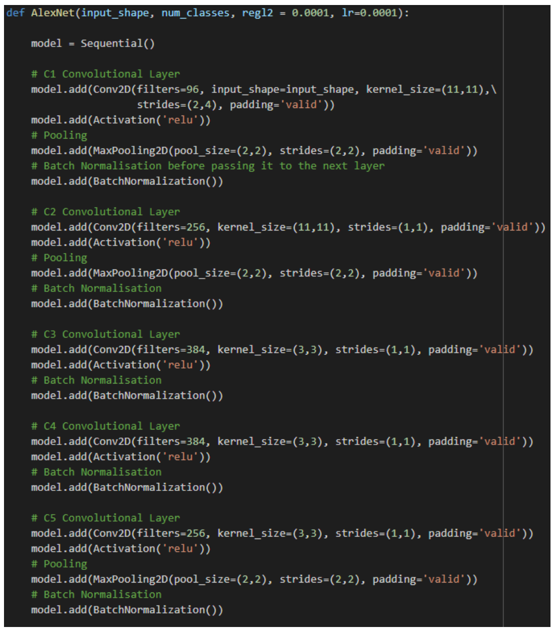
In the homework have been used two different Convolutional neural networks. Those are AlexNet and

Both of them follow the same idea but they apply two different ways to achieve the final result that is given an array of numbers, it will output numbers that describe the probability of the image being a certain class.

3.1 AlexNet

The architecture consists of 5 Convolutional Layers and 3 Fully Connected Layers.

3.1.1 AlexNet Convolutional layers



```python
def AlexNet(input_shape, num_classes, regl2 = 0.0001, lr=0.0001):

    model = Sequential()

    # C1 Convolutional Layer
    model.add(Conv2D(filters=96, input_shape=input_shape, kernel_size=(11,11),\
                     strides=(2,4), padding='valid'))
    model.add(Activation('relu'))
    # Pooling
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
    # Batch Normalisation before passing it to the next layer
    model.add(BatchNormalization())

    # C2 Convolutional Layer
    model.add(Conv2D(filters=256, kernel_size=(11,11), strides=(1,1), padding='valid'))
    model.add(Activation('relu'))
    # Pooling
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
    # Batch Normalisation
    model.add(BatchNormalization())

    # C3 Convolutional Layer
    model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'))
    model.add(Activation('relu'))
    # Batch Normalisation
    model.add(BatchNormalization())

    # C4 Convolutional Layer
    model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'))
    model.add(Activation('relu'))
    # Batch Normalisation
    model.add(BatchNormalization())

    # C5 Convolutional Layer
    model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid'))
    model.add(Activation('relu'))
    # Pooling
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
    # Batch Normalisation
    model.add(BatchNormalization())
```

In Input to the CNN, the values of the shape of images and the number of classes are passed.

Every convolutional layer has different parameters that are:

The Number of Filters that the convolutional layer will learn, increasing going closer the output predictions.

In C1 there is the input shape, given in input, instead of the next layers where takes in input the previous layer.

The kernel size specifies the width and the height of the 2D convolution window.

The strides specify the "step" of the convolution along the x and y axis of the input volume.

Padding valid means valid parameter, the input volume is not zero-padded and the spatial dimensions are allowed to reduce via the natural application of convolution.

The ‘relu(Rectified Linear Unit)’ activation have been applied to each layers so that all the negative values are not passed to the next layer.

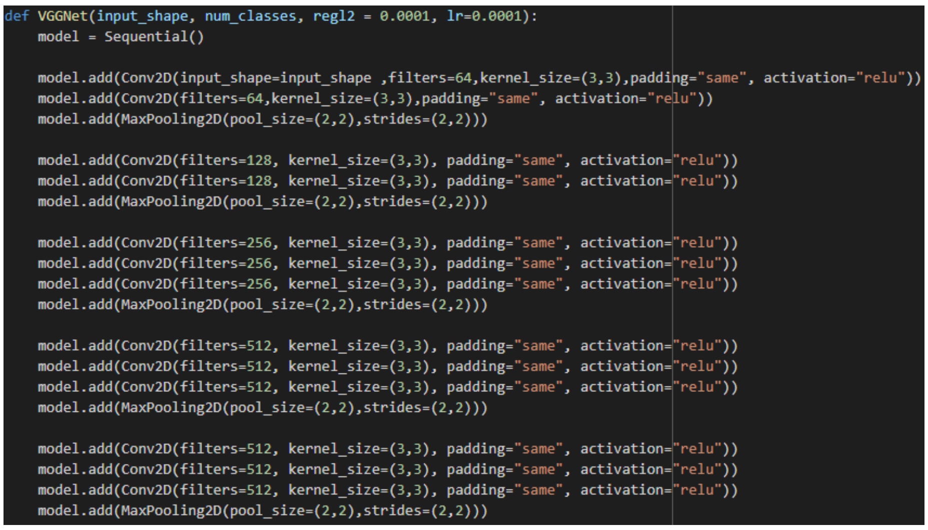
A max Pooling function have been applied to every layer. This is used to reduce the spatial dimensions of the output volume.

The output layer is activated by a ‘softmax’ function. Softmax converts a real vector to a vector of categorical probabilities. The elements of the output vector are in range (0, 1) and sum to 1. Each vector is handled independently. Softmax, like in this case, is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution.

3.2 VGGNet

It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

3.2.1 VGGNet Convolutional layers



```python
def VGGNet(input_shape, num_classes, regl2 = 0.0001, lr=0.0001):
    model = Sequential()

    model.add(Conv2D(input_shape=input_shape ,filters=64,kernel_size=(3,3),padding="same", activation="relu"))
    model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

    model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

    model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
```

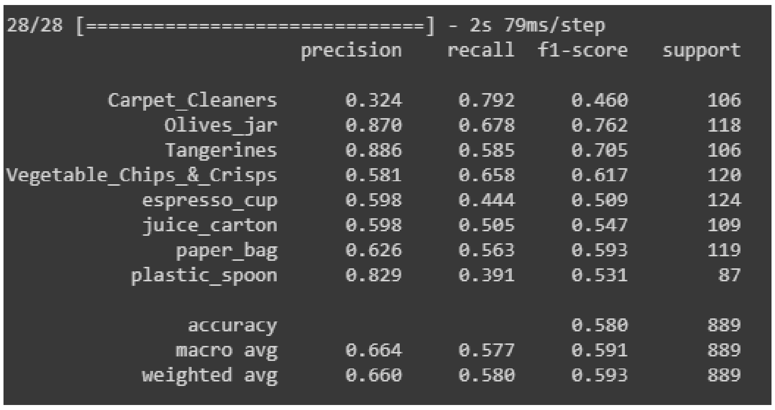
Differently from AlexNet, instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture.

4 Model evaluation

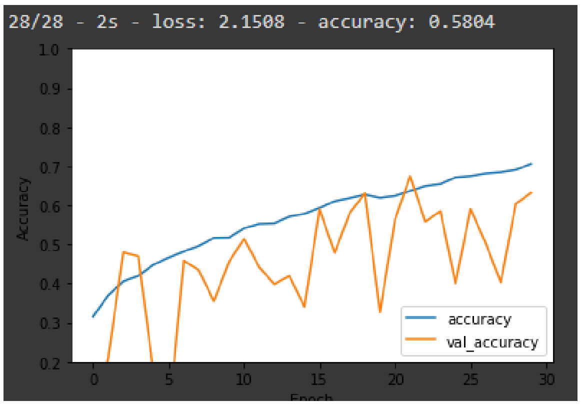
VGGNet is the improved version of AlexNet, in fact It came later AlexNet. Although VGGNet is based on AlexNet, there are several differences between the two:

- Instead of using large receptive fields like AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Because there are now three ReLU units instead of just one, the decision function is more discriminative.
- The small-size convolution filters allow VGG to have a large number of weight layers; of course, more layers lead to improved performance.

4.1 The results



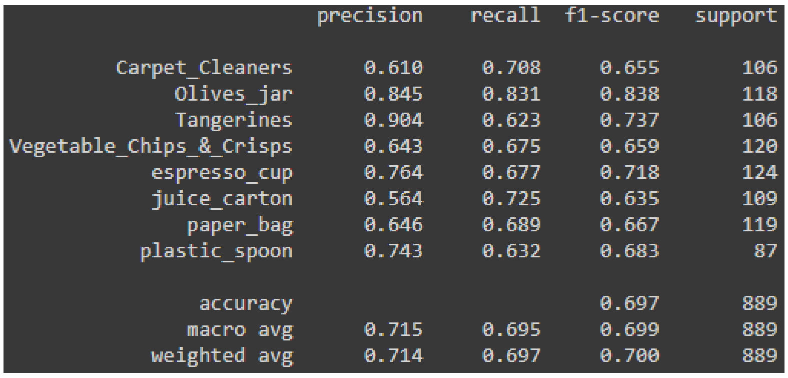
```
28/28 [==============================] - 2s 79ms/step
                         precision    recall  f1-score   support

        Carpet_Cleaners      0.324     0.792     0.460       106
             Olives_jar      0.870     0.678     0.762       118
             Tangerines      0.886     0.585     0.705       106
Vegetable_Chips_&_Crisps     0.581     0.658     0.617       120
           espresso_cup      0.598     0.444     0.509       124
           juice_carton      0.598     0.505     0.547       109
              paper_bag      0.626     0.563     0.593       119
          plastic_spoon      0.829     0.391     0.531        87

               accuracy                          0.580       889
              macro avg      0.664     0.577     0.591       889
           weighted avg      0.660     0.580     0.593       889
```



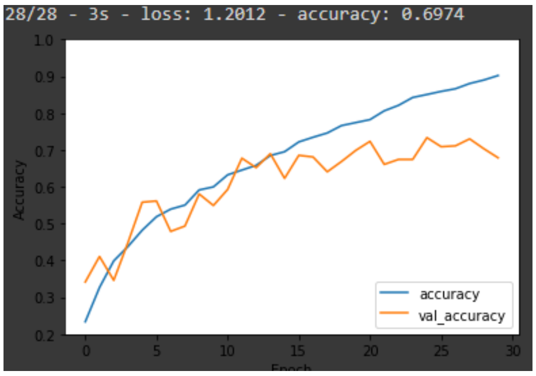
```
28/28 - 2s - loss: 2.1508 - accuracy: 0.5804
```

As can be seen in the images above, the results for the alexNet model are not good at all, having in certain epoch a very poor accuracy on the Test dataset. The overall accuracy on the Test set is 0.580 .

The reasons for this score could be multiple, for instance, I noticed a lot of image noise, i.e., images that are not solid in respect of the class, in some classes, such as the 'carpet cleaners' class. In fact, those classes have a really poor precision that have repercussions on the overall precision.



```
                     precision    recall  f1-score   support

      Carpet_Cleaners      0.610     0.708     0.655       106
           Olives_jar      0.845     0.831     0.838       118
           Tangerines      0.904     0.623     0.737       106
Vegetable_Chips_&_Crisps   0.643     0.675     0.659       120
          espresso_cup      0.764     0.677     0.718       124
          juice_carton      0.564     0.725     0.635       109
             paper_bag      0.646     0.689     0.667       119
         plastic_spoon      0.743     0.632     0.683        87

             accuracy                           0.697       889
            macro avg      0.715     0.695     0.699       889
         weighted avg      0.714     0.697     0.700       889
```



```
28/28 - 3s - loss: 1.2012 - accuracy: 0.6974
```

Since VGGNet is an improving version of AlexNet, there are no surprises that this CNN is performing better. One main difference in respect of AlexNet, besides the fact of a better Accuracyon the Test set, is that VGGNet performed in a more linear way, across all the epoch, on the val_accuracy.

4.1.1 Ways to improve results in VGGNet

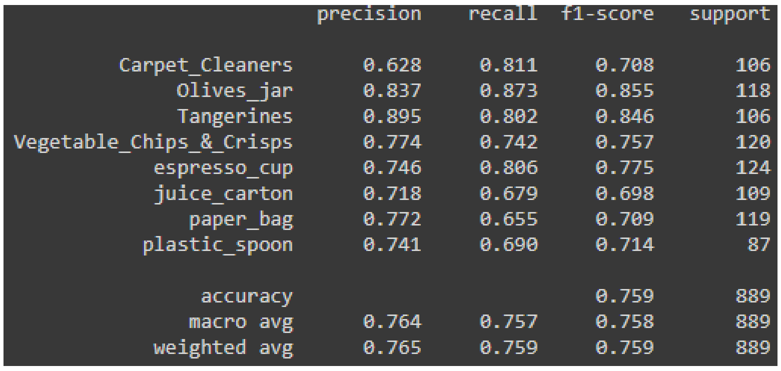
The first way to improve VGGNet is to read the documentation of the CNN and notice that the better results are achieved resizing the images of the dataset with width = 224 and length = 224.

The second way is to tune the learning rate of the Adam Optimizer. The learning rate is an hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. The Adam optimizer adapt the learning rates during train steps. In the previous run the learning rate was set to 0.0001.

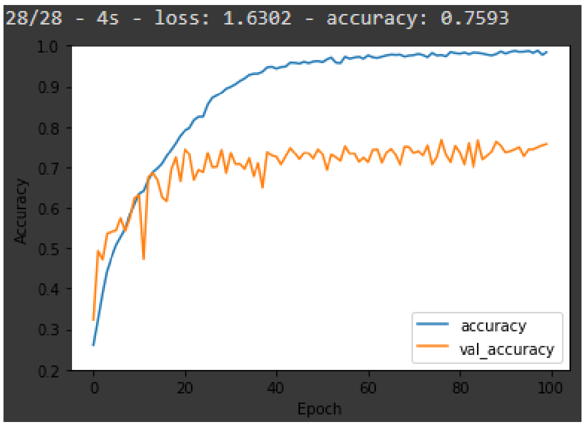
The third way could be to increase the number of epochs, trying to find the number where the accuracy converges.

Applying the first and the third method (not the second one because there are not significant improving), augmenting the number of epochs to 100 from the 30 of the previous running, there are significant improving, especially on the accuracy on the training set that converges more or less at the 70th epoch.

Less improving on the accuracy on the test and the validation set.



```
                          precision    recall  f1-score   support

          Carpet_Cleaners     0.628     0.811     0.708       106
               Olives_jar     0.837     0.873     0.855       118
              Tangerines     0.895     0.802     0.846       106
Vegetable_Chips_&_Crisps     0.774     0.742     0.757       120
            espresso_cup     0.746     0.806     0.775       124
             juice_carton     0.718     0.679     0.698       109
                paper_bag     0.772     0.655     0.709       119
            plastic_spoon     0.741     0.690     0.714        87

                 accuracy                         0.759       889
                macro avg     0.764     0.757     0.758       889
             weighted avg     0.765     0.759     0.759       889
```



```
28/28 - 4s - loss: 1.6302 - accuracy: 0.7593
```

5 Sources

- https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
- https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
- https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
- https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c
- MLEx11.AlexNet_ARGOS