2020

Report Homework2

MACHINE LEARNING

PAPA EDOARDO MATRICOLA: 1962169

Sommario

1. Project Overview

The goal of this project is provide a solution for the classification of images regarding objects in a home environment.

In particular, my enviroments are:

- Containers/basket_container;
- Tableware/sushi_plate;
- Drinks/Tea_drink_bottle;
- Cutlery/dinnerware;
- Fruits/Oranges;
- Snacks/Pretzels;
- Food/pasta_sides;
- Cleaning_stuff/Dust_Cloths.

1.1 Dataset

My dataset is composed by 8 different classes with a lot of images for each one:

- Containers/basket_container have 1187 images;
- Tableware/sushi_plate have 1114 images;
- Drinks/Tea_drink_bottle have 991 images;
- Cutlery/dinnerware have 956 images;
- Fruits/Oranges have 1002 images;
- Snacks/Pretzels have 1032 images;
- Food/pasta_sides have 996 images;
- Cleaning_stuff/Dust_Cloths have 1230 images.

1.2 Summarize

In this project i evaluated two different type of model. The first model that i used is a VGG16. VGG16 is a Convolutional Neural Network that it was proposed by Karen Simonyan and Andrew Zisserman of Oxford Robotics Institute in the the year 2014. The second model that i used is a own model that i built on my own.
For both model i split the dataset in two parts. The first part has 77% of the total dataset and this part it will use for train the model. The second part has 33% of the total dataset and this part it will use for test the model.
Both model receive the images input with same size 94x94. Indeed the images was processed before making all smaller and with same size.

2. Initial Process

The first step that i did was import all the needed library and after that, i loaded all the dataset. Now, we had a one single big dataset, but we want have two datasets, one for train and one for test. So i thirth thing that i did was split the dataset into 2 differents one. I decided to use a keras library (*ImageDataGenerator*) to take the images from the different datasets and process them. I also decided to use this library because it has a particular ability to use data argumentation specified by arguments to the class constructor.

I used *ImageDataGenerator* not in a classical way. I'll explain, usually we use *ImageDataGenerator* with method called "*flow_from_directory*" where one important argument is a path of the dataset. In my case i havent't a one directory for the train data and one directory for test data. So, i must did in a different way. Infact, in my case i split the dataset into 2 different dataset and i use another method instead of "*flow_from_directory*" called "*flow*". In this way, i can pass to this function directly the "array" of specific dataset.

After that i build the two different model with two difference characteristic; I plot the model and i train it. The last thing that I do is evaluate the model doing the prediction and plotting the classification_report, confusion_matrix and graphics.

3. Own CNN "*EdoardoPapaNet*" For Image Classification

For making own CNN image classifir the first thing that we just do it's build a model.

3.1 Make Model

The first thing that we need to know is that CNN is composed by a hierarchical levels. In this network we can have a lot of layers. The first layer is the input level and this is directly connected with the pixels of the image. In this project the pixels of image is called "*input_shape*". The layers that we find between first and last layer is called "*hidden layers*". The last level are generally fully connected.
This CNN is created so that all the layers will be convolutional.
All the images that are inside the dataset are *"rgb"* images, so it means that if we want create a own CNN that consider also the color, the important thing that we must consider is create a *input_shape* with size of image that we consider, plus the factor 3 becacuse the color is "*rgb*".
Example: if we have a image with size 300x150 (rgb) the input shape will be (300,150,3).
Instead, if we don't consider the color, we just transform the rgb-image to black&white-image and after that create a *input_shape* with size of image that we consider, plus the factor 2 because the color is black&white.

Inside the convolutional layers i decide to use a ReLU activation function. This function allow to flattens all the negative values to zero, while leaving everything unchanged values equals to or greather then zero.

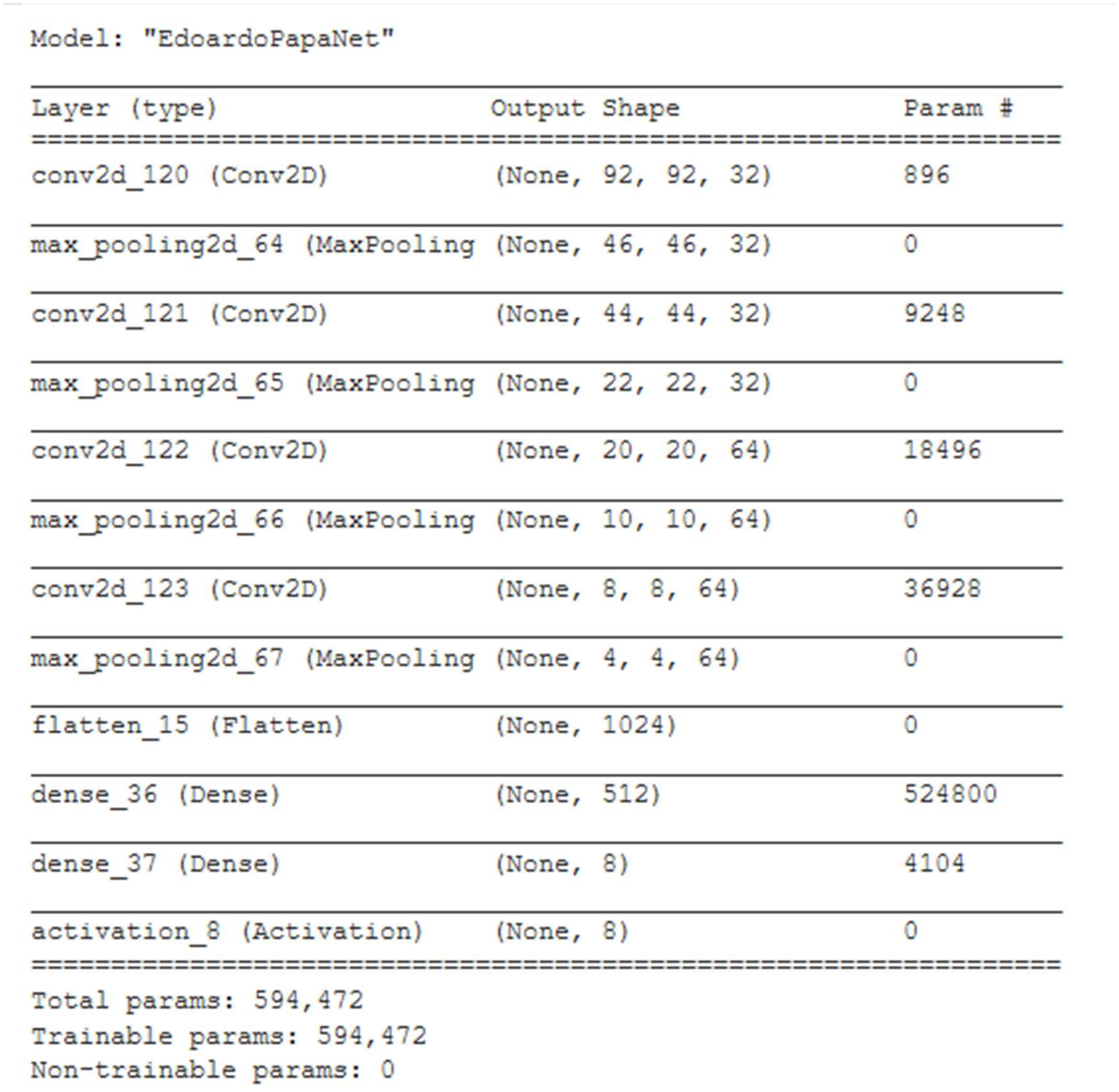
I choose to insert, after convolutional layers, a pooling methods. It's allow to decrease the size of the matrix. This is needed because otherwise i would go to have a lot of amount of data that i wouldn't know how to manage. I used MaxPooling for reduce the size of the matrix.
For example i used the MaxPooling after each convolutional layer. I decided that the size of the MaxPooling is 2x2. In this way we "*create*" a square of 2x2 grid that slide in the matrix and extrapolate the best (keep the bit more interesting) value inside the square.
I also choose to insert, into this CNN, the dropout layer. The dropout layer prevents the overfitting.

The last dense layer inside this CNN must have exactly the number of classes that are inside the dataset and the activation function must be “*softmax*” because in this case we have a multiclass problem.

I decide to use as optimizer “*rmsprop*” with learning rate 0.0001. After a lot of tests it was the one optimal value that provided the best performance.

The model ends when the compile function is executed.

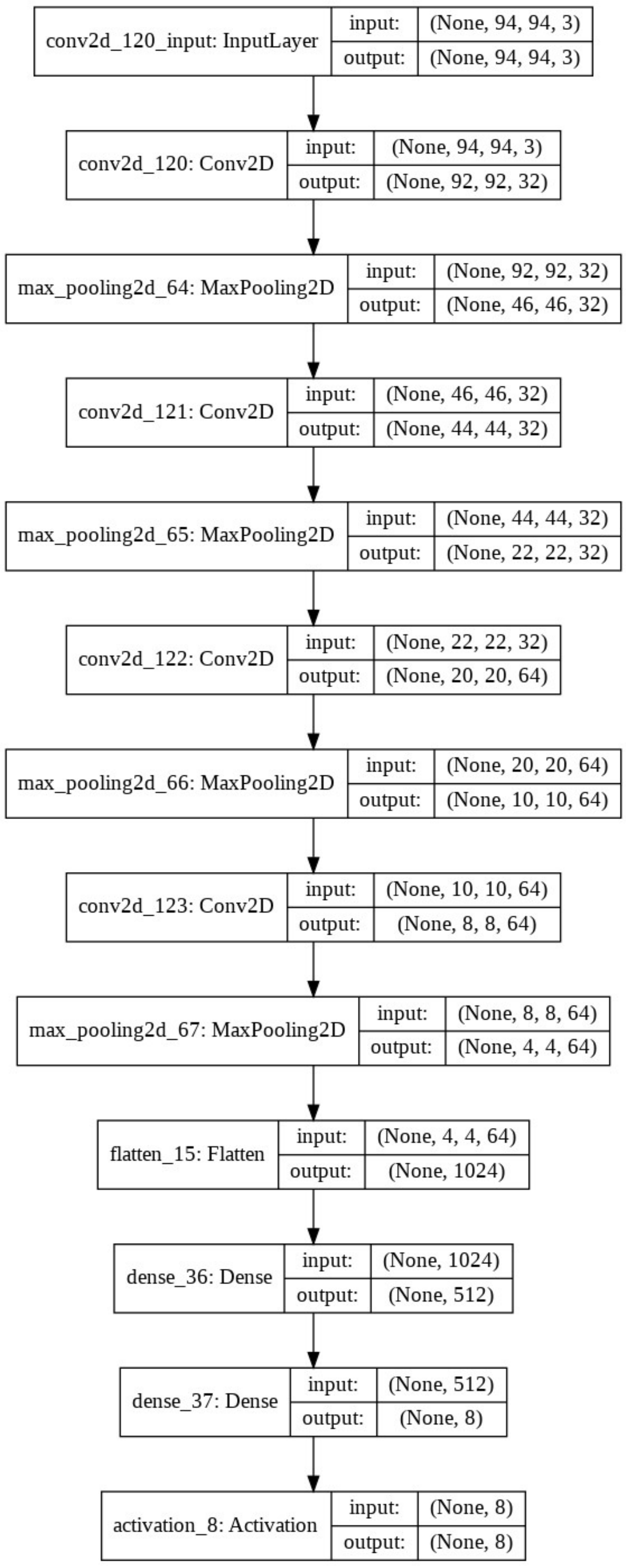
3.2 EdoardoPapaNet’s model



```
Model: "EdoardoPapaNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_120 (Conv2D)          (None, 92, 92, 32)        896
_________________________________________________________________
max_pooling2d_64 (MaxPooling (None, 46, 46, 32)        0
_________________________________________________________________
conv2d_121 (Conv2D)          (None, 44, 44, 32)        9248
_________________________________________________________________
max_pooling2d_65 (MaxPooling (None, 22, 22, 32)        0
_________________________________________________________________
conv2d_122 (Conv2D)          (None, 20, 20, 64)        18496
_________________________________________________________________
max_pooling2d_66 (MaxPooling (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_123 (Conv2D)          (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_67 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_15 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_36 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_37 (Dense)             (None, 8)                 4104
_________________________________________________________________
activation_8 (Activation)    (None, 8)                 0
=================================================================
Total params: 594,472
Trainable params: 594,472
Non-trainable params: 0
```

Figura 1 – EdoardoPapaNet

3.3 Plot EdoardoPapaNet's model



conv2d_120_input: InputLayer | input: (None, 94, 94, 3) | output: (None, 94, 94, 3)
conv2d_120: Conv2D | input: (None, 94, 94, 3) | output: (None, 92, 92, 32)
max_pooling2d_64: MaxPooling2D | input: (None, 92, 92, 32) | output: (None, 46, 46, 32)
conv2d_121: Conv2D | input: (None, 46, 46, 32) | output: (None, 44, 44, 32)
max_pooling2d_65: MaxPooling2D | input: (None, 44, 44, 32) | output: (None, 22, 22, 32)
conv2d_122: Conv2D | input: (None, 22, 22, 32) | output: (None, 20, 20, 64)
max_pooling2d_66: MaxPooling2D | input: (None, 20, 20, 64) | output: (None, 10, 10, 64)
conv2d_123: Conv2D | input: (None, 10, 10, 64) | output: (None, 8, 8, 64)
max_pooling2d_67: MaxPooling2D | input: (None, 8, 8, 64) | output: (None, 4, 4, 64)
flatten_15: Flatten | input: (None, 4, 4, 64) | output: (None, 1024)
dense_36: Dense | input: (None, 1024) | output: (None, 512)
dense_37: Dense | input: (None, 512) | output: (None, 8)
activation_8: Activation | input: (None, 8) | output: (None, 8)

Figura 2 - Plot EdoardoPapaNet

3.4 Fit EdoardoPapaNet's model

For execute the fit of this model i used a *fit_generator* function that trains the model on data generated batch-by-batch.

This function takes different parameter as a input values:

- train_set
- epochs
- step_per_epoch
- validation_data
- validation_steps

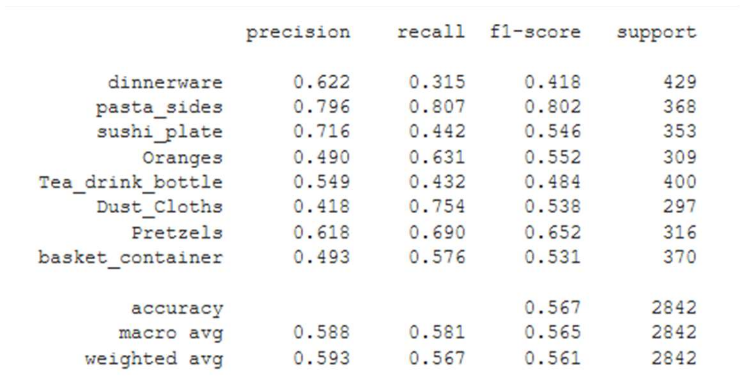
For the first parameter, i passed the *train_generator* generated precedently.
For the *epochs* i passed the number of the iterations.
For *step_per_epoch* i passed the total number of steps. This value is typically be equal to *train_generator.n // train_generator.batch_size* (where n is number of samples).
For *validation_data* corresponds to validation_generator that we generated previously.
For *validation_steps* corresponds to total number of steps and typically is *validation_generator.n // validation_generator.batch_size+1*

The last two parameters have been added to see precisely the behavior of the model being applied on the train comparing it with the test. If difference between train and test accuracy is high then we find a overfitting problem.

3.5 Evaluation EdoardoPapaNet's model

For do the evaluation of the model i decided to use "*classification_report*". I decided to use this rappresentation because it allow to see the precision, recall, f1_score and accuracy. Below there're a rapresentation of the tests that i did for find the best configuration for the model.

3.5.1 With learning-rate = 0.0001 without Dropout



```
                 precision    recall  f1-score   support

      dinnerware      0.622     0.315     0.418       429
     pasta_sides      0.796     0.807     0.802       368
     sushi_plate      0.716     0.442     0.546       353
         Oranges      0.490     0.631     0.552       309
Tea_drink_bottle      0.549     0.432     0.484       400
     Dust_Cloths      0.418     0.754     0.538       297
        Pretzels      0.618     0.690     0.652       316
basket_container      0.493     0.576     0.531       370

        accuracy                          0.567      2842
       macro avg      0.588     0.581     0.565      2842
    weighted avg      0.593     0.567     0.561      2842
```

Figura 3 - EdoardoPapaNet Classification Report rl = 0.0001

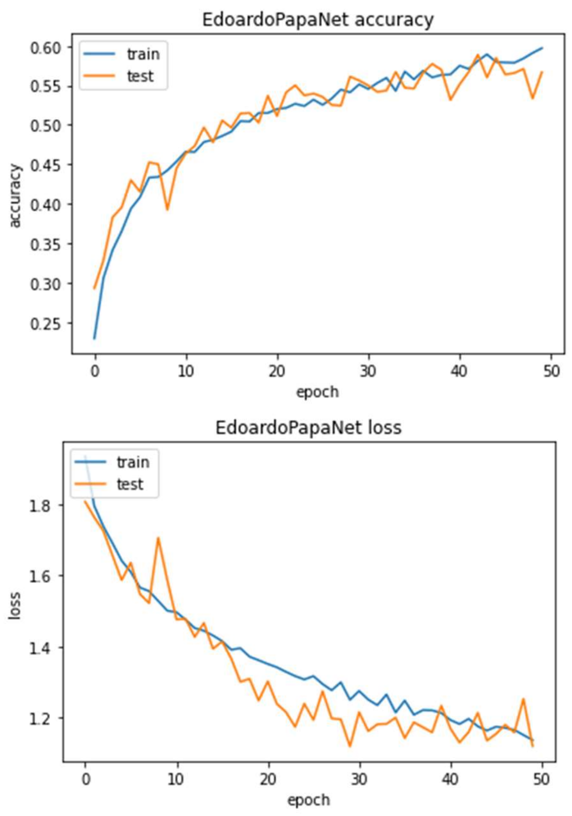
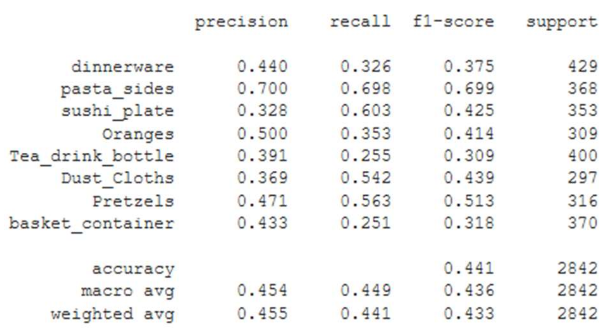


Figura 4 - EdoardoPapaNet Graphics rl = 0.0001

3.5.2 With learning-rate = 0.00001 without Dropout



```
                  precision    recall  f1-score   support

      dinnerware      0.440     0.326     0.375       429
     pasta_sides      0.700     0.698     0.699       368
     sushi_plate      0.328     0.603     0.425       353
         Oranges      0.500     0.353     0.414       309
Tea_drink_bottle      0.391     0.255     0.309       400
     Dust_Cloths      0.369     0.542     0.439       297
        Pretzels      0.471     0.563     0.513       316
basket_container      0.433     0.251     0.318       370

        accuracy                          0.441      2842
       macro avg      0.454     0.449     0.436      2842
    weighted avg      0.455     0.441     0.433      2842
```

Figura 5 - EdoardoPapaNet Classification Report rl = 0.00001

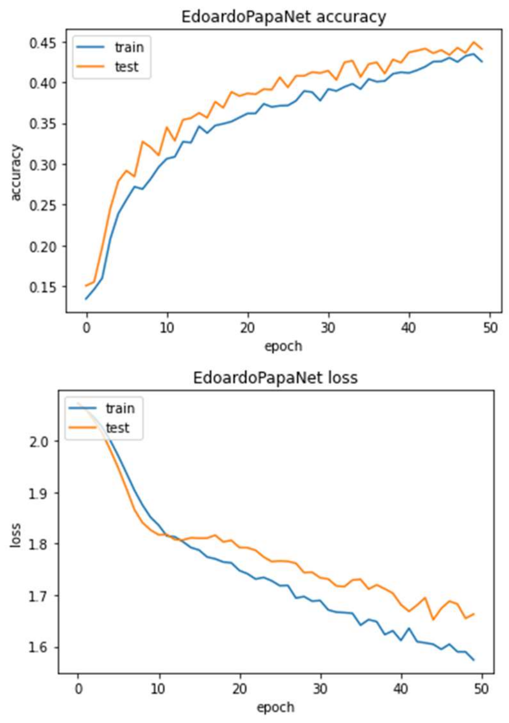
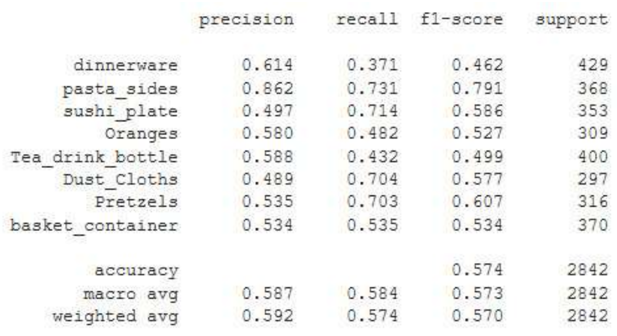


Figura 6 - EdoardoPapaNet Graphics rl = 0.00001

We can observe that we have the same model but with different learning-rate. In the first case (learning-rate:0.0001) can we see that the accuracy is 0.567. The model doesen't produce a good performance but the accuracy is better then the second (learning-rate:0.00001). In the first case (learning-rate:0.0001) we have a slight overfitting and it is less in the second case but the second case has a less accuracy then the first.

3.5.3 With learning-rate = 0.0001 with dropout = 0.18



```
                  precision    recall  f1-score   support

       dinnerware      0.614     0.371     0.462       429
      pasta_sides      0.862     0.731     0.791       368
      sushi_plate      0.497     0.714     0.586       353
          Oranges      0.580     0.482     0.527       309
 Tea_drink_bottle      0.588     0.432     0.499       400
      Dust_Cloths      0.489     0.704     0.577       297
         Pretzels      0.535     0.703     0.607       316
 basket_container      0.534     0.535     0.534       370

         accuracy                          0.574      2842
        macro avg      0.587     0.584     0.573      2842
     weighted avg      0.592     0.574     0.570      2842
```

Figura 7 - EdoardoPapaNet Classification Report rl = 0.0001 dropout = 0.18

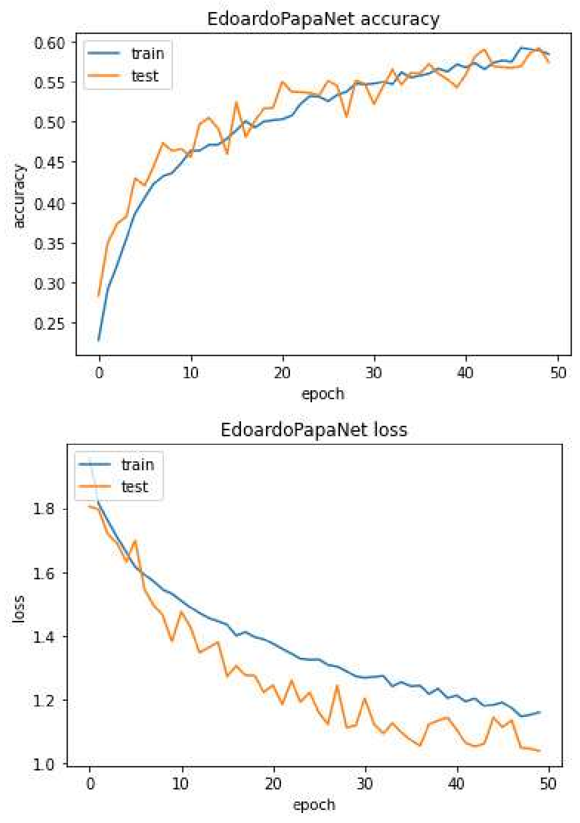
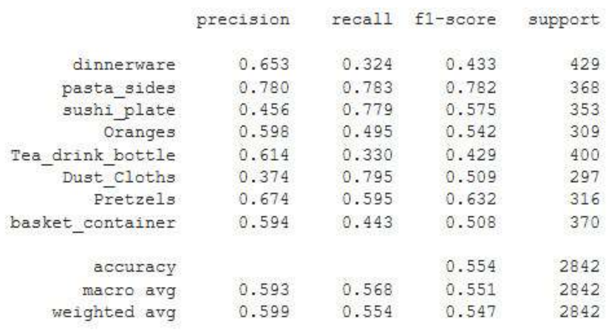


Figura 8 - EdoardoPapaNet Graphics rl = 0.0001 dropout = 0.18

3.5.4 With learning-rate = 0.0001 with dropout = 0.22



```
                  precision    recall  f1-score   support

       dinnerware      0.653     0.324     0.433       429
      pasta_sides      0.780     0.783     0.782       368
      sushi_plate      0.456     0.779     0.575       353
          Oranges      0.598     0.495     0.542       309
Tea_drink_bottle      0.614     0.330     0.429       400
      Dust_Cloths      0.374     0.795     0.509       297
         Pretzels      0.674     0.595     0.632       316
 basket_container      0.594     0.443     0.508       370

         accuracy                          0.554      2842
        macro avg      0.593     0.568     0.551      2842
     weighted avg      0.599     0.554     0.547      2842
```

Figura 9 - EdoardoPapaNet Classification Report rl = 0.0001 dropout = 0.22

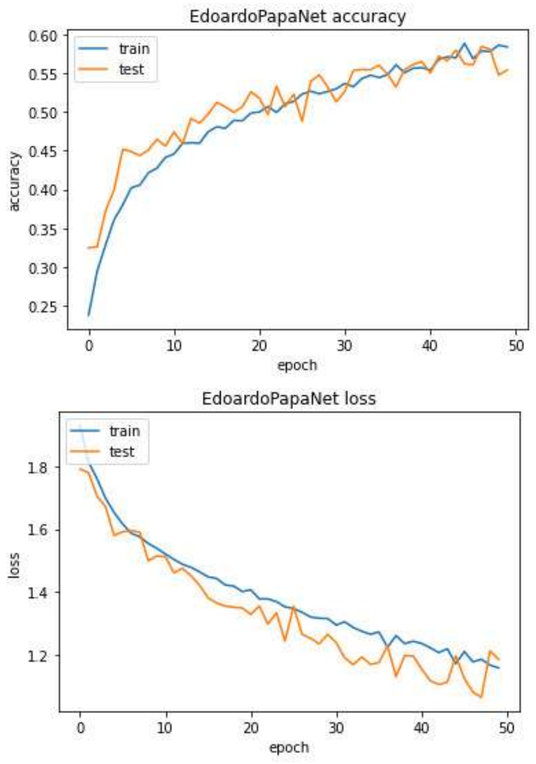
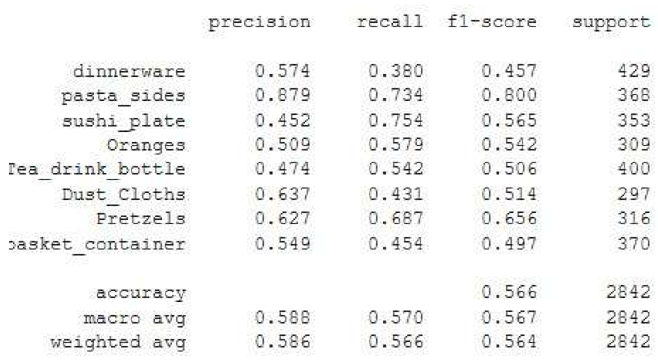


Figura 10 - EdoardoPapaNet Graphics rl = 0.0001 dropout = 0.22

3.5.5 With learning-rate = 0.0001 with dropout = 0.25



```
                  precision    recall  f1-score   support

      dinnerware      0.574     0.380     0.457       429
     pasta_sides      0.879     0.734     0.800       368
     sushi_plate      0.452     0.754     0.565       353
         Oranges      0.509     0.579     0.542       309
Tea_drink_bottle      0.474     0.542     0.506       400
     Dust_Cloths      0.637     0.431     0.514       297
        Pretzels      0.627     0.687     0.656       316
basket_container      0.549     0.454     0.497       370

        accuracy                          0.566      2842
       macro avg      0.588     0.570     0.567      2842
    weighted avg      0.586     0.566     0.564      2842
```

Figura 11 - EdoardoPapaNet Classification Report rl = 0.0001 dropout = 0.25

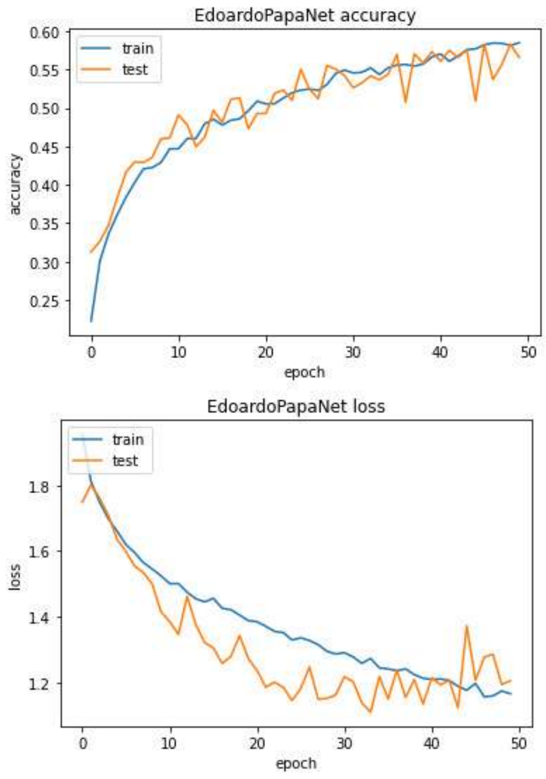


Figura 12 - EdoardoPapaNet Graphics rl = 0.0001 dropout = 0.25

Now i execute the algorithm with fixed learning-rete (0.0001) and i add a *dropout* layer with different values. I started with trying 0.18, leater on with 0,25 and finally with 0,22. I started with a low value and i immediatly saw that i it didn't better then the previous, infact it was very similar. I decided to up the value to 0.25. Unfortunately the model started to get worse. So, i decided to use a middle values,0.22. Now the model is better then the others that i've tried. With *dropout*, with value 0.22, we have a slight overfitting.

In summary, I decided to try to improve my model by first increasing and decreasing the learning-rate, then by how much to update the network weights in order to obtain the best performance. The second thing that I decided to do is to insert a dropout layer. In this way, we randomly select and remove neurons in a neural network during the training phase. This random removal of neurons prevents excessive co-adaptation of neurons and thus reduces the likelihood of network overfiting. Several examples were run with different dropout layer "rate" values to see if the model could improve / deteriorate.

3.6 General Problems

The model isn't very accouracy. This "problem" not depend only on our model but also on the dataset. Infact, the dataset contains noisy images, images that are difficult to classify or image that aren't into correct class. This increases the difficulty to finding a neural network that can fit this dataset.

This model is not totaly bad, but it never give a rather high level of security.

4. CNN "*VGG16*" For Image Classification

As i previously said, this model was proposed by K. Simonyan and A. Zisserman from the University of Oxford.

This is the architecture:

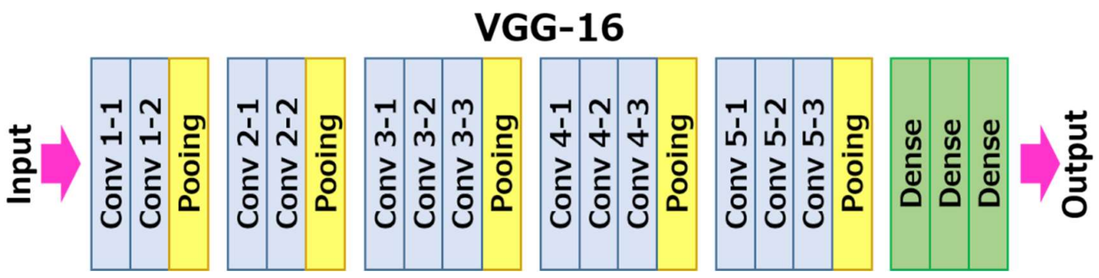


Figura 13 - VGG16 model

The input of the first convolutional layer is a size of the own image. The image is passed through a stack of convolutional layers, where the filtyers were used with very small receptive field 3x3. In one of the configurations, it also utilizer 1x1 convolutiona filter. The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third contains 8 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU) non-linearity.

4.1 Fit Model

For fitting this model, the steps are equals of the previous classification model. So, i used a *fit_generator* function that trains the model on data generated batch-by-batch.

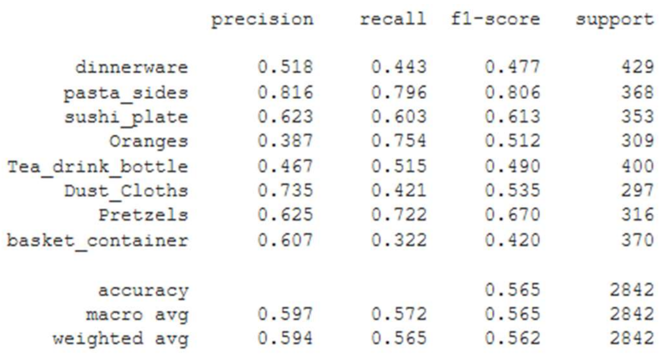
This function takes different parameter as a input values:

- train_set
- epochs
- step_per_epoch
- validation_data
- validation_steps

For the expleation of all the previous point, you can find it on chapter 3.4.

4.2 Evaluation VGG16 Model

For do the evaluation of the model i decided to use "*classification_report*". I decided to use this rappresentation because it allow to see the precision, recall, f1_score and accuracy. Below there're a rapresentation of the tests that i did for find the best configuration for the model.



```
                  precision    recall  f1-score   support

      dinnerware      0.518     0.443     0.477       429
     pasta_sides      0.816     0.796     0.806       368
     sushi_plate      0.623     0.603     0.613       353
         Oranges      0.387     0.754     0.512       309
Tea_drink_bottle      0.467     0.515     0.490       400
     Dust_Cloths      0.735     0.421     0.535       297
        Pretzels      0.625     0.722     0.670       316
basket_container      0.607     0.322     0.420       370

        accuracy                          0.565      2842
       macro avg      0.597     0.572     0.565      2842
    weighted avg      0.594     0.565     0.562      2842
```

Figura 14 - VGG16 Classification Report

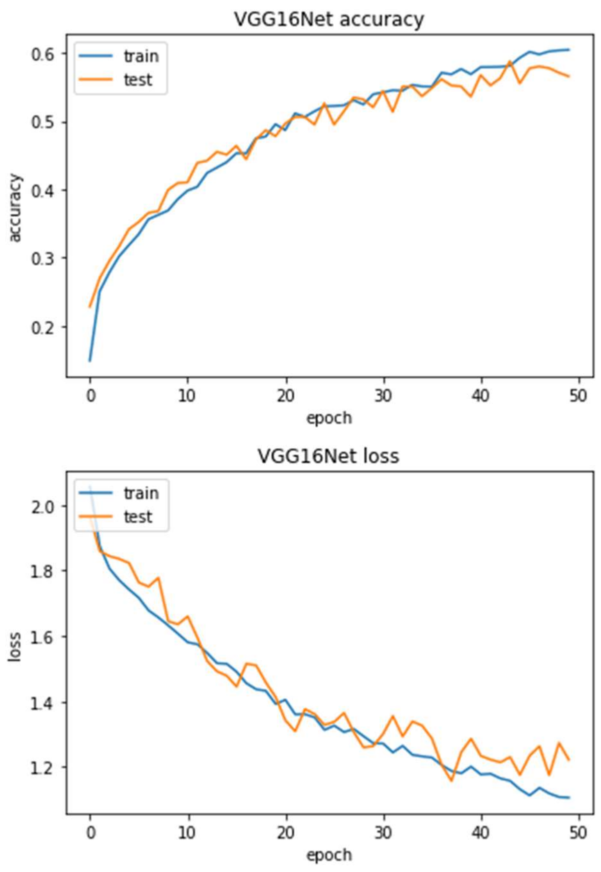


Figura 15 - VGG16 Graphics

We can observe that the accuracy isn't the best, but the model is quite good. With this model we have a slight overfitting but after all the result is good. The model isn't perfect but in this case, seeing that the trend of both accuracy and loss is more or less stable, we can still consider this model good.

5. Conclusion

At the end of this project i can say that the best model for Image Classification is VGG16 even if isn’t perfect. It have more stable loss and accuracy then the other model that we saw (even changeing some parameter). We must also consider that our model is not entirely bad. It must also be considered that the dataset that both models have used have within noisy images and images that doesn’t belong to that specific class, thus introducing small errors.

Riferimenti Sitografici

I. https://neurohive.io/en/popular-networks/vgg16/