# Machine Learning
# homework 2

Saverio Taliani 1841465

January 7, 2023

## Contents

# 1 Introduction

The following discussion has been developed as a solution for the proposed task. The latter consisted in solving an image classification problem, the dataset to be used is on student's choice.
Here is presented a solution based on neural networks on a subsection of ten classes from food-101 dataset.

# 2 Classification

## 2.1 Data Processing

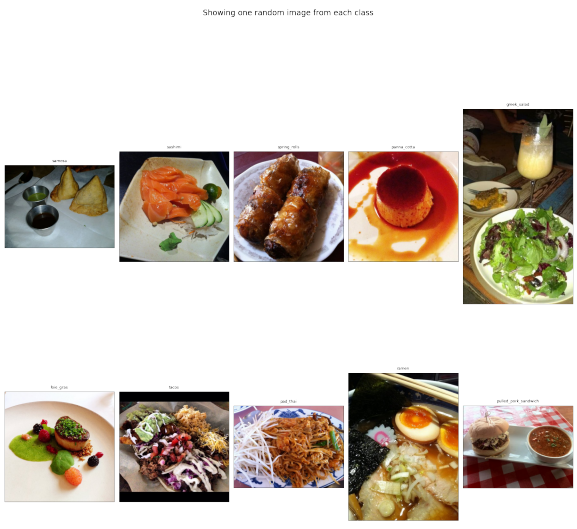
The first choice to take for this homework was the dataset to be used. The following analysis was performed using ten classes from the dataset food-101.
Food-101 is a dataset present in the most common sources, in fact it is available on Kaggle and Tensorflow, anyway in order to avoid to download and split the dataset each time it revealed useful to download it upload the tar.gz file con Google Drive and extract there the ten selected classes and then proceed to separate training and test.
Each class contains 1000 images each one capturing the dish from different angles and different zoom, moreover each picture shows a different preparation with a different quantity of product different plate and surroundings.
It is relevant to highlight that the dataset is noisy, there are some images for each class which does not contain a dish of the considered food but sometimes something different for example humans.

### 2.1.1 Normalization

Before proceeding with the description of the solution algorithm it is worthy to say that the images contained in the dataset where all different in shape and size.
Hence before processing all the images through the neural network, it was necessary to scale them and adapt them to the dimension of $118 \times 224$.

### 2.1.2 Splitting

To prepare data for the algorithm it was necessary to split the dataset into train set and test set.
Among data in the folder there was a text file containing a particular split which was useful to avoid noise in the test set.
The division brought to a train set of 750 units and a test set of 250 units per class. The total train set had dimension 7500 images while test set was composed by 2500 samples.



### 2.1.3 Augmentation

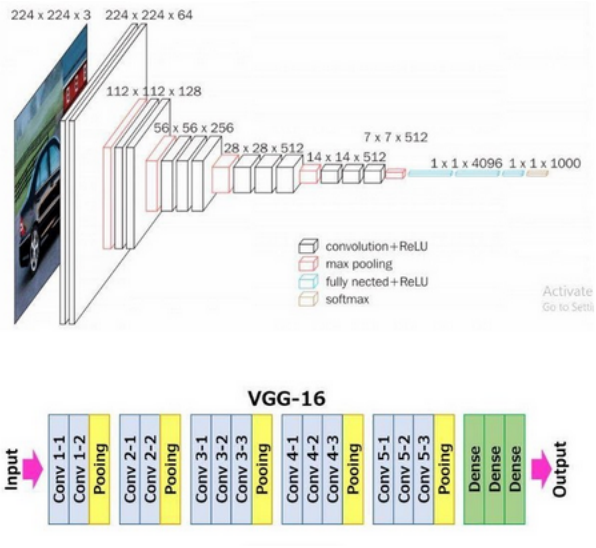
Since the dataset was sufficiently large and in order to keep all the process a bit lighter the augmentation was performed online using the function ImageDataGenerator. The allowed actions consisted in horizontal flips and rotation of a maximum of 20 degree.

## 2.2 Model choice

The choice of the networks was not easy at first. The idea was to try with a small network created ad hoc, but the results were really far to be presentable.
Then it came the idea to use a small network trained for scratch, unfortunately being not in possess of a gpu brings the problem of restricted computational time due to cloud computing restrictions. Again, the results were not satisfying.
At the end searching for benchmarks on the chosen dataset the decision was to try to apply transfer learning by using employing already tested architectures.

### 2.2.1 VGG16

For the first attempts was employed a small network VGG16.
VGG16 is a convolutional neural network (CNN). A CNN has an input layer, an output layer, and various hidden layers. VGG16 is considered to be one of the best computer vision models.




This network has $14,714,688$ trainable parameters, the idea of transfer learning is to use pretrained weights that comes from "ImageNet".
To make the pretrained CNN works on a specific dataset it is mandatory to put on top of it a Dense layer with a number of neurons equal to the number of classes that are taken in consideration and softmax activation function. It is also possible to add some custom layer to better classify samples from your dataset. The resulting network is a combination of the pretrained architecture that is useful to understand better low-level features of the images, while the top layers are more useful to fit on high level features.
In this analysis it was used a sequence of three dense layers with relu and tanh activation functions and two batch normalization layers at the top if the VGG16, moreover two dropout layers were applied in order to reduce overfitting.
Before going into details of the conducted experiments it is worthy to deepen the introduced concepts:

1. Activation functions, The primary role of the Activation Function is to transform the summed weighted input from the node into an output value to be fed to the next hidden layer or as output. In general it adds non linearity to the network.
   tanh:
   $$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1}$$
   softmax:
   $$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2}$$

2. Dense layer, In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer. A dense layer is used for changing the dimension of the vectors by using every neuron.

3. Batch normalization, is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

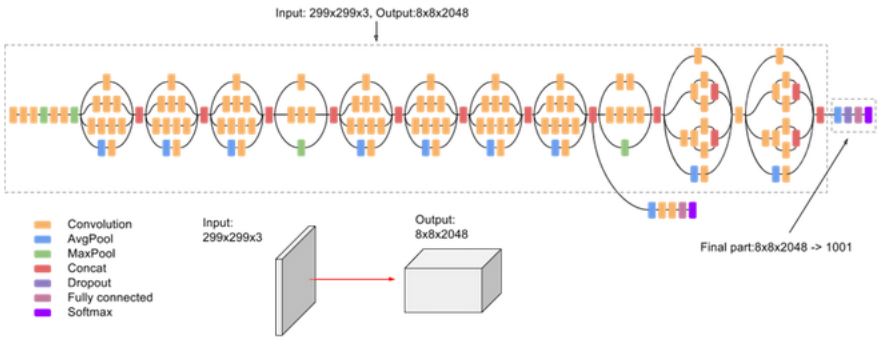
4. Dropout, is a regularization method which brings the CNN to randomly shut down a part of the neurons during training phase in order to reduce over-fitting. The result is that the architecture of the network is constantly changing causing the CNN to adapt and not to rely on a specific set of neurons.

### 2.2.2 Inception V3

The second approach experiment was conducted with an Inception V3 network. It is a superior version of the basic model Inception V1 which was introduced as GoogLeNet in 2014. When multiple deep layers of convolutions were used in a model it resulted in the overfitting of the data. To avoid this from happening the inception V1 model uses the idea of using multiple filters of varied sizes on the same level. Thus, in the inception models instead of having deep layers, we have parallel layers thus making our model wider rather than making it deeper. The Inception model is made up of multiple Inception modules. The basic module of the Inception V1 model is made up of four parallel layers.

- $1 \times 1$ convolution
- $3 \times 3$ convolution
- $5 \times 5$ convolution
- $3 \times 3$ max pooling

Where convolution is the process of transforming an image by applying a kernel over each pixel and its local neighbors across the entire image.
While pooling is the process used to reduce the dimensions of the feature map. There are different types of pooling, but the most common ones are max pooling and average pooling.



From the picture it appears clear that this is a huge model, drastically different from the VGG16 CNN.
This network has $21,768,352$ trainable parameters, to adapt it to the specific problem the same techniques as before so three dense layer this time with relu activation function and the final layer.
Relu function:

$$f(x) = \max(0, x) \tag{3}$$

In the next chapter are presented the results obtained by employing this network.

## 2.3 Evaluation

### 2.3.1 Vgg16

The training of this network went through two steps.

1. Training of the custom layers and last layer of the network.
2. Tuning of all parameters with a small learning rate

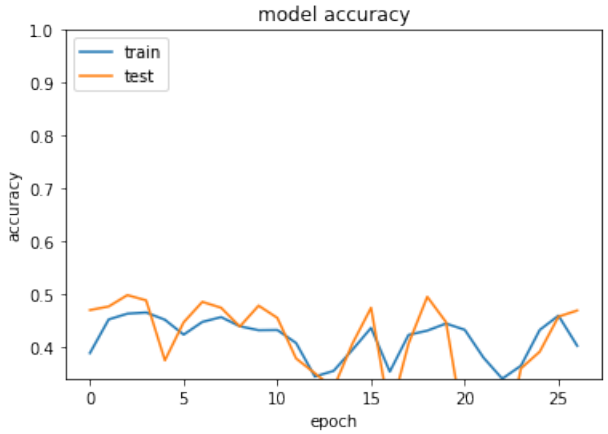
The performance of the network before training where not good because of the presence of the non trained custom layers.

```
84/84 [==============================] - 1692s 20s/step - loss: 2.4713 - accuracy: 0.1060
Test loss: 2.471262
Test accuracy: 0.106000
```

The first experiment was conducted by employing the following parameters

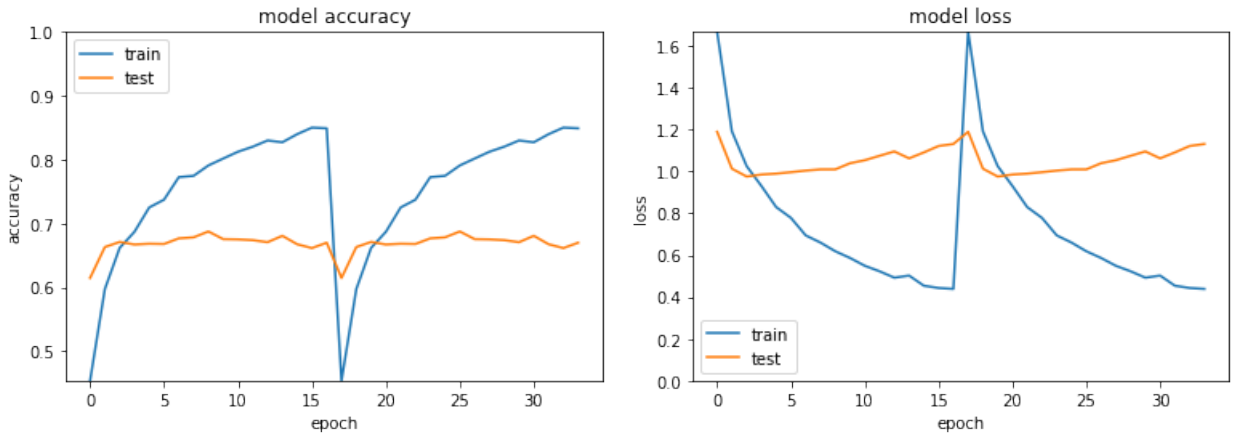
- Batch size: 30
- Activation function for custom layers: tanh
- Trainable parameters: $2,194,014$
- Loss function: cross entropy loss
- Learning rate: $1 \times 10^{-3}$

After 35 epochs of training which last 31 minutes and 52 seconds, the result was a total mess:




The network was clearly unable to learn how to classify the given images. This issue may be related to the problem known as "Exploding gradient" which may affect the training.

Exploding gradient occurs when the derivatives or slope will get larger and larger as we go backward with every layer during backpropagation. This situation is the exact opposite of the vanishing gradients. This problem happens because of weights, not because of the activation function. Due to high weight values, the derivatives will also higher so that the new weight varies a lot to the older weight, and the gradient will never converge. So, it may result in oscillating around minima and never come to global minima point.
One feasible way to avoid this issue is to change activation function, indeed this was the case.
By using the Relu activation function the problem was solved.




(a) Metrics (b) Confusion metrics

The sudden variation in the middle of the training represent the change of training phase, the network reached quickly a 0.65 accuracy then the learning rate was decreased to $1 \times 10^{-4}$ and all the layers were set as trainable. The choice of a smaller learning rate was taken in order to slightly modify weight to better feet the dataset but avoiding large modification that could brought to a performance decrease.
After the switch, the accuracy slowly increased to reach a good 0.792 and this was the maximum result obtained after 16 epochs in a training of 20.

```
84/84 [==============================] - 16s 192ms/step - loss: 0.7551 - accuracy: 0.7840
Test loss: 0.755060
Test accuracy: 0.784000
```

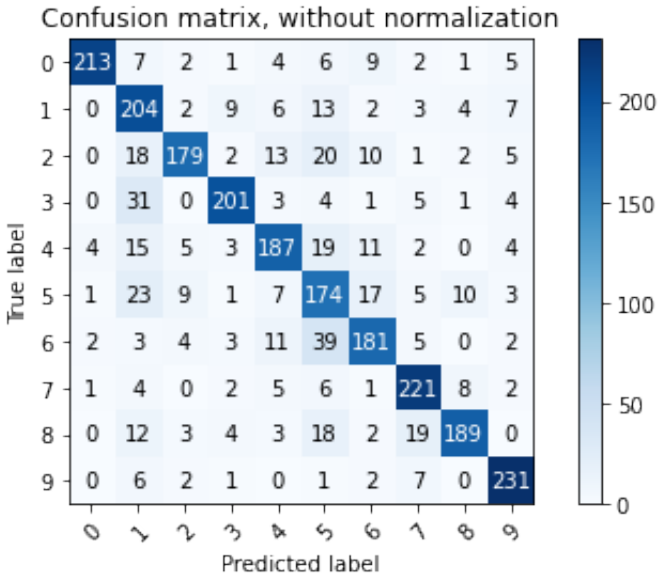
Anyway the accuracy is not that high to be enough to describe the situation. Further analysis are more suitable to understand the nature of this result.

```
84/84 [==============================] - 17s 201ms/step
                    precision    recall  f1-score   support

            samosa      0.964     0.852     0.904       250
           sashimi      0.632     0.816     0.712       250
      spring_rolls      0.869     0.716     0.785       250
       panna_cotta      0.885     0.804     0.843       250
       greek_salad      0.782     0.748     0.765       250
         foie_gras      0.580     0.696     0.633       250
             tacos      0.767     0.724     0.745       250
          pad_thai      0.819     0.884     0.850       250
             ramen      0.879     0.756     0.813       250
pulled_pork_sandwich      0.878     0.924     0.901       250

          accuracy                          0.792      2500
         macro avg      0.806     0.792     0.795      2500
      weighted avg      0.806     0.792     0.795      2500
```

By looking at, precision, recall and f1-score indicators for each class it appear clear that some

classes are easier to learn than others.
For example "Samosa" is by the greatest success of this classification, this is not really surprising since at a visual inspection of the samples they appear very distinguishable thanks to their peculiar shape.

Other classes like "geek salad" and "foie gras" are instead incredibly various in appearance and so very difficult to catch even by humans.
A clear inspection on this aspect is given by the confusion matrix.




Where the order of classes is: samosa, sashimi, spring rolls, panna cotta, greek salad, foie gras, tacos, pad thai, ramen, pulled pork sandwich.

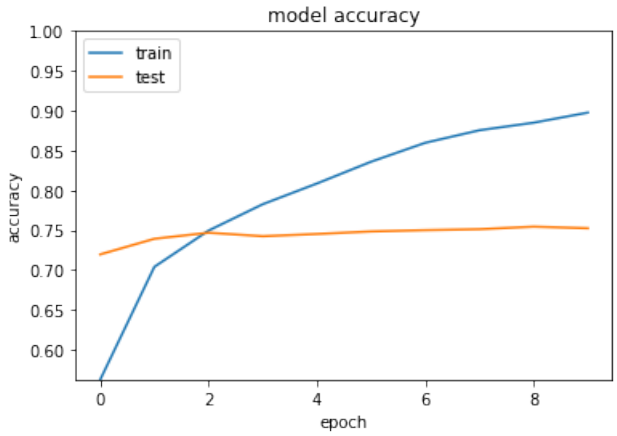
### 2.3.2 Inception V3

Another experiment set of experiments were conducted by using the Inception V3 CNN. Being this a huge model, it is expected that training times are greater, in fact with InceptionV3 it is better to keep a low number of epochs in order not to exceed the computational time given by cloud computing services.
As before on the top of the network there are three custom dense layers following a flatten operation.

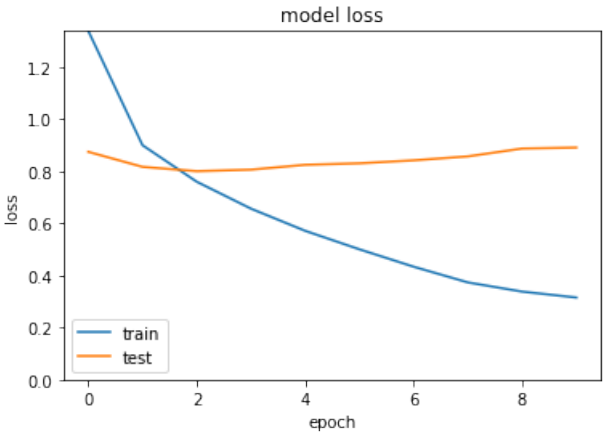
The parameters chosen to train were:

- Batch size: 30
- Activation function for custom layers in order: relu, tanh, softmax
- Trainable parameters: $2,090,470$
- Loss function: cross entropy loss
- Learning rate: $1 \times 10^{-3}$

The trainning of 10 epochs gave the following result:




(c) Metrics




(d) Confusion metrics

In few epochs the training brings the accuracy around 0.90, but the evaluation accuracy does not seem to be really changed reaching a maximum of 0.75, in the meanwhile the loss after a small deflection is increasing, it seems that there might be an over-fitting issue, so in this case the dimension of the network did not come to help. This over-fitting issue is slightly solved by eliminating the custom layers on the top and reducing the learning rate, in this case train and test accuracy grow quite uniformly but very slow and starting from a low accuracy point that would take hundreds of epochs to become satisfying.
Anyway the results obtained in term of:

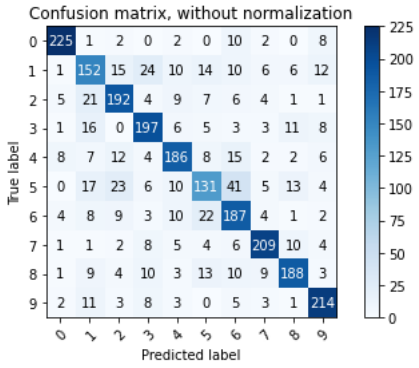
accuracy

```
84/84 [==============================] - 16s 184ms/step - loss: 0.8909 - accuracy: 0.7524
Test loss: 0.890931
Test accuracy: 0.752400
```

f1 score

```
84/84 [==============================] - 14s 172ms/step
                    precision    recall  f1-score   support

            samosa       0.907     0.900     0.904       250
           sashimi       0.626     0.608     0.617       250
       spring_rolls       0.733     0.768     0.750       250
       panna_cotta       0.746     0.788     0.767       250
       greek_salad       0.762     0.744     0.753       250
         foie_gras       0.642     0.524     0.577       250
             tacos       0.638     0.748     0.689       250
         pad_thai        0.846     0.836     0.841       250
             ramen       0.807     0.752     0.778       250
pulled_pork_sandwich     0.817     0.856     0.836       250

          accuracy                           0.752      2500
         macro avg       0.752     0.752     0.751      2500
      weighted avg       0.752     0.752     0.751      2500
```

and confusion matrix




are quite similar to the output of the smaller network. Confirming the hypothesis on which class is easier to classify and which is less understandable

# 3 Conclusions

In conclusion to this analysis, it is possible to extrapolate some interesting aspects to learn from. First noise makes dataset trickier to work with, to be honest the record achieved from ML experts is around 0.95 accuracy so it is not impossible to reach excellent results but is not that easy either.
The second think to understand is that with limited computational power or computational time it is preferable to choose smaller networks that gives the possibility to try different settings and have a better result. This does not mean that smaller networks are better, and this is in fact supported by the fact that the benchmark on this dataset is leaded by deep neural network.
Finally, not all activation functions can work with any dataset so before going crazy because a CNN seems not to learn it is worthy to try different approaches, this could be time demanding but it may bring to the desired results. The learning rate is also a fundamental part in the process because it gives can change a lot the result of a training, high values give a boost at the starting but then may bring to unstable learning process, while to small rates brings slow improvements that sometimes get stuck too far from the desired result (local minima). An effective way to act is to change rates, gradually decrease until convergence.