**Visual Intelligence Course**

**Project report**

Nicolò Squarzoni

a.y. 2022/2023

**Summary**

It all started in 2012, when Stéphane Mallat published a paper on the scattering transform, titled "Scattering Representations for Recognition," in the IEEE Transactions on Pattern Analysis and Machine Intelligence. The scattering transform was introduced as a tool for signal and image analysis, and demonstrated its effectiveness in various recognition tasks, such as object recognition, texture classification, and speech recognition.

Since that time, Mallat’s paper has collected multiple citation and has consistently influenced the research in the field of machine learning and signal processing.

The scope of this project is to classify a dataset of images divided in two classes through a Convolutional Neural Network and through a Multilayer Perceptron fed with the features generated by the scattering transform of the input images, eventually comparing their classification performance.

**Background**

There are considerable differences between doing image classification with a Convolutional Neural Network and with the scattering transform.

The first architecture is a deep neural network divided in two parts: convolutional layers and fully connected layers.

The second architecture uses the scattering transform to perform the feature extraction and the neural network features only fully connected layers.

According to literature they both have advantages and disadvantages and the main ones are listed below.

Advantages of Convolutional Neural Networks:

* Currently the state-of-the-art approach for image classification with very high accuracy on large datasets
* Ability to learn features directly from the raw image data through training, without the need for manual feature engineering
* Pre-trained Convolutional Neural Network models can be fine-tuned on new datasets with similar characteristics, which can save time and computational resources
* Ability to handle variations in image scale, rotation, translation, and illumination.

Disadvantages of Convolutional Neural Networks:

* Typically deeper and more computationally expensive than other approaches, which can limit their use in resource-constrained environments.
* Require large amounts of training data to learn effective features, which can be a limitation for tasks with limited data availability.
* The learned features in Convolutional Neural Networks can be difficult to interpret and understand, which can make it challenging to diagnose and fix errors or biases in the model.
* Possibility to overfitting to the training data, which can result in poor generalization performance on new data.

Advantages of scattering transforms:

* Can be computed efficiently, making them computationally faster than some other approaches
* Ability to handle variations in image scale, rotation, translation, and illumination
* Stability to small deformations in the input image, which can be useful for tasks such as object detection
* The features extracted are interpretable and can provide insight into the underlying structure of the input data.

Disadvantages of scattering transforms:

* Generally less accurate than Convolutional Neural Networks for image classification tasks, especially on large datasets.
* Use of predefined mathematical operations to extract features, which can limit their ability to capture complex and abstract features.
* Not easily transferable to other tasks or datasets, as they are specifically designed for image classification.
* May not be suitable for all image classification tasks, especially those that require highly specialized features or have specific constraints.

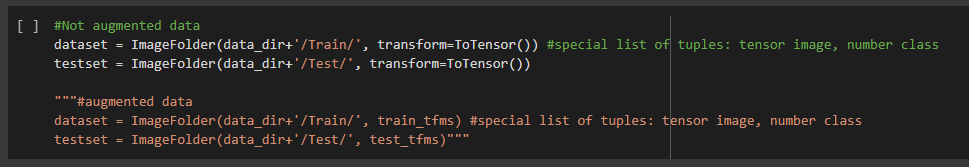
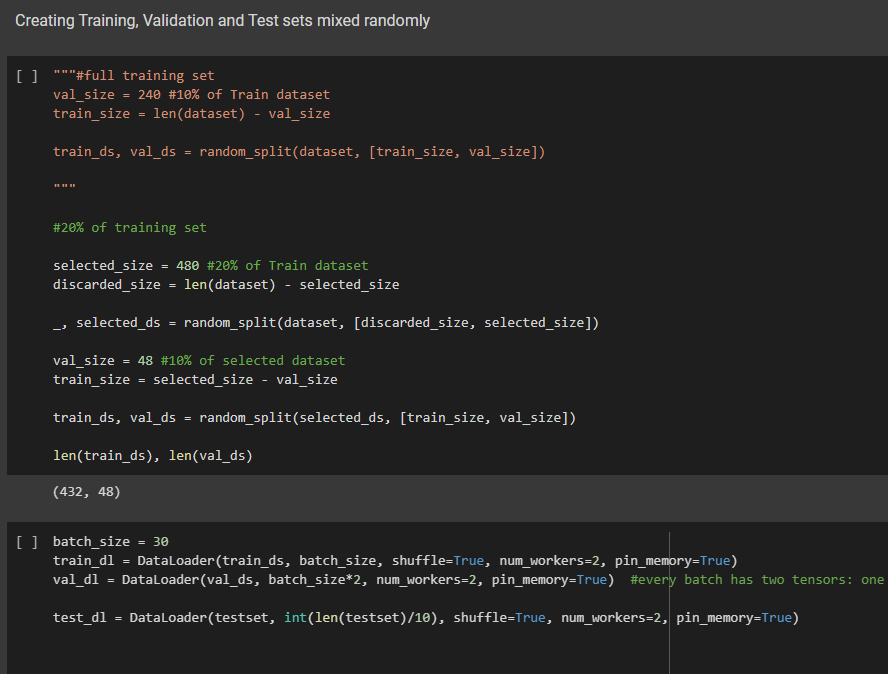
This project aims to empirically verify such advantages and disadvantages.

**Project set up**

The first action has been the selection of the dataset. It has been chosen a dataset of 128 x 128 pixel colored images divided in two classes, flowers and dogs, and each class features a training subset of 1200 images and a test subset of 400 images.

Then it has been chosen the environment where to write the code and run all the tests. Since the neural network training requires a consistent computational capacity and a consistent amount of time, Google Colab has resulted a great option, due to the possibility to access for free GPUs and upload the dataset on Google Drive, other than running multiple sessions in parallel. The result is a user-friendly notebook, featuring almost all the necessary Python libraries, which performs faster and remote computation.

According to the project requirements, the framework used to create and manage the neural networks is Pytorch.

The first implementation step has been the creation of a specific data structure from the image dataset to feed the neural network for training, validation and testing.

Figures: code for loading training and test datasets

After loading the images on Google Drive, some very handful functions from Torchvision library have been used. ‘’ImageFolder’’ imports the images converted into tensors to a list of tuples containing the image tensor and the class label; this allowed to create a data set and a test set.

They were both ordered, meaning that the list had first all the tensor images belonging to the class 0 and then all the tensors belonging to the class 1. This situation was not ideal and it has been necessary to shuffle the tensors in order to optimize the networks training.

The shuffling has been carried out in two steps.

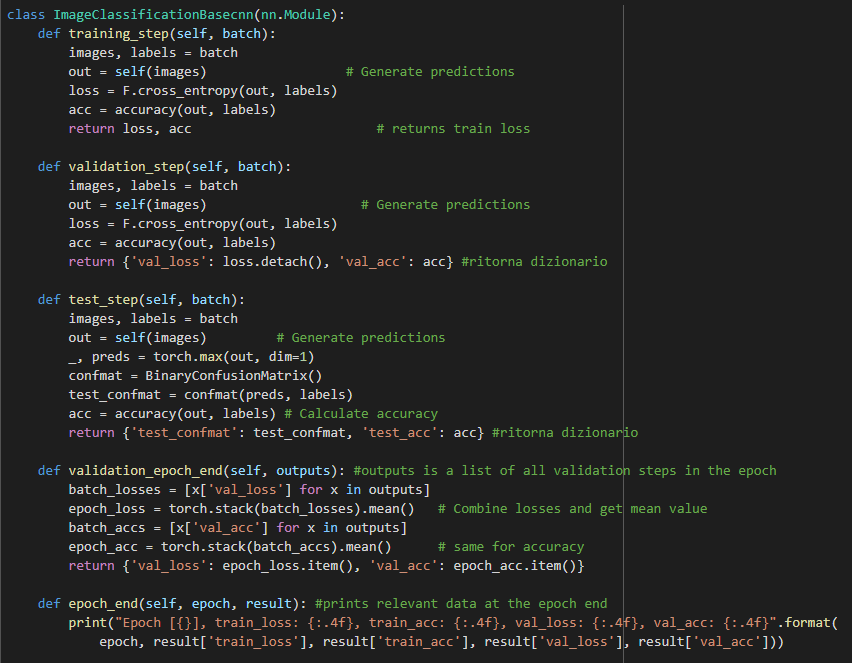
First, during the random splitting of the data set into a training set and a validation set, where the validation set has been defined as 10% of the dataset following the hold out approach, which has been chosen to maintain simplicity in the overall structure.

Second, all the created sets have been used to create three new data primitives called ‘’DataLoader’’ which divided the input set in iterable batches of the selected dimension created randomly. These data structures have been designed to be the neural networks input.

After the data structure design, the classification models have been implemented.

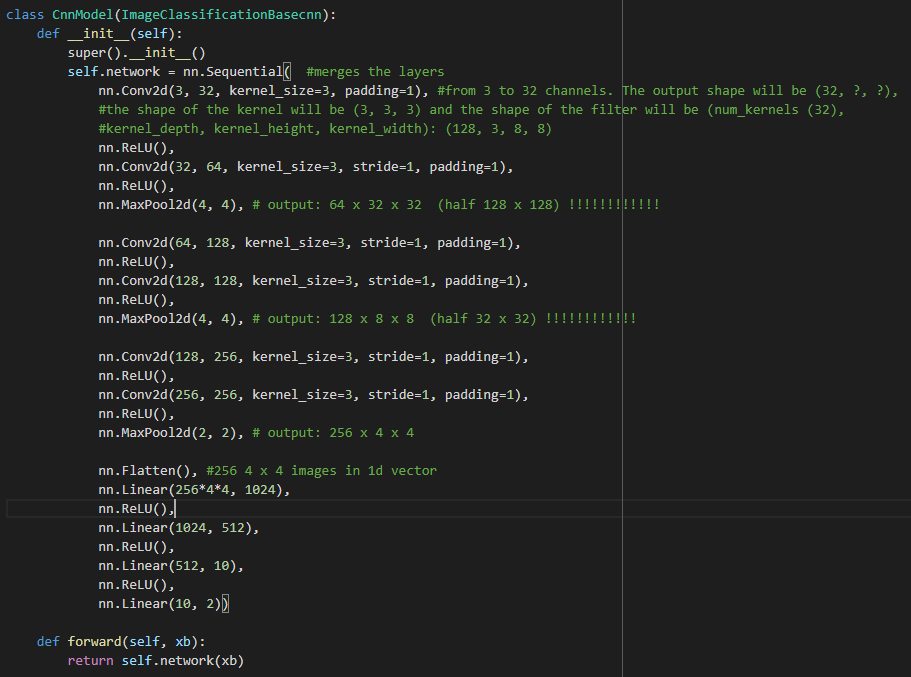
According to the project requirement it has been created a Convolutional Neural Network model and a Multilayer Perceptron designed to receive as input the scattering transform of a set of images. The MLP of the second model had to be the same included in the CNN.

In order to maintain simplicity it has been chosen a basic CNN featuring only convolutional layers and pooling layers for the feature extraction part and a MLP for the for the fully connected part.

 Figure: parent class with methods for models

It has first been created a class inheriting from the base class nn.Module, available from Pytorch library for the purpose of building neural networks. The new class ‘’ImageClassificationBasecnn’’ integrates different methods to manage training, validation and testing, where ‘’validation\_epoch\_end’’ is used in the end of each training epoch to store into a list the average validation loss and accuracy, and ‘’epoch\_end’’ prints at the end of each training epoch average accuracy and loss for debugging purposes.

The CNN class model inherits all the methods from ‘’ImageClassificationBasecnn’’.

 Figure: CNN class

There are four convolutional layers and three pooling layers that transform a 3 channels, 128x128 pixel image in a 256 channels, 4x4 pixel ‘’equivalent image’’.

This equivalent image is flattened into a 1d vector and fed to the MLP that reduce with four layers the number of inputs from 4096 to 2. The relu steps are used to maintain the outputs from 0 and 1 to simplify gradient and backtracking computation, other than to introduce non-linearity.

A very similar approach has been adopted for the scattering + MLP model, with some important differences.

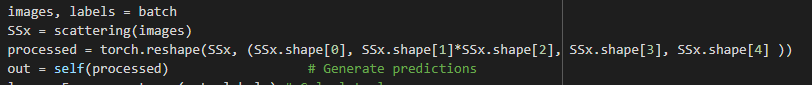
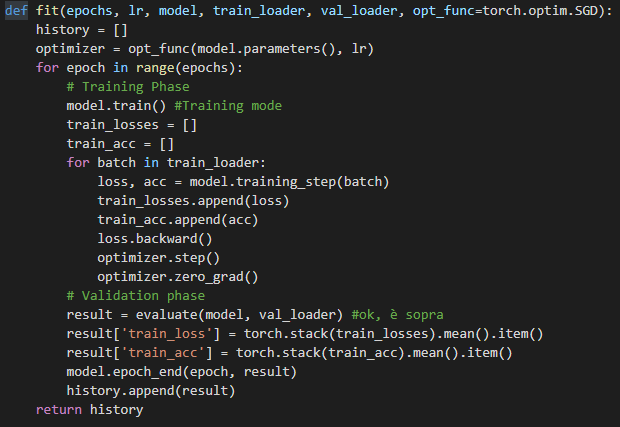
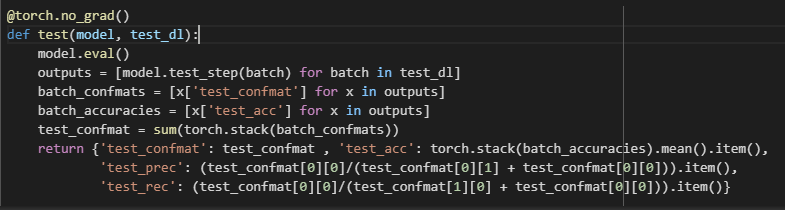
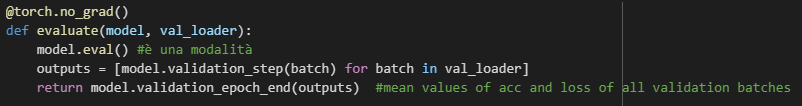
The new class inheriting from the base class nn.Module ‘’ImageClassificationBasescat’’ has a modification on the methods ‘’training\_step’’, ‘’validation\_step’’ and ‘’test\_step’’ where the model prediction is generated because the input is not anymore the image tensor and it is now the scattering transform output.

Figure: Code difference between CNN and Scattering + MLP models methods

As before the scattering class model inherits all the methods from ‘’ImageClassificationBasescat’’.

This model features only the fully connected layers of the CNN model, and only the first layer input is modified to get the all image scattering transform invariants plus the approximation.

After defining the models, the training, validation and testing functions have been created.

Figures: code of training, valideation and test functions

The training function ‘’fit’’, given the hyper parameters, the model and the dataloaders related to the training and validation set, for each epoch performs first the training on each batch of the training dataloader and then calculates the average validation loss and accuracy, storing these values on a list.

The default optimization function is the stochastic gradient descent, and all the training procedure uses Pytorch library functions that consistently simplify the coding.

The ‘’evaluate’’ function used to calculate the average validation loss and accuracy in the epoch features a specific decorator that blocks the gradient calculation in order to save computational capacity.

The testing function computes and returns the four main performance KPI for classification models: confusion matrix, accuracy, precision and recall.

Finally, since the Google Colab runtime tends to disconnect after some hours, in order to not lose all the training data some functions to save on Google Drive the trained model and the training logs have been created together with the functions to retrieve such stored data and the functions to plot the accuracy and loss trends during training.

**Testing and results:**

The first part of the test has been related to the models training. For both models the first tentative training has been performed setting 70 training epochs, a learning rate of 0,0001, and Adam algorithm as optimizer. Adam is considered more efficient than SGD since it tends to optimize each network parameter with a different learning rate.

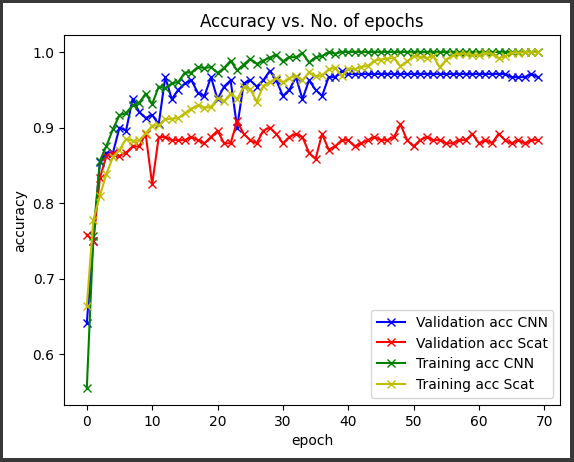
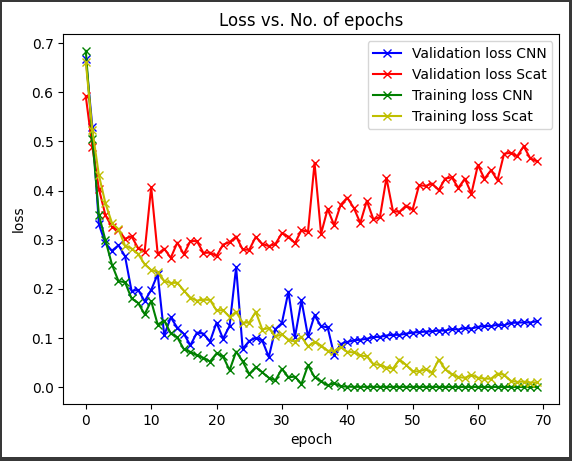
For the scattering + MLP model training it has been necessary to implement the scattering transform. For this purpose it has been chosen to use the Kymatio framework, apparently the most complete and efficient currently available, other than compatible with Pytorch. After importing the related library, it is only necessary to instantiate the object related to the specific 2d scattering class, setting as parameters the scale, the number of angles, the maximum scattering transform level, which cannot be higher than 2, and the input image shape. The following parameters have been chosen to guarantee a similar input to the MLP compared to the fully connected layers in the CNN.



Figure: scattering transform parameters

The object can be fed with the images tensor and the output is a tensor featuring all the scattering invariants at the two levels plus the approximation. This for each image channel.

The results on the first training have confirmed what reported from literature: CNN on a fairly big dataset performed better, with the CNN model getting to a stable 96% of accuracy while the scattering + MLP model sets a bit below 90% of accuracy. It can also been noticed that the scattering + MLP featured some overfitting

Figure: accuracy and loss of first experiment

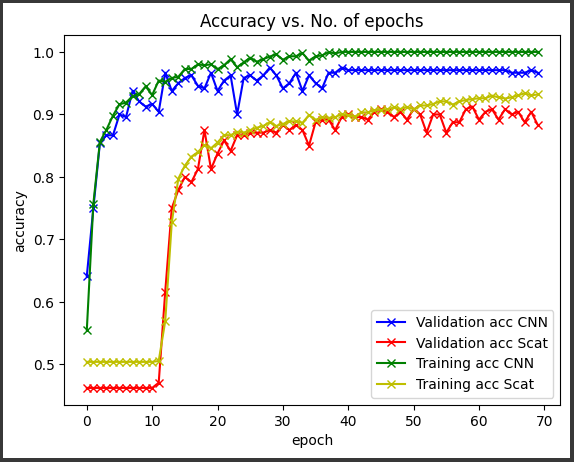
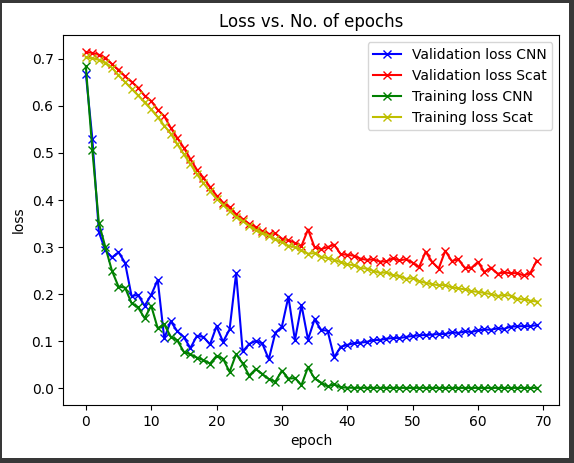
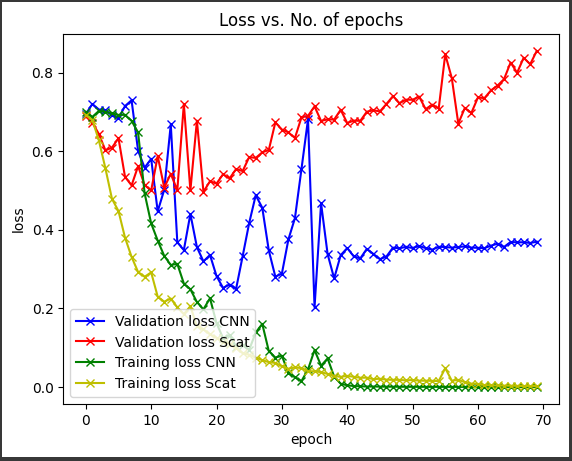
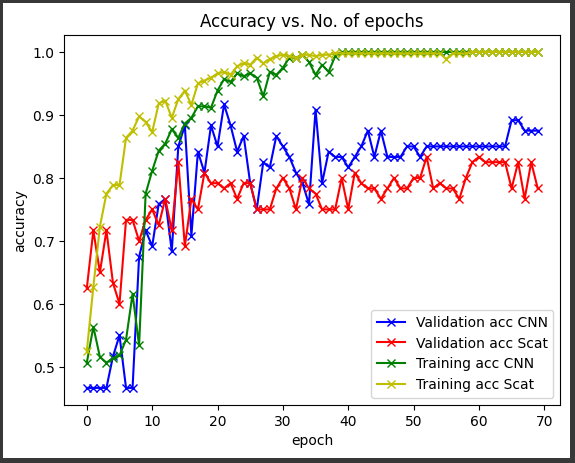
For this reason it has been chosen repeat the training with a learning rate of 0,00001. In this case it can be noticed that the scattering + MPL overfitting has been solved. Testing results are similar to previous

Figure: accuracy and loss of second experiment

Testing results

(class 0 = dogs, class 1 = flowers):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Confusion matrix** | **Accuracy** | **Precision** | **Recall** |
| **CNN** | |  |  | | --- | --- | | 393 | 7 | | 18 | 382 | | 96,87% | 98,25% | 95,62% |
| **Scattering + MLP** | |  |  | | --- | --- | | 369 | 31 | | 54 | 346 | | 89,38% | 92,25% | 87,23% |

Then it has been chosen to repeat the training with only 20% of the training data and a learning rate of 0,0001. In this case it can be noticed that the scattering + MLP model features a strong overfitting.

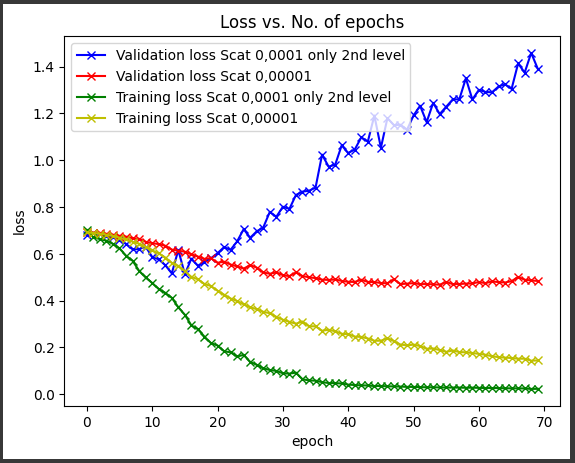
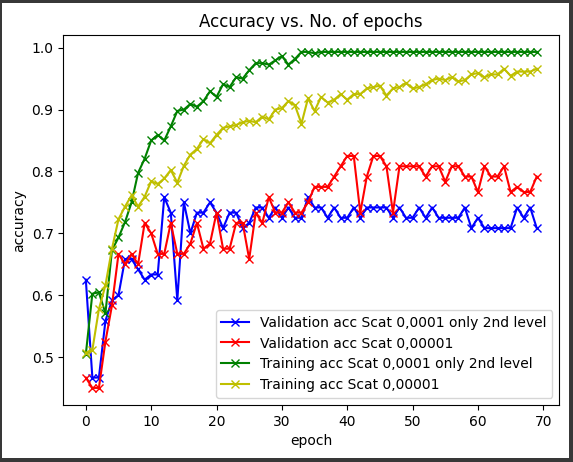
In order to eliminate this effect the model has been modified feeding to the MLP only the second level of scattering transform invariants. It can be noticed from the images below that this action just further lowered the model performance and the solution was instead just to reduce the learning rate by 1/10. Anyway there was still a negative gap between the scattering + MLP and CNN models performance. It has been also added the results with a simpler CNN used for further experiments later on, featuring 3 convolutional layers respectively with 8, 16 and 32 filters. In this case it can be noticed that the gap with the scattering + MLP model is smaller.

Figure: accuracy and loss of fourth experiment

Testing results: (class 0 = dogs, class 1 = flowers):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Confusion matrix** |  | **Accuracy** | **Precision** | **Recall** |
| **CNN 4 layers** | |  |  | | --- | --- | | 378 | 22 | | 39 | 361 | |  | 92,37% | 94,5% | 90,65% |
| **CNN 3 layers 8-16-32** | |  |  | | --- | --- | | 342 | 58 | | 46 | 354 | |  | 87% | 85,5% | 88,14% |
| **Scattering + MLP** | |  |  | | --- | --- | | 344 | 56 | | 74 | 326 | |  | 83,75% | 86,00% | 82,3% |

Another interesting aspect to analyze is the difference between the CNN convolutional filters and the scattering filters.

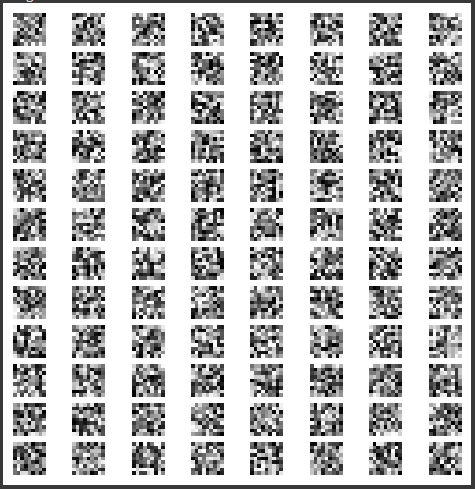
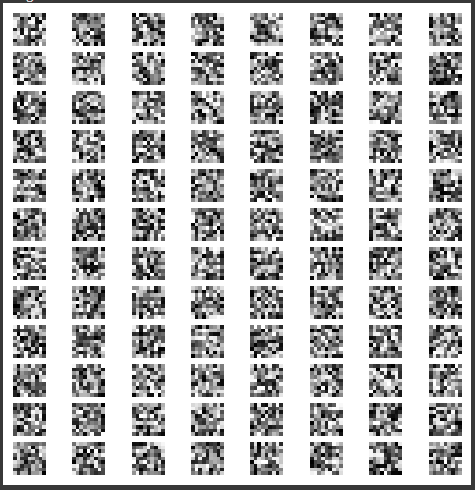
Two different functions have been crated to plot the first layer of filters for the CNN and the wavelet filters related to the scattering transform.

About the CNN filters, it has been possible to color plot only the first layer because they are the only ones featuring three channels, thus being plottable as a colored image. Below the two sets of filters before and after the CNN training.

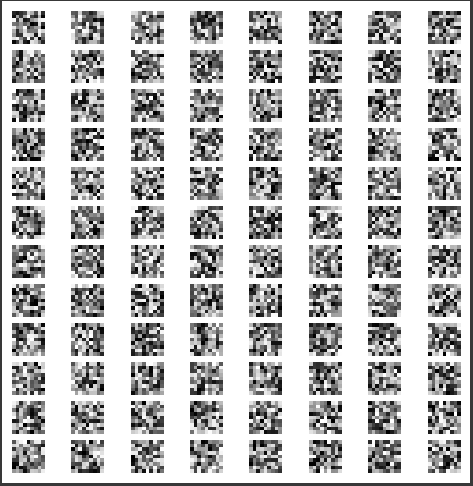
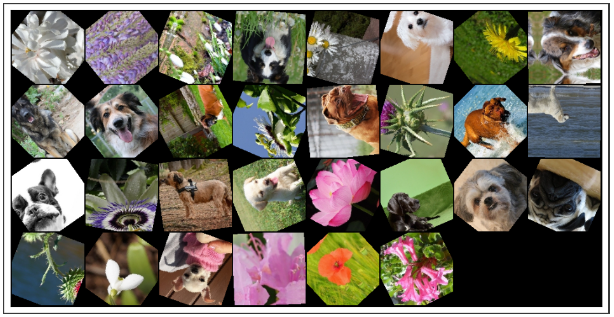
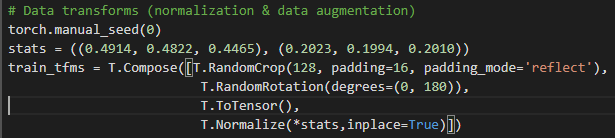
Figure: filters of the first layer of the first model of CNN (RGB)

Since from 3x3 kernels it is not possible to recongize any visual feature after the training it has been used an other CNN model, featuring 9x9 kernels in the first two convolutional layers. This model is the one used during the lab session. The

images show the first layer of filters before and after the standard training separating the three channels. It can be noticed that not really clearly visible features appear in the filters after the training.

 Figure: filters of the first layer of the second model of CNN gray scale

Thus a second training has been performed on the model where images were rotated with random angles in a range 0 -180° and cropped, thus performing data augmentation.



Figures: filters of the first layer of the second model of CNN gray scale and data augmentation code and example

Even after this training it is not possible to notice clear visual features in the filters. An other attempt performing only rotations between -45° and + 45° on the dataset images has been done without any better result even inreasing the numbers of training epoch to 150. In all the above tests the validation accuracy at the end of the training was always around 92%.

The following experiment was then to change the model reducing the number of filters in the first layer from 32 to 16 and reducing the number of convolutinal layers from 4 to 3, so 16, 32, 64 filters.

Finally some results could be noticed and also it could be observed that performing data augmentation with random rotations between -45° and + 45° increased the visibility of visual features in the filters, reaching an accuracy of about 90%. (left image training without data augmentation, right image with data augmentation)



Figure: filters of the first layer of the second model of CNN gray scale without and with data augmentation

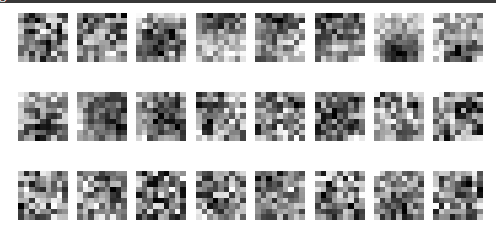
Since there were still several filters randomly filled it has been decided to repeat the test with a model with three layers and 8, 16, 32 filters respectively. The only better result obtained though was the accuracy, increased to 96%.

Figure: filters of the first layer of the third model of CNN gray scale

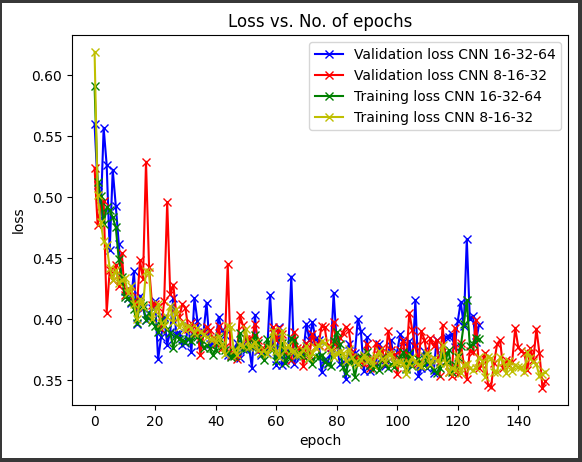
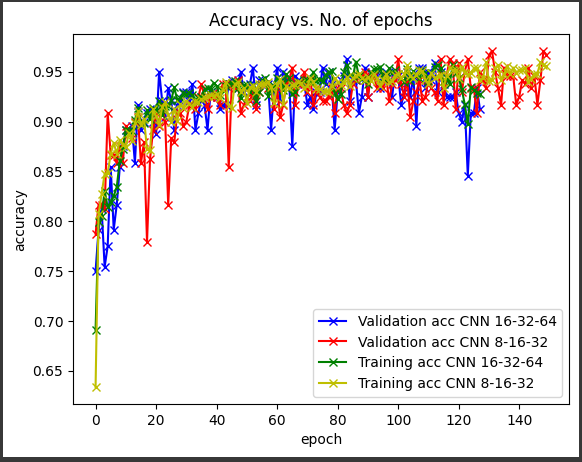
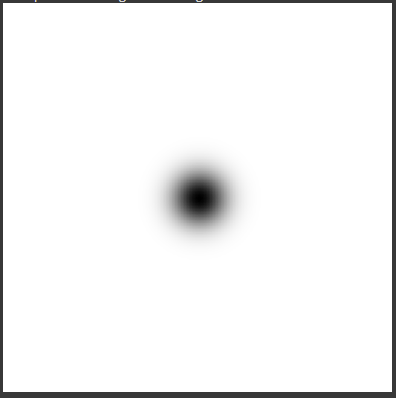
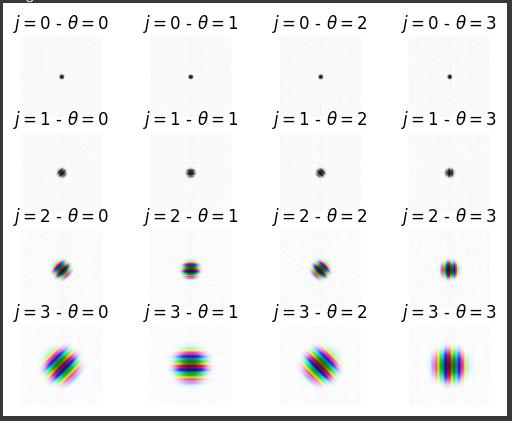


Figure: accuracy and loss of described experiment

To explain the observed results on different models one hypothesis could be that a too complex model can not capture well visual features from a not very wide and various dataset.

Finally below it is shown a representation of Morlet wavelet filters used to implement the scattering transform in Kymatio framework plus the scaling function

**Annex**

CNN models used after the first presented. In order: CNN used in lab (4 convolutional layers 32,32,64,64), CNN with 3 convolutional layers 16,32,64, CNN with 3 convolutional layers 8,16,32.

