Automated Decision Making Project

MILP for adversarial instances in neural networks

Matteo Mosconi

Dataset used: cats vs. dogs

The dataset has 25’000 items for the training part, 12’500 are cats, the label is 0, and 12’500 are dogs, the label is 1. It has 5’000 unlabeled images for the testing.
The images are colorized and are all real picture, so they have different dimensions. I’ve applied some operation in the preprocessing part in order to be able to feed them to the selected network properly.
The chosen dimension is 100 * 100 (* 3, if we want to preserve the color).

0

1

https://www.kaggle.com/competitions/dogs-vs-cats/data

Neural Networks

The main type of network used is the classical fully connected one. Not only is the most classical of the network paradigms but it is also the one cited by Fischetti and Jo in the paper “Deep Neural Networks and Mixed Integer Linear Optimization”. The structure chosen here is as follow:

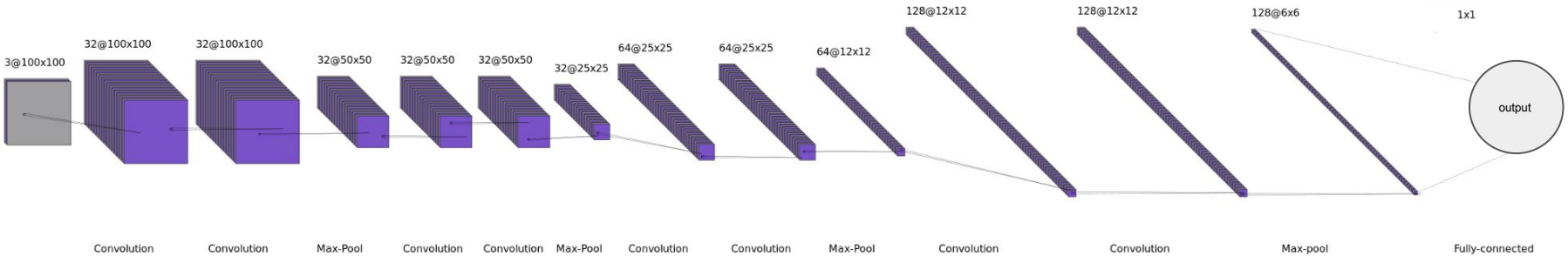
- a first layer of 32 neurons densely connected to the input
- ReLU
- a second layer of 32 neurons densely connected to the layer before
- ReLU
- the final layer, composed by only one neuron, it is sufficient due to the fact that the task is a binary classification, so if the output is closer to 1, the prediction will be dog, vice versa it will be cat.
- finally a sigmoid, it simply helps the output to stay in the range between 0 and 1.

```
# Structure of the net
self.net = nn.Sequential(nn.Flatten(),
                         nn.Linear(self.input_dimension, 32), nn.ReLU(),
                         nn.Linear(32, 32), nn.ReLU(),
                         nn.Linear(32, self.output_dimension)
                         )
```

The problem with FC networks in image classifications, and more in general in computer vision tasks, is that they are not able to extract features with that mechanism that its called receptive fields, that instead is one thing that a CNN (Convolutional Neural Network) does very well. This type of network is in fact the state of the art for what concern image classification. In any case for simple images (like numbers of the MNIST or other black and white simple images) FC-only approach could be effective (an example could be recognition of letters and numbers on a vehicle license plate).

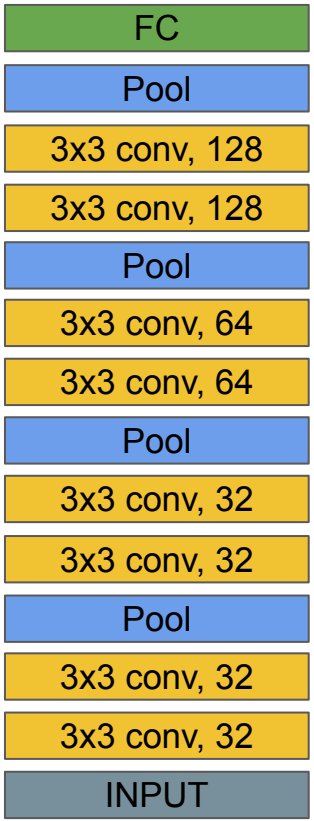
So I’ve tried as an additional result to implement, train, and model also a CNN architecture.

CNN architecture



Some details

Input: [100x100x3] or [100x100x1]	params: 0
Conv3-32: [100x100x32]	params: (3*3*3)*32 = 864
Conv3-32: [100x100x32]	params: (3*3*32)*32 = 9216
Pool2: [50x50x32]	params: 0
Conv3-32: [50x50x32]	params: (3*3*32)*32 = 9216
Conv3-32: [50x50x32]	params: (3*3*32)*32 = 9216
Pool2: [25x25x32]	params: 0
Conv3-64: [25x25x64]	params: (3*3*32)*64 = 18432
Conv3-64: [25x25x64]	params: (3*3*64)*64 = 36864
Pool2: [12x12x64]	params: 0
Conv3-128: [12x12x128]	params: (3*3*64)*128 = 73728
Conv3-128: [12x12x128]	params: (3*3*128)*128 = 147456
Pool2: [6x6x128]	params: 0
FC: [1x1x1]	params: 6*6*128*1 = 589824
	Total number of params: 309600



Training and testing

I’ve trained the two types of networks both with color images and grayscale ones. For 10 epochs over the entire training dataset, with a learning rate of 10^(-3). The accuracy ratios are as expected, there is basically no difference between colorized and grayscale images, and that’s because a dog is different from a cat more from the shape point of view than from the color.

The accuracy of the FC is around 62%.

The accuracy of the CNN is around 82%.

In any case due to the fact that the purpose of the project is to generate adversarial instances by optimizing a properly created mathematical model, I will take as sample for the adversarial generation only the ones that are classified well by the networks, in this way we will have a sample that fools the network on something that the network itself classifies correctly and with a good certainty.

Utilizzo
44%

Memoria GPU
2,0/14,0 GB

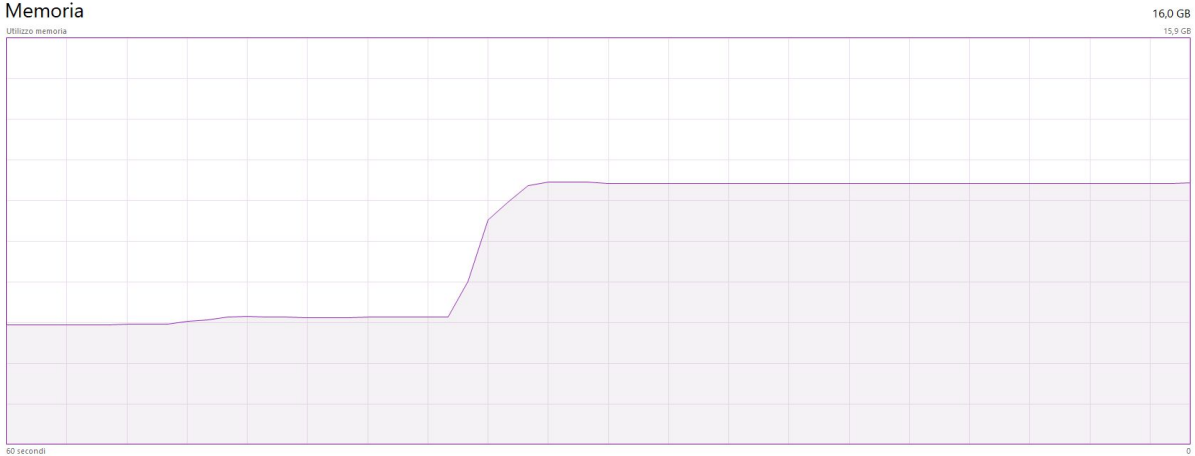
Memoria GPU dedicata
1,9/6,0 GB

Memoria GPU condivisa
0,1/8,0 GB

Temperatura GPU
65 °C

Versione driver:	27.21.14.5751
Data driver:	22/11/2020
Versione DirectX:	12 (FL 12.1)
Località fisica:	Bus PCI 1, dispositivo 0, funzione 0
Memoria riservata per l'hardware:	92,0 MB

Memoria



In uso (compressa)
10,2 GB (554 MB)

Disponibile
5,7 GB

Vincolata
16,1/22,9 GB

Cache
4,6 GB

Pool di paging
426 MB

Pool non di paging
433 MB

Velocità:	2667 MHz
Slot utilizzati:	2 di 4
Fattore formato:	DIMM
Riservata per l'hardware:	84,5 MB

MILP

The MILP for the FC network follows the approach used in the paper “Deep Neural Networks and Mixed Integer Linear Optimization” from Fischetti and Jo for the basic model, plus the constraint and the objective function related to the generation of adversarial instances:

$$\min \sum_{k=0}^{K} \sum_{j=1}^{n_k} c_j^k x_j^k + \sum_{k=1}^{K} \sum_{j=1}^{n_k} \gamma_j^k z_j^k$$

$$\left.\begin{array}{r} \sum_{i=1}^{n_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k \\ x_j^k, s_j^k \geq 0 \\ z_j^k \in \{0,1\} \\ z_j^k = 1 \rightarrow x_j^k \leq 0 \\ z_j^k = 0 \rightarrow s_j^k \leq 0 \end{array}\right\} k = 1, \ldots, K,\ j = 1, \ldots, n_k$$

$$lb_j^0 \leq x_j^0 \leq ub_j^0, \qquad j = 1, \ldots, n_0$$

$$\left.\begin{array}{r} lb_j^k \leq x_j^k \leq ub_j^k \\ \overline{lb}_j^k \leq s_j^k \leq \overline{ub}_j^k \end{array}\right\} k = 1, \ldots, K,\ j = 1, \ldots, n_k.$$

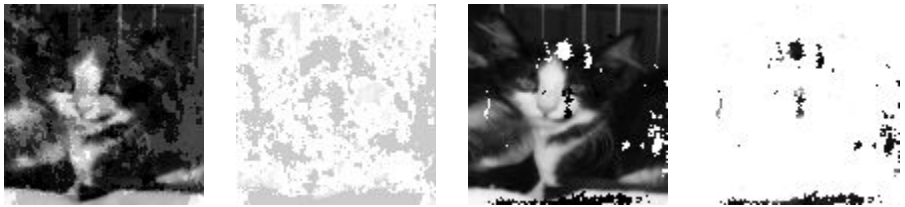
$$-d_j \leq x_j^0 - \tilde{x}_j^0 \leq d_j,\ d_j \geq 0, \qquad \text{for } j = 1, \ldots, n_0$$

```
if sample_label == 1:
    m.addConstr(out <= -0.001)
elif sample_label == 0:
    m.addConstr(out >= 0.001)
```

A key role in the constraints is played by the inequality related to out value (which can be greater or smaller than 0, dependently of what the starting label of the sample that we want to modify is).

Additional constraints can be added based on our needs, like the one that try to limit the modification that each pixel will have:

```
# add constraints on some limitations on pixel distances
m.addConstr(d[j] <= 0.2)
```



Another constraint, or better a component of the objective function in this case, could be for example, to not limit the final values to be over or under the 0 threshold, but also to maximize or minimize it bringing it near to the value of the label opposite to the one in entrance.

For more details on the implementation it is better to see the code directly, especially for the model born from the CNN. In any case, reported here there is the only hint that the paper cited before gives for what concern max pooling, a crucial operation in CNNs.

$$x = MaxPool(y_1, \ldots, y_t) = \max\{y_1, \ldots, y_t\}$$

$$\left.\begin{aligned} \sum_{i=1}^{t} z_i = 1 \\ x \geq y_i, \\ z_i = 1 \rightarrow x \leq y_i \\ z_i \in \{0,1\} \end{aligned}\right\} i = 1, \cdots, t$$

Testing and results

As said before for the testing I've created a script that selects a batch randomly and return the best predicted sample for that batch by the network. In this way we are sure that we try to fool a network that "knows" a to classify that particular sample correctly.

Once the optimization of the model is finished the adversarial instance is produced and we can test if it can fool the prediction of the network as it should.

Another interesting thing that it can be tested is to try with the adversarial instance, produced by the fully connected network model, to fool the convolutional one. For the examples with which I've tried this is not possible, and it is easy understandable due to the higher robustness of the CNN and the different mechanism that it contains.

In the following I've reported some examples of result produced by the fully connected model and finally some conclusions.

Conclusions

I want to conclude by saying what this project (and the course) has thought to me. Neural networks are an effective way to achieve many tasks, as obviously also mathematical optimization is, but the combination of the two can brings very interesting results in different domains of application.

The fact that a neural network can be converted to a mathematical model (even if very big, see the code related to the CNN) can be exploited in order to use the ability of a MILP solver to find an optimal solution (or in any case a very good heuristic one) to test the robustness of the network.

The drawback is that state of the art networks are more and more complex, in terms of both number of layer and types of operations that are present inside them. So if one wants to solve them as a MILP problem, other than an hardware powerful enough to do so, he will need some more focused and tailored optimization techniques that help to simplify the model itself, or in any case something that can help the solver (like gurobi) to solve.