Advanced Algorithms project report

Giacomo Fabris
University of Trento
giacomo.fabris-1@studenti.unitn.it

1. Introduction

Objective of the project is to create a classifier which is able to determine which illness (if any) affects the lungs of a patient, based on their thoracic X-rays images. Illnesses shall be classified in 4 categories: none, bacterial infection, COVID-19, viral infection (non-COVID).

The dataset on which the classifier is trained is composed of about 6000 examples, made of (a) the black-white thoracic X-rays image, 256 pixel wide and high, and (b) a feature vector of 84 entries, extracted from the image. Of the entire dataset, 60% shall be considered for training, 20% for validation and 20% for testing. It shall be noted that the amount of examples for each class varies greatly: the examples of COVID-19 are very few compared to the other classes, which are more equally represented.

Performance shall be measured as the average over classes of the sensitivity, i.e. the proportion of examples from a given class correctly identified as such.

2. Proposed Method

I considered the use of neural networks to solve the classification task, performing two main experiments:

1. using as input the feature vectors only, I implemented a simple fully connected feed-forward neural network;

2. using as input both the pictures and the feature vectors, I implemented a more complex neural network, composed of a convolution stage on the image and a fully-connected stage which collects the result of the CNN and the feature vector, producing the output.

2.1. Fully connected network on feature vectors

The feed-forward neural network is composed of 4 layers of, respectively, 84, 42, 21 and 4 perceptron. Increasing further the number of layers had no sensible effect on the precision of the model; similarly, the effect of changing the number of perceptrons of each hidden layer was not dramatic, above a certain threshold. Each convolution step is composed of a convolution I experimented with two optimizers, Stochastic Gradient Descent and Adagrad, and with two activation functions, ReLU and sigmoid. The final evaluation has been performed with the Adagrad optimizer, setting an initial learning rate of 5e-4, without weight decay. ReLU has been chosen as activation function. The training has been scheduled using a multi-step learning rate scheduler with a milestone at the 20th epoch; the entire learning lasts 35 epochs. The loss function being used is a weighted cross-entropy loss, where weights compensate the fact that the dataset is not homogeneous in terms of number of instances. For any class C of the dataset D, the weight has been set to $1000|C|^{-1}$; alternative weights of $|C|^{-2}$ and $|C|^{-0.5}$ has been considered but resulted in an unbalanced true positive rate between classes.

2.2. Deep network on image dataset

The deep network is composed of two main segments. The first involves the image processing, and is composed of two main convolution steps, while the second is a feed-forward network which processes the output of the convolution network and the feature vector input. ReLU is chosen as activation function for each step of the network.

Each convolution step is composed of a convolution layer with kernel size 5*5 and stride 1; a dropout layer with $p = 0.2$ is inserted after each convolution; then, max pooling with a 3*3 filter is performed to reduce dimensionality.

The output tensor is then linearized and fed into a first layer of perceptrons and normalized; the same processing is performed on the feature vector. Then, the results of the two normalizations are concatenated together into a single vector, which goes through an additional dropout layer with $p = 0.2$ and two perceptron layers.

The network comprises dropout layers because while experimenting I had issues with overfitting, thus I considered both L2 regolarizers (weight decay) of the loss function and dropout layers; I chose the latter because it yielded a better performance of the model in terms of the accuracy measure defined in the problem statement.

Training has been performed using the Adagrad optimizer, setting an initial learning rate of 1e-3. Similarly to the previous network, the loss function is cross-entropy loss, whose weights have been set proportional to the inverse of the number of examples of each class.

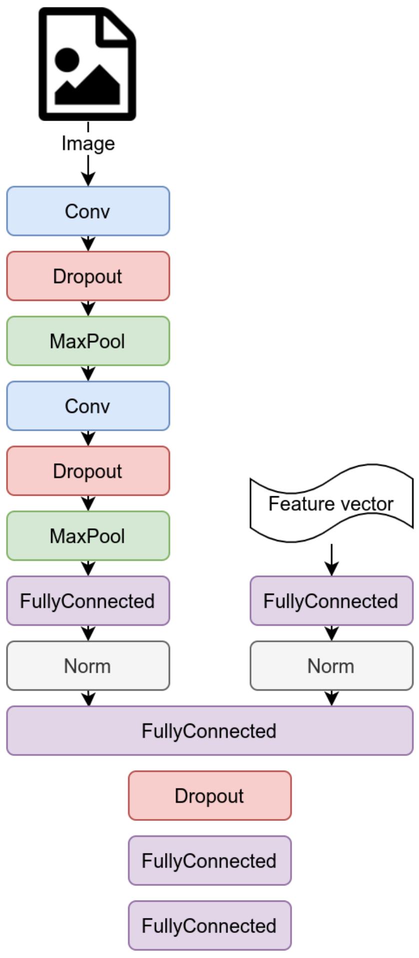


Figure 1: Deep network structure

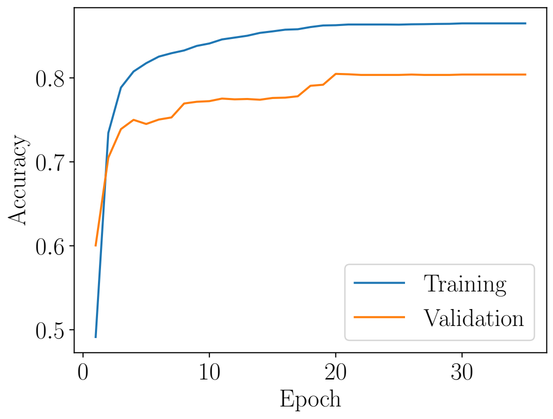


Figure 2: Network #1: learning performance

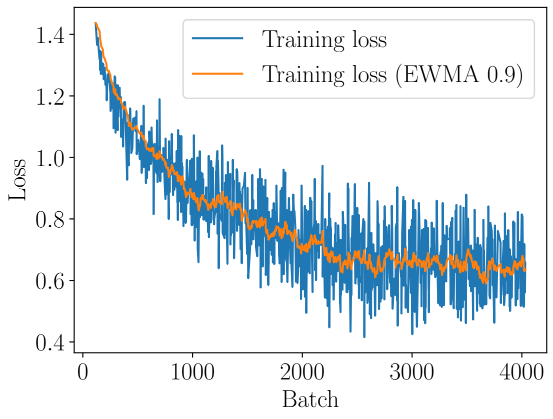
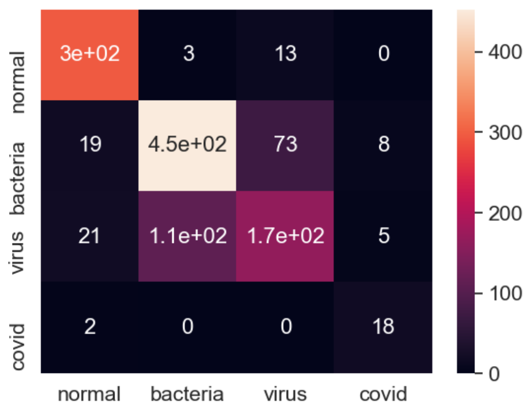


Figure 3: Network #1: loss value of each training batch

3. Results

The algorithm has been implemented using PyTorch; the original source code is contained in the `src` directory of the attached archive. The folder `models_saved` contain the trained network and the run log of the experiments reported in this section. The `models` subfolder contains the serialized weights of the network, which may be loaded by the `nn.py` or `deepnn.py` source file; the `output` subfolder contains the test run output and the `tb_logs` subfolders contain the telemetry data of the training collected using tensorboard. The folder `report` contains the source file of this document and the code used for the generation of plots.

The following graphs show two experiments, one with each network described in the previous section; the performance after each epoch is evaluated and the final confusion matrix is shown.



	normal	bacteria	virus	covid
normal	3e+02	3	13	0
bacteria	19	4.5e+02	73	8
virus	21	1.1e+02	1.7e+02	5
covid	2	0	0	18

Figure 4: Network #1: confusion matrix

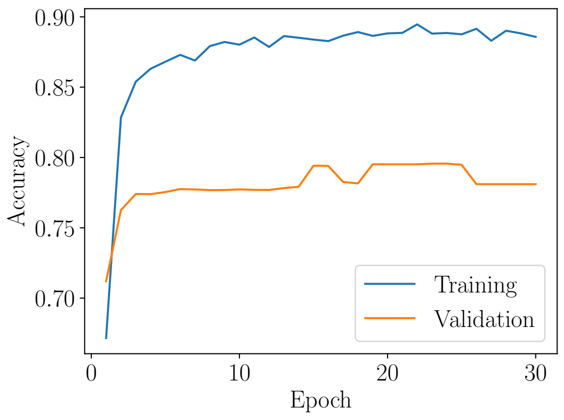


Figure 5: Network #2: learning performance

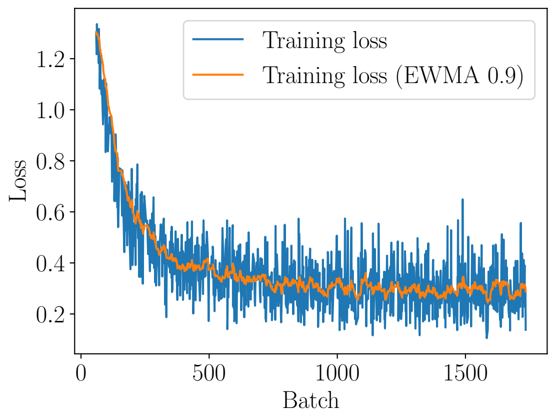


Figure 6: Network #2: loss value of each training batch

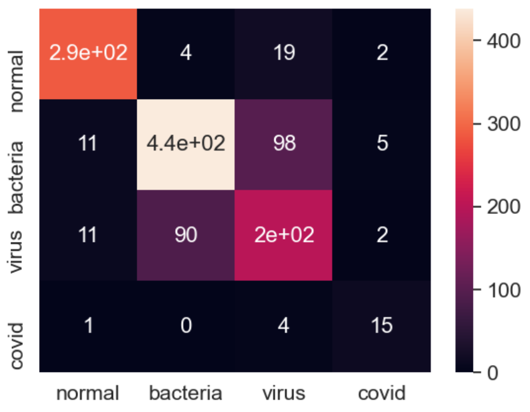


Figure 7: Network #2: confusion matrix