AML Assignment 3 - report

April 28, 2020

Fabio Montello (1834411), Francesco Russo (1449025), Michele Cernigliaro (1869097)

1 Overview

In this report we proceed to answer as requested all the questions of the homework 3. For each point we are going to write a brief description using figures and formulas whenever needed.

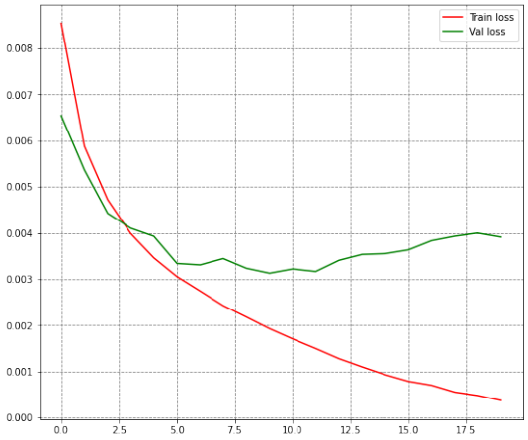
2 Question 1a

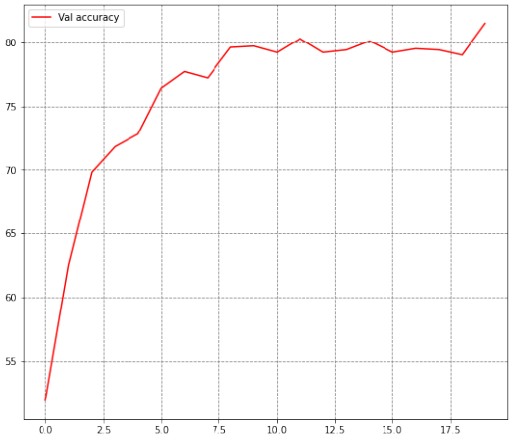
Here below we present the ConvNet implementation with pytorch:

```
class ConvNet(nn.Module):
    def __init__(self, input_size, hidden_layers, num_classes, norm_layer=None):
        super(ConvNet, self).__init__()
        layers = []
        # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
        # First ConvBlock with input size (i.e. C=3) and first hidden layer(i.e. 128)
        layers.append(nn.Conv2d(input_size, hidden_layers[0], kernel_size=3, stride=1, padding=1))
        if norm_layer=="BN":
          layers.append(nn.BatchNorm2d(hidden_layers[0], eps 1e-05, momentum 0.1,
                                       affine=True, track_running_stats=True))
        layers.append(nn.ReLU())
        layers.append(nn.MaxPool2d(kernel_size 2, stride 2))

        # Adding the other blocks
        for Din, Dout in zip(hidden_layers[:-1], hidden_layers[1:]):

          layers.append(nn.Conv2d(Din, Dout, kernel_size=3, stride=1, padding=1))
          if norm_layer=="BN":
            layers.append(nn.BatchNorm2d(Dout, eps=1e-05, momentum=0.1,
                                         affine=True, track_running_stats=True))
          layers.append(nn.ReLU())
          layers.append(nn.MaxPool2d(kernel_size=2, stride=2))

        # stacking convolutional blocks
        self.ConvBlocks = nn.Sequential(*layers)
        self.Dout = hidden_layers[-1]

        # fully connected layer
        self.Dense = nn.Linear(hidden_layers[-1], num_classes)
```

Notice that we included also the Batch Normalization as requested in the point 2a, but the code

has been executed without any normalization layer set ("norm_layer" flag set to None). We will plot the traces of the loss and accuracy values along 20 training epochs:



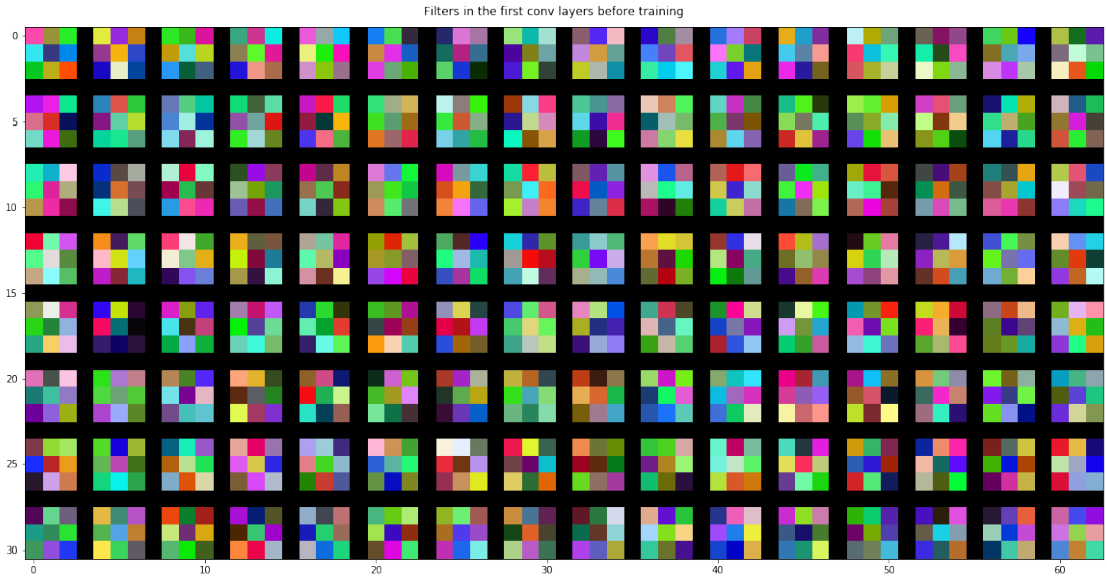


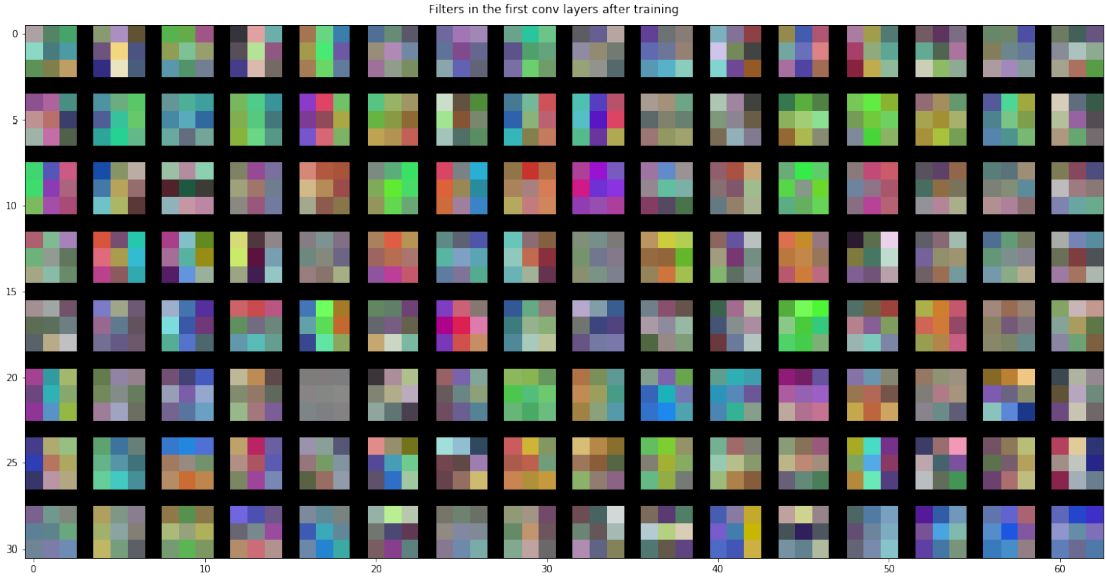
Our best result (see early stopping at question 2b) obtained 81.4% of accuracy on the validation and 79.3% on the test set. According to the loss graph, it seems that model is slightly overfitting starting from the seventh epoch. We will try to reduce the overfitting in the next steps with data augmentation and dropout, as requested from the homework.

3 Question 1b

Implementing the function 'PrintModelSize' we found out that our convnet has 7,678,474 parameters without the batch layers, and 7,682,826 considering also the new scale (γ) and shift (β) parameters to be learned in each BN layer.

4 Question 1c



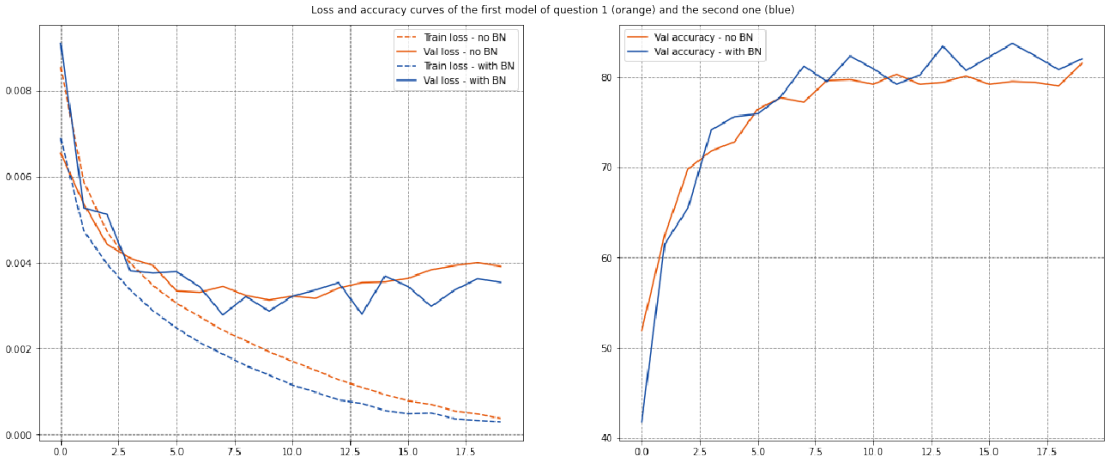


In the first image we cannot distinguish any distinct pattern, as we expected since the weights are initialized randomly. In the second image, altough the filters are 3x3, we are able to recognize low level feautures such as edge and/or color detection.

5 Question 2a: batch normalization

In order to add batch normalization layers in each Convolutional block, it is sufficient to set the "norm_layer" flag to "BN". As we designed the ConvNet class, it will add BN's automatically (see the architecture in the first image in question 1a).

As suggested by the question 2a, we performed the training with the same hyperparameters used in the tasks in question 1. We report below the loss and accuracy curves comparing the two models (with and without batch normalization).



The loss curves are slightly better for the model with batch normalization (even if its validation curve seems a bit more fuzzy, but almost always lower than the one of the model without BN). The same trend is observable in the accuracy plot, where the model with batch normalization exceeds the normal one starting from the 3rd epoch.

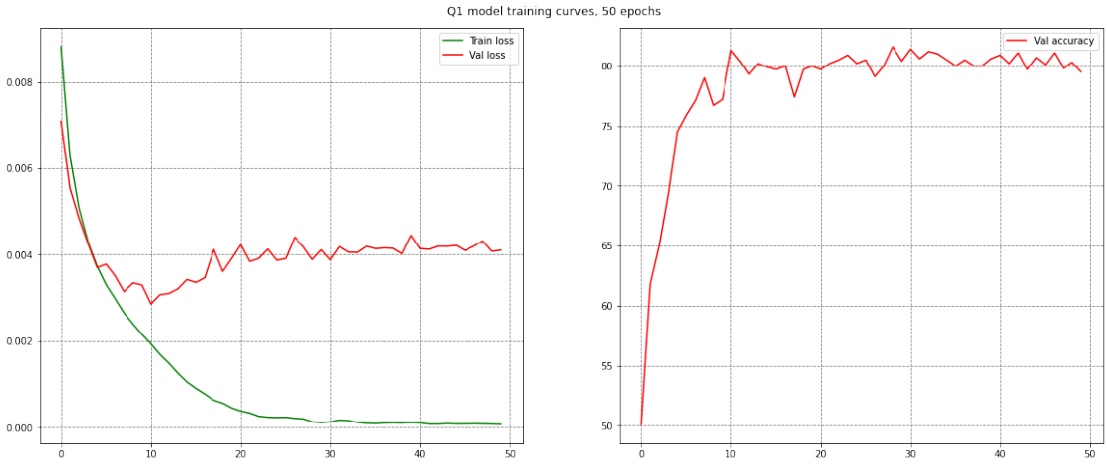
6 Question 2b: Early stopping

As already mentioned, we performed early stopping (to all the models implemented in this homework) in order to save the best model weights (according to the validation set) for the final predictions on the test set.

In question 2b it was asked moreover to increase the training epochs to 50 in both Q1a and Q2a, and compare for each one of them the best model and latest model. Here below we report the results we obtained after the training procedure:

Q1a - No Batch Normalization

We report below the training curves for 50 epochs.



The table below indicates the differences between the latest and the best model:

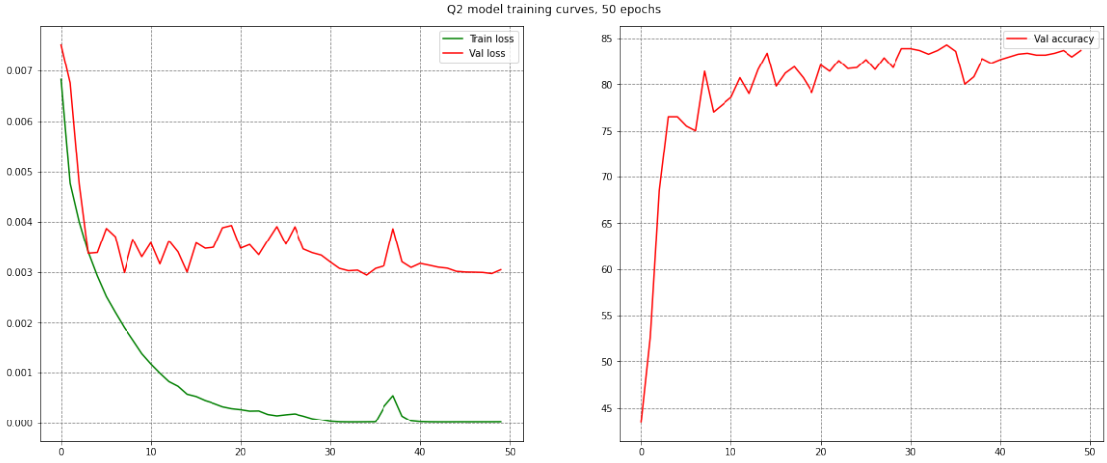
[8]:

	latest_model	best_model
train_loss	7.408e-5	1.33e-4
val_loss	4.11e-3	1.34e-4
val_acc	79.5%	81.6%

At the end, the latest model has 79.4% of accuracy on the **test set** while the best one 79.1%

Q2a - Batch Normalization

We report the training curves for 50 epochs and the table for the latest and best models comparison:



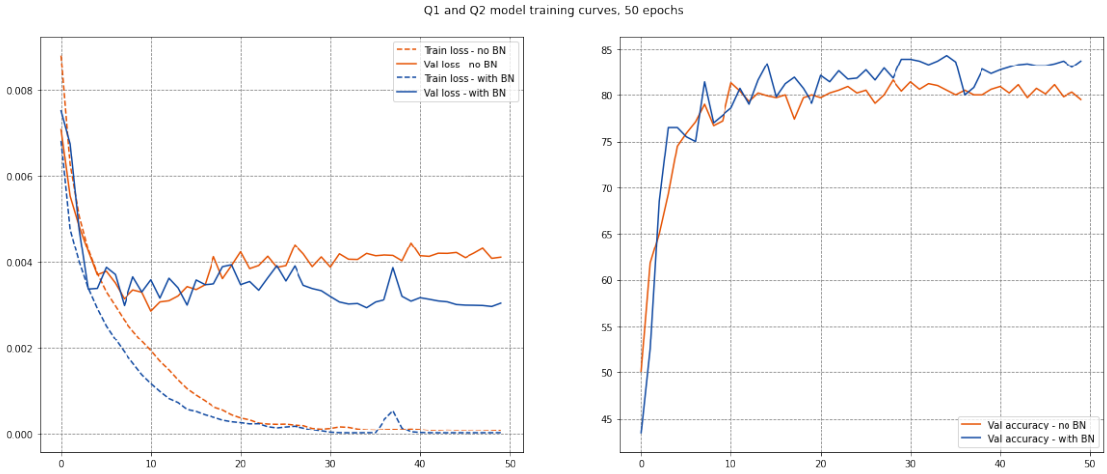
[10]:

	latest_model	best_model
train_loss	1.650e-5	1.6e-5
val_loss	0.003	1.6e-5
val_acc	83.7%	84.3%

At the end, the latest model gives 82.7% of accuracy on the **test set** while the best model reaches 83.7%

Q1a and Q2a - overwiew

As we've done before, we can visualize the training curves together. The result is the following:



To sum up, from all of this results we clearly see that:

- Even if the batch normalization does not helps directly with overfitting, it clearly enhances the model performances, allowing us to increase the model accuracy up to 2-3% (in our case, ~+4% on the final accuracy over the test set).
- Moreover, the early stopping helped our ConvNet model with batch normalization to save the parameters which **generalize well**, therefore giving us better results also on the test set.