Homework 3: Convolutional Neural Network CIFAR10 Classification

Elisa Ferrara

1 Introduction

I solved the exercise first of all with a simple CNN network followed by a MLP net for classification that I created from scratch. The performances were gradually boosted through various methods as explained in 2. Later, after a research on papers I found that for 32x32 images of CIFAR10 some interesting networks are ResNet, DenseNe and VGGNet. I deepened my trials with VGG as in 3. The results are comparable. I choose to present "MiniVGG" net for grading, however in the code you can still run the so called "ConvNet" to extract the weights.

2 CNN created by me

I started with a CNN made of 2 convolutional layer of increasing dimension and a pooling layer to ensure translational invariance. Then classification was done through an MLP made of 2 layers. Accuracy on test set was around 50%. Before working on some tips to increase it, I decided to deepen the network a little bit. So the final network includes:

1. Block: Conv+Pool
2. Block: Conv+Conv+Pool
3. Block: Conv+Conv+Conv
4. MLP: Lin+dropout+Lin

2.1 Notes on the design

- Input number on first layer is 3 because of the 3 channels RGB
- Output of last linear layer is 10 because I have 10 classes
- Rule followed: increase at every convolutional layer the number of filters

2.2 Performance boosting tricks

- **Batch Normalization**. Helped in training in stability and fast convergence. Added 2 percentage points on accuracy.

- **Dropout**. Limits overfit: when validation accuracy grows, also test accuracy grows.

- **Pooling layers manage**. In a first implementation I had 4 pooling layers: this made me reduce model capacities because the final images were 2x2. Performances increased when I switched to just two pooling layers.

- **Optimize scheduler**. I observed that if I trained for 10 epochs with a certain learning rate and then for the following 10 epochs with a reduced one instead of simply 20 epochs with the same lr, performances overall increased of a couple of percentage points. that's why I used a scheduler that decreases of 1/10 the lr if accuracy does not improve in 4 epochs. I also implemented early stopping in case of absence of improvement during 5 epochs.

- **Data Augmentation**. This gives 10 percentage points to accuracy especially on validation set. It is necessary to be careful because too much augmentation causes overfit. I triplicated the dataset adding first an horizontal flip and then a random crop.

- **Optimizer choice**. Tested with SGD and Adam and different learning rates but the first one gave better results.

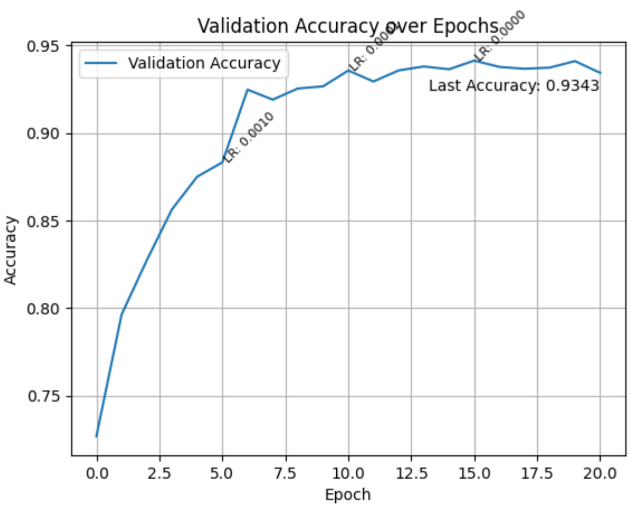

Figure 1: ConvNet training accuracy

Accuracy on validation set: **94%**
Accuracy on test set: **84%**

3 Mini Modified VGG

VGG was created for bigger size images (224x224). The original VGGNet has in fact 5 pooling layers which I cannot of course replicate because my images size is only 32x32. Therefore I modified it making it less deep. The final result can be summarized in the following:

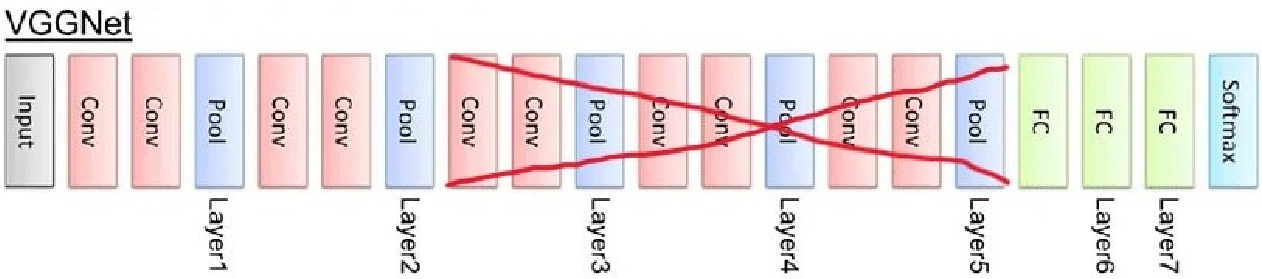

Figure 2: MiniVGG net scheme

I also included Batch Normalization Layers and Dropout both in between convolutional+pooling blocks (with an high rate, 0.50) and in the final MLP. I used augmented dataset as in the first network and I left batch size at 64. I also tried with deeper VGG versions - with another conv-conv-pool block - but the results improved by a few tenths of a percentage point. However, the training time was really higher so I decided it was not worth it. Instead I improved by 5 % in both validation and testing set by deepening the fully connected layer dedicated to classification.

Accuracy on validation set: **85%**
Accuracy on test set: **85%**