Assignment 2

Student: STEFANO ANDREOTTI; Student's email: andres@usi.ch

November 3, 2024

Image classification using CNNs

2 I loaded the dataset using torchvision datasets, so got the data already into PyTorch. I extracted list of the labels and the list of images from the training set. From the labels i was able to get the class names which are: Plane, Car, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck. Then i found one image for each class, reported in figure 1. After this i calculated the class distribution for training and test set which i report in plot number 2. The training set is made out of 50 000 entries that are equally distributed over the 10 classes making each class have 5 000 images. Test set have only 1 000 image for each class and it is identically distributed.

Figure 1: Images for each class

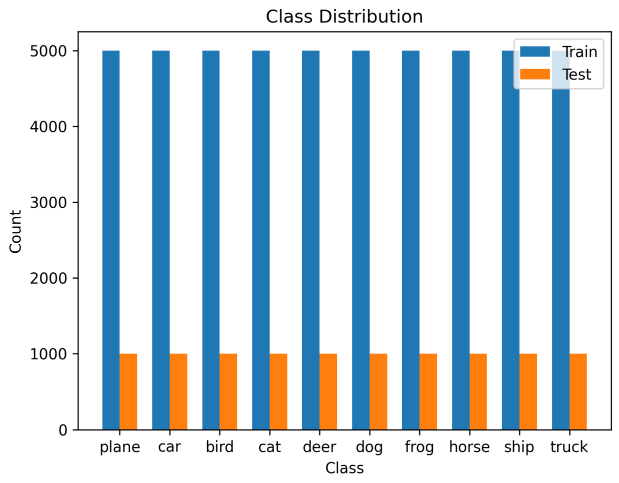

Figure 2: Class distribution

3 Initially the entries are in PIL format which is a type that we cannot use to do this deep learning task. Using a transformer function i have transformed it into tensors. An image translated into tensor has 3 dimensions, the first is the width of the image, the second is the heigth of the image and the third dimension indicates the channels of the images, in our case all the images have 3 channels representing the colors Red, Green and Blue.

4 In this step i have done the normalization of the images to work on images that have mean equal to 0 and standard deviation equal to 1. To normalize the images i used the torchvision function normalize that i have included into the transformer described in the previous point. The transformer applied at the data loading now is casting the images into tensors and normalizing them.

5 I have spitted the test dataset in half using the random split function from torch utils, the result of this operation are 2 datasets of 500 entries each that will be the test dataset and validation dataset.

6 For this model i have tried different architectures, firstly i have tried to double the fully connected layer and adding blocks of convolutional layers with max pooling layer but i noted that after 2 blocks the model will overfit the dataset because the loss function of the validation set will increase while the training validation was decreasing. I kept 2 blocks made of: Conv - Conv - Activ - Pool with 2 fully connected layers. After i found this architecture i have tried to change the pool layer trying to use an average pooling instead of a max pool layer but it has a worst performance than the max pooling. At the end i tried to add another fully connected layer because the out channels of the last layer are 64 multiplied for the height (5) and the width (5) of the current image after the 2 blocks of the network. I thought about trying to decrease the big jump from 1600 features to 128, then a layer lowering the 128 to 64 and the last layer to classify this 64 features into 10 which are out classes. In the end i archived an accuracy of 65.2%. As an example i report 3 that show one trial where there was overfitting, as we can see the validation loss line continue to increase over the epochs.

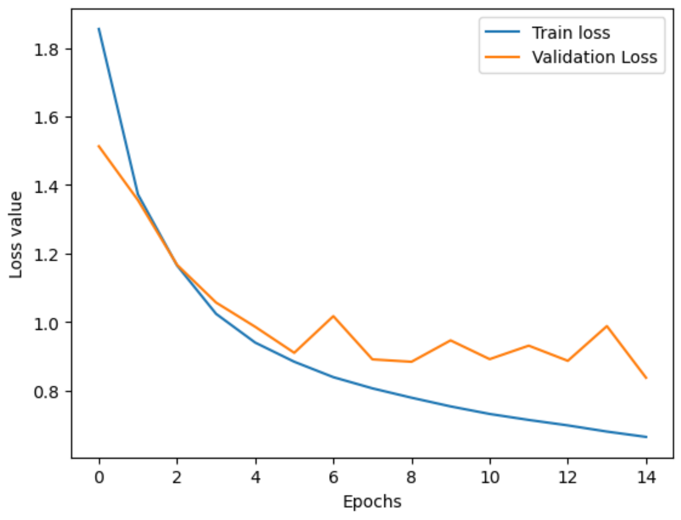

Figure 3: Overfitting trial

7 For monitoring the training phase i decided to print the loss and accuracy values every 1/3 of batch and at the end of each batch. To monitor the test set i decided to print the loss value and accuracy only at the end of each epoch because the test set is made out of only one batch and splitting it into thirds is not useful to look at the tendency of the loss function epoch after epoch. To tune the model i started from the hyper parameters values setting them as noted into the assignment text and keeping fixed batch size and epochs i tried to use both SGD and Adam as optimizers and slowly change the learning rate. I came up with a learning rate of 0.033 that is a good value for a fast convergence but it does not overfit. For the epochs i kept 4 epochs because do training for more epochs end in a overfitting situation where the test loss go always up and the training loss continues to descent. I have tried a smaller structure but it was too slow comparing to the one i have presented in terms of convergence into a good solution with a low loss and high accuracy, beside having a more deep structure will not provide solid improvements and lead to overfit faster than the architecture presented.

8 In plot 4 we can see the evolution of training and validation loss, in particular we can see that both loss is lowering their values epoch after epoch. The last epoch show that we are not in overfitting because both the training loss and the validation loss are decreasing and not diverging.

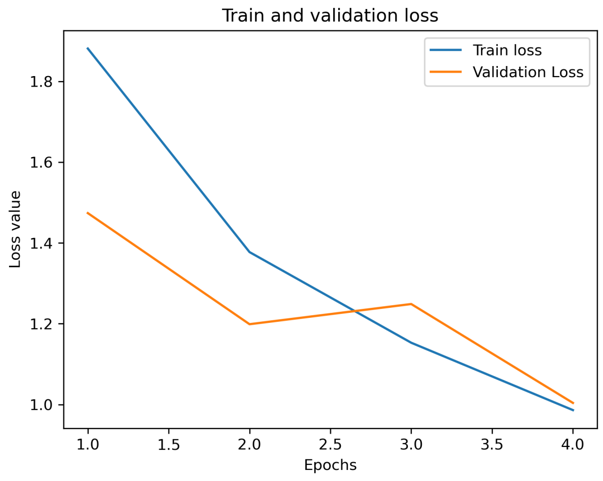

Figure 4: Loss function in training and validation

9 Firstly i changed the architecture by adding batch normalization after each convolutional layer, this helped with accelerating training and it reduces the impact of internal covariate shift. Then i made the CNN out of 3 blocks of Conv $\rightarrow$ Normal $\rightarrow$ Activ $\rightarrow$ Conv $\rightarrow$ Normal $\rightarrow$ Activ $\rightarrow$ Pool, i choosed the GeLU as activation function beacuse it performs weel in image classification tasks. After the 3 blocks described i flatten the data then i used a fully connected layer made out of 3 neural networks with batch normalization and drouput layer to prevent overfitting. I have done other trials like not use the dropout, changing the optimizer but they will not perform as well as this architecture. The last modification that i have done is apply data augmentation to the dataset, in particular i have used random horizontal flip and random rotation. I have done some trials changing also the colors of the data but it wasn't helpful. As hyper parameters i added some epochs archiving 80% of accuracy with just 7 epochs but adding more epochs lead to the best results of 83,74%. As optimizer i used Adam with a learning rate of 0.0283. In figure 5 i show the training and validation loss, showing that no overfitting is detected.

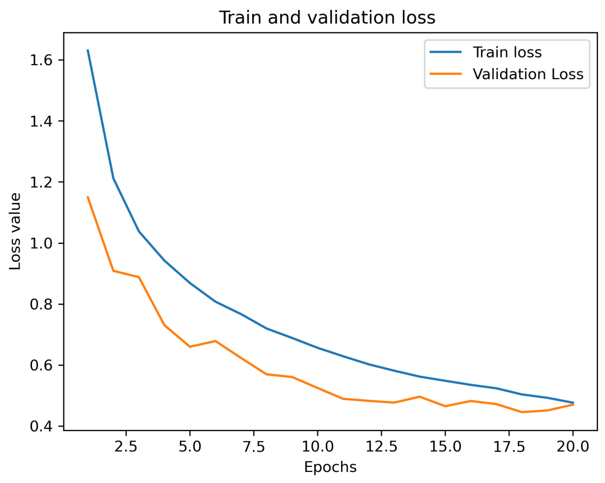

Figure 5: Loss function in training and validation

10 I have done 5 different runs with the model of point 6 and 4 epochs. I printed the accuracy of validation set for each model showing that using a different seed for the training will heavily impact the accuracy. In my case accuracy that i got was: 65.64% using seed equal to 5, 62,98% for seed 6, 62,8% for seed 7, 63,74% for seed 8 58,18% and for seed 9 . I can conclude that setting the random seed fixed is very important to allow the reproducibility of data because otherwise the results are pretty much different training after training.