Deep Learning: Assignment 2

Jury Andrea D'Onofrio

October 2022

1 Image Classification Using ConvNets

1.1 Dataset

1.1.1

I load the dataset via `torchvision.datasets.CIFAR10`. The number of images is:

- 50000 for the training set;
- 10000 for the test set.

Figure 1 shows some training images. I use the transpose function on each image in order to have the color channel in the last dimension.

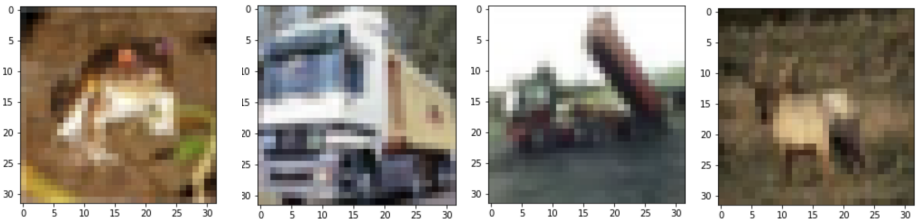


Figure 1: Training samples

1.1.2

I compose multiple transformation using `torchvision.transforms.Compose` in which I standardize the data using `Normalize`. I apply the function to the argument transform.

1.1.3

To verify the values of the means and the standard deviation, I decide to import a new `CIFAR10` dataset with no standardized data. I compute the mean and the std using all 50000 training images. The results are:

- mean for (r, g, b): (0.4914, 0.4822, 0.4465);
- std for (r, g, b): (0.247, 0.243, 0.262).

They are very close to the given values.

1.1.4

Using `SubsetRandomSampler`. I split the train set into:

- A validation set composed by the last 1000 images;
- A training set composed by the first 49000 images.

I create the training and validation dataloaders using the train set:

- The training dataloader uses the training sampler;
- The validation dataloader uses the validation sampler;

1.2

I implement the convolutional neural network as asked, please see the source code.
The summary of the my neural network is shown in Figure 2. I use the default values for the hyper-parameters, where not specified.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 30, 30]             896
            Conv2d-2          [-1, 32, 28, 28]           9,248
         MaxPool2d-3          [-1, 32, 14, 14]               0
            Conv2d-4          [-1, 64, 12, 12]          18,496
            Conv2d-5          [-1, 64, 10, 10]          36,928
         MaxPool2d-6            [-1, 64, 5, 5]               0
            Linear-7                 [-1, 512]         819,712
            Linear-8                  [-1, 10]           5,130
================================================================
Total params: 890,410
Trainable params: 890,410
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.59
Params size (MB): 3.40
Estimated Total Size (MB): 4.00
----------------------------------------------------------------
None
```

Figure 2: Model

1.3

1.3.1

I implement the training pipeline. I choose to monitor the current training loss and accuracy every 200 steps and I monitor the validation accuracy after each epoch. I keep track of the best best validation accuracy and the corresponding epoch. I obtain 74.8% as best validation accuracy in epoch 18.

1.3.2

I use the given hyper-parameters to train the model.

1.3.3

I train the model with the given hyper-parameters and the final validation accuracy is 73.8%.

1.3.4

The model overfits. As Figure 3 and Figure 4 show, the validation loss, from epoch 10, crosses the training loss and it seems to increase. The validation accuracy plateaus at the same epoch.
The training loss and the training accuracy have a good progress. In fact, while the training loss decreases the training accuracy increases.

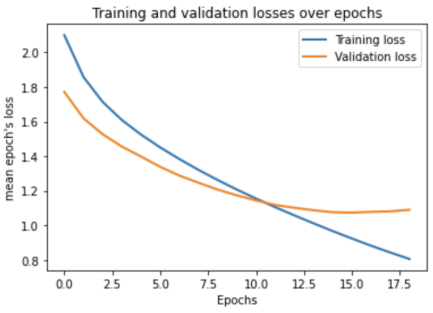


Figure 3: Training and Validation loss

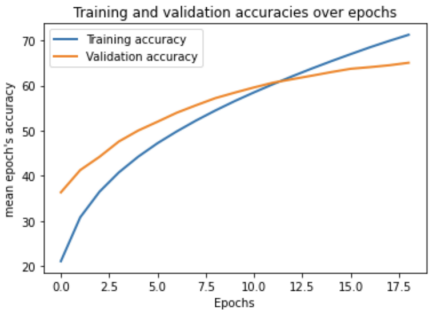


Figure 4: Training and Validation accuracy

1.3.5

The parameter p is the probability of the elements of the input tensor to be selected and set to 0. Therefore, the activation function doesn't receive the contribution of those elements.
I train the model for 100.

- $p = 0.2 \rightarrow$ As Figure 5 and Figure 6 show, the model overfits. The validation loss crosses the training loss. I achieve the best validation accuracy equal to 80.1% in epoch 89.

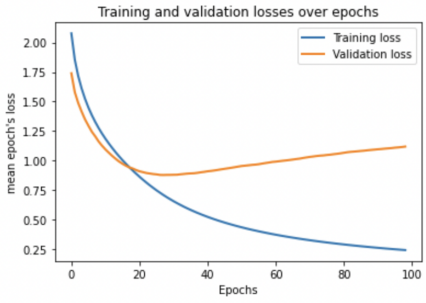


Figure 5: Training and Validation loss $p = 0.2$

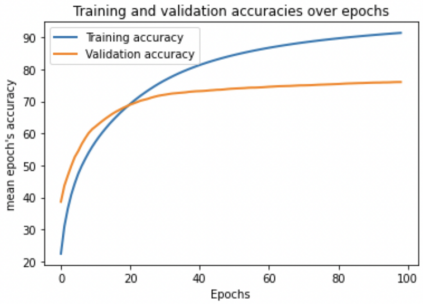


Figure 6: Training and Validation accuracy $p = 0.2$

- $p = 0.5 \rightarrow$ As Figure 7 and Figure 8 shows, the model overfits but it is better than the previous one. I achieve the best validation accuracy equal to 83.0% in epoch 77.

Figure 7: Training and Validation loss $p = 0.5$

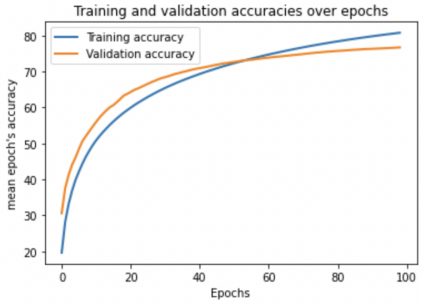


Figure 8: Training and Validation accuracy $p = 0.5$

- $p = 0.8 \rightarrow$ As Figure 9 and Figure 10 show, the model works. I achieve the best validation accuracy equal to 76.2% in epoch 91.

My best model has $p = 0.8$ and best validation accuracy 76.2%.

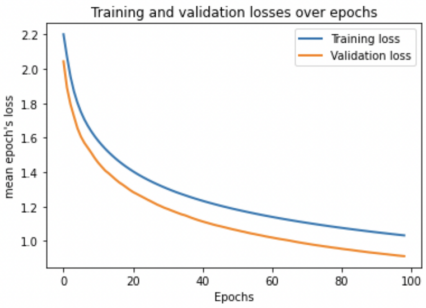


Figure 9: Training and Validation loss $p = 0.8$

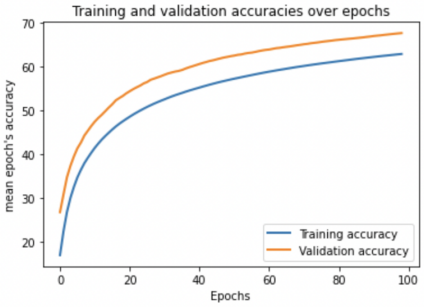


Figure 10: Training and Validation accuracy $p = 0.8$

1.3.6

With my best model gotten above, I achieve a test accuracy equal to 73.67%. I visualize 7 images.

1. Figure 13
2. Figure 14
3. Figure 15
4. Figure 16
5. Figure 17
6. Figure 18
7. Figure 19

The predictions are mostly correct.

1.3.7

To achieve a test accuracy above 80%:

- I set the number of epochs to 100;
- I increase the learning rate to 0.002;
- I set the dropout with $p = 0.5$;
- I add a dropout layer after the first fully-connected layer.

In this way, I get a test accuracy equal to 81.79%. Figure 11 and Figure 12 show the evolution of the loss and accuracy for training and validation set.

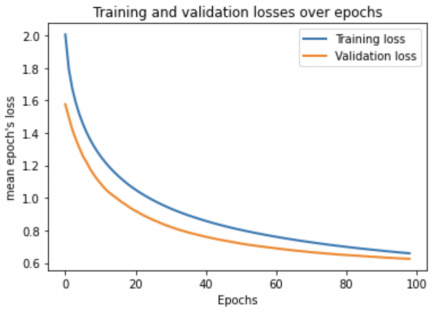


Figure 11: Training and Validation loss

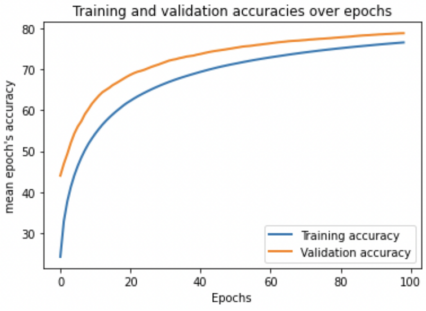


Figure 12: Training and Validation accuracy

1.4

1.4.1

I use the softmax activation function in order to transform the output of the last layer in a probability distribution.

1.4.2

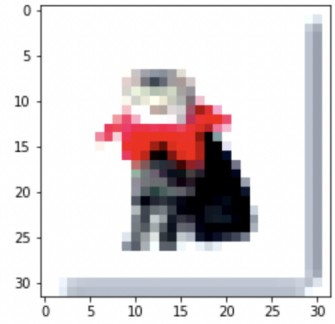
The name of this quantity is velocity.

1.4.3

I can use 1D convolution for texts.

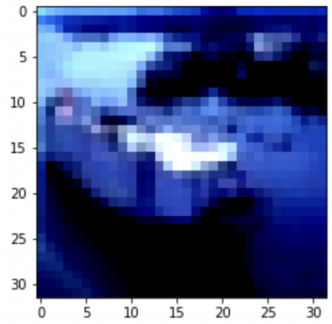
1.4.4

- At training time, the dropout randomly selects some of the elements of the input tensor with probability p;
- At test time, the dropout is deactivated thus all elements are passed.



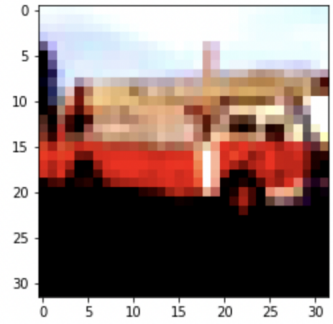
```
output probability distribution:
tensor([0.0431, 0.0408, 0.0141, 0.0863, 0.0098, 0.0242, 0.0241, 0.0055, 0.4751,
        0.2770])
max probability distribution is in position: 8, value = 0.475
prediction = ship, label = cat
```

Figure 13: prediction: Ship, label: Cat



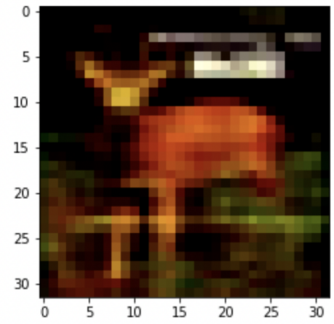
```
output probability distribution:
tensor([3.9229e-02, 2.1991e-01, 8.5837e-04, 1.5266e-03, 9.2417e-04, 3.8212e-04,
        1.2918e-03, 2.7459e-04, 7.1604e-01, 1.9556e-02])
max probability distribution is in position: 8, value = 0.716
prediction = ship, label = ship
```

Figure 14: prediction: Ship, label: Ship



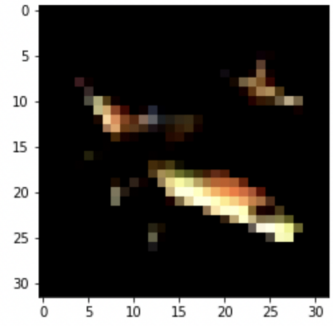
```
output probability distribution:
tensor([5.0243e-02, 8.7488e-02, 9.1917e-04, 9.2521e-04, 9.1193e-04, 1.0529e-04,
        2.2457e-04, 1.1316e-03, 3.9718e-02, 8.1833e-01])
max probability distribution is in position: 9, value = 0.818
prediction = truck, label = truck
```

Figure 15: prediction: Truck, label: Truck



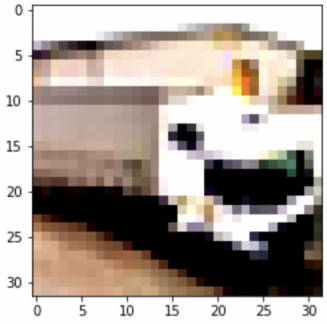
```
output probability distribution:
tensor([0.0020, 0.0027, 0.0706, 0.0273, 0.7295, 0.0813, 0.0080, 0.0754, 0.0019,
        0.0015])
max probability distribution is in position: 4, value = 0.730
prediction = deer, label = deer
```

Figure 16: prediction: Deer, label: Deer



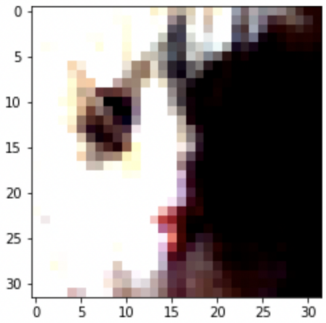
```
output probability distribution:
tensor([0.1347, 0.4429, 0.0236, 0.1340, 0.0227, 0.0644, 0.1587, 0.0061, 0.0043,
        0.0087])
max probability distribution is in position: 1, value = 0.443
prediction = car, label = plane
```

Figure 17: prediction: Car, label: Plane



```
output probability distribution:
tensor([4.2918e-02, 2.2807e-02, 2.7609e-03, 8.8067e-03, 2.2556e-03, 1.7654e-03,
        5.2886e-04, 1.8597e-03, 3.6012e-02, 8.8029e-01])
max probability distribution is in position: 9, value = 0.880
prediction = truck, label = truck
```

Figure 18: prediction: Truck, label: Truck




```
output probability distribution:
tensor([3.0655e-05, 8.2275e-07, 2.7754e-03, 8.1182e-01, 4.7889e-02, 1.2969e-01,
        1.3825e-03, 6.4054e-03, 4.6274e-06, 2.4829e-07])
max probability distribution is in position: 3, value = 0.812
prediction = cat, label = cat
```

Figure 19: prediction: Cat, label: Cat