Deep Learning Lab
Terekhin Daniil
Assignment 2

1.1 Dataset

1. In the train dataset I took 50000 images, in the test 10000 (figure 1). You also can see the 5 random images from training dataset (figure 2)

```
train_dataset
```

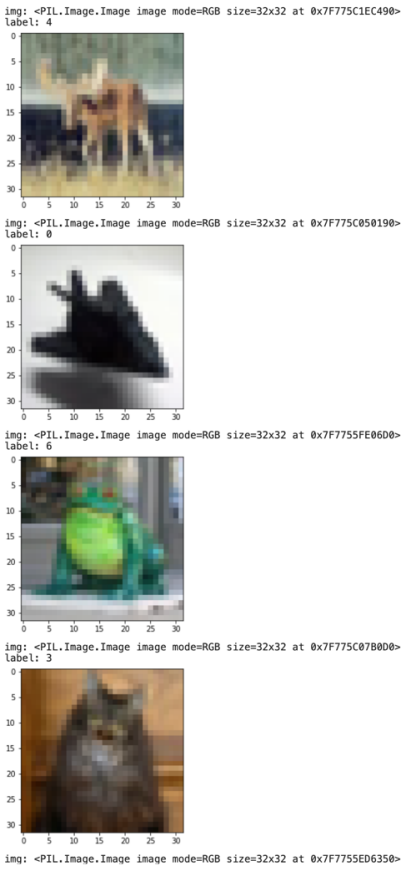
```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ../dataSet_CIFAR10
    Split: Train
```

```
test_dataset
```

```
Dataset CIFAR10
    Number of datapoints: 10000
    Root location: ../dataSet_CIFAR10
    Split: Test
```

+ Code + Markdown

(Figure 1)



(Figure 2)

2. I normalized values for each channel. Used Compose and Normalize for it (Figure 3).

```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ../dataSet_CIFAR10
    Split: Train
    StandardTransform
Transform: Compose(
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=(0, 0, 0), std=(1, 1, 1))
           )
```

(Figure 3)

3. Mean/std statistics across channels above 50000 training dataset (Figure 4).

```
Mean across the First channel: 0.4913996756076813
Mean across the Second channel: 0.48215845227241516
Mean across the Third channel: 0.44653093814849854
-----------------------
Std across the First channel: 0.20230092108249664
Std across the Second channel: 0.19941283762454987
Std across the Third channel: 0.20096160471439362
```

(Figure 4)

4. Created Validation dataset by splitting training dataset to training and validation (49000 and 1000 images) (Figure 5).

```
print(len(val_indices))
print(len(train_indices))
```

```
1000
49000
```

(Figure 5)

1.2 Model

1. Created an architecture of convolutional neural network using rectified linear activation functions (ReLUs) (Figure 6)

```python
class Conv_network(nn.Module):
    def __init__(self):
        super(Conv_network, self).__init__()
        self.conv_1 = nn.Conv2d(3, 32, kernel_size=3) # Convolutional layer 1: 32 filters, 3 × 3

        self.conv_2 = nn.Conv2d(32, 32, kernel_size=3) # Convolutional layer 2: 32 filters, 3 × 3

        self.pool_1 = nn.MaxPool2d(2, 2) # Max-pooling layer 1: 2 × 2 windows
        self.conv_3 = nn.Conv2d(32, 64, kernel_size=3) # Convolutional layer 3: 64 filters, 3 × 3

        self.conv_4 = nn.Conv2d(64, 64, kernel_size=3) # Convolutional layer 4: 64 filters, 3 × 3
        self.pool_2 = nn.MaxPool2d(2, 2) # Max-pooling layer 2: 2 × 2 windows

        self.fc1 = nn.Linear(64*5*5, 512) # Fully connected layer 1: 512 units
        self.fc2 = nn.Linear(512, 10) # Output layer with num_classes


    def forward(self, x):

        x = F.relu(self.conv_1(x))      # conv_1

        x = self.pool_1(F.relu(self.conv_2(x))) # conv_2 + pool_1

        x = F.relu(self.conv_3(x)) # conv_3

        x = self.pool_2(F.relu(self.conv_4(x))) # conv_4 + pool_2

        x = nn.Flatten()(x)

        x = F.relu(self.fc1(x)) # fully connected layer
        x = self.fc2(x) # fully connected layer + output layer

        return x
```

(Figure 6)

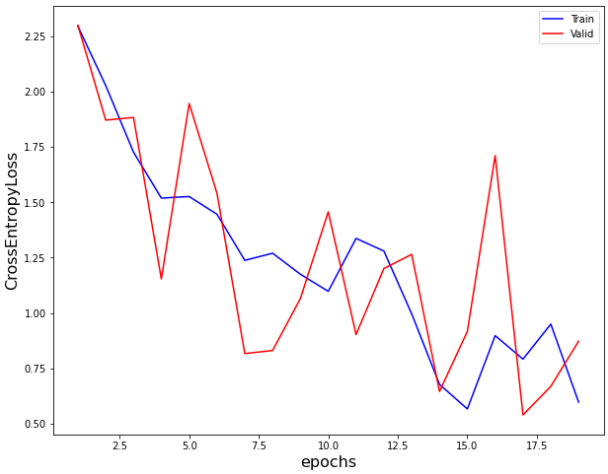
1.3 Training

1. I Implemented the training pipeline and printed validation accuracy after each epoch.
2. Used the following hyperparameters to train my model.
3. Trained my model and got 68.9 % accuracy (Figure 7).

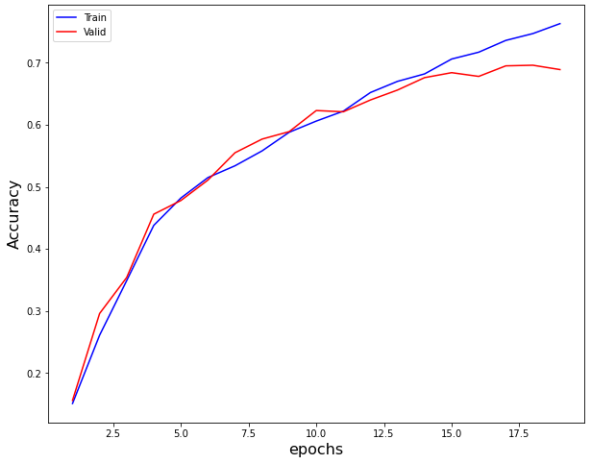
```
Validation accuracy:  67.8 %
epoch: 17, steps: 0, train_loss: 0.77, running_acc: 71.9 %
epoch: 17, steps: 500, train_loss: 0.833, running_acc: 74.2 %
epoch: 17, steps: 1000, train_loss: 0.529, running_acc: 74.1 %
epoch: 17, steps: 1500, train_loss: 0.792, running_acc: 73.6 %
Validation accuracy:  69.5 %
epoch: 18, steps: 0, train_loss: 0.583, running_acc: 81.2 %
epoch: 18, steps: 500, train_loss: 1.12, running_acc: 75.5 %
epoch: 18, steps: 1000, train_loss: 0.639, running_acc: 74.8 %
epoch: 18, steps: 1500, train_loss: 0.95, running_acc: 74.7 %
Validation accuracy:  69.6 %
epoch: 19, steps: 0, train_loss: 0.816, running_acc: 78.1 %
epoch: 19, steps: 500, train_loss: 0.714, running_acc: 76.4 %
epoch: 19, steps: 1000, train_loss: 0.865, running_acc: 76.3 %
epoch: 19, steps: 1500, train_loss: 0.598, running_acc: 76.3 %
Validation accuracy:  68.9 %
Finished Training
```

(Figure 7)

4. Ploted the evolution of the loss (Figure 8) and accuracy (Figure 9) for your training and validation set. In the loss graph we can see that the value of cross entropy loss for validation data fluctuate a lot compared to train data. In the accuracy graph we can see that accuracy for training dataset bigger than for validation dataset during last couple epochs. It means that there is overfitting.



(Figure 8)



(Figure 9)

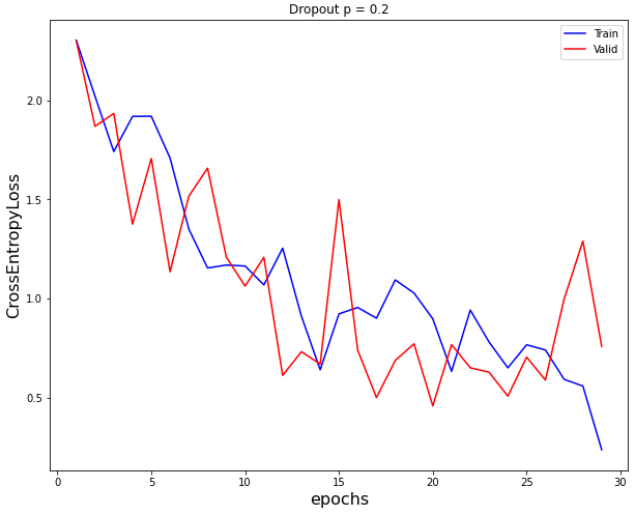
5. I used dropout for regularization. The argument “p” in torch.nn.Dropout means probability of deleting each neurons in the hidden layer.

   Model with dropout (p = 0.2) had Validation accuracy: 76.0 %. Cross entropy loss (Figure 10), accuracy (Figure 11). We can see that overfitting of the model reduced, accuracy increased, cross entropy values fluctuate less.

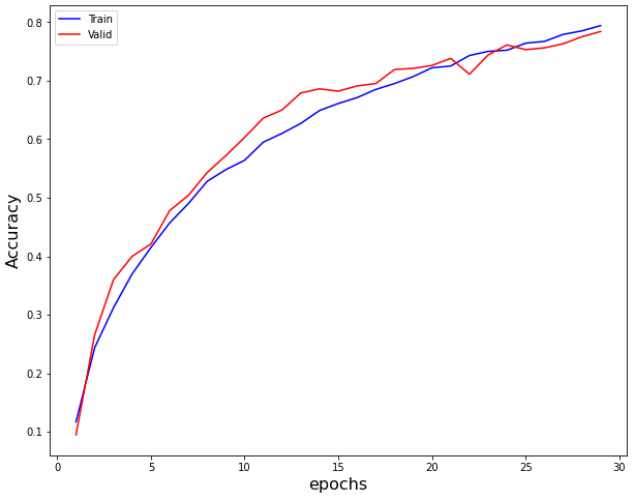
   Model with dropout (p = 0.5) had Validation accuracy: 74.2 %. Cross entropy loss (Figure 12), accuracy (Figure 13). Cross entropy loss had bigger spreading compared to model without Dropout. Even though validation accuracy became bigger than train accuracy, the final validation accuracy was less than previous model had.

   Model with dropout (p = 0.8) had Validation accuracy: 62.6 %. Cross entropy loss (Figure 14), accuracy (Figure 15) (these pictures have title “Dropout p = 0.5”, it is

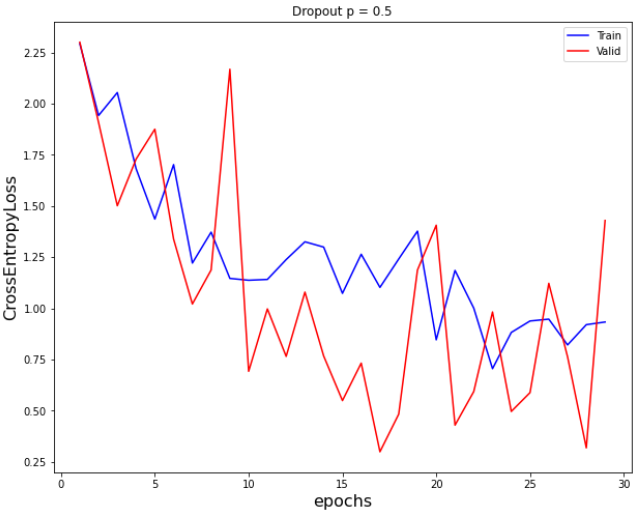
a mistake, real values of p is 0.8). Cross entropy loss had practically the same spreading as model without Dropout, but final accuracy less.



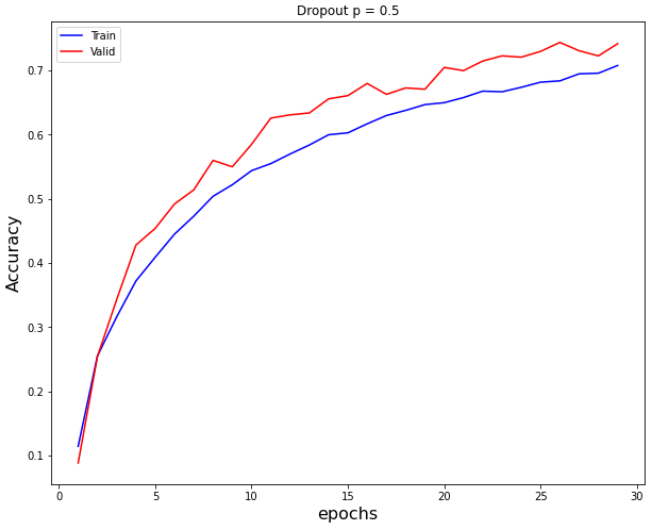
(Figure 10)



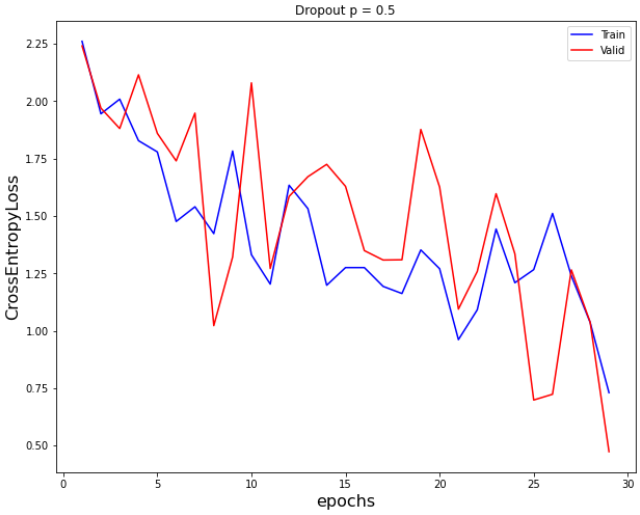
(Figure 11)



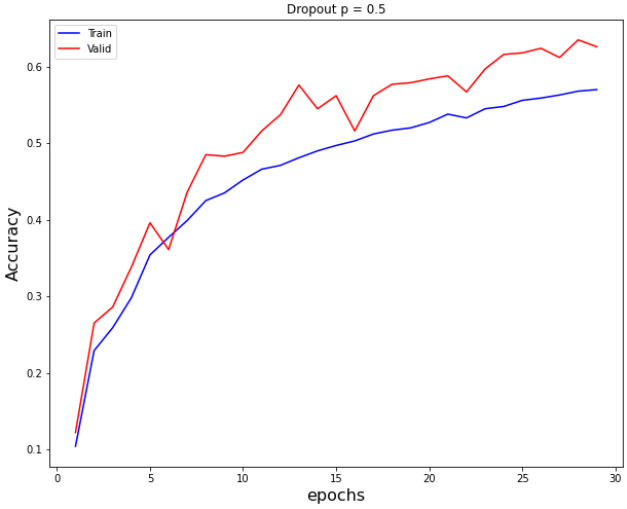
(Figure 12)



(Figure 13)



(Figure 14)



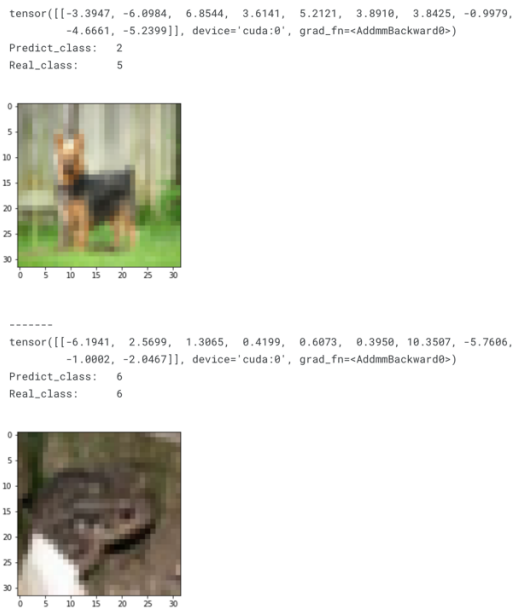
(Figure 15)

6. Test accuracy of final model with dropoup (p = 0.2) was 74.8 %. Randomly selected pictures from test dataset (Figure 16, 17, 18) with output probability distributions of belonging to one or another class.

   In Figure 16 we can see two pictures. In the first one there is class number 5 (dog), model predicted here class number 2 (bird). That's wrong. In the second picture there is class 6 (frog), model also predicted it like class 6.

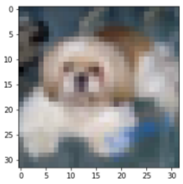
   In Figure 17 there are also two pictures. In the first one there is class 5 (dog), model predicted it correct. In the second one there is class 9 (truck), model also predicted it correct.

   In the Figure 18 there is only one picture. Real class is 7 (horse), model predicted it like class 7.

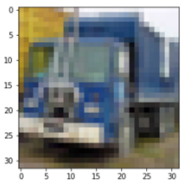


(Figure 16)

```
tensor([[-4.4554, -4.3399, -1.4367,  6.3436,  2.0924,  8.4022,  1.1403, -0.4300,
         -0.4086, -5.9247]], device='cuda:0', grad_fn=<AddmmBackward0>)
Predict_class:   5
Real_class:      5
```

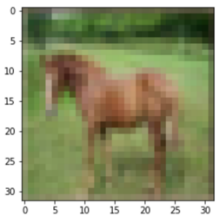


```
-------
tensor([[-0.4960,  4.5736, -1.4612, -1.7229, -3.7316, -4.1865, -2.9213, -5.4509,
          2.6297, 11.3473]], device='cuda:0', grad_fn=<AddmmBackward0>)
Predict_class:   9
Real_class:      9
```



(Figure 17)

```
tensor([[-4.6549, -6.6194,  0.5849, -0.5010,  9.2416,  5.4835, -1.2387, 10.8417,
         -5.7325, -5.8915]], device='cuda:0', grad_fn=<AddmmBackward0>)
Predict_class:   7
Real_class:      7
```



(Figure 18)

7. After several attempts I got Test accuracy: 79.3 % (Figure 19).

```
Test accuracy:  79.3 %
```

(Figure 19)

In the first try I decreased batch size from 32 to 22 and increase epochs from 30 to 35. Test accuracy: 75.6 %.

In the second try I changed optimizer from SGD to Adam, decrease batch size from 22 to 16 and increased epochs from 35 to 40. Test accuracy: 76.7 %

In the third try I added one feed forward layer and change input and output parameters for convolutional layers (Figure 20), decreased batch size from 16 to 10, increased epochs from 40 to 50.

```python
class Conv_network(nn.Module):
    def __init__(self):
        super(Conv_network, self).__init__()
        self.conv_1 = nn.Conv2d(3, 32, kernel_size=3) # Convolutional layer 1: 32 filters, 3 × 3

        self.conv_2 = nn.Conv2d(32, 64, kernel_size=3) # Convolutional layer 2: 32 filters, 3 × 3

        self.pool_1 = nn.MaxPool2d(2, 2) # Max-pooling layer 1: 2 × 2 windows
        self.conv_3 = nn.Conv2d(64, 128, kernel_size=3) # Convolutional layer 3: 64 filters, 3 × 3

        self.conv_4 = nn.Conv2d(128, 256, kernel_size=3) # Convolutional layer 4: 64 filters, 3 × 3

        self.pool_2 = nn.MaxPool2d(2, 2) # Max-pooling layer 2: 2 × 2 windows

        self.fc1 = nn.Linear(256*5*5, 1024) # Fully connected layer 1: 512 units
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 10) # Output layer with num_classes

        self.dropout = nn.Dropout(0.2)



    def forward(self, x):

        x = F.relu(self.conv_1(x))      # conv_1
        x = self.pool_1(F.relu(self.conv_2(x))) # conv_2 + pool_1
        x = self.dropout(x)

        x = F.relu(self.conv_3(x)) # conv_3

        x = self.pool_2(F.relu(self.conv_4(x))) # conv_4 + pool_2
        x = self.dropout(x)

        x = nn.Flatten()(x)

        x = F.relu(self.fc1(x)) # fully connected layer
        x = F.relu(self.fc2(x))
        x = self.fc3(x) # fully connected layer + output layer
        return x
```

(Figure 20)

1.4 Questions:

1. We use softmax activation function for multiclass classification in the output layer for getting probability to belonging to each class.
2. Momentum = 0.9
3. 1D convolution can be used for time series data
4. During training dropout use for deleting neurons (probability for each neuron equals parameter p). During the test, you must also leave a dropout due to the fact that the model is already used to get values not from each neurol, but from 80% of all neurons on this layer and without dropout activation function would receive 100% of neurons, training and testing should match to teach other.