

## **Отчёт по лабораторной работе 10**

Компьютерная графика

Тема:

“НС на Pytorch”

**Работу выполнил:** Кутдусов Р. К.

**Группа:** А-13а-19

**Вариант:** 10

**Лектор дисциплины:** Бартедьев О. В.

## 1. Постановка задачи.

Пополнить MNIST 11-м классом "не цифры", поместив в него случайно выбранные буквы EMNIST. 11-й класс формируется из представителей всех классов EMNIST.

Используя **PyTorch**, создать и обучить модель нейронной сети для классификации примеров сформированного набора данных.

## 2. Формирование 11-ого класса MNIST.

- a) Для обучающей выборки сначала формируем список из 26 элементов. Элементы списка представляют собой массивы из индексов – массив из индексов для каждой буквы алфавита отдельно.

Формирование:

```
ind = []
for i in range(26):
    ind.append(np.where(emnist_ltrn == i)[0])
```

- b) Затем для каждого элемента списка массивов индексов оставляем только по 231 примеру для первых 20-ти классов и по 230 примеров для остальных 6-ти классов.
- c) Затем формируем ещё один список, содержащий непосредственно изображения для каждого класса отдельно.

Формирование:

```
emnist_train = []
for i in range(26):
    mnist_train.append(emnist_trn[ind[i]])
```

- d) Выводим изображения первого класса – “А, а”. Для проверки.



- e) Переходим к созданию непосредственно обучающей выборки: изображения и их метки.

Копируем в новый объект изображения из MNIST:

```
x_train = np.copy(mnist_trn)
```

Добавляем в `x_train` элементы из уже сформированного 11-ого класса:

```
for i in range(26):
    x_train = np.append(x_train, mnist_train[i], axis = 0)
[in]: x_train.shape
[out]: (66000, 1, 28, 28)
```

Копируем метки изображений MNIST:

```
y_train = np.copy(mnist_ltrn)
```

Добавляем в `y_train` 6000 меток 10:

```
y_train = np.append(y_train, [10] * 6000, axis = 0)
[in]: y_train.shape
[out]: (66000,)
```

- f) Изображения и метки нужно перемешать. Создаём массив из перемешанных индексов:

```
ind = np.random.permutation(x_train.shape[0])
```

```
x_train = x_train[ind]
y_train = y_train[ind]
```

g) Выводим первые 15 изображений обучающей выборки и их метки:



h) Аналогично действия для проверочной выборки. В проверочную порцию данных 11-го класса добавляются по 38 примеров из первых 20-и классов EMNIST и по 40 из последующих ( $38 * 20 + 40 * 6 = 1000$ ).

### 3. Подготовка данных, построение модели и обучение.

Разделим обучающую выборку на пакеты:

```
trn_data = [[x, int(y)] for x, y in zip(x_train, y_train)]
trn_loader = DataLoader(trn_data, batch_size = batch_size, shuffle = True)
```

Подготовим проверочные данные:

```
tst_data = torch.tensor(x_test)
tst_target = torch.tensor(np.array(y_test, dtype = np.int64))
```

Построим модель и распечатаем её:

```
Net(
  (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (pool2d1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (pool2d2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (pool2d3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (b_norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (b_norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (b_norm3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc1): Linear(in_features=256, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=11, bias=True)
)
```

Torchsummary:

| Layer (type)                            | Output Shape       | Param # |
|---|--------------------|---------|
| Conv2d-1                                | [256, 64, 26, 26]  | 640     |
| Conv2d-2                                | [256, 64, 24, 24]  | 36,928  |
| MaxPool2d-3                             | [256, 64, 12, 12]  | 0       |
| BatchNorm2d-4                           | [256, 64, 12, 12]  | 128     |
| Conv2d-5                                | [256, 128, 10, 10] | 73,856  |
| Conv2d-6                                | [256, 128, 8, 8]   | 147,584 |
| MaxPool2d-7                             | [256, 128, 4, 4]   | 0       |
| BatchNorm2d-8                           | [256, 128, 4, 4]   | 256     |
| Conv2d-9                                | [256, 256, 2, 2]   | 295,168 |
| MaxPool2d-10                            | [256, 256, 1, 1]   | 0       |
| BatchNorm1d-11                          | [256, 256]         | 512     |
| Linear-12                               | [256, 512]         | 131,584 |
| Linear-13                               | [256, 11]          | 5,643   |
| =====                                   |                    |         |
| Total params: 692,299                   |                    |         |
| Trainable params: 692,299               |                    |         |
| Non-trainable params: 0                 |                    |         |
| =====                                   |                    |         |
| Input size (MB): 0.77                   |                    |         |
| Forward/backward pass size (MB): 245.52 |                    |         |
| Params size (MB): 2.64                  |                    |         |
| Estimated Total Size (MB): 248.93       |                    |         |
| =====                                   |                    |         |

## Обучение с выводом точности:

Число эпох 20

Обучение

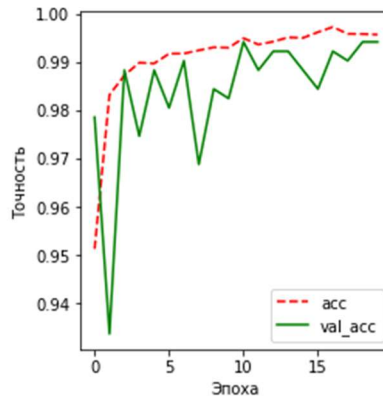
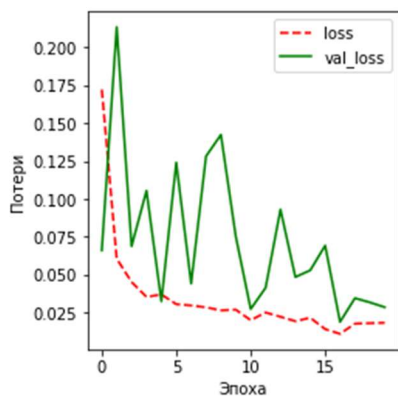
```
Эпоха: 1 loss: 0.172161 acc: 0.9512 val_loss: 0.066092 val_acc: 0.9785
Эпоха: 2 loss: 0.060592 acc: 0.9832 val_loss: 0.213382 val_acc: 0.9336
Эпоха: 3 loss: 0.045192 acc: 0.9872 val_loss: 0.068769 val_acc: 0.9883
Эпоха: 4 loss: 0.035363 acc: 0.9898 val_loss: 0.105414 val_acc: 0.9746
Эпоха: 5 loss: 0.036963 acc: 0.9897 val_loss: 0.032413 val_acc: 0.9883
Эпоха: 6 loss: 0.030561 acc: 0.9917 val_loss: 0.124034 val_acc: 0.9805
Эпоха: 7 loss: 0.029716 acc: 0.9917 val_loss: 0.044146 val_acc: 0.9902
Эпоха: 8 loss: 0.028462 acc: 0.9924 val_loss: 0.128120 val_acc: 0.9688
Эпоха: 9 loss: 0.026496 acc: 0.9930 val_loss: 0.142341 val_acc: 0.9844
Эпоха: 10 loss: 0.026949 acc: 0.9929 val_loss: 0.075120 val_acc: 0.9824
Эпоха: 11 loss: 0.019992 acc: 0.9949 val_loss: 0.027345 val_acc: 0.9941
Эпоха: 12 loss: 0.025070 acc: 0.9936 val_loss: 0.041374 val_acc: 0.9883
Эпоха: 13 loss: 0.022252 acc: 0.9942 val_loss: 0.093022 val_acc: 0.9922
Эпоха: 14 loss: 0.019349 acc: 0.9951 val_loss: 0.048472 val_acc: 0.9922
Эпоха: 15 loss: 0.021584 acc: 0.9950 val_loss: 0.052928 val_acc: 0.9883
Эпоха: 16 loss: 0.014129 acc: 0.9962 val_loss: 0.069231 val_acc: 0.9844
Эпоха: 17 loss: 0.010998 acc: 0.9972 val_loss: 0.018858 val_acc: 0.9922
Эпоха: 18 loss: 0.017726 acc: 0.9958 val_loss: 0.034530 val_acc: 0.9902
Эпоха: 19 loss: 0.018033 acc: 0.9958 val_loss: 0.031759 val_acc: 0.9941
Эпоха: 20 loss: 0.018310 acc: 0.9957 val_loss: 0.028579 val_acc: 0.9941
```

## 4. Сохранение модели в файле.

```
if save_model:
    print('Сохранение весов модели')
    torch.save(model.state_dict(), fn_w)
```

## 5. Графики обучения.

Потери и точность



## 6. Загрузка модели из файла, точность проверочного множества по классам.

```
if load_model:
    print('Загрузка весов из файла', fn_w)
    model.load_state_dict(torch.load(fn_w, map_location = torch.device(device)))
```

Потери: 0.058914

Точность: 0.9917

Точность по классам:

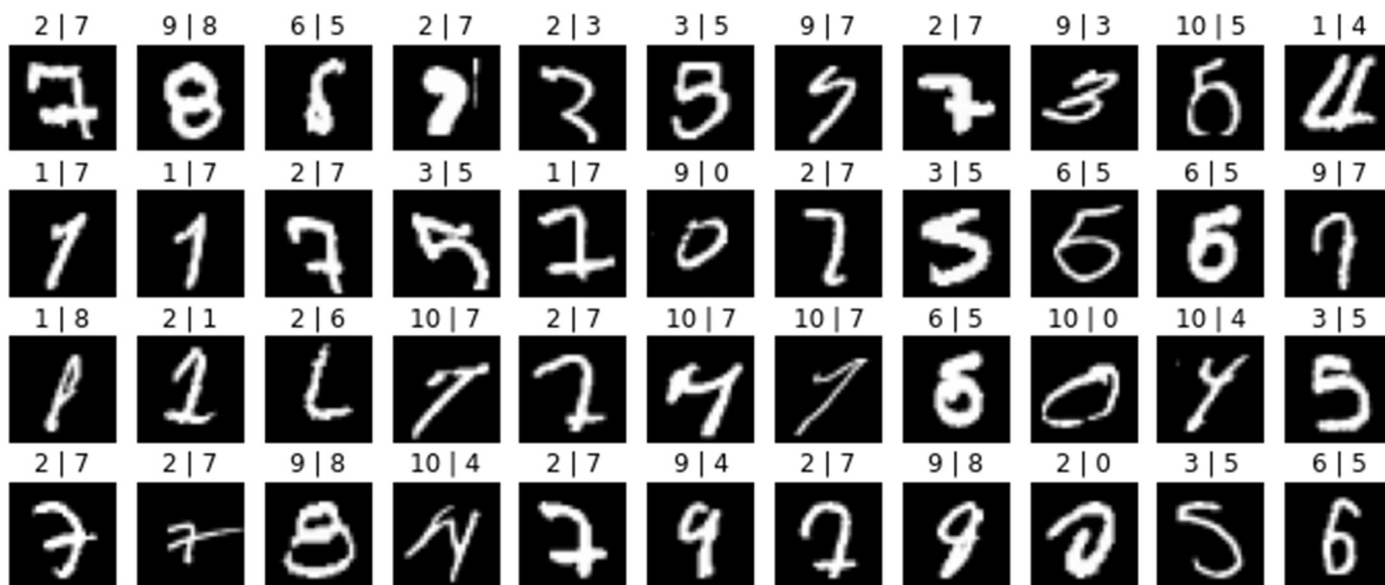
```
0 - 99.49
1 - 99.47
2 - 99.90
3 - 99.80
4 - 99.59
5 - 98.09
6 - 99.27
7 - 98.05
8 - 99.49
9 - 98.61
```

## 7. Вывод оценки качества модели посредством `classification_report` из `sklearn.metrics`.

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Класс 0      | 0.99      | 0.99   | 0.99     | 980     |
| Класс 1      | 0.99      | 0.99   | 0.99     | 1135    |
| Класс 2      | 0.99      | 1.00   | 0.99     | 1032    |
| Класс 3      | 0.99      | 1.00   | 0.99     | 1010    |
| Класс 4      | 0.99      | 1.00   | 0.99     | 982     |
| Класс 5      | 0.99      | 0.98   | 0.99     | 892     |
| Класс 6      | 0.99      | 0.99   | 0.99     | 958     |
| Класс 7      | 1.00      | 0.98   | 0.99     | 1028    |
| Класс 8      | 0.99      | 0.99   | 0.99     | 974     |
| Класс 9      | 0.99      | 0.99   | 0.99     | 1009    |
| Класс 10     | 0.99      | 0.99   | 0.99     | 1000    |
| accuracy     |           |        | 0.99     | 11000   |
| macro avg    | 0.99      | 0.99   | 0.99     | 11000   |
| weighted avg | 0.99      | 0.99   | 0.99     | 11000   |

## 8. Вывод неверно классифицированных изображений.

Прогноз | реальный класс



## 9. Список неверно классифицированных изображений проверочного множества. Список упорядочен по числу ошибок в классе.

Индекс неверно классифицированного изображения | прогноз

Класс : 2

795 | 0



Класс : 3

5697 | 5



10010 | 8



Класс : 4

143 | 10



1100 | 7



2779 | 9



7940 | 9



Класс : 0

34 | 7



1314 | 8



8766 | 2



9292 | 10



10129 | 6



Класс : 8

2117 | 7



5650 | 0



6059 | 2



9200 | 9



10244 | 0



Класс : 1

1742 | 10



2407 | 10



3216 | 3



3302 | 6



7965 | 10



8981 | 3



Класс : 6

1977 | 0



2300 | 10



2339 | 8



4295 | 4



5040 | 0



5361 | 1



5984 | 1



Класс : 10

4430 | 1



7639 | 5



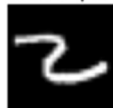
8118 | 3



9261 | 8



9877 | 2



10566 | 2



10794 | 2



Класс : 9

846 | 5



3103 | 4



3821 | 4



5331 | 4



6029 | 8



6160 | 4



7001 | 5



7194 | 3



7878 | 6



8363 | 4



8417 | 4



8767 | 8



10134 | 5



10376 | 4



Класс : 5

440 | 10



761 | 6



817 | 3



1256 | 6



1607 | 3



2822 | 3



3219 | 0



3848 | 6



4105 | 3



5202 | 3



5807 | 3



5829 | 6



5933 | 10



7962 | 0



7963 | 10



8734 | 3



10004 | 3



Класс : 7

281 | 9



321 | 2



617 | 2



2738 | 2



2778 | 10



3198 | 1



3724 | 9



3958 | 1



4066 | 2



4700 | 2



4855 | 2



7277 | 8



7762 | 2



7817 | 2



8490 | 10



9150 | 1



9790 | 9



10107 | 10



10570 | 2



10598 | 1

