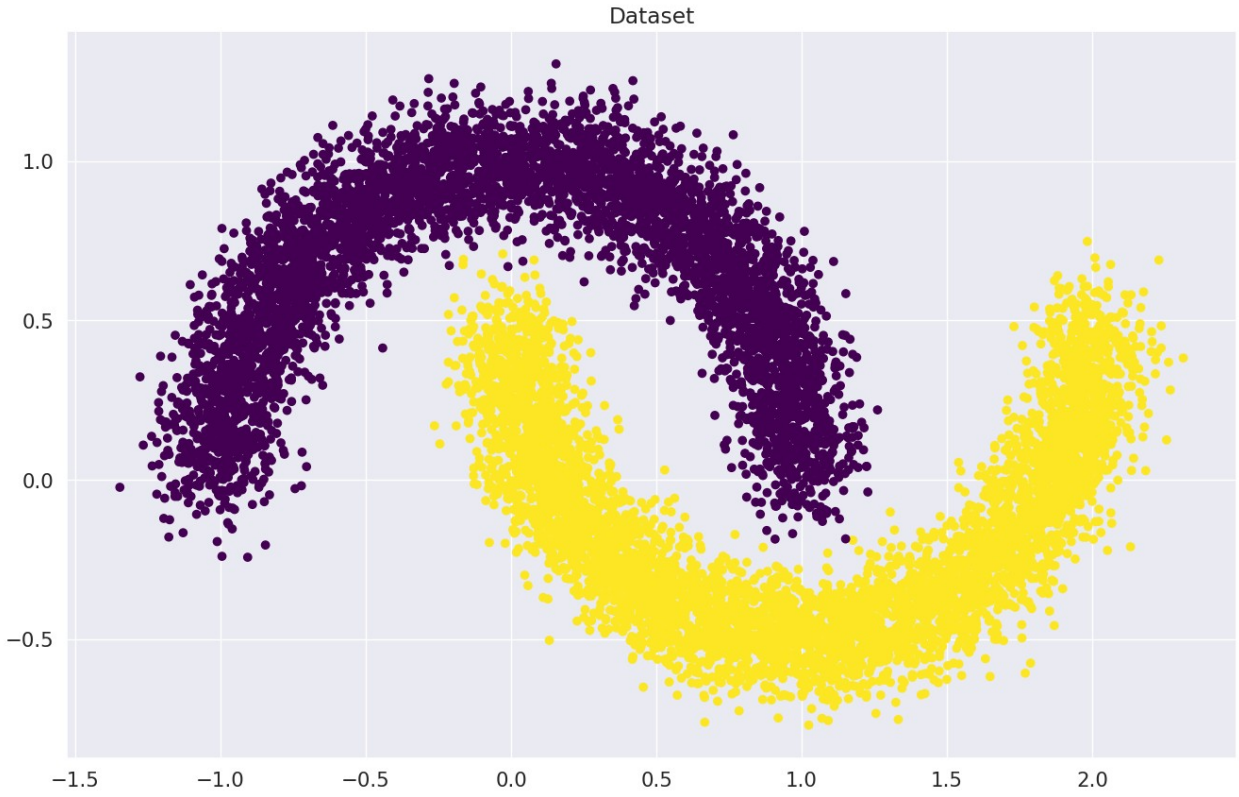
В этом занятии вам предстоит потренироваться построению нейронных сетей с помощью библиотеки Pytorch. Делать мы это будем на нескольких датасетах.

```
import numpy as np

import seaborn as sns
from matplotlib import pyplot as plt

from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

import torch
from torch import nn
from torch.nn import functional as F

from torch.utils.data import TensorDataset, DataLoader

sns.set(style="darkgrid", font_scale=1.4)
```

Часть 1. Датасет moons

Давайте сгенерируем датасет и посмотрим на него!

```
X, y = make_moons(n_samples=10000, random_state=42, noise=0.1)

plt.figure(figsize=(16, 10))
plt.title("Dataset")
plt.scatter(X[:, 0], X[:, 1], c=y, cmap="viridis")
plt.show()
```



Сделаем train/test split

```
X_train, X_val, y_train, y_val = train_test_split(X, y,
random_state=42)
```

Загрузка данных

В PyTorch загрузка данных как правило происходит налету (иногда датасеты не помещаются в оперативную память). Для этого используются две сущности `Dataset` и `DataLoader`.

1. `Dataset` загружает каждый объект по отдельности.
2. `DataLoader` группирует объекты из `Dataset` в батчи.

Так как наш датасет достаточно маленький мы будем использовать `TensorDataset`. Все, что нам нужно, это перевести из массива numpy в тензор с типом `torch.float32`.

Задание. Создайте тензоры с обучающими и тестовыми данными

```
X_train_t = torch.tensor(X_train)
y_train_t = torch.tensor(y_train)
X_val_t = torch.tensor(X_val)
y_val_t = torch.tensor(y_val)
```

Создаем `Dataset` и `DataLoader`.

```
train_dataset = TensorDataset(X_train_t, y_train_t)
val_dataset = TensorDataset(X_val_t, y_val_t)
train_dataloader = DataLoader(train_dataset, batch_size=128)
val_dataloader = DataLoader(val_dataset, batch_size=128)
```

Logistic regression is my profession

Напоминание Давайте вспоним, что происходит в логистической регрессии. На входе у нас есть матрица объект-признак X и столбец-вектор y – метки из $\{0,1\}$ для каждого объекта. Мы хотим найти такую матрицу весов W и смещение b (bias), что наша модель $XW+b$ будет каким-то образом предсказывать класс объекта. Как видно на выходе наша модель может выдавать число в интервале от $(-\infty;\infty)$. Этот выход как правило называют "логитами" (logits). Нам необходимо перевести его на интервал от $[0;1)$ для того, чтобы он выдавал нам вероятность принадлежности объекта к кассу один, также лучше, чтобы эта функция была монотонной, быстро считалась, имела производную и на $-\infty$ имела значение 0, а на $+\infty$ имела значение 1. Такой класс функций называется сигмоидыю. Чаще всего в качестве сигмоида берут

$$\sigma(x)=\frac{1}{1+e^{-x}}.$$

Задание. Реализация логистической регрессии

Вам необходимо написать модуль на PyTorch реализующий $logits=XW+b$, где W и b – параметры (`nn.Parameter`) модели. Иначе говоря, здесь мы реализуем своими руками модуль `nn.Linear` (в этом пункте его использование запрещено). Инициализируйте веса нормальным распределением (`torch.randn`).

```
class LinearRegression(nn.Module):
    def __init__(self, in_features: int, out_features: int, bias: bool
= True):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(in_features,
out_features))
        self.bias = bias
        if bias:
            self.bias_term = nn.Parameter(torch.randn(out_features))

    def forward(self, x):
        x = x @ self.weights
        if self.bias:
            x += self.bias_term
        return x

linear_regression = LinearRegression(2, 1)
loss_function = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(linear_regression.parameters(), lr=0.05)
```

Вопрос 1. Сколько обучаемых параметров у получившейся модели? Имеется в виду суммарное количество отдельных числовых переменных, а не количество тензоров.

На вход у нас имеется на 2 нейрона, а на выход 1. Так же мы используем bias. Следовательно, параметров у обучающей модели (2*1)+1

Train loop

Вот псевдокод, который поможет вам разобраться в том, что происходит во время обучения

```
for epoch in range(max_epochs):  # <----------- итерируемся по датасету несколько раз
    for x_batch, y_batch in dataset:  # <------ итерируемся по датасету. Так как мы используем SGD а не GD, то берем батчи заданного размера
        optimizer.zero_grad()  # <------------- обуляем градиенты модели
        outp = model(x_batch)  # <------------- получаем "логиты" из модели
        loss = loss_func(outp, y_batch)  # <--- считаем "лосс" для логистической регрессии
        loss.backward()  # <------------------- считаем градиенты
        optimizer.step()  # <------------------ делаем шаг градиентного спуска
        if convergence:  # <------------------- в случае сходимости выходим из цикла
            break
```

В коде ниже добавлено логирование `accuracy` и `loss`.

Задание. Реализация цикла обучения

```
tol = 1e-3
losses = []
max_epochs = 100
prev_weights = torch.zeros_like(linear_regression.weights)
stop_it = False
for epoch in range(max_epochs):
    for it, (X_batch, y_batch) in enumerate(train_dataloader):
        optimizer.zero_grad()
        X_batch = X_batch.type(torch.float32)
        y_batch = y_batch.type(torch.float32)
        y_batch = y_batch.view(-1, 1)  # изменить y_batch на (N, 1)
        outp =  linear_regression(X_batch)
        loss =  loss_function(outp, y_batch) # YOUR CODE. Compute loss
        loss.backward()
        losses.append(loss.detach().flatten()[0])
        optimizer.step()
        probabilities =  torch.sigmoid(outp)# YOUR CODE. Compute
```

```
Probabilities
        preds = (probabilities > 0.5).type(torch.long)
        batch_acc = (preds.flatten() ==
y_batch).type(torch.float32).sum() / y_batch.size(0)

        if (it + epoch * len(train_dataloader)) % 100 == 0:
            print(f"Iteration: {it + epoch * len(train_dataloader)}\
nBatch accuracy: {batch_acc}")
        current_weights = linear_regression.weights.detach().clone()
        if (prev_weights - current_weights).abs().max() < tol:
            print(f"\nIteration: {it + epoch *
len(train_dataloader)}.Convergence. Stopping iterations.")
            stop_it = True
            break
        prev_weights = current_weights
    if stop_it:
        break
```

```
Iteration: 0
Batch accuracy: 65.90625
Iteration: 100
Batch accuracy: 63.53125
Iteration: 200
Batch accuracy: 64.390625
Iteration: 300
Batch accuracy: 64.3125
Iteration: 400
Batch accuracy: 64.09375
Iteration: 500
Batch accuracy: 64.03125
Iteration: 600
Batch accuracy: 65.875
Iteration: 700
Batch accuracy: 63.75
Iteration: 800
Batch accuracy: 64.125
Iteration: 900
Batch accuracy: 63.609375

Iteration: 932.Convergence. Stopping iterations.
```

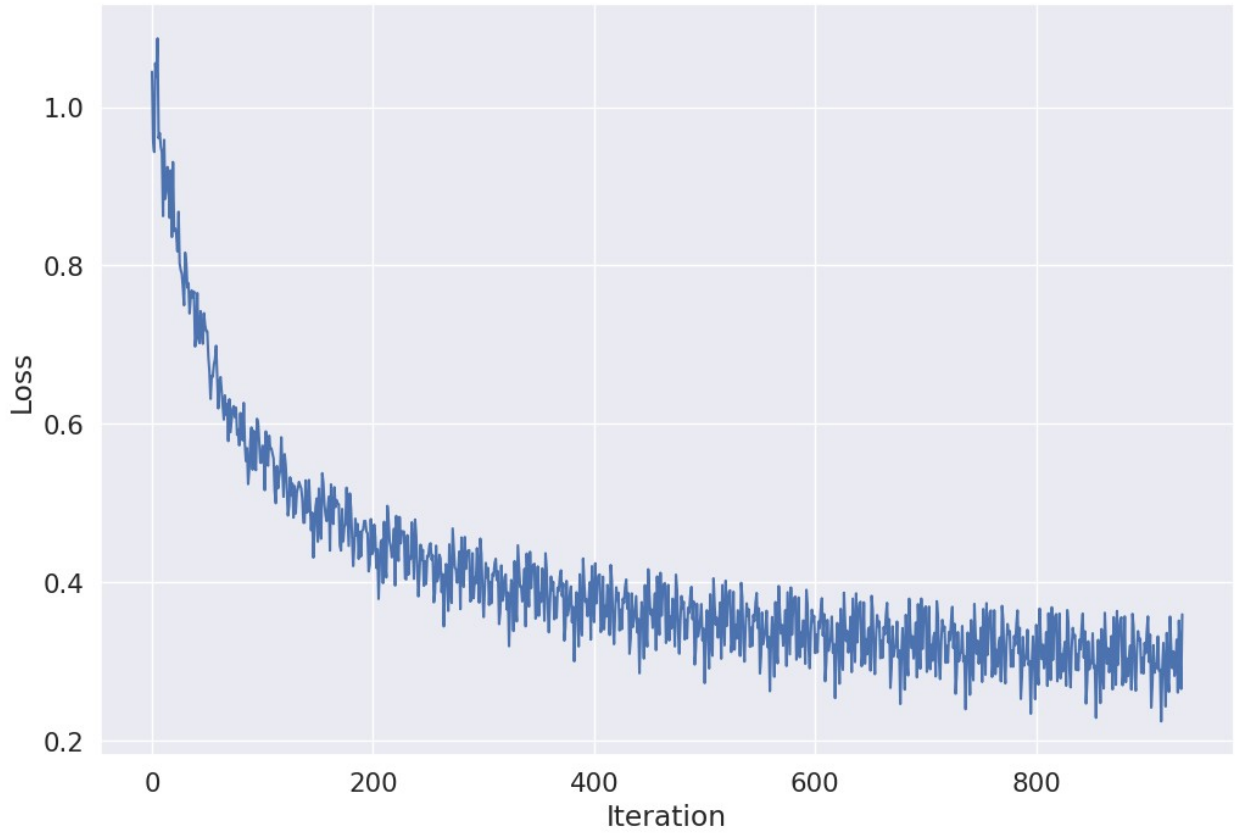
Вопрос 2. Сколько итераций потребовалось, чтобы алгоритм сошелся?

Ответ: 932

Визуализируем результаты

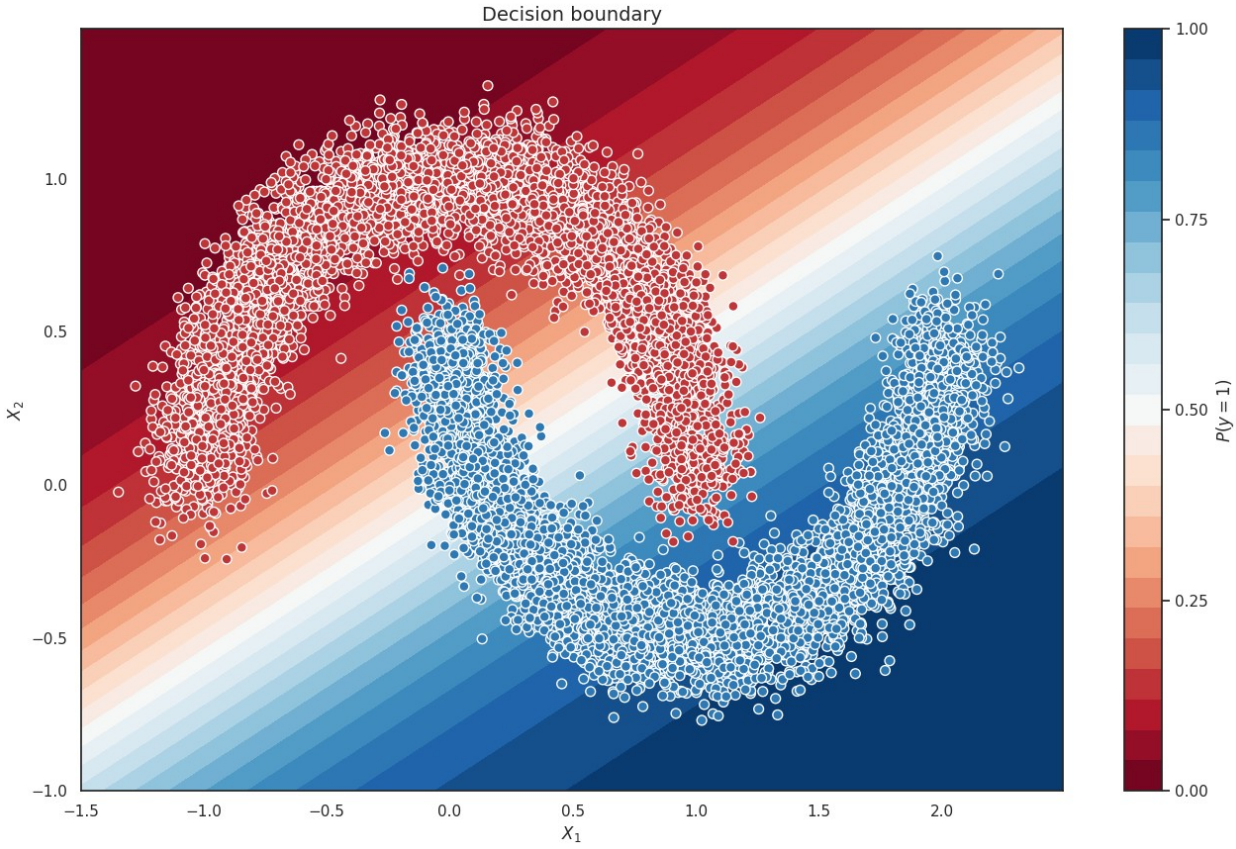
```
plt.figure(figsize=(12, 8))
plt.plot(range(len(losses)), losses)
plt.xlabel("Iteration")
```

```
plt.ylabel("Loss")
plt.show()
```




```
import numpy as np

sns.set(style="white")

xx, yy = np.mgrid[-1.5:2.5:.01, -1.:1.5:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
batch = torch.from_numpy(grid).type(torch.float32)
with torch.no_grad():
    probs = torch.sigmoid(linear_regression(batch).reshape(xx.shape))
    probs = probs.numpy().reshape(xx.shape)

f, ax = plt.subplots(figsize=(16, 10))
ax.set_title("Decision boundary", fontsize=14)
contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",
                      vmin=0, vmax=1)
ax_c = f.colorbar(contour)
ax_c.set_label("$P(y = 1)$")
ax_c.set_ticks([0, .25, .5, .75, 1])

ax.scatter(X[100:,0], X[100:, 1], c=y[100:], s=50,
```

```
            cmap="RdBu", vmin=-.2, vmax=1.2,
            edgecolor="white", linewidth=1)

ax.set(xlabel="$X_1$", ylabel="$X_2$")
plt.show()
```



Задание. Реализуйте predict и посчитайте accuracy на test.

```
@torch.no_grad()
def predict(dataloader, model):
    model.eval()
    predictions = np.array([])
    for x_batch, _ in dataloader:
        #<YOUR CODE>

        with torch.no_grad():
            nn_prediction = model(torch.FloatTensor(x_batch))
#X_test))
            nn_prediction = nn_prediction.tolist()

        preds = np.array([int(x[0] > 0.5) for x in nn_prediction])
#YOUR CODE. Compute predictions
        predictions = np.hstack((predictions,
```

```
preds.numpy().flatten()))
    return predictions.flatten()

from sklearn.metrics import accuracy_score

y_true = y_batch

accuracy = accuracy_score(y_true.numpy(), preds.numpy())
print("Accuracy: ", accuracy)
```

```
Accuracy:  0.8203125
```

Вопрос 3

Какое `accuracy` получается после обучения?

Ответ: 0.8203125

Часть 2. Датасет MNIST

Датасет MNIST содержит рукописные цифры. Загрузим датасет и создадим DataLoader-ы. Пример можно найти в семинаре по полносвязным нейронным сетям.

```
import os
from torchvision.datasets import MNIST
from torchvision import transforms as tfs

from torchsummary import summary



data_tfs = tfs.Compose([
    tfs.ToTensor(),
    tfs.Normalize((0.5), (0.5))
])

# install for train and test
root = './'
train_dataset = MNIST(root, train=True,  transform=data_tfs,
download=True)
val_dataset  = MNIST(root, train=False, transform=data_tfs,
download=True)

train_dataloader =  torch.utils.data.DataLoader(train_dataset,
batch_size=4,
                                                shuffle=True, num_workers=2)
# YOUR CODE GOES HERE
valid_dataloader =  torch.utils.data.DataLoader(val_dataset,
batch_size=4,
```

```
                                                shuffle=False, num_workers=2)
# YOUR CODE GOES HERE
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
images-idx3-ubyte.gz to ./MNIST/raw/train-images-idx3-ubyte.gz

100%|██████████| 9.91M/9.91M [00:00<00:00, 34.1MB/s]

Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
labels-idx1-ubyte.gz to ./MNIST/raw/train-labels-idx1-ubyte.gz

100%|██████████| 28.9k/28.9k [00:00<00:00, 1.06MB/s]

Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz to ./MNIST/raw/t10k-images-idx3-ubyte.gz

100%|██████████| 1.65M/1.65M [00:00<00:00, 8.04MB/s]

Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz
```

```
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|██████████| 4.54k/4.54k [00:00<00:00, 8.01MB/s]

Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw
```

Часть 2.1. Полносвязные нейронные сети

Сначала решим MNIST с помощью полносвязной нейронной сети.

```
class Identical(nn.Module):
    def forward(self, x):
        return x
```

Задание. Простая полносвязная нейронная сеть

Создайте полносвязную нейронную сеть с помощью класса Sequential. Сеть состоит из:

- Уплощения матрицы в вектор (nn.Flatten);
- Двух скрытых слоёв из 128 нейронов с активацией nn.ELU;
- Выходного слоя с 10 нейронами.

Задайте лосс для обучения (кросс-энтропия).

```
activation = nn.ELU


model = nn.Sequential(
    nn.Flatten(),                          # Уплощение
    nn.Linear(784, 128),                  # Предполагая, что вход имеет размер 784 (например, изображение 28x28)
    activation(),
    nn.Linear(128, 128),                  # Второй скрытый слой
    activation(),
    nn.Linear(128, 10)                     # Выходной слой с 10 нейронами
)
    #YOUR CODE. Add layers to your sequential class

criterion = torch.nn.CrossEntropyLoss() #YOUR CODE. Select a loss function
optimizer = torch.optim.Adam(model.parameters())

loaders = {"train": train_dataloader, "valid": valid_dataloader}

device = "cuda" if torch.cuda.is_available() else "cpu"
```

Train loop (seriously)

Давайте разберемся с кодом ниже, который подойдет для 90% задач в будущем.

```
for epoch in range(max_epochs):  # <--------------- итерируемся по датасету несколько раз
    for k, dataloader in loaders.items():  # <----- несколько dataloader для train / valid / test
        for x_batch, y_batch in dataloader:  # <--- итерируемся по датасету. Так как мы используем SGD а не GD, то берем батчи заданного размера
            if k == "train":
                model.train()  # <------------------ переводим модель в режим train
                optimizer.zero_grad()  # <--------- обнуляем градиенты модели
                outp = model(x_batch)
                loss = criterion(outp, y_batch) # <-считаем "лосс" для логистической регрессии
                loss.backward()  # <--------------- считаем градиенты
                optimizer.step()  # <-------------- делаем шаг градиентного спуска
            else:  # <----------------------------- test/eval
                model.eval()  # <------------------ переводим модель в режим eval
                with torch.no_grad():  # <--------- НЕ считаем градиенты
                    outp = model(x_batch)  # <------------- получаем "логиты" из модели
            count_metrics(outp, y_batch)  # <-------------- считаем метрики
```

Задание. Дополните цикл обучения.

```
max_epochs = 10
accuracy = {"train": [], "valid": []}
for epoch in range(max_epochs):
    for k, dataloader in loaders.items():

        epoch_correct = 0
        epoch_all = 0

        for x_batch, y_batch in dataloader:
            x_batch = x_batch.view(x_batch.size(0), -1)
            if k == "train":
                model.train()  # Устанавливаем режим обучения
                optimizer.zero_grad()  # Обнуляем градиенты
                outp = model(x_batch)

                 # YOUR CODE. Set model to ``train`` mode and
```

```
calculate outputs. Don't forget zero_grad!
            else:
                model.eval()  # Устанавливаем режим валидации
                with torch.no_grad():  # Не вычисляем градиенты
                    outp = model(x_batch)
                 # YOUR CODE. Set model to ``eval`` mode and calculate
outputs
            preds = outp.argmax(-1)
            correct = (preds == y_batch).sum()  # Подсчет правильных
предсказаний  # YOUR CODE GOES HERE
            all = y_batch.size(0)  # Общее количество элементов # YOUR
CODE GOES HERE

            epoch_correct += correct.item()
            epoch_all += all
            if k == "train":
                loss = criterion(outp, y_batch)  # Вычисляем потерю
                loss.backward()  # Обратное распространение
                optimizer.step()  # Обновляем параметры

                # YOUR CODE. Calculate gradients and make a step of
your optimizer
        if k == "train":
            print(f"Epoch: {epoch+1}")
        print(f"Loader: {k}. Accuracy: {epoch_correct/epoch_all}")
        accuracy[k].append(epoch_correct/epoch_all)
```

```
Epoch: 1
Loader: train. Accuracy: 0.9068833333333334
Loader: valid. Accuracy: 0.9513
Epoch: 2
Loader: train. Accuracy: 0.9455833333333333
Loader: valid. Accuracy: 0.948
Epoch: 3
Loader: train. Accuracy: 0.9553333333333334
Loader: valid. Accuracy: 0.9465
Epoch: 4
Loader: train. Accuracy: 0.9596
Loader: valid. Accuracy: 0.9517
Epoch: 5
Loader: train. Accuracy: 0.96315
Loader: valid. Accuracy: 0.9605
Epoch: 6
Loader: train. Accuracy: 0.96525
Loader: valid. Accuracy: 0.9579
Epoch: 7
Loader: train. Accuracy: 0.9669333333333333
Loader: valid. Accuracy: 0.9608
Epoch: 8
Loader: train. Accuracy: 0.9699833333333333
```

```
Loader: valid. Accuracy: 0.9637
Epoch: 9
Loader: train. Accuracy: 0.9702166666666666
Loader: valid. Accuracy: 0.9672
Epoch: 10
Loader: train. Accuracy: 0.9709666666666666
Loader: valid. Accuracy: 0.9679
```

Задание. Протестируйте разные функции активации.

Попробуйте разные функции активации. Для каждой функции активации посчитайте массив validation accuracy. Лучше реализовать это в виде функции, берущей на вход активацию и получающей массив из accuracies.

```
elu_accuracy = accuracy["valid"]


def train_and_validate(activation_function, train_dataloader,
valid_dataloader, max_epochs, device):
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, 128),
        activation_function(),
        nn.Linear(128, 128),
        activation_function(),
        nn.Linear(128, 10)
    ).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters())

    validation_accuracy = []

    for epoch in range(max_epochs):
        # Training phase
        model.train()
        for x_batch, y_batch in train_dataloader:
            optimizer.zero_grad()
            x_batch = x_batch.view(x_batch.size(0), -1).to(device)  #
Убедитесь, что x_batch на правильном устройстве
            outp = model(x_batch)
            loss = criterion(outp, y_batch.to(device))
            loss.backward()
            optimizer.step()

        # Validation phase
        model.eval()
        epoch_correct = 0
        epoch_all = 0
```

```
        with torch.no_grad():
            for x_batch, y_batch in valid_dataloader:
                x_batch = x_batch.view(x_batch.size(0), -1).to(device)
                outp = model(x_batch)
                preds = outp.argmax(-1)
                epoch_correct += (preds ==
y_batch.to(device)).sum().item()
                epoch_all += y_batch.size(0)

        accuracy = epoch_correct / epoch_all
        validation_accuracy.append(accuracy)
        print(f"Epoch {epoch+1}/{max_epochs}, Validation Accuracy:
{accuracy:.4f}")

    return validation_accuracy

# YOUR CODE. Do the same thing with other activations (it's better to
wrap into a function that returns a list of accuracies)

def test_activation_function(activation_function):
    return train_and_validate(activation_function, train_dataloader,
valid_dataloader, max_epochs, device)

plain_accuracy = test_activation_function(Identical)
relu_accuracy = test_activation_function(nn.ReLU) #YOUR CODE
leaky_relu_accuracy = test_activation_function(nn.LeakyReLU) #YOUR
CODE
```

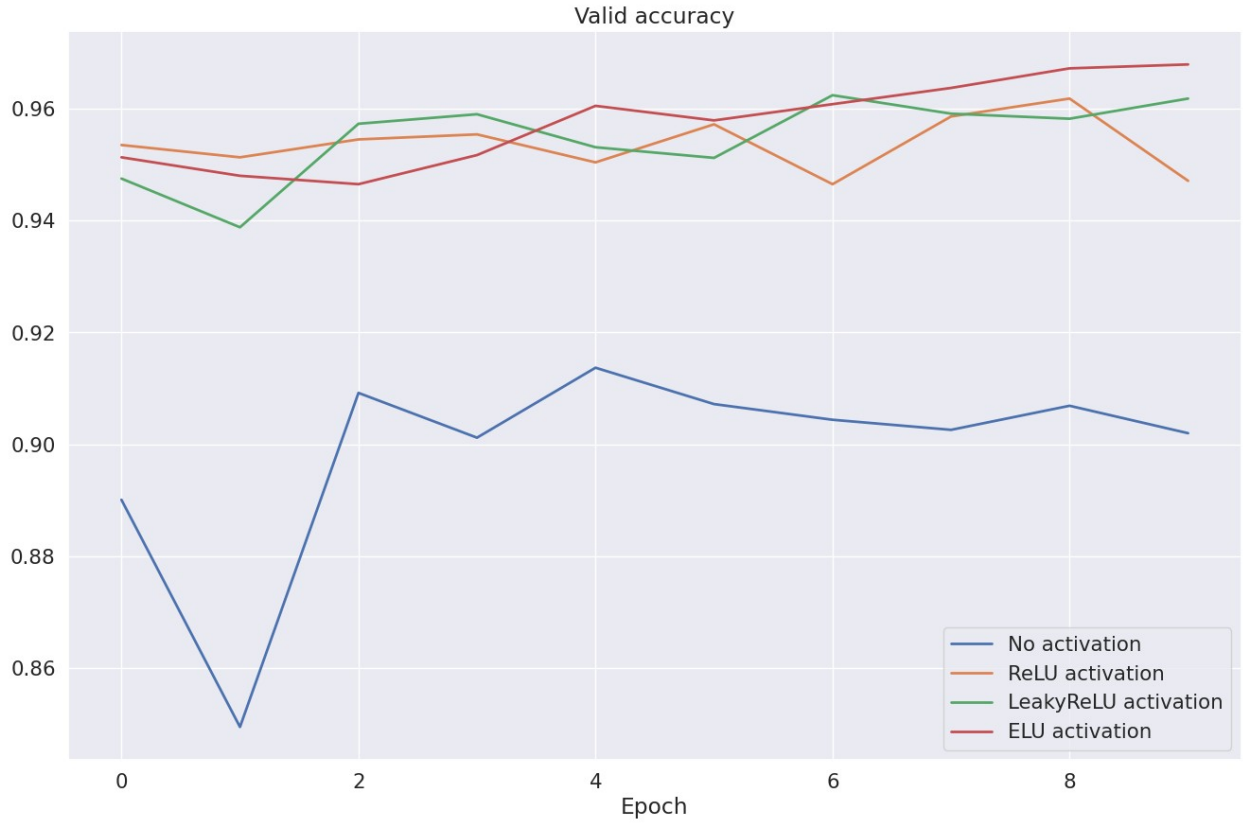
```
Epoch 1/10, Validation Accuracy: 0.8901
Epoch 2/10, Validation Accuracy: 0.8495
Epoch 3/10, Validation Accuracy: 0.9092
Epoch 4/10, Validation Accuracy: 0.9012
Epoch 5/10, Validation Accuracy: 0.9137
Epoch 6/10, Validation Accuracy: 0.9072
Epoch 7/10, Validation Accuracy: 0.9044
Epoch 8/10, Validation Accuracy: 0.9026
Epoch 9/10, Validation Accuracy: 0.9069
Epoch 10/10, Validation Accuracy: 0.9020
Epoch 1/10, Validation Accuracy: 0.9535
Epoch 2/10, Validation Accuracy: 0.9513
Epoch 3/10, Validation Accuracy: 0.9545
Epoch 4/10, Validation Accuracy: 0.9554
Epoch 5/10, Validation Accuracy: 0.9504
Epoch 6/10, Validation Accuracy: 0.9572
Epoch 7/10, Validation Accuracy: 0.9465
Epoch 8/10, Validation Accuracy: 0.9586
Epoch 9/10, Validation Accuracy: 0.9618
Epoch 10/10, Validation Accuracy: 0.9471
Epoch 1/10, Validation Accuracy: 0.9475
Epoch 2/10, Validation Accuracy: 0.9388
```

```
Epoch 3/10, Validation Accuracy: 0.9573
Epoch 4/10, Validation Accuracy: 0.9590
Epoch 5/10, Validation Accuracy: 0.9531
Epoch 6/10, Validation Accuracy: 0.9512
Epoch 7/10, Validation Accuracy: 0.9624
Epoch 8/10, Validation Accuracy: 0.9591
Epoch 9/10, Validation Accuracy: 0.9582
Epoch 10/10, Validation Accuracy: 0.9618
```

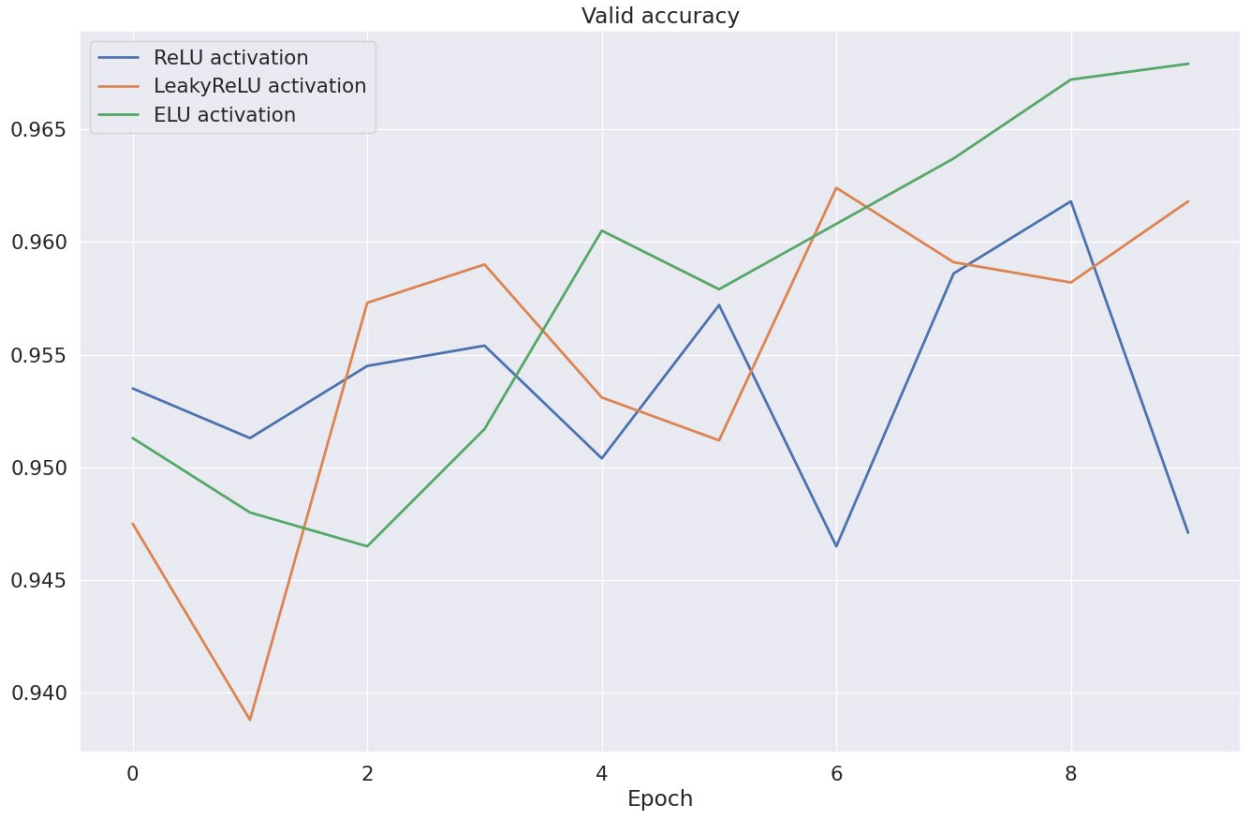
Accuracy

Построим график accuracy/epoch для каждой функции активации.

```
sns.set(style="darkgrid", font_scale=1.4)

plt.figure(figsize=(16, 10))
plt.title("Valid accuracy")
plt.plot(range(max_epochs), plain_accuracy, label="No activation",
linewidth=2)
plt.plot(range(max_epochs), relu_accuracy, label="ReLU activation",
linewidth=2)
plt.plot(range(max_epochs), leaky_relu_accuracy, label="LeakyReLU
activation", linewidth=2)
plt.plot(range(max_epochs), elu_accuracy, label="ELU activation",
linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```




```
plt.figure(figsize=(16, 10))
plt.title("Valid accuracy")
plt.plot(range(max_epochs), relu_accuracy, label="ReLU activation",
linewidth=2)
plt.plot(range(max_epochs), leaky_relu_accuracy, label="LeakyReLU
activation", linewidth=2)
plt.plot(range(max_epochs), elu_accuracy, label="ELU activation",
linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```



Вопрос 4. Какая из активаций показала наивысший `accuracy` к концу обучения?

Ответ: `ELU` показывает лучший `accuracy`

Часть 2.2 Сверточные нейронные сети

Ядра

Сначала немного поработам с самим понятием ядра свёртки.

```
!wget https://img.the-village.kz/the-village.com.kz/post-cover/5x5-
I6oiwjmq79dMCZMEbA-default.jpg -O sample_photo.jpg

--2024-11-18 10:27:54--  https://img.the-village.kz/the-
village.com.kz/post-cover/5x5-I6oiwjmq79dMCZMEbA-default.jpg
Resolving img.the-village.kz (img.the-village.kz)... 5.9.226.237
Connecting to img.the-village.kz (img.the-village.kz)|
5.9.226.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49337 (48K) [image/jpeg]
Saving to: ‘sample_photo.jpg’

sample_photo.jpg    100%[===================>]  48.18K   209KB/s    in
0.2s
```

```
2024-11-18 10:27:55 (209 KB/s) - ‘sample_photo.jpg’ saved
[49337/49337]
```

```
import cv2
sns.set(style="white")
img = cv2.imread("sample_photo.jpg")
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(12, 8))
plt.imshow(RGB_img)
plt.show()
```

Попробуйте посмотреть как различные свертки влияют на фото. Например, попробуйте

А)

```
[0, 0, 0],
[0, 1, 0],
[0, 0, 0]
```

Б)

```
[0, 1, 0],
[0, -2, 0],
[0, 1, 0]
```

В)

```
[0, 0, 0],
[1, -2, 1],
[0, 0, 0]
```

Г)

```
[0, 1, 0],
[1, -4, 1],
[0, 1, 0]
```

Д)

```
[0, -1, 0],
[-1, 5, -1],
[0, -1, 0]
```

Е)

```
[0.0625, 0.125, 0.0625],
[0.125, 0.25, 0.125],
[0.0625, 0.125, 0.0625]
```

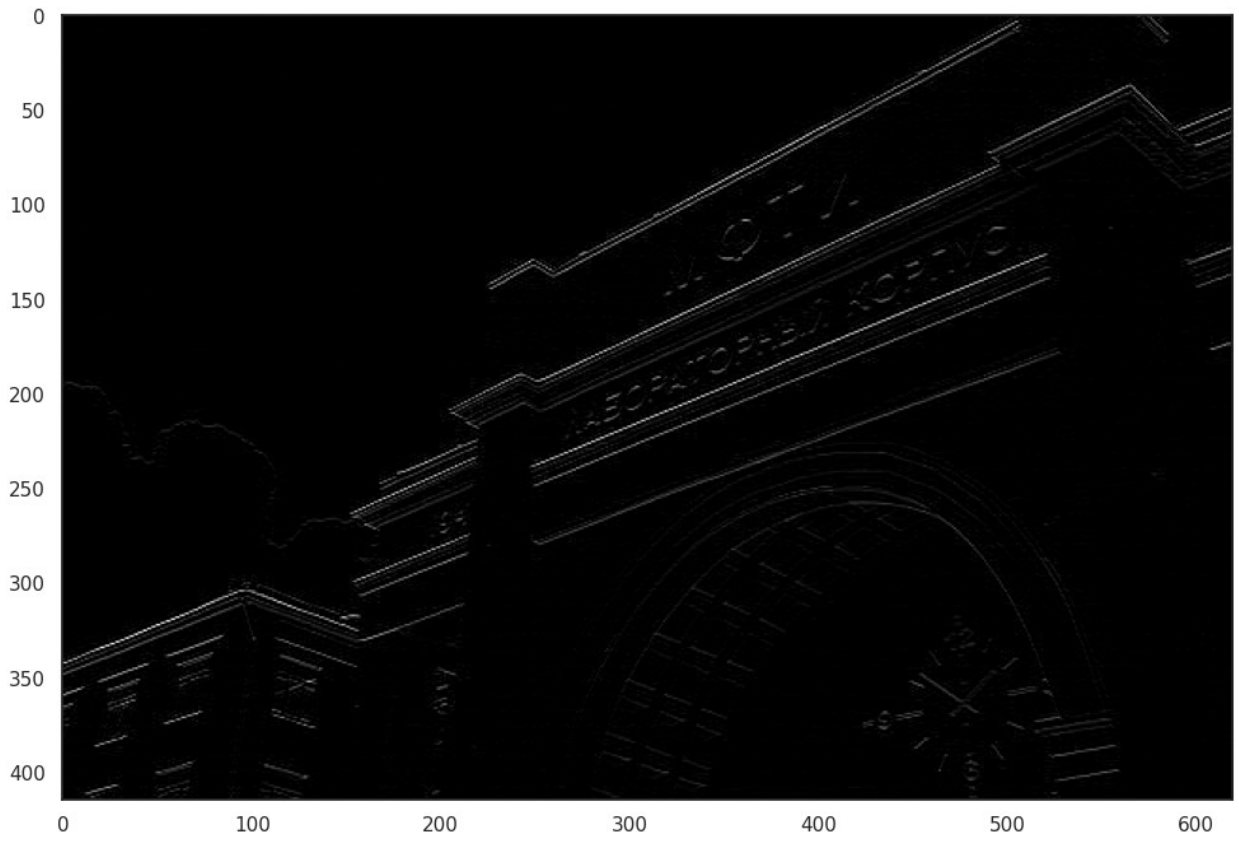
Не стесняйтесь пробовать свои варианты!

```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = torch.tensor([
    [0, 0, 0],
    [0, 1, 0],
    [0, 0, 0]
]).reshape(1, 1, 3, 3).type(torch.float32)

kernel = kernel.repeat(3, 3, 1, 1)
img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

result = F.conv2d(img_t, kernel)[0]


plt.figure(figsize=(12, 8))
result_np = result.permute(1, 2, 0).numpy() / 256 / 3
```

```
plt.imshow(result_np)
plt.show()
```


```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = torch.tensor([
    [0, 1, 0],
    [0, -2, 0],
    [0, 1, 0]
]).reshape(1, 1, 3, 3).type(torch.float32)

kernel = kernel.repeat(3, 3, 1, 1)
img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

result = F.conv2d(img_t, kernel)[0]


plt.figure(figsize=(12, 8))
result_np = result.permute(1, 2, 0).numpy() / 256 / 3

plt.imshow(result_np)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = torch.tensor([
    [0, 0, 0],
    [1, -2, 1],
    [0, 0, 0]
]).reshape(1, 1, 3, 3).type(torch.float32)

kernel = kernel.repeat(3, 3, 1, 1)
img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

result = F.conv2d(img_t, kernel)[0]


plt.figure(figsize=(12, 8))
result_np = result.permute(1, 2, 0).numpy() / 256 / 3

plt.imshow(result_np)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = torch.tensor([
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]
]).reshape(1, 1, 3, 3).type(torch.float32)

kernel = kernel.repeat(3, 3, 1, 1)
img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

result = F.conv2d(img_t, kernel)[0]


plt.figure(figsize=(12, 8))
result_np = result.permute(1, 2, 0).numpy() / 256 / 3

plt.imshow(result_np)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = torch.tensor([
    [0, -1, 0],
[-1, 5, -1],
[0, -1, 0]
]).reshape(1, 1, 3, 3).type(torch.float32)

kernel = kernel.repeat(3, 3, 1, 1)
img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

result = F.conv2d(img_t, kernel)[0]


plt.figure(figsize=(12, 8))
result_np = result.permute(1, 2, 0).numpy() / 256 / 3

plt.imshow(result_np)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = torch.tensor([
    [0.0625, 0.125, 0.0625],
[0.125, 0.25, 0.125],
[0.0625, 0.125, 0.0625]
]).reshape(1, 1, 3, 3).type(torch.float32)

kernel = kernel.repeat(3, 3, 1, 1)
img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

result = F.conv2d(img_t, kernel)[0]


plt.figure(figsize=(12, 8))
result_np = result.permute(1, 2, 0).numpy() / 256 / 3

plt.imshow(result_np)
plt.show()
```

Вопрос 5. Как можно описать действия ядер, приведенных выше? Сопоставьте для каждой буквы число.

1) Размытие - Е

2) Увеличение резкости - Д

3) Тождественное преобразование - А

4) Выделение вертикальных границ - В

5) Выделение горизонтальных границ - Б

6) Выделение границ - Г

Ответ: ЕДАВБГ

Задание. Реализуйте LeNet

Если мы сделаем параметры сверток обучаемыми, то можем добиться хороших результатов для задач компьютерного зрения. Реализуйте архитектуру LeNet, предложенную еще в 1998 году! На этот раз используйте модульную структуру (без помощи класса Sequential).

Наша нейронная сеть будет состоять из

- Свёртки 3x3 (1 карта на входе, 6 на выходе) с активацией ReLU;

- MaxPooling-а 2x2;
- Свёртки 3x3 (6 карт на входе, 16 на выходе) с активацией ReLU;
- MaxPooling-а 2x2;
- Уплощения (nn.Flatten);
- Полносвязного слоя со 120 нейронами и активацией ReLU;
- Полносвязного слоя с 84 нейронами и активацией ReLU;
- Выходного слоя из 10 нейронов.

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # Первый свёрточный слой
        self.conv1 = nn.Conv2d(1, 6, kernel_size=3)  # 1 вход, 6
выходов
        self.pool1 = nn.MaxPool2d(kernel_size=2)     # MaxPooling 2x2

        # Второй свёрточный слой
        self.conv2 = nn.Conv2d(6, 16, kernel_size=3) # 6 входов, 16
выходов
        self.pool2 = nn.MaxPool2d(kernel_size=2)     # MaxPooling 2x2

        # Полносвязные слои
        self.fc1 = nn.Linear(16 * 53 * 53, 120)      # Вход 16*53*53
после двух пуллингов
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)                  # Выход 10
нейронов

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = x.view(-1, 16 * 53 * 53)  # Уплощение
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)               # Выходной слой
        return x

model = LeNet().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

loaders = {"train": train_dataloader, "valid": valid_dataloader}
```

Задание. Обучите CNN

Используйте код обучения, который вы написали для полносвязной нейронной сети.

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

model = nn.Sequential(
    nn.Flatten(),                          # Уплотнение
    nn.Linear(784, 128),                  # Первый скрытый слой
    nn.ReLU(),                             # Активация
    nn.Linear(128, 128),                   # Второй скрытый слой
    nn.ReLU(),                             # Активация
    nn.Linear(128, 10)                     # Выходной слой (для 10
классов)
)

for epoch in range(max_epochs):
    model.train()
    for x_batch, y_batch in train_dataloader:
        optimizer.zero_grad()
        outp = model(x_batch)                          # Прогон данных
через модель
        loss = criterion(outp, y_batch)                # Вычисление потерь
        loss.backward()                                 # Обратное
распространение
        optimizer.step()                                # Обновление
параметров
```

Сравним с предыдущем пунктом

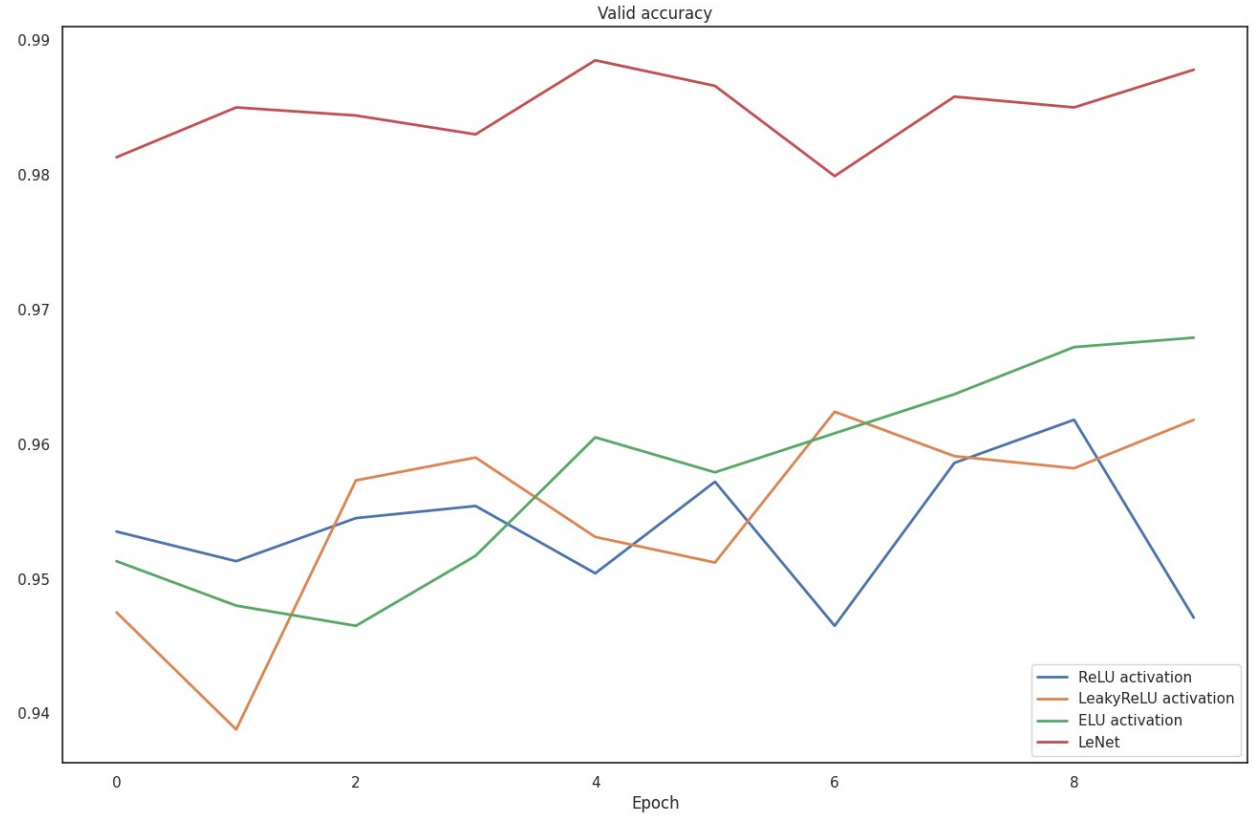
```
lenet_accuracy = []

for epoch in range(max_epochs):
    model.train()
    for x_batch, y_batch in train_dataloader:
        optimizer.zero_grad()
        outp = model(x_batch)
        loss = criterion(outp, y_batch)
        loss.backward()
        optimizer.step()

    # Валидация
    model.eval()
    epoch_correct = 0
    epoch_all = 0
    with torch.no_grad():
        for x_batch, y_batch in valid_dataloader:
            outp = model(x_batch)
            preds = outp.argmax(-1)
            epoch_correct += (preds == y_batch).sum().item()
            epoch_all += y_batch.size(0)
```

```
    # Вычисляем точность и сохраняем
    accuracy_value = epoch_correct / epoch_all
    lenet_accuracy.append(accuracy_value)


import torch.optim as optim

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # Первый свёрточный слой
        self.conv1 = nn.Conv2d(1, 6, kernel_size=3)  # 1 вход, 6
выходов
        self.pool1 = nn.MaxPool2d(kernel_size=2)     # MaxPooling 2x2

        # Второй свёрточный слой
        self.conv2 = nn.Conv2d(6, 16, kernel_size=3) # 6 входов, 16
выходов
        self.pool2 = nn.MaxPool2d(kernel_size=2)     # MaxPooling 2x2

        # Инициализация полносвязных слоев, вычислим размер
автоматически
        self._calculate_fc_input_size()

        # Полносвязные слои
        self.fc1 = nn.Linear(self.fc_input_size, 120)  # Вход
размером, вычисленным автоматически
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)  # Выход 10 нейронов

    def _calculate_fc_input_size(self):
        # Создаём тестовый тензор с размером (1, 28, 28) (например,
для изображений 28x28)
        with torch.no_grad():
            dummy_input = torch.zeros(1, 1, 28, 28)  # Пример для
изображений 28x28
            x = self.conv1(dummy_input)
            x = self.pool1(x)
            x = self.conv2(x)
            x = self.pool2(x)
            # Получаем размер после всех операций свертки и пуллинга
            self.fc_input_size = x.view(-1).size(0)  # Вычисляем
размер для полносвязного слоя

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = torch.flatten(x, 1)  # Уплощение
```

```
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)  # Выходной слой
        return x


# Замените это на устройство, если используете CUDA
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Инициализация модели, функции потерь и оптимизатора
model = LeNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# Тренировка
max_epochs = 10


lenet_accuracy = []

for epoch in range(max_epochs):
    model.train()
    for x_batch, y_batch in train_dataloader:
        x_batch, y_batch = x_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        outp = model(x_batch)  # Прогон данных через модель
        loss = criterion(outp, y_batch)  # Вычисление потерь
        loss.backward()  # Обратное распространение
        optimizer.step()

    # Валидация
    model.eval()
    epoch_correct = 0
    epoch_all = 0
    with torch.no_grad():
        for x_batch, y_batch in valid_dataloader:
            x_batch, y_batch = x_batch.to(device), y_batch.to(device)
            outp = model(x_batch)
            preds = outp.argmax(dim=-1)
            epoch_correct += (preds == y_batch).sum().item()
            epoch_all += y_batch.size(0)

    accuracy_value = epoch_correct / epoch_all
    lenet_accuracy.append(accuracy_value)
    print(f"Epoch {epoch + 1}/{max_epochs}, Accuracy:
{accuracy_value:.4f}")

# Выводим финальную точность
print(f"Final Accuracy: {lenet_accuracy[-1]:.4f}")
```

```
Epoch 1/10, Accuracy: 0.9813
Epoch 2/10, Accuracy: 0.9850
Epoch 3/10, Accuracy: 0.9844
Epoch 4/10, Accuracy: 0.9830
Epoch 5/10, Accuracy: 0.9885
Epoch 6/10, Accuracy: 0.9866
Epoch 7/10, Accuracy: 0.9799
Epoch 8/10, Accuracy: 0.9858
Epoch 9/10, Accuracy: 0.9850
Epoch 10/10, Accuracy: 0.9878
Final Accuracy: 0.9878
```

```
plt.figure(figsize=(16, 10))
plt.title("Valid accuracy")
plt.plot(range(max_epochs), relu_accuracy, label="ReLU activation",
linewidth=2)
plt.plot(range(max_epochs), leaky_relu_accuracy, label="LeakyReLU
activation", linewidth=2)
plt.plot(range(max_epochs), elu_accuracy, label="ELU activation",
linewidth=2)
plt.plot(range(max_epochs), lenet_accuracy, label="LeNet",
linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```



```
lenet_accuracy = accuracy["valid"]
```

Вопрос 6 Какое `accuracy` получается после обучения с точностью до двух знаков после запятой?

Ответ: 0,99