

▼ ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

Дисциплина: Методы машинного обучения

Студент: Шалыгин Георгий

Группа: НФИ-02

▼ Москва 2023

▼ Вариант № 10

Загрузите заданный в индивидуальном задании набор данных из Tensorflow Datasets, включая указанные в задании независимые признаки и метку класса.

- Набор данных: wine_quality
- Независимые признаки: features/density, features/chlorides
- Метка класса: quality

```
import tensorflow_datasets as tfds
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
data = tfds.load('wine_quality', split='train')
```

```
data = tfds.as_dataframe(data)
```

```
data.head()
```

	features/alcohol	features/chlorides	features/citric acid	features/density	features, a
0	9.0	0.054	0.34	1.00080	
1	12.2	0.063	0.49	0.99110	
2	11.2	0.029	0.11	0.99076	
3	9.0	0.110	0.27	0.99672	
4	12.0	0.035	0.30	0.99016	



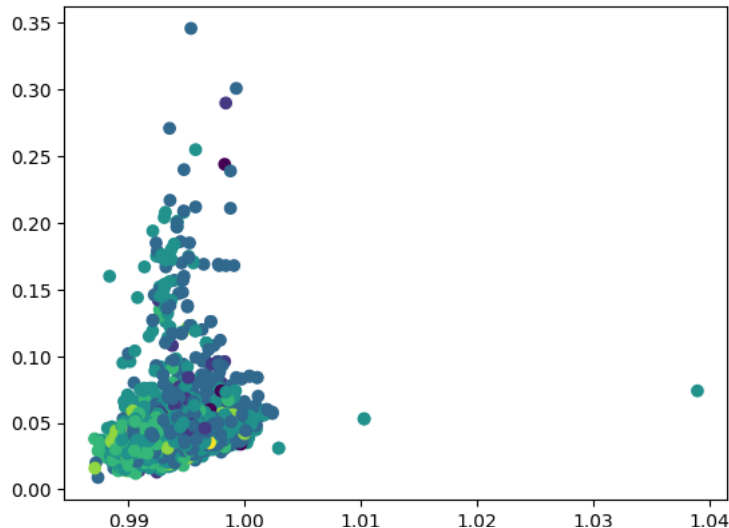
2. Визуализируйте точки набора данных на плоскости с координатами, соответствующими двум независимым признакам, отображая точки различных классов разными цветами. Подпишите оси и рисунок, создайте легенду для классов набора данных.

```
data['quality'].unique()
```

```
array([5, 6, 4, 8, 7, 3, 9], dtype=int32)
```

```
plt.scatter(data['features/density'], data['features/chlorides'], c=data['quality'])
```

```
<matplotlib.collections.PathCollection at 0x7f55e57d2890>
```

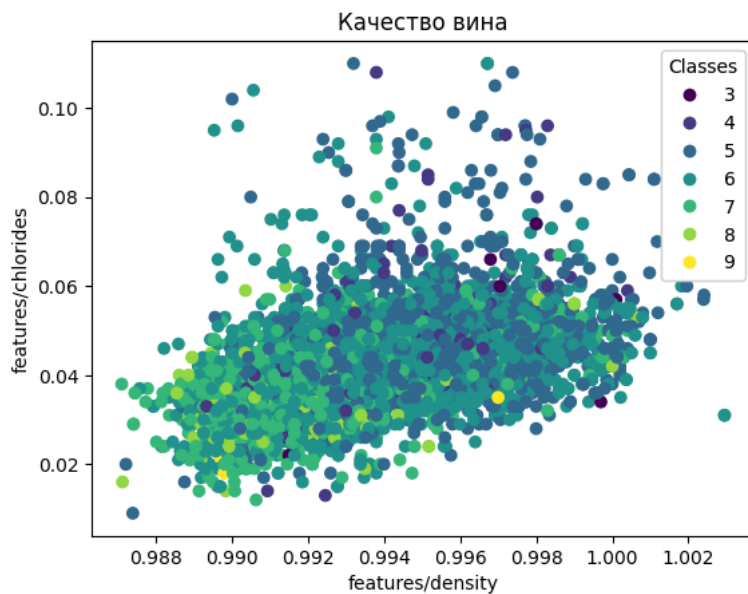


очевидно есть выбросы, уберем выходящие за 3 сигма

```
data = data[np.abs(data['features/density'] - data['features/density'].mean()) < 3 * data['features/density'].std()]
data = data[np.abs(data['features/chlorides'] - data['features/chlorides'].mean()) < 3 * data['features/chlorides'].std()]
```

```
fig, ax = plt.subplots()
scatter = ax.scatter(data['features/density'], data['features/chlorides'], c=data['quality'])
legend1 = ax.legend(*scatter.legend_elements(), title="Classes")
ax.add_artist(legend1)
ax.set_xlabel('features/density')
ax.set_ylabel('features/chlorides')
ax.set_title('Качество вина')
```

```
Text(0.5, 1.0, 'Качество вина')
```



3. Если признак с метками классов содержит более двух классов, то объедините некоторые классы, чтобы получить набор для бинарной классификации. Объединяйте классы таким образом, чтобы положительный и отрицательный классы были сопоставимы по количеству точек.

#выведем размеры классов

```
for c in np.sort(data['quality'].unique()):
    print(c, len(data[data['quality'] == c]))
```

```
3 19
4 160
5 1409
6 2149
7 878
8 173
9 5
```

#объединим классы

```
y_new = data['quality'].copy()
```

```
y_new[y_new < 6] = 0
y_new[y_new >= 6] = 1
```

```
#новые размеры
for c in y_new.unique():
    print(c, len(y_new[y_new == c]))
```

```
0 1588
1 3205
```

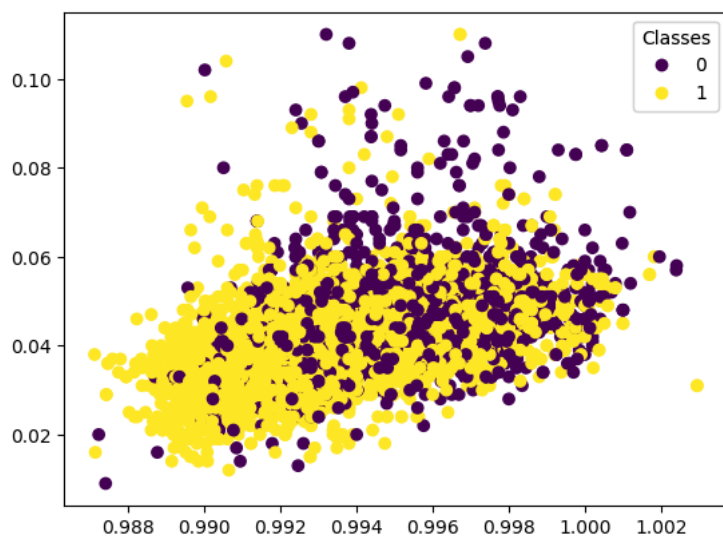
```
data.head()
```

	features/alcohol	features/chlorides	features/citric acid	features/density	features, a
0	9.0	0.054	0.34	1.00080	
1	12.2	0.063	0.49	0.99110	
2	11.2	0.029	0.11	0.99076	
3	9.0	0.110	0.27	0.99672	
4	12.0	0.035	0.30	0.99016	



```
fig, ax = plt.subplots()
scatter = ax.scatter(data['features/density'], data['features/chlorides'], c=y_new)
legend1 = ax.legend(*scatter.legend_elements(), title="Classes")
ax.add_artist(legend1)
```

<matplotlib.legend.Legend at 0x7f55f58c22f0>



- Разбейте набор данных из двух признаков и меток класса на обучающую и тестовую выборки. Постройте нейронную сеть с нормализующим слоем и параметрами, указанными в индивидуальном задании, для бинарной классификации и обучите ее на обучающей выборке. Оцените качество бинарной классификации при помощи матрицы ошибок для тестовой выборки.

- Параметры глубокой нейронной сети: кол-во скрытых слоев – 3, кол-во нейронов в скрытом слое – 128.

```
n = len(data)
train, test = data[:int(0.8*n)], data[int(0.8*n):]
y_train, y_test = y_new[:int(0.8*n)], y_new[int(0.8*n):]
```

```
X_train = train[['features/density', 'features/chlorides']]
X_test = test[['features/density', 'features/chlorides']]
```

```
feature_normalizer = tf.keras.layers.Normalization(axis=None, input_shape=(X_train.shape[1],))
feature_normalizer.adapt(X_train)
```

```
model = tf.keras.Sequential([
    feature_normalizer,
```

```

tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
Model: "sequential_56"

```

Layer (type)	Output Shape	Param #
normalization_32 (Normalization)	(None, 2)	3
dense_189 (Dense)	(None, 128)	384
dense_190 (Dense)	(None, 128)	16512
dense_191 (Dense)	(None, 128)	16512
dense_192 (Dense)	(None, 1)	129

```

Total params: 33,540
Trainable params: 33,537
Non-trainable params: 3

model.compile(loss= 'bce',
              optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01),
              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20)

Epoch 1/20
120/120 [=====] - 2s 3ms/step - loss: 0.6409 - accuracy: 0.6690
Epoch 2/20
120/120 [=====] - 0s 3ms/step - loss: 0.6354 - accuracy: 0.6716
Epoch 3/20
120/120 [=====] - 0s 3ms/step - loss: 0.6343 - accuracy: 0.6716
Epoch 4/20
120/120 [=====] - 0s 4ms/step - loss: 0.6336 - accuracy: 0.6716
Epoch 5/20
120/120 [=====] - 0s 3ms/step - loss: 0.6340 - accuracy: 0.6716
Epoch 6/20
120/120 [=====] - 0s 3ms/step - loss: 0.6344 - accuracy: 0.6716
Epoch 7/20
120/120 [=====] - 0s 2ms/step - loss: 0.6346 - accuracy: 0.6716
Epoch 8/20
120/120 [=====] - 0s 2ms/step - loss: 0.6340 - accuracy: 0.6716
Epoch 9/20
120/120 [=====] - 0s 2ms/step - loss: 0.6339 - accuracy: 0.6716
Epoch 10/20
120/120 [=====] - 0s 2ms/step - loss: 0.6338 - accuracy: 0.6716
Epoch 11/20
120/120 [=====] - 0s 2ms/step - loss: 0.6340 - accuracy: 0.6716
Epoch 12/20
120/120 [=====] - 0s 2ms/step - loss: 0.6342 - accuracy: 0.6716
Epoch 13/20
120/120 [=====] - 0s 2ms/step - loss: 0.6343 - accuracy: 0.6716
Epoch 14/20
120/120 [=====] - 0s 2ms/step - loss: 0.6336 - accuracy: 0.6716
Epoch 15/20
120/120 [=====] - 0s 2ms/step - loss: 0.6340 - accuracy: 0.6716
Epoch 16/20
120/120 [=====] - 0s 2ms/step - loss: 0.6340 - accuracy: 0.6716
Epoch 17/20
120/120 [=====] - 0s 2ms/step - loss: 0.6344 - accuracy: 0.6716
Epoch 18/20
120/120 [=====] - 0s 2ms/step - loss: 0.6336 - accuracy: 0.6716
Epoch 19/20
120/120 [=====] - 0s 2ms/step - loss: 0.6337 - accuracy: 0.6716
Epoch 20/20
120/120 [=====] - 0s 2ms/step - loss: 0.6333 - accuracy: 0.6716

y_pred = model.predict(X_test)

30/30 [=====] - 0s 1ms/step

y_pred[y_pred>0.5] = 1
y_pred[y_pred<=0.5] = 0

from sklearn.metrics import confusion_matrix as cm

```

```
cm(y_test, y_pred)

array([[ 0, 329],
       [ 0, 630]])
```

Да уж.....

5. Визуализируйте границы принятия решений построенной нейронной сетью на обучающей и тестовой выборках.

```
xmin = min(X_train['features/density'].min(), X_test['features/density'].min())
xmax = max(X_train['features/density'].max(), X_test['features/density'].max())
xdelta = abs(xmax - xmin) * 0.1
x = np.linspace(xmin-xdelta, xmax+xdelta, 50)

ymin = min(X_train['features/chlorides'].min(), X_test['features/chlorides'].min())
ymax = max(X_train['features/chlorides'].max(), X_test['features/chlorides'].max())
ydelta = abs(ymax - ymin) * 0.1
y = np.linspace(ymin-ydelta, ymax+ydelta, 50)

x, y = np.meshgrid(x, y)

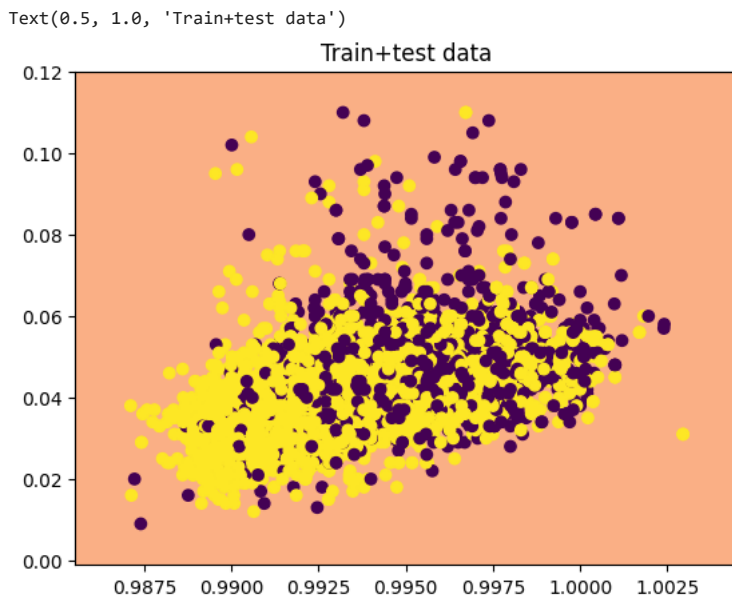
shape_ = x.shape

X_ = np.column_stack((x.ravel(), y.ravel()))

y_pred = np.round(model.predict(X_))
y_pred = y_pred.reshape(x.shape)

79/79 [=====] - 0s 1ms/step

plt.contourf(x, y, y_pred, cmap=plt.cm.RdYlBu, alpha=0.7)
plt.scatter(data['features/density'], data['features/chlorides'], c=y_new)
plt.title('Train+test data') #все равно на всех выборках одинаково...
```



6. Визуализируйте ROC-кривую для построенного классификатора и вычислите площадь под ROC-кривой методом трапеций или иным методом.

```
def true_false_positive(threshold_vector, y_test):
    true_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 1)
    true_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 0)
    false_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 0)
    false_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 1)

    tpr = true_positive.sum() / (true_positive.sum() + false_negative.sum())
    fpr = false_positive.sum() / (false_positive.sum() + true_negative.sum())

    return tpr, fpr
```

```
def roc_from_scratch(probabilities, y_test, partitions=100):
    roc = np.array([])
    for i in range(partitions + 1):
        threshold_vector = np.greater_equal(probabilities, i / partitions).astype(int)
        tpr, fpr = true_false_positive(threshold_vector, y_test)
        roc = np.append(roc, [fpr, tpr])

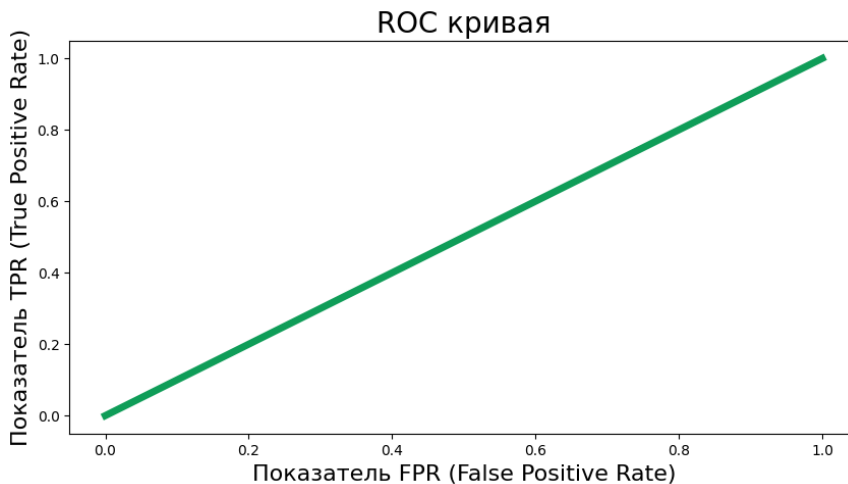
    return roc.reshape(-1, 2)
```

```
prediction = model.predict(X_test)
```

```
plt.figure(figsize=(10,5))
```

```
ROC = roc_from_scratch(prediction.reshape(-1), y_test, partitions=50)
#plt.scatter(ROC[:,0], ROC[:,1], color='#0F9D58', s=100)
plt.plot(ROC[:,0], ROC[:,1], color='#0F9D58', lw=5)
plt.title('ROC кривая', fontsize=20)
plt.xlabel('Показатель FPR (False Positive Rate)', fontsize=16)
plt.ylabel('Показатель TPR (True Positive Rate)', fontsize=16);
```

30/30 [=====] - 0s 1ms/step



```
xroc = ROC[:,0]
yroc = ROC[:,1]
```

```
yroc = yroc[np.argsort(xroc)]
xroc = np.sort(xroc)
sum = 0
cur = (0, 0)
for xi, yi in zip(xroc, yroc):
    sum += (xi - cur[0]) * (cur[1] + yi) / 2
    cur = (xi, yi)
print('AUC ROC = ', sum)
```

AUC ROC = 0.5

- Обучите на полном наборе данных нейронную сеть с одним слоем и одним выходным нейроном с функцией активации сигмоида и определите дополнительный признак, отличный от указанных в задании двух независимых признаков, принимающий непрерывные значения и являющийся важным по абсолютному значению веса в обученной нейронной сети.

```
X2 = data.drop(['quality'], axis=1)
```


```
normalizer2 = tf.keras.layers.Normalization(axis=None, input_shape=(X2.shape[1],))
normalizer2.adapt(X2)
```

```
model2 = tf.keras.Sequential([
    normalizer2,
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model2.compile(loss='bce')
model2.fit(X2, y2, callbacks=[keras.callbacks.History()])

imp = { 'feature' : np.array(X2.columns),
        'importance' : np.abs(np.ravel(model2.layers[1].kernel))}

pd.DataFrame(imp).sort_values(by=['importance'], ascending=False)
```

	feature	importance	
10	features/volatile acidity	0.948869	
4	features/fixed acidity	0.626269	
3	features/density	0.538984	
8	features/sulphates	0.411232	
5	features/free sulfur dioxide	0.302936	
9	features/total sulfur dioxide	0.302479	
6	features/pH	0.264806	
2	features/citric acid	0.241127	
7	features/residual sugar	0.151700	
1	features/chlorides	0.109739	
0	features/alcohol	0.006000	

```
data['features/volatile acidity'] #выбираем, т.к. features/chlorides уже выбран независимым
```

```
0      0.32
1      0.27
2      0.43
3      0.41
4      0.34
...
4893   0.16
4894   0.24
4895   0.36
4896   0.45
4897   0.36
Name: features/volatile acidity, Length: 4793, dtype: float32
```

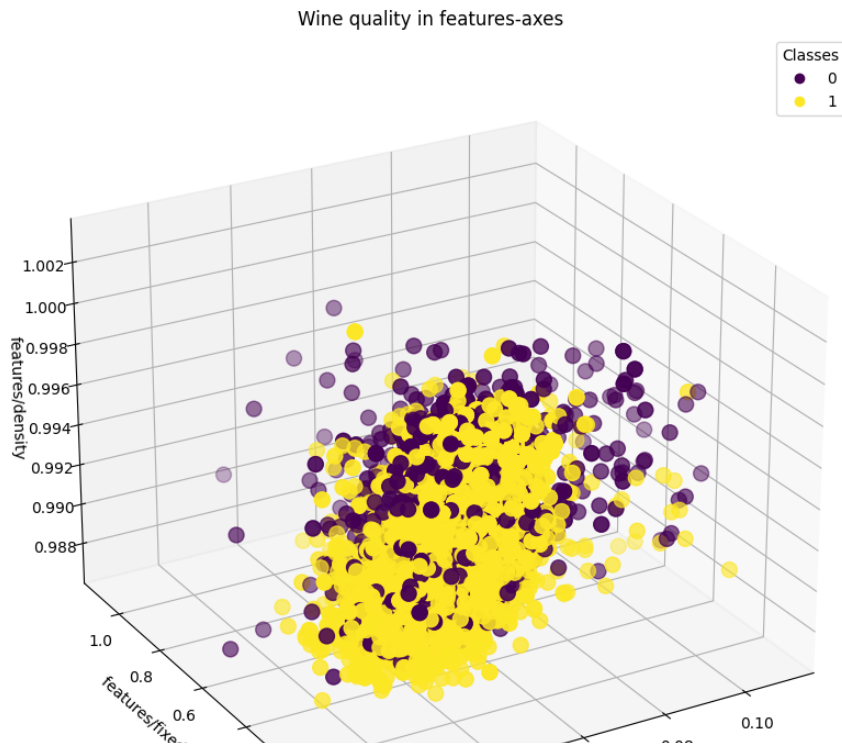
- Визуализируйте точки набора данных в трехмерном пространстве с координатами, соответствующими трем независимым признакам, отображая точки различных классов разными цветами. Подпишите оси и рисунок, создайте легенду для классов набора данных.

```
fig = plt.figure(figsize=(12,10))
ax = plt.axes(projection='3d')

zs = data['features/density']
xs = data['features/chlorides']
ys = data['features/volatile acidity']

scatter3d = ax.scatter( xs, ys, zs, s=100, c=y_new )
ax.set_zlabel('features/density')
ax.set_xlabel('features/chlorides')
ax.set_ylabel('features/fixed acidity')
ax.set_title('Wine quality in features-axes')
ax.legend(*scatter3d.legend_elements(), title="Classes")

ax.view_init( azim=-120, elev=25 )
```



9. Разбейте полный набор данных на обучающую и тестовую выборки. Постройте нейронную сеть с нормализующим слоем и параметрами, указанными в индивидуальном задании, для многоклассовой классификации и обучите ее на обучающей выборке.

```
X_train = train.drop(['quality'], axis=1)
X_test = test.drop(['quality'], axis=1)
y_train = train['quality']
y_test = test['quality']

y_train.unique()

array([5, 6, 4, 8, 7, 3, 9], dtype=int32)

def to_one_hot(labels):
    results = np.zeros((labels.shape[0], len(labels.unique())))
    for i, label in enumerate(labels):
        results[i, label-3] = 1.
    return results

y_train = to_one_hot(y_train)
y_test = to_one_hot(y_test)
y_train.shape, y_test.shape

((3834, 7), (959, 7))

feature_normalizer = tf.keras.layers.Normalization(axis=None, input_shape=(X_train.shape[1],))
feature_normalizer.adapt(X_train)

model3 = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(7, activation='sigmoid')
])

model3.compile(loss= 'categorical_crossentropy',
               optimizer = tf.keras.optimizers.Adam(learning_rate = 0.005),
               metrics=['accuracy'])

history = model3.fit(X_train, y_train, epochs=128,
                    validation_split = 0.2)
```



```

Epoch 101/128
96/96 [=====] - 0s 3ms/step - loss: 1.0219 - accuracy: 0.5399 - val_loss: 1.1394 - val_accuracy: 0.4863
Epoch 102/128
96/96 [=====] - 0s 3ms/step - loss: 1.0030 - accuracy: 0.5523 - val_loss: 1.1418 - val_accuracy: 0.5007
Epoch 103/128
96/96 [=====] - 0s 3ms/step - loss: 0.9909 - accuracy: 0.5566 - val_loss: 1.1725 - val_accuracy: 0.4850
Epoch 104/128
96/96 [=====] - 0s 3ms/step - loss: 0.9809 - accuracy: 0.5540 - val_loss: 1.2088 - val_accuracy: 0.4798
Epoch 105/128
96/96 [=====] - 0s 3ms/step - loss: 0.9970 - accuracy: 0.5549 - val_loss: 1.1486 - val_accuracy: 0.4876
Epoch 106/128
96/96 [=====] - 0s 3ms/step - loss: 0.9792 - accuracy: 0.5562 - val_loss: 1.2016 - val_accuracy: 0.4759
Epoch 107/128
96/96 [=====] - 0s 3ms/step - loss: 1.0050 - accuracy: 0.5497 - val_loss: 1.1977 - val_accuracy: 0.4707
Epoch 108/128
96/96 [=====] - 0s 3ms/step - loss: 0.9832 - accuracy: 0.5546 - val_loss: 1.1673 - val_accuracy: 0.4850
Epoch 109/128
96/96 [=====] - 0s 3ms/step - loss: 0.9863 - accuracy: 0.5546 - val_loss: 1.1312 - val_accuracy: 0.5124
Epoch 110/128
96/96 [=====] - 0s 3ms/step - loss: 0.9657 - accuracy: 0.5634 - val_loss: 1.1348 - val_accuracy: 0.4863
Epoch 111/128
96/96 [=====] - 0s 3ms/step - loss: 0.9721 - accuracy: 0.5579 - val_loss: 1.1638 - val_accuracy: 0.4941
Epoch 112/128
96/96 [=====] - 0s 3ms/step - loss: 0.9832 - accuracy: 0.5589 - val_loss: 1.1727 - val_accuracy: 0.4876
Epoch 113/128
96/96 [=====] - 0s 3ms/step - loss: 0.9729 - accuracy: 0.5611 - val_loss: 1.1362 - val_accuracy: 0.5007
Epoch 114/128
96/96 [=====] - 0s 3ms/step - loss: 0.9756 - accuracy: 0.5556 - val_loss: 1.1327 - val_accuracy: 0.4902
Epoch 115/128
96/96 [=====] - 0s 3ms/step - loss: 0.9701 - accuracy: 0.5706 - val_loss: 1.1586 - val_accuracy: 0.5228
Epoch 116/128
96/96 [=====] - 0s 3ms/step - loss: 0.9767 - accuracy: 0.5527 - val_loss: 1.1600 - val_accuracy: 0.4798
Epoch 117/128
96/96 [=====] - 0s 3ms/step - loss: 0.9889 - accuracy: 0.5572 - val_loss: 1.1432 - val_accuracy: 0.4967
Epoch 118/128
96/96 [=====] - 0s 3ms/step - loss: 0.9826 - accuracy: 0.5566 - val_loss: 1.1520 - val_accuracy: 0.4889
Epoch 119/128
96/96 [=====] - 0s 3ms/step - loss: 0.9678 - accuracy: 0.5608 - val_loss: 1.1485 - val_accuracy: 0.5085
Epoch 120/128
96/96 [=====] - 0s 3ms/step - loss: 0.9675 - accuracy: 0.5742 - val_loss: 1.1826 - val_accuracy: 0.4902
Epoch 121/128
96/96 [=====] - 0s 3ms/step - loss: 0.9546 - accuracy: 0.5690 - val_loss: 1.1519 - val_accuracy: 0.5098
Epoch 122/128
96/96 [=====] - 0s 3ms/step - loss: 0.9529 - accuracy: 0.5693 - val_loss: 1.1479 - val_accuracy: 0.5020
Epoch 123/128
96/96 [=====] - 0s 3ms/step - loss: 0.9474 - accuracy: 0.5813 - val_loss: 1.1516 - val_accuracy: 0.4811
Epoch 124/128
96/96 [=====] - 0s 3ms/step - loss: 0.9426 - accuracy: 0.5742 - val_loss: 1.2124 - val_accuracy: 0.4889
Epoch 125/128
96/96 [=====] - 0s 3ms/step - loss: 0.9467 - accuracy: 0.5771 - val_loss: 1.1879 - val_accuracy: 0.5046
Epoch 126/128
96/96 [=====] - 0s 3ms/step - loss: 0.9806 - accuracy: 0.5673 - val_loss: 1.1856 - val_accuracy: 0.4954
Epoch 127/128
96/96 [=====] - 0s 3ms/step - loss: 0.9425 - accuracy: 0.5791 - val_loss: 1.1471 - val_accuracy: 0.5046
Epoch 128/128
96/96 [=====] - 0s 3ms/step - loss: 0.9566 - accuracy: 0.5608 - val_loss: 1.2297 - val_accuracy: 0.4811

```

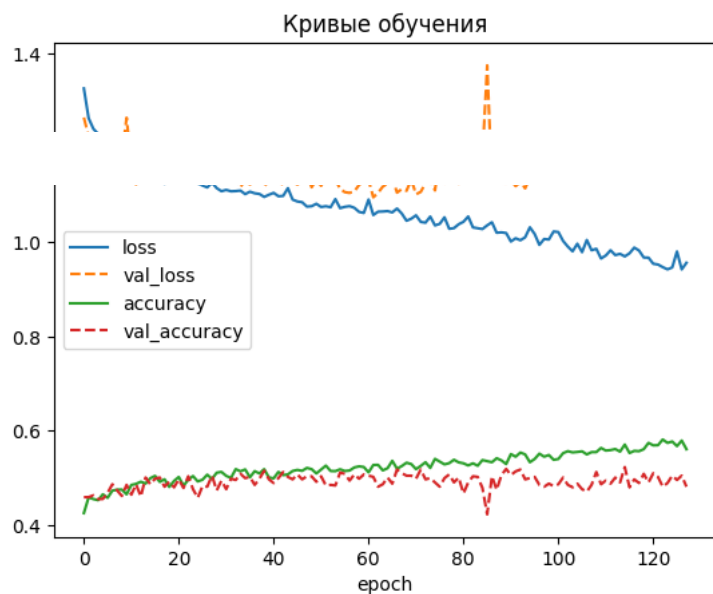
10. Постройте кривые обучения в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

```

plt.plot(range(len(history.history['loss'])), history.history['loss'], label='loss')
plt.plot(range(len(history.history['val_loss'])), history.history['val_loss'], '--', label='val_loss')
plt.plot(range(len(history.history['accuracy'])), history.history['accuracy'], label='accuracy')
plt.plot(range(len(history.history['val_accuracy'])), history.history['val_accuracy'], '--', label='val_accuracy')
plt.xlabel('epoch')
plt.title('Кривые обучения')
plt.legend()

```

<matplotlib.legend.Legend at 0x7f55e182e2c0>



[Платные продукты Colab](#) - [Отменить подписку](#)

✓ 1 сек. выполнено в 19:32

