РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

Дисциплина: Методы машинного обучения

Москва 2022

Вариант № 14

1. Загрузите заданный в индивидуальном задании набор данных с изображениями из Tensorflow Datasets с разбиением на обучающую и тестовую выборки.

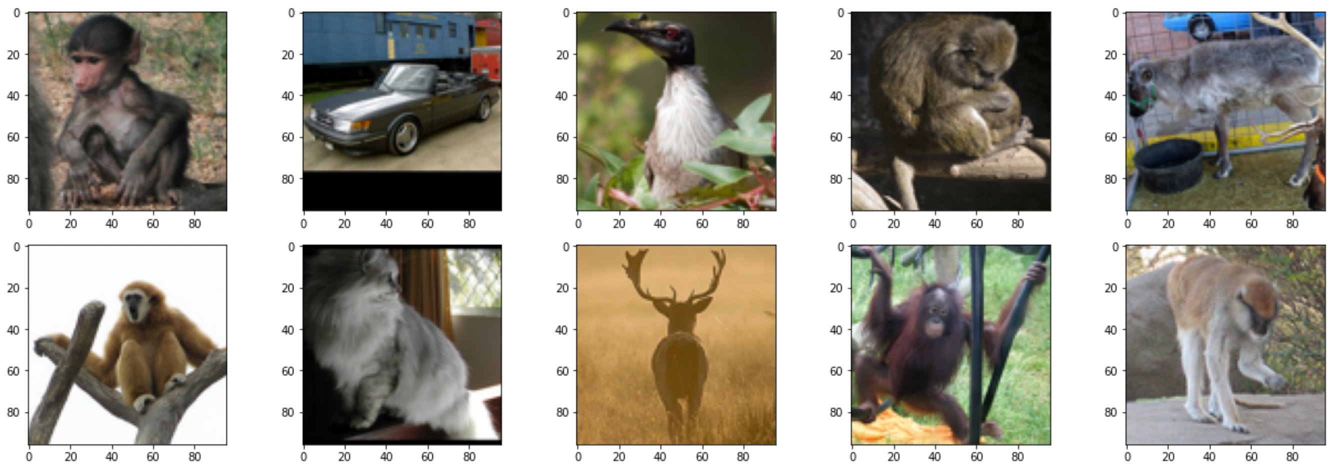
Ввод [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import tensorflow as tf
import tensorflow_datasets as tfds
df_train = tfds.as_dataframe(tfds.load("stl10", split=['train','test'])[0])
df_test  = tfds.as_dataframe(tfds.load("stl10", split=['train','test'])[1])
```

```
Downloading and preparing dataset stl10/1.0.0 (download: 2.46 GiB, generated: 1.86 Gi
B, total: 4.32 GiB) to /root/tensorflow_datasets/stl10/1.0.0...

Dl Completed...: 0 url [00:00, ? url/s]

Dl Size...: 0 MiB [00:00, ? MiB/s]

Extraction completed...: 0 file [00:00, ? file/s]




0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/stl10/1.0.0.incomplete3H4D
VI/stl10-train.tfrecord

  0%|          | 0/5000 [00:00<?, ? examples/s]

0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/stl10/1.0.0.incomplete3H4D
VI/stl10-test.tfrecord

  0%|          | 0/8000 [00:00<?, ? examples/s]

0 examples [00:00, ? examples/s]

Shuffling and writing examples to /root/tensorflow_datasets/stl10/1.0.0.incomplete3H4D
VI/stl10-unlabelled.tfrecord

  0%|          | 0/100000 [00:00<?, ? examples/s]

Dataset stl10 downloaded and prepared to /root/tensorflow_datasets/stl10/1.0.0. Subseq
uent calls will reuse this data.
```

2. Визуализируйте несколько изображений, отобранных случайным образом из обучающей выборки.

Ввод [3]:

```
import random
from PIL import Image, ImageOps
train_labels = df_train['label'].to_numpy(dtype=np.float32)
test_labels = df_test['label'].to_numpy(dtype=np.float32)
train_images = np.zeros(shape=(df_train.shape[0],96,96,3), dtype=np.float32)
test_images  = np.zeros(shape=(df_test.shape[0],96,96,3), dtype=np.float32)
train_images = np.zeros(shape=(df_train.shape[0],96,96,3), dtype=np.float32)
test_images  = np.zeros(shape=(df_test.shape[0],96,96,3), dtype=np.float32)
for idx in range(train_labels.shape[0]):
    train_images[idx,:,:,:] = np.array(Image.fromarray(df_train.iloc[idx]['image']))
for idx in range(test_labels.shape[0]):
    test_images[idx,:,:,:] = np.array(Image.fromarray(df_test.iloc[idx]['image']))
train_images /= 255
test_images  /= 255
def plot_random_sample(images):
    n = 10
    imgs = random.sample(list(images), n)
    num_row = 2
    num_col = 5
    fig, axes = plt.subplots(num_row, num_col, figsize=(3.5 * num_col, 3 * num_row))
    for i in range(num_row * num_col):
        img = imgs[i]
        ax = axes[i // num_col, i % num_col]
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
plot_random_sample(test_images)
```



3. Оставьте в наборе изображения двух классов, указанных в индивидуальном задании первыми. Обучите нейронные сети MLP и CNN задаче бинарной классификации изображений (архитектура сетей по вашему усмотрению). Количество эпох обучения указано в индивидуальном задании.

Ввод [4]:
```
df_train1,df_test1 = [],[]
for i in df_train.values:
  if i[1] in (4,5):
    df_train1.append(i)
df_train = pd.DataFrame(df_train1)
for i in df_test.values:
  if i[1] in (4,5):
    df_test1.append(i)
df_test = pd.DataFrame(df_test1)
train_labels = df_train[1].to_numpy(dtype=np.float32)
test_labels = df_test[1].to_numpy(dtype=np.float32)
train_labels.shape, test_labels.shape
train_images = np.zeros(shape=(df_train.shape[0],96,96,3), dtype=np.float32)
test_images  = np.zeros(shape=(df_test.shape[0],96,96,3), dtype=np.float32)
train_images = np.zeros(shape=(df_train.shape[0],96,96,3), dtype=np.float32)
test_images  = np.zeros(shape=(df_test.shape[0],96,96,3), dtype=np.float32)
for idx in range(train_labels.shape[0]):
    train_images[idx,:,:,:] = np.array(Image.fromarray(df_train.iloc[idx][0]))
for idx in range(test_labels.shape[0]):
    test_images[idx,:,:,:] = np.array(Image.fromarray(df_test.iloc[idx][0]))
train_images /= 255
test_images  /= 255
train_labels1 = []
for i in train_labels:
  if i==4:
    train_labels1.append(0)
  else:
    train_labels1.append(1)
train_labels = train_labels1

test_labels1 = []
for i in test_labels:
  if i==4:
    test_labels1.append(0)
  else:
    test_labels1.append(1)
test_labels = test_labels1
np.unique(test_labels)
tf.random.set_seed(42)
model_1 = tf.keras.Sequential([tf.keras.layers.Input(shape=(96, 96, 3)),
                               tf.keras.layers.Flatten(),tf.keras.layers.Dense(128, act
model_1.compile(loss=tf.keras.losses.binary_crossentropy,
                optimizer=tf.keras.optimizers.Adam(),
                metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
history_1 = model_1.fit(train_images,np.array(train_labels),
                        epochs=20,
                        batch_size=256,
                        validation_data=(test_images, np.array(test_labels)))
```

```
Epoch 1/20
4/4 [==============================] - 3s 565ms/step - loss: 7.4788 - accuracy: 0.5030
- val_loss: 2.9873 - val_accuracy: 0.5000
Epoch 2/20
4/4 [==============================] - 1s 347ms/step - loss: 6.2322 - accuracy: 0.4870
- val_loss: 1.5987 - val_accuracy: 0.5000
Epoch 3/20
4/4 [==============================] - 1s 333ms/step - loss: 2.5050 - accuracy: 0.4870
- val_loss: 3.2543 - val_accuracy: 0.5006
Epoch 4/20
4/4 [==============================] - 1s 373ms/step - loss: 2.4025 - accuracy: 0.5330
- val_loss: 1.7485 - val_accuracy: 0.5081
Epoch 5/20
4/4 [==============================] - 1s 321ms/step - loss: 1.8519 - accuracy: 0.5320
```

```
- val_loss: 1.7445 - val_accuracy: 0.5131
Epoch 6/20
4/4 [==============================] - 1s 240ms/step - loss: 1.6975 - accuracy: 0.5330
- val_loss: 2.1869 - val_accuracy: 0.5188
Epoch 7/20
4/4 [==============================] - 1s 198ms/step - loss: 1.6601 - accuracy: 0.5490
- val_loss: 0.9653 - val_accuracy: 0.5738
Epoch 8/20
4/4 [==============================] - 1s 211ms/step - loss: 1.2129 - accuracy: 0.5670
- val_loss: 1.3679 - val_accuracy: 0.5356
Epoch 9/20
4/4 [==============================] - 1s 177ms/step - loss: 1.0047 - accuracy: 0.6010
- val_loss: 0.9002 - val_accuracy: 0.5894
Epoch 10/20
4/4 [==============================] - 1s 197ms/step - loss: 0.8245 - accuracy: 0.6190
- val_loss: 0.8454 - val_accuracy: 0.5931
Epoch 11/20
4/4 [==============================] - 1s 208ms/step - loss: 0.8282 - accuracy: 0.6020
- val_loss: 0.9700 - val_accuracy: 0.5663
Epoch 12/20
4/4 [==============================] - 1s 181ms/step - loss: 0.7750 - accuracy: 0.6260
- val_loss: 0.8329 - val_accuracy: 0.5725
Epoch 13/20
4/4 [==============================] - 1s 213ms/step - loss: 0.6899 - accuracy: 0.6320
- val_loss: 0.8738 - val_accuracy: 0.5669
Epoch 14/20
4/4 [==============================] - 1s 205ms/step - loss: 0.7214 - accuracy: 0.6190
- val_loss: 0.7654 - val_accuracy: 0.5925
Epoch 15/20
4/4 [==============================] - 1s 177ms/step - loss: 0.7555 - accuracy: 0.6080
- val_loss: 0.7756 - val_accuracy: 0.5819
Epoch 16/20
4/4 [==============================] - 1s 211ms/step - loss: 0.7975 - accuracy: 0.6130
- val_loss: 0.9420 - val_accuracy: 0.5575
Epoch 17/20
4/4 [==============================] - 1s 218ms/step - loss: 0.8600 - accuracy: 0.5660
- val_loss: 1.0348 - val_accuracy: 0.5431
Epoch 18/20
4/4 [==============================] - 1s 210ms/step - loss: 0.8106 - accuracy: 0.5790
- val_loss: 0.8189 - val_accuracy: 0.5850
Epoch 19/20
4/4 [==============================] - 1s 182ms/step - loss: 0.7425 - accuracy: 0.6180
- val_loss: 0.8499 - val_accuracy: 0.5756
Epoch 20/20
4/4 [==============================] - 1s 177ms/step - loss: 0.7758 - accuracy: 0.6010
- val_loss: 0.7402 - val_accuracy: 0.6169
```

Ввод [5]:
```
tf.random.set_seed(42)
model_2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), input_shape=(96, 96, 3),
                           activation='relu'),tf.keras.layers.MaxPool2D(pool_size=(2, 2)
])
model_2.compile(loss=tf.keras.losses.binary_crossentropy,
                optimizer=tf.keras.optimizers.Adam(),
                metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)
history_2 = model_2.fit(train_images,np.array(train_labels),
                        epochs=20,
                        batch_size=256,
                        validation_data=(test_images, np.array(test_labels)))
```

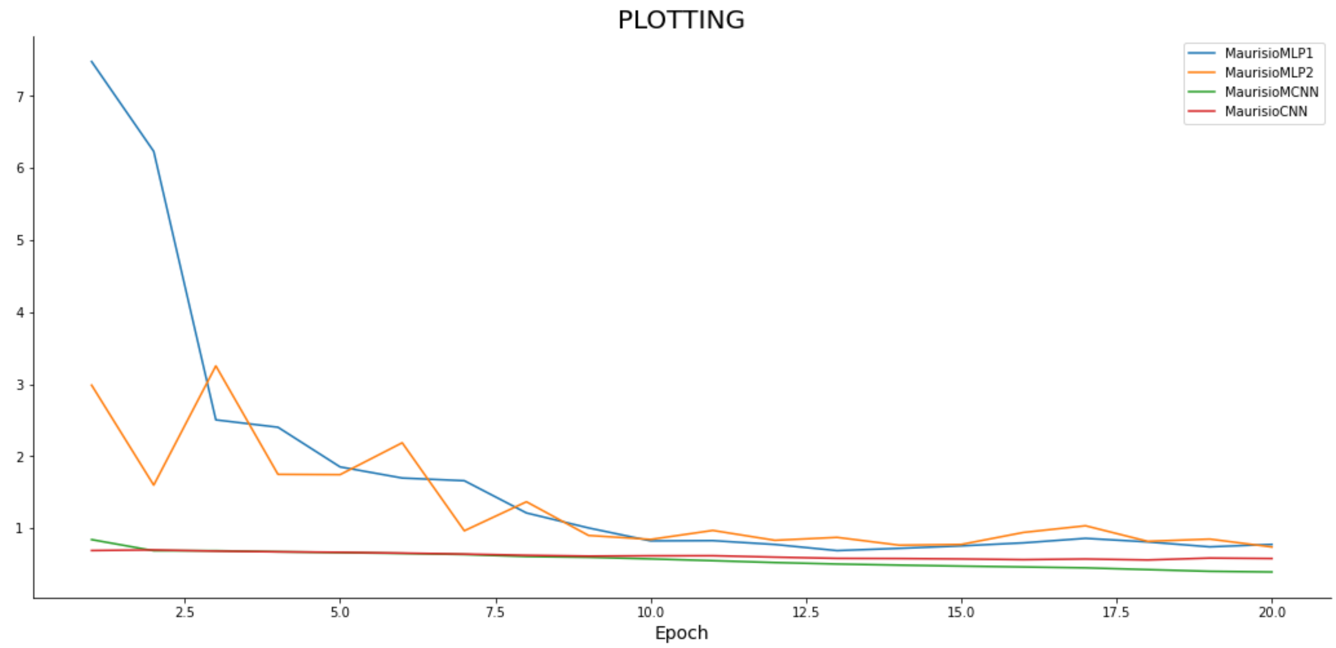
```
Epoch 1/20
4/4 [==============================] - 9s 2s/step - loss: 0.8427 - accuracy: 0.5110 -
val_loss: 0.6911 - val_accuracy: 0.5050
Epoch 2/20
4/4 [==============================] - 8s 2s/step - loss: 0.6884 - accuracy: 0.5190 -
val_loss: 0.6994 - val_accuracy: 0.5000
Epoch 3/20
4/4 [==============================] - 8s 2s/step - loss: 0.6884 - accuracy: 0.5000 -
val_loss: 0.6828 - val_accuracy: 0.5000
Epoch 4/20
4/4 [==============================] - 10s 3s/step - loss: 0.6742 - accuracy: 0.5110 -
val_loss: 0.6735 - val_accuracy: 0.5381
Epoch 5/20
4/4 [==============================] - 9s 3s/step - loss: 0.6624 - accuracy: 0.5310 -
val_loss: 0.6659 - val_accuracy: 0.5756
Epoch 6/20
4/4 [==============================] - 8s 2s/step - loss: 0.6506 - accuracy: 0.6050 -
val_loss: 0.6554 - val_accuracy: 0.6600
Epoch 7/20
4/4 [==============================] - 8s 2s/step - loss: 0.6356 - accuracy: 0.7040 -
val_loss: 0.6419 - val_accuracy: 0.6612
Epoch 8/20
4/4 [==============================] - 8s 2s/step - loss: 0.6077 - accuracy: 0.7170 -
val_loss: 0.6254 - val_accuracy: 0.6650
Epoch 9/20
4/4 [==============================] - 8s 2s/step - loss: 0.5969 - accuracy: 0.7050 -
val_loss: 0.6132 - val_accuracy: 0.6750
Epoch 10/20
4/4 [==============================] - 8s 2s/step - loss: 0.5747 - accuracy: 0.7240 -
val_loss: 0.6179 - val_accuracy: 0.6687
Epoch 11/20
4/4 [==============================] - 8s 2s/step - loss: 0.5501 - accuracy: 0.7360 -
val_loss: 0.6189 - val_accuracy: 0.6369
Epoch 12/20
4/4 [==============================] - 8s 2s/step - loss: 0.5240 - accuracy: 0.7770 -
val_loss: 0.5993 - val_accuracy: 0.6925
Epoch 13/20
4/4 [==============================] - 8s 2s/step - loss: 0.5038 - accuracy: 0.7690 -
val_loss: 0.5821 - val_accuracy: 0.6919
Epoch 14/20
4/4 [==============================] - 8s 2s/step - loss: 0.4883 - accuracy: 0.7880 -
val_loss: 0.5799 - val_accuracy: 0.6919
Epoch 15/20
4/4 [==============================] - 8s 2s/step - loss: 0.4747 - accuracy: 0.8020 -
val_loss: 0.5740 - val_accuracy: 0.7019
Epoch 16/20
4/4 [==============================] - 8s 2s/step - loss: 0.4628 - accuracy: 0.8020 -
val_loss: 0.5645 - val_accuracy: 0.7138
Epoch 17/20
4/4 [==============================] - 8s 2s/step - loss: 0.4507 - accuracy: 0.8170 -
```

```
val_loss: 0.5735 - val_accuracy: 0.7163
Epoch 18/20
4/4 [==============================] - 8s 2s/step - loss: 0.4261 - accuracy: 0.8290 -
val_loss: 0.5601 - val_accuracy: 0.7269
Epoch 19/20
4/4 [==============================] - 8s 2s/step - loss: 0.4026 - accuracy: 0.8500 -
val_loss: 0.5865 - val_accuracy: 0.6956
Epoch 20/20
4/4 [==============================] - 8s 2s/step - loss: 0.3931 - accuracy: 0.8370 -
val_loss: 0.5800 - val_accuracy: 0.7044
```

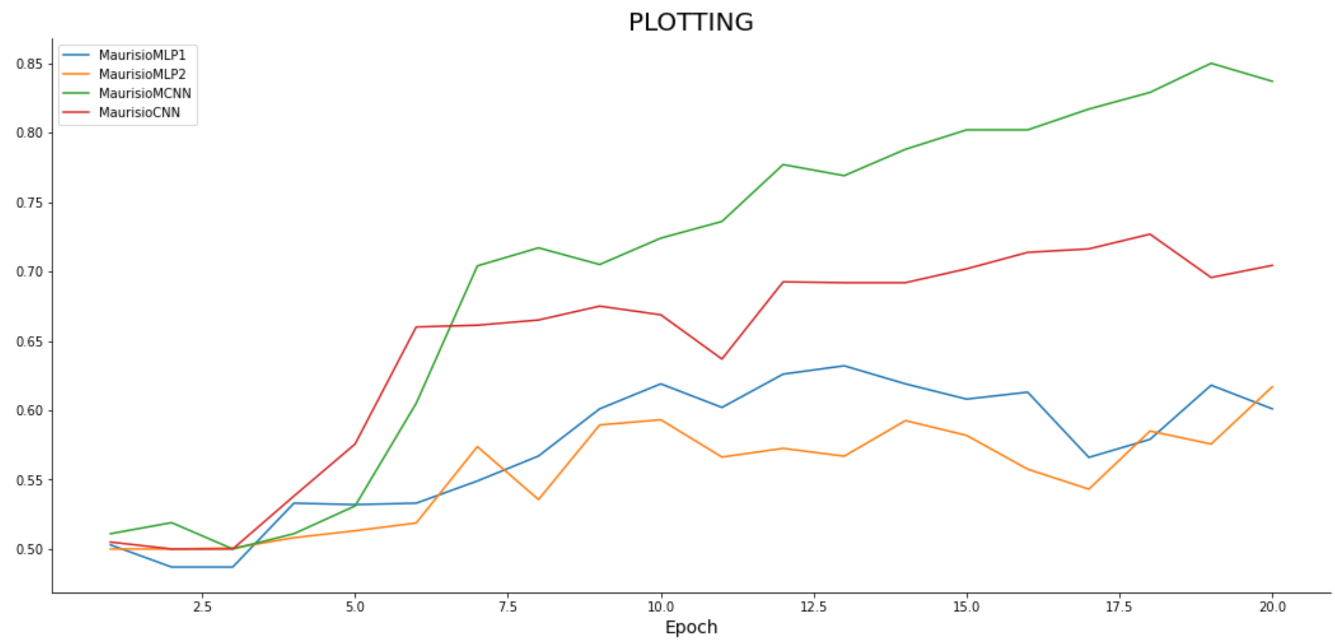
4. Постройте кривые обучения нейронных сетей для показателей ошибки и аккуратности в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

Ввод [6]:

```python
from matplotlib import rcParams
rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
plt.plot(np.arange(1, 21), history_1.history['loss'], label='MaurisioMLP1')
plt.plot(np.arange(1, 21), history_1.history['val_loss'], label='MaurisioMLP2')
plt.plot(np.arange(1, 21), history_2.history['loss'], label='MaurisioMCNN')
plt.plot(np.arange(1, 21), history_2.history['val_loss'], label='MaurisioCNN')
plt.title('PLOTTING', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```



Ввод [7]:
```python
from matplotlib import rcParams
rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
plt.plot(np.arange(1, 21), history_1.history['accuracy'], label='MaurisioMLP1')
plt.plot(np.arange(1, 21), history_1.history['val_accuracy'], label='MaurisioMLP2')
plt.plot(np.arange(1, 21), history_2.history['accuracy'], label='MaurisioMCNN')
plt.plot(np.arange(1, 21), history_2.history['val_accuracy'], label='MaurisioCNN')
plt.title('PLOTTING', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```



5. Сравните качество бинарной классификации нейронными сетями при помощи матрицы ошибок для тестовой выборки.

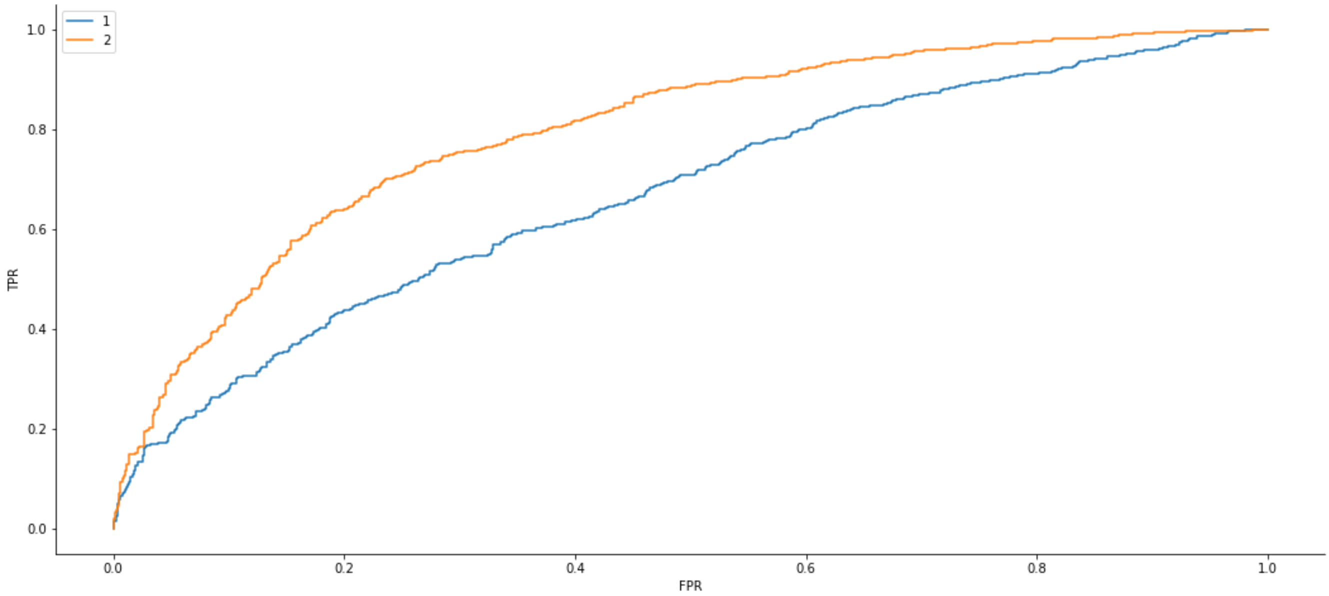
Ввод [8]:
```python
print (confusion_matrix(test_labels, np.round(abs(model_1.predict(test_images)))))
confusion_matrix(test_labels, np.round(abs(model_2.predict(test_images))))
```

```
[[650 150]
 [463 337]]
```

Out[8]:
```
array([[677, 123],
       [350, 450]])
```

6. Визуализируйте ROC-кривые для построенных классификаторов на одном рисунке (с легендой) и вычислите площади под ROC-кривыми.

Ввод [9]:
```
fpr, tpr, _ = metrics.roc_curve(test_labels, model_1.predict(test_images), pos_label=1)
fpr1, tpr1, _ = metrics.roc_curve(test_labels, model_2.predict(test_images), pos_label=1
plt.plot(fpr,tpr, label='1')
plt.plot(fpr1,tpr1, label='2')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.legend()
plt.show()
```



Ввод [10]:
```
auc = metrics.roc_auc_score(test_labels, model_1.predict(test_images))
print (auc)
auc = metrics.roc_auc_score(test_labels, model_2.predict(test_images))
auc
```

```
0.6685109375
```

Out[10]:
```
0.7947421875
```

7. Оставьте в наборе изображения трех классов, указанных в индивидуальном задании. Обучите нейронные сети MLP и CNN задаче многоклассовой классификации изображений (архитектура сетей по вашему усмотрению). Количество эпох обучения указано в индивидуальном задании.

Ввод [12]:

```
df_train = tfds.as_dataframe(tfds.load("stl10", split=['train','test'])[0])
df_test  = tfds.as_dataframe(tfds.load("stl10", split=['train','test'])[1])
df_train1 = []
for i in df_train.values:
  if i[1] in (4,5,6):
    df_train1.append(i)
df_train = pd.DataFrame(df_train1)
df_test1 = []
for i in df_test.values:
  if i[1] in (4,5,6):
    df_test1.append(i)
df_test = pd.DataFrame(df_test1)

train_labels = df_train[1].to_numpy(dtype=np.float32)
test_labels = df_test[1].to_numpy(dtype=np.float32)
train_labels.shape, test_labels.shape
train_images = np.zeros(shape=(df_train.shape[0],96,96,3), dtype=np.float32)
test_images  = np.zeros(shape=(df_test.shape[0],96,96,3), dtype=np.float32)
train_images.shape, test_images.shape
train_images = np.zeros(shape=(df_train.shape[0],96,96,3), dtype=np.float32)
test_images  = np.zeros(shape=(df_test.shape[0],96,96,3), dtype=np.float32)
train_images.shape, test_images.shape
for idx in range(train_labels.shape[0]):
    train_images[idx,:,:,:] = np.array(Image.fromarray(df_train.iloc[idx][0]))
for idx in range(test_labels.shape[0]):
    test_images[idx,:,:,:] = np.array(Image.fromarray(df_test.iloc[idx][0]))
train_images.shape, test_images.shape
train_images /= 255
test_images  /= 255

train_labels1 = []
for i in train_labels:
  if i==4:
    train_labels1.append(0)
  elif i==5:
    train_labels1.append(1)
  else:
    train_labels1.append(2)
train_labels = train_labels1

test_labels1 = []
for i in test_labels:
  if i==4:
    test_labels1.append(0)
  elif i==5:
    test_labels1.append(1)
  else:
    test_labels1.append(2)
test_labels = test_labels1
```

Ввод [13]:
```
def to_one_hot(labels, dimension=11):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
train_labels = to_one_hot(train_labels, 3)
test_labels = to_one_hot(test_labels,3)

tf.random.set_seed(42)
model_1 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(96, 96, 3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')])
model_1.compile(loss=tf.keras.losses.categorical_crossentropy,
                optimizer=tf.keras.optimizers.Adam(),
                metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
history_1 = model_1.fit(train_images,
                        train_labels,epochs=20,
                        batch_size=256,
                        validation_data=(test_images, test_labels))
```

```
Epoch 1/20
6/6 [==============================] - 4s 392ms/step - loss: 15.8309 - accuracy: 0.564
7 - val_loss: 12.1657 - val_accuracy: 0.5556
Epoch 2/20
6/6 [==============================] - 1s 260ms/step - loss: 7.2502 - accuracy: 0.5693
- val_loss: 4.1874 - val_accuracy: 0.5617
Epoch 3/20
6/6 [==============================] - 2s 314ms/step - loss: 2.3687 - accuracy: 0.5809
- val_loss: 1.0987 - val_accuracy: 0.6575
Epoch 4/20
6/6 [==============================] - 2s 283ms/step - loss: 1.2726 - accuracy: 0.6120
- val_loss: 1.0930 - val_accuracy: 0.6603
Epoch 5/20
6/6 [==============================] - 2s 285ms/step - loss: 1.1802 - accuracy: 0.6427
- val_loss: 1.0880 - val_accuracy: 0.6674
Epoch 6/20
6/6 [==============================] - 2s 316ms/step - loss: 1.1021 - accuracy: 0.6624
- val_loss: 1.0845 - val_accuracy: 0.6640
Epoch 7/20
6/6 [==============================] - 1s 234ms/step - loss: 1.0719 - accuracy: 0.6702
- val_loss: 1.0799 - val_accuracy: 0.6679
Epoch 8/20
6/6 [==============================] - 1s 194ms/step - loss: 1.0636 - accuracy: 0.6691
- val_loss: 1.0729 - val_accuracy: 0.6653
Epoch 9/20
6/6 [==============================] - 1s 181ms/step - loss: 1.0603 - accuracy: 0.6702
- val_loss: 1.0759 - val_accuracy: 0.6654
Epoch 10/20
6/6 [==============================] - 1s 183ms/step - loss: 1.0469 - accuracy: 0.6718
- val_loss: 1.0709 - val_accuracy: 0.6696
Epoch 11/20
6/6 [==============================] - 1s 181ms/step - loss: 1.0380 - accuracy: 0.6742
- val_loss: 1.0711 - val_accuracy: 0.6676
Epoch 12/20
6/6 [==============================] - 1s 181ms/step - loss: 1.0350 - accuracy: 0.6749
- val_loss: 1.0651 - val_accuracy: 0.6683
Epoch 13/20
6/6 [==============================] - 1s 183ms/step - loss: 1.0294 - accuracy: 0.6771
- val_loss: 1.0680 - val_accuracy: 0.6686
Epoch 14/20
6/6 [==============================] - 1s 168ms/step - loss: 1.0285 - accuracy: 0.6764
- val_loss: 1.0622 - val_accuracy: 0.6706
```

```
Epoch 15/20
6/6 [==============================] - 1s 185ms/step - loss: 1.0237 - accuracy: 0.6798
- val_loss: 1.0669 - val_accuracy: 0.6703
Epoch 16/20
6/6 [==============================] - 1s 189ms/step - loss: 1.0245 - accuracy: 0.6849
- val_loss: 1.0593 - val_accuracy: 0.6700
Epoch 17/20
6/6 [==============================] - 1s 184ms/step - loss: 1.0159 - accuracy: 0.6820
- val_loss: 1.0595 - val_accuracy: 0.6728
Epoch 18/20
6/6 [==============================] - 1s 192ms/step - loss: 1.0146 - accuracy: 0.6849
- val_loss: 1.0585 - val_accuracy: 0.6726
Epoch 19/20
6/6 [==============================] - 1s 180ms/step - loss: 1.0150 - accuracy: 0.6824
- val_loss: 1.0611 - val_accuracy: 0.6729
Epoch 20/20
6/6 [==============================] - 1s 180ms/step - loss: 1.0173 - accuracy: 0.6858
- val_loss: 1.0629 - val_accuracy: 0.6721
```

Ввод [14]:

```
tf.random.set_seed(42)
model_2 = tf.keras.Sequential([tf.keras.layers.Conv2D(filters=8,
                                                      kernel_size=(3, 3), input_shape=(9
model_2.compile(loss=tf.keras.losses.categorical_crossentropy
                ,optimizer=tf.keras.optimizers.Adam(),metrics=[tf.keras.metrics.BinaryAc
history_2 = model_2.fit(train_images,
                        train_labels,epochs=20,batch_size=256,validation_data=(test_imag
```

```
Epoch 1/20
6/6 [==============================] - 10s 2s/step - loss: 1.7912 - accuracy: 0.6220 -
val_loss: 1.1112 - val_accuracy: 0.6667
Epoch 2/20
6/6 [==============================] - 10s 2s/step - loss: 1.0989 - accuracy: 0.6669 -
val_loss: 1.0879 - val_accuracy: 0.6667
Epoch 3/20
6/6 [==============================] - 9s 2s/step - loss: 1.0771 - accuracy: 0.6671 -
val_loss: 1.0683 - val_accuracy: 0.6667
Epoch 4/20
6/6 [==============================] - 11s 2s/step - loss: 1.0586 - accuracy: 0.6704 -
val_loss: 1.0487 - val_accuracy: 0.6778
Epoch 5/20
6/6 [==============================] - 10s 2s/step - loss: 1.0374 - accuracy: 0.6842 -
val_loss: 1.0267 - val_accuracy: 0.6799
Epoch 6/20
6/6 [==============================] - 9s 2s/step - loss: 1.0234 - accuracy: 0.6860 -
val_loss: 1.0118 - val_accuracy: 0.6867
Epoch 7/20
6/6 [==============================] - 9s 2s/step - loss: 1.0079 - accuracy: 0.6942 -
val_loss: 1.0103 - val_accuracy: 0.6907
Epoch 8/20
6/6 [==============================] - 10s 2s/step - loss: 0.9906 - accuracy: 0.6971 -
val_loss: 0.9920 - val_accuracy: 0.6979
Epoch 9/20
6/6 [==============================] - 9s 2s/step - loss: 0.9766 - accuracy: 0.7033 -
val_loss: 0.9805 - val_accuracy: 0.7011
Epoch 10/20
6/6 [==============================] - 9s 2s/step - loss: 0.9567 - accuracy: 0.7100 -
val_loss: 0.9693 - val_accuracy: 0.7043
Epoch 11/20
6/6 [==============================] - 9s 2s/step - loss: 0.9401 - accuracy: 0.7182 -
val_loss: 0.9715 - val_accuracy: 0.7033
Epoch 12/20
6/6 [==============================] - 9s 2s/step - loss: 0.9285 - accuracy: 0.7171 -
val_loss: 0.9515 - val_accuracy: 0.7110
Epoch 13/20
6/6 [==============================] - 9s 2s/step - loss: 0.9080 - accuracy: 0.7260 -
val_loss: 0.9447 - val_accuracy: 0.7150
Epoch 14/20
6/6 [==============================] - 9s 2s/step - loss: 0.8903 - accuracy: 0.7342 -
val_loss: 0.9374 - val_accuracy: 0.7167
Epoch 15/20
6/6 [==============================] - 9s 2s/step - loss: 0.8838 - accuracy: 0.7322 -
val_loss: 0.9480 - val_accuracy: 0.7085
Epoch 16/20
6/6 [==============================] - 9s 2s/step - loss: 0.8837 - accuracy: 0.7342 -
val_loss: 0.9522 - val_accuracy: 0.7044
Epoch 17/20
6/6 [==============================] - 9s 2s/step - loss: 0.8593 - accuracy: 0.7364 -
val_loss: 0.9324 - val_accuracy: 0.7154
Epoch 18/20
6/6 [==============================] - 9s 2s/step - loss: 0.8371 - accuracy: 0.7467 -
val_loss: 0.9183 - val_accuracy: 0.7208
Epoch 19/20
6/6 [==============================] - 9s 2s/step - loss: 0.8236 - accuracy: 0.7513 -
```

```
val_loss: 0.9354 - val_accuracy: 0.7137
Epoch 20/20
6/6 [==============================] - 10s 2s/step - loss: 0.8151 - accuracy: 0.7478 -
val_loss: 0.9321 - val_accuracy: 0.7164
```

8. Сравните качество многоклассовой классификации нейронными сетями при помощи матрицы ошибок (для трех классов) для тестовой выборки.

Ввод [15]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(test_labels.argmax(axis=1), model_1.predict(test_images).argmax(axis=1)
```

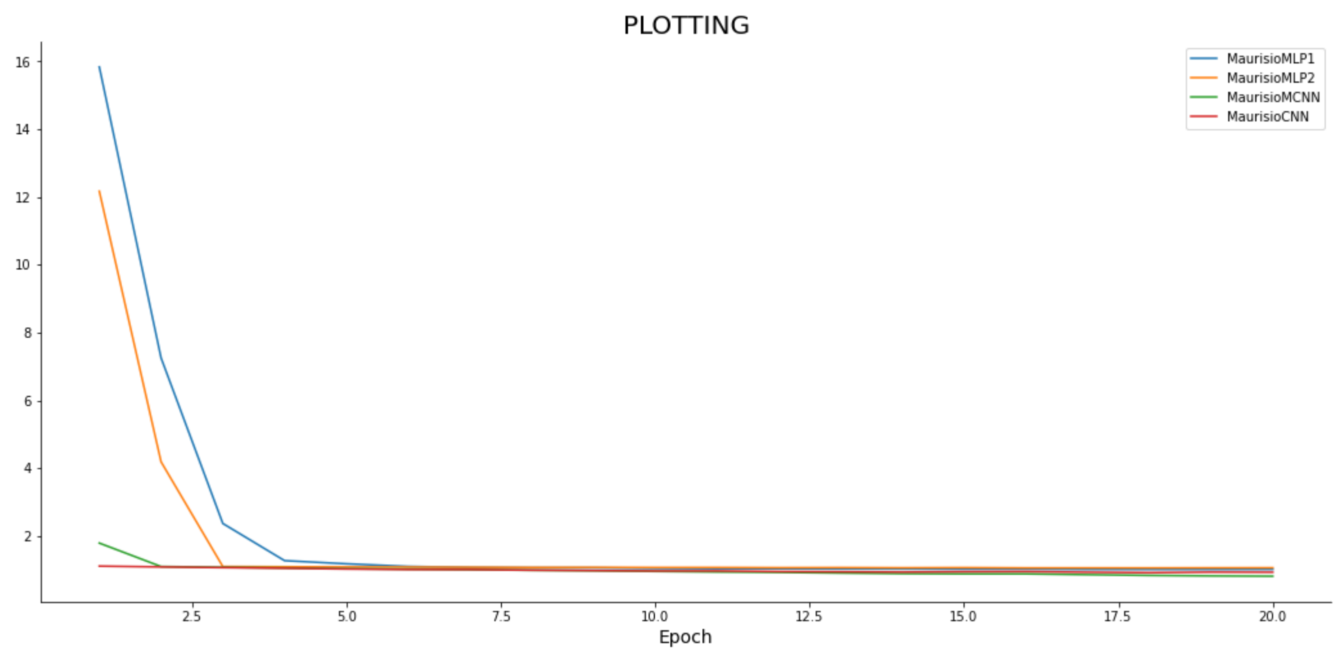
Out[15]:
```
array([[628,  98,  74],
       [509, 130, 161],
       [347, 142, 311]])
```

Ввод [16]:
```
confusion_matrix(test_labels.argmax(axis=1), model_2.predict(test_images).argmax(axis=1)
```

Out[16]:
```
array([[391, 235, 174],
       [174, 315, 311],
       [ 48, 136, 616]])
```

9. Постройте кривые обучения нейронных сетей для показателей ошибки и аккуратности в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

Ввод [17]:
```
from matplotlib import rcParams
rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
plt.plot(np.arange(1, 21), history_1.history['loss'], label='MaurisioMLP1')
plt.plot(np.arange(1, 21), history_1.history['val_loss'], label='MaurisioMLP2')
plt.plot(np.arange(1, 21), history_2.history['loss'], label='MaurisioMCNN')
plt.plot(np.arange(1, 21), history_2.history['val_loss'], label='MaurisioCNN')
plt.title('PLOTTING', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```



Ввод [18]:

```
from matplotlib import rcParams
rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
plt.plot(np.arange(1, 21), history_1.history['accuracy'], label='MaurisioMLP1')
plt.plot(np.arange(1, 21), history_1.history['val_accuracy'], label='MaurisioMLP2')
plt.plot(np.arange(1, 21), history_2.history['accuracy'], label='MaurisioMCNN')
plt.plot(np.arange(1, 21), history_2.history['val_accuracy'], label='MaurisioCNN')
plt.title('PLOTTING', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```