

HW5\_Petrov.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Reconnect

+ Code + Text

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра математического моделирования и искусственного интеллекта

▼ ОТЧЕТ ПО КОНТРОЛЬНОЙ РАБОТЕ № 5

Дисциплина: Методы машинного обучения

Студент: Петров Артем Евгеньевич

Группа: НКНбд-01-21

Москва 2024

---

Вариант 23

1. Набор данных deep\_weeds

2. Классы с метками 3,4,5

3. Требования к архитектуре сети MLP:

Кол-во скрытых слоев 6

Кол-во нейронов 50 в первом скрытом слое, увеличивающееся на 5 с каждым последующим скрытым слоем

Оптимизатор RMSprop

Функция активации в скрытых слоях swish

Регуляризация L1 в каждом нечетном скрытом слое

4. Требования к архитектуре сети CNN:

Кол-во сверточных слоев 3

Количество фильтров в сверточных слоях 16

Размеры фильтра 5x5

Оптимизатор Adam

Функция активации в сверточных слоях leaky\_relu

Функция активации в скрытых плотных слоях elu

Слой dropout после последнего слоя пулинга

5. Показатель качества бинарной классификации:

индекс Фоулка – Мэллоуса, равный корню квадратному из  $TP/(TP + TN) * TP/(TP + FP)$

6. Показатель качества многоклассовой классификации:

средняя полнота классов, где полнота (recall) класса равна доле правильных предсказаний для всех точек, принадлежащих этому классу.

---

Задание

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

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

3. Оставьте в наборе изображения двух классов, указанных в индивидуальном задании первыми. Обучите нейронные сети MLP и CNN задаче бинарной классификации изображений (требования к архитектуре сетей указаны в индивидуальном задании).  
Отследите обучение нейронных сетей и укажите, на сколько процентов снизились в результате обучения потери по отношению к потерям на первой эпохе обучения. Оцените результаты обучения нейронных сетей (варианты: нейронная сеть обучилась, недообучилась, переобучилась).

4. Постройте кривые обучения нейронных сетей бинарной классификации для показателей потерь и доли верных ответов в зависимости от эпохи обучения, подписывая оси и рисуя легенду.

5. Сравните качество бинарной классификации нейронными сетями при помощи показателя качества, указанного в индивидуальном задании.

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

7. Оставьте в наборе изображения трех классов, указанных в индивидуальном задании. Обучите нейронные сети MLP и CNN задаче многоклассовой классификации изображений (требования к архитектуре сетей указаны в индивидуальном задании).

8. Сравните качество многоклассовой классификации нейронными сетями при помощи показателя качества, указанного в индивидуальном задании.

9. Постройте кривые обучения нейронных сетей многоклассовой классификации для показателей ошибки и доли верных ответов в зависимости от эпохи обучения, подписывая оси и рисуя легенду. Сопроводите программный код необходимыми комментариями.

---

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

изображениями из Tensorflow Datasets с разбиением на обучающую и тестовую выборки.

## Набор данных deep\_weeds

```
[ ] !pip install -q tfds-nightly
!pip install tf-nightly-gpu
!pip install -U tensorflow_datasets

>Show hidden output

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

[ ] # ds_train = tfds.load('deep_weeds', split='train', shuffle_files=True)
# df = pd.read_csv("https://drive.usercontent.google.com/download?id=1xnK3B6K6KekDI55VwJ0vnc2IGoDga9cj&export=download&authuser=0")
# df.head()

[ ] import pathlib

[ ] dataset_url = 'https://drive.usercontent.google.com/download?id=1xnK3B6K6KekDI55VwJ0vnc2IGoDga9cj&export=download&authuser=0&confirm=t&uuid=a9621148-cb97-4374-b4b2-ba0c87e60728&at=APZUnTV00Fggt-jH0_K4oe

[ ] archive = tf.keras.utils.get_file(origin = dataset_url, extract = True)

[ ] archive = '/root/.keras/datasets'

[ ] data_dir = pathlib.Path(archive).with_suffix('')
data_dir

>Show PosixPath('/root/.keras/datasets')

[ ] image_count = len(list(data_dir.glob('*.*jpg')))
print(image_count)

>Show 17509

[ ] import PIL
import PIL.Image

[ ] tmp = data_dir.glob('*')
weeds = [str(x) for x in tmp if len(str(x).split('-'))>2]
weeds.sort()

[ ] # weed = list(data_dir.glob('*')).sort()
PIL.Image.open(str(weeds[0]))

>Show 
```

```
[ ] print(weeds[:5])
len(weeds)

>Show 17509 entries, 0 to 17508
[ ] import pandas as pd

[ ] df = pd.read_csv('https://raw.githubusercontent.com/AlexOlsen/DeepWeeds/master/labels/labels.csv')

[ ] df.head()
df['Label'] = df['Label'].apply(lambda x: int(x))
df.info()

>Show <class 'pandas.core.frame.DataFrame'>
RangeIndex: 17509 entries, 0 to 17508
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   Filename  17509 non-null  object  
 1   Label     17509 non-null  int64  
 2   Species   17509 non-null  object  
dtypes: int64(1), object(2)
memory usage: 418.5+ KB

[ ] df['Label'].values[0:5]
len(df['Label'].values)
print(df['Label'].value_counts())

>Show Label
8    9106
0    1125
6    1074
1    1064
4    1062
2    1031
3    1022
7    1016
5    1009
```

```
Name: count, dtype: int64
```

```
[ ] import os
import shutil

[ ] train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    str(data_dir),
    labels = list(df['Label'].values),
    color_mode = 'rgb',
    seed = 123,
    batch_size = 32,
    validation_split=0.2,
    subset = "training"
)
```

```
Found 17509 files belonging to 9 classes.
Using 14008 files for training.
```

```
[ ] val_ds = tf.keras.utils.image_dataset_from_directory(
    str(data_dir),
    labels = list(df['Label'].values),
    color_mode = 'rgb',
    seed = 123,
    batch_size = 32,
    validation_split=0.2,
    subset = "validation"
)
```

```
Found 17509 files belonging to 9 classes.
Using 3501 files for validation.
```

```
[ ] train_ds.class_names
```

```
Found 0, 1, 2, 3, 4, 5, 6, 7, 8
```

```
[ ] type(train_ds)
```

```
tensorflow.python.data.ops.prefetch_op._PrefetchDataset
def __init__(input_dataset, buffer_size, slack_period=None, name=None)
A `Dataset` that asynchronously prefetches its input.
```

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

```
[ ] import numpy as np

[ ] # train_df = list(train_ds)

[ ] # del train_df

[ ] # train_data = [(example.numpy(), label.numpy()) for example, label in train_ds if len(example) < 10]

[ ] [(train_features, label_batch)] = train_ds.take(1)

[ ] # np.array(train_features[0]).shape
np.array(train_features[int(np.random.random()*10)]).shape
```

```
Found (256, 256, 3)
```

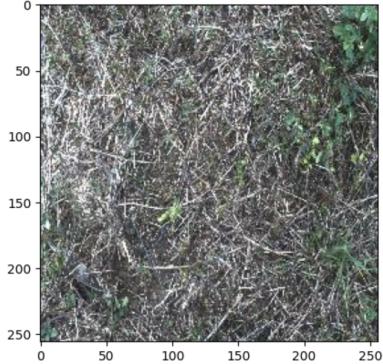
```
[ ] import matplotlib.pyplot as plt
```

```
[ ] from PIL import Image, ImageOps
```

```
[ ] img = Image.fromarray(np.array(train_features[0]).astype(np.uint8))
```

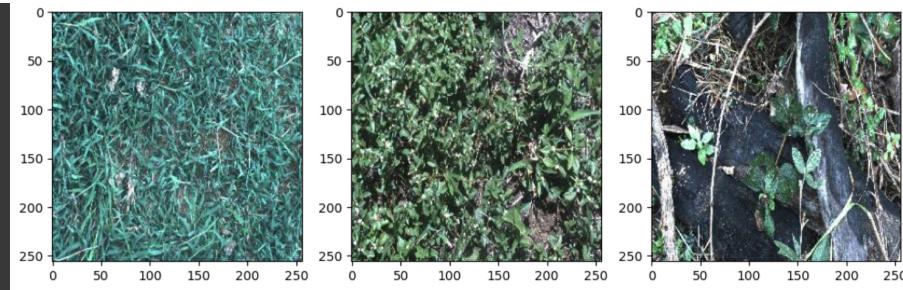
```
[ ] plt.imshow(img)
```

```
Found <matplotlib.image.AxesImage at 0x7f67dbeda00>
```



```
[ ] def take_some(train_features):
    _, ax = plt.subplots(1, 3, figsize = (12, 6))
    ax[0].imshow(np.array(train_features[int(np.random.random()*10)]).astype(np.uint8))
    ax[1].imshow(np.array(train_features[int(np.random.random()*10)]).astype(np.uint8))
    ax[2].imshow(np.array(train_features[int(np.random.random()*10)]).astype(np.uint8))

[ ] take_some(train_features)
```



3. Оставьте в наборе изображения двух классов, указанных в индивидуальном задании первыми. Обучите нейронные сети MLP и CNN задаче бинарной классификации изображений (требования к архитектуре сетей указаны в индивидуальном задании). Отследите обучение нейронных сетей и укажите, на сколько процентов снизились в результате обучения потери по отношению к потерям на первой эпохе обучения. Оцените результаты обучения нейронных сетей (варианты: нейронная сеть обучилась, недообучилась, переобучилась).

✓ Классы с метками 3,4,5

Требования к архитектуре сети MLP:

1. Кол-во скрытых слоев 6
2. Кол-во нейронов 50 в первом скрытом слое, увеличивающееся на 5 с каждым последующим скрытым слоем
3. Оптимизатор RMSprop
4. Функция активации в скрытых слоях swish
5. Регуляризация L1 в каждом нечетном скрытом слое

Требования к архитектуре сети CNN:

1. Кол-во сверточных слоев 3
2. Количество фильтров в сверточных слоях 16
3. Размеры фильтра 5x5
4. Оптимизатор Adam
5. Функция активации в сверточных слоях leaky\_relu
6. Функция активации в скрытых плотных слоях elu
7. Слой dropout после последнего слоя пулинга

Показатель качества бинарной классификации:

– Индекс Фоулка – Мэллоуса, равный корню квадратному из  $TP/(TP + TN) * TP/(TP + FP)$

Показатель качества многоклассовой классификации:

– Средняя полнота классов, где полнота (recall) класса равна доле правильных предсказаний для всех точек, принадлежащих этому классу.

```
[ ] images = list(data_dir.glob('*.jpg'))
len(images)

[ ] -----
NameError: Traceback (most recent call last)
<ipython-input-2-a338912df6f0> in <cell line: 1>()
----> 1 images = list(data_dir.glob('*.jpg'))
      2 len(images)

NameError: name 'data_dir' is not defined
```

```
[ ] data_dir

[ ] -----
NameError: Traceback (most recent call last)
<ipython-input-3-799f440d6e58> in <cell line: 1>()
----> 1 data_dir

NameError: name 'data_dir' is not defined
```

```
[ ] df.head()

[ ]   Filename  Label   Species
 0  20160928-140314-0.jpg     0 Chinee apple
 1  20160928-140337-0.jpg     0 Chinee apple
 2  20160928-140731-0.jpg     0 Chinee apple
 3  20160928-140747-0.jpg     0 Chinee apple
 4  20160928-141107-0.jpg     0 Chinee apple
```

```
[ ] df.info()
```

```


[ ] <class 'pandas.core.frame.DataFrame'>
RangeIndex: 17509 entries, 0 to 17508
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   Filename  17509 non-null   object  
 1   Label     17509 non-null   int64  
 2   Species   17509 non-null   object  
dtypes: int64(1), object(2)
memory usage: 418.5+ KB

[ ] images.sort()
images[:5]

[ ] [PosixPath('/root/.keras/datasets/20160928-140314-0.jpg'),
 PosixPath('/root/.keras/datasets/20160928-140337-0.jpg'),
 PosixPath('/root/.keras/datasets/20160928-140731-0.jpg'),
 PosixPath('/root/.keras/datasets/20160928-140747-0.jpg'),
 PosixPath('/root/.keras/datasets/20160928-141107-0.jpg')]

[ ] df_binary = df[(df['Label'] == 3) | (df['Label'] == 4)]
df_binary['Label'].value_counts()

[ ] Label
4    1062
3    1022
Name: count, dtype: int64

[ ] df_binary['Filename'] = df_binary['Filename'].apply(lambda x: str(data_dir) + '/' + x)

[ ] <ipython-input-22-013cc915c75c>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
df_binary['Filename'] = df_binary['Filename'].apply(lambda x: str(data_dir) + '/' + x)

[ ] df_binary.info()

[ ] <class 'pandas.core.frame.DataFrame'>
Index: 2084 entries, 2158 to 6587
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   Filename  2084 non-null   object  
 1   Label     2084 non-null   int64  
 2   Species   2084 non-null   object  
dtypes: int64(1), object(2)
memory usage: 65.1+ KB

[ ] new_dest = '/root/.keras/bina'
def move_chosen(paths, folder):
    for i in range(len(paths)):
        shutil.copy(paths[i], folder)

[ ] os.mkdir(new_dest, mode = 777)

[ ] 
[ ] -----[FileExistsError]-----[Traceback (most recent call last)]-----[FileExistsError]
<ipython-input-25-24d79bb5339e> in <cell line: 1>()
----> 1 os.mkdir(new_dest, mode = 777)

FileExistsError: [Errno 17] File exists: '/root/.keras/bina'

[ ] move_chosen(df_binary['Filename'].values, new_dest)

[ ] len(os.listdir(new_dest))

[ ] 2084

[ ] # data_dir = pathlib.Path(new_dest).with_suffix('')
# len(list(data_dir.glob('*.jpg')))

[ ] # df_binary.head()

[ ] train_bin = tf.keras.preprocessing.image_dataset_from_directory(
    str(new_dest),
    labels = list(df_binary['Label'].values),
    color_mode = 'rgb',
    seed = 123,
    batch_size = 64,
    validation_split=0.3,
    subset = "training",
    shuffle = True,
)


```

Found 2084 files belonging to 2 classes.  
Using 1459 files for training.

```


[ ] val_bin = tf.keras.utils.image_dataset_from_directory(
    str(new_dest),
    labels = list(df_binary['Label'].values),
    color_mode = 'rgb',
    seed = 123,
    batch_size = 64,
    validation_split=0.4,
    subset = "validation",
    shuffle = True
)


```

Found 2084 files belonging to 2 classes.  
Using 833 files for validation.

```


[ ] 


```

▼ MLP

```


[ ] # def TN(y_true, y_predict):


```

```

#     assert len(y_true) == len(y_predict)
#     return np.sum((y_true == 0) & (y_predict == 0))

# def FP(y_true, y_predict):
#     assert len(y_true) == len(y_predict)
#     return np.sum((y_true == 0) & (y_predict == 1))

# def FN(y_true, y_predict):
#     assert len(y_true) == len(y_predict)
#     return np.sum((y_true == 1) & (y_predict == 0))

# def TP(y_true, y_predict):
#     assert len(y_true) == len(y_predict)
#     return np.sum((y_true == 1) & (y_predict == 1))

```

Индекс Фоулка – Мэллоуса, равный корню квадратному из  $\text{TP}/(\text{TP} + \text{TN}) * \text{TP}/(\text{TP} + \text{FP})$

```
[ ] # def FMI(y_true, y_predict):
#     assert len(y_true) == len(y_predict)
#     return tf.sqrt((TP(y_true, y_predict)/(TP(y_true, y_predict) + TN(y_true, y_predict))) * (TP(y_true, y_predict)/(TP(y_true, y_predict) + FP(y_true, y_predict))))
```

```
[ ] tf.random.set_seed(51)
```

```
❶ mlp = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(256, 256, 3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(50, activation = 'swish', kernel_regularizer=tf.keras.regularizers.L1(0.01)),
    tf.keras.layers.Dense(55, activation = 'swish'),
    tf.keras.layers.Dense(60, activation = 'swish', kernel_regularizer=tf.keras.regularizers.L1(0.01)),
    tf.keras.layers.Dense(65, activation = 'swish'),
    tf.keras.layers.Dense(70, activation = 'swish', kernel_regularizer=tf.keras.regularizers.L1(0.01)),
    tf.keras.layers.Dense(75, activation = 'swish'),
    tf.keras.layers.Dense(1, activation = 'sigmoid'),
])
```

```
[ ] # def Fawkes_index(y_true, y_predict):
#     try:
#         return tf.sqrt(TP(y_true, y_predict)/(TP(y_true, y_predict) + TN(y_true, y_predict)) * TP(y_true, y_predict)/(TP(y_true, y_predict) + FP(y_true, y_predict)))
#     except:
#         return 0.0
```

```
[ ] # !pip install tensorflow-model-analysis
# import tfma
```

```
[ ] # import tensorflow_model_analysis as tfma
```

```
[ ] mlp.compile(
    loss = tf.keras.losses.binary_crossentropy,
    optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.01),
    # metrics = [tf.keras.metrics.FalseNegatives(name = 'FN')]
    metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)
```

```
[ ] # type(train_bin)
```

```
[ ] # train_lst = list(train_bin)
```

```
[ ] train_data = [(example.numpy().astype(np.uint8), label.numpy()) for example, label in train_bin]
```

```
[ ] train_data[0][0][0]
```

```
❷ ndarray (256, 256, 3) show data
```



```
[ ] # marks = [0 if x == 3 else 1 for x in train_data[0][1]]
# np.array(marks, dtype = np.int64)
```

```
❸ x_tmp = []
for i in range(len(train_data)):
    for j in range(len(train_data[i][0])):
        x_tmp.append(train_data[i][0][j]/255)
```

```
[ ] x_train = np.array(x_tmp)
```

```
[ ] # x_train[0]
```

```
[ ] y_tmp = []
for i in range(len(train_data)):
    for j in range(len(train_data[i][1])):
        y_tmp.append(0 if train_data[i][1][j] == 3 else 1)
```

```
[ ] y_train = np.array(y_tmp)
```

```
❹ mlp_hist = mlp.fit(
    x=x_train,
    y=y_train,
    verbose = 1,
    batch_size = 32,
    epochs=15
```

```
        epochs=15,
        validation_split = 0.2,
    )
```

Show hidden output

Вывод: MLP классификация не обучилась. Это связано с тем, что на фотографиях четко не выделены сами виды марихуаны, поэтому там зачастую присутствует другая зелень и т.п.

## CNN

1. Кол-во сверточных слоев 3
2. Количество фильтров в сверточных слоях 16
3. Размеры фильтра 5x5
4. Оптимизатор Adam
5. Функция активации в сверточных слоях leaky\_relu
6. Функция активации в скрытых плотных слоях elu
7. Слой dropout после последнего слоя пулинга

### ▼ Показатель качества бинарной классификации:

- Индекс Фоулка – Мэллоуса, равный корню квадратному из  $TP/(TP + TN) * TP/(TP + FP)$

```
[ ] tf.random.set_seed(41)

[ ] cnn = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5,5), input_shape=(256, 256,3), activation = 'leaky_relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2), padding = 'same'),
    tf.keras.layers.Conv2D(filters=16, kernel_sizes=(5,5), activation = 'leaky_relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2), padding = 'same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5,5), activation = 'leaky_relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2), padding = 'same'),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(256, activation = 'elu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid')
])

[ ] cnn.compile(
    loss = tf.keras.losses.binary_crossentropy,
    optimizer = tf.keras.optimizers.Adam(),
    metrics = [tf.keras.metrics.BinaryAccuracy(name = 'accuracy')]
)

[ ] cnn_hist = cnn.fit(
    x=x_train,
    y=y_train,
    verbose = 1,
    batch_size = 32,
    epochs=15,
    validation_split = 0.2,
)

→ Epoch 1/15
37/37 [=====] - 126s 3s/step - loss: 0.9335 - accuracy: 0.4944 - val_loss: 0.6726 - val_accuracy: 0.5890
Epoch 2/15
37/37 [=====] - 111s 3s/step - loss: 0.6859 - accuracy: 0.5621 - val_loss: 0.6617 - val_accuracy: 0.5719
Epoch 3/15
37/37 [=====] - 104s 3s/step - loss: 0.6250 - accuracy: 0.6298 - val_loss: 0.6583 - val_accuracy: 0.6027
Epoch 4/15
37/37 [=====] - 109s 3s/step - loss: 0.5646 - accuracy: 0.7129 - val_loss: 0.6395 - val_accuracy: 0.6575
Epoch 5/15
37/37 [=====] - 109s 3s/step - loss: 0.4864 - accuracy: 0.7823 - val_loss: 0.6228 - val_accuracy: 0.6747
Epoch 6/15
37/37 [=====] - 109s 3s/step - loss: 0.3880 - accuracy: 0.8286 - val_loss: 0.6498 - val_accuracy: 0.6678
Epoch 7/15
37/37 [=====] - 106s 3s/step - loss: 0.3100 - accuracy: 0.8749 - val_loss: 0.7617 - val_accuracy: 0.6233
Epoch 8/15
37/37 [=====] - 109s 3s/step - loss: 0.1913 - accuracy: 0.9366 - val_loss: 0.8964 - val_accuracy: 0.6884
Epoch 9/15
37/37 [=====] - 108s 3s/step - loss: 0.1598 - accuracy: 0.9469 - val_loss: 0.8618 - val_accuracy: 0.6849
Epoch 10/15
37/37 [=====] - 107s 3s/step - loss: 0.0861 - accuracy: 0.9743 - val_loss: 1.0067 - val_accuracy: 0.6541
Epoch 11/15
37/37 [=====] - 110s 3s/step - loss: 0.0525 - accuracy: 0.9846 - val_loss: 0.9468 - val_accuracy: 0.6336
Epoch 12/15
37/37 [=====] - 105s 3s/step - loss: 0.0341 - accuracy: 0.9931 - val_loss: 1.0179 - val_accuracy: 0.6438
Epoch 13/15
37/37 [=====] - 108s 3s/step - loss: 0.1259 - accuracy: 0.9512 - val_loss: 1.3093 - val_accuracy: 0.5445
Epoch 14/15
37/37 [=====] - 107s 3s/step - loss: 0.0977 - accuracy: 0.9657 - val_loss: 1.0005 - val_accuracy: 0.6096
Epoch 15/15
37/37 [=====] - 109s 3s/step - loss: 0.0683 - accuracy: 0.9811 - val_loss: 1.1404 - val_accuracy: 0.6575
```

Выводы: CNN слегка переобучилась, хотя точность составляет около 66%, что вполне достойно с учетом того, что картинки не очищены от другой зелени деревьев. Чтобы этого избежать, можно было бы применить какой-нибудь фильтр, чтобы понятнее выделить наши виды марихуаны на картинках.

4. Постройте кривые обучения нейронных сетей бинарной классификации для
- ▼ показателей потерь и доли верных ответов в зависимости от эпохи обучения, подписьвая оси и рисунок и создавая легенду.

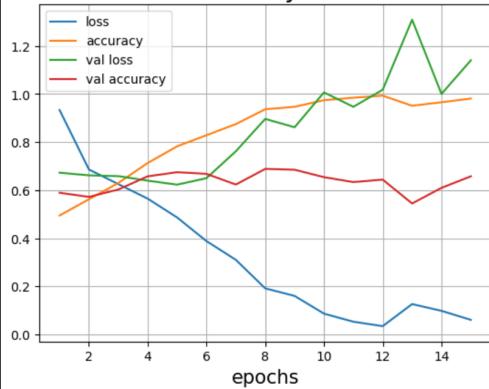
## ▼ CNN

Double-click (or enter) to edit

```
[ ] plt.plot(np.arange(1, 16), cnn_hist.history['loss'], label = 'loss')
plt.plot(np.arange(1, 16), cnn_hist.history['accuracy'], label = 'accuracy')
plt.plot(np.arange(1, 16), cnn_hist.history['val_loss'], label = 'val loss')
plt.plot(np.arange(1, 16), cnn_hist.history['val_accuracy'], label = 'val accuracy')
plt.title('Показатели обучения CNN', size = 20)
plt.xlabel('epochs', size = 15)
plt.grid(True)
plt.legend()
```

☒ <matplotlib.legend.Legend at 0x7f3234b83550>

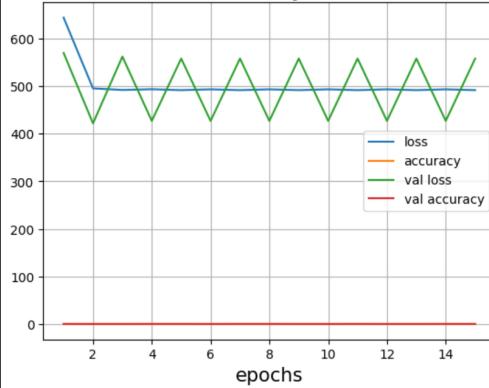
Показатели обучения CNN



```
[ ] plt.plot(np.arange(1, 16), mlp_hist.history['loss'], label = 'loss')
plt.plot(np.arange(1, 16), mlp_hist.history['accuracy'], label = 'accuracy')
plt.plot(np.arange(1, 16), mlp_hist.history['val_loss'], label = 'val loss')
plt.plot(np.arange(1, 16), mlp_hist.history['val_accuracy'], label = 'val accuracy')
plt.title('Показатели обучения MLP', size = 20)
plt.xlabel('epochs', size = 15)
plt.grid(True)
plt.legend()
```

☒ <matplotlib.legend.Legend at 0x7f323c34d630>

Показатели обучения MLP



5. Сравните качество бинарной классификации нейронными сетями при помощи показателя качества, указанного в индивидуальном задании.

В данном случае очевидно, что CNN выигрывает у MLP: 66% против 50%

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

```
[ ] def TN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 0))
```

```
[ ] def FP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 1))
```

```
[ ] def FN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 0))
```

```
[ ] def FP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 1))
```

```
[ ] def top_scores(y_true, y_predict):
```

```

[ ] tp = TP(y_true, y_predict)
fn = FN(y_true, y_predict)
try:
    return tp / (tp + fn)
except:
    return 0.0

def fpr_score(y_true, y_predict):
    fp = FP(y_true, y_predict)
    tn = TN(y_true, y_predict)
    try:
        return fp / (fp + tn)
    except:
        return 0.0

[ ] 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)

[ ] val_data = [(example.numpy().astype(np.uint8), label.numpy()) for example, label in val_bin]

[ ] x_tmp = []
for i in range(len(val_data)):
    for j in range(len(val_data[i][0])):
        x_tmp.append(val_data[i][0][j]/255)

[ ] val_x = np.array(x_tmp)

[ ] y_tmp = []
for i in range(len(val_data)):
    for j in range(len(val_data[i][1])):
        y_tmp.append(0 if val_data[i][1][j] == 3 else 1)

[ ] val_y = np.array(y_tmp)

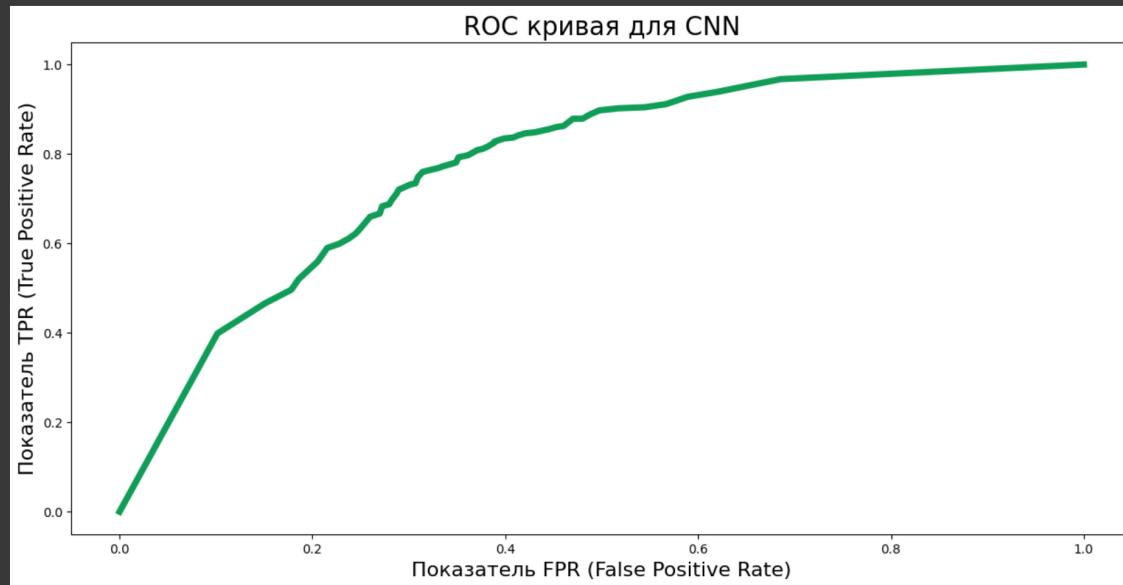
[ ] pred = cnn.predict(val_x)

Σ 27/27 [=====] - 20s 703ms/step

[ ] plt.figure(figsize = (15,7))

ROC = roc_from_scratch(pred.reshape(-1), val_y, partitions = 50)
plt.plot(ROC[:,0],ROC[:,1],color='#0F9D5B',lw=5)
plt.title('ROC кривая для CNN',fontsize=20)
plt.xlabel('Показатель FPR (False Positive Rate)',fontsize=16)
plt.ylabel('Показатель TPR (True Positive Rate)',fontsize=16);

```



Для MLP, как мне кажется, строить смысла нет, так как она у нас не обучилась совсем.

7. Оставьте в наборе изображения трех классов, указанных в индивидуальном задании. Обучите нейронные сети MLP и CNN задаче многоклассовой классификации изображений (требования к архитектуре сетей указаны в

индивидуальном задании).

```
[ ] images = list(data_dir.glob('*.*'))  
len(images)
```

17509

```
[ ] df.head()
```

	Filename	Label	Species
0	20160928-140314-0.jpg	0	Chinee apple
1	20160928-140337-0.jpg	0	Chinee apple
2	20160928-140731-0.jpg	0	Chinee apple
3	20160928-140747-0.jpg	0	Chinee apple
4	20160928-141107-0.jpg	0	Chinee apple

Next steps: [View recommended plots](#)

```
▶ df_triple = df[(df['Label'] == 3) | (df['Label'] == 4) | (df['Label'] == 5)]  
df_triple['Label'].value_counts()
```

```
Label  
4    1062  
3    1022  
5    1009  
Name: count, dtype: int64
```

```
[ ] df_triple['Filename'] = df_triple['Filename'].apply(lambda x: str(data_dir) + '/' + x)
```

```
☒ <ipython-input-25-bbea389a3365>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
df_triple['Filename'] = df_triple['Filename'].apply(lambda x: str(data_dir) + '/' + x)
```

```
[ ] df_triple.head()
```

	Filename	Label	Species
2150	/root/.keras/datasets/20170706-110255-0.jpg	4	Prickly acacia
3142	/root/.keras/datasets/20170727-093941-1.jpg	4	Prickly acacia
3144	/root/.keras/datasets/20170727-094049-1.jpg	4	Prickly acacia
3145	/root/.keras/datasets/20170727-111533-3.jpg	4	Prickly acacia
3146	/root/.keras/datasets/20170727-111716-3.jpg	4	Prickly acacia

Next steps: [View recommended plots](#)

```
[ ] new_dest = '/root/.keras/trip'
```

```
[ ] os.mkdir(new_dest, mode = 777)
```

```
☒ FileExistsError: Traceback (most recent call last)  
<ipython-input-28-24d79bb5339e> in <cell line: 1>()  
----> 1 os.mkdir(new_dest, mode = 777)  
  
FileExistsError: [Errno 17] File exists: '/root/.keras/trip'
```

```
[ ] move_chosen(df_triple['Filename'].values, new_dest)
```

```
☒ NameError: Traceback (most recent call last)  
<ipython-input-29-d93b51a18f74> in <cell line: 1>()  
----> 1 move_chosen(df_triple['Filename'].values, new_dest)  
  
NameError: name 'move_chosen' is not defined
```

```
[ ] df_triple['Label'] = df_triple['Label'].apply(lambda x: x-3)
```

```
☒ <ipython-input-30-9c1cc6553758>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
df_triple['Label'] = df_triple['Label'].apply(lambda x: x-3)
```

```
[ ] df_triple['Label'].value_counts()
```

```
Label  
1    1062  
0    1022  
2    1009  
Name: count, dtype: int64
```

```
[ ] def to_one_hot(labels, dimension=3):  
    # results = np.zeros((len(labels), dimension))  
    results = []  
    for i, label in enumerate(labels):  
        res = np.zeros(dimension)  
        res[label] = 1  
        results.append(list(res))  
    return results
```

```
▶ res = to_one_hot(df_triple['Label'].values)  
# list(list(res[:]))[:5]
```

```
[ ] train_y = np.array(res[:1238])
```

```
[ ] train_y.shape
[ ] (1547, 3)

[ ] val_y = np.array(res[2000:])

[ ] train_bin = tf.keras.preprocessing.image_dataset_from_directory(
    str(new_dest),
    labels = None,
    color_mode = 'rgb',
    seed = 0,
    batch_size = 32,
    validation_split = 0.6,
    subset = 'training',
    # shuffle = True,
)

```

Found 3093 files belonging to 1 classes.  
Using 1238 files for training.

```
[ ] # val_bin = tf.keras.preprocessing.image_dataset_from_directory(
#     str(new_dest),
#     labels = None,
#     color_mode = 'rgb',
#     seed = 123,
#     batch_size = 32,
#     validation_split = 0.3,
#     subset = 'validation',
#     shuffle = True,
# )

```

Found 3093 files belonging to 1 classes.  
Using 927 files for validation.

```
[ ] train_data = [example.numpy().astype(np.uint8) for example in train_bin]

[ ]
[ ] 32

[ ] # train_data = data[:1200]

[ ] x_tmp = []
for i in range(len(train_data)):
    for j in range(len(train_data[i])):
        x_tmp.append(train_data[i][j]/255)

[ ] train_x = np.array(x_tmp)

[ ] train_x.shape
[ ] (1238, 256, 256, 3)

[ ] val_data = data[2000:]

[ ] x_tmp1 = []
for i in range(len(val_data)):
    for j in range(len(val_data[i])):
        x_tmp1.append(val_data[i][j]/255)

[ ] val_x = np.array(x_tmp1)

[ ] cnn = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5,5), input_shape=(256, 256,3), activation = 'leaky_relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2), padding = 'same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5,5), activation = 'leaky_relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2), padding = 'same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5,5), activation = 'leaky_relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2), padding = 'same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(256, activation = 'elu'),
    tf.keras.layers.Dense(3, activation = 'softmax')
])

[ ] cnn.compile(
    loss = tf.keras.losses.categorical_crossentropy,
    optimizer = tf.keras.optimizers.Adam(),
    metrics = [tf.keras.metrics.Recall(name = 'recall')]
)

[ ] cnn_hist = cnn.fit (
    x=train_x,
    y=train_y,
    verbose = 1,
    batch_size = 64,
    epochs = 5,
    validation_split = 0.2
)
```

Epoch 1/5  
16/16 [=====] - 90s 6s/step - loss: 1.4418 - recall: 0.7939 - val\_loss: 4.1312 - val\_recall: 0.0000e+00  
Epoch 2/5  
16/16 [=====] - 87s 5s/step - loss: 0.4411 - recall: 0.9394 - val\_loss: 2.9098 - val\_recall: 0.0000e+00  
Epoch 3/5  
16/16 [=====] - 83s 5s/step - loss: 0.2428 - recall: 0.9394 - val\_loss: 3.0171 - val\_recall: 0.0000e+00  
Epoch 4/5  
16/16 [=====] - 82s 5s/step - loss: 0.2332 - recall: 0.9394 - val\_loss: 3.5917 - val\_recall: 0.0000e+00  
Epoch 5/5  
16/16 [=====] - 89s 5s/step - loss: 0.2237 - recall: 0.9394 - val\_loss: 3.6312 - val\_recall: 0.0000e+00

Нейронная сеть переобучилась

## ▼ MLP

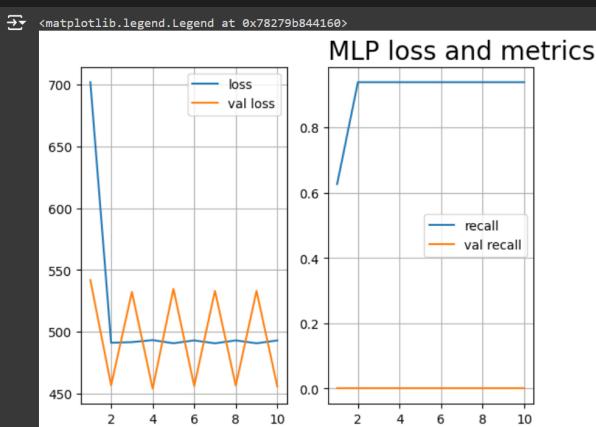
```
❶ mlp = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(256, 256, 3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(50, activation = 'swish', kernel_regularizer=tf.keras.regularizers.L1(0.01)),
    tf.keras.layers.Dense(55, activation = 'swish'),
    tf.keras.layers.Dense(60, activation = 'swish', kernel_regularizer=tf.keras.regularizers.L1(0.01)),
    tf.keras.layers.Dense(65, activation = 'swish'),
    tf.keras.layers.Dense(70, activation = 'swish', kernel_regularizer=tf.keras.regularizers.L1(0.01)),
    tf.keras.layers.Dense(75, activation = 'swish'),
    tf.keras.layers.Dense(3, activation = 'softmax'),
])
[ ] mlp.compile(
    loss = tf.keras.losses.categorical_crossentropy,
    optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.01),
    # metrics = [tf.keras.metrics.FalseNegatives(name = 'FN')]
    metrics=[tf.keras.metrics.Recall(name='recall')]
)
[ ] mlp_hist = mlp.fit(
    x = train_x,
    y = train_y,
    verbose = 1,
    batch_size = 32,
    epochs = 10,
    validation_split = 0.2
)
❷ Epoch 1/10
31/31 [=====] - 11s 319ms/step - loss: 701.7808 - recall: 0.6273 - val_loss: 541.8279 - val_recall: 0.0000e+00
Epoch 2/10
31/31 [=====] - 9s 283ms/step - loss: 491.1332 - recall: 0.9394 - val_loss: 456.7256 - val_recall: 0.0000e+00
Epoch 3/10
31/31 [=====] - 10s 320ms/step - loss: 491.6363 - recall: 0.9394 - val_loss: 532.2427 - val_recall: 0.0000e+00
Epoch 4/10
31/31 [=====] - 9s 282ms/step - loss: 493.2687 - recall: 0.9394 - val_loss: 454.0655 - val_recall: 0.0000e+00
Epoch 5/10
31/31 [=====] - 9s 280ms/step - loss: 490.7346 - recall: 0.9394 - val_loss: 534.6848 - val_recall: 0.0000e+00
Epoch 6/10
31/31 [=====] - 9s 308ms/step - loss: 493.0424 - recall: 0.9394 - val_loss: 456.2167 - val_recall: 0.0000e+00
Epoch 7/10
31/31 [=====] - 8s 275ms/step - loss: 490.6798 - recall: 0.9394 - val_loss: 533.0677 - val_recall: 0.0000e+00
Epoch 8/10
31/31 [=====] - 9s 297ms/step - loss: 493.0476 - recall: 0.9394 - val_loss: 456.4922 - val_recall: 0.0000e+00
Epoch 9/10
31/31 [=====] - 11s 353ms/step - loss: 490.7021 - recall: 0.9394 - val_loss: 533.0388 - val_recall: 0.0000e+00
Epoch 10/10
31/31 [=====] - 8s 273ms/step - loss: 493.0178 - recall: 0.9394 - val_loss: 455.7873 - val_recall: 0.0000e+00
```

8. Сравните качество многоклассовой классификации нейронными сетями при помощи показателя качества, указанного в индивидуальном задании.

В данном случае у нас снова выигрывает CNN.

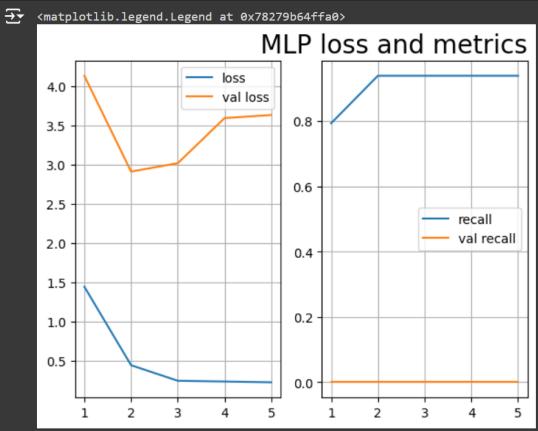
9. Постройте кривые обучения нейронных сетей многоклассовой классификации для показателей ошибки и доли верных ответов в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду. Сопроводите программный код необходимыми комментариями.

```
❶ _, ax = plt.subplots(1, 2)
ax[0].plot(np.arange(1, 11), mlp_hist.history['loss'], label = 'loss')
ax[1].plot(np.arange(1, 11), mlp_hist.history['recall'], label = 'recall')
ax[0].plot(np.arange(1, 11), mlp_hist.history['val_loss'], label = 'val loss')
ax[1].plot(np.arange(1, 11), mlp_hist.history['val_recall'], label = 'val recall')
plt.title('MLP loss and metrics', size = 20, loc = 'left')
# ax[1].title('MLP accuracy', size = 20)
# ax[0].xlabel('epochs', size = 15)
ax[0].grid(True)
ax[1].grid(True)
ax[0].legend()
ax[1].legend()
```



```
[ ] _, ax = plt.subplots(1, 2)
ax[0].plot(np.arange(1, 6), cnn_hist.history['loss'], label = 'loss')
```

```
ax[1].plot(np.arange(1, 6), cnn_hist.history['recall'], label = 'recall')
ax[0].plot(np.arange(1, 6), cnn_hist.history['val_loss'], label = 'val loss')
ax[1].plot(np.arange(1, 6), cnn_hist.history['val_recall'], label = 'val recall')
plt.title('MLP loss and metrics',size = 20, loc = 'right')
# ax[1].title('MLP accuracy', size = 20)
# ax[0].xlabel('epochs', size = 15)
ax[0].grid(True)
ax[1].grid(True)
ax[0].legend()
ax[1].legend()
```



[ ]

Colab paid products - Cancel contracts here

✓ 0s completed at 5:58PM

