

✓ РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра математического моделирования и искусственного интеллекта

ОТЧЕТ ПО КОНТРОЛЬНОЙ РАБОТЕ № 6

Дисциплина: Методы машинного обучения

Студент: Маслова Анастасия

Группа: НКНбд-01-21

Москва 2024

В соответствии с индивидуальным заданием, указанным в записной книжке команды, выполните следующие работы:

1. Загрузите заданный в индивидуальном задании набор данных с изображениями из Tensorflow Datasets с разбиением на обучающую, валидационную и тестовую выборки. Если при дальнейшей работе с данными возникнет нехватка вычислительных ресурсов, то разрешение изображений можно уменьшить.
2. Оставьте в наборе изображения, указанных в индивидуальном задании, и визуализируйте несколько изображений.
3. Постройте нейронные сети MLP, CNN и RNN для задачи многоклассовой классификации изображений (требования к архитектуре сетей указаны в индивидуальном задании), используя функцию потерь, указанную в индивидуальном задании. Подберите такие параметры, как функции активации, оптимизатор, начальная скорость обучения, размер мини-пакета и др. самостоятельно, обеспечивая обучение нейронных сетей. Обучайте нейронные сети с использованием валидационной выборки, сформированной в п. 1. Останавливайте обучение нейронных сетей в случае роста потерь на валидационной выборке на нескольких эпохах обучения подряд. Для каждой нейронной сети выведите количество потребовавшихся эпох обучения.
4. Оцените качество многоклассовой классификации нейронными сетями MLP, CNN и RNN на тестовой выборке при помощи показателя качества, указанного в индивидуальном задании, и выведите архитектуру нейронной сети с лучшим качеством.

5. Визуализируйте кривые обучения трех построенных моделей для показателя потерь на валидационной выборке на одном рисунке в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду. Используйте для визуализации относительные потери (потери, деленные на начальные потери на первой эпохе).
6. Визуализируйте кривые обучения трех построенных моделей для показателя доли верных ответов на валидационной выборке на одном рисунке в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.
7. Используя модель нейронной сети с лучшей долей верных ответов на тестовой выборке, определите для каждого из классов два изображения в тестовой выборке, имеющие минимальную и максимальную вероятности классификации в правильный класс, и визуализируйте эти изображения.

Вариант 23

1. Набор данных oxford_iiit_pet с изменением разрешения до 96x64
2. Классы с метками 3,14,25,35,36
3. Требования к архитектуре сети MLP:

Функциональный API при создании

Функция потерь: категориальная кросс-энтропия

Кол-во скрытых слоев 6

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

Использование слоев с регуляризацией L2

4. Требования к архитектуре сети CNN:

Последовательный API со списком слоев при создании

Функция потерь: разреженная категориальная кросс-энтропия

Кол-во сверточных слоев 3

Количество фильтров в сверточных слоях 16

Размеры фильтра 5x5

Использование слоев пакетной нормализации

5. Требования к архитектуре сети RNN:

Последовательный API с методом add() при создании

Функция потерь: категориальная кросс-энтропия

Слой LSTM с 64 нейронами

Использование слоев dropout

6. Показатель качества многоклассовой классификации:

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

Решение


1. Загрузите заданный в индивидуальном задании набор данных с изображениями из Tensorflow Datasets с разбиением на обучающую, валидационную и тестовую выборки. Если при дальнейшей работе с данными возникнет нехватка вычислительных ресурсов, то разрешение изображений можно уменьшить.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import tensorflow_datasets as tfds
from PIL import Image, ImageOps



from tensorflow.keras import models
from tensorflow.keras import layers
from keras.regularizers import l1_l2

from tensorflow.keras.losses import CategoricalCrossentropy

ds = tfds.load('oxford_iiit_pet', split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'])
df_train = tfds.as_dataframe(ds[0])
df_val = tfds.as_dataframe(ds[1])
df_test = tfds.as_dataframe(ds[2])
df_train.head(3)
```




	file_name	image	label	segmentation_mask	species
0	b'Sphynx_158.jpg'	[[[3, 3, 3], [5, 3, 4], [7, 5, 6], [4, 2, 3], ... [[[5, 9, 8],	33	[[[2], [2], [2], [2], [2], [2], [2], [2], [2],...	0



Далее: [Посмотреть рекомендованные графики](#)

```
df_train.shape, df_val.shape, df_test.shape
```



```
((2944, 5), (368, 5), (368, 5))
```

2. Оставьте в наборе изображения, указанных в индивидуальном задании, и визуализируйте несколько изображений.

```
x = df_train[df_train['label'] == 3]
y = df_train[df_train['label'] == 14]
z = df_train[df_train['label'] == 25]
a = df_train[df_train['label'] == 35]
b = df_train[df_train['label'] == 36]
```

```
x['label'] = 0
y['label'] = 1
z['label'] = 2
a['label'] = 3
b['label'] = 4
```

```
df_train1 = pd.concat([x, y, z, a, b])
Y_train1 = df_train1['label']
df_train1 = df_train1['image']
```

```
x = df_test[df_test['label'] == 3]
y = df_test[df_test['label'] == 14]
z = df_test[df_test['label'] == 25]
v = df_test[df_test['label'] == 35]
z = df_test[df_test['label'] == 36]
```

```
x['label'] = 0
y['label'] = 1
z['label'] = 2
a['label'] = 3
b['label'] = 4
```

```
df_test1 = pd.concat([x, y, z, a, b])
Y_test1 = df_test1['label']
df_test1 = df_test1['image']
```

```
x = df_val[df_val['label'] == 3]
y = df_val[df_val['label'] == 14]
z = df_val[df_val['label'] == 25]
a = df_val[df_val['label'] == 35]
b = df_val[df_val['label'] == 36]
```

```
x['label'] = 0
y['label'] = 1
z['label'] = 2
a['label'] = 3
b['label'] = 4
df_val1 = pd.concat([x, y, z, a, b])
Y_val1 = df_val1['label']
df_val1 = df_val1['image']
```

```
Y_train1.value_counts()
```



```
<ipython-input-130-472133ecd876>:7: 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/10min/boolean_indexing.html
x['label'] = 0

```
<ipython-input-130-472133ecd876>:8: 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/10min/boolean_indexing.html
y['label'] = 1

```
<ipython-input-130-472133ecd876>:9: 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/10min/boolean_indexing.html
z['label'] = 2

```
<ipython-input-130-472133ecd876>:10: 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/10min/boolean_indexing.html
a['label'] = 3

```
<ipython-input-130-472133ecd876>:11: 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/10min/boolean_indexing.html
b['label'] = 4

```
<ipython-input-130-472133ecd876>:23: 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/10min/boolean_indexing.html
x['label'] = 0

```
<ipython-input-130-472133ecd876>:24: 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/10min/boolean_indexing.html
y['label'] = 1

```
<ipython-input-130-472133ecd876>:25: 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/10min/boolean_indexing.html
z['label'] = 2

```
<ipython-input-130-472133ecd876>:26: 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/10min/boolean_indexing.html
a['label'] = 3

```
<ipython-input-130-472133ecd876>:27: 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
```

```
df_train1.head()
```

```
⇒ 15      [[[250, 212, 41], [245, 218, 41], [252, 222, 4...
    80      [[[0, 1, 0], [0, 1, 0], [0, 1, 0], [0, 1, 0], ...
    97      [[[124, 21, 24], [125, 22, 25], [126, 23, 26],...
    101     [[[6, 0, 28], [7, 0, 31], [6, 1, 31], [7, 0, 3...
    122     [[[171, 143, 129], [169, 140, 134], [169, 137,...
    Name: image, dtype: object
```

```
df_train1[15].shape
```

```
⇒ (375, 500, 3)
```

```
Y_train1.shape
```

```
⇒ (406,)
```

```
Y_train2 = list(Y_train1)
```

```
Y_test2 = list(Y_test1)
```

```
Y_val2 = list(Y_val1)
```

```
for i in range(len(Y_train2)):
```

```
    tmp = [0]*5
```

```
    tmp[Y_train2[i]] = 1
```

```
    Y_train2[i] = tmp
```

```
for i in range(len(Y_test2)):
```

```
    tmp = [0]*5
```

```
    tmp[Y_test2[i]] = 1
```

```
    Y_test2[i] = tmp
```

```
for i in range(len(Y_val2)):
```

```
    tmp = [0]*5
```

```
    tmp[Y_val2[i]] = 1
```

```
    Y_val2[i] = tmp
```

```
Y_train1 = np.array(Y_train1)
```

```
Y_train2 = np.array(Y_train2)
```

```
Y_test1 = np.array(Y_test1)
```

```
Y_test2 = np.array(Y_test2)
```

```
Y_val1 = np.array(Y_val1)
```

```
Y_val2 = np.array(Y_val2)
```

```
import random

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 every image
    for i in range(num_row * num_col):
        # Read the image
        img = imgs[i]
        # Display the image
        ax = axes[i // num_col, i % num_col]
        ax.imshow(img)

    plt.tight_layout()
    plt.show()

df_tr = np.zeros(shape=(df_train1.shape[0],64,96,3), dtype=np.float32)
df_te = np.zeros(shape=(df_test1.shape[0],64,96,3), dtype=np.float32)
df_va = np.zeros(shape=(df_val1.shape[0],64,96,3), dtype=np.float32)

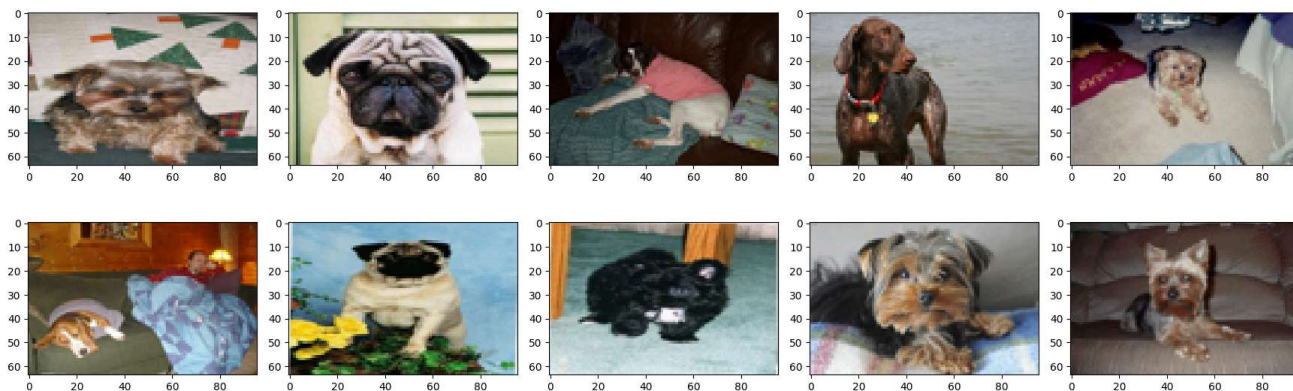
for i in range(len(df_train1)):
    df_tr[i,:,:,:] = np.array(Image.fromarray(df_train1.iloc[i]).resize((96,64)))
for i in range(len(df_test1)):
    df_te[i,:,:,:] = np.array(Image.fromarray(df_test1.iloc[i]).resize((96,64)))
for i in range(len(df_val1)):
    df_va[i,:,:,:] = np.array(Image.fromarray(df_val1.iloc[i]).resize((96,64)))

df_tr /= 255
df_te /= 255
df_va /= 255

df_tr.shape, Y_train1.shape

→ ((406, 64, 96, 3), (406,))

plot_random_sample(df_tr)
```



3. Постройте нейронные сети MLP, CNN и RNN для задачи многоклассовой классификации изображений (требования к архитектуре сетей указаны в индивидуальном задании), используя функцию потерь, указанную в индивидуальном задании. Подберите такие параметры, как функции активации, оптимизатор, начальная скорость обучения, размер мини-пакета и др. самостоятельно, обеспечивая обучение нейронных сетей. Обучайте нейронные сети с использованием валидационной выборки, сформированной в п. 1. Останавливайте обучение нейронных сетей в случае роста потерь на валидационной выборке на нескольких эпохах обучения подряд. Для каждой нейронной сети выведите количество потребовавшихся эпох обучения.

Требования к архитектуре сети MLP:

Функциональный API при создании

Функция потерь: категориальная кросс-энтропия

Кол-во скрытых слоев 6

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

Использование слоев с регуляризацией L2


```

from tensorflow.keras.utils import to_categorical


# Convert target data to one-hot encoded format
Y_train1_encoded = to_categorical(Y_train1, num_classes=5)
Y_val1_encoded = to_categorical(Y_val1, num_classes=5)

inputs = tf.keras.Input(shape=(64, 96, 3))
x = tf.keras.layers.Flatten()(inputs)
x = tf.keras.layers.Dense(50, activation="leaky_relu")(x)
x = tf.keras.layers.Dense(55, activation="leaky_relu", kernel_regularizer='l2')(x)
x = tf.keras.layers.Dense(60, activation="leaky_relu")(x)
x = tf.keras.layers.Dense(65, activation="leaky_relu", kernel_regularizer='l2')(x)
x = tf.keras.layers.Dense(70, activation="leaky_relu")(x)
x = tf.keras.layers.Dense(75, activation="leaky_relu", kernel_regularizer='l2')(x)
outputs = tf.keras.layers.Dense(5, activation="softmax")(x)

mlp = tf.keras.Model(inputs=inputs, outputs=outputs)
mlp.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.002),
            loss=CategoricalCrossentropy(),
            metrics=['categorical_accuracy'])

history1 = mlp.fit(
    df_tr,
    Y_train1_encoded,
    epochs=50,
    validation_data=(df_va, Y_val1_encoded),
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)],
    batch_size=29
)

```

 Epoch 1/50
 14/14 [=====] - 2s 23ms/step - loss: 4.1590 - categorical_ac
 Epoch 2/50
 14/14 [=====] - 0s 8ms/step - loss: 3.4909 - categorical_acc
 Epoch 3/50
 14/14 [=====] - 0s 7ms/step - loss: 3.2956 - categorical_acc
 Epoch 4/50
 14/14 [=====] - 0s 8ms/step - loss: 3.1424 - categorical_acc
 Epoch 5/50
 14/14 [=====] - 0s 8ms/step - loss: 3.0663 - categorical_acc
 Epoch 6/50
 14/14 [=====] - 0s 9ms/step - loss: 3.0129 - categorical_acc
 Epoch 7/50
 14/14 [=====] - 0s 8ms/step - loss: 3.1566 - categorical_acc
 Epoch 8/50
 14/14 [=====] - 0s 8ms/step - loss: 2.8700 - categorical_acc
 Epoch 9/50
 14/14 [=====] - 0s 8ms/step - loss: 2.8014 - categorical_acc
 Epoch 10/50
 14/14 [=====] - 0s 7ms/step - loss: 2.6788 - categorical_acc
 Epoch 11/50
 14/14 [=====] - 0s 10ms/step - loss: 2.5270 - categorical_ac
 Epoch 12/50
 14/14 [=====] - 0s 8ms/step - loss: 2.4697 - categorical_acc
 Epoch 13/50

```
14/14 [=====] - 0s 8ms/step - loss: 2.5896 - categorical_acc
Epoch 14/50
14/14 [=====] - 0s 9ms/step - loss: 2.4237 - categorical_acc
Epoch 15/50
14/14 [=====] - 0s 8ms/step - loss: 2.3106 - categorical_acc
Epoch 16/50
14/14 [=====] - 0s 9ms/step - loss: 2.4144 - categorical_acc
Epoch 17/50
14/14 [=====] - 0s 8ms/step - loss: 2.2199 - categorical_acc
Epoch 18/50
14/14 [=====] - 0s 8ms/step - loss: 2.0922 - categorical_acc
Epoch 19/50
14/14 [=====] - 0s 8ms/step - loss: 2.1848 - categorical_acc
Epoch 20/50
14/14 [=====] - 0s 8ms/step - loss: 1.9858 - categorical_acc
Epoch 21/50
14/14 [=====] - 0s 9ms/step - loss: 1.9947 - categorical_acc
Epoch 22/50
14/14 [=====] - 0s 8ms/step - loss: 1.9380 - categorical_acc
Epoch 23/50
14/14 [=====] - 0s 8ms/step - loss: 1.8123 - categorical_acc
Epoch 24/50
14/14 [=====] - 0s 7ms/step - loss: 1.6720 - categorical_acc
```

Требования к архитектуре сети CNN:

Последовательный API со списком слоев при создании

Функция потерь: разреженная категориальная кросс-энтропия

Кол-во сверточных слоев 3

Количество фильтров в сверточных слоях 16

Размеры фильтра 5x5

Использование слоев пакетной нормализации

```
Y_train1_integer = np.argmax(Y_train1_encoded, axis=1)
```

```
Y_val1_integer = np.argmax(Y_val1_encoded, axis=1)
```

```

cnn = tf.keras.Sequential([
    tf.keras.Input(shape=(64, 96, 3)),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5), activation='selu', kernel_regu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5), activation='selu', kernel_regu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5), activation='selu', kernel_regu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(5, activation='softmax')
])

```

```

cnn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.002),
            loss=tf.keras.losses.SparseCategoricalCrossentropy(),
            metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')])

```

```

history2 = cnn.fit(
    df_tr,
    Y_train1_integer,
    epochs=30,
    validation_data=(df_va, Y_val1_integer),
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)],
    batch_size=29
)

```

```

⇒ Epoch 1/30
14/14 [=====] - 2s 35ms/step - loss: 8.0752 - accuracy: 0.31
Epoch 2/30
14/14 [=====] - 0s 12ms/step - loss: 6.6065 - accuracy: 0.56
Epoch 3/30
14/14 [=====] - 0s 12ms/step - loss: 5.3976 - accuracy: 0.67
Epoch 4/30
14/14 [=====] - 0s 12ms/step - loss: 4.4416 - accuracy: 0.72
Epoch 5/30
14/14 [=====] - 0s 11ms/step - loss: 3.6341 - accuracy: 0.75
Epoch 6/30
14/14 [=====] - 0s 12ms/step - loss: 3.0501 - accuracy: 0.75
Epoch 7/30
14/14 [=====] - 0s 14ms/step - loss: 2.6773 - accuracy: 0.74
Epoch 8/30
14/14 [=====] - 0s 12ms/step - loss: 2.3261 - accuracy: 0.81
Epoch 9/30
14/14 [=====] - 0s 13ms/step - loss: 2.0359 - accuracy: 0.84
Epoch 10/30
14/14 [=====] - 0s 13ms/step - loss: 1.9000 - accuracy: 0.81
Epoch 11/30
14/14 [=====] - 0s 13ms/step - loss: 1.7849 - accuracy: 0.86
Epoch 12/30
14/14 [=====] - 0s 15ms/step - loss: 1.6479 - accuracy: 0.87
Epoch 13/30
14/14 [=====] - 0s 13ms/step - loss: 1.5256 - accuracy: 0.89
Epoch 14/30
14/14 [=====] - 0s 13ms/step - loss: 1.5281 - accuracy: 0.84

```

Epoch 15/30

14/14 [=====] - 0s 13ms/step - loss: 1.4286 - accuracy: 0.90

Требования к архитектуре сети RNN:

Последовательный API с методом add() при создании

Функция потерь: категориальная кросс-энтропия

Слой LSTM с 64 нейронами

Использование слоев dropout

```
df_tr2 = df_tr.reshape(df_tr.shape[0], df_tr.shape[-1], -1)
df_te2 = df_te.reshape(df_te.shape[0], df_te.shape[-1], -1)
df_va2 = df_va.reshape(df_va.shape[0], df_va.shape[-1], -1)
```

```
rnn = tf.keras.Sequential()
rnn.add(tf.keras.layers.LSTM(64))
rnn.add(tf.keras.layers.Dropout(0.2))
rnn.add(tf.keras.layers.Dense(5, activation="softmax"))
```

```
rnn.compile(optimizer=tf.keras.optimizers.Adam(0.0001),
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=[tf.keras.metrics.CategoricalAccuracy(name='accuracy')])
```

```
history3 = rnn.fit(df_tr2,
                  Y_train2,
                  epochs=50,
                  validation_data=(df_va2, Y_val2),
                  batch_size=14,
                  callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])
```



```
Epoch 1/50
29/29 [=====] - 2s 21ms/step - loss: 1.6611 - accuracy: 0.18
Epoch 2/50
29/29 [=====] - 0s 6ms/step - loss: 1.5949 - accuracy: 0.241
Epoch 3/50
29/29 [=====] - 0s 7ms/step - loss: 1.5667 - accuracy: 0.275
Epoch 4/50
29/29 [=====] - 0s 5ms/step - loss: 1.5436 - accuracy: 0.312
Epoch 5/50
29/29 [=====] - 0s 5ms/step - loss: 1.5054 - accuracy: 0.364
Epoch 6/50
29/29 [=====] - 0s 7ms/step - loss: 1.4613 - accuracy: 0.384
Epoch 7/50
29/29 [=====] - 0s 5ms/step - loss: 1.4602 - accuracy: 0.403
Epoch 8/50
29/29 [=====] - 0s 5ms/step - loss: 1.4717 - accuracy: 0.359
Epoch 9/50
29/29 [=====] - 0s 6ms/step - loss: 1.4088 - accuracy: 0.406
Epoch 10/50
29/29 [=====] - 0s 6ms/step - loss: 1.3877 - accuracy: 0.421
```

```

Epoch 11/50
29/29 [=====] - 0s 6ms/step - loss: 1.3424 - accuracy: 0.448
Epoch 12/50
29/29 [=====] - 0s 6ms/step - loss: 1.3231 - accuracy: 0.465
Epoch 13/50
29/29 [=====] - 0s 5ms/step - loss: 1.3200 - accuracy: 0.475
Epoch 14/50
29/29 [=====] - 0s 5ms/step - loss: 1.2579 - accuracy: 0.527
Epoch 15/50
29/29 [=====] - 0s 5ms/step - loss: 1.2624 - accuracy: 0.502
Epoch 16/50
29/29 [=====] - 0s 6ms/step - loss: 1.2057 - accuracy: 0.536

```

```

print("MLP epochs:", len(history1.history['loss']))
print("RNN epochs:", len(history2.history['loss']))
print("CNN epochs:", len(history3.history['loss']))

```

```

↔ MLP epochs: 24
   RNN epochs: 15
   CNN epochs: 16

```

4. Оцените качество многоклассовой классификации нейронными сетями MLP, CNN и RNN на тестовой выборке при помощи показателя качества, указанного в индивидуальном задании, и выведите архитектуру нейронной сети с лучшим качеством.

Показатель качества многоклассовой классификации: минимальная полнота классов, где полнота (recall) класса равна доле правильных предсказаний для всех точек, принадлежащих этому классу.

```

X1 = mlp.predict(df_te)
X2 = cnn.predict(df_te)
X3 = rnn.predict(df_te2)

```

```

↔ 7/7 [=====] - 0s 3ms/step
   7/7 [=====] - 0s 3ms/step
   7/7 [=====] - 1s 3ms/step

```


```
for i in range(len(X1)):
    for j in range(len(X1[i])):
        if X1[i][j] == max(X1[i]):
            X1[i][j] = 1
        else:
            X1[i][j] = 0
for i in range(len(X2)):
    for j in range(len(X2[i])):
        if X2[i][j] == max(X2[i]):
            X2[i][j] = 1
        else:
            X2[i][j] = 0
for i in range(len(X3)):
    for j in range(len(X3[i])):
        if X3[i][j] == max(X3[i]):
            X3[i][j] = 1
        else:
            X3[i][j] = 0
```

```
X11 = np.array(X1, dtype=np.int32)
X22 = np.array(X2, dtype=np.int32)
X33 = np.array(X3, dtype=np.int32)
```


```
m1 = tf.keras.metrics.Recall()
m2 = tf.keras.metrics.Recall()
m3 = tf.keras.metrics.Recall()
```

```
m1.update_state(X11, Y_test2)
m2.update_state(X22, Y_test2)
m3.update_state(X33, Y_test2)
```

```
m1.result().numpy(), m2.result().numpy(), m3.result().numpy()
```

 (0.53, 0.42, 0.45)

```
mlp.summary()
```

 Model: "model_1"

Layer (type)	Output Shape	Param #
=====		
input_3 (InputLayer)	[(None, 64, 96, 3)]	0
flatten_2 (Flatten)	(None, 18432)	0
dense_12 (Dense)	(None, 50)	921650
dense_13 (Dense)	(None, 55)	2805
dense_14 (Dense)	(None, 60)	3360
dense_15 (Dense)	(None, 65)	3965

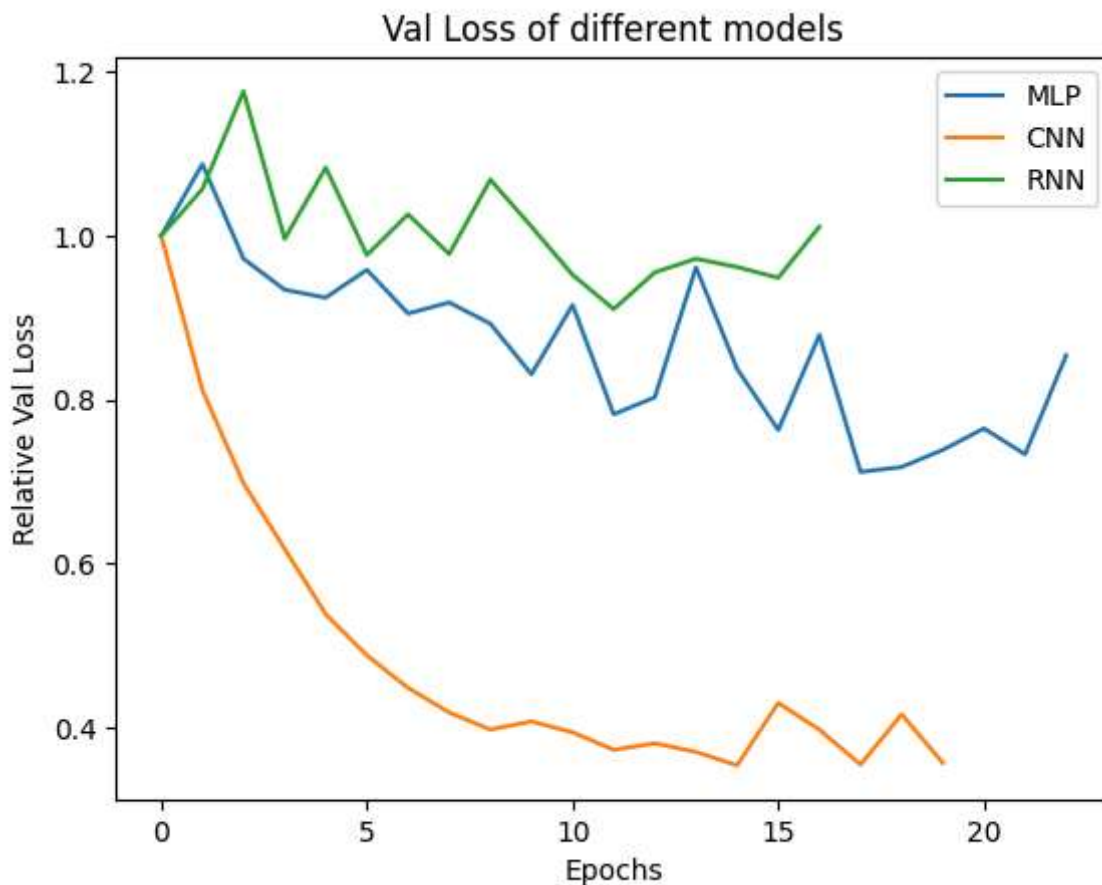
dense_16 (Dense)	(None, 70)	4620
dense_17 (Dense)	(None, 75)	5325
dense_18 (Dense)	(None, 5)	380

```
=====
Total params: 942105 (3.59 MB)
Trainable params: 942105 (3.59 MB)
Non-trainable params: 0 (0.00 Byte)
```

5. Визуализируйте кривые обучения трех построенных моделей для показателя потерь на валидационной выборке на одном рисунке в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду. Используйте для визуализации относительные потери (потери, деленные на начальные потери на первой эпохе).

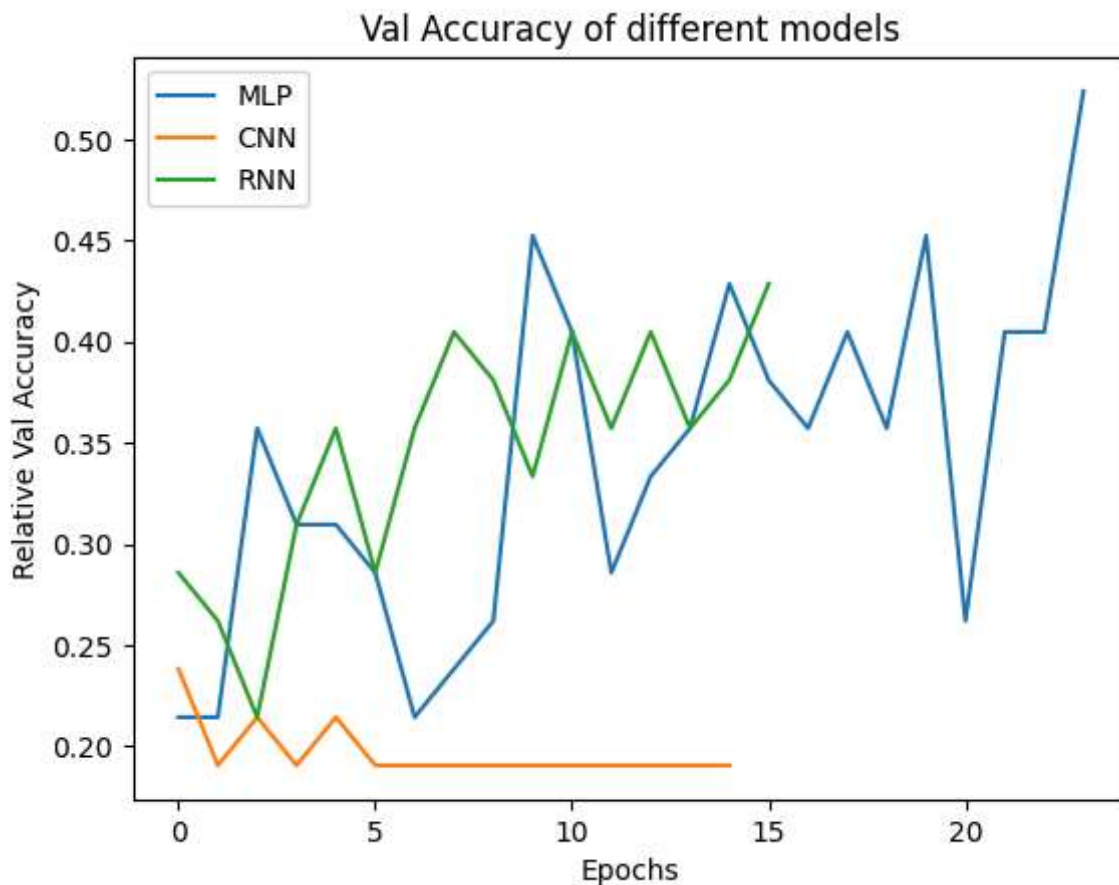
```
a1 = history1.history['val_loss']
a1 = [x/history1.history['val_loss'][0] for x in a1]
a2 = history2.history['val_loss']
a2 = [x/history2.history['val_loss'][0] for x in a2]
a3 = history3.history['val_loss']
a3 = [x/history3.history['val_loss'][0] for x in a3]

plt.plot([i for i in range(len(history1.history['val_loss']))], a1, label='MLP')
plt.plot([i for i in range(len(history2.history['val_loss']))], a2, label='CNN')
plt.plot([i for i in range(len(history3.history['val_loss']))], a3, label='RNN')
plt.xlabel('Epochs')
plt.ylabel('Relative Val Loss')
plt.title('Val Loss of different models')
plt.legend();
```



6. Визуализируйте кривые обучения трех построенных моделей для показателя доли верных ответов на валидационной выборке на одном рисунке в зависимости от эпохи обучения, подписывая оси и рисунок и создавая легенду.

```
plt.plot([i for i in range(len(history1.history['val_categorical_accuracy']))], history1.history['val_categorical_accuracy'])
plt.plot([i for i in range(len(history2.history['val_accuracy']))], history2.history['val_accuracy'])
plt.plot([i for i in range(len(history3.history['val_accuracy']))], history3.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Relative Val Accuracy')
plt.title('Val Accuracy of different models')
plt.legend();
```

7. Используя модель нейронной сети с лучшей долей верных ответов на тестовой выборке, определите для каждого из классов два изображения в тестовой выборке, имеющие минимальную и максимальную вероятности классификации в правильный класс, и визуализируйте эти изображения.

```
A = [0]*5 # max
```

```
B = [0]*5 # min
```

```
for i in range(len(X1)):
    for j in range(len(X1[i])):
        if X1[i][j] > X1[A[j]][j] and Y_test2[i][j] == 1:
            A[j] = i

        if X1[i][j] < X1[B[j]][j] and Y_test2[i][j] == 1:
            B[j] = i
```

A, B

```
([1, 0, 27, 30, 114], [0, 18, 0, 0, 0])
```

Class 1.

Max

```
plt.imshow(Image.fromarray(df_test1.iloc[A[0]]))
```

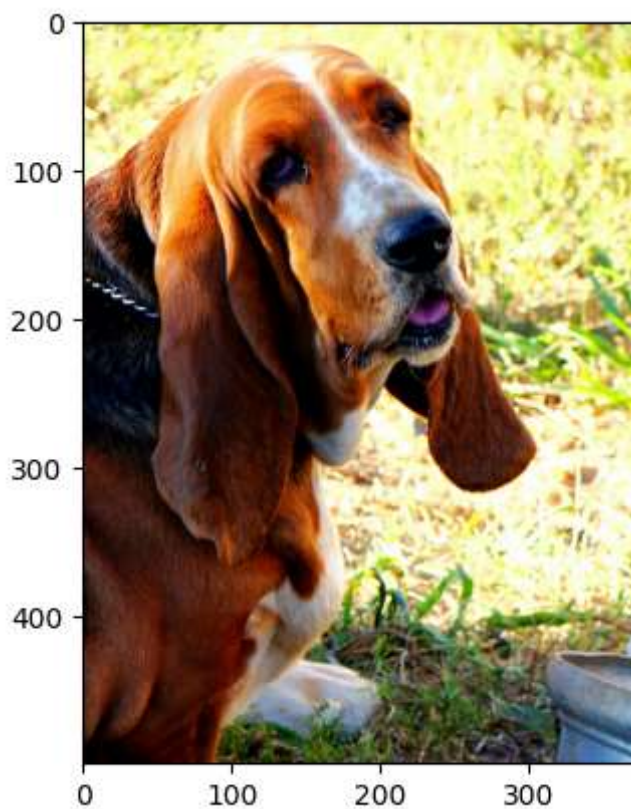
↪ <matplotlib.image.AxesImage at 0x7e142788afe0>



Min

```
plt.imshow(Image.fromarray(df_test1.iloc[B[0]]))
```

↪ <matplotlib.image.AxesImage at 0x7e142770c520>

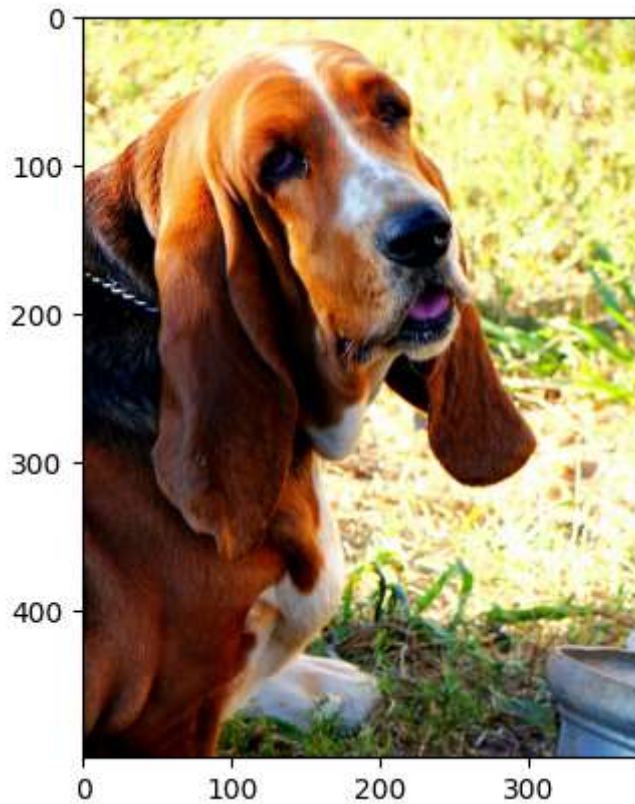


Class 2

Max


```
plt.imshow(Image.fromarray(df_test1.iloc[A[1]]))
```

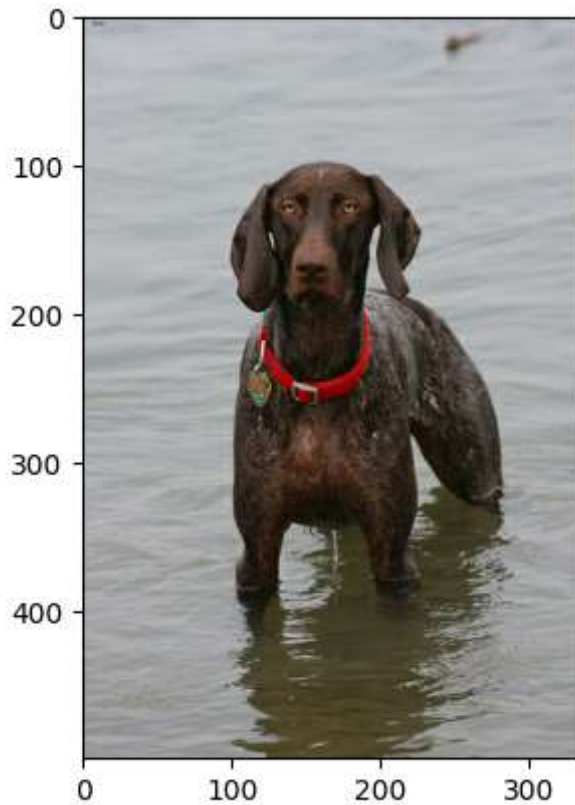
↗ <matplotlib.image.AxesImage at 0x7e142775b220>



Min

```
plt.imshow(Image.fromarray(df_test1.iloc[B[1]]))
```

 <matplotlib.image.AxesImage at 0x7e14277b6a70>



Class 3

Max


```
plt.imshow(Image.fromarray(df_test1.iloc[A[2]]))
```

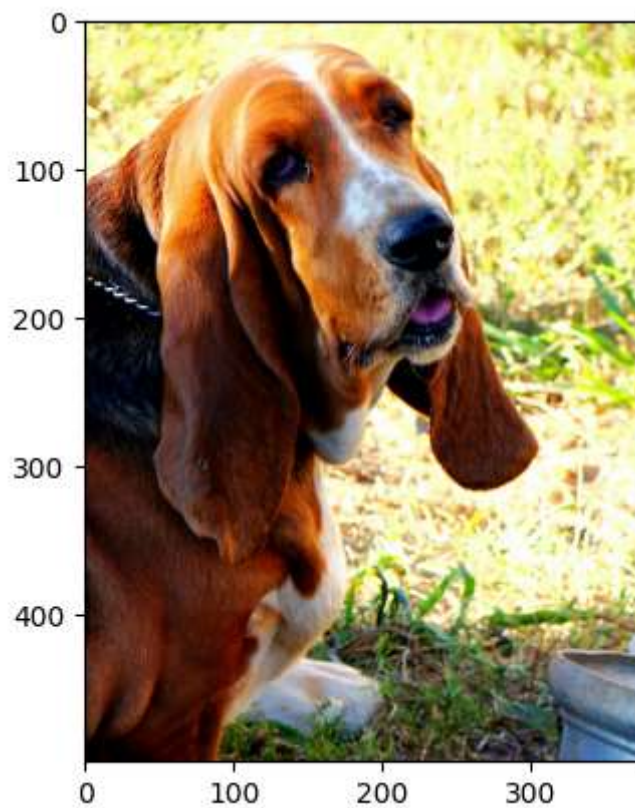
 <matplotlib.image.AxesImage at 0x7e14279313c0>



Min

```
plt.imshow(Image.fromarray(df_test1.iloc[B[2]]))
```

 <matplotlib.image.AxesImage at 0x7e14277fcc70>



Class 4

Max

```
plt.imshow(Image.fromarray(df_test1.iloc[A[3]]))
```


 <matplotlib.image.AxesImage at 0x7e14279e0250>

0

