ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

канд. техн. наук , доцент		В. Ю. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Классификация изображений. Сверточные сети. Предобученные сверточные сети.

по курсу: Интеллектуальный анализ данных на основе методов машинного обучения

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	4131з		М. Д. Быстров
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2026

Задание

Дан набор данных изображений цветов, состоящий из k классов, разнесенных по отдельным папкам (набор найти самостоятельно). Сформировать обучающую, валидационную и тестовую выборки из изображений исходного набора. Преобразовать исходные изображения, имеющие разную размерность матрицы, к одной размерности.

Задание

1. Построить сверточную нейронную сеть из комбинации слоев: Conv2D и MaxPooling, длинны, допустимой приведенной размерностью входных изображений. В качестве итоговых слоев классификатора применить полносвязную сеть (не менее двух слоев), позволяющую классифицировать на k классов. Пример см. в лекциях. Выполнить обучение построенной сети для решения задачи классификации изображений цветов из данного набора по k классам с одновременной валидацией. В качестве методов обучения использовать алгоритмы RMSProp или Adam. Выполнить при этом оценки точности обучения и валидации, а также ошибки потерь. Построить соответствующие графики. При этом использовать генератор изображений, формирующий наборы данных из папок с изображениями, аугментацию данных (Lect_4_ИАДНОММО), минипакетный способ обучения, регуляризацию l1,l2 или дропаут в целях борьбы с переобучением.

2. Построить сверточную нейронную сеть, позволяющую классифицировать на k классов исходный набор данных изображений цветов. Использовать при построении сети сверточный блок одной из предобученных сверточных моделей в Keras (выбрать в https://keras.io/api/applications/) и один из двух подходов трансферного обучения, рассмотренных в лекциях (Lect_4_ИАДНОММО). Пример см. в лекциях. Выполнить обучение построенной сети для решения задачи классификации изображений данного набора по k классам с одновременной валидацией. В качестве методов обучения использовать RMSProp или Adam. Выполнить при этом оценки точности обучения и валидации, а также ошибки потерь. Построить соответствующие графики. При этом использовать генератор изображений, формирующий наборы данных из папок с изображениями,

аугментацию в зависимости от метода трансферного обучения, мини-пакетный способ обучения, регуляризацию l1,l2 или дропаут в целях борьбы с переобучением.

3. Сравнить полученные точности и потери для построенных глубоких сетей на этапе тестирования. Использовать метрики, необходимые в зависимости от сбалансированности классов набора.

Результат выполнения задания

Лабораторная работа №3

Классификация изображений. Сверточные сети. Предобученные сверточные сети.

Подготовка среды

```
# установим kaggle
%pip install kagglehub
Collecting kagglehub
  Downloading kagglehub-0.3.13-py3-none-any.whl.metadata (38 kB)
Requirement already satisfied: packaging in c:\users\admin\anaconda3\lib\site-packages
(from kagglehub) (24.1)
Requirement already satisfied: pyyaml in c:\users\admin\anaconda3\lib\site-packages
(from kagglehub) (6.0.1)
Requirement already satisfied: requests in c:\users\admin\anaconda3\lib\site-packages
(from kagglehub) (2.32.3)
Requirement already satisfied: tqdm in c:\users\admin\anaconda3\lib\site-packages
(from kagglehub) (4.66.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\admin\anaconda3\lib\site-packages (from requests->kagglehub) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\admin\anaconda3\lib\site-
packages (from requests->kagglehub) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\admin\anaconda3\lib\site-packages (from requests->kagglehub) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\admin\anaconda3\lib\site-packages (from requests->kagglehub) (2025.11.12)
Requirement already satisfied: colorama in c:\users\admin\anaconda3\lib\site-packages
(from tqdm->kagglehub) (0.4.6)
Downloading kagglehub-0.3.13-py3-none-any.whl (68 kB)
Installing collected packages: kagglehub
Successfully installed kagglehub-0.3.13
Note: you may need to restart the kernel to use updated packages.
# скачаем датасет
import kagglehub

# Download latest version
path = kagglehub.dataset_download("alxmamaev/flowers-recognition")

print("Path to dataset files:", path)
Path to dataset files: C:\Users\Admin\.cache\kagglehub\datasets\alxmamaev\flowers-
recognition\versions\2
```

Подготовка датасета

```
seed = 7
from keras.preprocessing import image_dataset_from_directory
import tensorflow as tf

data_dir = path + "\\flowers"

# читаем датасет из папки: тренировочный и валидационный
```

```
train_ds, val_ds = image_dataset_from_directory(
    data_dir,
    image_size=(150, 150),
    batch_size=32, # пакеты по 32 изображения - мини-пакетное обучение
    label_mode='categorical',
    validation_split=0.2, # 20% под валидационный датасет
    subset="both",
    seed=seed
)

batches = tf.data.experimental.cardinality(val_ds)

# разделяем валидационный датасет на тестовый и валидационный)
test_ds = val_ds.take(batches // 2)
val_ds = val_ds.skip(batches // 2)

print(f"{tf.data.experimental.cardinality(val_ds).numpy()} for validation")
print(f"{tf.data.experimental.cardinality(test_ds).numpy()} for test")
```

```
Found 4317 files belonging to 5 classes.
Using 3454 files for training.
Using 863 files for validation.
14 for validation
13 for test
```

Проверим сбалансированность данных (распределение по классам)

```
import numpy as np

y_labels = []
for images, labels in train_ds:
    y_labels.extend(labels.numpy())
for images, labels in val_ds:
    y_labels.extend(labels.numpy())
for images, labels in test_ds:
    y_labels.extend(labels.numpy())

class_counts = np.sum(y_labels, axis=0)

for i, count in enumerate(class_counts):
    print(f"Класс {i}: {int(count)} примеров")
```

```
Класс 0: 764 примеров
Класс 1: 1052 примеров
Класс 2: 784 примеров
Класс 3: 733 примеров
Класс 4: 984 примеров
```

Данные достаточно сбалансированы, нет сильного преимущества одних классов над другими

1. Обучение Conv2D и MaxPooling

Сборка модели

```
import tensorflow as tf
from keras import layers, models, regularizers, optimizers

# количество классов
num_classes = 5

regularizer = regularizers.l1(0.0001)
#regularizer = None
```

```
model = models.Sequential([
    # входные изображения
    layers.Input(shape=(150, 150, 3)),

    # аугментация (должно работать только при обучении)
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomBrightness(0.1),

    # нормализация - пиксели по каждому из цветов занимают байт. приводим к диапазону
[0,1]
    layers.Rescaling(1./255),

    # Conv2D + MaxPooling2D
    layers.Conv2D(32, (3, 3), activation='relu', padding="same",
                  kernel_regularizer=regularizer),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu', padding="same",
                  kernel_regularizer=regularizer),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(96, (3, 3), activation='relu', padding="same",
                  kernel_regularizer=regularizer),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(96, (3, 3), activation='relu', padding="same",
                  kernel_regularizer=regularizer),
    layers.MaxPooling2D((2, 2)),

    # полносвязные слои для получения классификации
    layers.Flatten(),
    layers.Dense(512, activation='relu',
                 kernel_regularizer=regularizer),
    # layers.Dense(32, activation='relu',
    #               kernel_regularizer=regularizer),
    #layers.Dropout(0.5), # Хорошая практика вместе с L1

    # Выходной слой для многоклассовой классификации
    layers.Dense(num_classes, activation='softmax')
])

# 4. Компиляция
model.compile(
    optimizer=optimizers.Adam(0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
random_flip_5 (RandomFlip)	(None, 150, 150, 3)	0

random_rotation_5 (RandomRotation)	(None, 150, 150, 3)	0
random_zoom_5 (RandomZoom)	(None, 150, 150, 3)	0
random_brightness_5 (RandomBrightness)	(None, 150, 150, 3)	0
rescaling_2 (Rescaling)	(None, 150, 150, 3)	0
conv2d_4 (Conv2D)	(None, 150, 150, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_5 (Conv2D)	(None, 75, 75, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 37, 37, 64)	0
conv2d_6 (Conv2D)	(None, 37, 37, 96)	55,392
max_pooling2d_6 (MaxPooling2D)	(None, 18, 18, 96)	0
conv2d_7 (Conv2D)	(None, 18, 18, 96)	83,040
max_pooling2d_7 (MaxPooling2D)	(None, 9, 9, 96)	0
flatten_5 (Flatten)	(None, 7776)	0
dense_10 (Dense)	(None, 512)	3,981,824
dense_11 (Dense)	(None, 5)	2,565

Total params: 4,142,213 (15.80 MB)

Trainable params: 4,142,213 (15.80 MB)

Non-trainable params: 0 (0.00 B)

Обучение модели

```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    #steps_per_epoch=50,
```

```
    epochs=100)
Epoch 1/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 390ms/step - accuracy: 0.3049 - loss: 2.9367 - val_accuracy: 0.4251 - val_loss: 1.6964
Epoch 2/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 51s 473ms/step - accuracy: 0.5035 - loss: 1.5184 - val_accuracy: 0.5369 - val_loss: 1.4941
Epoch 3/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 51s 468ms/step - accuracy: 0.5483 - loss: 1.3922 - val_accuracy: 0.5794 - val_loss: 1.3773
Epoch 4/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 444ms/step - accuracy: 0.5935 - loss: 1.2623 - val_accuracy: 0.5772 - val_loss: 1.3798
Epoch 5/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 443ms/step - accuracy: 0.6022 - loss: 1.2113 - val_accuracy: 0.5257 - val_loss: 1.4569
Epoch 6/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 460ms/step - accuracy: 0.6245 - loss: 1.1613 - val_accuracy: 0.6130 - val_loss: 1.2209
Epoch 7/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 56s 520ms/step - accuracy: 0.6288 - loss: 1.1440 - val_accuracy: 0.6242 - val_loss: 1.2666
Epoch 8/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 440ms/step - accuracy: 0.6329 - loss: 1.1081 - val_accuracy: 0.6063 - val_loss: 1.2507
Epoch 9/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 459ms/step - accuracy: 0.6306 - loss: 1.1108 - val_accuracy: 0.5996 - val_loss: 1.2384
Epoch 10/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 459ms/step - accuracy: 0.6503 - loss: 1.0730 - val_accuracy: 0.6242 - val_loss: 1.2173
Epoch 11/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 438ms/step - accuracy: 0.6587 - loss: 1.0533 - val_accuracy: 0.6398 - val_loss: 1.1133
Epoch 12/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 412ms/step - accuracy: 0.6587 - loss: 1.0232 - val_accuracy: 0.6622 - val_loss: 1.0991
Epoch 13/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 432ms/step - accuracy: 0.6647 - loss: 1.0269 - val_accuracy: 0.6465 - val_loss: 1.1274
Epoch 14/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 49s 450ms/step - accuracy: 0.6769 - loss: 0.9988 - val_accuracy: 0.6555 - val_loss: 1.0938
Epoch 15/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 51s 473ms/step - accuracy: 0.6783 - loss: 0.9876 - val_accuracy: 0.6711 - val_loss: 1.0455
Epoch 16/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 56s 521ms/step - accuracy: 0.6899 - loss: 0.9617 - val_accuracy: 0.6868 - val_loss: 1.0229
Epoch 17/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 465ms/step - accuracy: 0.6838 - loss: 0.9624 - val_accuracy: 0.7114 - val_loss: 0.9901
```

```
Epoch 18/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 53s 492ms/step - accuracy: 0.6995 - loss: 0.9409 - val_accuracy: 0.6532 - val_loss: 1.0513
Epoch 19/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 52s 482ms/step - accuracy: 0.7082 - loss: 0.9229 - val_accuracy: 0.6577 - val_loss: 1.0920
Epoch 20/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 51s 470ms/step - accuracy: 0.7166 - loss: 0.9206 - val_accuracy: 0.6801 - val_loss: 0.9999
Epoch 21/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 53s 495ms/step - accuracy: 0.6914 - loss: 0.9385 - val_accuracy: 0.6488 - val_loss: 1.1136
Epoch 22/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 51s 470ms/step - accuracy: 0.7160 - loss: 0.9078 - val_accuracy: 0.7159 - val_loss: 0.9718
Epoch 23/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 427ms/step - accuracy: 0.7250 - loss: 0.8912 - val_accuracy: 0.6890 - val_loss: 1.0213
Epoch 24/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 424ms/step - accuracy: 0.7258 - loss: 0.8855 - val_accuracy: 0.7002 - val_loss: 0.9554
Epoch 25/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 423ms/step - accuracy: 0.7238 - loss: 0.8907 - val_accuracy: 0.6779 - val_loss: 0.9997
Epoch 26/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 421ms/step - accuracy: 0.7247 - loss: 0.8842 - val_accuracy: 0.6711 - val_loss: 1.0227
Epoch 27/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 53s 491ms/step - accuracy: 0.7177 - loss: 0.8859 - val_accuracy: 0.7002 - val_loss: 0.9736
Epoch 28/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 54s 494ms/step - accuracy: 0.7325 - loss: 0.8852 - val_accuracy: 0.6868 - val_loss: 1.0370
Epoch 29/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 51s 469ms/step - accuracy: 0.7305 - loss: 0.8727 - val_accuracy: 0.6667 - val_loss: 1.0890
Epoch 30/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 49s 453ms/step - accuracy: 0.7400 - loss: 0.8755 - val_accuracy: 0.6913 - val_loss: 0.9914
Epoch 31/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 462ms/step - accuracy: 0.7334 - loss: 0.8656 - val_accuracy: 0.6935 - val_loss: 1.0491
Epoch 32/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 55s 506ms/step - accuracy: 0.7458 - loss: 0.8479 - val_accuracy: 0.6913 - val_loss: 1.0206
Epoch 33/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 56s 520ms/step - accuracy: 0.7383 - loss: 0.8454 - val_accuracy: 0.7092 - val_loss: 0.9543
Epoch 34/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 53s 490ms/step - accuracy: 0.7403 - loss: 0.8434 - val_accuracy: 0.6779 - val_loss: 0.9799
Epoch 35/100
```

```
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 429ms/step - accuracy: 0.7455 - loss: 0.8303 - val_accuracy: 0.7002 - val_loss: 1.0611
Epoch 36/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 49s 452ms/step - accuracy: 0.7449 - loss: 0.8410 - val_accuracy: 0.6711 - val_loss: 1.0302
Epoch 37/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 465ms/step - accuracy: 0.7504 - loss: 0.8314 - val_accuracy: 0.6935 - val_loss: 1.0253
Epoch 38/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 438ms/step - accuracy: 0.7533 - loss: 0.8268 - val_accuracy: 0.7069 - val_loss: 1.0010
Epoch 39/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 435ms/step - accuracy: 0.7533 - loss: 0.8282 - val_accuracy: 0.6935 - val_loss: 1.0176
Epoch 40/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 459ms/step - accuracy: 0.7432 - loss: 0.8211 - val_accuracy: 0.6823 - val_loss: 1.0339
Epoch 41/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 448ms/step - accuracy: 0.7507 - loss: 0.8297 - val_accuracy: 0.6734 - val_loss: 1.0548
Epoch 42/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 50s 460ms/step - accuracy: 0.7565 - loss: 0.8118 - val_accuracy: 0.7025 - val_loss: 1.0122
Epoch 43/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 448ms/step - accuracy: 0.7519 - loss: 0.8252 - val_accuracy: 0.6823 - val_loss: 0.9846
Epoch 44/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 416ms/step - accuracy: 0.7675 - loss: 0.7932 - val_accuracy: 0.6846 - val_loss: 1.1243
Epoch 45/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 400ms/step - accuracy: 0.7493 - loss: 0.8189 - val_accuracy: 0.7002 - val_loss: 0.9900
Epoch 46/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 400ms/step - accuracy: 0.7606 - loss: 0.8017 - val_accuracy: 0.6846 - val_loss: 1.0223
Epoch 47/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 403ms/step - accuracy: 0.7620 - loss: 0.7881 - val_accuracy: 0.6779 - val_loss: 1.0541
Epoch 48/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 414ms/step - accuracy: 0.7713 - loss: 0.7843 - val_accuracy: 0.6846 - val_loss: 1.0771
Epoch 49/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 442ms/step - accuracy: 0.7565 - loss: 0.7918 - val_accuracy: 0.6823 - val_loss: 1.0008
Epoch 50/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 401ms/step - accuracy: 0.7690 - loss: 0.7833 - val_accuracy: 0.6823 - val_loss: 1.0618
Epoch 51/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 430ms/step - accuracy: 0.7710 - loss: 0.7683 - val_accuracy: 0.7136 - val_loss: 0.9734
Epoch 52/100
```

```
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 410ms/step - accuracy: 0.7698 - loss: 0.7809 - val_accuracy: 0.7069 - val_loss: 0.9741
Epoch 53/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 415ms/step - accuracy: 0.7797 - loss: 0.7602 - val_accuracy: 0.6980 - val_loss: 0.9850
Epoch 54/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 423ms/step - accuracy: 0.7721 - loss: 0.7780 - val_accuracy: 0.6711 - val_loss: 1.0481
Epoch 55/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 409ms/step - accuracy: 0.7724 - loss: 0.7732 - val_accuracy: 0.7025 - val_loss: 0.9811
Epoch 56/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 401ms/step - accuracy: 0.7803 - loss: 0.7695 - val_accuracy: 0.6801 - val_loss: 1.1091
Epoch 57/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 400ms/step - accuracy: 0.7834 - loss: 0.7617 - val_accuracy: 0.7069 - val_loss: 1.0288
Epoch 58/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 399ms/step - accuracy: 0.7860 - loss: 0.7354 - val_accuracy: 0.6711 - val_loss: 1.0767
Epoch 59/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 410ms/step - accuracy: 0.7829 - loss: 0.7443 - val_accuracy: 0.6779 - val_loss: 1.0623
Epoch 60/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 416ms/step - accuracy: 0.7887 - loss: 0.7409 - val_accuracy: 0.6957 - val_loss: 1.0017
Epoch 61/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 410ms/step - accuracy: 0.7875 - loss: 0.7463 - val_accuracy: 0.6801 - val_loss: 0.9966
Epoch 62/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 414ms/step - accuracy: 0.7915 - loss: 0.7218 - val_accuracy: 0.6913 - val_loss: 1.0248
Epoch 63/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 427ms/step - accuracy: 0.7933 - loss: 0.7302 - val_accuracy: 0.6823 - val_loss: 0.9767
Epoch 64/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 427ms/step - accuracy: 0.7936 - loss: 0.7395 - val_accuracy: 0.6868 - val_loss: 1.1010
Epoch 65/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 423ms/step - accuracy: 0.7881 - loss: 0.7452 - val_accuracy: 0.6801 - val_loss: 1.0614
Epoch 66/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 427ms/step - accuracy: 0.7817 - loss: 0.7532 - val_accuracy: 0.6756 - val_loss: 1.0325
Epoch 67/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 432ms/step - accuracy: 0.7855 - loss: 0.7551 - val_accuracy: 0.6890 - val_loss: 1.0177
Epoch 68/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 417ms/step - accuracy: 0.7924 - loss: 0.7218 - val_accuracy: 0.6913 - val_loss: 1.0303
Epoch 69/100
```

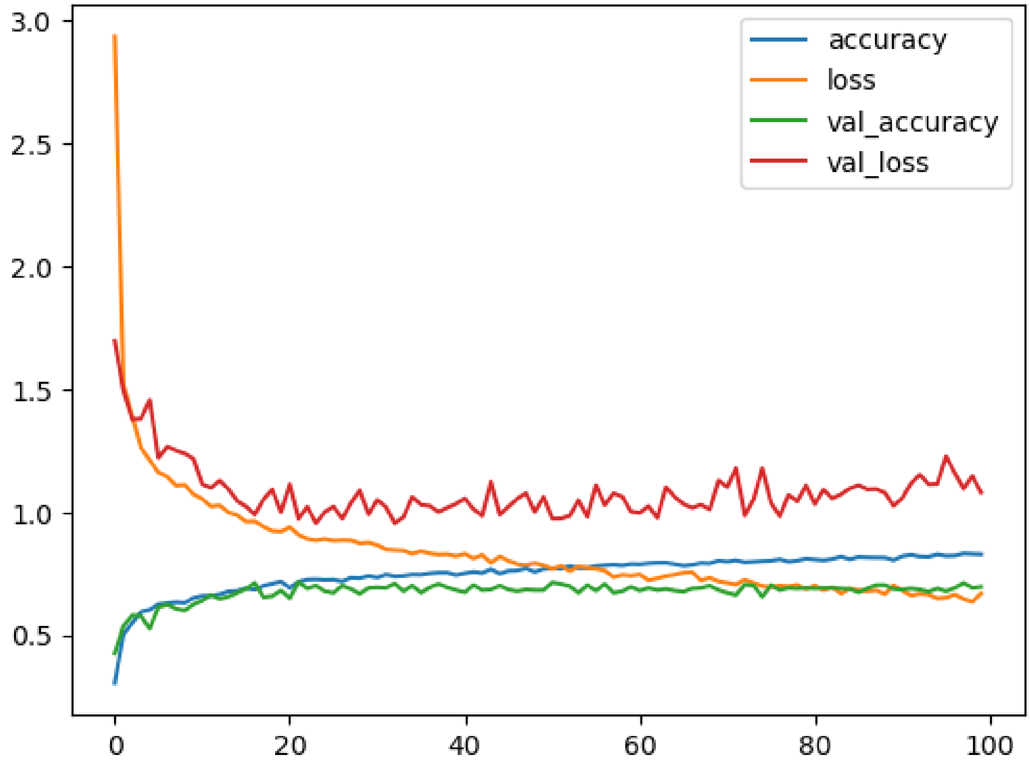
```
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 410ms/step - accuracy: 0.7910 - loss: 0.7324 - val_accuracy: 0.7002 - val_loss: 1.0103
Epoch 70/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 413ms/step - accuracy: 0.8023 - loss: 0.7177 - val_accuracy: 0.6823 - val_loss: 1.1283
Epoch 71/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 423ms/step - accuracy: 0.7985 - loss: 0.7115 - val_accuracy: 0.6711 - val_loss: 1.1024
Epoch 72/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 418ms/step - accuracy: 0.8037 - loss: 0.7051 - val_accuracy: 0.6622 - val_loss: 1.1789
Epoch 73/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 423ms/step - accuracy: 0.7953 - loss: 0.7243 - val_accuracy: 0.7047 - val_loss: 0.9854
Epoch 74/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 49s 449ms/step - accuracy: 0.7976 - loss: 0.7112 - val_accuracy: 0.7002 - val_loss: 1.0520
Epoch 75/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 440ms/step - accuracy: 0.8002 - loss: 0.6985 - val_accuracy: 0.6555 - val_loss: 1.1783
Epoch 76/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 431ms/step - accuracy: 0.8020 - loss: 0.6946 - val_accuracy: 0.7025 - val_loss: 1.0372
Epoch 77/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 444ms/step - accuracy: 0.8078 - loss: 0.7004 - val_accuracy: 0.6823 - val_loss: 0.9834
Epoch 78/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 56s 517ms/step - accuracy: 0.7982 - loss: 0.6973 - val_accuracy: 0.6913 - val_loss: 1.0696
Epoch 79/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 71s 414ms/step - accuracy: 0.8023 - loss: 0.7032 - val_accuracy: 0.6890 - val_loss: 1.0435
Epoch 80/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 49s 453ms/step - accuracy: 0.8107 - loss: 0.6859 - val_accuracy: 0.6913 - val_loss: 1.1095
Epoch 81/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 446ms/step - accuracy: 0.8069 - loss: 0.7009 - val_accuracy: 0.6890 - val_loss: 1.0326
Epoch 82/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 441ms/step - accuracy: 0.8037 - loss: 0.6818 - val_accuracy: 0.6913 - val_loss: 1.0908
Epoch 83/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 424ms/step - accuracy: 0.8089 - loss: 0.6938 - val_accuracy: 0.6913 - val_loss: 1.0554
Epoch 84/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 428ms/step - accuracy: 0.8188 - loss: 0.6680 - val_accuracy: 0.6890 - val_loss: 1.0727
Epoch 85/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 436ms/step - accuracy: 0.8078 - loss: 0.6884 - val_accuracy: 0.6868 - val_loss: 1.0963
Epoch 86/100
```

```
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 422ms/step - accuracy: 0.8173 - loss: 0.6822 - val_accuracy: 0.6734 - val_loss: 1.1093
Epoch 87/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 48s 442ms/step - accuracy: 0.8156 - loss: 0.6777 - val_accuracy: 0.6890 - val_loss: 1.0926
Epoch 88/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 45s 417ms/step - accuracy: 0.8150 - loss: 0.6812 - val_accuracy: 0.7025 - val_loss: 1.0941
Epoch 89/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 392ms/step - accuracy: 0.8144 - loss: 0.6660 - val_accuracy: 0.7025 - val_loss: 1.0789
Epoch 90/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 42s 386ms/step - accuracy: 0.8046 - loss: 0.7013 - val_accuracy: 0.6868 - val_loss: 1.0255
Epoch 91/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 43s 394ms/step - accuracy: 0.8202 - loss: 0.6802 - val_accuracy: 0.6846 - val_loss: 1.0572
Epoch 92/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 44s 408ms/step - accuracy: 0.8263 - loss: 0.6604 - val_accuracy: 0.6890 - val_loss: 1.1171
Epoch 93/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 427ms/step - accuracy: 0.8191 - loss: 0.6671 - val_accuracy: 0.6846 - val_loss: 1.1518
Epoch 94/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 428ms/step - accuracy: 0.8176 - loss: 0.6641 - val_accuracy: 0.6756 - val_loss: 1.1131
Epoch 95/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 426ms/step - accuracy: 0.8277 - loss: 0.6486 - val_accuracy: 0.6890 - val_loss: 1.1148
Epoch 96/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 46s 429ms/step - accuracy: 0.8217 - loss: 0.6514 - val_accuracy: 0.6779 - val_loss: 1.2269
Epoch 97/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 431ms/step - accuracy: 0.8225 - loss: 0.6641 - val_accuracy: 0.6913 - val_loss: 1.1588
Epoch 98/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 433ms/step - accuracy: 0.8315 - loss: 0.6457 - val_accuracy: 0.7114 - val_loss: 1.0948
Epoch 99/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 439ms/step - accuracy: 0.8295 - loss: 0.6354 - val_accuracy: 0.6913 - val_loss: 1.1464
Epoch 100/100
108/108 ━━━━━━━━━━━━━━━━━━━━ 47s 429ms/step - accuracy: 0.8280 - loss: 0.6680 - val_accuracy: 0.6957 - val_loss: 1.0796
```

Построим график потерь и точности

```python
#model.save("C:\\Users\\Admin\\Desktop\\GUAP\\IAD\\lab3\\classifier.keras")

import pandas as pd
import matplotlib.pyplot as plt
pd.DataFrame(history.history).plot()
plt.show()
```



Проверим точность и потери на тестовом наборе данных

```
model.evaluate(test_ds)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.7043 - loss:
1.0715
[1.071543574333191, 0.7043269276618958]
```

Точность на тестовой выборке сравнима с точностью на валидационной выборке

2. Трансферное обучение

Создадим сверточное ядро

```
import keras

#
conv_base = keras.applications.MobileNetV2(
    input_shape=(150,150,3),
    alpha=1.0,
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    pooling=None,
)

conv_base.trainable = False

conv_base.summary()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_30704\3791830724.py:5: UserWarning:
`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192,
224]. Weights for input shape (224, 224) will be loaded as the default.
  conv_base = keras.applications.MobileNetV2(
Model: "mobilenetv2_1.00_224"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_5 (InputLayer)	(None, 150, 150, 3)	0	-
Conv1 (Conv2D)	(None, 75, 75, 32)	864	input_layer_5[0]…
bn_Conv1 (BatchNormalizatio…	(None, 75, 75, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 75, 75, 32)	0	bn_Conv1[0][0]
expanded_conv_dept… (DepthwiseConv2D)	(None, 75, 75, 32)	288	Conv1_relu[0][0]
expanded_conv_dept… (BatchNormalizatio…	(None, 75, 75, 32)	128	expanded_conv_de…
expanded_conv_dept… (ReLU)	(None, 75, 75, 32)	0	expanded_conv_de…
expanded_conv_proj… (Conv2D)	(None, 75, 75, 16)	512	expanded_conv_de…
expanded_conv_proj… (BatchNormalizatio…	(None, 75, 75, 16)	64	expanded_conv_pr…
block_1_expand (Conv2D)	(None, 75, 75, 96)	1,536	expanded_conv_pr…
block_1_expand_BN (BatchNormalizatio…	(None, 75, 75, 96)	384	block_1_expand[0…
block_1_expand_relu (ReLU)	(None, 75, 75, 96)	0	block_1_expand_B…
block_1_pad (ZeroPadding2D)	(None, 77, 77, 96)	0	block_1_expand_r…

block_1_depthwise (DepthwiseConv2D)	(None, 38, 38, 96)	864	block_1_pad[0][0]
block_1_depthwise_… (BatchNormalizatio…	(None, 38, 38, 96)	384	block_1_depthwis…
block_1_depthwise_… (ReLU)	(None, 38, 38, 96)	0	block_1_depthwis…
block_1_project (Conv2D)	(None, 38, 38, 24)	2,304	block_1_depthwis…
block_1_project_BN (BatchNormalizatio…	(None, 38, 38, 24)	96	block_1_project[…
block_2_expand (Conv2D)	(None, 38, 38, 144)	3,456	block_1_project_…
block_2_expand_BN (BatchNormalizatio…	(None, 38, 38, 144)	576	block_2_expand[0…
block_2_expand_relu (ReLU)	(None, 38, 38, 144)	0	block_2_expand_B…
block_2_depthwise (DepthwiseConv2D)	(None, 38, 38, 144)	1,296	block_2_expand_r…
block_2_depthwise_… (BatchNormalizatio…	(None, 38, 38, 144)	576	block_2_depthwis…
block_2_depthwise_… (ReLU)	(None, 38, 38, 144)	0	block_2_depthwis…
block_2_project (Conv2D)	(None, 38, 38, 24)	3,456	block_2_depthwis…
block_2_project_BN (BatchNormalizatio…	(None, 38, 38, 24)	96	block_2_project[…
block_2_add (Add)	(None, 38, 38,	0	block_1_project_…

	24)		block_2_project_…
block_3_expand (Conv2D)	(None, 38, 38, 144)	3,456	block_2_add[0][0]
block_3_expand_BN (BatchNormalizatio…	(None, 38, 38, 144)	576	block_3_expand[0…
block_3_expand_relu (ReLU)	(None, 38, 38, 144)	0	block_3_expand_B…
block_3_pad (ZeroPadding2D)	(None, 39, 39, 144)	0	block_3_expand_r…
block_3_depthwise (DepthwiseConv2D)	(None, 19, 19, 144)	1,296	block_3_pad[0][0]
block_3_depthwise_… (BatchNormalizatio…	(None, 19, 19, 144)	576	block_3_depthwis…
block_3_depthwise_… (ReLU)	(None, 19, 19, 144)	0	block_3_depthwis…
block_3_project (Conv2D)	(None, 19, 19, 32)	4,608	block_3_depthwis…
block_3_project_BN (BatchNormalizatio…	(None, 19, 19, 32)	128	block_3_project[…
block_4_expand (Conv2D)	(None, 19, 19, 192)	6,144	block_3_project_…
block_4_expand_BN (BatchNormalizatio…	(None, 19, 19, 192)	768	block_4_expand[0…
block_4_expand_relu (ReLU)	(None, 19, 19, 192)	0	block_4_expand_B…
block_4_depthwise (DepthwiseConv2D)	(None, 19, 19, 192)	1,728	block_4_expand_r…

block_4_depthwise_… (BatchNormalizatio…	(None, 19, 19, 192)	768	block_4_depthwis…
block_4_depthwise_… (ReLU)	(None, 19, 19, 192)	0	block_4_depthwis…
block_4_project (Conv2D)	(None, 19, 19, 32)	6,144	block_4_depthwis…
block_4_project_BN (BatchNormalizatio…	(None, 19, 19, 32)	128	block_4_project[…
block_4_add (Add)	(None, 19, 19, 32)	0	block_3_project_… block_4_project_…
block_5_expand (Conv2D)	(None, 19, 19, 192)	6,144	block_4_add[0][0]
block_5_expand_BN (BatchNormalizatio…	(None, 19, 19, 192)	768	block_5_expand[0…
block_5_expand_relu (ReLU)	(None, 19, 19, 192)	0	block_5_expand_B…
block_5_depthwise (DepthwiseConv2D)	(None, 19, 19, 192)	1,728	block_5_expand_r…
block_5_depthwise_… (BatchNormalizatio…	(None, 19, 19, 192)	768	block_5_depthwis…
block_5_depthwise_… (ReLU)	(None, 19, 19, 192)	0	block_5_depthwis…
block_5_project (Conv2D)	(None, 19, 19, 32)	6,144	block_5_depthwis…
block_5_project_BN (BatchNormalizatio…	(None, 19, 19, 32)	128	block_5_project[…
block_5_add (Add)	(None, 19, 19, 32)	0	block_4_add[0][0… block_5_project_…

block_6_expand (Conv2D)	(None, 19, 19, 192)	6,144	block_5_add[0][0]
block_6_expand_BN (BatchNormalizatio…	(None, 19, 19, 192)	768	block_6_expand[0…
block_6_expand_relu (ReLU)	(None, 19, 19, 192)	0	block_6_expand_B…
block_6_pad (ZeroPadding2D)	(None, 21, 21, 192)	0	block_6_expand_r…
block_6_depthwise (DepthwiseConv2D)	(None, 10, 10, 192)	1,728	block_6_pad[0][0]
block_6_depthwise_… (BatchNormalizatio…	(None, 10, 10, 192)	768	block_6_depthwis…
block_6_depthwise_… (ReLU)	(None, 10, 10, 192)	0	block_6_depthwis…
block_6_project (Conv2D)	(None, 10, 10, 64)	12,288	block_6_depthwis…
block_6_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_6_project[…
block_7_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_6_project_…
block_7_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_7_expand[0…
block_7_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_7_expand_B…
block_7_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_7_expand_r…
block_7_depthwise_…	(None, 10, 10,	1,536	block_7_depthwis…

(BatchNormalizatio…	384)		
block_7_depthwise_… (ReLU)	(None, 10, 10, 384)	0	block_7_depthwis…
block_7_project (Conv2D)	(None, 10, 10, 64)	24,576	block_7_depthwis…
block_7_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_7_project[…
block_7_add (Add)	(None, 10, 10, 64)	0	block_6_project_… block_7_project_…
block_8_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_7_add[0][0]
block_8_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_8_expand[0…
block_8_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_8_expand_B…
block_8_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_8_expand_r…
block_8_depthwise_… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_8_depthwis…
block_8_depthwise_… (ReLU)	(None, 10, 10, 384)	0	block_8_depthwis…
block_8_project (Conv2D)	(None, 10, 10, 64)	24,576	block_8_depthwis…
block_8_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_8_project[…
block_8_add (Add)	(None, 10, 10, 64)	0	block_7_add[0][0… block_8_project_…

block_9_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_8_add[0][0]
block_9_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_9_expand[0…
block_9_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_9_expand_B…
block_9_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_9_expand_r…
block_9_depthwise_… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_9_depthwis…
block_9_depthwise_… (ReLU)	(None, 10, 10, 384)	0	block_9_depthwis…
block_9_project (Conv2D)	(None, 10, 10, 64)	24,576	block_9_depthwis…
block_9_project_BN (BatchNormalizatio…	(None, 10, 10, 64)	256	block_9_project[…
block_9_add (Add)	(None, 10, 10, 64)	0	block_8_add[0][0… block_9_project_…
block_10_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_9_add[0][0]
block_10_expand_BN (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_10_expand[…
block_10_expand_re… (ReLU)	(None, 10, 10, 384)	0	block_10_expand_…
block_10_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_10_expand_…
block_10_depthwise… (BatchNormalizatio…	(None, 10, 10, 384)	1,536	block_10_depthwi…

block_10_depthwise… (ReLU)	(None, 10, 10, 384)	0	block_10_depthwi…
block_10_project (Conv2D)	(None, 10, 10, 96)	36,864	block_10_depthwi…
block_10_project_BN (BatchNormalizatio…	(None, 10, 10, 96)	384	block_10_project…
block_11_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_10_project…
block_11_expand_BN (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_11_expand[…
block_11_expand_re… (ReLU)	(None, 10, 10, 576)	0	block_11_expand_…
block_11_depthwise (DepthwiseConv2D)	(None, 10, 10, 576)	5,184	block_11_expand_…
block_11_depthwise… (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_11_depthwi…
block_11_depthwise… (ReLU)	(None, 10, 10, 576)	0	block_11_depthwi…
block_11_project (Conv2D)	(None, 10, 10, 96)	55,296	block_11_depthwi…
block_11_project_BN (BatchNormalizatio…	(None, 10, 10, 96)	384	block_11_project…
block_11_add (Add)	(None, 10, 10, 96)	0	block_10_project… block_11_project…
block_12_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_11_add[0][…
block_12_expand_BN	(None, 10, 10,	2,304	block_12_expand[…

(BatchNormalizatio…	576)		
block_12_expand_re… (ReLU)	(None, 10, 10, 576)	0	block_12_expand_…
block_12_depthwise (DepthwiseConv2D)	(None, 10, 10, 576)	5,184	block_12_expand_…
block_12_depthwise… (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_12_depthwi…
block_12_depthwise… (ReLU)	(None, 10, 10, 576)	0	block_12_depthwi…
block_12_project (Conv2D)	(None, 10, 10, 96)	55,296	block_12_depthwi…
block_12_project_BN (BatchNormalizatio…	(None, 10, 10, 96)	384	block_12_project…
block_12_add (Add)	(None, 10, 10, 96)	0	block_11_add[0][… block_12_project…
block_13_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_12_add[0][…
block_13_expand_BN (BatchNormalizatio…	(None, 10, 10, 576)	2,304	block_13_expand[…
block_13_expand_re… (ReLU)	(None, 10, 10, 576)	0	block_13_expand_…
block_13_pad (ZeroPadding2D)	(None, 11, 11, 576)	0	block_13_expand_…
block_13_depthwise (DepthwiseConv2D)	(None, 5, 5, 576)	5,184	block_13_pad[0][…
block_13_depthwise… (BatchNormalizatio…	(None, 5, 5, 576)	2,304	block_13_depthwi…

block_13_depthwise… (ReLU)	(None, 5, 5, 576)	0	block_13_depthwi…
block_13_project (Conv2D)	(None, 5, 5, 160)	92,160	block_13_depthwi…
block_13_project_BN (BatchNormalizatio…	(None, 5, 5, 160)	640	block_13_project…
block_14_expand (Conv2D)	(None, 5, 5, 960)	153,600	block_13_project…
block_14_expand_BN (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_14_expand[…
block_14_expand_re… (ReLU)	(None, 5, 5, 960)	0	block_14_expand_…
block_14_depthwise (DepthwiseConv2D)	(None, 5, 5, 960)	8,640	block_14_expand_…
block_14_depthwise… (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_14_depthwi…
block_14_depthwise… (ReLU)	(None, 5, 5, 960)	0	block_14_depthwi…
block_14_project (Conv2D)	(None, 5, 5, 160)	153,600	block_14_depthwi…
block_14_project_BN (BatchNormalizatio…	(None, 5, 5, 160)	640	block_14_project…
block_14_add (Add)	(None, 5, 5, 160)	0	block_13_project… block_14_project…
block_15_expand (Conv2D)	(None, 5, 5, 960)	153,600	block_14_add[0][…
block_15_expand_BN (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_15_expand[…

block_15_expand_re… (ReLU)	(None, 5, 5, 960)	0	block_15_expand_…
block_15_depthwise (DepthwiseConv2D)	(None, 5, 5, 960)	8,640	block_15_expand_…
block_15_depthwise… (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_15_depthwi…
block_15_depthwise… (ReLU)	(None, 5, 5, 960)	0	block_15_depthwi…
block_15_project (Conv2D)	(None, 5, 5, 160)	153,600	block_15_depthwi…
block_15_project_BN (BatchNormalizatio…	(None, 5, 5, 160)	640	block_15_project…
block_15_add (Add)	(None, 5, 5, 160)	0	block_14_add[0][… block_15_project…
block_16_expand (Conv2D)	(None, 5, 5, 960)	153,600	block_15_add[0][…
block_16_expand_BN (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_16_expand[…
block_16_expand_re… (ReLU)	(None, 5, 5, 960)	0	block_16_expand_…
block_16_depthwise (DepthwiseConv2D)	(None, 5, 5, 960)	8,640	block_16_expand_…
block_16_depthwise… (BatchNormalizatio…	(None, 5, 5, 960)	3,840	block_16_depthwi…
block_16_depthwise… (ReLU)	(None, 5, 5, 960)	0	block_16_depthwi…
block_16_project	(None, 5, 5, 320)	307,200	block_16_depthwi…

```
│ (Conv2D)                │                      │               │                        │
├─────────────────────────┼──────────────────────┼───────────────┼────────────────────────┤
│ block_16_project_BN     │ (None, 5, 5, 320)    │         1,280 │ block_16_project…      │
│ (BatchNormalizatio…     │                      │               │                        │
├─────────────────────────┼──────────────────────┼───────────────┼────────────────────────┤
│ Conv_1 (Conv2D)         │ (None, 5, 5,         │       409,600 │ block_16_project…      │
│                         │ 1280)                │               │                        │
├─────────────────────────┼──────────────────────┼───────────────┼────────────────────────┤
│ Conv_1_bn               │ (None, 5, 5,         │         5,120 │ Conv_1[0][0]           │
│ (BatchNormalizatio…     │ 1280)                │               │                        │
├─────────────────────────┼──────────────────────┼───────────────┼────────────────────────┤
│ out_relu (ReLU)         │ (None, 5, 5,         │             0 │ Conv_1_bn[0][0]        │
│                         │ 1280)                │               │                        │
└─────────────────────────┴──────────────────────┴───────────────┴────────────────────────┘
 Total params: 2,257,984 (8.61 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 2,257,984 (8.61 MB)
```

Соберем пайплайн

```
from keras.applications import mobilenet_v2

pred_model = models.Sequential([
    # входные изображения
    layers.Input(shape=(150, 150, 3)),

    # аугментация (должно работать только при обучении)
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomBrightness(0.1),

    # препроцессинг для сверточного ядра
    layers.Lambda(mobilenet_v2.preprocess_input, name="preprocessing"),

    # сверточное ядро
    conv_base,

    # полносвязный классификатор
    layers.Flatten(),
    layers.Dense(256),
    layers.Dropout(0.5),
    layers.Dense(5, activation="softmax")
])

pred_model.compile(
    optimizer=optimizers.Adam(0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

pred_model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
random_flip_4 (RandomFlip)	(None, 150, 150, 3)	0
random_rotation_4 (RandomRotation)	(None, 150, 150, 3)	0
random_zoom_4 (RandomZoom)	(None, 150, 150, 3)	0
random_brightness_4 (RandomBrightness)	(None, 150, 150, 3)	0
preprocessing (Lambda)	(None, 150, 150, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 5, 5, 1280)	2,257,984
flatten_4 (Flatten)	(None, 32000)	0
dense_8 (Dense)	(None, 256)	8,192,256
dropout_3 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 5)	1,285

Total params: 10,451,525 (39.87 MB)

Trainable params: 8,193,541 (31.26 MB)

Non-trainable params: 2,257,984 (8.61 MB)

Обучение

```
import keras.callbacks as clbks

callbacks = [
    clbks.EarlyStopping(monitor="val_loss", patience=20),
    clbks.ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=7)
]

pred_history = pred_model.fit(
    train_ds,
    epochs=80,
    validation_data=val_ds,
    callbacks=callbacks
)
```

```
Epoch 1/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 42s 346ms/step - accuracy: 0.7082 - loss: 19.0873 - val_accuracy: 0.8210 - val_loss: 9.2984 - learning_rate: 0.0010
Epoch 2/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 323ms/step - accuracy: 0.8075 - loss: 10.9645 - val_accuracy: 0.7539 - val_loss: 15.2330 - learning_rate: 0.0010
Epoch 3/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.8309 - loss: 8.3308 - val_accuracy: 0.7987 - val_loss: 11.4608 - learning_rate: 0.0010
Epoch 4/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.8471 - loss: 6.7108 - val_accuracy: 0.8054 - val_loss: 11.8353 - learning_rate: 0.0010
Epoch 5/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.8772 - loss: 4.8581 - val_accuracy: 0.8300 - val_loss: 7.8194 - learning_rate: 0.0010
Epoch 6/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.8830 - loss: 3.8715 - val_accuracy: 0.8389 - val_loss: 6.8386 - learning_rate: 0.0010
Epoch 7/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 327ms/step - accuracy: 0.8813 - loss: 3.9301 - val_accuracy: 0.8345 - val_loss: 6.6347 - learning_rate: 0.0010
Epoch 8/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.8920 - loss: 3.2645 - val_accuracy: 0.8143 - val_loss: 8.2810 - learning_rate: 0.0010
Epoch 9/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 329ms/step - accuracy: 0.8972 - loss: 2.7899 - val_accuracy: 0.8591 - val_loss: 5.9028 - learning_rate: 0.0010
Epoch 10/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 328ms/step - accuracy: 0.8935 - loss: 2.2407 - val_accuracy: 0.8479 - val_loss: 6.1850 - learning_rate: 0.0010
Epoch 11/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 328ms/step - accuracy: 0.9039 - loss: 2.3780 - val_accuracy: 0.8345 - val_loss: 4.5595 - learning_rate: 0.0010
Epoch 12/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 330ms/step - accuracy: 0.9068 - loss: 2.1162 - val_accuracy: 0.8523 - val_loss: 4.7103 - learning_rate: 0.0010
Epoch 13/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 328ms/step - accuracy: 0.9201 - loss: 1.4573 - val_accuracy: 0.8501 - val_loss: 3.8198 - learning_rate: 0.0010
Epoch 14/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 325ms/step - accuracy: 0.9221 - loss: 1.4595 - val_accuracy: 0.8143 - val_loss: 5.4844 - learning_rate: 0.0010
Epoch 15/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 337ms/step - accuracy: 0.9068 - loss: 1.5094 - val_accuracy: 0.8434 - val_loss: 4.1543 - learning_rate: 0.0010
Epoch 16/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 37s 346ms/step - accuracy: 0.9192 - loss: 1.2868 - val_accuracy: 0.8345 - val_loss: 3.9649 - learning_rate: 0.0010
Epoch 17/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 39s 356ms/step - accuracy: 0.9215 - loss: 1.1889 - val_accuracy: 0.8188 - val_loss: 4.5844 - learning_rate: 0.0010
Epoch 18/80
```

```
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 330ms/step - accuracy: 0.9328 - loss: 0.9885 - val_accuracy: 0.8098 - val_loss: 3.7648 - learning_rate: 0.0010
Epoch 19/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 37s 339ms/step - accuracy: 0.9241 - loss: 0.8724 - val_accuracy: 0.8412 - val_loss: 3.1014 - learning_rate: 0.0010
Epoch 20/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 39s 317ms/step - accuracy: 0.9241 - loss: 0.8774 - val_accuracy: 0.8345 - val_loss: 3.6524 - learning_rate: 0.0010
Epoch 21/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 328ms/step - accuracy: 0.9268 - loss: 0.8637 - val_accuracy: 0.7942 - val_loss: 3.8808 - learning_rate: 0.0010
Epoch 22/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.9288 - loss: 0.7395 - val_accuracy: 0.8456 - val_loss: 3.0943 - learning_rate: 0.0010
Epoch 23/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 318ms/step - accuracy: 0.9337 - loss: 0.7165 - val_accuracy: 0.8345 - val_loss: 2.7981 - learning_rate: 0.0010
Epoch 24/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 317ms/step - accuracy: 0.9392 - loss: 0.7247 - val_accuracy: 0.8658 - val_loss: 2.6718 - learning_rate: 0.0010
Epoch 25/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 318ms/step - accuracy: 0.9288 - loss: 0.6529 - val_accuracy: 0.8658 - val_loss: 2.1555 - learning_rate: 0.0010
Epoch 26/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 322ms/step - accuracy: 0.9282 - loss: 0.7487 - val_accuracy: 0.8255 - val_loss: 3.6476 - learning_rate: 0.0010
Epoch 27/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 40s 313ms/step - accuracy: 0.9328 - loss: 0.7444 - val_accuracy: 0.8300 - val_loss: 3.4678 - learning_rate: 0.0010
Epoch 28/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 317ms/step - accuracy: 0.9383 - loss: 0.6995 - val_accuracy: 0.8345 - val_loss: 3.0757 - learning_rate: 0.0010
Epoch 29/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.9221 - loss: 0.8586 - val_accuracy: 0.8143 - val_loss: 3.1674 - learning_rate: 0.0010
Epoch 30/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 330ms/step - accuracy: 0.9334 - loss: 0.8246 - val_accuracy: 0.8680 - val_loss: 2.9245 - learning_rate: 0.0010
Epoch 31/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 39s 363ms/step - accuracy: 0.9343 - loss: 0.7341 - val_accuracy: 0.8680 - val_loss: 2.5302 - learning_rate: 0.0010
Epoch 32/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 334ms/step - accuracy: 0.9305 - loss: 0.7021 - val_accuracy: 0.8121 - val_loss: 3.0588 - learning_rate: 0.0010
Epoch 33/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 323ms/step - accuracy: 0.9433 - loss: 0.6392 - val_accuracy: 0.8501 - val_loss: 2.4704 - learning_rate: 2.0000e-04
Epoch 34/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 334ms/step - accuracy: 0.9574 - loss: 0.3710 - val_accuracy: 0.8412 - val_loss: 2.7464 - learning_rate: 2.0000e-04
Epoch 35/80
```

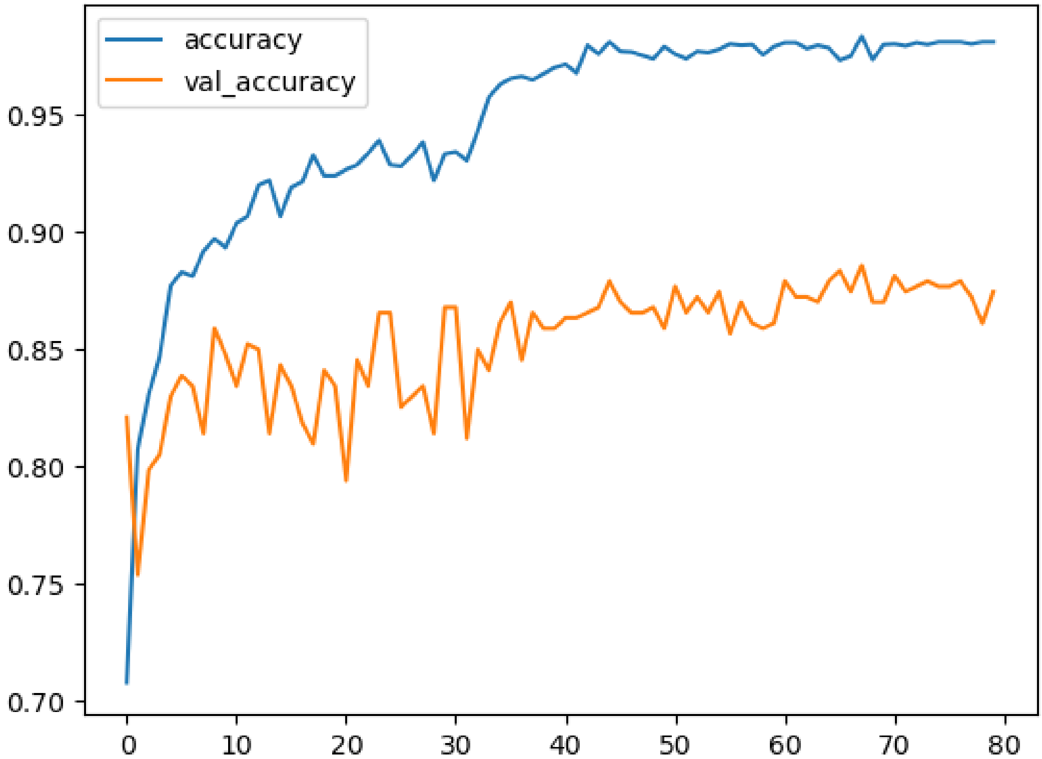
```
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 328ms/step - accuracy: 0.9629 - loss: 0.2962 - val_accuracy: 0.8613 - val_loss: 2.4116 - learning_rate: 2.0000e-04
Epoch 36/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 317ms/step - accuracy: 0.9655 - loss: 0.2806 - val_accuracy: 0.8702 - val_loss: 2.2739 - learning_rate: 2.0000e-04
Epoch 37/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 317ms/step - accuracy: 0.9664 - loss: 0.2706 - val_accuracy: 0.8456 - val_loss: 2.5577 - learning_rate: 2.0000e-04
Epoch 38/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 330ms/step - accuracy: 0.9650 - loss: 0.2826 - val_accuracy: 0.8658 - val_loss: 2.2076 - learning_rate: 2.0000e-04
Epoch 39/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 40s 371ms/step - accuracy: 0.9676 - loss: 0.2390 - val_accuracy: 0.8591 - val_loss: 2.1805 - learning_rate: 2.0000e-04
Epoch 40/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 37s 342ms/step - accuracy: 0.9702 - loss: 0.2195 - val_accuracy: 0.8591 - val_loss: 2.2355 - learning_rate: 4.0000e-05
Epoch 41/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 325ms/step - accuracy: 0.9716 - loss: 0.1925 - val_accuracy: 0.8635 - val_loss: 2.2234 - learning_rate: 4.0000e-05
Epoch 42/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.9679 - loss: 0.2016 - val_accuracy: 0.8635 - val_loss: 2.2088 - learning_rate: 4.0000e-05
Epoch 43/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 322ms/step - accuracy: 0.9797 - loss: 0.1333 - val_accuracy: 0.8658 - val_loss: 2.1548 - learning_rate: 4.0000e-05
Epoch 44/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 320ms/step - accuracy: 0.9760 - loss: 0.1666 - val_accuracy: 0.8680 - val_loss: 2.1548 - learning_rate: 4.0000e-05
Epoch 45/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.9812 - loss: 0.1416 - val_accuracy: 0.8792 - val_loss: 2.1352 - learning_rate: 4.0000e-05
Epoch 46/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 320ms/step - accuracy: 0.9771 - loss: 0.1507 - val_accuracy: 0.8702 - val_loss: 2.0959 - learning_rate: 4.0000e-05
Epoch 47/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.9768 - loss: 0.1617 - val_accuracy: 0.8658 - val_loss: 2.1379 - learning_rate: 4.0000e-05
Epoch 48/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 320ms/step - accuracy: 0.9754 - loss: 0.1661 - val_accuracy: 0.8658 - val_loss: 2.0705 - learning_rate: 4.0000e-05
Epoch 49/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 323ms/step - accuracy: 0.9739 - loss: 0.1808 - val_accuracy: 0.8680 - val_loss: 2.1165 - learning_rate: 4.0000e-05
Epoch 50/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.9792 - loss: 0.1110 - val_accuracy: 0.8591 - val_loss: 2.1034 - learning_rate: 4.0000e-05
Epoch 51/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 324ms/step - accuracy: 0.9760 - loss: 0.1442 - val_accuracy: 0.8770 - val_loss: 2.0293 - learning_rate: 4.0000e-05
Epoch 52/80
```

```
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 335ms/step - accuracy: 0.9739 - loss: 0.1759 - val_accuracy: 0.8658 - val_loss: 2.1496 - learning_rate: 4.0000e-05
Epoch 53/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 41s 336ms/step - accuracy: 0.9771 - loss: 0.1371 - val_accuracy: 0.8725 - val_loss: 2.0435 - learning_rate: 4.0000e-05
Epoch 54/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 325ms/step - accuracy: 0.9765 - loss: 0.1532 - val_accuracy: 0.8658 - val_loss: 2.0651 - learning_rate: 4.0000e-05
Epoch 55/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 319ms/step - accuracy: 0.9780 - loss: 0.1262 - val_accuracy: 0.8747 - val_loss: 2.0170 - learning_rate: 4.0000e-05
Epoch 56/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.9803 - loss: 0.0943 - val_accuracy: 0.8568 - val_loss: 2.0953 - learning_rate: 4.0000e-05
Epoch 57/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 319ms/step - accuracy: 0.9797 - loss: 0.1146 - val_accuracy: 0.8702 - val_loss: 2.0438 - learning_rate: 4.0000e-05
Epoch 58/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 319ms/step - accuracy: 0.9800 - loss: 0.1135 - val_accuracy: 0.8613 - val_loss: 2.0281 - learning_rate: 4.0000e-05
Epoch 59/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 321ms/step - accuracy: 0.9757 - loss: 0.1491 - val_accuracy: 0.8591 - val_loss: 1.9966 - learning_rate: 4.0000e-05
Epoch 60/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 319ms/step - accuracy: 0.9792 - loss: 0.1129 - val_accuracy: 0.8613 - val_loss: 2.1567 - learning_rate: 4.0000e-05
Epoch 61/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 319ms/step - accuracy: 0.9809 - loss: 0.0877 - val_accuracy: 0.8792 - val_loss: 1.9471 - learning_rate: 4.0000e-05
Epoch 62/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 318ms/step - accuracy: 0.9809 - loss: 0.1411 - val_accuracy: 0.8725 - val_loss: 1.9295 - learning_rate: 4.0000e-05
Epoch 63/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 318ms/step - accuracy: 0.9783 - loss: 0.1207 - val_accuracy: 0.8725 - val_loss: 1.9344 - learning_rate: 4.0000e-05
Epoch 64/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 335ms/step - accuracy: 0.9797 - loss: 0.1102 - val_accuracy: 0.8702 - val_loss: 1.8714 - learning_rate: 4.0000e-05
Epoch 65/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 319ms/step - accuracy: 0.9786 - loss: 0.1180 - val_accuracy: 0.8792 - val_loss: 1.8578 - learning_rate: 4.0000e-05
Epoch 66/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 318ms/step - accuracy: 0.9734 - loss: 0.1552 - val_accuracy: 0.8837 - val_loss: 1.8497 - learning_rate: 4.0000e-05
Epoch 67/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 320ms/step - accuracy: 0.9751 - loss: 0.1355 - val_accuracy: 0.8747 - val_loss: 1.9520 - learning_rate: 4.0000e-05
Epoch 68/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 34s 318ms/step - accuracy: 0.9835 - loss: 0.0825 - val_accuracy: 0.8859 - val_loss: 1.7828 - learning_rate: 4.0000e-05
Epoch 69/80
```

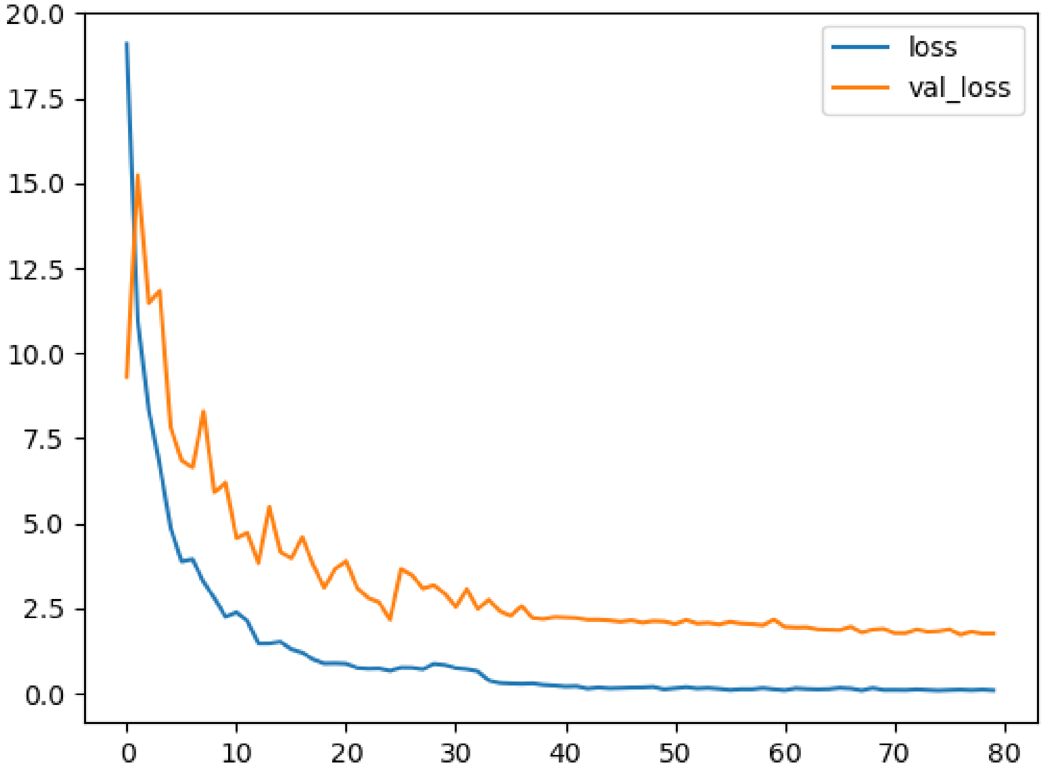
```
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 325ms/step - accuracy: 0.9737 - loss: 0.1495 - val_accuracy: 0.8702 - val_loss: 1.8643 - learning_rate: 4.0000e-05
Epoch 70/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 320ms/step - accuracy: 0.9800 - loss: 0.0946 - val_accuracy: 0.8702 - val_loss: 1.8858 - learning_rate: 4.0000e-05
Epoch 71/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 329ms/step - accuracy: 0.9803 - loss: 0.0958 - val_accuracy: 0.8814 - val_loss: 1.7642 - learning_rate: 4.0000e-05
Epoch 72/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 37s 340ms/step - accuracy: 0.9794 - loss: 0.0919 - val_accuracy: 0.8747 - val_loss: 1.7613 - learning_rate: 4.0000e-05
Epoch 73/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 333ms/step - accuracy: 0.9809 - loss: 0.1110 - val_accuracy: 0.8770 - val_loss: 1.8707 - learning_rate: 4.0000e-05
Epoch 74/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 38s 355ms/step - accuracy: 0.9800 - loss: 0.0952 - val_accuracy: 0.8792 - val_loss: 1.8031 - learning_rate: 4.0000e-05
Epoch 75/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 36s 336ms/step - accuracy: 0.9812 - loss: 0.0804 - val_accuracy: 0.8770 - val_loss: 1.8176 - learning_rate: 4.0000e-05
Epoch 76/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 322ms/step - accuracy: 0.9812 - loss: 0.0936 - val_accuracy: 0.8770 - val_loss: 1.8730 - learning_rate: 4.0000e-05
Epoch 77/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 320ms/step - accuracy: 0.9812 - loss: 0.1037 - val_accuracy: 0.8792 - val_loss: 1.7153 - learning_rate: 4.0000e-05
Epoch 78/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 35s 323ms/step - accuracy: 0.9803 - loss: 0.0905 - val_accuracy: 0.8725 - val_loss: 1.8082 - learning_rate: 4.0000e-05
Epoch 79/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 38s 348ms/step - accuracy: 0.9812 - loss: 0.1039 - val_accuracy: 0.8613 - val_loss: 1.7483 - learning_rate: 4.0000e-05
Epoch 80/80
108/108 ━━━━━━━━━━━━━━━━━━━━ 39s 359ms/step - accuracy: 0.9812 - loss: 0.0871 - val_accuracy: 0.8747 - val_loss: 1.7498 - learning_rate: 4.0000e-05
```

График потерь и точности

```
pd.DataFrame(pred_history.history)[["accuracy", "val_accuracy"]].plot()
plt.show()
```




```
pd.DataFrame(pred_history.history)[["loss", "val_loss"]].plot()
plt.show()
```



Проверка на тестовом наборе

```
#pred_model.save("C:\\Users\\Admin\\Desktop\\GUAP\\IAD\\lab3\\mobile_classifier.keras")

pred_model.evaluate(test_ds)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 202ms/step - accuracy: 0.8510 - loss:
2.1384
[2.1383538246154785, 0.8509615659713745]
```

На тестовом наборе модель показала 85% точности - примерно так же, как и на валидационной выборке.

3. Сравнение моделей

Сравним модель, обученную с нуля, и модель, обученную с использованием предобученного сверточного блока.

```
# собираем тестовые метки в один массив
test_labels = []
for x,labels in test_ds:
    test_labels.extend(labels.numpy())

test_labels = np.argmax(test_labels, axis=1)

# наименования классов
class_names = train_ds.class_names
```

Отчет о классификации для обученной с нуля модели

```
from sklearn.metrics import classification_report

model.evaluate(test_ds)
pred_labels = model.predict(test_ds)
pred_labels = np.argmax(pred_labels, axis=1)
print(classification_report(test_labels, pred_labels, target_names=class_names))
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.7043 - loss:
1.0715
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step
              precision    recall  f1-score   support

       daisy       0.77      0.71      0.74        69
   dandelion       0.80      0.66      0.72        96
        rose       0.45      0.46      0.46        54
   sunflower       0.74      0.84      0.79        83
       tulip       0.70      0.75      0.73       114

    accuracy                           0.70       416
   macro avg       0.69      0.69      0.69       416
weighted avg       0.71      0.70      0.70       416
```

Отчет о классификации для модели, обученной с использованием сверточного блока

```
from sklearn.metrics import classification_report, multilabel_confusion_matrix

pred_model.evaluate(test_ds)
pred_labels = pred_model.predict(test_ds)
pred_labels = np.argmax(pred_labels, axis=1)
print(classification_report(test_labels, pred_labels, target_names=class_names))
#print(multilabel_confusion_matrix(test_labels, pred_labels, labels=class_names))
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 169ms/step - accuracy: 0.8510 - loss:
2.1384
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 169ms/step
              precision    recall  f1-score   support
```

```
       daisy       0.82      0.87      0.85        69
   dandelion       0.98      0.83      0.90        96
        rose       0.72      0.76      0.74        54
   sunflower       0.86      0.93      0.89        83
       tulip       0.84      0.84      0.84       114

    accuracy                           0.85       416
   macro avg       0.84      0.85      0.84       416
weighted avg       0.86      0.85      0.85       416
```

Из метрик видно, что сеть, обученная с использованием предобученного блока, показала гораздо более лучшие результаты при распознавании всех классов. Интересно, что розы распознаются заметно хуже остальных классов в обеих моделях.

Также стоит отметить, что в обеих моделях наблюдается переобучение - на тренировочной выборке показатели сильно лучше, чем на валидационной и тестовой. Для датасета приведенной небольшой размерности это ожидаемо.

Вывод

В ходе выполнения третьей лабораторной работы обучено две модели многоклассовой классификации изображений. Использован датасет с изображениями цветов "alxmamaev/flowers-recognition".

Первая нейросеть - сверточная нейросеть из слоев Conv2D и MaxPooling2D. Вторая нейросеть - нейросеть с предобученным ядром MobileNetV2 с добавлением полносвязного классификатора. Ядро предобучено на наборе данных ImageNet.

С помощью сравнения метрик на тестовом наборе показано, что нейросеть с предобученным ядром имеет лучшую эффективность, что показывает оправданность использования предобученных нейросетей общего назначения при создании моделей для решения конкретных прикладных задач.