

Домашняя контрольная работа №2

Студент: Мальцев С. А.

Группа: ЕТ-313

```
In [45]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, roc_curve,
from sklearn.preprocessing import label_binarize
import os
import cv2
from pathlib import Path
import random
from tqdm import tqdm
import json

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, applications
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckPoint
from tensorflow.keras.optimizers import AdamW, Adam
from tensorflow.keras.regularizers import l2

SEED = 42
np.random.seed(SEED)
tf.random.set_seed(SEED)
random.seed(SEED)
```

Часть 1. Генерация синтетического набора данных

- **Классы:** 4 класса животных (dog, cat, racoon, capybara)
- **Фоны:** garbage, house, lawn, pond (имеют разные веса для каждого класса)
- **Количество изображений:**
 - Обучающая выборка: 600 изображений на класс
 - Валидационная выборка: 100 изображений на класс
 - Тестовая выборка: 100 изображений на класс
- **Аугментации:** масштабирование, поворот, отражение, изменение яркости/контраста, добавление шума

```
In [46]: BASE_PATH = "synthetic_dataset"
TRAIN_PATH = os.path.join(BASE_PATH, "train")
VAL_PATH = os.path.join(BASE_PATH, "validation")
TEST_PATH = os.path.join(BASE_PATH, "test")

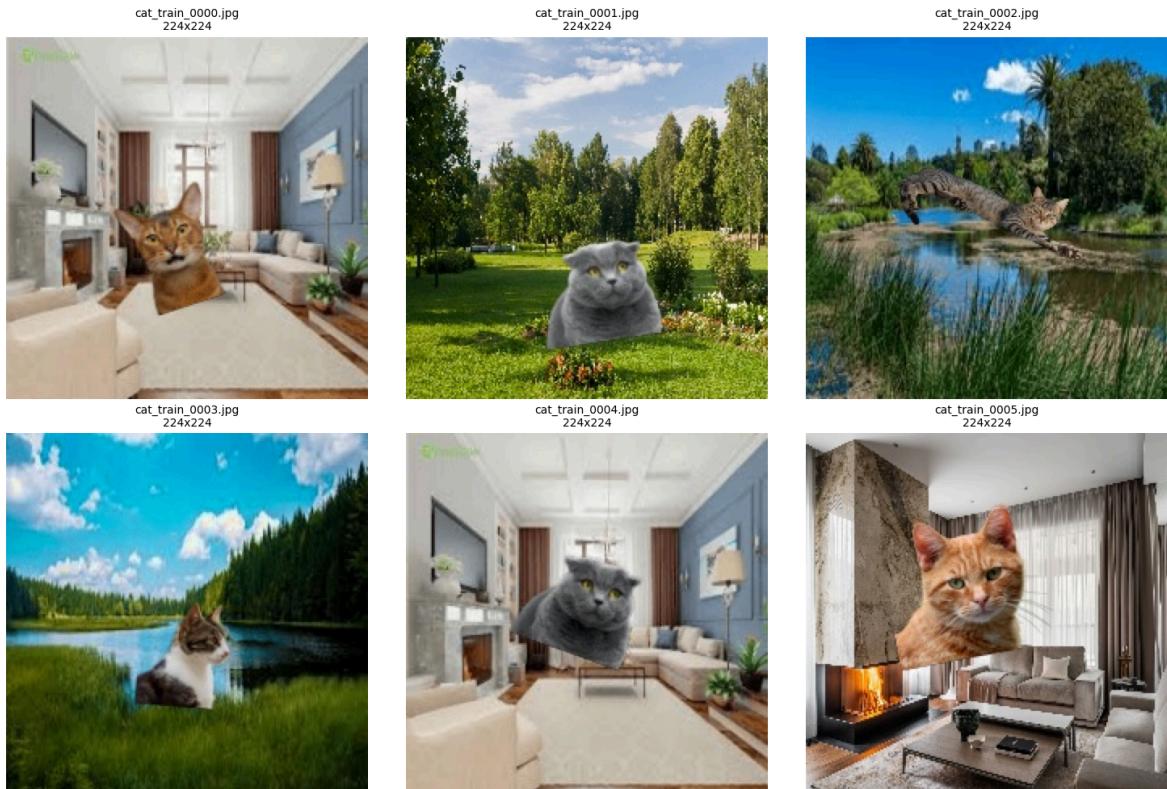
CLASSES = ['capybara', 'cat', 'dog']
NUM_CLASSES = len(CLASSES)
```

```
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
```

```
In [47]: FOLDER_PATH = os.path.join(TRAIN_PATH, "cat")
folder = Path(FOLDER_PATH)
image_files = []
for f in os.listdir(FOLDER_PATH):
    if f.lower().endswith('.jpg', '.jpeg', '.png', '.bmp')):
        image_files.append(f)
image_files = image_files[:6]

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
for i, filename in enumerate(image_files):
    if i >= 6:
        break
    img_path = os.path.join(FOLDER_PATH, filename)
    img = cv2.imread(img_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    row = i // 3
    col = i % 3

    axes[row, col].imshow(img_rgb)
    axes[row, col].set_title(f"{filename}\n{img.shape[1]}x{img.shape[0]}", fontstyle='italic')
    axes[row, col].axis('off')
plt.tight_layout()
plt.show()
```



Часть 2. Создание и дообучение нейронной сети

```
In [48]: train_datagen = ImageDataGenerator(
    rescale=1./255,
```

```

        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        brightness_range=[0.8, 1.2],
        fill_mode='nearest'
    )

val_test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True,
    seed=SEED
)

val_generator = val_test_datagen.flow_from_directory(
    VAL_PATH,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False,
    seed=SEED
)

test_generator = val_test_datagen.flow_from_directory(
    TEST_PATH,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False,
    seed=SEED
)

print(f"\nКоличество классов: {train_generator.num_classes}")
print(f"Классы: {list(train_generator.class_indices.keys())}")
print(f"Размер батча: {BATCH_SIZE}")
print(f"Размер тренировочной выборки: {train_generator.samples}")
print(f"Размер валидационной выборки: {val_generator.samples}")
print(f"Размер тестовой выборки: {test_generator.samples}")

```

Found 1800 images belonging to 3 classes.

Found 300 images belonging to 3 classes.

Found 300 images belonging to 3 classes.

Количество классов: 3

Классы: ['capybara', 'cat', 'dog']

Размер батча: 32

Размер тренировочной выборки: 1800

Размер валидационной выборки: 300

Размер тестовой выборки: 300

In [49]: `def create_transfer_learning_model(input_shape=(224, 224, 3),
 num_classes=4,
 fine_tune_layers=10):`

```

base_model = applications.ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=input_shape,
    pooling='avg'
)

base_model.trainable = False

model = keras.Sequential([
    base_model,
    layers.Dropout(0.3),
    layers.Dense(256, activation='relu',
                 kernel_regularizer=l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

if fine_tune_layers > 0:
    base_model.trainable = True
    for layer in base_model.layers[:-fine_tune_layers]:
        layer.trainable = False
    print(f"Дообучение последних {fine_tune_layers} слоев базовой модели")

return model, base_model

```

```

In [50]: model, base_model = create_transfer_learning_model(
    input_shape=(224, 224, 3),
    num_classes=NUM_CLASSES,
    fine_tune_layers=20
)

print("Архитектура модели:")
model.summary()

trainable_params = np.sum([tf.keras.backend.count_params(w) for w in model.trainable_variables])
non_trainable_params = np.sum([tf.keras.backend.count_params(w) for w in model.non_trainable_variables])
total_params = trainable_params + non_trainable_params

print("\nКоличество параметров:")
print(f" Обучаемые: {trainable_params:,}")
print(f" Необучаемые: {non_trainable_params:,}")
print(f" Всего: {total_params:,}")

```

Дообучение последних 20 слоев базовой модели

Архитектура модели:

Model: "sequential_3"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2048)	23,587,712
dropout_6 (Dropout)	(None, 2048)	0
dense_6 (Dense)	(None, 256)	524,544
batch_normalization_3 (BatchNormalization)	(None, 256)	1,024
dropout_7 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 3)	771



Total params: 24,114,051 (91.99 MB)

Trainable params: 8,408,067 (32.07 MB)

Non-trainable params: 15,705,984 (59.91 MB)

Количество параметров:

Обучаемые: 8,408,067

Необучаемые: 15,705,984

Всего: 24,114,051

Использую предобученный ResNet50 на датасете ImageNet, ResNet50 выбрал потому что самый простой выбор (также нам на семинарах ML говорили его брать, если еще не знаете что делать). 10 слоев для дообучения не так много и не так мало, плюсом данных не так много и они синтетические. Новые слои более менее выбирал случайно, dropout-ы для регуляризации, не сильно большие сверточные слои.

Часть 3. Обоснование выбора гиперпараметров

In [51]:

```
optimizer = AdamW(
    learning_rate=0.00002,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False
)

loss = 'categorical_crossentropy'
metrics = [
    keras.metrics.CategoricalAccuracy(name='accuracy'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
    # tfa.metrics.F1Score(name='f1_score'), Что-то с этим сделать надо
    keras.metrics.AUC(name='auc', multi_label=True)
]

model.compile(
```

```

        optimizer=optimizer,
        loss=loss,
        metrics=metrics
    )

    print("Модель успешно скомпилирована")
    print(f"Оптимизатор: {optimizer}")
    print(f"Learning rate: {optimizer.learning_rate.numpy()}")
    print(f"Функция потерь: {loss}")
    print(f"Метрики: {[m.name for m in metrics]})")

```

Модель успешно скомпилирована
Оптимизатор: <keras.src.optimizers.adamw.AdamW object at 0x0000022EA9CBC410>
Learning rate: 1.9999999494757503e-05
Функция потерь: categorical_crossentropy
Метрики: ['accuracy', 'precision', 'recall', 'auc']

```

In [52]: callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1,
        mode='min'
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=7,
        min_lr=1e-7,
        verbose=1,
        mode='min'
    ),
    ModelCheckpoint(
        'best_model.keras',
        monitor='val_accuracy',
        save_best_only=True,
        mode='max',
        verbose=1
    )
]

```

```

In [53]: hyperparams_table = pd.DataFrame({
    'Параметр': [
        'Оптимизатор',
        'Learning rate',
        'Функция потерь',
        'Batch size',
        'Эпохи',
        'Метрики'
    ],
    'Значение': [
        'AdamW',
        '0.00002',
        'Categorical Cross-Entropy',
        '32',
        '25 (ранняя остановка)',
        'Accuracy, Precision, Recall, F1, AUC'
    ],
    'Обоснование': [

```

```

        'Адаптивный learning rate, быстрая сходимость, стабильность',
        'Уменьшенный LR для fine-tuning, предотвращение разрушения предобученных',
        'Многоклассовая классификация с one-hot кодированием меток',
        'Баланс между скоростью обучения и стабильностью градиентов',
        'Предотвращение переобучения через раннюю остановку',
        'Метрики'

    ])

print("Таблица выбора гиперпараметров:")
display(hyperparams_table)

```

Таблица выбора гиперпараметров:

	Параметр	Значение	Обоснование
0	Оптимизатор	AdamW	Адаптивный learning rate, быстрая сходимость, ...
1	Learning rate	0.00002	Уменьшенный LR для fine-tuning, предотвращение...
2	Функция потерь	Categorical Cross-Entropy	Многоклассовая классификация с one-hot кодиров...
3	Batch size	32	Баланс между скоростью обучения и стабильность...
4	Эпохи	25 (ранняя остановка)	Предотвращение переобучения через раннюю остан...
5	Метрики	Accuracy, Precision, Recall, F1, AUC	Метрики

In [54]: EPOCHS = 25

```

print("Начало обучения модели...")
print(f"Количество эпох: {EPOCHS}")
print(f"Размер тренировочной выборки: {train_generator.samples}")
print(f"Размер валидационной выборки: {val_generator.samples}")

history = model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=val_generator,
    callbacks=callbacks,
    verbose=1,
    steps_per_epoch=len(train_generator),
    validation_steps=len(val_generator)
)

```

Начало обучения модели...

Количество эпох: 25

Размер тренировочной выборки: 1800

Размер валидационной выборки: 300

Epoch 1/25

57/57 ━━━━━━ 0s 1s/step - accuracy: 0.3994 - auc: 0.5518 - loss: 2.1311 - precision: 0.4065 - recall: 0.3526

Epoch 1: val_accuracy improved from None to 0.19667, saving model to best_model.keras

57/57 ━━━━━━ 109s 2s/step - accuracy: 0.3956 - auc: 0.5551 - loss: 2.1151 - precision: 0.4031 - recall: 0.3511 - val_accuracy: 0.1967 - val_auc: 0.4245 - val_loss: 1.6191 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 2.0000e-05

Epoch 2/25

57/57 ━━━━━━ 0s 957ms/step - accuracy: 0.3777 - auc: 0.5690 - loss: 2.0845 - precision: 0.3921 - recall: 0.3397

Epoch 2: val_accuracy improved from 0.19667 to 0.31000, saving model to best_model.keras

57/57 ━━━━━━ 64s 1s/step - accuracy: 0.3928 - auc: 0.5715 - loss: 2.0867 - precision: 0.4025 - recall: 0.3511 - val_accuracy: 0.3100 - val_auc: 0.4533 - val_loss: 1.6019 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 2.0000e-05

Epoch 3/25

57/57 ━━━━━━ 0s 908ms/step - accuracy: 0.3977 - auc: 0.5726 - loss: 2.0620 - precision: 0.4028 - recall: 0.3501

Epoch 3: val_accuracy improved from 0.31000 to 0.35333, saving model to best_model.keras

57/57 ━━━━━━ 61s 1s/step - accuracy: 0.4189 - auc: 0.6008 - loss: 1.9658 - precision: 0.4262 - recall: 0.3739 - val_accuracy: 0.3533 - val_auc: 0.5258 - val_loss: 1.5553 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - learning_rate: 2.0000e-05

Epoch 4/25

57/57 ━━━━━━ 0s 915ms/step - accuracy: 0.4278 - auc: 0.6068 - loss: 1.9421 - precision: 0.4440 - recall: 0.3947

Epoch 4: val_accuracy did not improve from 0.35333

57/57 ━━━━━━ 60s 1s/step - accuracy: 0.4361 - auc: 0.6165 - loss: 1.9212 - precision: 0.4503 - recall: 0.4006 - val_accuracy: 0.3333 - val_auc: 0.5089 - val_loss: 1.6156 - val_precision: 0.3464 - val_recall: 0.2067 - learning_rate: 2.0000e-05

Epoch 5/25

57/57 ━━━━━━ 0s 909ms/step - accuracy: 0.4467 - auc: 0.6281 - loss: 1.8572 - precision: 0.4597 - recall: 0.3977

Epoch 5: val_accuracy did not improve from 0.35333

57/57 ━━━━━━ 60s 1s/step - accuracy: 0.4450 - auc: 0.6219 - loss: 1.8877 - precision: 0.4571 - recall: 0.4000 - val_accuracy: 0.3500 - val_auc: 0.5523 - val_loss: 1.6060 - val_precision: 0.4118 - val_recall: 0.2100 - learning_rate: 2.0000e-05

Epoch 6/25

57/57 ━━━━━━ 0s 902ms/step - accuracy: 0.4860 - auc: 0.6629 - loss: 1.7887 - precision: 0.5024 - recall: 0.4357

Epoch 6: val_accuracy improved from 0.35333 to 0.44000, saving model to best_model.keras

57/57 ━━━━━━ 60s 1s/step - accuracy: 0.4589 - auc: 0.6426 - loss: 1.8297 - precision: 0.4709 - recall: 0.4139 - val_accuracy: 0.4400 - val_auc: 0.6639 - val_loss: 1.5346 - val_precision: 0.4876 - val_recall: 0.1967 - learning_rate: 2.0000e-05

Epoch 7/25

57/57 ━━━━━━ 0s 898ms/step - accuracy: 0.4583 - auc: 0.6527 - loss: 1.7758 - precision: 0.4667 - recall: 0.4122

Epoch 7: val_accuracy improved from 0.44000 to 0.52333, saving model to best_model.keras

```
1.keras
57/57 ━━━━━━━━ 60s 1s/step - accuracy: 0.4622 - auc: 0.6537 - loss:
1.7786 - precision: 0.4714 - recall: 0.4206 - val_accuracy: 0.5233 - val_auc: 0.7
679 - val_loss: 1.4261 - val_precision: 0.6232 - val_recall: 0.4300 - learning_rate: 2.0000e-05
Epoch 8/25
57/57 ━━━━━━━━ 0s 906ms/step - accuracy: 0.4763 - auc: 0.6600 - loss:
1.7799 - precision: 0.4933 - recall: 0.4301
Epoch 8: val_accuracy improved from 0.52333 to 0.59667, saving model to best_mode
1.keras
57/57 ━━━━━━━━ 60s 1s/step - accuracy: 0.4872 - auc: 0.6685 - loss:
1.7719 - precision: 0.5013 - recall: 0.4400 - val_accuracy: 0.5967 - val_auc: 0.8
228 - val_loss: 1.3260 - val_precision: 0.6189 - val_recall: 0.5467 - learning_rate: 2.0000e-05
Epoch 9/25
57/57 ━━━━━━━━ 0s 897ms/step - accuracy: 0.4707 - auc: 0.6502 - loss:
1.7986 - precision: 0.4867 - recall: 0.4277
Epoch 9: val_accuracy improved from 0.59667 to 0.61667, saving model to best_mode
1.keras
57/57 ━━━━━━━━ 60s 1s/step - accuracy: 0.4689 - auc: 0.6540 - loss:
1.7927 - precision: 0.4905 - recall: 0.4306 - val_accuracy: 0.6167 - val_auc: 0.8
243 - val_loss: 1.2867 - val_precision: 0.6388 - val_recall: 0.5600 - learning_rate: 2.0000e-05
Epoch 10/25
57/57 ━━━━━━━━ 0s 900ms/step - accuracy: 0.4667 - auc: 0.6462 - loss:
1.8175 - precision: 0.4812 - recall: 0.4173
Epoch 10: val_accuracy improved from 0.61667 to 0.66667, saving model to best_mode
1.keras
57/57 ━━━━━━━━ 60s 1s/step - accuracy: 0.4983 - auc: 0.6750 - loss:
1.7338 - precision: 0.5102 - recall: 0.4450 - val_accuracy: 0.6667 - val_auc: 0.8
280 - val_loss: 1.2379 - val_precision: 0.6980 - val_recall: 0.5933 - learning_rate: 2.0000e-05
Epoch 11/25
57/57 ━━━━━━━━ 0s 891ms/step - accuracy: 0.5226 - auc: 0.6863 - loss:
1.7511 - precision: 0.5329 - recall: 0.4782
Epoch 11: val_accuracy did not improve from 0.66667
57/57 ━━━━━━━━ 81s 1s/step - accuracy: 0.5094 - auc: 0.6807 - loss:
1.7532 - precision: 0.5182 - recall: 0.4589 - val_accuracy: 0.6600 - val_auc: 0.8
269 - val_loss: 1.2428 - val_precision: 0.6920 - val_recall: 0.6067 - learning_rate: 2.0000e-05
Epoch 12/25
57/57 ━━━━━━━━ 0s 899ms/step - accuracy: 0.5117 - auc: 0.6986 - loss:
1.6701 - precision: 0.5359 - recall: 0.4691
Epoch 12: val_accuracy did not improve from 0.66667
57/57 ━━━━━━━━ 59s 1s/step - accuracy: 0.5094 - auc: 0.6952 - loss:
1.6814 - precision: 0.5295 - recall: 0.4639 - val_accuracy: 0.6167 - val_auc: 0.8
276 - val_loss: 1.2929 - val_precision: 0.6442 - val_recall: 0.5733 - learning_rate: 2.0000e-05
Epoch 13/25
57/57 ━━━━━━━━ 0s 2s/step - accuracy: 0.4799 - auc: 0.6713 - loss: 1.
7549 - precision: 0.4954 - recall: 0.4437
Epoch 13: val_accuracy did not improve from 0.66667
57/57 ━━━━━━━━ 137s 2s/step - accuracy: 0.4961 - auc: 0.6824 - loss:
1.7107 - precision: 0.5134 - recall: 0.4572 - val_accuracy: 0.6433 - val_auc: 0.8
346 - val_loss: 1.2164 - val_precision: 0.6805 - val_recall: 0.6033 - learning_rate: 2.0000e-05
Epoch 14/25
57/57 ━━━━━━━━ 0s 902ms/step - accuracy: 0.5135 - auc: 0.6932 - loss:
1.6864 - precision: 0.5286 - recall: 0.4649
Epoch 14: val_accuracy did not improve from 0.66667
```

```
57/57 ━━━━━━━━ 59s 1s/step - accuracy: 0.5156 - auc: 0.7006 - loss: 1.6584 - precision: 0.5308 - recall: 0.4694 - val_accuracy: 0.6300 - val_auc: 0.8390 - val_loss: 1.2443 - val_precision: 0.6579 - val_recall: 0.5833 - learning_rate: 2.0000e-05
Epoch 15/25
57/57 ━━━━━━ 0s 901ms/step - accuracy: 0.5196 - auc: 0.7157 - loss: 1.6312 - precision: 0.5337 - recall: 0.4791
Epoch 15: val_accuracy did not improve from 0.66667
57/57 ━━━━━━ 59s 1s/step - accuracy: 0.5278 - auc: 0.7073 - loss: 1.6557 - precision: 0.5443 - recall: 0.4878 - val_accuracy: 0.6633 - val_auc: 0.8403 - val_loss: 1.2227 - val_precision: 0.6923 - val_recall: 0.6000 - learning_rate: 2.0000e-05
Epoch 16/25
57/57 ━━━━━━ 0s 877ms/step - accuracy: 0.5247 - auc: 0.7151 - loss: 1.6080 - precision: 0.5397 - recall: 0.4880
Epoch 16: val_accuracy improved from 0.66667 to 0.67333, saving model to best_model.keras
57/57 ━━━━━━ 59s 1s/step - accuracy: 0.5294 - auc: 0.7128 - loss: 1.6269 - precision: 0.5426 - recall: 0.4850 - val_accuracy: 0.6733 - val_auc: 0.8459 - val_loss: 1.2069 - val_precision: 0.7004 - val_recall: 0.6467 - learning_rate: 2.0000e-05
Epoch 17/25
57/57 ━━━━━━ 0s 895ms/step - accuracy: 0.5316 - auc: 0.7199 - loss: 1.6340 - precision: 0.5375 - recall: 0.4749
Epoch 17: val_accuracy did not improve from 0.67333
57/57 ━━━━━━ 59s 1s/step - accuracy: 0.5111 - auc: 0.7076 - loss: 1.6507 - precision: 0.5281 - recall: 0.4706 - val_accuracy: 0.6700 - val_auc: 0.8396 - val_loss: 1.2254 - val_precision: 0.6787 - val_recall: 0.6267 - learning_rate: 2.0000e-05
Epoch 18/25
57/57 ━━━━━━ 0s 877ms/step - accuracy: 0.5267 - auc: 0.7168 - loss: 1.6324 - precision: 0.5386 - recall: 0.4757
Epoch 18: val_accuracy did not improve from 0.67333
57/57 ━━━━━━ 58s 1s/step - accuracy: 0.5361 - auc: 0.7165 - loss: 1.6275 - precision: 0.5507 - recall: 0.4889 - val_accuracy: 0.6533 - val_auc: 0.8406 - val_loss: 1.2760 - val_precision: 0.6703 - val_recall: 0.6233 - learning_rate: 2.0000e-05
Epoch 19/25
57/57 ━━━━━━ 0s 900ms/step - accuracy: 0.5173 - auc: 0.7195 - loss: 1.5988 - precision: 0.5357 - recall: 0.4799
Epoch 19: val_accuracy did not improve from 0.67333
57/57 ━━━━━━ 283s 5s/step - accuracy: 0.5406 - auc: 0.7255 - loss: 1.5975 - precision: 0.5604 - recall: 0.5000 - val_accuracy: 0.6633 - val_auc: 0.8510 - val_loss: 1.2025 - val_precision: 0.6842 - val_recall: 0.6067 - learning_rate: 2.0000e-05
Epoch 20/25
57/57 ━━━━━━ 0s 898ms/step - accuracy: 0.5477 - auc: 0.7252 - loss: 1.5934 - precision: 0.5651 - recall: 0.4965
Epoch 20: val_accuracy improved from 0.67333 to 0.70333, saving model to best_model.keras
57/57 ━━━━━━ 60s 1s/step - accuracy: 0.5350 - auc: 0.7191 - loss: 1.6167 - precision: 0.5584 - recall: 0.4911 - val_accuracy: 0.7033 - val_auc: 0.8607 - val_loss: 1.2251 - val_precision: 0.7292 - val_recall: 0.6733 - learning_rate: 2.0000e-05
Epoch 21/25
57/57 ━━━━━━ 0s 893ms/step - accuracy: 0.5126 - auc: 0.7143 - loss: 1.6309 - precision: 0.5400 - recall: 0.4708
Epoch 21: val_accuracy did not improve from 0.70333
57/57 ━━━━━━ 59s 1s/step - accuracy: 0.5211 - auc: 0.7145 - loss: 1.6274 - precision: 0.5423 - recall: 0.4806 - val_accuracy: 0.6667 - val_auc: 0.8607 - val_loss: 1.2251 - val_precision: 0.7292 - val_recall: 0.6733 - learning_rate: 2.0000e-05
```

```

544 - val_loss: 1.2389 - val_precision: 0.7052 - val_recall: 0.6300 - learning_rate: 2.0000e-05
Epoch 22/25
57/57 ━━━━━━━━ 0s 894ms/step - accuracy: 0.5387 - auc: 0.7202 - loss: 1.5834 - precision: 0.5575 - recall: 0.4919
Epoch 22: val_accuracy did not improve from 0.70333
57/57 ━━━━━━ 59s 1s/step - accuracy: 0.5328 - auc: 0.7184 - loss: 1.5997 - precision: 0.5480 - recall: 0.4850 - val_accuracy: 0.6500 - val_auc: 0.8632 - val_loss: 1.2227 - val_precision: 0.6787 - val_recall: 0.6267 - learning_rate: 2.0000e-05
Epoch 23/25
57/57 ━━━━━━ 0s 891ms/step - accuracy: 0.5353 - auc: 0.7050 - loss: 1.6793 - precision: 0.5428 - recall: 0.4818
Epoch 23: val_accuracy did not improve from 0.70333
57/57 ━━━━━━ 58s 1s/step - accuracy: 0.5489 - auc: 0.7205 - loss: 1.6270 - precision: 0.5586 - recall: 0.4978 - val_accuracy: 0.6467 - val_auc: 0.8569 - val_loss: 1.2657 - val_precision: 0.6607 - val_recall: 0.6167 - learning_rate: 2.0000e-05
Epoch 24/25
57/57 ━━━━━━ 0s 879ms/step - accuracy: 0.5453 - auc: 0.7509 - loss: 1.5220 - precision: 0.5723 - recall: 0.5099
Epoch 24: val_accuracy did not improve from 0.70333
57/57 ━━━━━━ 58s 1s/step - accuracy: 0.5372 - auc: 0.7309 - loss: 1.5874 - precision: 0.5607 - recall: 0.5000 - val_accuracy: 0.6500 - val_auc: 0.8461 - val_loss: 1.2754 - val_precision: 0.6691 - val_recall: 0.6200 - learning_rate: 2.0000e-05
Epoch 24: early stopping
Restoring model weights from the end of the best epoch: 19.

```

Часть 4. Графики обучения и метрики

```

In [55]: def plot_training_history(history):
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))

    axes[0, 0].plot(history.history['loss'], label='Train Loss', linewidth=2)
    axes[0, 0].plot(history.history['val_loss'], label='Val Loss', linewidth=2)
    axes[0, 0].set_title('Model Loss', fontsize=14, fontweight='bold')
    axes[0, 0].set_xlabel('Epoch')
    axes[0, 0].set_ylabel('Loss')
    axes[0, 0].legend()
    axes[0, 0].grid(True, alpha=0.3)

    axes[0, 1].plot(history.history['accuracy'], label='Train Accuracy', linewidth=2)
    axes[0, 1].plot(history.history['val_accuracy'], label='Val Accuracy', linewidth=2)
    axes[0, 1].set_title('Model Accuracy', fontsize=14, fontweight='bold')
    axes[0, 1].set_xlabel('Epoch')
    axes[0, 1].set_ylabel('Accuracy')
    axes[0, 1].legend()
    axes[0, 1].grid(True, alpha=0.3)

    axes[0, 2].plot(history.history['precision'], label='Train Precision', linewidth=2)
    axes[0, 2].plot(history.history['val_precision'], label='Val Precision', linewidth=2)
    axes[0, 2].set_title('Model Precision', fontsize=14, fontweight='bold')
    axes[0, 2].set_xlabel('Epoch')
    axes[0, 2].set_ylabel('Precision')
    axes[0, 2].legend()
    axes[0, 2].grid(True, alpha=0.3)

```

```
axes[1, 0].plot(history.history['recall'], label='Train Recall', linewidth=2)
axes[1, 0].plot(history.history['val_recall'], label='Val Recall', linewidth=2)
axes[1, 0].set_title('Model Recall', fontsize=14, fontweight='bold')
axes[1, 0].set_xlabel('Epoch')
axes[1, 0].set_ylabel('Recall')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

axes[1, 1].plot(history.history['auc'], label='Train AUC', linewidth=2)
axes[1, 1].plot(history.history['val_auc'], label='Val AUC', linewidth=2)
axes[1, 1].set_title('Model AUC', fontsize=14, fontweight='bold')
axes[1, 1].set_xlabel('Epoch')
axes[1, 1].set_ylabel('AUC')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

if 'lr' in history.history:
    axes[1, 2].plot(history.history['lr'], label='Learning Rate', linewidth=2)
    axes[1, 2].set_title('Learning Rate Schedule', fontsize=14, fontweight='bold')
    axes[1, 2].set_xlabel('Epoch')
    axes[1, 2].set_ylabel('Learning Rate')
    axes[1, 2].legend()
    axes[1, 2].grid(True, alpha=0.3)
    axes[1, 2].set_yscale('log')

plt.tight_layout()
plt.show()

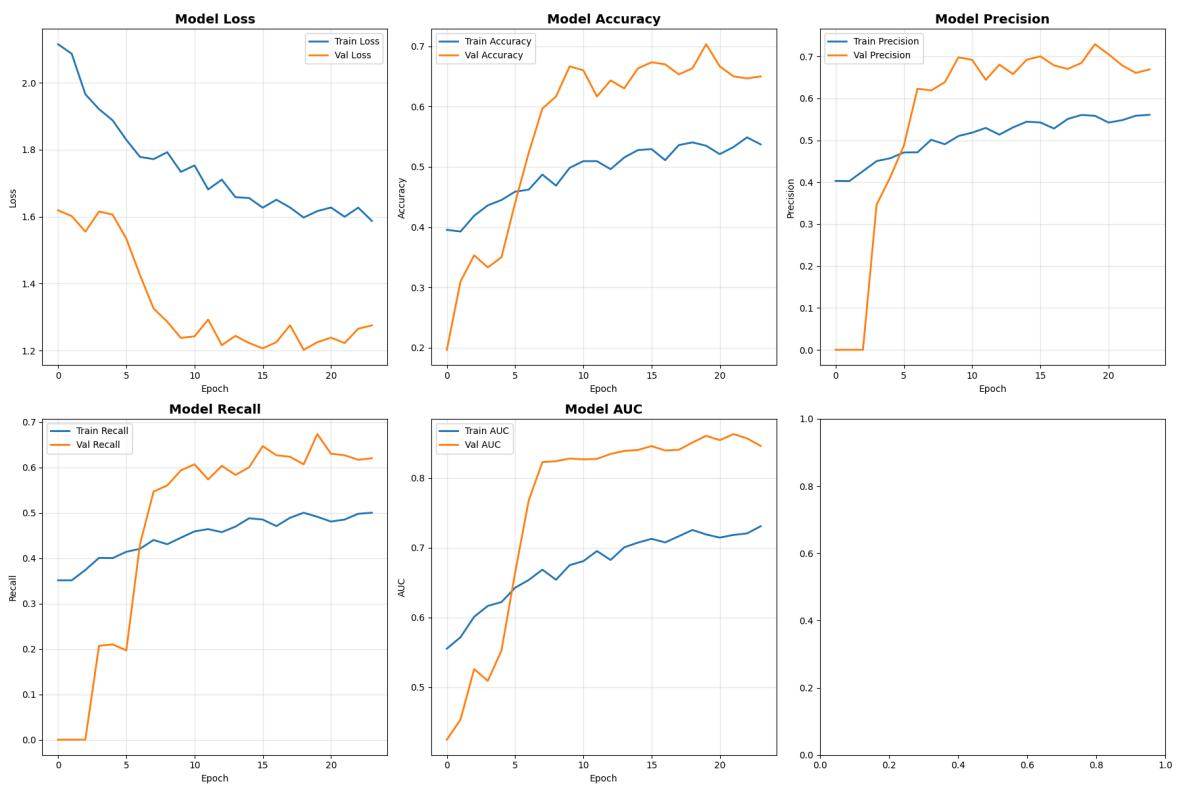
final_train_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_train_acc = history.history['accuracy'][-1]
final_val_acc = history.history['val_accuracy'][-1]

print(f"Финальный тренировочный loss: {final_train_loss:.4f}")
print(f"Финальный валидационный loss: {final_val_loss:.4f}")
print(f"Финальная тренировочная accuracy: {final_train_acc:.4f}")
print(f"Финальная валидационная accuracy: {final_val_acc:.4f}")

loss_gap = abs(final_train_loss - final_val_loss)
acc_gap = abs(final_train_acc - final_val_acc)

print(f"Разрыв в loss: {loss_gap:.4f}")
print(f"Разрыв в accuracy: {acc_gap:.4f}")

plot_training_history(history)
```



Финальный тренировочный loss: 1.5874

Финальный валидационный loss: 1.2754

Финальная тренировочная accuracy: 0.5372

Финальная валидационная accuracy: 0.6500

Разрыв в loss: 0.3120

Разрыв в accuracy: 0.1128

Очень странная тенденция, модель похоже вообще не могла узнать ни кого из датасета и просто переобучалась на сгенеренных данных (их относительно мало и они похожи несмотря на все аугментации)

Пытался это исправить сменой модели, изменением слоев, кол-во слоев участвующих в дообучении, гиперпараметры менял, ничего не помогло. Видимо создание синтетического датасета само по себе неправильно :/

```
In [60]: model = keras.models.load_model('best_model.keras')
print("\nОценка модели на тестовой выборке:")
test_results = model.evaluate(test_generator, verbose=0)

metrics_names = model.metrics_names
for name, value in zip(metrics_names, test_results):
    print(f"{name}: {value:.4f}")
```

Оценка модели на тестовой выборке:

loss: 1.1698

compile_metrics: 0.6933

```
In [57]: test_generator.reset()
y_pred = model.predict(test_generator, verbose=1)
y_pred_classes = np.argmax(y_pred, axis=1)

y_true = test_generator.classes
class_names = list(test_generator.class_indices.keys())

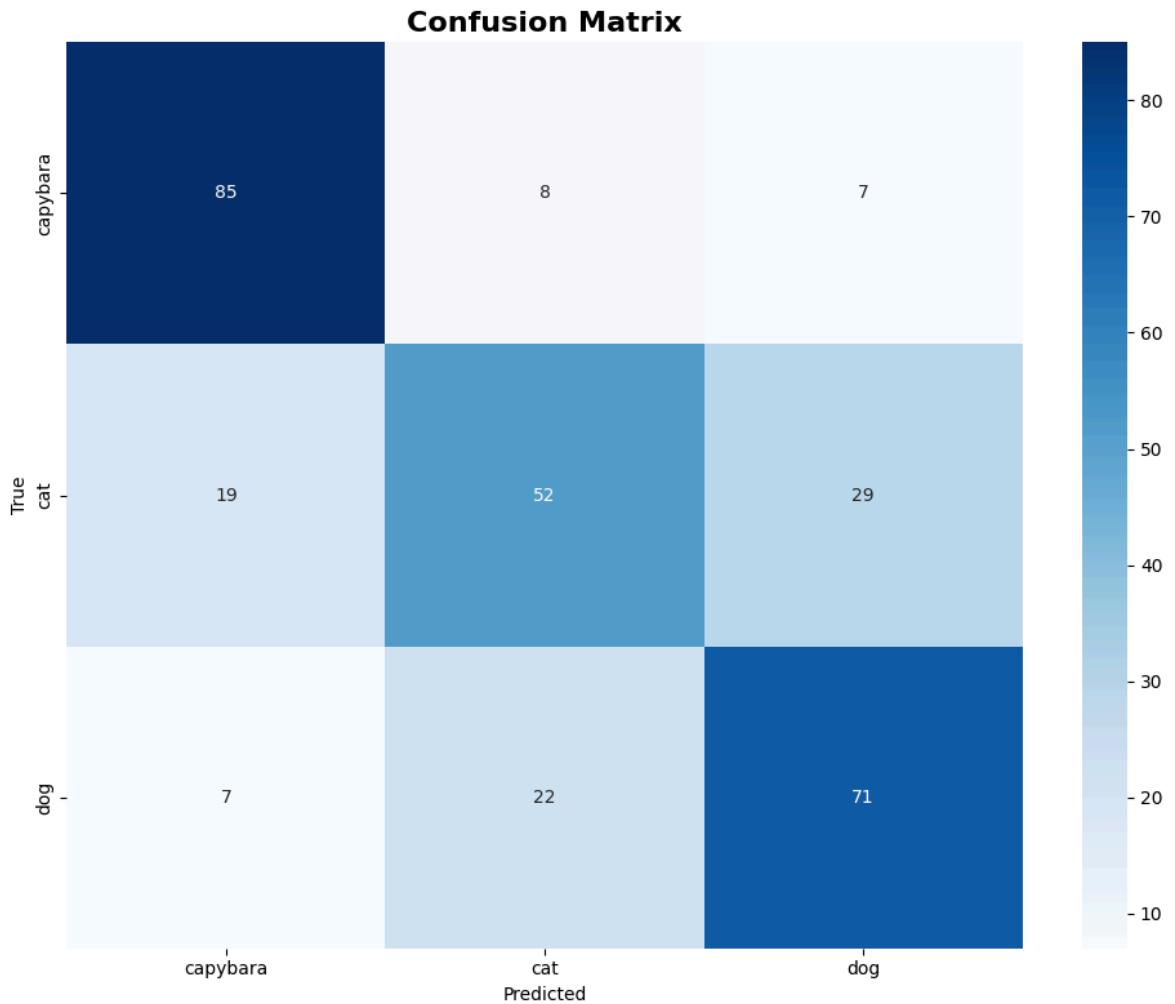
cm = confusion_matrix(y_true, y_pred_classes)
```

```

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.tight_layout()
plt.show()

```

10/10 ————— 11s 888ms/step



```

In [58]: y_true_bin = label_binarize(y_true, classes=range(len(class_names)))

plt.figure(figsize=(12, 10))

fpr = {}
tpr = {}
roc_auc = {}

for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

colors = ['blue', 'green', 'red', 'orange']
for i, color in zip(range(len(class_names)), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'{class_names[i]} (AUC = {roc_auc[i]:.3f})')

```

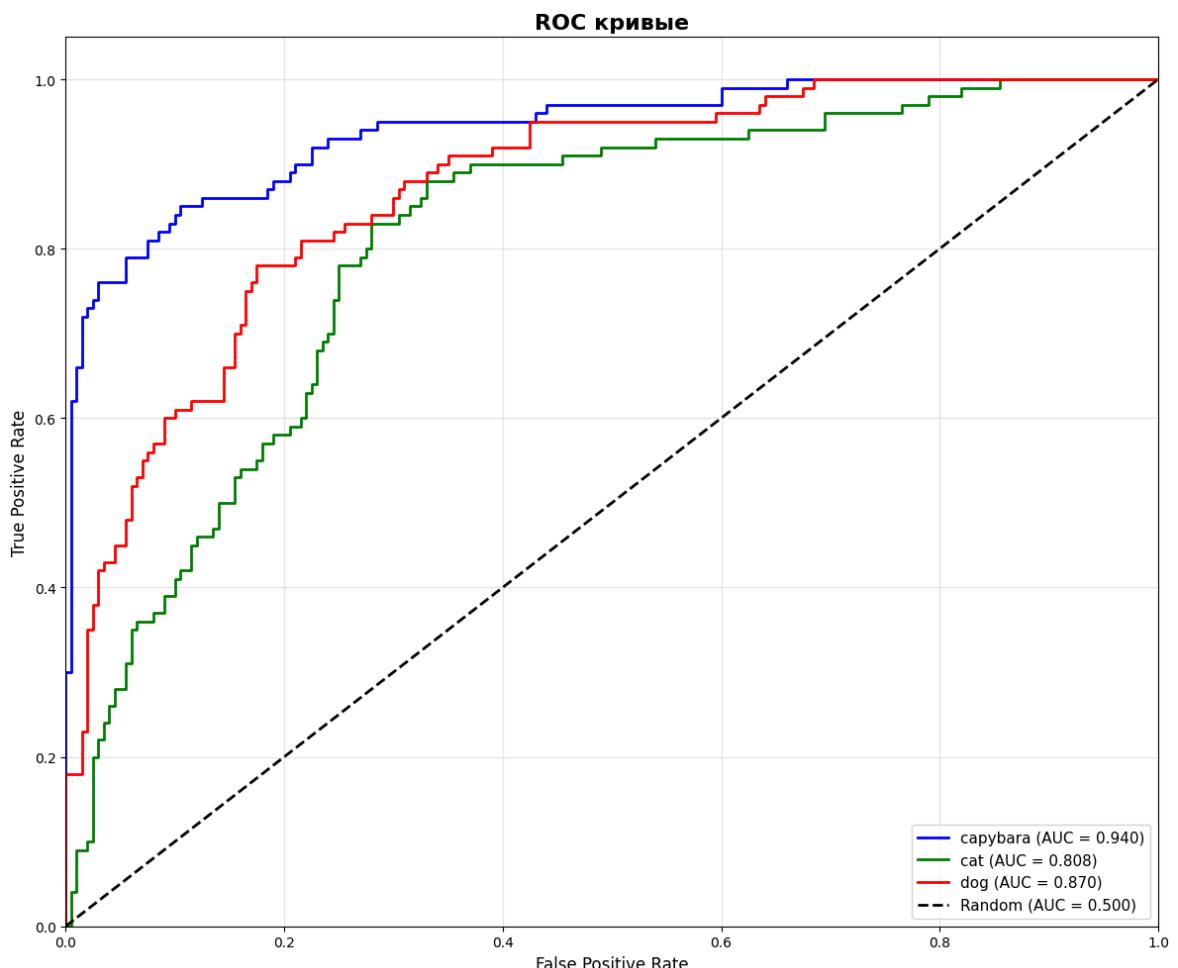
```

plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random (AUC = 0.500)')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('ROC кривые', fontsize=16, fontweight='bold')
plt.legend(loc="lower right", fontsize=11)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

mean_auc = np.mean(list(roc_auc.values()))
print(f"Средний AUC: {mean_auc:.4f}")

```



Средний AUC: 0.8729

```

In [61]: report = classification_report(y_true, y_pred_classes,
                                         target_names=class_names,
                                         digits=4,
                                         output_dict=True)

report_df = pd.DataFrame(report).transpose()

print("\nТаблица метрик по классам:")
display(report_df[['precision', 'recall', 'f1-score']].iloc[:-3])

```

Таблица метрик по классам:

	precision	recall	f1-score
capybara	0.765766	0.85	0.805687
cat	0.634146	0.52	0.571429
dog	0.663551	0.71	0.685990

Вывод

Синтетический датасет оказался совсем неестественным (летающие коты). Хоть это и позволило создать датасет практически из ничего, но он не имеет ценности, ведь даже с аугментациями изображения выглядят очень похожими.

Модель получилась такая себе и похоже переобучилась на данных, не знаю как так получилось что стоковая модель не могла распознать никого из датасета. Похоже неправильно что-то сделал в генерации датасета.