Ask Gemini Create a function that accepts a list of numbers and returns the sum of the numbers

---- INT_STEP -- Подключение библиотек.

```
# ================================================================
# ---- INT_STEP -- Подключение библиотек.
# ================================================================
# ---- Стандартные библиотеки ----
import os, sys, shutil, glob, time
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams['font.family'] = 'DejaVu Sans'
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import yaml
import tqdm

# ---- Scikit-learn ----
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import (
    confusion_matrix, classification_report, f1_score, precision_score, recall_score,
    roc_curve, auc, precision_recall_curve, average_precision_score
)
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.utils import compute_class_weight

# ---- Keras / TensorFlow ----
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.layers import (
    Conv2D, MaxPooling2D, Dense, Dropout, Flatten, BatchNormalization,
    GlobalAveragePooling2D, Input, Add
)
from tensorflow.keras.callbacks import (
    EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, CSVLogger
)
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# ---- Keras applications (предобученные модели) ----
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input

# ---- Дополнительные библиотеки ----
import cv2
import PIL, PIL.Image, PIL.ImageDraw, PIL.ImageFont
import tqdm.notebook as tqdm_nb

# ---- Настройка отображения в ноутбуке (если запуск в Jupyter) ----
%matplotlib inline
%load_ext tensorboard
%load_ext google.colab.drive

# ---- Версии для проверки совместимости ----
print("Python:", sys.version.split()[0])
print("TensorFlow:", tf.__version__)
print("Keras:", keras.__version__)
print("OpenCV:", cv2.__version__)
print("Pillow:", PIL.__version__)
print("Scikit-learn:", sklearn.__version__)
print("Pandas:", pd.__version__)
print("Numpy:", np.__version__)
print("Matplotlib:", matplotlib.__version__)
print("Seaborn:", sns.__version__)

# ---- Проверка доступности GPU ----
print("\n---- Проверка GPU ----")
gpu_devices = tf.config.list_physical_devices('GPU')
if len(gpu_devices) > 0:
    print("GPU доступен:", len(gpu_devices), "устройств")
    for gpu in gpu_devices:
        print(gpu)
    # Включаем динамическое выделение памяти, чтобы не занимать всю память сразу
    for gpu in gpu_devices:
        tf.config.experimental.set_memory_growth(gpu, True)
else:
    print("GPU не найден, будем использовать CPU")

# ---- Версия библиотеки для сохранения моделей ----
print("\n---- Версии для сохранения ----")
print("Pickle:", pickle.__version__)
print("Yaml:", yaml.__version__)

print("\n---- Подключение завершено ----")
```

---- Python: Дополнительно в ноутбуке ----

```
%pip install -q -U albumentations
```

Requirement already satisfied: boost-py!=1.70,>=1.69.0 in /usr/local/lib/python3.10/dist-packages (1.7.0)

---- WORKER_2 ----

```
# ================================================================
# ---- WORKER_2 ---- (версия 2 — с поддержкой ансамбля)
# ================================================================
# ---- Импорты ----
import os, sys, shutil, glob, time
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams['font.family'] = 'DejaVu Sans'
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import yaml
import tqdm

# ---- Scikit-learn ----
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import (
    confusion_matrix, classification_report, f1_score, precision_score, recall_score,
    roc_curve, auc, precision_recall_curve, average_precision_score
)
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.utils import compute_class_weight

# ---- Keras / TensorFlow ----
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.layers import (
    Conv2D, MaxPooling2D, Dense, Dropout, Flatten, BatchNormalization,
    GlobalAveragePooling2D, Input, Add
)
from tensorflow.keras.callbacks import (
    EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, CSVLogger
)
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# ---- Keras applications (предобученные модели) ----
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input

# ---- Дополнительные библиотеки ----
import cv2
import PIL, PIL.Image, PIL.ImageDraw, PIL.ImageFont
import tqdm.notebook as tqdm_nb

# ---- Версии для проверки совместимости ----
print("Python:", sys.version.split()[0])
print("TensorFlow:", tf.__version__)
print("Keras:", keras.__version__)
print("OpenCV:", cv2.__version__)
print("Pillow:", PIL.__version__)
print("Scikit-learn:", sklearn.__version__)
print("Pandas:", pd.__version__)
print("Numpy:", np.__version__)
print("Matplotlib:", matplotlib.__version__)
print("Seaborn:", sns.__version__)

# ---- Проверка доступности GPU ----
print("\n---- Проверка GPU ----")
gpu_devices = tf.config.list_physical_devices('GPU')
if len(gpu_devices) > 0:
    print("GPU доступен:", len(gpu_devices), "устройств")
    for gpu in gpu_devices:
        print(gpu)
    # Включаем динамическое выделение памяти, чтобы не занимать всю память сразу
    for gpu in gpu_devices:
        tf.config.experimental.set_memory_growth(gpu, True)
else:
    print("GPU не найден, будем использовать CPU")

# ---- Версия библиотеки для сохранения моделей ----
print("\n---- Версии для сохранения ----")
print("Pickle:", pickle.__version__)
print("Yaml:", yaml.__version__)

print("\n---- Подключение завершено ----")

# ---- Монтирование Google Drive ----
from google.colab.drive import mount, flush_and_unmount
mount('/content/drive')

# ---- Пути к данным ----
DATA_ROOT = '/content/drive/MyDrive/yolo_training/1000kyolo'
TRAIN_DIR = os.path.join(DATA_ROOT, 'train')
VALID_DIR = os.path.join(DATA_ROOT, 'valid')
TEST_DIR  = os.path.join(DATA_ROOT, 'test')

# ---- Проверка наличия папок ----
print("\n---- Проверка путей ----")
for name, path in [('train', TRAIN_DIR), ('valid', VALID_DIR), ('test', TEST_DIR)]:
    exists = os.path.exists(path)
    print(f"{name}: {path} {'✓' if exists else '✗ НЕТ!'}")

# ---- Сколько файлов в каждой папке ----
print("\n---- Количество файлов ----")
for name, path in [('train', TRAIN_DIR), ('valid', VALID_DIR), ('test', TEST_DIR)]:
    if os.path.exists(path):
        n = len(os.listdir(path))
        print(f"{name}: {n} файлов")

# ---- Параметры ----
IMG_SIZE = 224
BATCH_SIZE = 32
SEED = 42

# ---- Фиксируем random seed для воспроизводимости ----
np.random.seed(SEED)
tf.random.set_seed(SEED)

print("\n---- Готово ----")

# ---- Сколько классов ----
print("\n---- Классы ----")
import os, glob
classes = sort(os.listdir(TRAIN_DIR))
print("Классов:", len(classes))
print("Первые 10:", classes[:10])

# ---- Сколько файлов в train ----
print("\n---- Train ----")
for c in classes[:10]:
    n = len(os.listdir(os.path.join(TRAIN_DIR, c)))
    print(f"{c}: {n}")

# ---- Сколько файлов в valid ----
print("\n---- Valid ----")
for c in classes[:10]:
    n = len(os.listdir(os.path.join(VALID_DIR, c)))
    print(f"{c}: {n}")

# ---- Сколько файлов в test ----
print("\n---- Test ----")
for c in classes[:10]:
    n = len(os.listdir(os.path.join(TEST_DIR, c)))
    print(f"{c}: {n}")

# ---- Сколько всего ----
print("\n---- Всего ----")
print("train:", len(glob.glob(os.path.join(TRAIN_DIR, '*/*'))))
print("valid:", len(glob.glob(os.path.join(VALID_DIR, '*/*'))))
print("test:", len(glob.glob(os.path.join(TEST_DIR, '*/*'))))

# ---- Сколько классов ----
NUM_CLASSES = len(classes)
print("\n---- Классов:", NUM_CLASSES, "----
```

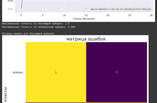
```
# Сверточная нейронная сеть для классификации снимков грудной клетки
# Использует предобученную ResNet18 с дообучением последних слоев
# Датасет: Kaggle Chest X-Ray Images (Pneumonia)
import os
import shutil
import kagglehub
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import transforms, datasets, models
from PIL import Image, ImageOps, ImageEnhance, ImageFilter
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from torchsummary import summary

# Скачиваем датасет
path = kagglehub.dataset_download("paultimothymooney/chest-xray-pneumonia")
print("Путь к датасету:", path)

# Определяем пути к train, val, test
base_path = os.path.join(path, "chest_xray")
train_path = os.path.join(base_path, "train")
val_path = os.path.join(base_path, "val")
test_path = os.path.join(base_path, "test")

# Функция для подсчета файлов в каждой папке
def count_files(directory):
    counts = {}
    for class_name in ['NORMAL', 'PNEUMONIA']:
        class_path = os.path.join(directory, class_name)
        if os.path.exists(class_path):
            files = [f for f in os.listdir(class_path) if f.lower().endswith('.jpeg')]
            counts[class_name] = len(files)
    return counts

print("\n Статистика по датасету:")
for dataset, path in [('train', train_path), ('val', val_path), ('test', test_path)]:
    counts = count_files(path)
    print(f"{dataset}: {counts}")

# Трансформации для train с аугментацией
train_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1), scale=(0.9, 1.1)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Трансформации для validation/test (без аугментации)
val_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Загружаем датasеты
train_dataset = datasets.ImageFolder(train_path, transform=train_transform)
val_dataset = datasets.ImageFolder(val_path, transform=val_transform)
test_dataset = datasets.ImageFolder(test_path, transform=val_transform)

# Проверяем классы
print("\n Классы:", train_dataset.classes)

# Создаем загрузчики данных
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)

print(f"\n Батчей в train: {len(train_loader)}")
print(f" Батчей в val: {len(val_loader)}")
print(f" Батчей в test: {len(test_loader)}")

# Проверяем размерность одного батча
for images, labels in train_loader:
    print(f"\n Размерность батча: {images.shape}")
    print(f" Метки: {labels[:10]}")
    break

# Функция для визуализации примеров
def show_samples(dataset, num_samples=8):
    fig, axes = plt.subplots(2, 4, figsize=(12, 6))
    for i, ax in enumerate(axes.flat):
        if i < num_samples:
            image, label = dataset[i]
            image = image.permute(1, 2, 3, 0).numpy()
            image = np.clip(image, 0, 1)
            ax.imshow(image)
            ax.set_title(f'{dataset.classes[label]}')
            ax.axis('off')
    plt.tight_layout()
    plt.show()

print("\n Примеры из train:")
show_samples(train_dataset)

# Создаем модель
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"\n Устройство: {device}")

model = models.resNet18(pretrained=True)
print("\n Загружена предобученная ResNet18")

# Заменяем последний слой
num_classes = len(train_dataset.classes)
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Замораживаем все слои кроме последнего
for param in model.parameters:
    param.requires_grad = False
model.fc.requires_grad = True

model = model.to(device)

# Считаем параметры
total_params = sum(p.numpy() for p in model.parameters())
trainable_params = sum(p.numpy() for p in model.parameters() if p.requires_grad)
print(f" Всего параметров: {total_params:,}")
print(f" Обучаемых параметров: {trainable_params:,}")

# Функция обучения
def train_epoch(model, loader, criterion, optimizer):
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    for images, labels in tqdm(loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item * images.size(0)
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum.item
        total += labels.size(0)
    return total_loss / total, correct / total

# Функция валидации
def validate(model, loader, criterion):
    model.eval
    total_loss = 0
    correct = 0
    total = 0
    with torch.no_grad:
        for images, labels in tqdm(loader):
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            total_loss += loss.item * images.size(0)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum.item
            total += labels.size(0)
    return total_loss / total, correct / total

# Параметры обучения
criterion = nn.CrossEntropyLoss
optimizer = optim.Adam(model.fc.parameters, lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

num_epochs = 10
best_val_acc = 0
history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

print("\n Начинаем обучение...")
for epoch in range(num_epochs):
    print(f"\n Эпоха {epoch+1}/{num_epochs}")
    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer)
    val_loss, val_acc = validate(model, val_loader, criterion)
    scheduler.step
    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)
    print(f" Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}")
    print(f" Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(model.state_dict, 'best_model.pth')
        print(" Сохранена лучшая модель!")

print("\n Обучение завершено!")
print(f" Лучшая точность на validation: {best_val_acc:.4f}")

# Загружаем лучшую модель
model.load_state_dict(torch.load('best_model.pth'))

# Предсказания на test
print("\n Предсказания на test...")
test_loss, test_acc = validate(model, test_loader, criterion)
print(f" Test Loss: {test_loss:.4f}")
print(f" Test Acc: {test_acc:.4f}")

# Confusion matrix
all_preds = []
all_labels = []
for images, labels in tqdm(test_loader):
    images = images.to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)
    all_preds.extend(predicted.cpu.numpy)
    all_labels.extend(labels.numpy)

cm = confusion_matrix(all_labels, all_preds)
print("\n Confusion matrix:")
print(cm)

# Classification report
print("\n Classification report:")
print(classification_report(all_labels, all_preds, target_names=train_dataset.classes))

# Визуализация confusion matrix
plt.figure(figure=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=train_dataset.classes,
            yticklabels=train_dataset.classes)
plt.ylabel('Истинные метки')
plt.xlabel('Предсказанные метки')
plt.title('Confusion matrix')
plt.show

# Визуализация истории обучения
plt.figure(figure=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history['train_loss'], label='Train')
plt.plot(history['val_loss'], label='Val')
plt.ylabel('Loss')
plt.xlabel('Эпоха')
plt.legend
plt.grid
plt.subplot(1, 2, 2)
plt.plot(history['train_acc'], label='Train')
plt.plot(history['val_acc'], label='Val')
plt.ylabel('Accuracy')
plt.xlabel('Эпоха')
plt.legend
plt.grid
plt.tight_layout
plt.show

# Примеры предсказаний
print("\ Примеры предсказаний...")
for images, labels in test_loader:
    images = images.to(device)
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)
    break

fig, axes = plt.subplots(2, 4, figsize=(12, 6))
for i, ax in enumerate(axes.flat):
    if i < 8:
        image = images[i].permute(1, 2, 3, 0).numpy
        image = np.clip(image, 0, 1)
        ax.imshow(image)
        true_label = train_dataset.classes[labels[i]]
        pred_label = train_dataset.classes[predicted[i]]
        color = 'green' if true_label == pred_label else 'red'
        ax.set_title(f'Истинно: {true_label}\Предсказано: {pred_label}', color=color)
        ax.axis('off')
plt.tight_layout
plt.show

print("\ Готово!")
```

---- WORKER ----

```
[ 1 ] | [ 2 ] | [ 3 ] | [ 4 ] | [ 5 ] | [ 6 ] | [ 7 ] | [ 8 ] | [ 9 ] | [ 10 ] | [
```

```
# Импорт библиотек
import numpy as np
import tensorflow as tf
from tensorflow.keras import models, layers, callbacks
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Загрузка данных
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Нормализация пикселей
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Преобразование меток в one-hot encoding
num_classes = 10
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# Разделение на train и validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Создание модели
model = models.Sequential()

# Первый сверточный блок
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D(2, 2))

# Второй сверточный блок
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))

# Третий сверточный блок
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Полносвязные слои
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Компиляция модели
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Вывод архитектуры модели
model.summary()

# Обучение модели
history = model.fit(X_train, y_train,
                    epochs=50,
                    batch_size=32,
                    validation_data=(X_val, y_val),
                    verbose=1)

# Оценка на тестовом наборе
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Точность на тестовом наборе: {test_acc:.4f}")
print(f"Потери на тестовом наборе: {test_loss:.4f}")

# Визуализация истории обучения
plt.figure(figure_size=(12, 4))

plt.plot(history.history['accuracy'], label='Точность на обучающем наборе')
plt.plot(history.history['val_accuracy'], label='Точность на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Точность')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
Обучающая выборка: 40000 изображений
Проверочная выборка: 10000 изображений
Тестовая выборка: 10000 изображений
```



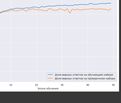
- Показать/скрыть

```
# Импорт необходимых библиотек
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Загрузка данных CIFAR-10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Нормализация пикселей
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Преобразование меток в one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Разделение на train и validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Создание модели
model = Sequential()

# Первый сверточный блок
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(2, 2))

# Второй сверточный блок
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))

# Третий сверточный блок
model.add(Conv2D(64, (3, 3), activation='relu'))

# Полносвязные слои
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# Компиляция модели
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Вывод архитектуры модели
model.summary()

# Обучение модели
history = model.fit(X_train, y_train,
                    epochs=50,
                    batch_size=32,
                    validation_data=(X_val, y_val),
                    verbose=1)

# Оценка на тестовом наборе
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Точность на тестовом наборе: {test_acc:.4f}")
print(f"Потери на тестовом наборе: {test_loss:.4f}")

# Визуализация истории обучения
plt.figure(figure_size=(12, 4))

plt.plot(history.history['accuracy'], label='Точность на обучающем наборе')
plt.plot(history.history['val_accuracy'], label='Точность на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Точность')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
Обучающая выборка: 40000 изображений
Проверочная выборка: 10000 изображений
Тестовая выборка: 10000 изображений
```

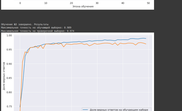
- Показать/скрыть

```
# Импорт необходимых библиотек
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Загрузка данных CIFAR-10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Нормализация пикселей
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Преобразование меток в one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Разделение на train и validation
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Создание модели
model = Sequential()

# Первый сверточный блок
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(2, 2))

# Второй сверточный блок
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))

# Третий сверточный блок
model.add(Conv2D(64, (3, 3), activation='relu'))

# Полносвязные слои
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# Компиляция модели
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Вывод архитектуры модели
model.summary()

# Обучение модели
history = model.fit(X_train, y_train,
                    epochs=50,
                    batch_size=32,
                    validation_data=(X_val, y_val),
                    verbose=1)

# Оценка на тестовом наборе
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Точность на тестовом наборе: {test_acc:.4f}")
print(f"Потери на тестовом наборе: {test_loss:.4f}")

# Визуализация истории обучения
plt.figure(figure_size=(12, 4))

plt.plot(history.history['accuracy'], label='Точность на обучающем наборе')
plt.plot(history.history['val_accuracy'], label='Точность на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Точность')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```




```
Обучающая выборка: 40000 изображений
Проверочная выборка: 10000 изображений
Тестовая выборка: 10000 изображений
```

```
Epoch 10/10
```



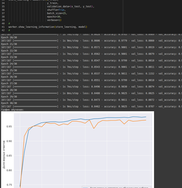
```
Частота классов в тестовом наборе: 0.500
Лучший порог по F1-мере: 0.500
Лучший F1-score на тестовом наборе: 0.974
```

```
Частота классов в тестовом наборе: 0.500
Лучший порог по F1-мере: 0.500
Лучший F1-score на тестовом наборе: 0.974
Лучший порог по Youden's J-statistic: 0.500
Лучший J-statistic на тестовом наборе: 0.974
```

Эксп. №7 Сделаем первый сверточный (97.4% 97.7%)

```
# 2025-10-04 14:11:18 Сделаем первый сверточный (97.4% 97.7%)
# Keras version: 2.15.0
# TF version: 2.15.0
# Image size: (64, 64)
# Train: 5000 x 64 x 64 x 1, test: 2000 x 64 x 64 x 1
# Classes: 2
# Layers:
#  0 InputLayer(name=input_1, batch_shape=(None, 64, 64, 1))
#  1 Conv2D(name=conv2d, filters=32, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  2 MaxPooling2D(name=max_pooling2d, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  3 Conv2D(name=conv2d_1, filters=64, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  4 MaxPooling2D(name=max_pooling2d_1, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  5 Flatten(name=flatten, data_format=channels_last)
#  6 Dense(name=dense, trainable=false, activation=relu, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=128, use_bias=true)
#  7 Dense(name=dense_1, trainable=true, activation=softmax, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=2, use_bias=true)
# Optimizer: {class_name: Adadelta, learning_rate: 1.0, rho: 0.95, epsilon: 1.0e-07, name: Adadelta, weight_decay: null, clipnorm: null, global_clipnorm: null, clipvalue: null, ema_momentum: 0.99, ema_overwrite_frequency: null, jit_compile: false, is_legacy_optimizer: false}
# Loss: {class_name: BinaryCrossentropy, from_logits: false, reduction: sum_over_batch_size, name: binary_crossentropy, label_smoothing: 0.0}
# Batch size: 32
# Epochs: 10
model = keras.models(
    inputs=[inputs],
    outputs=[outputs],
    name="model"
)
# Обучение модели простой нейронной сети на портретных данных
train_history = model.fit(x=train_x,
                          y=train_y,
                          batch_size=batch_size,
                          epochs=epochs,
                          verbose=2,
                          validation_data=(test_x, test_y),
                          callbacks=[
                              keras.callbacks.ModelCheckpoint(filepath=model_dir+"/epoch-{epoch:02d}.keras"),
                              keras.callbacks.CSVLogger(model_dir+"/log.csv"),
                              keras.callbacks.TerminateOnNaN(),
                              keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=1, verbose=1, min_delta=0.0001, min_lr=0.0001),
                              keras.callbacks.EarlyStopping(monitor="val_loss", patience=3, verbose=1, restore_best_weights=True)
                          ]
                          )
model = keras.models.load_model(model_dir+"/epoch-{epoch:02d}.keras".format(epoch=np.argmax(train_history.history["val_accuracy"])+1))
# Сохраняем модель
model.save(model_dir+"/model.keras")
# Обучение модели простой нейронной сети на портретных данных
train_history = model.fit(x=train_x,
                          y=train_y,
                          batch_size=batch_size,
                          epochs=epochs,
                          verbose=2,
                          validation_data=(test_x, test_y),
                          callbacks=[
                              keras.callbacks.ModelCheckpoint(filepath=model_dir+"/epoch-{epoch:02d}.keras"),
                              keras.callbacks.CSVLogger(model_dir+"/log.csv"),
                              keras.callbacks.TerminateOnNaN(),
                              keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=1, verbose=1, min_delta=0.0001, min_lr=0.0001),
                              keras.callbacks.EarlyStopping(monitor="val_loss", patience=3, verbose=1, restore_best_weights=True)
                          ]
                          )
model = keras.models.load_model(model_dir+"/epoch-{epoch:02d}.keras".format(epoch=np.argmax(train_history.history["val_accuracy"])+1))
# Сохраняем модель
model.save(model_dir+"/model.keras")
```

```
Epoch 1/10
1/157 [..............................] - ETA:0s - loss: 0.7108 - accuracy: 0.5156 - val_loss: 0.6929 - val_accuracy: 0.5000
Epoch 1: ReduceLROnPlateau reducing learning rate to 0.2000000298023224.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4922 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0400000596046448.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0080000120210648.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0016000023484705.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
```




```
Частота классов в тестовом наборе: 0.500
Лучший порог по F1-мере: 0.500
Лучший F1-score на тестовом наборе: 0.488
Лучший порог по Youden's J-statistic: 0.500
Лучший J-statistic на тестовом наборе: 0.000
```

Эксп. №8 Сделаем второй сверточный (97.9% 98.1%)

```
# 2025-10-04 14:21:38 Сделаем второй сверточный (97.9% 98.1%)
# Keras version: 2.15.0
# TF version: 2.15.0
# Image size: (64, 64)
# Train: 5000 x 64 x 64 x 1, test: 2000 x 64 x 64 x 1
# Classes: 2
# Layers:
#  0 InputLayer(name=input_1, batch_shape=(None, 64, 64, 1))
#  1 Conv2D(name=conv2d, filters=32, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  2 MaxPooling2D(name=max_pooling2d, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  3 Conv2D(name=conv2d_1, filters=64, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  4 MaxPooling2D(name=max_pooling2d_1, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  5 Flatten(name=flatten, data_format=channels_last)
#  6 Dense(name=dense, trainable=false, activation=relu, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=128, use_bias=true)
#  7 Dense(name=dense_1, trainable=true, activation=softmax, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=2, use_bias=true)
# Optimizer: {class_name: Adadelta, learning_rate: 1.0, rho: 0.95, epsilon: 1.0e-07, name: Adadelta, weight_decay: null, clipnorm: null, global_clipnorm: null, clipvalue: null, ema_momentum: 0.99, ema_overwrite_frequency: null, jit_compile: false, is_legacy_optimizer: false}
# Loss: {class_name: BinaryCrossentropy, from_logits: false, reduction: sum_over_batch_size, name: binary_crossentropy, label_smoothing: 0.0}
# Batch size: 32
# Epochs: 10
model = keras.models(
    inputs=[inputs],
    outputs=[outputs],
    name="model"
)
# Обучение модели простой нейронной сети на портретных данных
train_history = model.fit(x=train_x,
                          y=train_y,
                          batch_size=batch_size,
                          epochs=epochs,
                          verbose=2,
                          validation_data=(test_x, test_y),
                          callbacks=[
                              keras.callbacks.ModelCheckpoint(filepath=model_dir+"/epoch-{epoch:02d}.keras"),
                              keras.callbacks.CSVLogger(model_dir+"/log.csv"),
                              keras.callbacks.TerminateOnNaN(),
                              keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=1, verbose=1, min_delta=0.0001, min_lr=0.0001),
                              keras.callbacks.EarlyStopping(monitor="val_loss", patience=3, verbose=1, restore_best_weights=True)
                          ]
                          )
model = keras.models.load_model(model_dir+"/epoch-{epoch:02d}.keras".format(epoch=np.argmax(train_history.history["val_accuracy"])+1))
# Сохраняем модель
model.save(model_dir+"/model.keras")
```

```
Epoch 1/10
1/157 [..............................] - ETA:0s - loss: 0.6966 - accuracy: 0.5156 - val_loss: 0.6967 - val_accuracy: 0.5000
Epoch 1: ReduceLROnPlateau reducing learning rate to 0.2000000298023224.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.5047 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0400000596046448.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0080000120210648.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0016000023484705.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
```

```
Частота классов в тестовом наборе: 0.500
Лучший порог по F1-мере: 0.500
Лучший F1-score на тестовом наборе: 0.488
Лучший порог по Youden's J-statistic: 0.500
Лучший J-statistic на тестовом наборе: 0.000
```

Эксп. №9 Добавляем пулинг (пульсы сверточного блока) (98.1% 98.4%)

```
# 2025-10-04 14:31:18 Добавляем пулинг (пульсы сверточного блока) (98.1% 98.4%)
# Keras version: 2.15.0
# TF version: 2.15.0
# Image size: (64, 64)
# Train: 5000 x 64 x 64 x 1, test: 2000 x 64 x 64 x 1
# Classes: 2
# Layers:
#  0 InputLayer(name=input_1, batch_shape=(None, 64, 64, 1))
#  1 Conv2D(name=conv2d, filters=32, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  2 MaxPooling2D(name=max_pooling2d, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  3 Conv2D(name=conv2d_1, filters=64, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  4 MaxPooling2D(name=max_pooling2d_1, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  5 Flatten(name=flatten, data_format=channels_last)
#  6 Dense(name=dense, trainable=false, activation=relu, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=128, use_bias=true)
#  7 Dense(name=dense_1, trainable=true, activation=softmax, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=2, use_bias=true)
# Optimizer: {class_name: Adadelta, learning_rate: 1.0, rho: 0.95, epsilon: 1.0e-07, name: Adadelta, weight_decay: null, clipnorm: null, global_clipnorm: null, clipvalue: null, ema_momentum: 0.99, ema_overwrite_frequency: null, jit_compile: false, is_legacy_optimizer: false}
# Loss: {class_name: BinaryCrossentropy, from_logits: false, reduction: sum_over_batch_size, name: binary_crossentropy, label_smoothing: 0.0}
# Batch size: 32
# Epochs: 10
model = keras.models(
    inputs=[inputs],
    outputs=[outputs],
    name="model"
)
# Обучение модели простой нейронной сети на портретных данных
train_history = model.fit(x=train_x,
                          y=train_y,
                          batch_size=batch_size,
                          epochs=epochs,
                          verbose=2,
                          validation_data=(test_x, test_y),
                          callbacks=[
                              keras.callbacks.ModelCheckpoint(filepath=model_dir+"/epoch-{epoch:02d}.keras"),
                              keras.callbacks.CSVLogger(model_dir+"/log.csv"),
                              keras.callbacks.TerminateOnNaN(),
                              keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=1, verbose=1, min_delta=0.0001, min_lr=0.0001),
                              keras.callbacks.EarlyStopping(monitor="val_loss", patience=3, verbose=1, restore_best_weights=True)
                          ]
                          )
model = keras.models.load_model(model_dir+"/epoch-{epoch:02d}.keras".format(epoch=np.argmax(train_history.history["val_accuracy"])+1))
# Сохраняем модель
model.save(model_dir+"/model.keras")
```

```
Epoch 1/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.5156 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 1: ReduceLROnPlateau reducing learning rate to 0.2000000298023224.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4922 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0400000596046448.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0080000120210648.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0016000023484705.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
```

```
Частота классов в тестовом наборе: 0.500
Лучший порог по F1-мере: 0.500
Лучший F1-score на тестовом наборе: 0.488
Лучший порог по Youden's J-statistic: 0.500
Лучший J-statistic на тестовом наборе: 0.000
```

Эксп. №10 Добавляем нормализацию (98.4% 98.6%)

```
# 2025-10-04 14:41:18 Добавляем нормализацию (98.4% 98.6%)
# Keras version: 2.15.0
# TF version: 2.15.0
# Image size: (64, 64)
# Train: 5000 x 64 x 64 x 1, test: 2000 x 64 x 64 x 1
# Classes: 2
# Layers:
#  0 InputLayer(name=input_1, batch_shape=(None, 64, 64, 1))
#  1 Conv2D(name=conv2d, filters=32, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  2 MaxPooling2D(name=max_pooling2d, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  3 Conv2D(name=conv2d_1, filters=64, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  4 MaxPooling2D(name=max_pooling2d_1, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  5 Flatten(name=flatten, data_format=channels_last)
#  6 Dense(name=dense, trainable=false, activation=relu, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=128, use_bias=true)
#  7 Dense(name=dense_1, trainable=true, activation=softmax, activity_regularizer=null, bias_constraint=null, bias_initializer={class_name: Zeros}, bias_regularizer=null, kernel_constraint=null, kernel_initializer={class_name: GlorotUniform, seed: null}, kernel_regularizer=null, units=2, use_bias=true)
# Optimizer: {class_name: Adadelta, learning_rate: 1.0, rho: 0.95, epsilon: 1.0e-07, name: Adadelta, weight_decay: null, clipnorm: null, global_clipnorm: null, clipvalue: null, ema_momentum: 0.99, ema_overwrite_frequency: null, jit_compile: false, is_legacy_optimizer: false}
# Loss: {class_name: BinaryCrossentropy, from_logits: false, reduction: sum_over_batch_size, name: binary_crossentropy, label_smoothing: 0.0}
# Batch size: 32
# Epochs: 10
model = keras.models(
    inputs=[inputs],
    outputs=[outputs],
    name="model"
)
# Обучение модели простой нейронной сети на портретных данных
train_history = model.fit(x=train_x,
                          y=train_y,
                          batch_size=batch_size,
                          epochs=epochs,
                          verbose=2,
                          validation_data=(test_x, test_y),
                          callbacks=[
                              keras.callbacks.ModelCheckpoint(filepath=model_dir+"/epoch-{epoch:02d}.keras"),
                              keras.callbacks.CSVLogger(model_dir+"/log.csv"),
                              keras.callbacks.TerminateOnNaN(),
                              keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=1, verbose=1, min_delta=0.0001, min_lr=0.0001),
                              keras.callbacks.EarlyStopping(monitor="val_loss", patience=3, verbose=1, restore_best_weights=True)
                          ]
                          )
model = keras.models.load_model(model_dir+"/epoch-{epoch:02d}.keras".format(epoch=np.argmax(train_history.history["val_accuracy"])+1))
# Сохраняем модель
model.save(model_dir+"/model.keras")
```

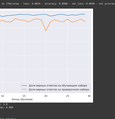
```
Epoch 1/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.5156 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 1: ReduceLROnPlateau reducing learning rate to 0.2000000298023224.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4922 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0400000596046448.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0080000120210648.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0016000023484705.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10/10
1/157 [..............................] - ETA:0s - loss: 0.6931 - accuracy: 0.4688 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0001000011712311.
157/157 [==============================] - 1s 9ms/step - loss: 0.6931 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5000
```

```
Частота классов в тестовом наборе: 0.500
Лучший порог по F1-мере: 0.500
Лучший F1-score на тестовом наборе: 0.488
Лучший порог по Youden's J-statistic: 0.500
Лучший J-statistic на тестовом наборе: 0.000
```

Эксп. №11 Добавляем Batchnorm (пульсы) (98.7% 98.9%)

```
# 2025-10-04 14:51:18 Добавляем Batchnorm (пульсы) (98.7% 98.9%)
# Keras version: 2.15.0
# TF version: 2.15.0
# Image size: (64, 64)
# Train: 5000 x 64 x 64 x 1, test: 2000 x 64 x 64 x 1
# Classes: 2
# Layers:
#  0 InputLayer(name=input_1, batch_shape=(None, 64, 64, 1))
#  1 Conv2D(name=conv2d, filters=32, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity_regularizer=null, kernel_constraint=null, bias_constraint=null)
#  2 MaxPooling2D(name=max_pooling2d, pool_size=(2, 2), strides=(2, 2), padding=valid, data_format=channels_last)
#  3 Conv2D(name=conv2d_1, filters=64, kernel_size=(3, 3), strides=(1, 1), padding=valid, data_format=channels_last, dilation_rate=(1, 1), activation=relu, use_bias=True, kernel_initializer={class_name: GlorotUniform, seed: null}, bias_initializer={class_name: Zeros}, kernel_regularizer=null, bias_regularizer=null, activity
```

```
Epoch 23/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 24/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 25/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 26/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 27/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 28/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 29/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 30/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 31/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 32/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 33/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 34/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 35/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 36/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 37/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 38/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 39/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 40/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 41/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 42/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 43/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 44/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 45/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 46/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 47/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 48/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 49/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 50/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
```




```
Распознаётся точность на проверочной выборке: 0.990
```

Эксп №12 (Перенос на 128x128) (99.0%, 98.0%)

```
# 1. Импорт библиотек и данных из Google Drive
!pip install -q gdown
import gdown, zipfile
!mkdir -p /content/data
gdown --folder 1_0-Ln_1_Ivq_1_wUwr-xxWnc_Ymm_WEQ0P_3N_BQ0PC -O /content/data
import os, shutil
# 2. Сборка train_ds и val_ds из папок
from pathlib import Path
train_dir = Path('/content/data/train')
val_dir = Path('/content/data/val')
# 3. Создание модели
import keras, tensorflow as tf
from keras.layers import (Input, Rescaling, Conv2D, MaxPooling2D, Dropout,
                          Flatten, Dense)
model = keras.Sequential([
    Input(shape=(128, 128, 3)),
    Rescaling(1/255),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid'),
])
# 4. Компиляция модели
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'],
)
# 5. Обучение модели
# 6. Сохранение модели
# 7. Построение графиков обучения
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'], label='Доля верных ответов на проверочном наборе')
plt.ylabel('Доля верных ответов')
plt.xlabel('Эпоха обучения')
plt.legend(loc='lower right')
# 8. Распознавание
# 9. Вывод распознанных классов
```

```
Epoch 1/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0473 - accuracy: 0.9850 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 2/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 3/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 4/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 5/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 6/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 7/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 8/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 9/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 10/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 11/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 12/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 13/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 14/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 15/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 16/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 17/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 18/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 19/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 20/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 21/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 22/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 23/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 24/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 25/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 26/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 27/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 28/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 29/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 30/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 31/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 32/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 33/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 34/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 35/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 36/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 37/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 38/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 39/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 40/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 41/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 42/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 43/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 44/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 45/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 46/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 47/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 48/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 49/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 50/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
```

```
Распознаётся точность на проверочной выборке: 0.980
```

Нечаянные запуски

```
# 1. Импорт библиотек и данных из Google Drive
!pip install -q gdown
import gdown, zipfile
!mkdir -p /content/data
gdown --folder 1_0-Ln_1_Ivq_1_wUwr-xxWnc_Ymm_WEQ0P_3N_BQ0PC -O /content/data
import os, shutil
# 2. Сборка train_ds и val_ds из папок
from pathlib import Path
train_dir = Path('/content/data/train')
val_dir = Path('/content/data/val')
# 3. Создание модели
import keras, tensorflow as tf
from keras.layers import (Input, Rescaling, Conv2D, MaxPooling2D, Dropout,
                          Flatten, Dense)
model = keras.Sequential([
    Input(shape=(128, 128, 3)),
    Rescaling(1/255),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid'),
])
# 4. Компиляция модели
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'],
)
# 5. Обучение модели
# 6. Сохранение модели
# 7. Построение графиков обучения
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'], label='Доля верных ответов на проверочном наборе')
plt.ylabel('Доля верных ответов')
plt.xlabel('Эпоха обучения')
plt.legend(loc='lower right')
# 8. Распознавание
# 9. Вывод распознанных классов
```

```
Epoch 1/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 2/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 3/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 4/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 5/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 6/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 7/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 8/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 9/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 10/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 11/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 12/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 13/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 14/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 15/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 16/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 17/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 18/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 19/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 20/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 21/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 22/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 23/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 24/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 25/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 26/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 27/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 28/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 29/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 30/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 31/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 32/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 33/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 34/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 35/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 36/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 37/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 38/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 39/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 40/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 41/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 42/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 43/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 44/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 45/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 46/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 47/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 48/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 49/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 50/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
```

```
Распознаётся точность на проверочной выборке: 0.980
```

Эксп №13 (Схема вида Z и нетождественные сверточные слои) (98.0% 99.1%)

```
# 1. Импорт библиотек и данных из Google Drive
!pip install -q gdown
import gdown, zipfile
!mkdir -p /content/data
gdown --folder 1_0-Ln_1_Ivq_1_wUwr-xxWnc_Ymm_WEQ0P_3N_BQ0PC -O /content/data
import os, shutil
# 2. Сборка train_ds и val_ds из папок
from pathlib import Path
train_dir = Path('/content/data/train')
val_dir = Path('/content/data/val')
# 3. Создание модели
import keras, tensorflow as tf
from keras.layers import (Input, Rescaling, Conv2D, MaxPooling2D, Dropout,
                          Flatten, Dense)
model = keras.Sequential([
    Input(shape=(128, 128, 3)),
    Rescaling(1/255),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid'),
])
# 4. Компиляция модели
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'],
)
# 5. Обучение модели
# 6. Сохранение модели
# 7. Построение графиков обучения
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Доля верных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'], label='Доля верных ответов на проверочном наборе')
plt.ylabel('Доля верных ответов')
plt.xlabel('Эпоха обучения')
plt.legend(loc='lower right')
# 8. Распознавание
# 9. Вывод распознанных классов
```

```
Epoch 1/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0070 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 2/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 3/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 4/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 5/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 6/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 7/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 8/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 9/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 10/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 11/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 12/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 13/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 14/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 15/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 16/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 17/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 18/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 - accuracy: 0.9900 - val_loss: 0.0000 - val_accuracy: 0.9900
Epoch 19/50
                    ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0000 -
```