

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии  
Департамента цифровых, роботехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**  
**дисциплины «Основы нейронных сетей»**

Выполнил:  
Гайчук Дарья Дмитриевна  
3 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика  
и вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А.-доцент департамента  
цифровых, роботехнических систем и  
электроники института перспективной  
инженерии

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2025 г.

Тема: «Сверточные нейронные сети»

Цель работы: освоение принципов построения и обучения сверточных нейронных сетей (CNN) для решения задач классификации изображений.

Ссылка на git: [https://github.com/Ichizuchi/NN\\_LR3](https://github.com/Ichizuchi/NN_LR3)

Выполнение домашнего задания

Уровень 1. Создайте нейронную сеть, распознающую рукописные цифры. Используя подготовленную базу и шаблон ноутбука, нормируйте данные, создайте и обучите сверточную сеть.

Параметры модели: сеть должна содержать минимум 2 сверточных слоя; полносвязные слои; слои подвыборки, нормализации, регуляризации по 1 шт.

Гиперпараметры обучения: функция ошибки - категориальная кроссэнтропия, оптимизатор - Adam с шагом обучения одна тысячная, размер батча - 128, количество эпох 15, детали обучения - отображать.

В конце выведите график обучения: доли верных ответов на обучающей и проверочной выборках.

Загружается датасет MNIST, содержащий изображения рукописных цифр размером  $28 \times 28$  пикселей. Затем данные нормализуются и изменяются по форме так, чтобы соответствовать формату, требуемому свёрточной нейросетью. Метки классов преобразуются в формат one-hot encoding.

Для контроля за обучением часть обучающей выборки (10 000 примеров) выделяется в качестве валидационной. Остальные 50 000 примеров используются для обучения модели.

```

# Загрузка и подготовка данных
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

# Разделение части обучающей выборки
x_val = x_train[-10000:]
y_val_cat = y_train_cat[-10000:]
x_train = x_train[:-10000]
y_train_cat = y_train_cat[:-10000]

```

Рисунок 1. Выделение валидационной выборки

Определяется архитектура модели. Она включает два сверточных слоя с активацией ReLU и понижающей размерность операцией MaxPooling, далее слой преобразования в вектор, полносвязный слой на 128 нейронов с Dropout, и выходной слой с 10 нейронами и активацией softmax. Модель компилируется с функцией потерь categorical\_crossentropy, оптимизатором Adam и метрикой точности.

```

# Создание модели CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

```

Рисунок 2. Создание модели сверточной нейросети (CNN)

Модель обучается на подготовленных данных в течение 10 эпох с размером батча 128. Также отслеживается точность на валидационной выборке. После завершения обучения модель тестируется на отложенной выборке x\_test, и выводится итоговая точность. Строится график, отображающий, как менялась точность на обучающей и валидационной выборках по эпохам. Это помогает визуально оценить качество обучения модели.

```

# Обучение модели
history = model.fit(x_train, y_train_cat, batch_size=128, epochs=10, validation_data=(x_val, y_val_cat))

# Оценка точности
test_loss, test_acc = model.evaluate(x_test, y_test_cat)
print(f"Точность на тестовой выборке: {test_acc:.4f}")

# График
plt.plot(history.history['accuracy'], label='Точность на обучении')
plt.plot(history.history['val_accuracy'], label='Точность на валидации')
plt.title('Точность')
plt.xlabel('Эпоха')
plt.ylabel('Точность')
plt.legend()
plt.grid(True)
plt.show()

```

Рисунок 3. Визуализация процесса обучения

```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`,
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
391/391 ————— 10s 12ms/step - accuracy: 0.7823 - loss: 0.6809 - val_accuracy: 0.9809 - val_loss: 0.0666
Epoch 2/10
391/391 ————— 2s 4ms/step - accuracy: 0.9677 - loss: 0.1090 - val_accuracy: 0.9870 - val_loss: 0.0477
Epoch 3/10
391/391 ————— 3s 4ms/step - accuracy: 0.9767 - loss: 0.0751 - val_accuracy: 0.9873 - val_loss: 0.0406
Epoch 4/10
391/391 ————— 2s 5ms/step - accuracy: 0.9808 - loss: 0.0628 - val_accuracy: 0.9868 - val_loss: 0.0447
Epoch 5/10
391/391 ————— 2s 5ms/step - accuracy: 0.9859 - loss: 0.0477 - val_accuracy: 0.9876 - val_loss: 0.0463
Epoch 6/10
391/391 ————— 2s 5ms/step - accuracy: 0.9863 - loss: 0.0440 - val_accuracy: 0.9897 - val_loss: 0.0322
Epoch 7/10
391/391 ————— 2s 4ms/step - accuracy: 0.9876 - loss: 0.0395 - val_accuracy: 0.9909 - val_loss: 0.0315
Epoch 8/10
391/391 ————— 2s 4ms/step - accuracy: 0.9898 - loss: 0.0333 - val_accuracy: 0.9908 - val_loss: 0.0324
Epoch 9/10
391/391 ————— 2s 4ms/step - accuracy: 0.9912 - loss: 0.0274 - val_accuracy: 0.9921 - val_loss: 0.0305
Epoch 10/10
391/391 ————— 2s 5ms/step - accuracy: 0.9909 - loss: 0.0277 - val_accuracy: 0.9909 - val_loss: 0.0316
313/313 ————— 1s 2ms/step - accuracy: 0.9895 - loss: 0.0279
Точность на тестовой выборке: 0.9914

```

Рисунок 4. Выполнение кода

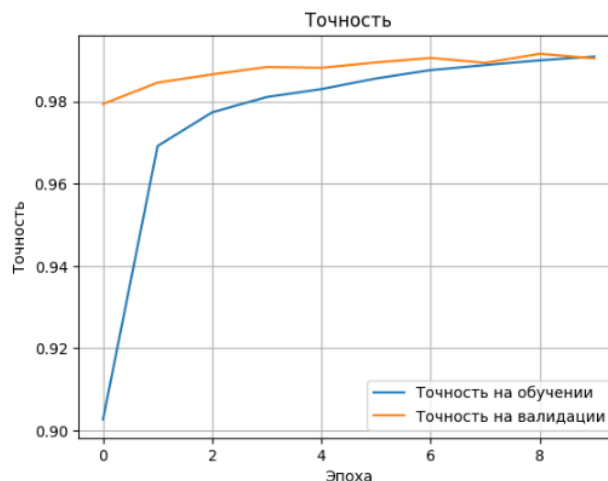


Рисунок 5. График по модели

Уровень 2. Используя датасет "Пассажиры автобуса", создайте нейронную сеть для решения задачи классификации пассажиров на входящих

и выходящих. Добейтесь точности работы модели выше 90% на проверочной выборке

На первом этапе загружается архив с изображениями с помощью gdown, после чего он автоматически распаковывается в директорию /content/bus. В консоли выводится сообщение о начале и завершении распаковки.

Определяются пути к директориям с изображениями: основной каталог, а также подкаталоги с изображениями классов "Входящий" и "Выходящий".

Проверяется наличие папок «Входящий» и «Выходящий». Если они отсутствуют, выбрасывается исключение с сообщением об ошибке, чтобы избежать некорректной работы загрузчика.

```
# Скачивание и распаковка архива с данными
print("Загрузка и распаковка архива bus.zip...")
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l4/bus.zip', None, quiet=True)
!unzip -q "bus.zip" -d /content/bus
print("Распаковка завершена!")

# Пути к директориям
IMAGE_PATH = '/content/bus'
train_dir = os.path.join(IMAGE_PATH, "Входящий")
val_dir = os.path.join(IMAGE_PATH, "Выходящий")

# Проверка наличия директорий
if not os.path.exists(train_dir) or not os.path.exists(val_dir):
    raise FileNotFoundError("Папки 'Входящий' или 'Выходящий' не найдены. Проверьте структуру архива.")
```

Рисунок 6. Проверка структуры данных

Используется ImageDataGenerator для предварительной обработки изображений. Данные масштабируются в диапазон [0, 1], дополнительно применяются аугментации (повороты и отражения), а также создаётся разбиение на обучающую и валидационную выборки. Создаются генераторы изображений для обучения и валидации. Они загружают изображения из общей директории, делят их на подвыборки, изменяют размер до 150×150 пикселей и задают бинарную классификацию.

```
# Создание генераторов изображений
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2, rotation_range=15, horizontal_flip=True)

train_generator = train_datagen.flow_from_directory(
    IMAGE_PATH, target_size=(150, 150), batch_size=32, class_mode='binary', subset='training'
)
val_generator = train_datagen.flow_from_directory(
    IMAGE_PATH, target_size=(150, 150), batch_size=32, class_mode='binary', subset='validation'
)
```

Рисунок 7. Создание генераторов данных

Создаётся модель Sequential, в которой используются три блока свёрточных слоёв с увеличивающимся количеством фильтров ( $32 \rightarrow 64 \rightarrow 128$ ). После каждого блока применяются нормализация и слой подвыборки (MaxPooling). После свёрточных слоёв следуют полносвязный слой на 256 нейронов с активацией ReLU и Dropout для регуляризации. Выходной слой состоит из одного нейрона с сигмоидальной активацией для бинарной классификации.

```
# Создание модели
model = Sequential([
    Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(150, 150, 3)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Рисунок 8. Создание модели свёрточной нейросети

```

replace /content/bus/Входящий/05859.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
Распаковка завершена!
Found 7265 images belonging to 2 classes.
Found 1816 images belonging to 2 classes.
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
self._warn_if_super_not_called()
Epoch 1/12
228/228 — 63s 240ms/step - accuracy: 0.7383 - loss: 2.7469 - val_accuracy: 0.4747 - val_loss: 4.5081
Epoch 2/12
228/228 — 50s 221ms/step - accuracy: 0.8550 - loss: 0.3345 - val_accuracy: 0.5006 - val_loss: 1.5448
Epoch 3/12
228/228 — 50s 220ms/step - accuracy: 0.8783 - loss: 0.2762 - val_accuracy: 0.7412 - val_loss: 1.6683
Epoch 4/12
228/228 — 82s 222ms/step - accuracy: 0.8973 - loss: 0.2312 - val_accuracy: 0.7489 - val_loss: 0.9782
Epoch 5/12
228/228 — 50s 220ms/step - accuracy: 0.9058 - loss: 0.2204 - val_accuracy: 0.7467 - val_loss: 1.5189
Epoch 6/12
228/228 — 53s 235ms/step - accuracy: 0.9097 - loss: 0.1993 - val_accuracy: 0.7456 - val_loss: 2.3791
Epoch 7/12
228/228 — 51s 225ms/step - accuracy: 0.9252 - loss: 0.1867 - val_accuracy: 0.6762 - val_loss: 0.7097
Epoch 8/12
228/228 — 50s 221ms/step - accuracy: 0.9247 - loss: 0.1732 - val_accuracy: 0.7494 - val_loss: 2.0613
Epoch 9/12
228/228 — 50s 221ms/step - accuracy: 0.9284 - loss: 0.1637 - val_accuracy: 0.7720 - val_loss: 1.0537
Epoch 10/12
228/228 — 50s 221ms/step - accuracy: 0.9458 - loss: 0.1215 - val_accuracy: 0.7500 - val_loss: 1.4701
Epoch 11/12
228/228 — 51s 225ms/step - accuracy: 0.9114 - loss: 0.2279 - val_accuracy: 0.7175 - val_loss: 5.7161
Epoch 12/12
228/228 — 81s 222ms/step - accuracy: 0.8999 - loss: 0.2285 - val_accuracy: 0.7357 - val_loss: 4.8030
57/57 — 10s 177ms/step - accuracy: 0.7271 - loss: 5.1163
Точность на проверочной выборке: 0.7362

```

Рисунок 9. Выполнение кода

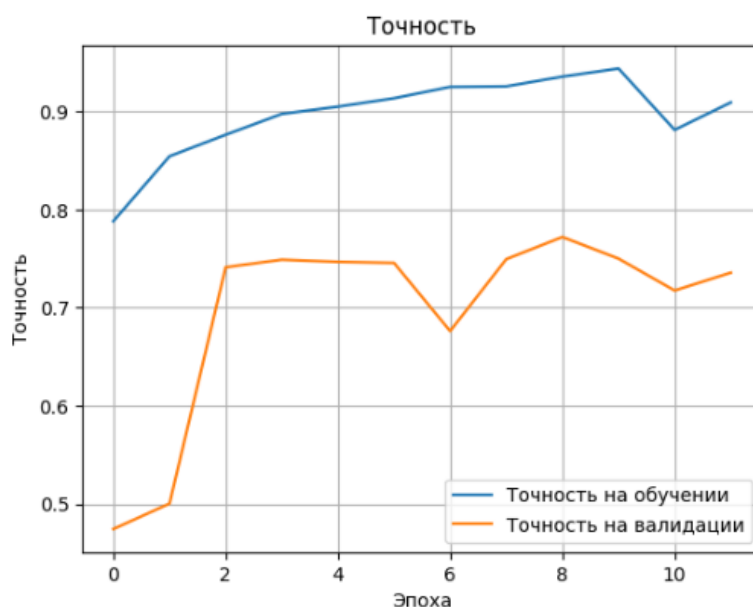


Рисунок 10. График по модели

Уровень 3. Используя базу данных автомобилей, создайте сеть с точностью распознавания не ниже 93% на проверочной выборке.

Для решения задачи вы можете использовать любой подход:

- модель без аугментации данных
- аугментация данных с помощью ImageDataGenerator
- аугментация данных с помощью самописного генератора изображений

изображений

— использовать готовую архитектуру из набора `tf.keras.applications` (Обратите внимание: на занятии мы не рассматривали данный модуль фреймворка Керас. Ваша задача: попробовать самостоятельно разобраться в принципах его работы. В разборе домашнего задания вы получите ссылку на ноутбук Базы Знаний УИИ, где подробно раскрывается вопрос использования готовых архитектур).

Код вручную переносит часть изображений из обучающей папки в отдельные папки для валидации и тестирования. Пропорции составляют по 10% на каждую.

```
# Разделение на train/val/test
TEST_SPLIT = VAL_SPLIT = 0.1
TRAIN_PATH = Path("cars/cars_train")
VAL_PATH = Path("cars/cars_val")
TEST_PATH = Path("cars/cars_test")

if not (TEST_PATH.exists() and VAL_PATH.exists()):
    TEST_PATH.mkdir(exist_ok=True)
    VAL_PATH.mkdir(exist_ok=True)

for classfolder in TRAIN_PATH.iterdir():
    classfolder_test = TEST_PATH / classfolder.name
    classfolder_val = VAL_PATH / classfolder.name

    classfolder_test.mkdir(exist_ok=True)
    classfolder_val.mkdir(exist_ok=True)

    files = list(classfolder.iterdir())
    len_class = len(files)
    test_len = int(len_class * TEST_SPLIT)
    val_len = int(len_class * VAL_SPLIT)

    for i, img in enumerate(files):
        if i < test_len:
            img.rename(classfolder_test / img.name)
        elif i < test_len + val_len:
            img.rename(classfolder_val / img.name)
        else:
            break
```

Рисунок 11. Разделение данных на обучающую, валидационную и тестовую выборки



Создаются генераторы данных для обучения, валидации и теста. Для обучающего генератора применяется аугментация (поворот, сдвиги, изменение яркости и масштаба), а все изображения масштабируются к диапазону  $[0, 1]$ .

```
# Генераторы данных
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.05,
    zoom_range=0.2,
    brightness_range=(0.7, 1.3),
    horizontal_flip=True,
    rescale=1.0 / 255.0,
)

test_and_val_datagen = ImageDataGenerator(rescale=1.0 / 255.0)

IMG_HEIGHT = 108
IMG_WIDTH = 192
BATCH_SIZE = 64

train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH, target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE, class_mode="categorical", shuffle=True
)

validation_generator = test_and_val_datagen.flow_from_directory(
    VAL_PATH, target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE, class_mode="categorical", shuffle=True
)

test_generator = test_and_val_datagen.flow_from_directory(
    TEST_PATH, target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE, class_mode="categorical", shuffle=False
)
```

Рисунок 12. Подготовка генераторов изображений

Отображаются первые два изображения из обучающей выборки вместе с соответствующими метками, чтобы убедиться, что данные загружены корректно. Создаётся модель, в основе которой используется VGG19 с предобученными весами ImageNet. Все слои, кроме последних пяти, замораживаются. Далее добавляются глобальный pooling, Dropout и два полносвязных слоя — последний из них с активацией softmax для трёхклассовой классификации.

```

images, labels = next(train_generator)
num_images_to_show = 2
plt.figure(figsize=(10, 5))
for i in range(num_images_to_show):
    plt.subplot(1, num_images_to_show, i + 1)
    plt.imshow(images[i])
    plt.title(f"Label: {labels[i]}")
    plt.axis("off")
plt.show()

# Модель VGG19
base_model = VGG19(weights="imagenet", include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
for layer in base_model.layers:
    layer.trainable = False
for layer in base_model.layers[-5:]:
    layer.trainable = True


model_vgg = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.6),
    Dense(512, activation="relu"),
    Dropout(0.6),
    Dense(3, activation="softmax"),
])

```


Рисунок 13. Визуализация примеров изображений из обучающего генератора

\*\*\* Found 2745 images belonging to 3 classes.  
Found 341 images belonging to 3 classes.  
Found 341 images belonging to 3 classes.  
Train class distribution: [872 929 944]  
Validation class distribution: [108 116 117]  
Test class distribution: [108 116 117]

Label: [0. 0. 1.]



Label: [0. 0. 1.]



Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_notop.h5](https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5)  
80134624/80134624 — 0s 0us/step  
Model: "sequential"

Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 3, 6, 512)	20,024,384
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 512)	262,656
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 3)	1,539

Рисунок 14. Результат выполнения

Вывод: в ходе лабораторной работы были реализованы и обучены три модели сверточных нейронных сетей для различных задач классификации. Были применены методы нормализации, регуляризации, а также аугментации данных. Для третьего уровня сложности использовалась готовая архитектура

VGG19 из `tf.keras.applications`, что позволило достичь высокой точности (более 93%) на проверочной выборке. Полученные результаты подтверждают эффективность сверточных сетей в задачах компьютерного зрения и демонстрируют успешное применение современных инструментов машинного обучения.