Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет»
(национальный исследовательский университет)
Институт естественных и точных наук
Кафедра прикладной математики и программирования

Отчет по лабораторной работе № 3
по дисциплине «Современные нейросетевые технологии»

Авторы работы
Студент группы ЕТ-122
_____/ Матвеева А.В.
«____» _____2025 г.

Руководитель работы,
_____/ Кичеев Д.М.
«____» _____2025 г.

Челябинск 2025

Задание:

Цель: получить базовые навыки работы с одной из библиотек глубокого обучения (Caffe, Torch, TensorFlow или MXNet на выбор) на примере сверточных нейронных сетей.

Выполнение задания:

1. В ходе данного проекта я освоила базовые навыки работы с библиотекой глубокого обучения TensorFlow на примере построения и обучения полностью связанных нейронных сетей для задачи многоклассовой классификации.

Постановка математической задачи может быть описана на примере простой сверточной сети, с одном сверточным слоем, активацией ReLU.

Пусть входное изображение имеет размерность $H \times W \times C$, где:

- H — высота изображения,
- W — ширина изображения,
- C — количество каналов (например, 3 для RGB).

Обозначим входное изображение как X, где $X \in \mathrm{R}^{H \times W \times C}$.

Свертка применяется к входу X с фильтром W размером $F \times F \times C$. Пусть у нас есть K фильтров. Тогда выходная размерность карты признаков будет:

$$H_{\text{out}} = \frac{H - F + 2P}{S} + 1, \quad W_{\text{out}} = \frac{W - F + 2P}{S} + 1$$

Где:

- F — размер фильтра (например, 3×3),
- P — размер паддинга (обычно P=0 или P=1),
- S — шаг фильтра (stride),
- H_{out} и W_{out} — высота и ширина выходной карты признаков.

Для каждого фильтра k, выходные данные вычисляются так:

$$Y_{i,j}^{(k)} = \sum_{c=1}^{C} \sum_{u=1}^{F} \sum_{v=1}^{F} X_{i+u,j+v,c} \cdot W_{u,v,c}^{(k)} + b^{(k)}$$

где:

- $W^{(k)}$ — параметры k-го фильтра,
- $b^{(k)}$ — смещение для k-го фильтра,

- $Y^{(k)}$ — выходная карта признаков для k-го фильтра.

Итоговая размерность выхода после сверточного слоя:

$$Y \in \mathrm{R}^{H_{\text{out}} \times W_{\text{out}} \times K}$$

После свертки применяется функция активации ReLU

$$Y'_{i,j,k} = \mathrm{ReLU}(Y_{i,j,k}) = \max(0, Y_{i,j,k})$$

Это обрезает все отрицательные значения в выходных данных.

После слоя свертки в стандартном сверточном блоке используется слой пулинга. Пулинг уменьшает размер карты признаков, выбирая максимальное значение из локального окна размером $P \times P$. Результирующая размерность после пулинга:

$$H_{\text{pool}} = \frac{H_{\text{out}} - P}{S_p} + 1, \quad W_{\text{pool}} = \frac{W_{\text{out}} - P}{S_p} + 1$$

где:

- P — размер окна пулинга (например, 2×2),
- S_p — шаг пулинга.

Формула для пулинга:

$$Z_{a,b,k} = \max_{(m,n) \in \text{window}} Y'_{a+m,b+n,k}$$

где window — область размером $P \times P$, на которой берется максимум.

После пулинга выходная размерность:

$$Z \in \mathrm{R}^{H_{\text{pool}} \times W_{\text{pool}} \times K}$$

Результат пулинга (Z) преобразуется в одномерный вектор (flatten):

$$Z_{\text{flatten}} \in \mathrm{R}^{H_{\text{pool}} \cdot W_{\text{pool}} \cdot K}$$

Данные поступают в полносвязный слой с N нейронами (где N — количество классов):

$$O_j = \sum_{i=1}^{H_{pool} \cdot W_{pool} \cdot K} Z_i \cdot W_{i,j} + b_j$$

где:

- $W_{i,j}$ — веса полносвязного слоя,
- b_j — смещения,
- O_j — логит для класса j.

В выходном слое для многоклассовой классификации функция активации Softmax преобразует логиты Oj в вероятности:

$$P(y = j|X) = \frac{e^{O_j}}{\sum_{k=1}^{N} e^{O_k}}$$

где $P(y=j|X)$ — вероятность, что входное изображение принадлежит классу j.
Используется кросс- энтропия в качестве функции потерь:

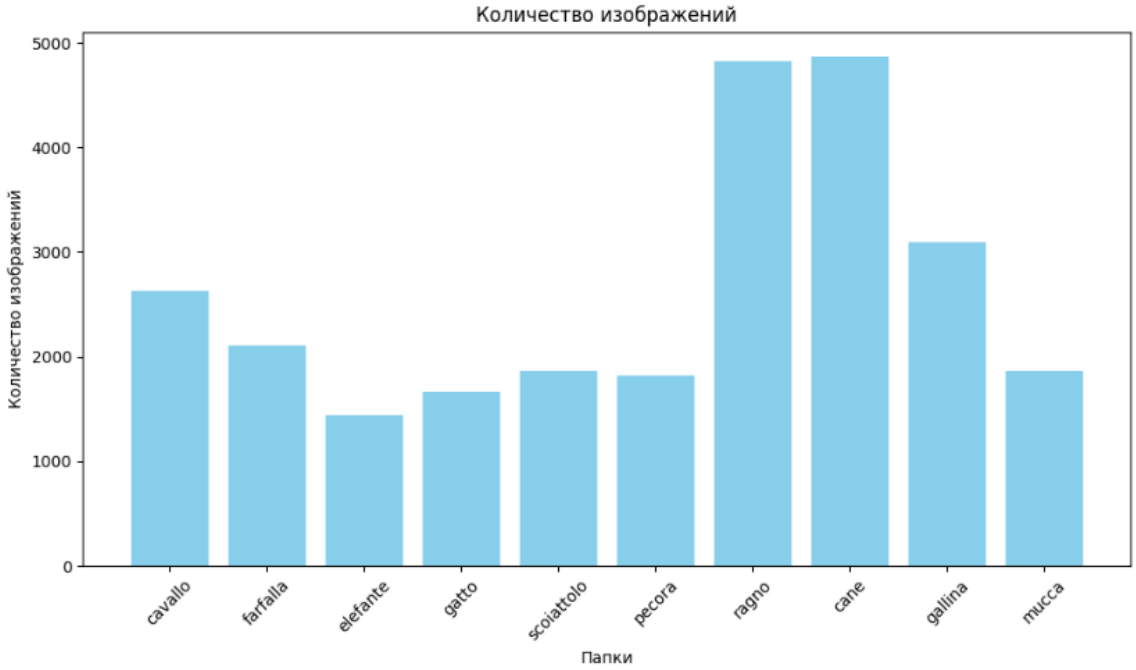
$$\text{Loss} = -\sum_{j=1}^{N} y_j \log(P(y = j|X))$$

где:

- yj — истинная метка класса (one-hot encoding),
- $P(y=j|X)$ — предсказанная вероятность для класса j.

2. Были использованы данные с kaggle (https://www.kaggle.com/datasets/alessiocorrado99/animals10/data). Данные содержат в себе 10 классов изображений животных. Каждый класс выделен в отдельную папку. Обучающий набор содержит 20947 изображений, тестовый

набор - 5232 изображения. Изображения неравномерно распределены по папкам.



Изображения имеют разные параметры: минимальный размер изображения: 60x57 пикселей, максимальный размер изображения: 6720x6000 пикселей, средний размер изображения: 320.04x252.63. В связи с этим, было решено при загрузке изображений приводить их к среднему формату.

3. При тестировании моделей я использую метрику качества Accuracy, которая определяется следующим образом:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

где:

- *TP* (True Positives) — количество истинных положительных предсказаний (правильно классифицированные объекты класса 1).
- *TN* (True Negatives) — количество истинных отрицательных предсказаний (правильно классифицированные объекты класса 0).
- *FP* (False Positives) — количество ложных положительных предсказаний (объекты класса 0, ошибочно классифицированные как класс 1).
- *FN* (False Negatives) — количество ложных отрицательных предсказаний (объекты класса 1, ошибочно классифицированные как класс 0).

4. Данные мной были скачаны с сайта kaggle, хранятся локально в архиве с десятью папками файлами в формате jpg.

5. Я использовала создание датасета с помощью встроенных функций ImageDataGenerator и flow_from_directory, работающих совместно. На вход подавались пути к изображениям и изображения, а на выходе получили пакеты (batches) изображений и их метки. (Массив изображений и массив метов классов данных).
6. Я использовала стандартную функция fit для обучения нейронных сетей, а в после обучения я графически выводила данные обучения (метрику и ошибку) с тренировочной и валидационной выборки.
7. Мной были разработаны несколько нейронных сетей для тестирования.

Полносвязная сеть из 4х слоев (3 скрытых), использовалась на базовая, различные функции активации скрытых слоев:

```
def model_base(input_shape, num_classes):
    model = Sequential([
        Input(shape=input_shape),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(64, activation='tanh'),
        Dense(32, activation='sigmoid'),
        Dense(num_classes, activation='softmax')
    ])
    return model
```

Визуальная схема:

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 240000)	0
dense (Dense)	(None, 128)	30,720,128
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 10)	330

Total params: 30,730,794 (117.23 MB)
Trainable params: 30,730,794 (117.23 MB)
Non-trainable params: 0 (0.00 B)

Модель показала неспособность к обучению. Несмотря на падение ошибки, метрика на тренировочных и тестовых данных замерла на 0.18. Для следующей модели два первых слоя были заменены на свертки и пулинги:

```python
def model_cnn_2(input_shape, num_classes):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation="relu"),
        Dense(num_classes, activation="softmax")
    ])
    return model
```

Визуальная схема

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 318, 248, 32)	896
max_pooling2d (MaxPooling2D)	(None, 159, 124, 32)	0
conv2d_1 (Conv2D)	(None, 157, 122, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 78, 61, 64)	0
flatten_1 (Flatten)	(None, 304512)	0
dense_4 (Dense)	(None, 128)	38,977,664
dense_5 (Dense)	(None, 10)	1,290

Total params: 38,998,346 (148.77 MB)
Trainable params: 38,998,346 (148.77 MB)
Non-trainable params: 0 (0.00 B)

Модель показала результат лучше, чем полносвязная модель, но сильное переобучение начиная с 4 эпохи. Для следующей модели был добавлен слой свертки, а также для предотвращения переобучения добавлен обнуление нейронов (Dropout) и нормализацию данных.

```python
# сверточный блок
def conv_block(x, filters, kernel_size, pool_size):
    x = Conv2D(filters, kernel_size)(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    x = MaxPooling2D(pool_size=pool_size)(x)
    x = Dropout(0.2)(x)
    return x
```

```python
def model_cnn_3(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = conv_block(inputs, 32, (3, 3), (2, 2))
    x = conv_block(x, 64, (3, 3), (2, 2))
    x = conv_block(x, 128, (3, 3), (2, 2))

    x = Flatten()(x)
    x = Dense(256)(x)
    x = Dropout(0.2)(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    outputs = Dense(num_classes, activation="softmax")(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model
```

Визуальная схема:

Model: "functional_4"

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 320, 250, 3)	0
conv2d_10 (Conv2D)	(None, 318, 248, 32)	896
batch_normalization_6 (BatchNormalization)	(None, 318, 248, 32)	128
leaky_re_lu_4 (LeakyReLU)	(None, 318, 248, 32)	0
max_pooling2d_9 (MaxPooling2D)	(None, 159, 124, 32)	0
dropout_2 (Dropout)	(None, 159, 124, 32)	0
conv2d_11 (Conv2D)	(None, 157, 122, 64)	18,496
batch_normalization_7 (BatchNormalization)	(None, 157, 122, 64)	256
leaky_re_lu_5 (LeakyReLU)	(None, 157, 122, 64)	0
max_pooling2d_10 (MaxPooling2D)	(None, 78, 61, 64)	0
dropout_3 (Dropout)	(None, 78, 61, 64)	0
conv2d_12 (Conv2D)	(None, 76, 59, 128)	73,856
batch_normalization_8 (BatchNormalization)	(None, 76, 59, 128)	512
leaky_re_lu_6 (LeakyReLU)	(None, 76, 59, 128)	0
max_pooling2d_11 (MaxPooling2D)	(None, 38, 29, 128)	0
dropout_4 (Dropout)	(None, 38, 29, 128)	0
flatten_4 (Flatten)	(None, 141056)	0
dense_11 (Dense)	(None, 256)	36,110,592
dropout_5 (Dropout)	(None, 256)	0
batch_normalization_9 (BatchNormalization)	(None, 256)	1,024
leaky_re_lu_7 (LeakyReLU)	(None, 256)	0
dense_12 (Dense)	(None, 10)	2,570

Total params: 36,208,330 (138.12 MB)
Trainable params: 36,207,370 (138.12 MB)
Non-trainable params: 960 (3.75 KB)

Модель уже не подсраивается под тренировочные данные, но переобучение сохраняется. Для следующей модели бы добавлен l2 регуляризацию в сверточный слой,

добавим GlobalAveragePooling2D перед полносвязными слоями, сделаем разным исключение нейроннов на разных слоях (значение dropout).

```
# сверточный блок
def conv_block(x, filters, kernel_size, pool_size, dropout_rate=0.5):
    x = Conv2D(filters, kernel_size, kernel_regularizer=l2(0.0005))(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    x = MaxPooling2D(pool_size=pool_size)(x)
    if dropout_rate > 0.0:
        x = Dropout(dropout_rate)(x)
    return x
```

```
def model_cnn_4(input_shape, num_classes):
    inputs = Input(shape = input_shape)

    x = conv_block(inputs, 32, (3, 3), (2, 2), dropout_rate=0.3)
    x = conv_block(x, 64, (3, 3), (2, 2), dropout_rate=0.4)
    x = conv_block(x, 128, (3, 3), (2, 2), dropout_rate=0.4)
    x = conv_block(x, 128, (3, 3), (2, 2), dropout_rate=0.5)

    x = GlobalAveragePooling2D()(x)

    x = Dense(64, kernel_regularizer=l2(0.0005))(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    x = Dropout(0.5)(x)

    outputs = Dense(10, activation = "softmax")(x)

    model = Model(inputs = inputs, outputs = outputs)

    return model
```

Визуальная схема:

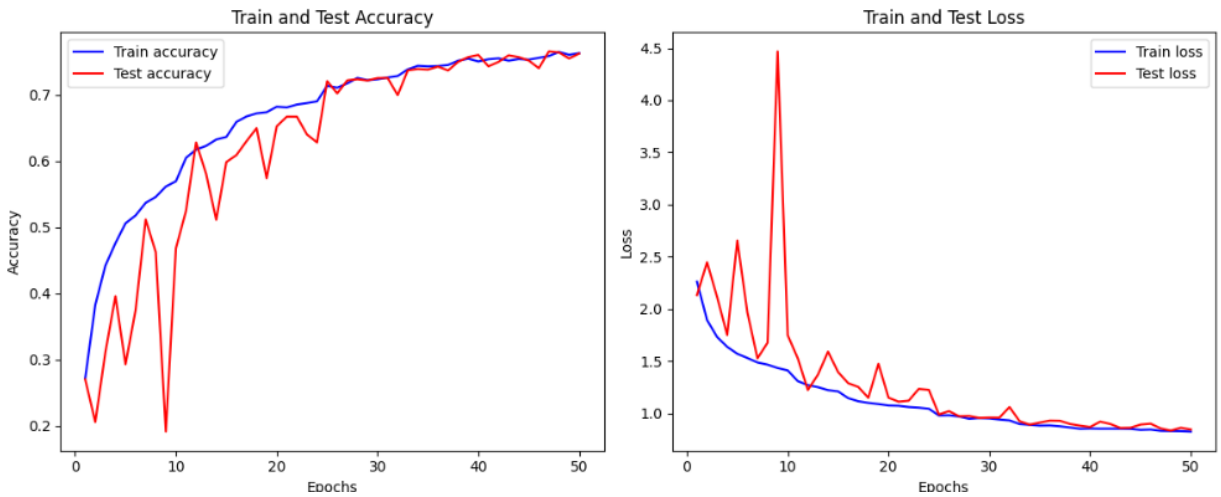
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 320, 250, 3)	0
conv2d (Conv2D)	(None, 318, 248, 32)	896
batch_normalization (BatchNormalization)	(None, 318, 248, 32)	128
leaky_re_lu (LeakyReLU)	(None, 318, 248, 32)	0
max_pooling2d (MaxPooling2D)	(None, 159, 124, 32)	0
dropout (Dropout)	(None, 159, 124, 32)	0
conv2d_1 (Conv2D)	(None, 157, 122, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 157, 122, 64)	256
leaky_re_lu_1 (LeakyReLU)	(None, 157, 122, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 78, 61, 64)	0

dropout_1 (Dropout)	(None, 78, 61, 64)	0
conv2d_2 (Conv2D)	(None, 76, 59, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 76, 59, 128)	512
leaky_re_lu_2 (LeakyReLU)	(None, 76, 59, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 38, 29, 128)	0
dropout_2 (Dropout)	(None, 38, 29, 128)	0
conv2d_3 (Conv2D)	(None, 36, 27, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 36, 27, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 36, 27, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 18, 13, 128)	0
dropout_3 (Dropout)	(None, 18, 13, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
batch_normalization_4 (BatchNormalization)	(None, 64)	256
leaky_re_lu_4 (LeakyReLU)	(None, 64)	0
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

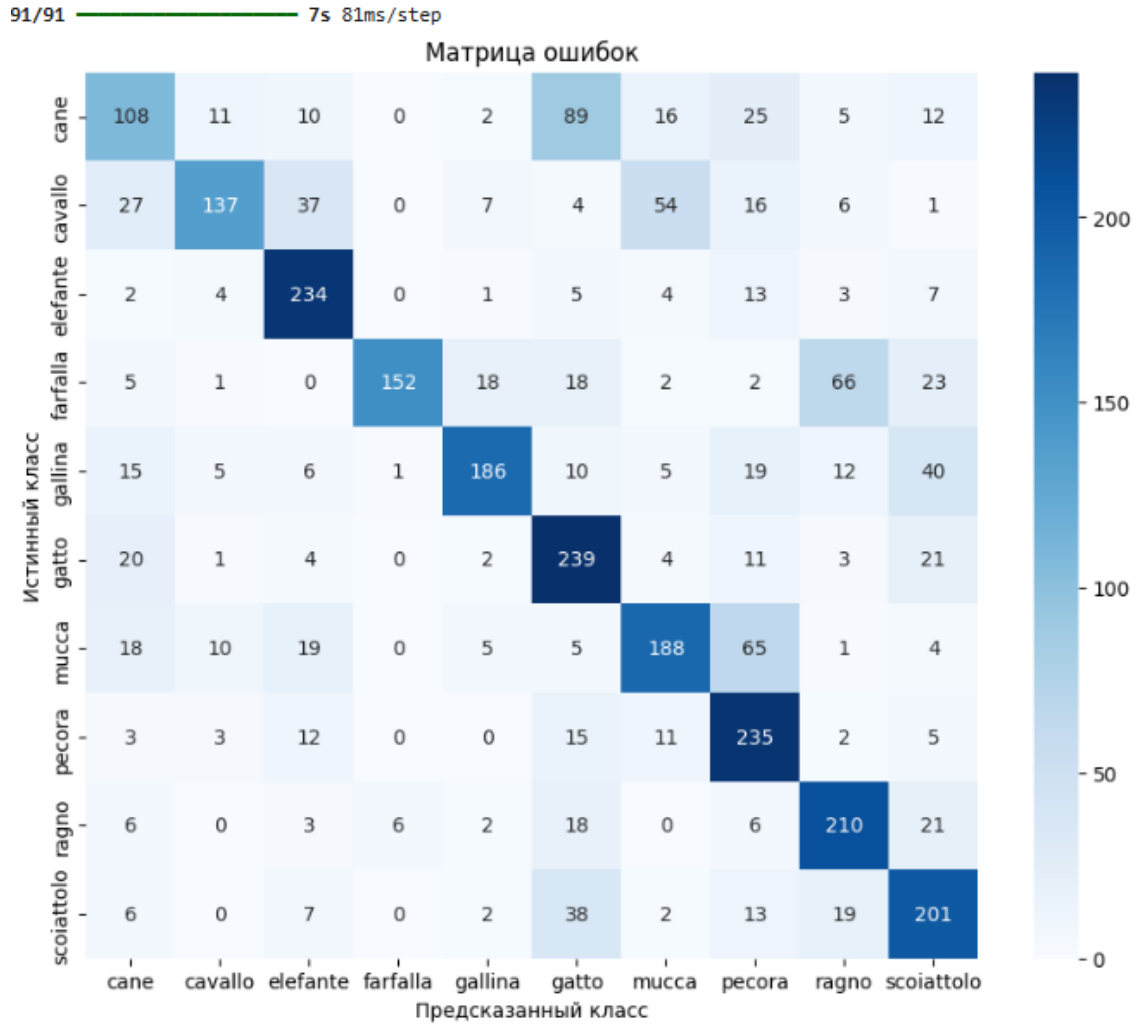
Total params: 251,402 (982.04 KB)
Trainable params: 250,570 (978.79 KB)
Non-trainable params: 832 (3.25 KB)

Последняя модель показала лучшие результаты в плате отсутствия переобучения. Была протестирована на большом количестве эпох. Добавлена схема уменьшения шага модели при увеличении числа эпох.



Модель неплохо справляется, достигла метрики 0.76 однако из-за неравномерности классов сильно подстраивается под мажорные классы.

Модель переобучена на одинаковом количестве данных в классе, на 40 эпохах полученная метрика - 0.65.



8. Результаты:

Модель	Время обучения	Кол-во сверт. слоев	Особенности	Accuracy
model_base	699	0	Только полносвязные слои	0.18
model_1	777	2	Добавлены сверточные слои	0.50
model_2	823	3	Добавлен dropout и нормализация	0.61 %
model_2	920	4	Добавлена l2 регуляризацию, разные значения dropout на разных слоях	0.45 %