Министерство цифрового развития

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт

по лабораторной работе № 5 «Нейронные сети для обработки изображений»

Выполнил:

студент группы ИП-213
Дмитриев Антон Александрович

Работу проверил: Преподаватель
Сороковых Дарья Анатольевна

Новосибирск 2025 г.

Введение (задание)

Задание

Тема: Нейронные сети для обработки изображений.

Цель: освоить на практике принципы построения, обучения и оценки нейронных сетей для решения базовых задач компьютерного зрения

1. Распределение вариантов

Формула: (Номер студента в списке) mod 6 + 1

Датасеты для вариантов:

1. MNIST - Рукописные цифры (10 классов)

2. CIFAR-10 - Объекты разных категорий (10 классов)

3. CIFAR-100 - Расширенная версия CIFAR-10 (100 классов)

4. Oxford Flowers 102 - Цветы (102 класса)

5. Stanford Dogs - Породы собак (120 классов)

6. Food-101 - Блюда питания (101 класс)

2. Создание архитектуры нейронной сети

Создайте и обучите сверточную нейронную сеть для решения задачи

классификации изображений.

Вариант архитектуры выбирается по формуле

(Номер студента в списке) mod 8 + 1

Вариант	Архитектура CNN	Регуляризация	Оптимизатор
1	2 сверточных слоя + 2 полносвязных	Dropout	Adam
2	3 сверточных слоя + 2 полносвязных	BatchNormalization	SGD
3	4 сверточных слоя + 2 полносвязных	Dropout	RMSprop
4	3 сверточных слоя + 3 полносвязных	GlobalAveragePooling	Adam
5	2 сверточных слоя + 2 полносвязных	GaussianDropout	Adam
6	4 сверточных слоя + 2 полносвязных	Dropout	SGD
7	3 сверточных слоя + 3 полносвязных	BatchNormalization + Dropout	Adam
8	2 сверточных слоя + 2 полносвязных	Dropout	RMSprop

3. Сделайте анализ графиков обучения и оценку точности на тестовой выборке

4. Отчёт

Оформите отчет в электронном виде, приложив ссылку на Jupyter Notebook/Google Colab, где код сопровождается краткими выводами по каждому шагу, электронный вид отчёта загрузите в формате pdf в ЭИОС.

Основная часть

1. Распределение вариантов

Номер в списке: 6

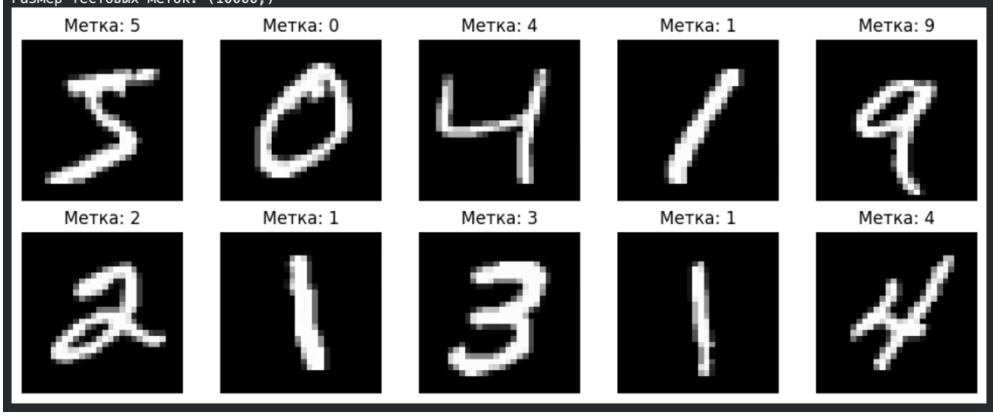
6 mod 6 + 1 = 1

Вариант: 1

Датасет: MNIST - Рукописные цифры (10 классов)

Ссылка: https://www.kaggle.com/datasets/hojjatk/mnist-dataset

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Загрузка данных MNIST...
Ошибка при загрузке данных: [Errno 2] No such file or directory: '/content/drive/MyDrive/lab5/train-labels-idx1-ubyte/train-labels-idx1-ubyte.gz'
Проверьте правильность путей к файлам.
Размер тренировочных изображений: (60000, 28, 28)
Размер тренировочных меток: (60000,)
Размер тестовых изображений: (10000, 28, 28)
Размер тестовых меток: (10000,)
```



2. Создание архитектуры нейронной сети

6 mod 8 + 1 = 7

Вариант: 7

Архитектура CNN: 3 сверточных слоя + 3 полносвязных

Регуляризация: BatchNormalization + Dropout

Оптимизатор: Adam

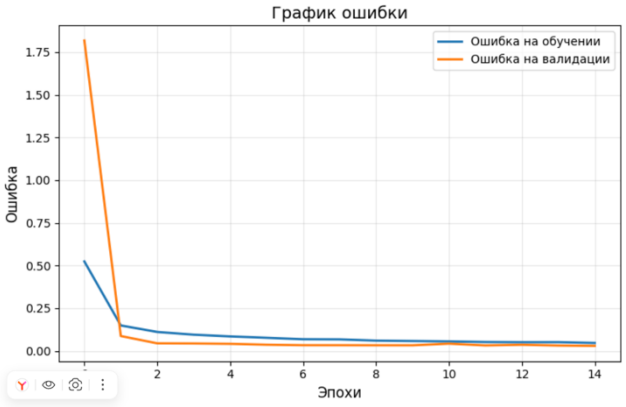
```
Форма тренировочных данных после преобразования: (60000, 28, 28, 1)
Форма тренировочных меток после преобразования: (60000, 10)
Model: "sequential_2"
```

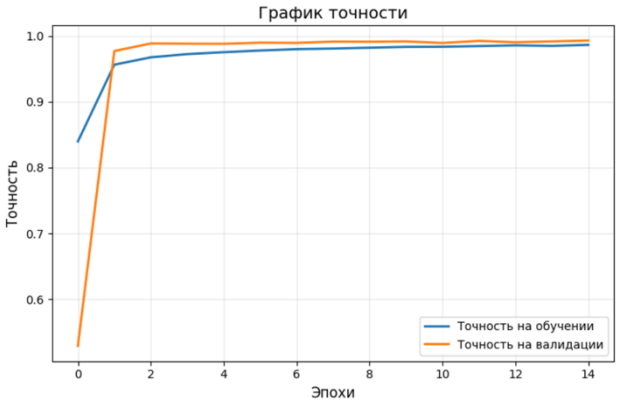
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization_10 (BatchNormalization)	(None, 26, 26, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_10 (Dropout)	(None, 13, 13, 32)	0
conv2d_7 (Conv2D)	(None, 11, 11, 64)	18,496
batch_normalization_11 (BatchNormalization)	(None, 11, 11, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_11 (Dropout)	(None, 5, 5, 64)	0
conv2d_8 (Conv2D)	(None, 3, 3, 128)	73,856
batch_normalization_12 (BatchNormalization)	(None, 3, 3, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 1, 1, 128)	0
dropout_12 (Dropout)	(None, 1, 1, 128)	0
flatten_2 (Flatten)	(None, 128)	0
dense_6 (Dense)	(None, 256)	33,024
batch_normalization_13 (BatchNormalization)	(None, 256)	1,024
dropout_13 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32,896
batch_normalization_14 (BatchNormalization)	(None, 128)	512
dropout_14 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290

```
 Total params: 162,314 (634.04 KB)
 Trainable params: 161,098 (629.29 KB)
 Non-trainable params: 1,216 (4.75 KB)
Начало обучения...
Epoch 1/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - accuracy: 0.6794 - loss: 1.0665 - val_accuracy: 0.5290 - val_loss: 1.8182
Epoch 2/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9514 - loss: 0.1657 - val_accuracy: 0.9772 - val_loss: 0.0875
Epoch 3/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9660 - loss: 0.1165 - val_accuracy: 0.9887 - val_loss: 0.0447
Epoch 4/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9728 - loss: 0.0943 - val_accuracy: 0.9883 - val_loss: 0.0439
Epoch 5/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9754 - loss: 0.0834 - val_accuracy: 0.9882 - val_loss: 0.0418
Epoch 6/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9779 - loss: 0.0757 - val_accuracy: 0.9898 - val_loss: 0.0367
Epoch 7/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9806 - loss: 0.0687 - val_accuracy: 0.9895 - val_loss: 0.0337
Epoch 8/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9808 - loss: 0.0687 - val_accuracy: 0.9915 - val_loss: 0.0337
Epoch 9/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9836 - loss: 0.0564 - val_accuracy: 0.9913 - val_loss: 0.0333
Epoch 10/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9838 - loss: 0.0577 - val_accuracy: 0.9918 - val_loss: 0.0331
Epoch 11/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9842 - loss: 0.0543 - val_accuracy: 0.9893 - val_loss: 0.0429
Epoch 12/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9851 - loss: 0.0497 - val_accuracy: 0.9927 - val_loss: 0.0325
Epoch 13/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9867 - loss: 0.0485 - val_accuracy: 0.9905 - val_loss: 0.0364
Epoch 14/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9842 - loss: 0.0533 - val_accuracy: 0.9918 - val_loss: 0.0318
Epoch 15/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9867 - loss: 0.0463 - val_accuracy: 0.9932 - val_loss: 0.0299
```

3. Анализ графиков обучения

```
Результаты на тестовой выборке:
Ошибка: 0.0317
Точность: 0.9922
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 954ms/step
```





Заключение

Ссылка на colab:

https://colab.research.google.com/drive/1bUcxxVt7Sjl8uUn8_hIc_6KtDJOBwt0M?usp=sharing

Код программы

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
import gzip
import os

def load_mnist_images(filename):
    if filename.endswith('.gz'):
        with gzip.open(filename, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=16)
    else:
        with open(filename, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=16)
    return data.reshape(-1, 28, 28)

def load_mnist_labels(filename):
    if filename.endswith('.gz'):
        with gzip.open(filename, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=8)
    else:
        with open(filename, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=8)
    return data

from google.colab import drive
drive.mount('/content/drive')

# Загрузка данных с указанных путей
print("Загрузка данных MNIST...")
try:
    x_train = load_mnist_images('/content/drive/MyDrive/lab5/train-images-
idx3-ubyte/train-images-idx3-ubyte')
    y_train = load_mnist_labels('/content/drive/MyDrive/lab5/train-labels-
idx1-ubyte/train-labels-idx1-ubyte.gz')
    x_test = load_mnist_images('/content/drive/MyDrive/lab5/t10k-images-
idx3-ubyte/t10k-images-idx3-ubyte.gz')
    y_test = load_mnist_labels('/content/drive/MyDrive/lab5/t10k-labels-
idx1-ubyte/t10k-labels-idx1-ubyte.gz')
    print("Данные успешно загружены!")
except Exception as e:
    print(f"Ошибка при загрузке данных: {e}")
    print("Проверьте правильность путей к файлам.")
    (x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()

print(f"Размер тренировочных изображений: {x_train.shape}")
print(f"Размер тренировочных меток: {y_train.shape}")
print(f"Размер тестовых изображений: {x_test.shape}")
```

```
print(f"Размер тестовых меток: {y_test.shape}")

# Визуализация нескольких примеров
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f'Метка: {y_train[i]}')
    plt.axis('off')
plt.tight_layout()
plt.show()

# Нормализация и подготовка данных
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, -1)  # (28, 28) -> (28, 28, 1)
x_test = np.expand_dims(x_test, -1)    # (28, 28) -> (28, 28, 1)

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

print(f"Форма тренировочных данных после преобразования: {x_train.shape}")
print(f"Форма тренировочных меток после преобразования:
{y_train_categorical.shape}")

# Создание модели CNN
model = keras.Sequential([
    # Первый сверточный блок
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    # Второй сверточный блок
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    # Третий сверточный блок
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    # Полносвязные слои
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
```

```
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(10, activation='softmax')
])

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

# Вывод архитектуры модели
model.summary()

# Обучение модели
print("Начало обучения...")
history = model.fit(x_train, y_train_categorical,
                    batch_size=128,
                    epochs=15,
                    validation_split=0.1,
                    verbose=1)
# Оценка на тестовых данных
test_loss, test_accuracy = model.evaluate(x_test, y_test_categorical,
verbose=0)
print(f'\nРезультаты на тестовой выборке:')
print(f'Ошибка: {test_loss:.4f}')
print(f'Точность: {test_accuracy:.4f}')

# Предсказание на нескольких тестовых примерах
predictions = model.predict(x_test[:10])
predicted_classes = np.argmax(predictions, axis=1)

# Визуализация предсказаний
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[i].squeeze(), cmap='gray')
    color = 'green' if y_test[i] == predicted_classes[i] else 'red'
    plt.title(f'Истина: {y_test[i]}, Предск.: {predicted_classes[i]}',
color=color)
    plt.axis('off')
plt.tight_layout()
plt.show()

# Построение графиков обучения
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Ошибка на обучении', linewidth=2)
```

```
plt.plot(history.history['val_loss'], label='Ошибка на валидации',
linewidth=2)
plt.title('График ошибки', fontsize=14)
plt.xlabel('Эпохи', fontsize=12)
plt.ylabel('Ошибка', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Точность на обучении',
linewidth=2)
plt.plot(history.history['val_accuracy'], label='Точность на валидации',
linewidth=2)
plt.title('График точности', fontsize=14)
plt.xlabel('Эпохи', fontsize=12)
plt.ylabel('Точность', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)

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