МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»

Кафедра инфокоммуникаций

«Введение в нейронные сети. Линейный слой (Dense)»

Отчет по лабораторной работе №1

по дисциплине «Обработка данных для построения систем искусственного интеллекта»

Выполнила студентка группы ИВТ-б-о-21-1
Горшков В.И. « »_____20__г.
Подпись студента_____
Работа защищена « »_____20__г.
Проверил канд. технических наук,
доцент, доцент кафедры инфокоммуникаций
Воронкин Р. А. _____
(подпись)

Ставрополь, 2024 г.

Цель работы: Изучить линейный слой (Dense) нейронных сетей, особенности выбора аппаратного ускорителя для Google Colab, нейрон смещения, оптимизаторы, функции активации и ошибок. Применить полученные знания для решения практических задач.

Ход работы:

Задача 1. Создайте систему компьютерного зрения, которая будет определять тип геометрической фигуры. Используя подготовленную базу и шаблон ноутбука проведите серию экспериментов по перебору гиперпараметров нейронной сети, распознающей три категории изображений (треугольник, круг, квадрат).

1. Поменяйте количество нейронов в сети, используя следующие значения:
   - один слой 10 нейронов
   - один слой 100 нейронов
   - один слой 5000 нейронов.
2. Поменяйте активационную функцию в скрытых слоях с relu на linear.
3. Поменяйте размеры batch_size:
   - 10
   - 100
   - 1000
4. Выведите на экран получившиеся точности.

Всего должно получиться 18 комбинаций указанных параметров.

Создайте сравнительную таблицу по результатам проведенных тестов.

Решение.

```
[ ]  # Подключение класса для создания нейронной сети прямого распространения
     from tensorflow.keras.models import Sequential
     # Подключение класса для создания полносвязного слоя
     from tensorflow.keras.layers import Dense, Flatten
     # Подключение оптимизатора
     from tensorflow.keras.optimizers import Adam
     # Подключение утилит для to_categorical
     from tensorflow.keras import utils
     # Подключение библиотеки для загрузки изображений
     from tensorflow.keras.preprocessing import image
     # Подключение библиотеки для работы с массивами
     import numpy as np
     # Подключение библиотек для отрисовки изображений
     import matplotlib.pyplot as plt
     # Подключение модуля для работы с файлами
     import os
     from sklearn.model_selection import train_test_split
     from tensorflow.keras.utils import to_categorical
     # Вывод изображения в ноутбуке, а не в консоли или файле
     %matplotlib inline
```

Рисунок 1.1 – Подключение библиотек

```
[ ]  # Распаковываем архив hw_light.zip в папку hw_light
     !unzip -q hw_light.zip
```

```
     replace hw_light/0/1.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
[ ]  # Путь к директории с базой
     base_dir = '/content/hw_light'
     # Создание пустого списка для загрузки изображений обучающей выборки
     x_train = []
     # Создание списка для меток классов
     y_train = []
     # Задание высоты и ширины загружаемых изображений
     img_height = 20
     img_width = 20
     # Перебор папок в директории базы
     for patch in os.listdir(base_dir):
         # Перебор файлов в папках
         for img in os.listdir(base_dir + '/' + patch):
             # Добавление в список изображений текущей картинки
             x_train.append(image.img_to_array(image.load_img(base_dir + '/' + patch + '/' + img,
                                                              target_size=(img_height, img_width),
                                                              color_mode='grayscale')))
             # Добавление в массив меток, соответствующих классам
             if patch == '0':
                 y_train.append(0)
             elif patch == '3':
                 y_train.append(1)
             else:
                 y_train.append(2)

     # Преобразование в numpy-массив загруженных изображений и меток классов
     x_train_org = np.array(x_train)
     y_train_org = np.array(y_train)
     # Вывод размерностей
     print('Размер массива x_train', x_train_org.shape)
     print('Размер массива y_train', y_train_org.shape)
```

```
Размер массива x_train (302, 20, 20, 1)
Размер массива y_train (302,)
```

Рисунок 1.2 – Загрузка изображений

```
[ ]  accuracy_lin = dict()
     accuracy_re = dict()
```

```
[ ]  # Номер картинки
     n = 34
     # Отрисовка картинки
     plt.imshow(x_train_org[n], cmap='gray')

     # Вывод n-й картинки
     plt.show()
```

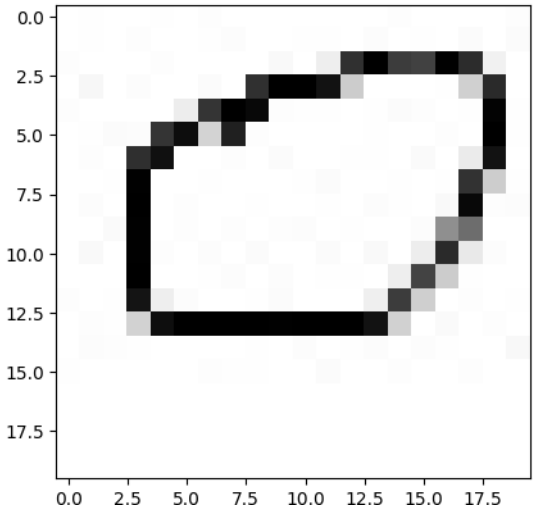


Рисунок 1.3 – Проверка содержимого массива изображений

```
[ ]  # Разделение данных на обучающий и тестовый наборы
     x_train_org, x_test_org, y_train_org, y_test_org = train_test_split(x_train_org, y_train_org, test_size=0.2, random_state=42)
```

```
[ ]  x_train = x_train_org.reshape(x_train_org.shape[0], -1)
     x_test = x_test_org.reshape(x_test_org.shape[0], -1)

     # Проверка результата
     print(f'Форма обучающих данных: {x_train.shape} -> {x_train_org.shape}')
     print(f'Форма  тестовых данных: {x_test.shape} ')
```

```
Форма обучающих данных: (241, 400) -> (241, 20, 20, 1)
Форма  тестовых данных: (61, 400)
```

```
[ ]  x_train = x_train.astype('float32') / 255.

     # Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
     x_test = x_test.astype('float32') / 255.
```

```
[ ]  CLASS_COUNT = 3
     # Преобразование ответов в формат one_hot_encoding
     y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
     y_test = utils.to_categorical(y_test_org, CLASS_COUNT)
```

```
[ ]  print(y_train.shape)
     print(y_train[0])
```

```
(241, 3)
[1. 0. 0.]
```

Рисунок 1.4 – Разделение и преобразование данных

```
[ ]  model1 = Sequential()
     model2 = Sequential()
```

```
[ ]  model1.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
     model1.add(Dense(15, activation='linear'))
     model1.add(Dense(10, activation='linear'))
     model1.add(Dense(3, activation='softmax')) # последний слой
```

```
[ ]  model2.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
     model2.add(Dense(15, activation='relu'))
     model2.add(Dense(10, activation='relu'))
     model2.add(Dense(3, activation='softmax')) # последний слой, три варианта
```

```
     model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
     model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
     # Вывод структуры модели
     print(model1.summary())
     print(model2.summary())
```

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_122 (Dense)           (None, 32)                12832

 dense_123 (Dense)           (None, 15)                495

 dense_124 (Dense)           (None, 10)                160

 dense_125 (Dense)           (None, 3)                 33

=================================================================
Total params: 13520 (52.81 KB)
Trainable params: 13520 (52.81 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Model: "sequential_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_126 (Dense)           (None, 32)                12832

 dense_127 (Dense)           (None, 15)                495

 dense_128 (Dense)           (None, 10)                160

 dense_129 (Dense)           (None, 3)                 33

=================================================================
Total params: 13520 (52.81 KB)
Trainable params: 13520 (52.81 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Рисунок 1.5 – Создание моделей с функциями активации «relu» и «linear»

```
[ ]  model1.fit(x_train,        # обучающая выборка, входные данные
                y_train,        # обучающая выборка, выходные данные
                batch_size=10, # кол-во примеров, которое обрабатывает нейронка перед одним изменением весов
                epochs=15,      # количество эпох, когда нейронка обучается на всех примерах выборки
                verbose=1)
```

```
Epoch 1/15
25/25 [==============================] - 1s 4ms/step - loss: 1.1016 - accuracy: 0.4647
Epoch 2/15
25/25 [==============================] - 0s 3ms/step - loss: 0.8159 - accuracy: 0.6805
Epoch 3/15
25/25 [==============================] - 0s 3ms/step - loss: 0.7598 - accuracy: 0.6805
Epoch 4/15
25/25 [==============================] - 0s 3ms/step - loss: 0.6416 - accuracy: 0.7178
Epoch 5/15
25/25 [==============================] - 0s 4ms/step - loss: 0.5777 - accuracy: 0.7552
Epoch 6/15
25/25 [==============================] - 0s 4ms/step - loss: 0.4808 - accuracy: 0.8174
Epoch 7/15
25/25 [==============================] - 0s 3ms/step - loss: 0.4291 - accuracy: 0.8548
Epoch 8/15
25/25 [==============================] - 0s 3ms/step - loss: 0.3933 - accuracy: 0.8714
Epoch 9/15
25/25 [==============================] - 0s 3ms/step - loss: 0.3590 - accuracy: 0.8714
Epoch 10/15
25/25 [==============================] - 0s 4ms/step - loss: 0.2750 - accuracy: 0.9004
Epoch 11/15
25/25 [==============================] - 0s 3ms/step - loss: 0.3164 - accuracy: 0.8963
Epoch 12/15
25/25 [==============================] - 0s 3ms/step - loss: 0.2215 - accuracy: 0.9212
Epoch 13/15
25/25 [==============================] - 0s 3ms/step - loss: 0.1884 - accuracy: 0.9378
Epoch 14/15
25/25 [==============================] - 0s 4ms/step - loss: 0.1604 - accuracy: 0.9502
Epoch 15/15
25/25 [==============================] - 0s 4ms/step - loss: 0.2006 - accuracy: 0.9336
<keras.src.callbacks.History at 0x7c1e22ba1180>
```

Рисунок 1.6 – Обучение модели

```
model2.fit(x_train,        # обучающая выборка, входные данные
           y_train,        # обучающая выборка, выходные данные
           batch_size=10, # кол-во примеров, которое обрабатывает нейронка перед одним изменением весов
           epochs=15,      # количество эпох, когда нейронка обучается на всех примерах выборки
           verbose=1)
```

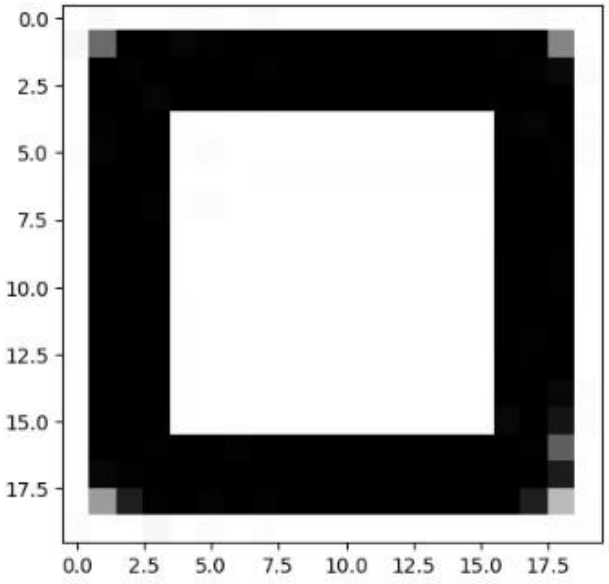
```
Epoch 1/15
25/25 [==============================] - 1s 4ms/step - loss: 1.0735 - accuracy: 0.3983
Epoch 2/15
25/25 [==============================] - 0s 3ms/step - loss: 0.9682 - accuracy: 0.4730
Epoch 3/15
25/25 [==============================] - 0s 3ms/step - loss: 0.8487 - accuracy: 0.6598
Epoch 4/15
25/25 [==============================] - 0s 3ms/step - loss: 0.7581 - accuracy: 0.7220
Epoch 5/15
25/25 [==============================] - 0s 4ms/step - loss: 0.6823 - accuracy: 0.7427
Epoch 6/15
25/25 [==============================] - 0s 3ms/step - loss: 0.6558 - accuracy: 0.7427
Epoch 7/15
25/25 [==============================] - 0s 3ms/step - loss: 0.5639 - accuracy: 0.8008
Epoch 8/15
25/25 [==============================] - 0s 3ms/step - loss: 0.5691 - accuracy: 0.7925
Epoch 9/15
25/25 [==============================] - 0s 4ms/step - loss: 0.5065 - accuracy: 0.8050
Epoch 10/15
25/25 [==============================] - 0s 3ms/step - loss: 0.5001 - accuracy: 0.8050
Epoch 11/15
25/25 [==============================] - 0s 3ms/step - loss: 0.4335 - accuracy: 0.8382
Epoch 12/15
25/25 [==============================] - 0s 4ms/step - loss: 0.4625 - accuracy: 0.8008
Epoch 13/15
25/25 [==============================] - 0s 3ms/step - loss: 0.3985 - accuracy: 0.8589
Epoch 14/15
25/25 [==============================] - 0s 4ms/step - loss: 0.3923 - accuracy: 0.8340
Epoch 15/15
25/25 [==============================] - 0s 4ms/step - loss: 0.3664 - accuracy: 0.8672
<keras.src.callbacks.History at 0x7c1e22a96e60>
```

Рисунок 1.7 – Обучение модели

```
[ ]  model1.save_weights('model1.h5')
     model1.load_weights('model1.h5')
     model2.save_weights('model2.h5')
     model2.load_weights('model2.h5')
```

Рисунок 1.8 – Запоминание и загрузка весов

```
[ ]  # Номер тестовой фигуры, которую будем распознавать
     n_rec = np.random.randint(x_test_org.shape[0])
     plt.imshow(x_test_org[n_rec], cmap='gray')
     plt.show()
```




```
[ ]  # Выбор нужной картинки из тестовой выборки
     x = x_test[n_rec]

     # Проверка формы данных
     print(x.shape)
```

```
(400,)
```

```
[ ]  # Добавление одной оси в начале, чтобы нейронка могла распознать пример
     # Массив из одного примера, так как нейронка принимает именно массивы примеров (батчи) для распознавания
     x = np.expand_dims(x, axis=0)

     # Проверка формы данных
     print(x.shape)
```

```
(1, 400)
```

Рисунок 1.9 – Выбор тестовой фигуры

```
prediction1 = model1.predict(x)
```

```
WARNING:tensorflow:5 out of the last 7 calls to <function Mod
1/1 [==============================] - 0s 53ms/step
```

```
prediction2 = model2.predict(x)
```

```
WARNING:tensorflow:6 out of the last 8 calls to <function Mod
1/1 [==============================] - 0s 54ms/step
```

```
# Вывод результата
print(prediction1)
print(prediction2)
```

```
[[3.7706855e-07 2.2622001e-04 9.9977344e-01]]
[[0.00524786 0.01102614 0.98372597]]
```

```
# Получение и вывод индекса самого большого элемента
pred = np.argmax(prediction2)
print(f'Распознана цифра: {pred}')
```

```
Распознана цифра: 2
```

```
print(y_test_org[n_rec])
```

```
2
```

Рисунок 1.10 – Распознавание фигуры и вывод результата

```
loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.6328 - accuracy: 0.7377
Model 1 - Test Accuracy: 0.7377049326896667
2/2 [==============================] - 0s 7ms/step - loss: 0.7258 - accuracy: 0.7541
Model 2 - Test Accuracy: 0.7540983557701111
```

```
accuracy_lin['n10_batch10'] = accuracy1
accuracy_re['n10_batch10'] = accuracy2
```

Рисунок 1.11 – Определение и запоминание точности модели

```
[ ]  model1 = Sequential()
     model2 = Sequential()

     model1.add(Dense(100, input_dim=400, activation='relu')) # Размер изображения 20 на 20
     model1.add(Dense(15, activation='linear'))
     model1.add(Dense(5, activation='linear'))
     model1.add(Dense(3, activation='softmax')) # последний слой

     model2.add(Dense(100, input_dim=400, activation='relu')) # Размер изображения 20 на 20
     model2.add(Dense(15, activation='relu'))
     model2.add(Dense(10, activation='relu'))
     model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

     model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
     model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

     model1.fit(x_train, y_train, batch_size=10, epochs=15, verbose=0)
     model2.fit(x_train, y_train, batch_size=10, epochs=15, verbose=0)

     <keras.src.callbacks.History at 0x7c1e1a2c4d30>
```

```
[ ]  model1.save_weights('model1.h5')
     model1.load_weights('model1.h5')
     model2.save_weights('model2.h5')
     model2.load_weights('model2.h5')
```

```
[ ]  loss1, accuracy1 = model1.evaluate(x_test, y_test)
     print("Model 1 - Test Accuracy:", accuracy1)

     loss2, accuracy2 = model2.evaluate(x_test, y_test)
     print("Model 2 - Test Accuracy:", accuracy2)

     2/2 [==============================] - 0s 7ms/step - loss: 0.5143 - accuracy: 0.8525
     Model 1 - Test Accuracy: 0.8524590134620667
     2/2 [==============================] - 0s 7ms/step - loss: 0.6563 - accuracy: 0.8033
     Model 2 - Test Accuracy: 0.8032786846160889
```

```
[ ]  accuracy_lin['n100_batch10'] = accuracy1
     accuracy_re['n100_batch10'] = accuracy2
```

Рисунок 1.12 – Расчет точности модели при изменении количества нейронов на 100

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(5000, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(5, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(5000, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(5, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=10, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=10, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')
```

```
[ ] loss1, accuracy1 = model1.evaluate(x_test, y_test)
    print("Model 1 - Test Accuracy:", accuracy1)

    loss2, accuracy2 = model2.evaluate(x_test, y_test)
    print("Model 2 - Test Accuracy:", accuracy2)

    accuracy_lin['n5000_batch10'] = accuracy1
    accuracy_re['n5000_batch10'] = accuracy2
```

```
2/2 [==============================] - 0s 7ms/step - loss: 1.4577 - accuracy: 0.6885
Model 1 - Test Accuracy: 0.688524603843689
2/2 [==============================] - 0s 7ms/step - loss: 0.9436 - accuracy: 0.5738
Model 2 - Test Accuracy: 0.5737704634666443
```

Рисунок 1.13 – Расчет точности модели при изменении количества нейронов на 5000

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(10, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(10, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=100, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=100, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')

loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)

accuracy_lin['n10_batch100'] = accuracy1
accuracy_re['n10_batch100'] = accuracy2
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.6635 - accuracy: 0.7377
Model 1 - Test Accuracy: 0.7377049326896667
2/2 [==============================] - 0s 9ms/step - loss: 0.6493 - accuracy: 0.7541
Model 2 - Test Accuracy: 0.7540983557701111
```

Рисунок 1.14 – Расчет точности модели при изменении количества нейронов на 10 и размера batch_size на 100

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(100, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(5, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(100, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(5, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=100, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=100, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')

loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)

accuracy_lin['n100_batch100'] = accuracy1
accuracy_re['n100_batch100'] = accuracy2
```

```
2/2 [==============================] - 0s 7ms/step - loss: 0.5908 - accuracy: 0.7541
Model 1 - Test Accuracy: 0.7540983557701111
2/2 [==============================] - 0s 7ms/step - loss: 0.8996 - accuracy: 0.6557
Model 2 - Test Accuracy: 0.6557376980781555
```

Рисунок 1.15 – Расчет точности модели при изменении количества нейронов на 100 и размера batch_size на 100

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(5000, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(5, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(5000, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(5, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=100, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=100, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')

loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)

accuracy_lin['n5000_batch100'] = accuracy1
accuracy_re['n5000_batch100'] = accuracy2
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.6520 - accuracy: 0.7377
Model 1 - Test Accuracy: 0.7377049326896667
2/2 [==============================] - 0s 8ms/step - loss: 0.9143 - accuracy: 0.5246
Model 2 - Test Accuracy: 0.5245901346206665
```

Рисунок 1.16 – Расчет точности модели при изменении количества нейронов на 5000 и размера batch_size на 100

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(5000, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(5, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(5000, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(5, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=1000, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=1000, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')

loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)

accuracy_lin['n5000_batch1000'] = accuracy1
accuracy_re['n5000_batch1000'] = accuracy2
```

```
2/2 [==============================] - 0s 8ms/step - loss: 1.0089 - accuracy: 0.5574
Model 1 - Test Accuracy: 0.5573770403862
2/2 [==============================] - 0s 8ms/step - loss: 1.0332 - accuracy: 0.5410
Model 2 - Test Accuracy: 0.5409836173057556
```

Рисунок 1.17 – Расчет точности модели при изменении количества нейронов на 5000 и размера batch_size на 1000

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(100, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(5, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(100, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(5, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=1000, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=1000, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')

loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)

accuracy_lin['n100_batch1000'] = accuracy1
accuracy_re['n100_batch1000'] = accuracy2
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.9382 - accuracy: 0.6066
Model 1 - Test Accuracy: 0.6065573692321777
2/2 [==============================] - 0s 7ms/step - loss: 0.9219 - accuracy: 0.5082
Model 2 - Test Accuracy: 0.5081967115402222
```

Рисунок 1.18 – Расчет точности модели при изменении количества нейронов на 100 и размера batch_size на 1000

```
model1 = Sequential()
model2 = Sequential()

model1.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model1.add(Dense(15, activation='linear'))
model1.add(Dense(10, activation='linear'))
model1.add(Dense(3, activation='softmax')) # последний слой

model2.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
model2.add(Dense(15, activation='relu'))
model2.add(Dense(10, activation='relu'))
model2.add(Dense(3, activation='softmax')) # последний слой, три варианта

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1.fit(x_train, y_train, batch_size=1000, epochs=15, verbose=0)
model2.fit(x_train, y_train, batch_size=1000, epochs=15, verbose=0)

model1.save_weights('model1.h5')
model1.load_weights('model1.h5')
model2.save_weights('model2.h5')
model2.load_weights('model2.h5')

loss1, accuracy1 = model1.evaluate(x_test, y_test)
print("Model 1 - Test Accuracy:", accuracy1)

loss2, accuracy2 = model2.evaluate(x_test, y_test)
print("Model 2 - Test Accuracy:", accuracy2)

accuracy_lin['n10_batch1000'] = accuracy1
accuracy_re['n10_batch1000'] = accuracy2
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.8159 - accuracy: 0.6721
Model 1 - Test Accuracy: 0.6721311211585999
2/2 [==============================] - 0s 8ms/step - loss: 0.9104 - accuracy: 0.4918
Model 2 - Test Accuracy: 0.49180328845977783
```

Рисунок 1.19 – Расчет точности модели при изменении количества нейронов на 10 и размера batch_size на 1000

```
import matplotlib.pyplot as plt

# Получение ключей и значений из словаря accuracy_lin
labels_lin = list(accuracy_lin.keys())
values_lin = list(accuracy_lin.values())

# Получение ключей и значений из словаря accuracy_re
labels_re = list(accuracy_re.keys())
values_re = list(accuracy_re.values())

# Создание графика
plt.figure(figsize=(10, 5))

# Построение графика для accuracy_lin
plt.plot(labels_lin, values_lin, label='Linear Activation')

# Построение графика для accuracy_re
plt.plot(labels_re, values_re, label='ReLU Activation')

# Добавление подписей осей и заголовка
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy of Different Models')
plt.legend()

# Поворот подписей на 90 градусов
plt.xticks(rotation=90)

# Отображение графика
plt.show()
```

Рисунок 1.20 – Создание графика, для отображения результатов

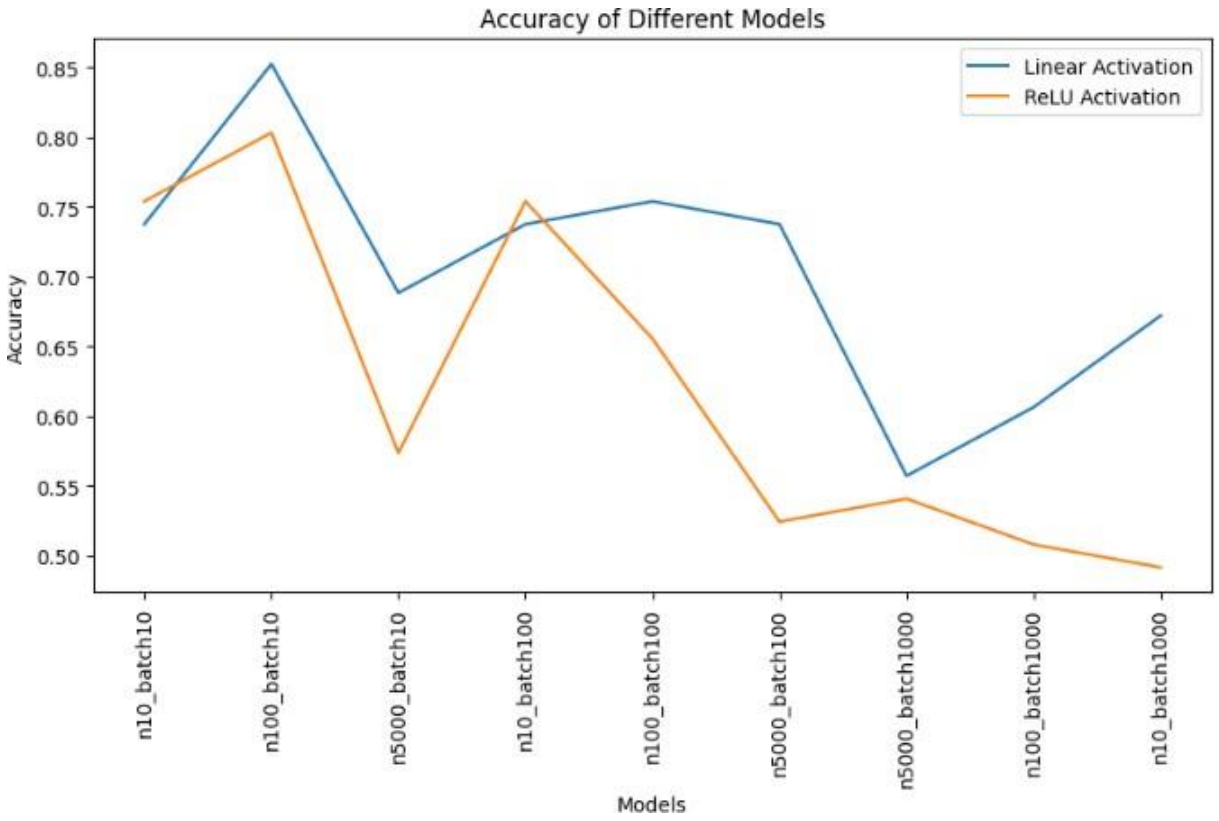


Рисунок 1.21 – Результаты сравнения

Задача 2. Самостоятельно напишите нейронную сеть, которая может стать составной частью системы бота для игры в «Крестики-нолики». Используя подготовленную базу изображений, создайте и обучите нейронную сеть, распознающую две категории изображений: крестики и нолики. Добейтесь точности распознавания более 95% (accuracy).

Решение.

```
[1]  # Подключение класса для создания нейронной сети прямого распространения
     from tensorflow.keras.models import Sequential
     # Подключение класса для создания полносвязного слоя
     from tensorflow.keras.layers import Dense
     # Подключение оптимизатора
     from tensorflow.keras.optimizers import Adam
     # Подключение утилит для to_categorical
     from tensorflow.keras import utils
     # Подключение библиотеки для загрузки изображений
     from tensorflow.keras.preprocessing import image
     # Подключение библиотеки для работы с массивами
     import numpy as np
     # Подключение модуля для работы с файлами
     import os
     from sklearn.model_selection import train_test_split
     # Подключение библиотек для отрисовки изображений
     import matplotlib.pyplot as plt
     from PIL import Image
     # Вывод изображения в ноутбуке, а не в консоли или файле
     %matplotlib inline
```

Рисунок 2.1 – Подключение библиотек

```
[2] # Загрузка датасета из облака
    import gdown
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l3/hw_pro.zip', None, quiet=True)
```

```
'hw_pro.zip'
```

```
[3] # Распаковываем архив hw_light.zip в папку hw_light
    !unzip -q hw_pro.zip
```

```
[4] # Путь к директории с базой
    base_dir = '/content/hw_pro'
    # Создание пустого списка для загрузки изображений обучающей выборки
    x_train = []
    # Создание списка для меток классов
    y_train = []
    # Задание высоты и ширины загружаемых изображений
    img_height = 20
    img_width = 20
    # Перебор папок в директории базы
    for patch in os.listdir(base_dir):
        # Перебор файлов в папках
        for img in os.listdir(base_dir + '/' + patch):
            # Добавление в список изображений текущей картинки
            x_train.append(image.img_to_array(image.load_img(base_dir + '/' + patch + '/' + img,
                                                             target_size=(img_height, img_width),
                                                             color_mode='grayscale')))
            # Добавление в массив меток, соответствующих классам
            if patch == '0':
                y_train.append(0)
            else:
                y_train.append(1)
    # Преобразование в numpy-массив загруженных изображений и меток классов
    x_train_org = np.array(x_train)
    y_train_org = np.array(y_train)
    # Вывод размерностей
    print('Размер массива x_train', x_train_org.shape)
    print('Размер массива y_train', y_train_org.shape)
```

```
Размер массива x_train (102, 20, 20, 1)
Размер массива y_train (102,)
```

Рисунок 2.2 – Подключение архива

```
[5] # Номер картинки
    n = 56
    # Отрисовка картинки
    plt.imshow(x_train_org[n], cmap='gray')

    # Вывод n-й картинки
    plt.show()
```

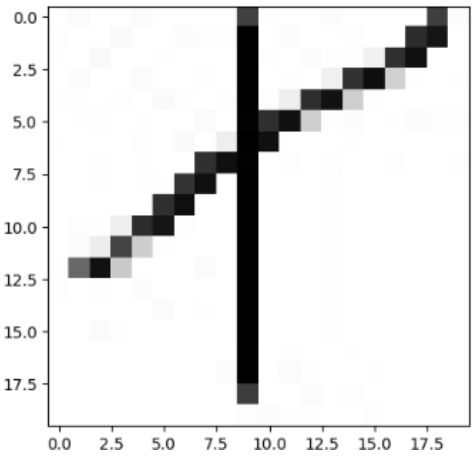


Рисунок 2.3 – Проверка содержания архива

```
[6] # Разделение данных на обучающий и тестовый наборы
    x_train_org, x_test_org, y_train_org, y_test_org = train_test_split(x_train_org, y_train_org, test_size=0.2, random_state=42)
```

```
[7] x_train = x_train_org.reshape(x_train_org.shape[0], -1)
    x_test = x_test_org.reshape(x_test_org.shape[0], -1)

    # Проверка результата
    print(f'Форма обучающих данных: {x_train.shape} -> {x_train_org.shape}')
    print(f'Форма  тестовых данных: {x_test.shape} ')
```

```
Форма обучающих данных: (81, 400) -> (81, 20, 20, 1)
Форма  тестовых данных: (21, 400)
```

```
[8] x_train = x_train.astype('float32') / 255.

    # Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
    x_test = x_test.astype('float32') / 255.
```

```
[9] CLASS_COUNT = 2
    # Преобразование ответов в формат one_hot_encoding
    y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
    y_test = utils.to_categorical(y_test_org, CLASS_COUNT)
```

```
[10] print(y_train.shape)
     print(y_train[0])
```

```
(81, 2)
[0. 1.]
```

Рисунок 2.4 – Распределение и преобразование данных

```
[50] model = Sequential()
```

```
[51] model.add(Dense(32, input_dim=400, activation='relu')) # Размер изображения 20 на 20
     model.add(Dense(15, activation='linear'))
     model.add(Dense(10, activation='linear'))
     model.add(Dense(2, activation='softmax')) # последний слой
```

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

     model.fit(x_train, y_train, batch_size=10, epochs=15, verbose=1)
```

```
Epoch 1/15
9/9 [==============================] - 1s 5ms/step - loss: 1.0467 - accuracy: 0.4938
Epoch 2/15
9/9 [==============================] - 0s 4ms/step - loss: 0.6919 - accuracy: 0.6049
Epoch 3/15
9/9 [==============================] - 0s 4ms/step - loss: 0.5959 - accuracy: 0.6790
Epoch 4/15
9/9 [==============================] - 0s 4ms/step - loss: 0.5052 - accuracy: 0.7778
Epoch 5/15
9/9 [==============================] - 0s 5ms/step - loss: 0.4623 - accuracy: 0.7901
Epoch 6/15
9/9 [==============================] - 0s 4ms/step - loss: 0.3913 - accuracy: 0.9136
Epoch 7/15
9/9 [==============================] - 0s 5ms/step - loss: 0.3333 - accuracy: 0.9259
Epoch 8/15
9/9 [==============================] - 0s 4ms/step - loss: 0.2996 - accuracy: 0.9506
Epoch 9/15
9/9 [==============================] - 0s 4ms/step - loss: 0.2431 - accuracy: 0.9506
Epoch 10/15
9/9 [==============================] - 0s 4ms/step - loss: 0.2067 - accuracy: 0.9630
Epoch 11/15
9/9 [==============================] - 0s 4ms/step - loss: 0.1720 - accuracy: 0.9383
Epoch 12/15
9/9 [==============================] - 0s 4ms/step - loss: 0.1341 - accuracy: 0.9630
Epoch 13/15
9/9 [==============================] - 0s 4ms/step - loss: 0.1596 - accuracy: 0.9630
Epoch 14/15
9/9 [==============================] - 0s 4ms/step - loss: 0.0995 - accuracy: 0.9753
Epoch 15/15
9/9 [==============================] - 0s 4ms/step - loss: 0.0857 - accuracy: 0.9753
<keras.src.callbacks.History at 0x783b90274700>
```

Рисунок 2.5 – Создание и обучение модели

```
model.save_weights('model.h5')
model.load_weights('model.h5')
```

```
[65] loss, accuracy = model.evaluate(x_test, y_test)
     print("Model - Test Accuracy:", accuracy)

     1/1 [==============================] - 0s 205ms/step - loss: 0.1782 - accuracy: 0.9524
     Model - Test Accuracy: 0.9523809552192688
```

Рисунок 2.7 – Запоминание весов и подсчет точности модели

Задача 3. Распознайте рукописную цифру, написанную на листе от руки. Последовательность шагов следующая:

- На бумаге рисуем произвольную цифру (желательно нарисовать цифру размером не более 5 * 5 мм и без наклона. В занятии нейронка обучалась на цифрах американских студентов. Эти цифры были написаны на тетрадных листах в клетку и имели схожий размер).
- Фотографируем. Загружаем фото в Collaboratory.
- С помощью функции image.load_img(path, target_size=(28, 28), color_mode = ‘grayscale’) загружаем картинку в переменную.
- С помощью функции image.img_to_array(img) преобразуем изображение в numpy-массив.
- Выполняем инверсию цветов, нормирование и решейп массива.
- Выполняем распознавание собственной рукописной цифры.

Примечание: точность распознавания рукописных цифр может быть достаточно низкой, т.к. рукописные цифры после преобразований хоть и похожи на содержащиеся в базе, но могут отличаться по конфигурации, толщине линий и т.д.

Решение.

```
[ ] from tensorflow.keras.datasets import mnist     # Библиотека с базой рукописных цифр
    from tensorflow.keras.models import Sequential  # Подключение класса создания модели Sequential
    from tensorflow.keras.layers import Dense       # Подключение класса Dense - полносвязный слой
    from tensorflow.keras import utils              # Утилиты для подготовки данных
    import numpy as np                              # Работа с массивами
    import matplotlib.pyplot as plt                 # Отрисовка изображений
    import image

    # Отрисовка изображений в ячейках ноутбука
    %matplotlib inline
```

Рисунок 3.1 – Подключение библиотек

```
# Загрузка из облака данных Mnist
(x_train_org, y_train_org), (x_test_org, y_test_org) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

```
# Вывод формы данных для обучения
x_train_org.shape
```

```
(60000, 28, 28)
```

Рисунок 3.2 – Загрузка данных для обучения

```
# Изменение формы входных картинок с 28x28 на 784
# первая ось остается без изменения, остальные складываются в вектор
x_train = x_train_org.reshape(x_train_org.shape[0], -1)
x_test = x_test_org.reshape(x_test_org.shape[0], -1)

# Проверка результата
print(f'Форма обучающих данных: {x_train_org.shape} -> {x_train.shape}')
print(f'Форма  тестовых данных: {x_test_org.shape} -> {x_test.shape}')
```

```
Форма обучающих данных: (60000, 28, 28) -> (60000, 784)
Форма  тестовых данных: (10000, 28, 28) -> (10000, 784)
```

```
[ ] # Нормализация входных картинок
    # Преобразование x_train в тип float32 (числа с плавающей точкой) и нормализация
    x_train = x_train.astype('float32') / 255.

    # Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
    x_test = x_test.astype('float32') / 255.
```

```
[ ] # Задание константы количества распознаваемых классов

    CLASS_COUNT = 10
```

```
[ ] # Преобразование ответов в формат one_hot_encoding
    y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
    y_test = utils.to_categorical(y_test_org, CLASS_COUNT)
```

```
[ ] # Вывод формы y_train
    # 60 тысяч примеров, каждый длины 10 по числу классов
    print(y_train.shape)
```

```
    (60000, 10)
```

```
[ ] # Вывод формы массива меток
    print(y_train_org.shape)
```

```
    (60000,)
```

Рисунок 3.3 – Преобразование данных

```
[ ] # Создание последовательной модели
    model = Sequential()

    # Добавление полносвязного слоя на 800 нейронов с relu-активацией
    model.add(Dense(800, input_dim=784, activation='relu'))

    # Добавление полносвязного слоя на 400 нейронов с relu-активацией
    model.add(Dense(400, activation='relu'))

    # Добавление полносвязного слоя с количеством нейронов по числу классов с softmax-активацией
    model.add(Dense(CLASS_COUNT, activation='softmax'))
```

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

    # Вывод структуры модели
    print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 800)               628000

 dense_1 (Dense)             (None, 400)               320400

 dense_2 (Dense)             (None, 10)                4010

=================================================================
Total params: 952410 (3.63 MB)
Trainable params: 952410 (3.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Рисунок 3.4 – Создание модели

```
[ ]  model.fit(x_train,        # обучающая выборка, входные данные
               y_train,        # обучающая выборка, выходные данные
               batch_size=128, # кол-во примеров, которое обрабатывает нейронка перед одним изменением весов
               epochs=15,      # количество эпох, когда нейронка обучается на всех примерах выборки
               verbose=1)      # 0 - не визуализировать ход обучения, 1 - визуализировать
```

```
Epoch 1/15
469/469 [==============================] - 5s 4ms/step - loss: 0.2067 - accuracy: 0.9380
Epoch 2/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0766 - accuracy: 0.9769
Epoch 3/15
469/469 [==============================] - 2s 4ms/step - loss: 0.0486 - accuracy: 0.9844
Epoch 4/15
469/469 [==============================] - 2s 4ms/step - loss: 0.0340 - accuracy: 0.9886
Epoch 5/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0266 - accuracy: 0.9918
Epoch 6/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0211 - accuracy: 0.9931
Epoch 7/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0198 - accuracy: 0.9930
Epoch 8/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0140 - accuracy: 0.9954
Epoch 9/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0155 - accuracy: 0.9948
Epoch 10/15
469/469 [==============================] - 2s 4ms/step - loss: 0.0127 - accuracy: 0.9958
Epoch 11/15
469/469 [==============================] - 2s 5ms/step - loss: 0.0119 - accuracy: 0.9961
Epoch 12/15
469/469 [==============================] - 2s 4ms/step - loss: 0.0100 - accuracy: 0.9967
Epoch 13/15
469/469 [==============================] - 2s 3ms/step - loss: 0.0098 - accuracy: 0.9967
Epoch 14/15
469/469 [==============================] - 2s 4ms/step - loss: 0.0102 - accuracy: 0.9968
Epoch 15/15
469/469 [==============================] - 2s 4ms/step - loss: 0.0109 - accuracy: 0.9964
<keras.src.callbacks.History at 0x7c6cd8075720>
```

```
[ ]  model.save_weights('model.h5')
     model.load_weights('model.h5')
```

Рисунок 3.5 – Обучение модели и запоминание весов

```
# Номер тестовой цифры, которую будем распознавать
n_rec = np.random.randint(x_test_org.shape[0])

# Отображение картинки из тестового набора под номером n_rec
plt.imshow(x_test_org[n_rec], cmap='gray')
plt.show()
```

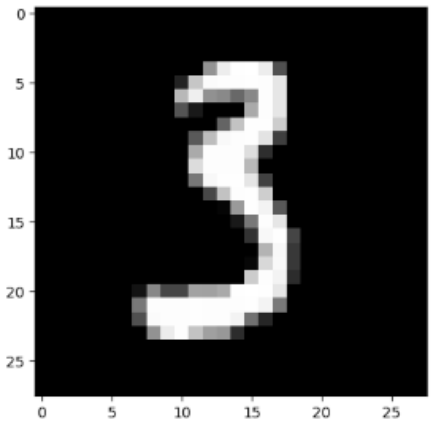


Рисунок 3.6 – Выбор тестовой цифры

```
# Выбор нужной картинки из тестовой выборки
x = x_test[n_rec]

# Проверка формы данных
print(x.shape)
```

```
(784,)
```

```
# Добавление одной оси в начале, чтобы нейронка могла распознать пример
# Массив из одного примера, так как нейронка принимает именно массивы примеров (батчи) для распознавания
x = np.expand_dims(x, axis=0)

# Проверка формы данных
print(x.shape)
```

```
(1, 784)
```

```
# Распознавание примера
prediction = model.predict(x)
```

```
1/1 [==============================] - 0s 108ms/step
```

```
# Вывод результата - вектор из 10 чисел
print(prediction)
```

```
[[2.9577045e-19 1.3576207e-13 2.8305990e-13 1.0000000e+00 8.5358716e-19
  1.8641625e-12 3.0995246e-22 1.3958715e-15 4.1087307e-12 1.1482763e-11]]
```

```
# Получение и вывод индекса самого большого элемента (это значение цифры, которую распознала сеть)
pred = np.argmax(prediction)
print(f'Распознана цифра: {pred}')
```

```
Распознана цифра: 3
```

Рисунок 3.7 – Распознавание цифры

```
import requests

# URL изображения
url = "https://drive.google.com/file/d/1l3SyEr2IzoIrUblsiO7J8p-Oc_QqjTGb/uc?export=download

# Скачать изображение
response = requests.get(url)

# Сохранить изображение в локальном файле
with open('image.jpg', 'wb') as f:
    f.write(response.content)
```

```
import requests
from PIL import Image
from io import BytesIO
from tensorflow.keras.preprocessing import image
```

```
# URL изображения
url = "https://drive.google.com/uc?id=1l3SyEr2IzoIrUblsiO7J8p-Oc_QqjTGb"

# Скачать изображение
response = requests.get(url)

# Открываем изображение из байтового потока
img = Image.open(BytesIO(response.content))

# Сохраняем изображение локально
img_path = '/content/image.jpg'
img.save(img_path)

# Загрузка изображения с помощью Keras
loaded_img = image.load_img(img_path, target_size=(28, 28), color_mode='grayscale')
```

Рисунок 3.8 – Загрузка изображения

```
plt.imshow(loaded_img, cmap='gray')

# Вывод n-й картинки
plt.show()
```

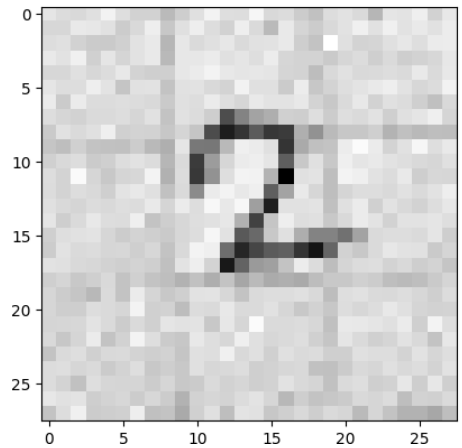


Рисунок 3.9 – Отображение изображения

```
[ ]  loaded_img_array = image.img_to_array(loaded_img)

     print("Размер массива numpy:", loaded_img_array.shape)

     Размер массива numpy: (28, 28, 1)

[ ]  # Инвертируем цвета
     inverted_img_array = 255 - loaded_img_array

[ ]  plt.imshow(inverted_img_array, cmap='gray')

     # Вывод n-й картинки
     plt.show()
```

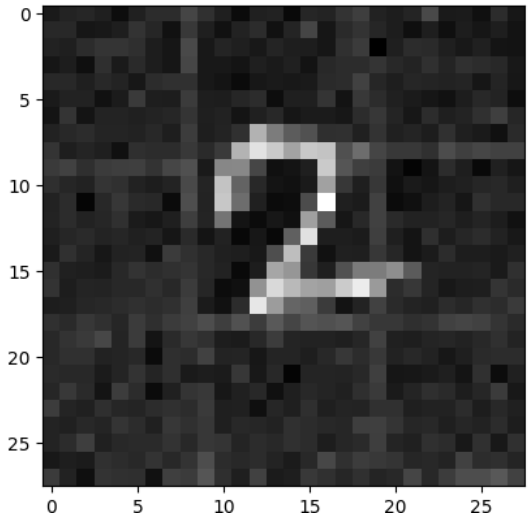


Рисунок 3.10 – Результат инвертирования цветов

```
[ ]  print("Размер массива numpy после преобразования:", inverted_img_array.shape)

     Размер массива numpy после преобразования: (28, 28, 1)

[ ]  # Изменяем форму массива на одномерный массив
     reshaped_img_array = inverted_img_array.reshape((1, 784))

     print("Размер массива numpy после преобразования:", reshaped_img_array.shape)

     Размер массива numpy после преобразования: (1, 784)

[ ]  normalized_img_array = reshaped_img_array.astype('float32') / 255.

[ ]  my_prediction = model.predict(normalized_img_array)

     1/1 [==============================] - 0s 18ms/step

[ ]  my_pred = np.argmax(my_prediction)
     print(f'Распознана цифра: {my_pred}')

     Распознана цифра: 2
```

Рисунок 3.11 – Нормализация данных и распознавание цифры

Выводы. В ходе выполнения лабораторной работы была протестирована модель с разными функциями активации в скрытых слоях, разным количеством нейронов и разным размером batch_size. При анализе результатов, можно сделать вывод, что наиболее точной оказалась модель с линейной функцией активации, сотней нейронов в одном из слоев и размером batch_size равным 10.

Используя подготовленную базу изображений, была создана и обучена нейронная сеть, распознающая две категории изображений: крестики и нолики, с точностью распознавания более 95%.

Была также выполнена работа по распознаванию написанной от руки цифры, с загрузкой и преобразованием изображения.