

ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Старший преподаватель

должность, уч. степень, звание

подпись, дата

В.В. Боженко

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

Свёрточные нейронные сети

по курсу: Машинное обучение

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4116

подпись, дата

Д. В. Коптев

инициалы, фамилия

Санкт-Петербург 2024

Цель работы

Обучить свёрточную сеть распознавать изображения

Ход работы

Были загружены необходимые для работы библиотеки (рисунок 1).

```
[12] import numpy
      from keras.utils import to_categorical
      import pandas as pd
      from keras.datasets import mnist
      from keras.models import Sequential
      from keras.layers import Dense
      from keras.layers import Dropout
      import matplotlib.pyplot as plt
      from sklearn.metrics import confusion_matrix
      import seaborn as sns
      import numpy as np
      from sklearn.metrics import classification_report
      from keras.layers import Dropout
      from keras.layers import Flatten
      from keras.layers import Conv2D
      from keras.layers import MaxPooling2D
      from keras import backend as K
      from PIL import Image
      from keras.datasets import cifar10
```

Рисунок 1 – Необходимые для работы библиотеки

Были загружены данные из набора MNIST. Было визуализировано 4 изображения из тренировочного набора и 4 изображения из валидационного набора (рисунок 2).

```
[5] for i in range(1, 5):
    plt.subplot(2, 2, i)
    plt.imshow(random.choice(X_train), cmap=plt.get_cmap('autumn'))
    plt.show()
```

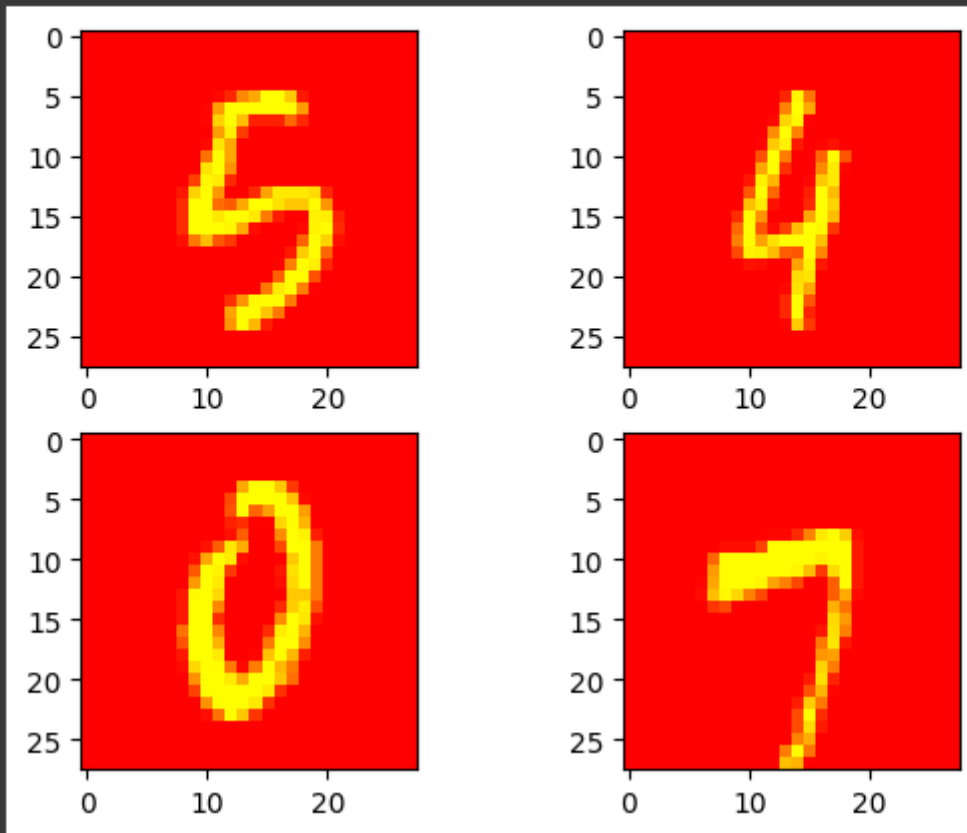


Рисунок 2 – Загрузка изображений

Данный набор данных содержит в себе изображения цифр и массив с цифрами как целевой признак. Учебный набор данных представлен в виде трехмерных массивов.

Чтобы провести предобработку данных необходимо представить этот массив в виде одномерного, т.к. каждый пиксель - отдельный входной признак (рисунок 3).

```
[55] num_pixels = X_train.shape[1] * X_train.shape[2]
      X_train = X_train.reshape(X_train.shape[0], num_pixels) # shape 0 для того чтобы получился массив размером (длина X_train, кол-во пикселей)
      X_test = X_test.reshape(X_test.shape[0], num_pixels)

[56] X_train.shape

(60000, 784)
```

Рисунок 3 – Предобработка данных

Значения пикселей заданы в диапазоне от 0 до 255. Для эффективного обучения нейронной сети необходимо масштабировать входные значения. Для этого нормализуем значения пикселей в диапазон от 0 до 1, разделив каждое значение на 255 (рисунок 4).

```
[57] X_train = X_train/255
     X_test = X_test/255
```

Рисунок 4 – Предобработка

Выходное значение - целое число от 0 до 9. Это задача классификации с несколькими классами. Необходимо преобразовать метки классов в формат, который нейронная сеть сможет "понять", с помощью `pr_utils.to_categorical()` можно создать двоичные матрицы (рисунок 5).

```
y_train = to_categorical(y_train) # подготовка к квалификации - вектор, в котором на месте определенного класса единица
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]
print(num_classes)
print(y_test.shape)
y_train.shape
```

```
10
(10000, 10)
(60000, 10)
```

Рисунок 5 – Преобразование меток классов

Необходимо создать полносвязную модель с двумя слоями. На входном слое будет использоваться линейная функция активации, на выходном - softmax. Функция потерь - `categorical_crossentropy` - функция потерь для многоклассовой классификации (рисунок 6). На рисунке 7 изображен процесс обучения.

```
model = Sequential()
model.add(Dense(64, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(10, activation = 'softmax'))

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

[19] print(X_train)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
```

Рисунок 6 – Создание полносвязной модели с двумя слоями

```
[20] history = model.fit(X_train,y_train, validation_data = (X_test, y_test),epochs = 20, batch_size = 200, verbose = 2)
history_df = pd.DataFrame(history.history)

Epoch 1/20
300/300 - 2s - loss: 0.4734 - accuracy: 0.8694 - val_loss: 0.2974 - val_accuracy: 0.9159 - 2s/epoch - 7ms/step
Epoch 2/20
300/300 - 1s - loss: 0.2966 - accuracy: 0.9161 - val_loss: 0.2756 - val_accuracy: 0.9242 - 795ms/epoch - 3ms/step
Epoch 3/20
300/300 - 1s - loss: 0.2885 - accuracy: 0.9287 - val_loss: 0.2676 - val_accuracy: 0.9250 - 892ms/epoch - 3ms/step
Epoch 4/20
300/300 - 1s - loss: 0.2724 - accuracy: 0.9238 - val_loss: 0.2700 - val_accuracy: 0.9229 - 791ms/epoch - 3ms/step
Epoch 5/20
300/300 - 1s - loss: 0.2663 - accuracy: 0.9255 - val_loss: 0.2789 - val_accuracy: 0.9235 - 816ms/epoch - 3ms/step
Epoch 6/20
300/300 - 1s - loss: 0.2619 - accuracy: 0.9271 - val_loss: 0.2755 - val_accuracy: 0.9238 - 756ms/epoch - 3ms/step
Epoch 7/20
300/300 - 1s - loss: 0.2588 - accuracy: 0.9285 - val_loss: 0.2724 - val_accuracy: 0.9228 - 815ms/epoch - 3ms/step
Epoch 8/20
300/300 - 1s - loss: 0.2575 - accuracy: 0.9284 - val_loss: 0.2693 - val_accuracy: 0.9260 - 820ms/epoch - 3ms/step
Epoch 9/20
300/300 - 1s - loss: 0.2538 - accuracy: 0.9299 - val_loss: 0.2685 - val_accuracy: 0.9268 - 1s/epoch - 4ms/step
Epoch 10/20
300/300 - 1s - loss: 0.2516 - accuracy: 0.9308 - val_loss: 0.2787 - val_accuracy: 0.9250 - 1s/epoch - 4ms/step
Epoch 11/20
300/300 - 1s - loss: 0.2501 - accuracy: 0.9309 - val_loss: 0.2755 - val_accuracy: 0.9241 - 815ms/epoch - 3ms/step
Epoch 12/20
300/300 - 1s - loss: 0.2491 - accuracy: 0.9309 - val_loss: 0.2735 - val_accuracy: 0.9262 - 796ms/epoch - 3ms/step
Epoch 13/20
300/300 - 1s - loss: 0.2481 - accuracy: 0.9313 - val_loss: 0.2687 - val_accuracy: 0.9271 - 862ms/epoch - 3ms/step
Epoch 14/20
300/300 - 1s - loss: 0.2467 - accuracy: 0.9325 - val_loss: 0.2718 - val_accuracy: 0.9246 - 873ms/epoch - 3ms/step
Epoch 15/20
300/300 - 1s - loss: 0.2464 - accuracy: 0.9321 - val_loss: 0.2760 - val_accuracy: 0.9267 - 757ms/epoch - 3ms/step
Epoch 16/20
300/300 - 1s - loss: 0.2450 - accuracy: 0.9324 - val_loss: 0.2713 - val_accuracy: 0.9268 - 776ms/epoch - 3ms/step
Epoch 17/20
300/300 - 1s - loss: 0.2440 - accuracy: 0.9320 - val_loss: 0.2689 - val_accuracy: 0.9279 - 874ms/epoch - 3ms/step
Epoch 18/20
300/300 - 1s - loss: 0.2428 - accuracy: 0.9324 - val_loss: 0.2786 - val_accuracy: 0.9255 - 790ms/epoch - 3ms/step
Epoch 19/20
300/300 - 1s - loss: 0.2441 - accuracy: 0.9324 - val_loss: 0.2786 - val_accuracy: 0.9244 - 765ms/epoch - 3ms/step
Epoch 20/20
300/300 - 1s - loss: 0.2420 - accuracy: 0.9323 - val_loss: 0.2742 - val_accuracy: 0.9256 - 821ms/epoch - 3ms/step
```

Рисунок 7 – Процесс обучения

Далее необходимо вывести `classification_report` И `confusion_matrix` (рисунок 8).

```

predictions = model.predict(X_test)
predictions = np.argmax(predictions,axis = 1)
y_test_argmax = np.argmax(y_test,axis = 1)
print("Classification Report: ")
print(classification_report(y_test_argmax,predictions))
# support - количество примеров каждого класса в наборе данных
# F1 score - гармоническое среднее точности и полноты
# recall - полнота - отношение правильно классифицированных положительных примеров к общему количеству фактических положительных примеров
# precision - отношение правильно классифицированных положительных примеров к общему количеству классифицированных как положительных

```

```

313/313 [=====] - 1s 2ms/step
Classification Report:

```

	precision	recall	f1-score	support
0	0.95	0.98	0.97	980
1	0.96	0.98	0.97	1135
2	0.94	0.90	0.92	1032
3	0.91	0.91	0.91	1010
4	0.93	0.93	0.93	982
5	0.90	0.87	0.88	892
6	0.93	0.95	0.94	958
7	0.93	0.93	0.93	1028
8	0.88	0.88	0.88	974
9	0.92	0.91	0.91	1009
accuracy			0.93	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.93	0.93	0.93	10000

```

[23] print("Confusion_matrix")
print(confusion_matrix(y_test_argmax, predictions))

```

```

Confusion_matrix
[[ 963  0  0  2  2  5  5  2  1  0]
 [  0 1114  3  2  0  1  4  2  9  0]
 [  7  8 924 15 10  5 14 11 35  3]
 [  4  1 18 918  0 24  5  9 24  7]
 [  1  3  4  2 915  0 13  5  8 31]
 [ 10  4  2 35  8 775 18 10 25  5]
 [ 10  3  4  2  7 16 913  1  2  0]
 [  1  4 23  5  5  2  0 958  3 27]
 [  9 12  4 19  8 28 14 13 856 11]
 [ 10  8  1  8 26  8  0 22  6 920]]

```

Рисунок 8 – Вывод classification_report и confusion_matrix

По classification_report и confusion matrix можно сделать выводы о том, что:

- Высокие значения precision говорят о том, что количество правильно классифицированных положительных примеров крайне высоко
- Высокие значения recall говорят о том, что модель классифицирует большую часть положительных наблюдений верно, т.е. модель успешно классифицирует большинство наблюдений для каждого класса
- Высокие значения F1-Score говорит о маленькой разнице между precision и recall
- Т.к. на главной диагонали confusion matrix располагаются высокие значения, можно сделать вывод о том, что модель верно классифицирует большинство значений

Необходимо построить график потерь и точности обучения по эпохам для данной модели. Для данной цели была создана функция plotting, которая выводит график зависимости потерь от эпохи и график зависимости точности от эпохи (рисунок 9)

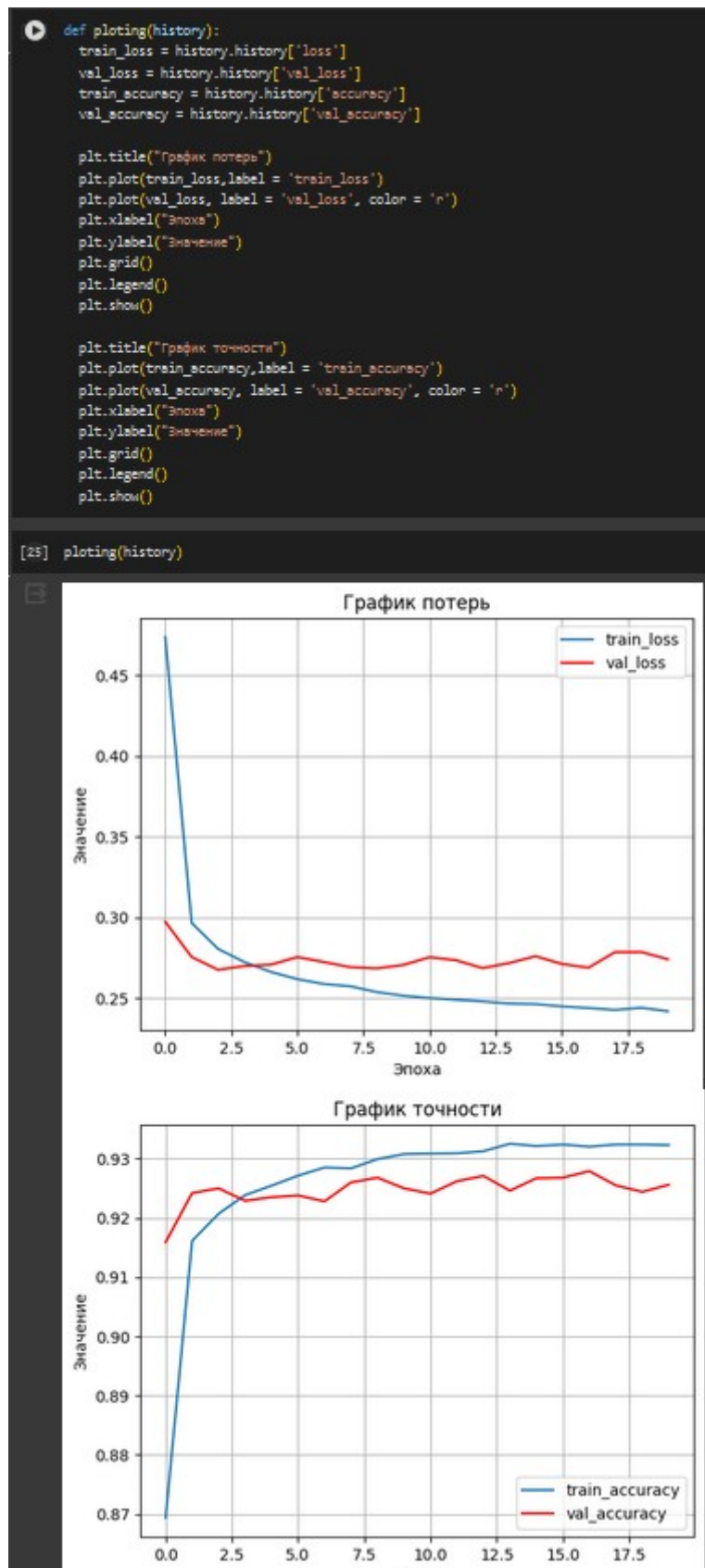


Рисунок 9 – Графики потерь и точности обучения по эпохам

По данным графикам можно сделать вывод о том, что значение потерь резко уменьшается с 1 по 3 эпохи, что говорит о том, что в этот момент модель обучается, затем

график сглаживается. Значение точности же резко возрастает с 1 по 3 эпохи, что также говорит о том, что в этот период модель обучается.

Нелинейная функция активации

Необходимо повторить данный эксперимент с другой архитектурой НС, а именно - изменить функцию активации на нелинейную. Отличие linear от sigmoid в том, что при использовании sigmoid значения получаются от 0 до 1, в случае с Linear от 0 до бесконечности (рисунок 10). На рисунке 11 изображена история обучения по эпохам

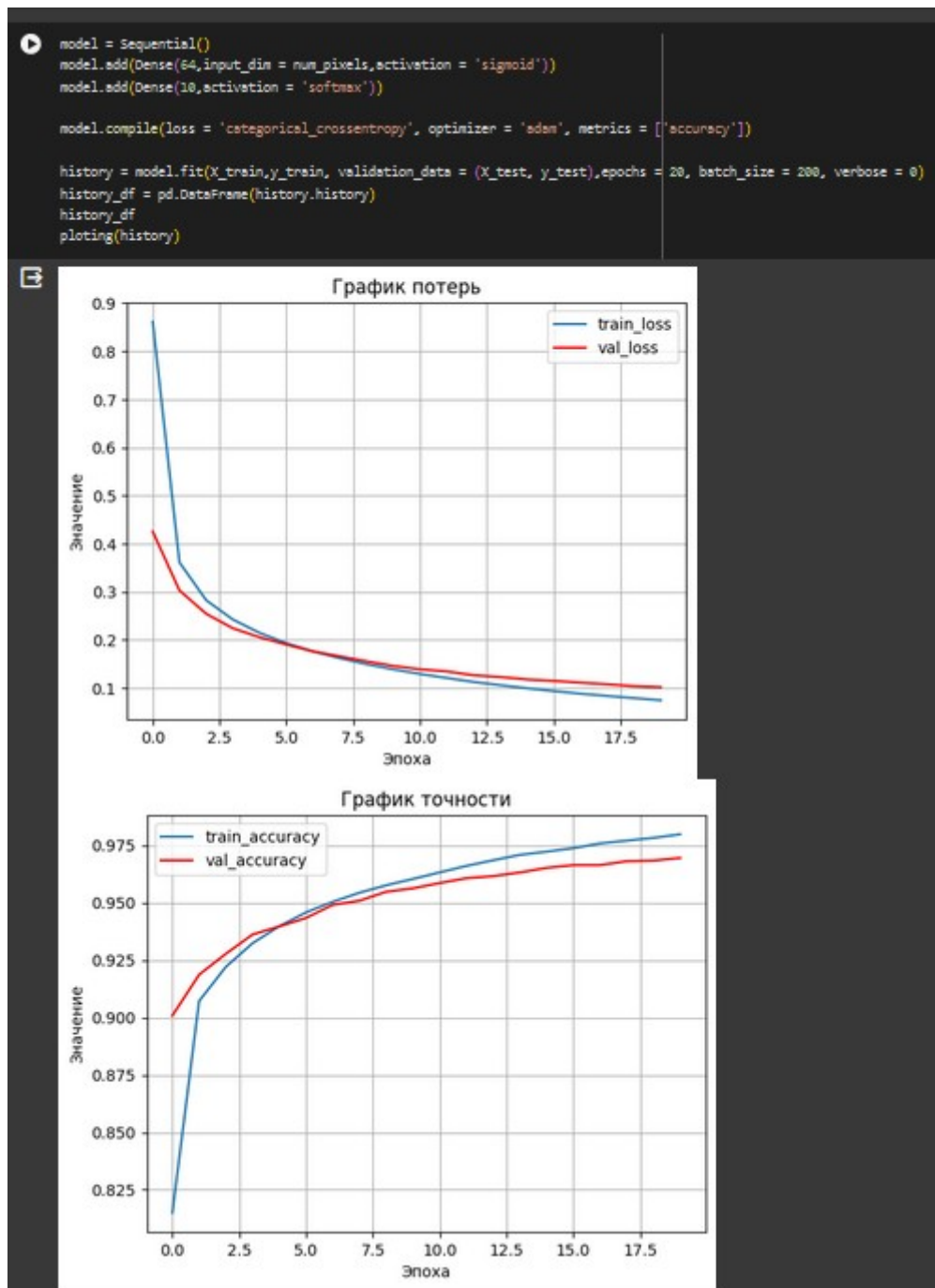


Рисунок 10 – График потерь и точности обучения по эпохам

history_df

	loss	accuracy	val_loss	val_accuracy
0	0.861302	0.814983	0.425401	0.9008
1	0.361272	0.907133	0.302838	0.9186
2	0.282510	0.922000	0.254318	0.9277
3	0.242359	0.932300	0.224160	0.9361
4	0.214539	0.939700	0.205555	0.9396
5	0.193402	0.945717	0.190335	0.9433
6	0.176038	0.950333	0.176126	0.9490
7	0.161903	0.954283	0.165883	0.9508
8	0.149508	0.957533	0.155489	0.9547
9	0.138788	0.960283	0.145909	0.9562
10	0.129287	0.963133	0.138841	0.9585
11	0.120904	0.965950	0.134570	0.9606
12	0.112723	0.968417	0.126588	0.9615
13	0.105991	0.970767	0.123012	0.9631
14	0.099578	0.972167	0.117875	0.9651
15	0.093745	0.973667	0.114901	0.9663
16	0.088357	0.975750	0.111227	0.9663
17	0.083670	0.976967	0.107834	0.9680
18	0.079089	0.978150	0.103871	0.9683
19	0.074739	0.979717	0.101629	0.9694

Рисунок 11 – История обучения по эпохам

По данному графику и истории видно, что данная модель более точно предсказывает значения, на что указывает значения val_loss и val_accuracy и положения графика

Модель №1

Количество слоёв было увеличено на 1, количество нейронов было уменьшено. Количество эпох – 40. На рисунках 12-13 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```
[28] model = Sequential()
model.add(Dense(32,input_dim = num_pixels,activation = 'linear'))
model.add(Dense(16,input_dim = num_pixels,activation = 'linear'))
model.add(Dense(10,activation = 'softmax'))

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

history = model.fit(X_train,y_train, validation_data = (X_test, y_test),epochs = 40, batch_size = 200, verbose = 0)
history_df = pd.DataFrame(history.history)
history_df
plotting(history)
```

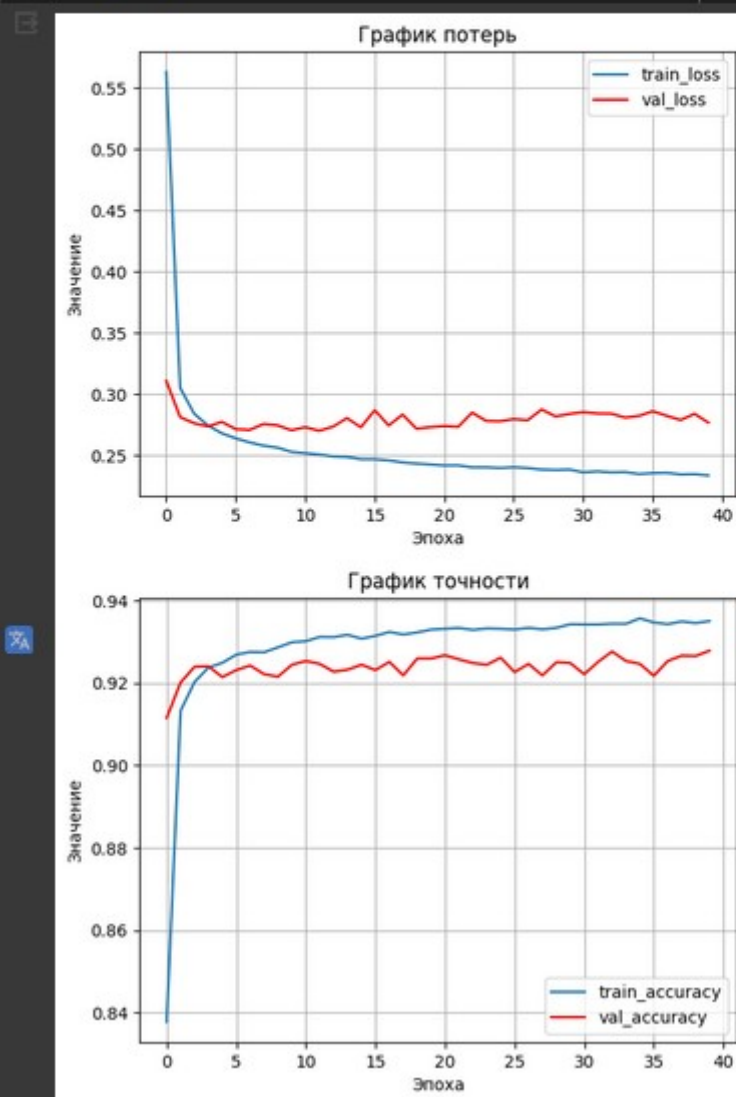


Рисунок 12 – Графики потерь и точности

history_df

8	0.236290	0.926617	0.274511	0.9215
9	0.252839	0.929850	0.270507	0.9244
10	0.251751	0.930100	0.272986	0.9253
11	0.250659	0.931150	0.270042	0.9246
12	0.249050	0.931100	0.273426	0.9227
13	0.248671	0.931650	0.280399	0.9232
14	0.246853	0.930717	0.272892	0.9244
15	0.246823	0.931400	0.286883	0.9231
16	0.245921	0.932350	0.274349	0.9251
17	0.244180	0.931733	0.283340	0.9218
18	0.243271	0.932233	0.271745	0.9259
19	0.242490	0.932917	0.273059	0.9259
20	0.241774	0.933117	0.273899	0.9267
21	0.241822	0.933317	0.273260	0.9257
22	0.240187	0.932850	0.284898	0.9248
23	0.240176	0.933200	0.278160	0.9244
24	0.239689	0.933117	0.277861	0.9261
25	0.240229	0.932933	0.279547	0.9226
26	0.239605	0.933350	0.278666	0.9246
27	0.238410	0.932983	0.287451	0.9218
28	0.238031	0.933350	0.281806	0.9250
29	0.238370	0.934217	0.283763	0.9248
30	0.236130	0.934167	0.285325	0.9221
31	0.236879	0.934167	0.284151	0.9251
32	0.236089	0.934333	0.284049	0.9276
33	0.236327	0.934317	0.280862	0.9253
34	0.234735	0.935650	0.282201	0.9246
35	0.235429	0.934650	0.286012	0.9217
36	0.235730	0.934233	0.282271	0.9253
37	0.234396	0.934900	0.278841	0.9266
38	0.234631	0.934500	0.283889	0.9265
39	0.233588	0.934983	0.276676	0.9278

Рисунок 13 – История обучения по эпохам

В сравнении с моделью с функцией активации linear и с двумя слоями, данная модель обладает чуть более лучшими значениями потерь и точности (отличие в сотых долях)

Модель №2

В данной модели было увеличено количество слоёв на 1, количество нейронов также было увеличено. На рисунках 14-15 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```

model = Sequential()
model.add(Dense(30, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(20, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(16, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(10, activation = 'softmax'))

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

history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 40, batch_size = 200, verbose = 0)
history_df = pd.DataFrame(history.history)
plotting(history)

```

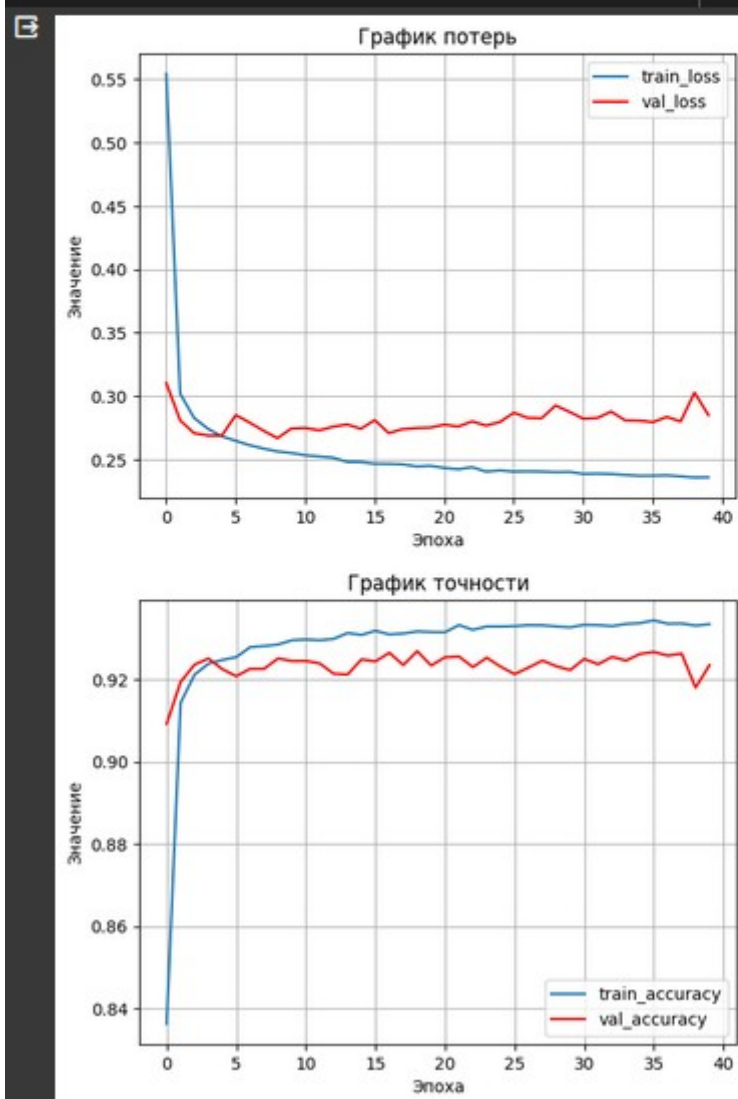
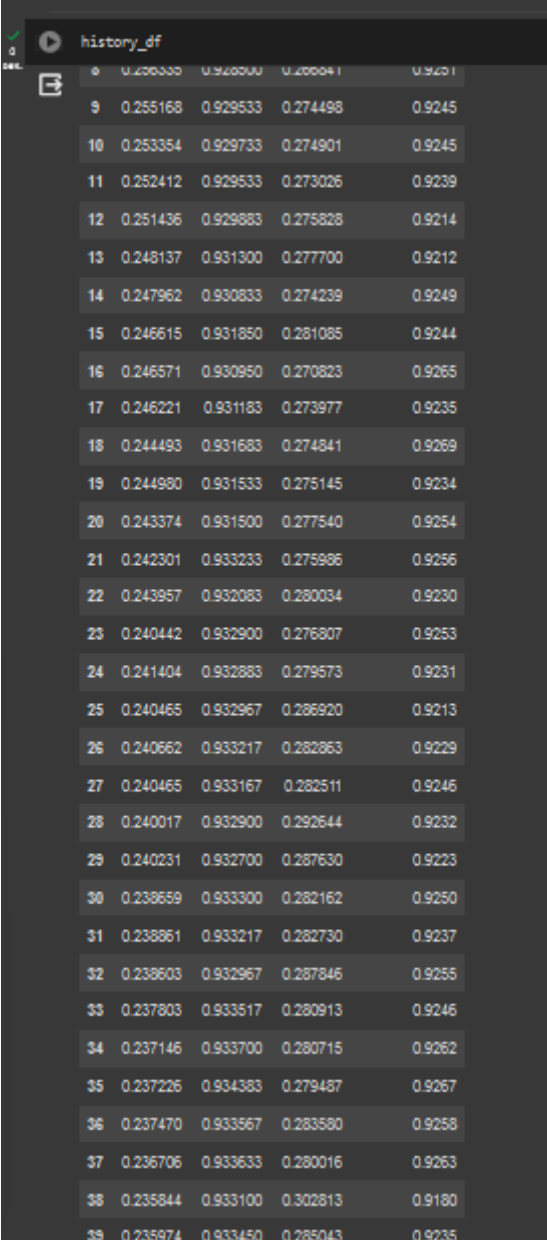


Рисунок 14 – Графики потерь и точности



	0.226333	0.926900	0.266041	0.9251
9	0.255168	0.929533	0.274498	0.9245
10	0.253354	0.929733	0.274901	0.9245
11	0.252412	0.929533	0.273026	0.9239
12	0.251436	0.929883	0.275828	0.9214
13	0.248137	0.931300	0.277700	0.9212
14	0.247962	0.930833	0.274239	0.9249
15	0.246615	0.931850	0.281085	0.9244
16	0.246571	0.930950	0.270823	0.9265
17	0.246221	0.931183	0.273977	0.9235
18	0.244493	0.931683	0.274841	0.9269
19	0.244980	0.931533	0.275145	0.9234
20	0.243374	0.931500	0.277540	0.9254
21	0.242301	0.933233	0.275986	0.9256
22	0.243957	0.932083	0.280034	0.9230
23	0.240442	0.932900	0.276807	0.9253
24	0.241404	0.932883	0.279573	0.9231
25	0.240465	0.932967	0.286920	0.9213
26	0.240662	0.933217	0.282863	0.9229
27	0.240465	0.933167	0.282511	0.9246
28	0.240017	0.932900	0.292644	0.9232
29	0.240231	0.932700	0.287630	0.9223
30	0.238659	0.933300	0.282162	0.9250
31	0.238861	0.933217	0.282730	0.9237
32	0.238603	0.932967	0.287846	0.9255
33	0.237803	0.933517	0.280913	0.9246
34	0.237146	0.933700	0.280715	0.9262
35	0.237226	0.934383	0.279487	0.9267
36	0.237470	0.933567	0.283580	0.9258
37	0.236706	0.933633	0.280016	0.9263
38	0.235844	0.933100	0.302813	0.9180
39	0.235974	0.933450	0.285043	0.9235

Рисунок 15 – История обучения модели по эпохам

Значения потерь и точности данной модели примерно такие же как и у предыдущей.

Модель №3

Был добавлен 1 слой и были добавлены нейроны. На рисунках 16-17 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```

model = Sequential()
model.add(Dense(40, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(30, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(20, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(16, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(10, activation = 'softmax'))

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

history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 40, batch_size = 200, verbose = 0)
history_df = pd.DataFrame(history.history)
history_df.plotting(history)

```

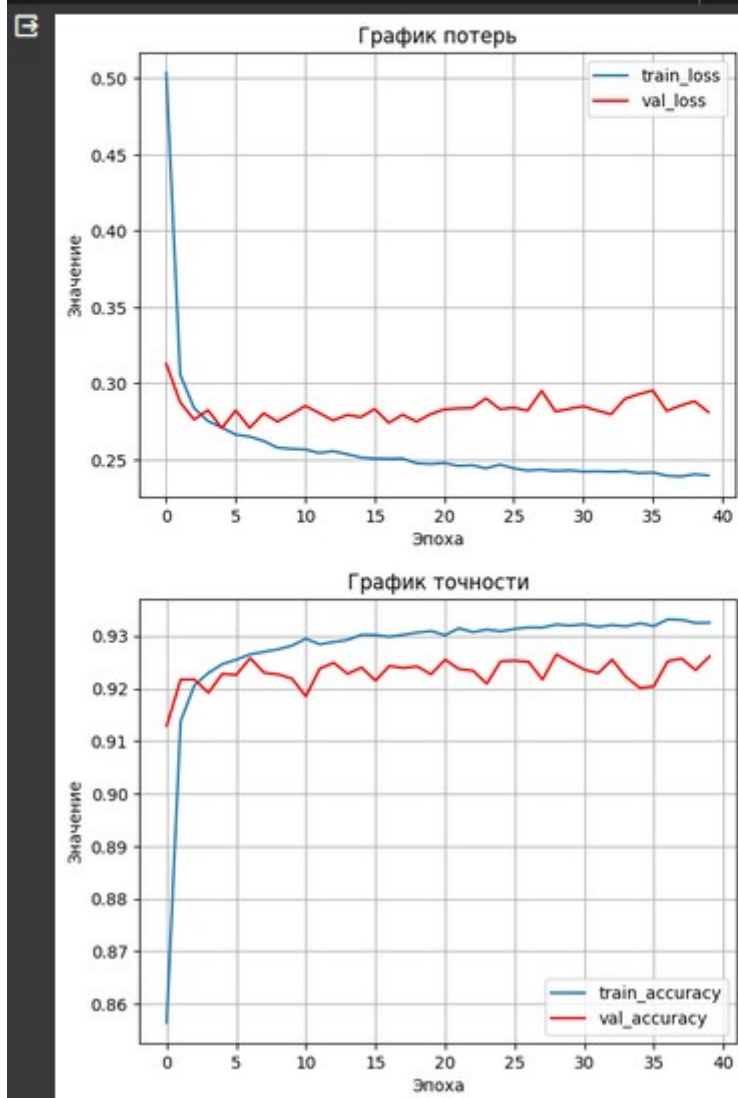
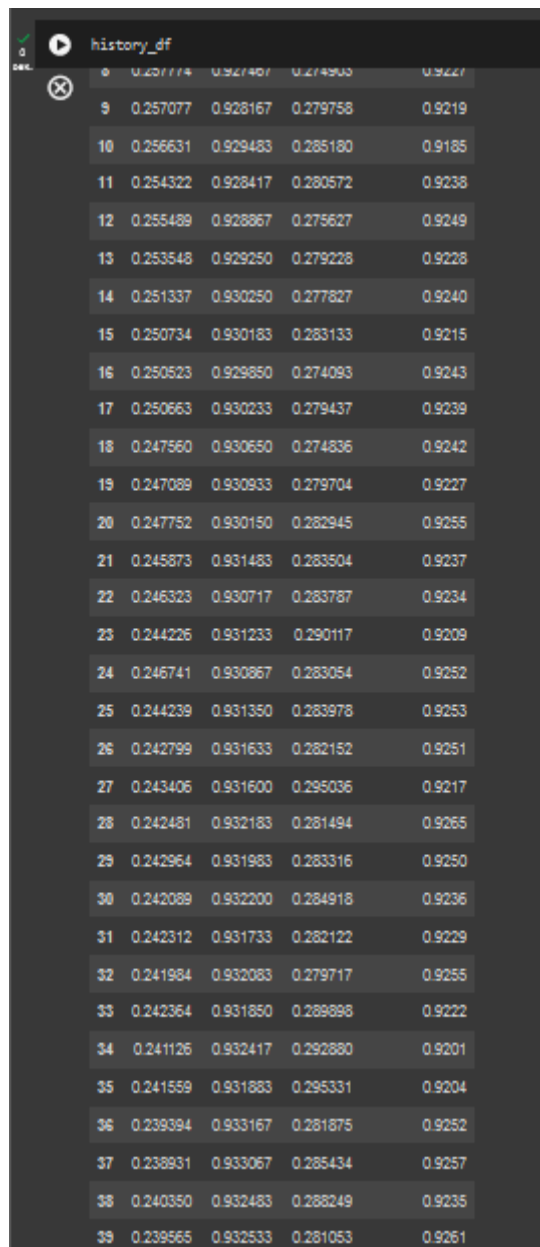


Рисунок 16 – Графики потерь и точности



	0	0.421774	0.927467	0.274993	0.9227
9	0.257077	0.928167	0.279758	0.9219	
10	0.256631	0.929483	0.285180	0.9185	
11	0.254322	0.928417	0.280572	0.9238	
12	0.255489	0.928867	0.275627	0.9249	
13	0.253548	0.929250	0.279228	0.9228	
14	0.251337	0.930250	0.277827	0.9240	
15	0.250734	0.930183	0.283133	0.9215	
16	0.250523	0.929850	0.274093	0.9243	
17	0.250663	0.930233	0.279437	0.9239	
18	0.247560	0.930650	0.274836	0.9242	
19	0.247089	0.930933	0.279704	0.9227	
20	0.247752	0.930150	0.282945	0.9255	
21	0.245873	0.931483	0.283504	0.9237	
22	0.246323	0.930717	0.283787	0.9234	
23	0.244226	0.931233	0.290117	0.9209	
24	0.246741	0.930867	0.283054	0.9252	
25	0.244239	0.931350	0.283978	0.9253	
26	0.242799	0.931633	0.282152	0.9251	
27	0.243406	0.931600	0.295036	0.9217	
28	0.242481	0.932183	0.281494	0.9265	
29	0.242964	0.931983	0.283316	0.9250	
30	0.242089	0.932200	0.284918	0.9236	
31	0.242312	0.931733	0.282122	0.9229	
32	0.241984	0.932083	0.279717	0.9255	
33	0.242364	0.931850	0.289898	0.9222	
34	0.241126	0.932417	0.292880	0.9201	
35	0.241559	0.931883	0.295331	0.9204	
36	0.239394	0.933167	0.281875	0.9252	
37	0.238931	0.933067	0.285434	0.9257	
38	0.240350	0.932483	0.288249	0.9235	
39	0.239565	0.932533	0.281053	0.9261	

Рисунок 17 – История обучения модели по эпохам

Модель №4

Был добавлен один слой и дополнительные нейроны. На рисунках 18-19 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```

model = Sequential()
model.add(Dense(50, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(40, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(30, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(20, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(16, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(10, activation = 'softmax'))

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

history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 40, batch_size = 200, verbose = 0)
history_df = pd.DataFrame(history.history)

plotting(history)

```

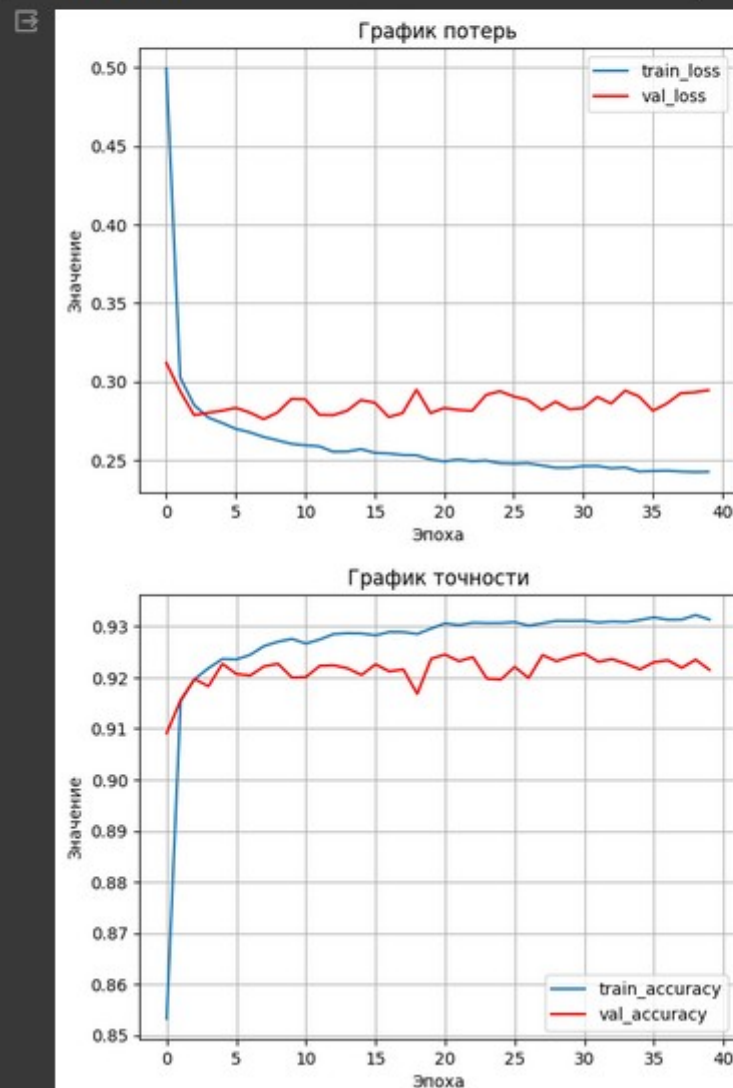
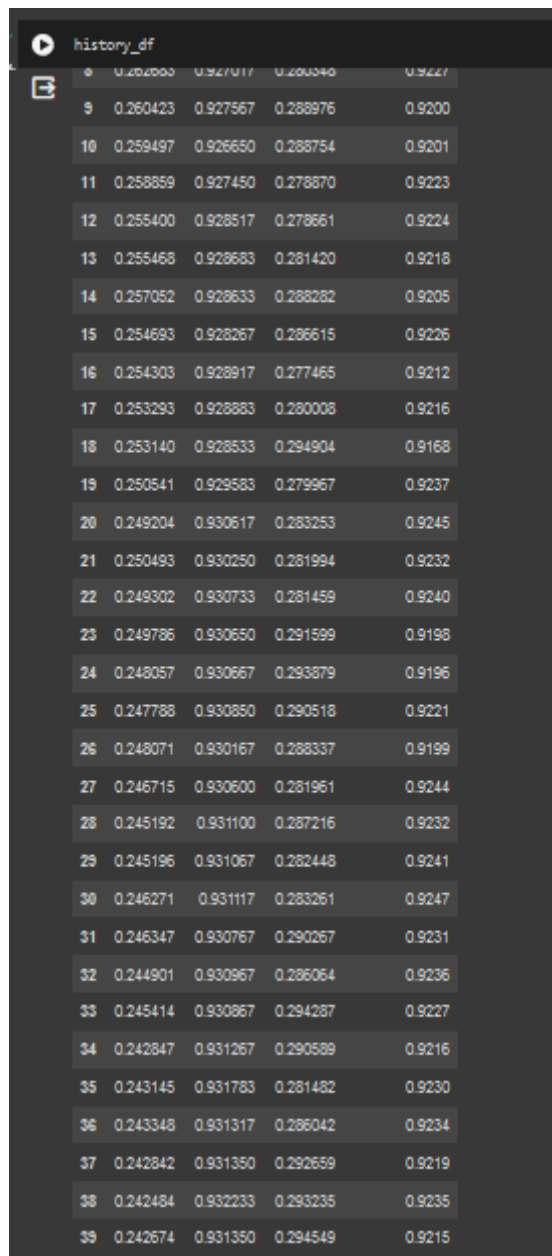


Рисунок 18 – Графики потерь и точности обучения по эпохам



The screenshot shows a Jupyter Notebook interface with a table titled 'history_df'. The table contains 40 rows of data, indexed from 0 to 39. Each row represents an epoch of training and contains four numerical values. The values generally show a decreasing trend over time, indicating the model's performance is improving.

	0	0.262063	0.927017	0.260340	0.9227
9	0.260423	0.927567	0.268976	0.9200	
10	0.259497	0.926650	0.268754	0.9201	
11	0.258859	0.927450	0.278870	0.9223	
12	0.255400	0.928517	0.278661	0.9224	
13	0.255468	0.928683	0.281420	0.9218	
14	0.257052	0.928633	0.268282	0.9205	
15	0.254693	0.928267	0.266615	0.9226	
16	0.254303	0.928917	0.277465	0.9212	
17	0.253293	0.928883	0.260008	0.9216	
18	0.253140	0.928533	0.294904	0.9168	
19	0.250541	0.929583	0.279967	0.9237	
20	0.249204	0.930617	0.283253	0.9245	
21	0.250493	0.930250	0.281994	0.9232	
22	0.249302	0.930733	0.281459	0.9240	
23	0.249786	0.930650	0.291599	0.9198	
24	0.248057	0.930667	0.293879	0.9196	
25	0.247788	0.930850	0.290518	0.9221	
26	0.248071	0.930167	0.288337	0.9199	
27	0.246715	0.930600	0.281961	0.9244	
28	0.245192	0.931100	0.287216	0.9232	
29	0.245196	0.931067	0.282448	0.9241	
30	0.246271	0.931117	0.283261	0.9247	
31	0.246347	0.930767	0.290267	0.9231	
32	0.244901	0.930967	0.286064	0.9236	
33	0.245414	0.930867	0.294287	0.9227	
34	0.242847	0.931267	0.290589	0.9216	
35	0.243145	0.931783	0.281482	0.9230	
36	0.243348	0.931317	0.286042	0.9234	
37	0.242842	0.931350	0.292659	0.9219	
38	0.242484	0.932233	0.293235	0.9235	
39	0.242674	0.931350	0.294549	0.9215	

Рисунок 19 – История обучения модели по эпохам

Модель №5

Был добавлен один слой и дополнительные нейроны. На рисунках 20-21 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```

model = Sequential()
model.add(Dense(60, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(50, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(40, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(30, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(20, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(16, input_dim = num_pixels, activation = 'linear'))
model.add(Dense(10, activation = 'softmax'))

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

history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 40, batch_size = 200, verbose = 0)
history_df = pd.DataFrame(history.history)

plotting(history)

```

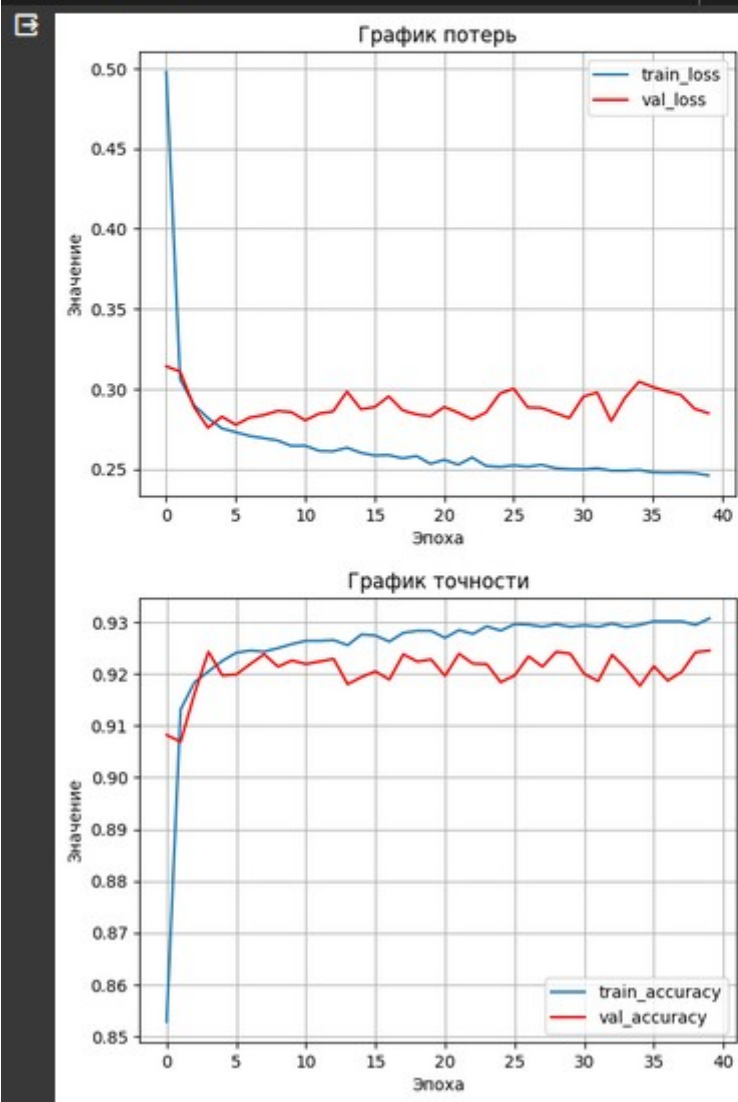
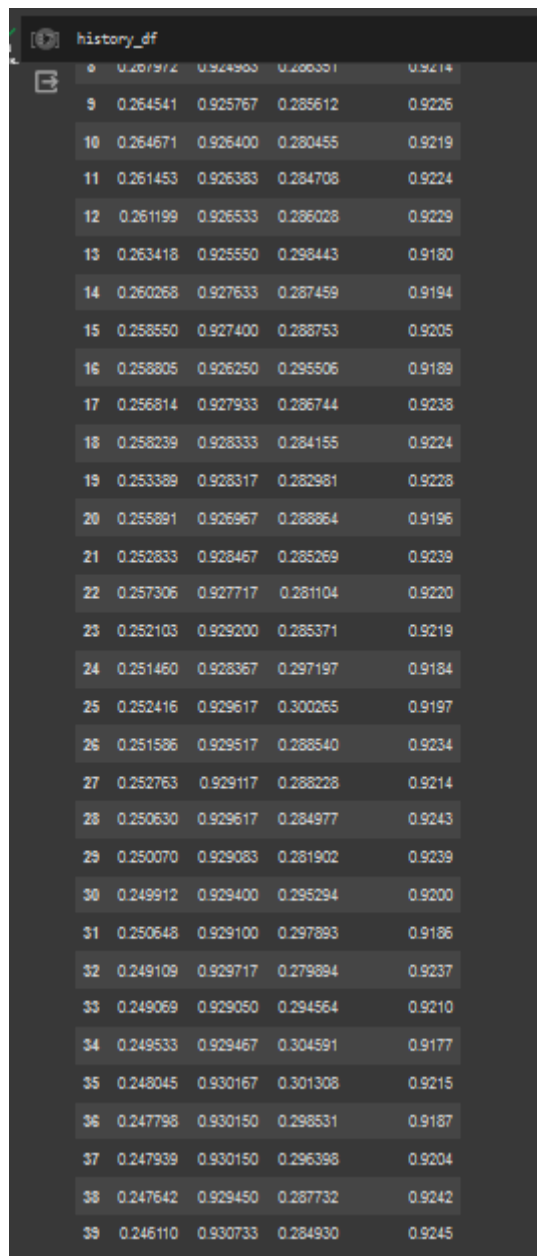


Рисунок 20 – Графики потерь и точности



The screenshot shows a Jupyter Notebook interface with a variable named 'history_df' containing a table of training history data. The table has 4 columns and 39 rows (index 8 to 38). The columns represent different metrics, likely loss and accuracy for training and validation sets.

	0	1	2	3
8	0.267912	0.924963	0.286331	0.9214
9	0.264541	0.925767	0.285612	0.9226
10	0.264671	0.926400	0.280455	0.9219
11	0.261453	0.926383	0.284708	0.9224
12	0.261199	0.926533	0.286028	0.9229
13	0.263418	0.925550	0.298443	0.9180
14	0.260268	0.927633	0.287459	0.9194
15	0.258650	0.927400	0.288753	0.9205
16	0.258805	0.926250	0.295506	0.9189
17	0.256814	0.927933	0.286744	0.9238
18	0.258239	0.928333	0.284155	0.9224
19	0.253389	0.928317	0.282981	0.9228
20	0.255891	0.926967	0.288864	0.9196
21	0.252833	0.928467	0.285269	0.9239
22	0.257306	0.927717	0.281104	0.9220
23	0.252103	0.929200	0.285371	0.9219
24	0.251460	0.928367	0.297197	0.9184
25	0.252416	0.929617	0.300265	0.9197
26	0.251586	0.929517	0.288540	0.9234
27	0.252763	0.929117	0.288228	0.9214
28	0.250630	0.929617	0.284977	0.9243
29	0.250070	0.929083	0.281902	0.9239
30	0.249912	0.929400	0.295294	0.9200
31	0.250648	0.929100	0.297893	0.9186
32	0.249109	0.929717	0.279894	0.9237
33	0.249069	0.929050	0.294564	0.9210
34	0.249533	0.929467	0.304591	0.9177
35	0.248045	0.930167	0.301308	0.9215
36	0.247798	0.930150	0.298531	0.9187
37	0.247939	0.930150	0.296398	0.9204
38	0.247642	0.929450	0.287732	0.9242
39	0.246110	0.930733	0.284930	0.9245

Рисунок 21 – История обучения модели по эпохам

По данным моделям можно сделать вывод о том, что при добавлении слоёв и увеличении количества нейронов на данных входных данных сложно добиться значительных изменений в качестве предсказания, возможно, стоит изменить другие параметры модели для достижения наилучшей производительности

Сверточная нейронная сеть №1

Была создана, скомпилирована и обучена сверточная нейронная сеть, содержащая в себе 1 сверточный слой с размером 3x3, 1 слой пулинга с размером 2x2, 1 слой дропаут с параметром 0.25, полносвязный слой с 128 нейронами и функцией активации relu, также выходной слой с функцией активации softmax. На рисунках 22-23 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```
[38] # Создание сверточной нейронной сети
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
# Компиляция
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=40,
                    batch_size=200, verbose=0)
history_df = pd.DataFrame(history.history)
#
plotting(history)
```

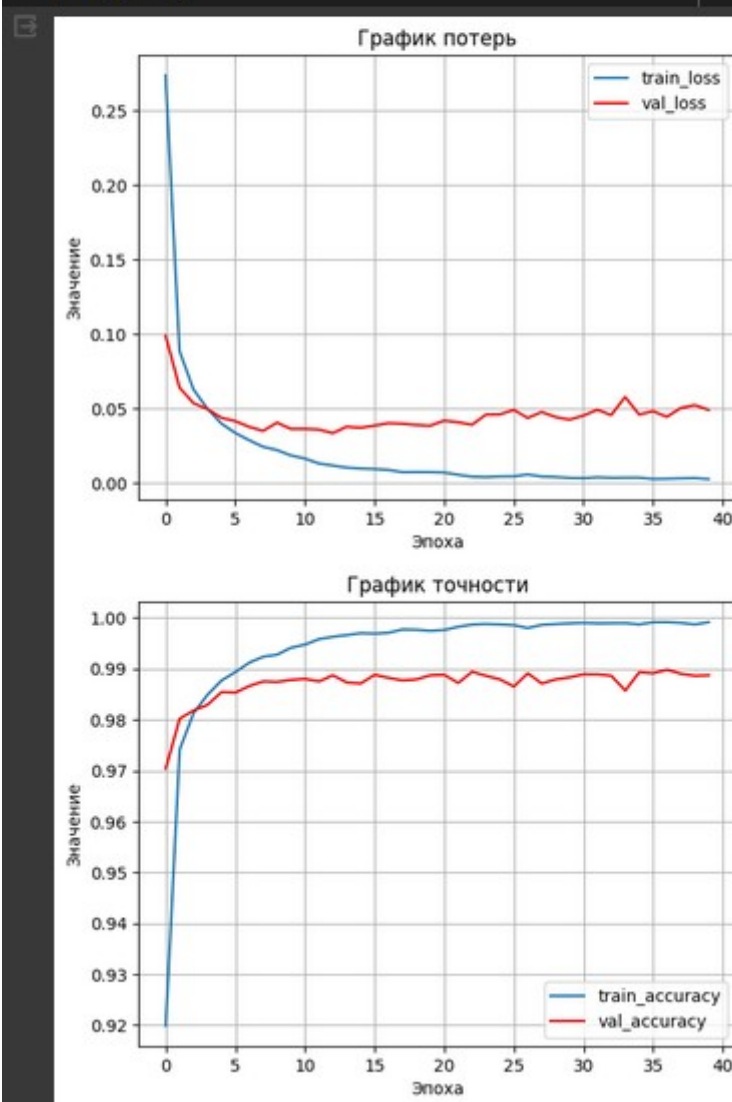


Рисунок 22 – График потерь и точности

```
[39] history_df
```

8	0.021971	0.992767	0.040396	0.9874
9	0.018372	0.994117	0.036167	0.9878
10	0.016290	0.994733	0.036199	0.9880
11	0.012976	0.995800	0.035797	0.9875
12	0.011586	0.996267	0.033317	0.9887
13	0.010196	0.996600	0.037684	0.9873
14	0.009565	0.996967	0.036992	0.9871
15	0.009276	0.996883	0.038421	0.9888
16	0.008676	0.997050	0.040032	0.9882
17	0.007131	0.997700	0.039686	0.9877
18	0.007215	0.997633	0.038817	0.9879
19	0.007177	0.997433	0.038316	0.9887
20	0.006905	0.997600	0.041786	0.9888
21	0.006399	0.998233	0.040698	0.9872
22	0.004113	0.998667	0.038857	0.9894
23	0.003811	0.998767	0.045826	0.9886
24	0.004263	0.998683	0.045966	0.9879
25	0.004344	0.998550	0.048941	0.9865
26	0.005600	0.997983	0.043547	0.9891
27	0.004224	0.998617	0.047552	0.9871
28	0.003869	0.998750	0.044182	0.9879
29	0.003269	0.998900	0.042434	0.9883
30	0.003101	0.999000	0.045111	0.9889
31	0.003681	0.998900	0.049065	0.9889
32	0.003337	0.998933	0.045497	0.9886
33	0.003430	0.998950	0.057690	0.9857
34	0.003441	0.998683	0.045839	0.9893
35	0.002566	0.999133	0.048099	0.9891
36	0.002699	0.999150	0.044336	0.9898
37	0.003000	0.999000	0.050262	0.9890
38	0.003189	0.998683	0.052176	0.9886
39	0.002469	0.999150	0.048962	0.9887

Рисунок 23 – История обучения модели по эпохам

Данная модель предсказывает значения более точно в сравнении с полносвязной моделью

Сверточная нейронная сеть №2

Данная модель обладает 3 сверточными слоями, 2 слоями пуллинга и одним слоем дропаут. Полносвязные слои такие же, как и в предыдущей модели. На рисунках 24-25 изображены графики потерь и точности обучения по эпохам и история обучения соответственно.

```

# Создание сверточной нейронной сети
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28, 28, 1))) # слой свертки с 32 фильтрами размером 3 на 3, функция активации relu
model.add(MaxPooling2D(pool_size=(2, 2))) # слой подвыборки (уменьшает размерность карт чтобы уменьшить кол-во параметров)
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3, 3), activation = 'relu'))
model.add(Dropout(0.25)) # выпадение 25% нейронов чтобы не было переобучения
model.add(Flatten()) # выравнивание данных перед подачей на полносвязный слой потому что следующий слой ожидает вектор
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
# Компиляция
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=40,
                    batch_size=200, verbose=0)
history_df = pd.DataFrame(history.history)
#
plotting(history)

```

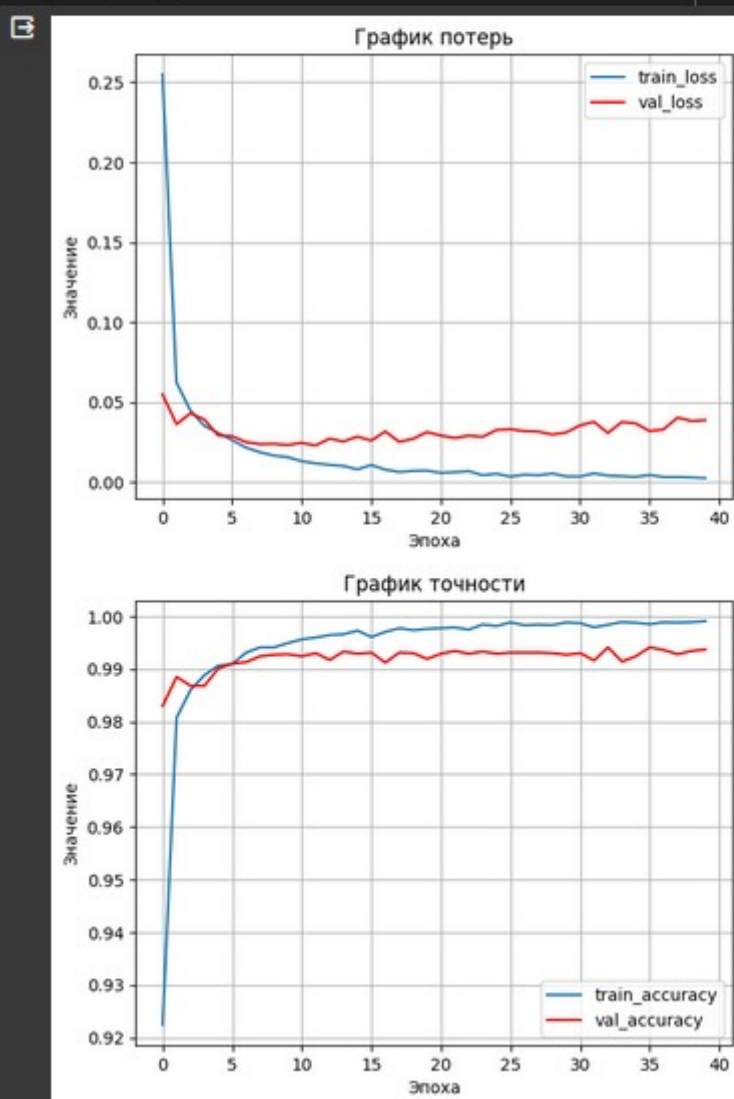


Рисунок 24 – Графики потерь и точности

Epoch	Value 1	Value 2	Value 3	Value 4
9	0.015717	0.994917	0.023351	0.9928
10	0.013222	0.995617	0.024591	0.9924
11	0.011834	0.995967	0.023089	0.9930
12	0.010910	0.996433	0.027267	0.9917
13	0.010249	0.996583	0.025520	0.9933
14	0.008077	0.997283	0.028596	0.9929
15	0.010814	0.996067	0.026026	0.9931
16	0.007949	0.997050	0.031809	0.9912
17	0.006474	0.997717	0.025347	0.9931
18	0.007158	0.997367	0.027126	0.9930
19	0.007337	0.997600	0.031418	0.9919
20	0.005898	0.997750	0.029230	0.9929
21	0.006324	0.997883	0.027721	0.9934
22	0.007028	0.997467	0.029188	0.9929
23	0.004460	0.998450	0.028409	0.9933
24	0.005383	0.998150	0.032730	0.9929
25	0.003552	0.998850	0.033229	0.9931
26	0.004812	0.998300	0.031916	0.9931
27	0.004428	0.998417	0.031703	0.9931
28	0.005519	0.998333	0.029855	0.9930
29	0.003740	0.998833	0.031155	0.9927
30	0.003636	0.998683	0.035570	0.9930
31	0.005595	0.997917	0.037786	0.9916
32	0.004213	0.998400	0.030796	0.9941
33	0.003836	0.998900	0.037669	0.9914
34	0.003411	0.998767	0.036789	0.9924
35	0.004595	0.998500	0.032016	0.9941
36	0.003309	0.998867	0.033053	0.9936
37	0.003353	0.998800	0.040371	0.9928
38	0.003094	0.998883	0.038338	0.9934
39	0.002624	0.999083	0.038863	0.9937

Рисунок 25 – История обучения модели по эпохам

Данная модель предсказывает значения точнее, чем предыдущая.

Выводы:

На основе полученных графиков и таблицы можно сделать следующие выводы:

- Наилучшие результаты показали сверточные нейронные сети.
- Наилучший показатель у сверточной нейронной сети с более сложной структурой
 - При одинаково количестве эпох = 40 наилучший результат показывали самые сложные нейронные сети
 - Функция активации sigmoid показала лучшие результаты в сравнении с функцией активации linear

Рукописные

Было создано два собственных изображения рукописной цифры, загружены в программу и проверен результат предсказания лучшей модели.

Для реализации данной задачи была написана функция, преобразовывающая изображения для работы с нейронной сетью (рисунок 26). Модель повторяет архитектуру модели сверточной нейронной сети №2 (рисунок 27).

```
def refactor_image(path):
    img = Image.open(path)
    img = img.resize((28, 28)) # т.к. выбрана модель CNN, то преобразуем размер на подходящий
    img = img.convert('L') # преобразования изображения в оттенки серого
    img_arr = np.array(img) # т.к. изображение в нейронной сети воспринимается как набор чисел, то его необходимо преобразовать в np.array для дальнейшей работы с ним
    img_arr = np.expand_dims(img_arr, axis=0) # добавляем размерность для работы с CNN
    return img_arr
```

Ри

Рисунок 26 – Функция преобработки изображения

```
[59] # Создание сверточной нейронной сети
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28, 28, 1))) # слой свертки с 32 фильтрами размером 3 на 3, функция активации relu
model.add(MaxPooling2D(pool_size=(2, 2))) # слой подвыборки (уменьшает размерность карт чтобы уменьшить кол-во параметров)
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(128, (3, 3), activation = 'relu'))
model.add(Dropout(0.25)) # выпадение 25% нейронов чтобы не было переобучения
model.add(Flatten()) # выравнивание данных перед подачей на полносвязный слой потому что следующий слой ожидает вектор
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
# Компиляция
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=40,
                    batch_size=200, verbose=0)
```

Рисунок 27 – Создание, компилирование и обучение модели

На основе данных изображений были выполнены предсказания (рисунок 28).


```
# загрузка
one = refactor_image('1_1.png')
two = refactor_image('2_1.png')
three = refactor_image('3_1.png')
four = refactor_image('4_1.png')
seven = refactor_image('7_1.png')
eight = refactor_image('8_1.png')

# предсказание для единицы
prediction_one = model.predict(one)
print(prediction_one)
index_one = np.argmax(prediction_one) # выбираем класс с наибольшей вероятностью
print("Первое изображение - ",index_one)
print("Написано было - 1")

# предсказание для двойки
prediction_two = model.predict(two)
print(prediction_two)
index_two = np.argmax(prediction_two) # выбираем класс с наибольшей вероятностью
print("Первое изображение - ",index_two)
print("Написано было - 2")

# предсказание для тройки
prediction_three = model.predict(three)
print(prediction_three)
index_three = np.argmax(prediction_three) # выбираем класс с наибольшей вероятностью
print("Первое изображение - ",index_three)
print("Написано было - 3")

# предсказание для четверки
prediction_four = model.predict(four)
print(prediction_four)
index_four = np.argmax(prediction_four) # выбираем класс с наибольшей вероятностью
print("Первое изображение - ",index_four)
print("Написано было - 4")

# предсказание для семерки
prediction_seven = model.predict(seven)
print(prediction_seven)
index_seven = np.argmax(prediction_seven) # выбираем класс с наибольшей вероятностью
print("Первое изображение - ",index_seven)
print("Написано было - 7")

# предсказание для восьмерки
prediction_eight = model.predict(eight)
print(prediction_eight)
index_eight = np.argmax(prediction_eight) # выбираем класс с наибольшей вероятностью
print("Первое изображение - ",index_eight)
print("Написано было - 8")

1/1 [=====] - 0s 29ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
Первое изображение - 8
Написано было - 1
1/1 [=====] - 0s 29ms/step
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Первое изображение - 0
Написано было - 2
1/1 [=====] - 0s 28ms/step
[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
Первое изображение - 4
Написано было - 3
1/1 [=====] - 0s 29ms/step
[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
Первое изображение - 4
Написано было - 4
1/1 [=====] - 0s 28ms/step
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
Первое изображение - 7
Написано было - 7
1/1 [=====] - 0s 29ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
Первое изображение - 9
Написано было - 8
```

Рисунок 28 – Результат предсказывания

Данная модель правильно предсказала 2 значения (4 и 7), остальные были предсказаны неверно. Для вывода изображений их необходимо было привести к правильному формату (рисунок 29).

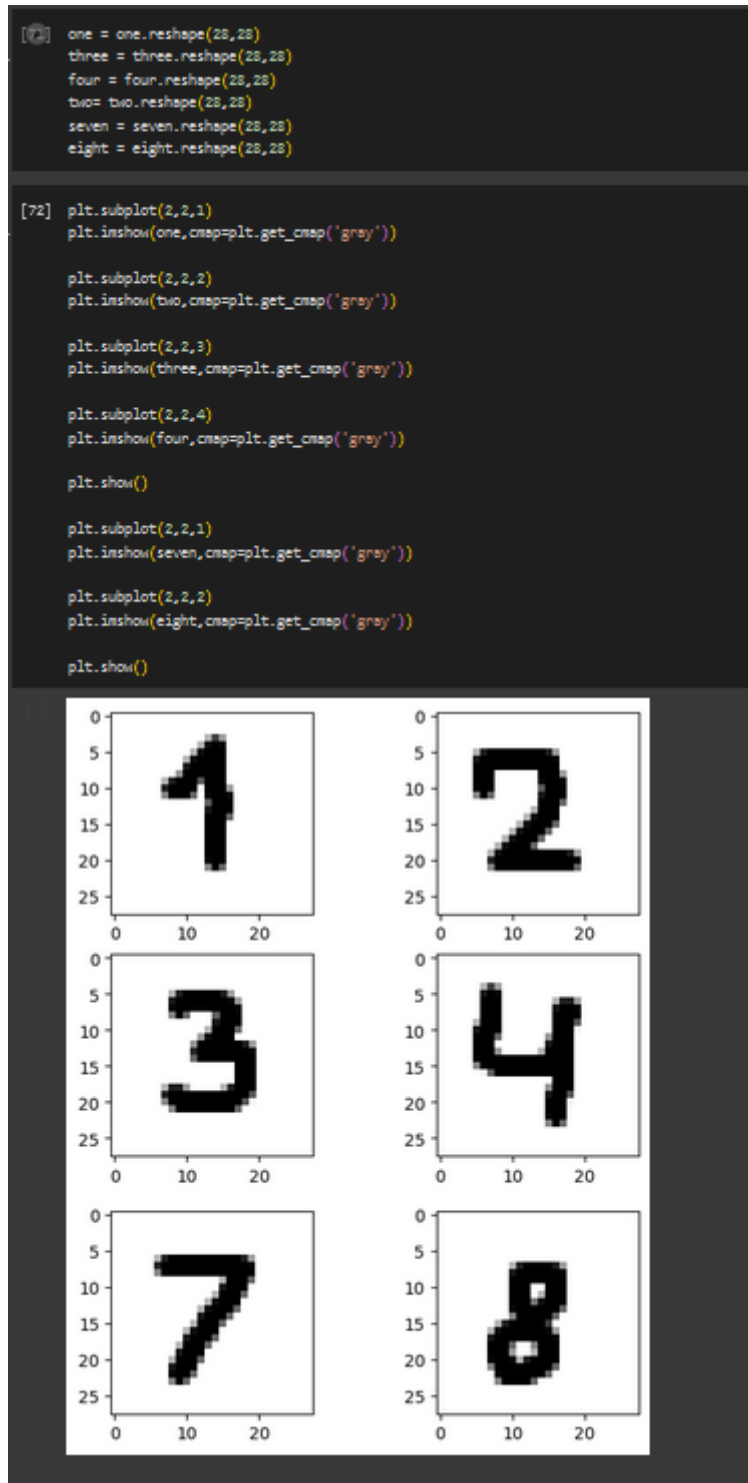
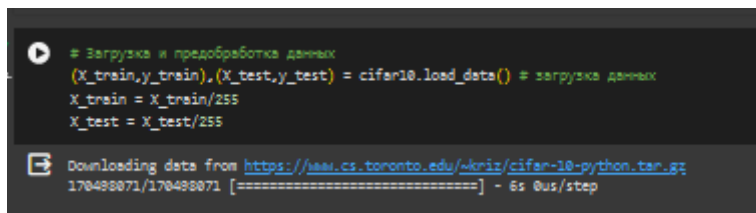


Рисунок 29 – Приведение изображений к корректному формату для вывода и вывод
Нейронная сеть для распознавания изображений набора данных CIFAR10

Датасет CIFAR-10 состоит из 60000 цветных изображений размером 32x32, поделенных на 10 классов. В датасете 50000 тренировочных изображений и 10000 тестовых. В данном датасете присутствуют такие классы как: самолет, автомобиль, птица, кот, олень, собака, лягушка, лошадь, корабль, грузовик. На рисунке 30 изображена загрузка и обработка данных.



```
# Загрузка и предобработка данных
(X_train,y_train),(X_test,y_test) = cifar10.load_data() # загрузка данных
X_train = X_train/255
X_test = X_test/255
```

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
170498071/170498071 [=====] - 6s 0us/step

Рисунок 30 – Загрузка и обработка данных

Примеры изображений были визуализированы (рисунок 31).

[46]

```
plt.subplot(2,2,1)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))

plt.subplot(2,2,2)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))

plt.subplot(2,2,3)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))

plt.subplot(2,2,4)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))

plt.show()

plt.subplot(2,2,1)
plt.imshow(X_test[0], cmap=plt.get_cmap('gray'))

plt.subplot(2,2,2)
plt.imshow(X_test[1], cmap=plt.get_cmap('gray'))

plt.subplot(2,2,3)
plt.imshow(X_test[2], cmap=plt.get_cmap('gray'))

plt.subplot(2,2,4)
plt.imshow(X_test[3], cmap=plt.get_cmap('gray'))

plt.show()
```



Рисунок 31 – Примеры изображений

Была создана модель с 3 сверточными слоями, двумя слоями подвыборки и с тремя слоями дропаута для избежания переобучения. На рисунке 32-33 изображены графики точности и потерь и история обучения модели по эпохам соответственно.

```
[47] # Создание сверточной нейронной сети
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(32, 32, 3))) # слой сверток с 32 фильтрами размером 3 на 3, функция активации relu
model.add(Conv2D(32,(3,3),activation = 'relu'))
model.add(MaxPooling2D()) # слой подвыборки
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),activation = 'relu'))
model.add(MaxPooling2D()) # слой подвыборки
model.add(Dropout(0.25))

model.add(Flatten()) # выравнивание данных перед подачей на полносвязный слой
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))

y_train = to_categorical(y_train) # преобразование правильных ответов в one-hot код
y_test = to_categorical(y_test)
# Компиляция
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
# Обучение модели
History = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, verbose=0)
History_df = pd.DataFrame(History.history)
#
plotting(History)
```

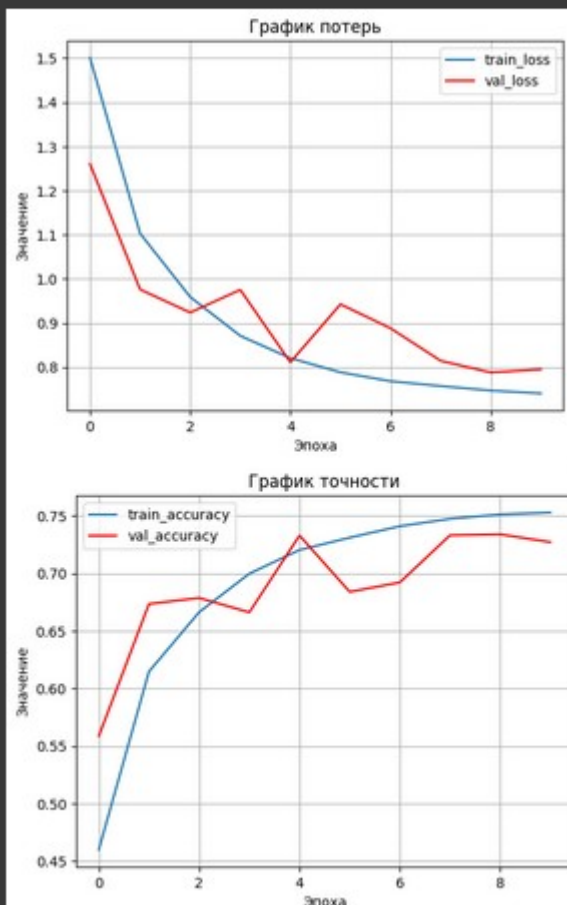
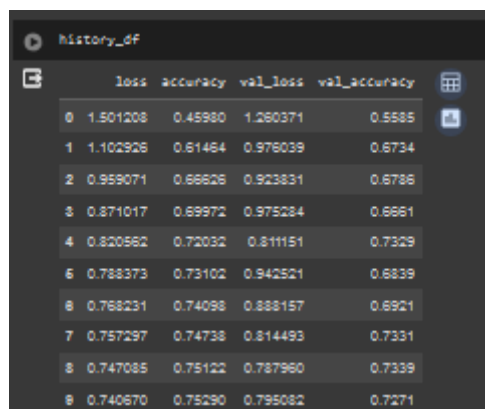


Рисунок 32 – Графики точности и потерь



	loss	accuracy	val_loss	val_accuracy
0	1.501208	0.45980	1.260371	0.5585
1	1.102926	0.61464	0.976039	0.6734
2	0.959071	0.66626	0.923831	0.6786
3	0.871017	0.69972	0.975284	0.6661
4	0.820562	0.72032	0.811151	0.7329
5	0.788373	0.73102	0.942521	0.6839
6	0.768231	0.74098	0.888157	0.6921
7	0.757297	0.74738	0.814493	0.7331
8	0.747085	0.75122	0.787960	0.7339
9	0.740670	0.75290	0.795082	0.7271

Рисунок 33 – История обучения модели по эпохам

По данным графикам можно сделать вывод о том, что значения точности и потерь улучшаются с каждой эпохой до 8, после которой заметно "сглаживание" графика, по чему можно понять, что значения в дальнейшем будут примерно такие же.

На рисунке 34 приведена таблица, содержащая данные о точности и потерях для каждой модели, обученной предсказывать рукописные цифры на основе набора данных MNIST.

Номер эксперимента	Модель	Архитектура	Количество слоев	Функция активации	Количество эпох	Точность	Потери
Линейная функция акт	DNN	D-64		2 linear	20	0.9282	0.275580
Нелинейная функция	DNN	D-64		2 sigmoid	20	0.9696	0.1051
Модель №1	DNN	D-32,D-16		3 linear	40	0.9243	0.284
Модель №2	DNN	D-40,D-30,D-20,D-16		4 linear	40	0.9232	0.2855
Модель №3	DNN	D-40,D-30,D-20,D-16		5 linear	40	0.9217	0.289608
Модель №4	DNN	D-50,D-40,D-30,D-20,D-16		6 linear	40	0.9247	0.29012
Модель №5	DNN	D-60,D-50,D-40,D-30,D-20,D-16		7 linear	40	0.9202	0.2986
Сверточная №1	CNN	Conv,Pool,Dropout,D-16		6 relu	40	0.98	0.04
Сверточная №2	CNN	Conv,Pool,Conv,Pool,C		9 relu	40	0.99	0.02

Рисунок 34 – Таблица

Ссылка на Colab

https://colab.research.google.com/drive/1JAsEGaxJbeELgIuA_dl7VxMTIyGuxv1b?usp=sharing

Выводы

В процессе выполнения данной лабораторной работы были получены навыки работы с полносвязными моделями нейронных сетей, с сверточными нейронными сетями для классификации изображений, были получены навыки анализирования оценок работы нейронных сетей с различной архитектурой. Были проведены тесты на наборе данных с изображениями рукописных цифр MNIST, в результате чего были сделаны выводы о том, что при обучении модели на данном наборе данных она достаточно точно предсказывает значения на валидационных данных из набора данных MNIST, однако, при тестировании данной модели на собственных рукописных цифрах результаты оказались куда хуже, что

может заключаться в значительном отличии цифр в наборе от написанных для тестов. Также были проведены тесты модели CIFAR10, в результате которых были сделаны выводы о том, что для точных предсказываний необходимо подбирать оптимальные настройки для каждого слоя модели.