Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем
Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №3
«Реализация сверточной нейронной сети»

Магистрант:
гр. 956241
Шуба И.А.

Проверил:
Заливако С. С.

Минск, 2020

ХОД РАБОТЫ

Задание.

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A … J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке:

- https://commondatastorage.googleapis.com/books1000/ notMNIST_large.tar.gz (большой набор данных);
- https://commondatastorage.googleapis.com/books1000/ notMNIST_small.tar.gz (маленький набор данных);

Описание данных на английском языке доступно по ссылке: http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html

Задание 1.

Реализуйте нейронную сеть с двумя сверточными слоями, и одним полносвязным с нейронами с кусочно-линейной функцией активации. Какова точность построенное модели?

Задание 2.

Замените один из сверточных слоев на слой, реализующий операцию пулинга (Pooling) с функцией максимума или среднего. Как это повлияло на точность классификатора?

Задание 3.

Реализуйте классическую архитектуру сверточных сетей LeNet-5 (http://yann.lecun.com/exdb/lenet/).

Задание 4.

Сравните максимальные точности моделей, построенных в лабораторных работах 1-3. Как можно объяснить полученные различия?

Результат выполнения:

Задание 1. Реализуйте нейронную сеть с двумя сверточными слоями, и одним полносвязным с нейронами с кусочно-линейной функцией активации. Какова точность построенное модели?

Загрузим датасет, как в предыдущих лабораторных работах.

```
X_features = pickle.load(open("X_features.pickle","rb"))
Y_label = pickle.load(open("Y_labels.pickle","rb"))
```

```
X_features = np.array(X_features) / 255 # normalization of data for easy to calculation
s.
Y_label = np.array(Y_label)
```

```
resolution = 28
classes = 10
```

```
X_features = X_features.reshape((-1, resolution, resolution, 1))
Y_labels = np_utils.to_categorical(Y_label, 10)
```

```
TRAIN = 200000
VAL = 10000
TEST = 19000
```

```
x_train, y_train = X_features[:TRAIN], Y_labels[:TRAIN]
x_val, y_val = X_features[TRAIN:TRAIN+VAL], Y_labels[TRAIN:TRAIN+VAL]
x_test, y_test = X_features[TRAIN+VAL:TRAIN+VAL+TEST], Y_labels[TRAIN+VAL:TRAIN+VAL+TES
T]
```

Рисунок 1 – Загрузка данных

Создадим нейронную сеть, состоящую из 2 сверточных слоев. Функции активации для каждого слоя – relu, для последнего слоя – softmax. В качестве алгоритма оптимизации выберем adam.

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu',
                 input_shape=(resolution, resolution, 1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
keras2ascii(model)
```

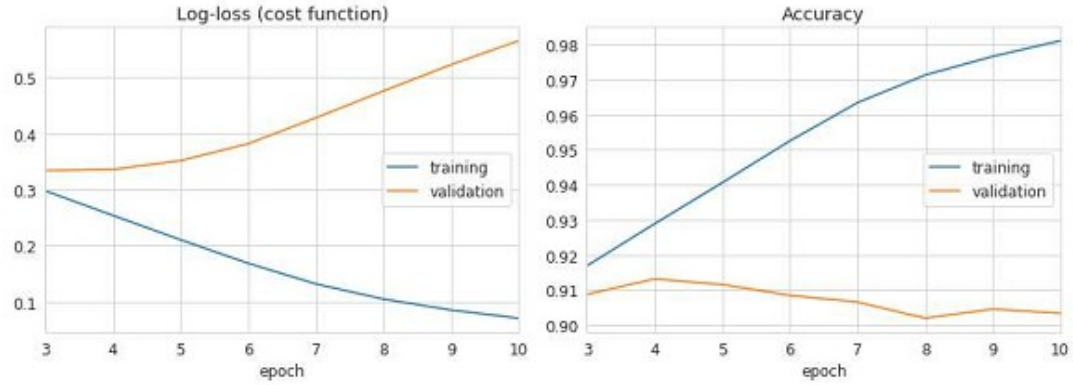
Рисунок 2 – Реализация сверточной нейронной сети

```
OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

    Input   #####     28   28    1
   Conv2D    \|/  -------------------       320     0.1%
     relu   #####     26   26   32
   Conv2D    \|/  -------------------     18496     4.8%
     relu   #####     24   24   64
  Flatten   ||||| -------------------         0     0.0%
            #####       36864
    Dense   XXXXX -------------------    368650    95.1%
  softmax   #####          10
```

Рисунок 3 – Схема сверточной нейронной сети

Обучение будет проводиться с размером батча 128 и с 10 эпохами.

```
model.fit(x_train, y_train,
          epochs=10,
          batch_size=128,
          validation_data=(x_val, y_val),
          callbacks=[PlotLossesKeras()],
          verbose=1)
```




```
Log-loss (cost function):
training   (min:    0.070, max:    0.442, cur:    0.070)
validation (min:    0.334, max:    0.565, cur:    0.565)

Accuracy:
training   (min:    0.879, max:    0.981, cur:    0.981)
validation (min:    0.899, max:    0.913, cur:    0.903)
```

Рисунок 4 – Результаты обучения

Как видно из графиков, точность на валидационных данных ниже, чем на обученных. Максимальная точность на валидационных данных составила 0,913, а минимальный log-loss – 0,334. Стоит обратить внимание, что log-loss должна снижаться с каждой эпохой, однако видно, что она растет. На графике с точностью произошло примерно тоже самое, мы ожидаем рост точности с каждой итерацией, однако происходит ее небольшой спад.

Проверим полученную модель на контрольной выборке.

```
y_pred = model.predict_classes(x_test)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1:', f1_score(y_test, y_pred, average='weighted'))
confusion_matrix(y_test, y_pred)
```

```
Accuracy: 0.9021578947368422
F1: 0.9022237317276522

Out[26]:

array([[1585,   26,   14,   22,   17,   23,   15,   45,   29,   17],
       [  19, 1681,   13,   41,   51,   22,   14,   29,   30,   17],
       [   4,   15, 1778,    9,   61,   21,   39,    8,   17,   10],
       [  11,   29,   15, 1694,   21,   16,   10,    9,   20,   28],
       [  12,   13,   37,    8, 1777,   22,   26,   11,   21,    8],
       [   9,   12,   17,   10,   52, 1751,   19,   21,   20,   19],
       [  13,   18,   45,   21,   40,   19, 1684,   14,   23,   12],
       [  20,   20,    5,   14,   18,   14,   15, 1681,   23,   16],
       [  13,   21,   20,   29,   27,   14,   13,   25, 1740,   52],
       [  11,   11,   10,   22,   21,   22,    9,   17,   48, 1770]])
```

Рисунок 5 – Результаты модели на контрольной выборке

Хоть точность немного и увеличилась по сравнению с моделью, полученной в прошлой лабораторной работе, однако поведение log-loss не соответствует норме.

Задание 2. Замените один из сверточных слоев на слой, реализующий операцию пулинга (Pooling) с функцией максимума или среднего. Как это повлияло на точность классификатора?

На основе модели созданной в предыдущем задании, заменим второй сверточный слой на слой реализующий операцию пулинга с функцией максимума. Размер окна при этом примем 2x2.

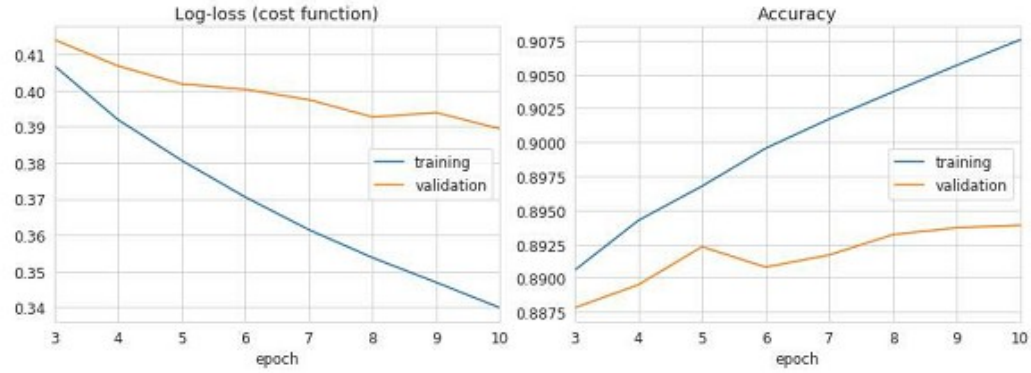
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
                 input_shape=(resolution, resolution, 1)))
model.add(MaxPool2D((2, 2)))
model.add(Flatten())
model.add(Dense(classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
keras2ascii(model)
```

Рисунок 6 – Реализация сверточной нейронной сети с операцией пулинга

```
    OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

        Input   #####     28   28    1
       Conv2D    \|/  -------------------       320     0.6%
         relu   #####     26   26   32
MaxPooling2D   Y max -------------------         0     0.0%
               #####     13   13   32
      Flatten  ||||| -------------------         0     0.0%
               #####         5408
        Dense  XXXXX -------------------     54090    99.4%
      softmax  #####           10
```

Рисунок 7 – Схема сверточной нейронной сети с операцией пулинга

```
model.fit(x_train, y_train,
          epochs=10,
          batch_size=128,
          validation_data=(x_val, y_val),
          callbacks=[PlotLossesKeras()],
          verbose=0)
```




```
Log-loss (cost function):
training   (min:    0.340, max:    0.527, cur:    0.340)
validation (min:    0.389, max:    0.457, cur:    0.389)

Accuracy:
training   (min:    0.858, max:    0.908, cur:    0.908)
validation (min:    0.876, max:    0.894, cur:    0.894)
```

Рисунок 8 – Результаты обучения

Как видно из графиков, точность на валидационных данных ниже, чем на обученных. Максимальная точность на валидационных данных составила 0,894, а минимальный log-loss – 0,389. Обратим внимание, что log-loss имеет уже негативный тренд, в отличие от предыдущей модели. Также заметим, что точность на валидационной выборке снизилась. Добавленная операция пулинга снизила пространственный размер изображения, профильтровала шум, хотя в размере изображения 28x28 это не совсем будет заметно. Данная техника применяется, чтобы сжимать изображение, а последующие сверточные слои должны как раз находить какие-то характерные признаки у

изображений. Если бы после операции пулинга последовал сверточный слой, то, возможно, точность классификатора бы увеличилась.

```
y_pred = model.predict_classes(x_test)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1:', f1_score(y_test, y_pred, average='weighted'))
confusion_matrix(y_test, y_pred)
```

```
Accuracy: 0.8964736842105263
F1: 0.8966734894771725

Out[30]:

array([[1621,   15,   11,   29,    7,   12,   21,   33,   33,   11],
       [  26, 1685,    8,   39,   25,   11,   33,   38,   39,   13],
       [  11,   15, 1776,    9,   23,   16,   67,    8,   29,    8],
       [  32,   24,    4, 1695,    6,    9,   12,   17,   28,   26],
       [  24,   34,   47,   15, 1681,   32,   41,   17,   41,    3],
       [  19,   17,   12,   12,   14, 1744,   30,   26,   42,   14],
       [  14,   22,   29,   21,   11,   14, 1720,   16,   26,   16],
       [  37,   25,    4,   19,    8,   17,   23, 1657,   31,    5],
       [  23,   21,   14,   20,   14,   14,   22,   33, 1720,   73],
       [  20,   13,    1,   29,   10,   18,   15,   21,   80, 1734]])
```

Рисунок 9 – Результаты модели на контрольной выборке

Задание 3. Реализуйте классическую архитектуру сверточных сетей LeNet-5 (http://yann.lecun.com/exdb/lenet/).

Архитектура LeNet-5 выглядит так:

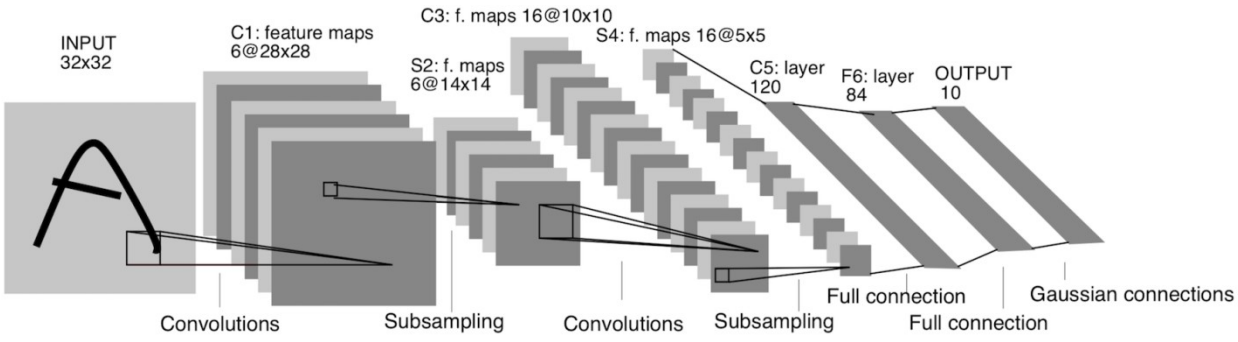


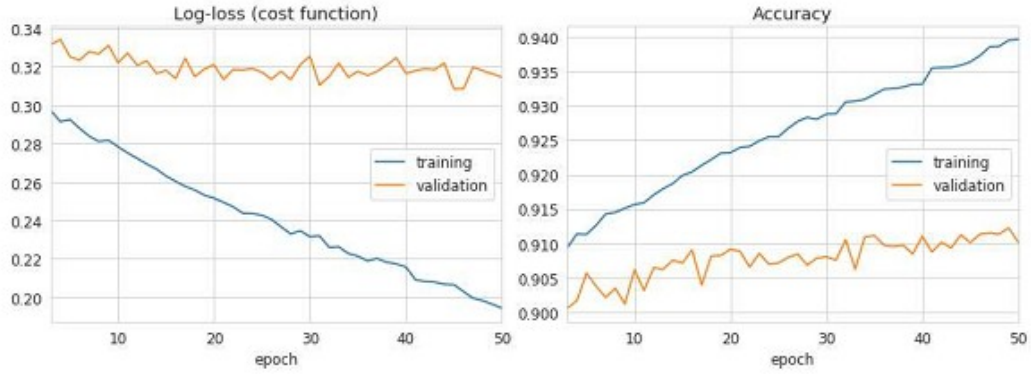
Рисунок 10 – Архитектура сети LeNet-5

```
model = keras.Sequential()

model.add(Conv2D(filters=6, kernel_size=(5, 5), strides=(1, 1), padding='same', activat
ion='tanh', input_shape=(resolution, resolution, 1)))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid'))

model.add(Conv2D(filters=16, kernel_size=(5, 5), strides=(1, 1), padding='valid', activ
ation='tanh'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(filters=120, kernel_size=(5, 5), strides=(1, 1), padding='valid', acti
vation='tanh'))

model.add(Flatten())

model.add(Dense(units=84, activation='tanh'))

model.add(Dense(units=classes, activation = 'softmax'))
```

Рисунок 11 – Реализация сети LeNet-5

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

```
model.fit(x_train, y_train,
          epochs=EPOCHS,
          batch_size=BATCH_SIZE,
          validation_data=(x_val, y_val),
          callbacks=[PlotLossesKeras()],
          verbose=1)
```




```
Log-loss (cost function):
training   (min:    0.194, max:    0.308, cur:    0.194)
validation (min:    0.308, max:    0.337, cur:    0.314)

Accuracy:
training   (min:    0.907, max:    0.940, cur:    0.940)
validation (min:    0.898, max:    0.912, cur:    0.910)
```

Рисунок 12 – Результаты обучения

Как видно из графиков, точность на валидационных данных ниже, чем на тренировочных. Максимальная точность на валидационных данных составила 0,912, а минимальный log-loss – 0,308.

```
y_pred = model.predict_classes(x_test)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1:', f1_score(y_test, y_pred, average='weighted'))
confusion_matrix(y_test, y_pred)
```

```
Accuracy: 0.9113684210526316
F1: 0.9113900645439705

Out[45]:

array([[1667,   20,   11,   17,   10,    9,    7,   31,   15,    6],
       [  27, 1727,   22,   31,   21,    8,   19,   29,   25,    8],
       [  10,   12, 1825,    7,   24,   14,   33,   14,   15,    8],
       [  27,   36,   15, 1694,   10,    8,   10,   11,   22,   20],
       [  16,   31,   35,   13, 1761,   17,   24,   11,   22,    5],
       [  13,   10,   15,   24,   28, 1752,   29,   22,   31,    6],
       [  14,   28,   46,   14,   21,   16, 1701,   16,   21,   12],
       [  30,   16,    9,   13,   13,    8,   15, 1691,   22,    9],
       [  15,   21,   26,   27,   25,   17,   16,   20, 1749,   38],
       [  22,    8,   17,   26,   16,   19,    9,   25,   50, 1749]])
```

Рисунок 13 – Результаты модели на контрольной выборке

Архитектура LeNet-5 увеличила точность модели до 0,911, однако точность и log-loss сильно расходятся, что говорит нам о переобучении на тренировочных данных. Возможно, если добавить сброс нейронов, а также применить регуляризацию, то точность модели бы увеличилась или хотя бы точность и log-loss были бы подобны, что предотвратило бы переобучение.

Задание 4. Сравните максимальные точности моделей, построенных в лабораторных работах 1-3. Как можно объяснить полученные различия?

В 1 лабораторной работе был построен простейший классификатор на основе логистической регрессии, что по сути своей является однослойным персептроном, точность составила 0,83. Во второй работе была реализована полносвязная сеть с 5 слоями, а также с применением методов предотвращающих переобучение: регуляризация и сброс нейронов. Точность на контрольной выборке составила ~0,9. В данной лабораторной работе был реализован классификатор на основе сверточной нейронной сети с 1 сверточным слоем и вторым, реализующим операцию пулинга. Точность такой модели составила ~0,91. Очевидно, что сверточные нейронные сети лучше всего работают с изображениями. Логистическая регрессия показала точность ниже всех, в силу того, она применяется для прогнозирования вероятности возникновения некоторого события по значениям множества признаков. Нейронные сети работают по-другому, поэтому заметно их различение в полученной точности. Сверточные сети в основном применяются с дву- и более мерными данными, а так как наши данные –

изображение, то вполне очевидно, что сверточные сети сработают лучше, при том, что было всего 2 слоя, когда как у полносвязной сети было 5 слоев.

Вывод:

В ходе выполнения лабораторной работы были изучены сверточные слои. Был построена простейшая сверточная нейронная сеть с 2 слоями. С помощью операции пулинга была снижена размерность в данных, и этой операцией было немного подавлено переобучение. Была реализована архитектура LeNet-5, максимальная точность на валидационных данных составила 0,912, а минимальный log-loss – 0,308. А на контрольной выборке точность составила 0,911.