Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:
«Машинное обучение»
По лабораторной работе №3
«Реализация сверточной нейронной сети»

Выполнил: Сенькович Дмитрий Сергеевич
магистрант кафедры информатики
группы №858642

Проверил: Стержанов Максим Валерьевич
доцент, кандидат технических наук

Минск 2019

Оглавление

1 Постановка задачи

Имеется большой и маленький наборы изображений цифр 28x28. Требуется обучить сверточную нейронную сеть на различных объемах данных.

2 Решение задачи

2.1 Чтение данных и формирование выборок данных

Чтение данных и формирование выборок останется практически таким же, как и в прошлой лабораторной работе. Единственное изменение: в этот раз будем использовать более традиционное представление для machine learning target значений - one hot encoding. Также следует немного изменить формат самих данных, приведя их к виду, ожидаемому сверточной нейронной сетью:

```
x_train = np.array(x_train).reshape(len(x_train), 28, 28, 1)

y_train = np_utils.to_categorical(np.array(y_train), len(CLASSES))
```

Теперь x_train и остальные выборки буду представлять из себя mx28x28x1 массивы, где m соответствует размеру выборки, а 1 - количество каналов в изначальном изображении. Здесь мы используем 1, так как изображения представлены в черно-белых оттенках. При работе с RGB изображениями использовалось бы 3 канала.

2.2 Реализация сверточной сети без пулинга

Для начала создадим сверточную сеть с двумя сверточными слоями и одним полносвязным:

```
model = keras.Sequential([

      keras.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),

    keras.layers.Conv2D(32, (3, 3), activation="relu"),

    keras.layers.Flatten(),

    keras.layers.Dense(128, activation="relu"),

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

])

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

```
print('Adam validation accuracy: %s' % train_and_compare_with_validation(200000, 10000, large_dataset_images, model), file=open("results_without_pooling.txt", "a"))

x_test, y_test = build_test_set(small_dataset_images)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print('Adam test accuracy: %s' % test_acc, file=open("results_without_pooling.txt", "a"))
```

Полученную сеть обучим в течение 50 эпох. Полученная точность на валидационной и тестовой выборке: 0.9043 и 0.9610703.

2.3 Реализация сверточной сети со слоем пулинга

Заменим в предыдущей сети второй уровень на пулинг:

```
model = keras.Sequential([

      keras.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),

    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    keras.layers.Flatten(),

    keras.layers.Dense(128, activation="relu"),

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

])

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

print('Adam validation accuracy: %s' % train_and_compare_with_validation(200000, 10000, large_dataset_images, model), file=open("results_with_pooling.txt", "a"))

x_test, y_test = build_test_set(small_dataset_images)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print('Adam test accuracy: %s' % test_acc, file=open("results_with_pooling.txt", "a"))
```

Получим точность 0.8962 и 0.9590964 на валидационной и тестовой выборках соответственно за 50 эпох обучения.

2.4 Реализация сверточной сети с архитектурой LeNet-5

Реализуем классическую сверточную сеть с архитектурой LeNet-5:

```
model = keras.Sequential([

    keras.layers.Conv2D(6, (3, 3), activation="relu", input_shape=(28, 28, 1)),

    keras.layers.AveragePooling2D(),

    keras.layers.Conv2D(16, (3, 3), activation="relu"),

    keras.layers.AveragePooling2D(),

    keras.layers.Flatten(),

    keras.layers.Dense(120, activation="relu"),

    keras.layers.Dense(84, activation="relu"),

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

])

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

print('Adam           validation           accuracy:           %s'           %
train_and_compare_with_validation(200000, 10000, large_dataset_images, model),
file=open("results_lenet5.txt", "a"))

x_test, y_test = build_test_set(small_dataset_images)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print('Adam test accuracy: %s' % test_acc, file=open("results_lenet5.txt",
"a"))
```

Заметим, что мы используем значительно меньшее количество фильтров. Данная сеть даст точность 0.8954 и 0.9549841 на валидационной и тестовой выборках за 50 эпох.

2.5 Более долгое обучения

Перед сравнением результатов в этой лабораторной работе и предыдущих двух обучим три модели из этой лабораторной работы в течение 200 эпох каждую. Получим следующие результаты:

Модель	Точность на валидационной выборке	Точность на тестовой выборке
Без пулинга	0.9101	0.9601
С пулингом	0.8982	0.9572
LeNet-5	0.8972	0.9521

Получили чуть меньшую точность, что объясняется, что после некоторого количества эпох нужно было использовать learning rate decay, потому что сеть осциллировала около оптимума.

2.6 Сравнение логистической регрессии, глубокой нейронной сети и сверточной сети

В результате обучения трех моделей получили следующие результаты: глубокая нейронная сеть дала значительный прирост в точности классификации по сравнению с логистической регрессией, а сверточные сети, даже самые простые, оказались немного лучше глубокой нейронной сети.

Первый факт объясняется тем, что, по сути, глубокая нейронная сети представляет из себя более сложную комбинацию из нейронов, каждая из который в некотором роде является логистической регрессией. Это позволяет нейронной сети изучать более сложные зависимости.

Второй факт, хоть и не сильно заметен в данных выбранных простейших конфигурациях сетей, объясняется тем, что в изображениях очень много фич. Полносвязные слои в таком случае избыточны, так как используют намного больше параметров и связей. Сверточные сети оптимизируют набор параметров, нужный для обучения, тем самым давая лучший результат за меньшее время. Для изображений это работает хорошо, потому что,

интуитивно, более близкие пиксели более связаны между собой и можно использовать гораздо меньше связей.