МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Искусственные нейронные сети»
Тема: Распознавание объектов на фотографиях

Студентка гр. 8382 _____ Кулачкова М.К.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы

Распознавание объектов на фотографиях (Object Recognition in Photographs) CIFAR-10 (классификация небольших изображений по десяти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик).

Задачи

- Ознакомиться со сверточными нейронными сетями
- Изучить построение модели в Keras в функциональном виде
- Изучить работу слоя разреживания (Dropout)

Требования

1. Построить и обучить сверточную нейронную сеть
2. Исследовать работу сети без слоя Dropout
3. Исследовать работу сети при разных размерах ядра свертки

Выполнение работы

Для распознавания объектов на изображениях целесообразно использовать сверточную нейронную сеть. Сверточная сеть способна находить на изображении шаблоны независимо от их местоположения, а также изучать пространственные иерархии шаблонов.

В работе будет использоваться модель, состоящая из четырех слоев свертки, двух слоев субдискретизации и двух полносвязных слоев. Сверточные слои выделяют наиболее полезные признаки в изображении и для каждого такого признака формируют карту признаков. Слой субдискретизации (пулинга) уменьшает разрешение изображения, тем самым сокращая количество параметров модели. Полносвязные слои осуществляют непосредственно классификацию.

Зададим начальные параметры модели.

```
num_train, depth, height, width = X_train.shape
kernel_size = 3
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.25
drop_prob_2 = 0.5
hidden_size = 512
```

Параметры `depth`, `height` и `width` необходимы для задания размеров входного слоя и определяются количеством каналов и размерами изображений тренировочного множества. Параметр `kernel_size` задает размер ядра свертки для всех сверточных слоев, здесь будет использоваться ядро 3×3. Параметр `pool_size` определяет размер окна субдискретизации, здесь используется окно 2×2, что означает, что после слоя субдискретизации ширина и высота изображения уменьшатся в два раза. Параметры `conv_depth_1` и `conv_depth_2` определяют число каналов (признаков), которые вычисляют слои свертки. Здесь первые два сверточных слоя будут вычислять 32 карты признаков, третий и четвертый – 64 карты признаков. Параметры `drop_prob_1` и `drop_prob_2` задают вероятности «выброса» нейронов из модели для слоев дропаута. Для слоев дропаута, следующих за слоями пулинга, это значение равно 0.25, для слоя, следующего за первым полносвязным слоем, – 0.5.

Построим модель. Выходной полносвязный слой использует функцию активации softmax для осуществления классификации изображений. Для регуляризации модели используются три слоя Dropout. полносвязного слоя с вероятностью дропаута 0.5.

```
inp = Input(shape=(depth, height, width))

conv_1 = Convolution2D(conv_depth_1, kernel_size,
padding='same', activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, kernel_size,
padding='same', activation='relu')(conv_1)
pool_1 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_2)
drop_1 = Dropout(drop_prob_1)(pool_1)
```

```
conv_3 = Convolution2D(conv_depth_2, kernel_size,
padding='same', activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, kernel_size,
padding='same', activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_4)
drop_2 = Dropout(drop_prob_1)(pool_2)

flat = Flatten()(drop_2)
hidden = Dense(hidden_size, activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(hidden)
out = Dense(num_classes, activation='softmax')(drop_3)

model = Model(inputs=inp, outputs=out)
```

Обучим полученную модель в течение 200 эпох батчами по 32 образца, затем оценим модель на тестовом множестве:

```
Epoch 200/200
     1407/1407 [==============================] - 8s 5ms/step
- loss: 0.1982 - accuracy: 0.9384 - val_loss: 0.9513 -
val_accuracy: 0.8054


313/313 [==============================] - 1s 3ms/step - loss:
455.0558 - accuracy: 0.4599
```

Как видно, полученная модель достигает достаточно хорошей точности на валидационном множестве, но обладает низкой точностью на тестовом множестве – всего 45.99 %.

На рисунках 1 и 2 изображены графики ошибок и точности на тренировочном и валидационном множествах в процессе обучения.

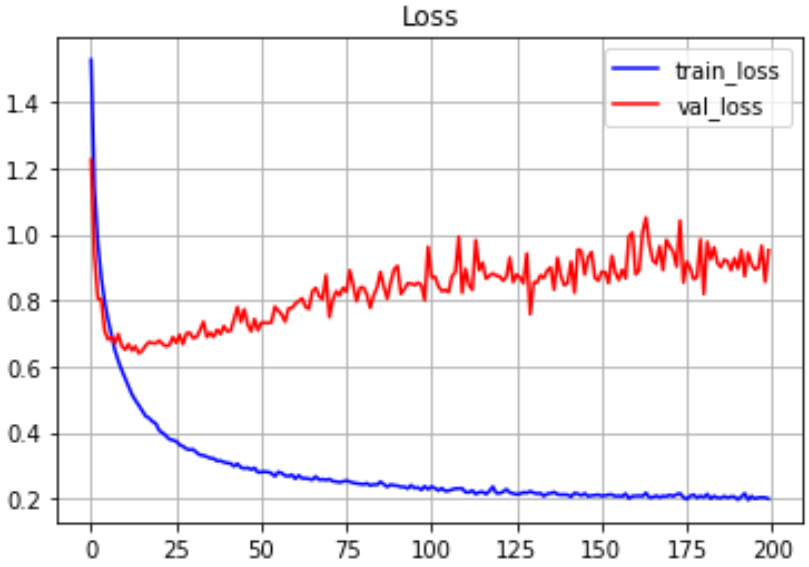

Рисунок 1

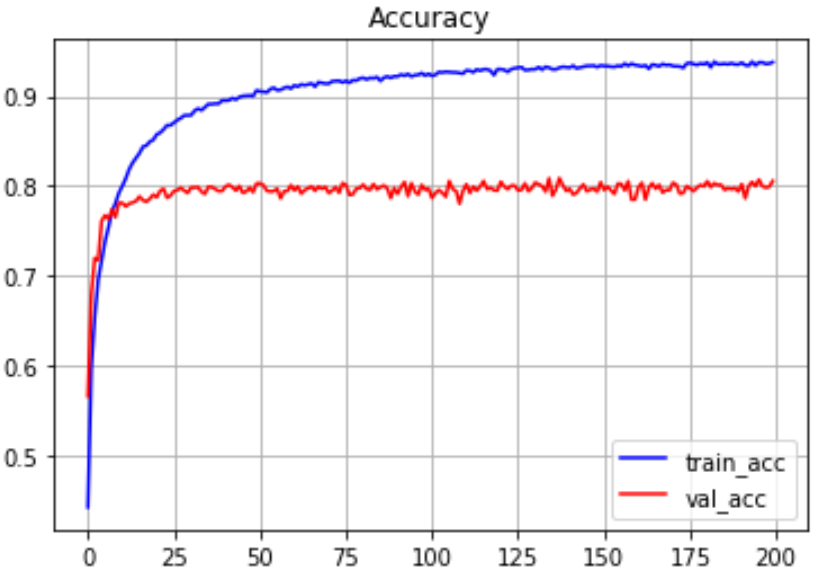

Рисунок 2

По графикам заметно, что произошло переобучение модели. Также видно, что и потери, и точность примерно после 90-ой эпохи практически не изменяются, поэтому можно сократить количество эпох обучения.

Обучим исходную модель в течение 90 эпох и оценим ее на тестовом множестве.

```
Epoch 90/90
1407/1407 [==============================] - 7s 5ms/step -
loss: 0.2349 - accuracy: 0.9236 - val_loss: 0.7946 -
val_accuracy: 0.8000
```

```
313/313 [==============================] - 1s 3ms/step -
loss: 337.0519 - accuracy: 0.4730
```

Точность модели практически не изменилась. Построим графики ошибок и точности в процессе обучения:

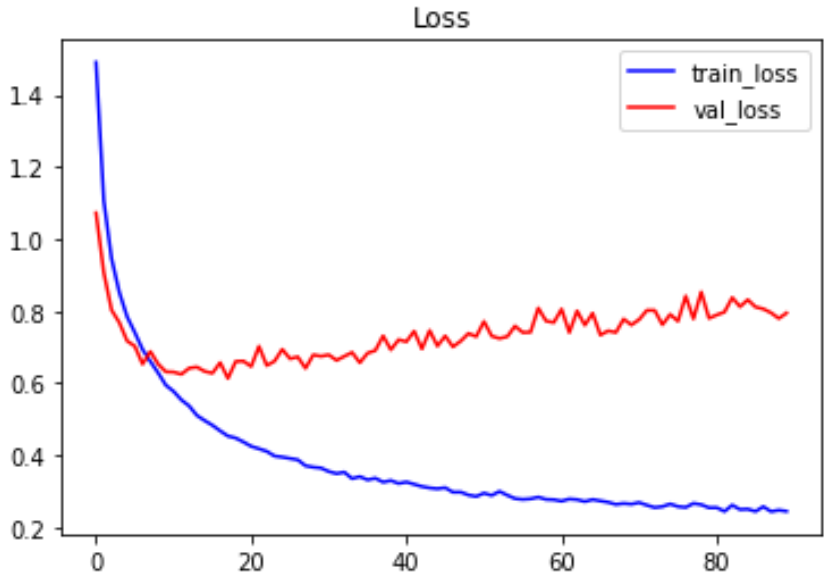

Рисунок 3

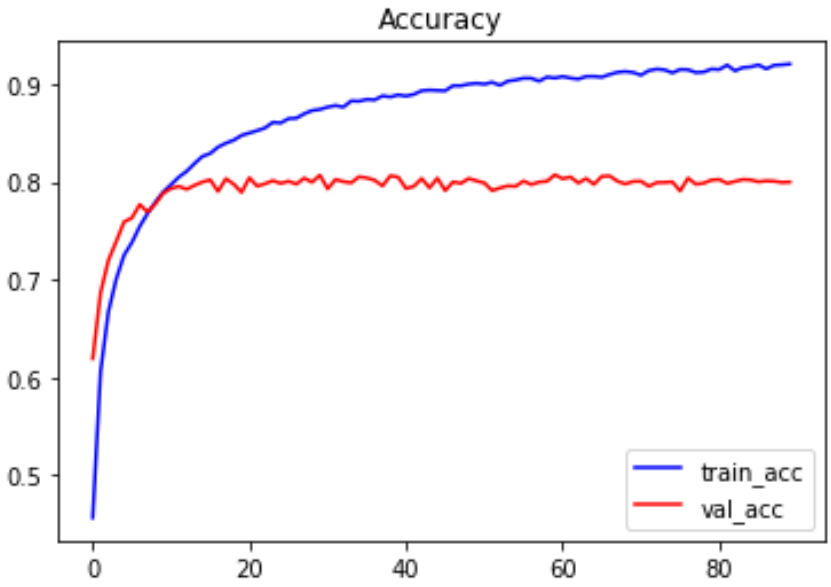

Рисунок 4

Dropout – это один из методов регуляризации нейронных сетей. Когда модель обладает большим числом параметров, отдельные группы нейронов могут придавать избыточное значение распознаваемым ими признакам, что приводит к ухудшению обобщающей способности модели – к переобучению.

Метод заключается в том, что для каждого нейрона, кроме нейронов на выходном слое, устанавливается вероятность p того, что этот нейрон будет выброшен из сети при обучении на каждом из тренировочных образцов. Таким образом, каждый из образцов будет обучать немного другую модель. При тестировании модели никакие нейроны не выбрасывается, но вывод каждого умножается на значение p. Так получается усредненная модель.

Для того, чтобы изучить влияние слоя Dropout на процесс обучения модели, уберем все такие слои из построенной ранее модели.

```
inp = Input(shape=(depth, height, width))

conv_1 = Convolution2D(conv_depth_1, kernel_size,
padding='same', activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, kernel_size,
padding='same', activation='relu')(conv_1)
pool_1 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_2)

conv_3 = Convolution2D(conv_depth_2, kernel_size,
padding='same', activation='relu')(pool_1)
conv_4 = Convolution2D(conv_depth_2, kernel_size,
padding='same', activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size,
pool_size))(conv_4)

flat = Flatten()(pool_2)
hidden = Dense(hidden_size, activation='relu')(flat)
out = Dense(num_classes, activation='softmax')(hidden)

model = Model(inputs=inp, outputs=out)
```

Обучим полученную модель и оценим ее на тестовом множестве:

```
Epoch 90/90
     1407/1407 [==============================] - 7s 5ms/step
- loss: 0.0628 - accuracy: 0.9901 - val_loss: 6.2186 -
val_accuracy: 0.7226

313/313 [==============================] - 1s 3ms/step -
loss: 2013.4464 - accuracy: 0.5817
```

Точность модели на валидационном множестве оказалась хуже, чем точность модели со слоями дропаута, однако точность на тестовом множестве несколько повысилась, однако потери в то же время значительно возросли.

На рисунках 5 и 6 изображены графики ошибок и точности на валидационном множестве моделей со слоями дропаута и без.

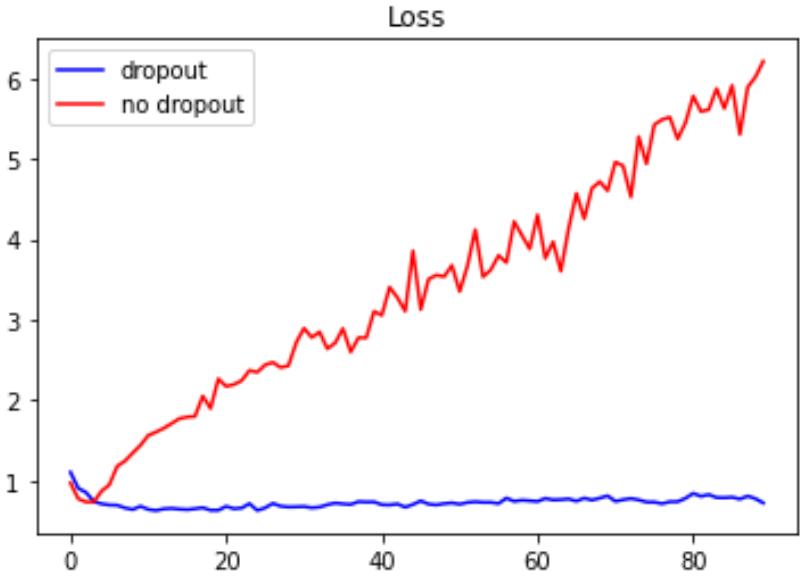

Рисунок 5

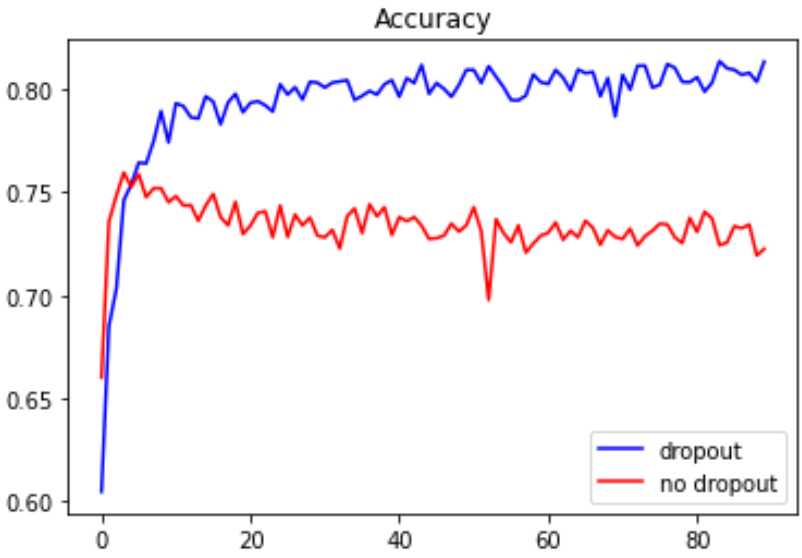

Рисунок 6

Потери модели без дропаута намного выше, а точность – ниже, что говорит о том, что дропаут благоприятно влияет на обучение модели.

Операция свертки заключается в том, что изображение делится на окна размера $m \times m$ и для каждого такого окна вычисляется сумма произведения его элементов на элементы матрицы размера $m \times m$ – ядра свертки. Каждому признаку, выделяемому слоем, соответствует свое ядро свертки.

Исследуем работу сети при разных размерах ядра свертки m. Вернем в модель слои дропаута и изменим размер ядра, задаваемый параметром `kernel_size`, на 5. Обучим полученную модель и оценим ее точность на тестовом множестве:

```
Epoch 90/90
1407/1407 [==============================] - 10s 7ms/step -
loss: 0.3976 - accuracy: 0.8718 - val_loss: 0.9736 -
val_accuracy: 0.7542

313/313 [==============================] - 1s 4ms/step -
loss: 384.3547 - accuracy: 0.4435
```

Также построим, обучим и оценим модель с ядром размера 7:

```
Epoch 90/90
1407/1407 [==============================] - 12s 8ms/step -
loss: 0.5125 - accuracy: 0.8325 - val_loss: 1.0644 -
val_accuracy: 0.7262

313/313 [==============================] - 1s 4ms/step -
loss: 462.3214 - accuracy: 0.5028
```

Построим графики ошибок и точности на валидационном множестве для моделей с тремя разными размерами ядра (см. рис. 7 и 8).

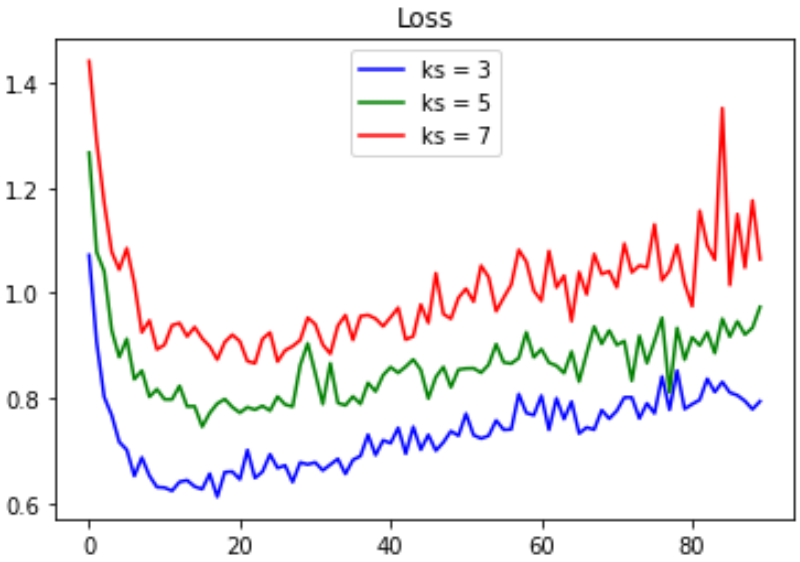

Рисунок 7

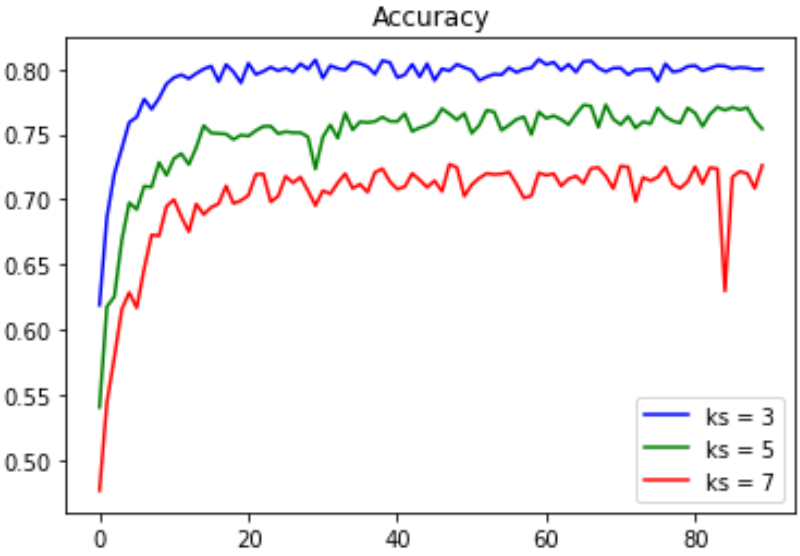

Рисунок 8

На графиках видно, что, чем меньше размер ядра, тем выше точность модели на валидационном множестве и тем меньше потери.

Выводы

В ходе выполнения лабораторной работы была создана нейронная сеть со сверточной архитектурой, осуществляющая классификацию изображений. Изучен принцип работы сверточных нейронных сетей, исследовано влияние слоя Dropout на процесс обучения, рассмотрена работа нейронной сети при различных размерах ядра свертки.