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. 7382 _____ Головина Е.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Реализовать 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.

- Построить и обучить сверточную нейронную сеть
- Исследовать работу сеть без слоя Dropout
- Исследовать работу сети при разных размерах ядра свертки

Ход работы.

1. Ознакомиться со сверточными нейронными сетями

Сверточные сети в первую очередь используются для таких данных, которые представляют собой многомерный массив (тензор), порядок расположения данных в которых важен и каждый единичный элемент описан несколькими свойствами.

Рассмотрим сверточные сети для классификации изображений. Предполагается, что пиксели, находящиеся близко друг к другу, теснее «взаимодействуют» при формировании интересующего нас признака, чем пиксели, расположенные далеко друг от друга. Также не имеет значения в каком конкретно месте изображения расположен интересующий нас объект.

Центральное понятие в сверточных сетях – это оператор свертки (ядро свертки) – матрица $h \times w$ построенная таким образом, что графически кодирует

какой-то признак. При умножении двумерного изображения (I) на нее вычисляется свернутое изображение. Умножение представляет собой скалярное умножение входящих в матрицу изображения матриц размерности ядра на ядро.

Каждый слой состоит из определенного количества ядер K с аддитивными составляющими смещения b для каждого ядра и вычисляет свертку выходного изображения предыдущего слоя с помощью каждого из ядер, каждый раз прибавляя составляющую смещения. Входной поток состоит из d каналов (RGB – для изображения). Для одного канала применяется следующая формула:

$$conv(I,K)_{x,y} = \sigma(b + \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} K_{ijk} \times I_{x+i-1,y+j-1,k})$$

Таким образом происходит сложение и масштабирование входных данных. Многослойный перцептрон (MLP) также смог бы справиться с данной задачей, но за гораздо большее количество времени.

Гиперпараметры, выбираемые до начала обучения:

- глубина (depth) – сколько ядер и коэффициентов смещения будет задействовано в одном слое;

- высота (height) и ширина (width) каждого ядра;

- шаг (stride) – на сколько смещается ядро на каждом шаге при вычислении следующего пикселя результирующего изображения;

- отступ (padding) – для дополнении нулями по краям для сохранения размерности изображения.

Для субдискретизации изображения используется слой подвыборки (слой субдискретизации, downsampling, pooling layer), который получает на вход маленькие фрагменты изображения (2×2) и объединяет каждый фрагмент в одно значение. Это достигается, например, выбором максимального значения.

Обычную архитектуру для распределения изображений по k классам можно разделить на две части: цепочка чередующихся слоев свертки/подвыборки (conv → pool) и несколько полносвязных слоев (принимающих пиксель как независимое значение) с слоем softmax в качестве

завершающего. После каждого сверточного или полносвязного слоя используется функция активации ReLU.

Один проход conv → pool сокращает длину и ширину определенного канала, но увеличивает его значение (глубину).

Так как у изображения может быть много лишних данных («шум») – есть прием регуляризации, помогающий нейронам не тратить время на изучение «шума». Данный прием называется dropout. Например, dropout с параметром *p* за одну итерацию обучения проходит по всем нейронам определенного слоя и с вероятностью *p* исключает их из сети на время итерации. Это заставит сеть обрабатывать ошибки и не полагаться на существование конкретного нейрона, а полагаться на «единое мнение» нейронов внутри слоя.

2. Изучить построение модели в Keras в функциональном виде

Модули необходимые для построения сети:

```
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Convolution2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.utils import np_utils
import numpy as np
```

Задание гиперпараметров:

- `batch_size` – количество обучающих образцов, обрабатываемых одновременно за одну итерацию алгоритма градиентного спуска;
- `num_epochs` – количество итераций обучающего алгоритма по всему обучающему множеству;
- `kernel_size` – размер ядра в сверточных слоях;
- `pool_size` – размер подвыборки в слоях подвыборки;
- `conv_depth` – количество ядер в сверточных слоях;
- `drop_prob` (dropout probability) – вероятность отклюения нейрона;
- `hidden_size` – количество нейронов в полносвязном слое MLP.

```
batch_size = 32
num_epochs = 200
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
```

Загрузка данных:

```
#загрузка данных и получение их параметров
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
num_train, depth, height, width = X_train.shape
num_test = X_test.shape[0]
num_classes = np.unique(y_train).shape[0]
#конвертация данных
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
#нормализация данных
X_train /= np.max(X_train)
X_test /= np.max(X_train)
#перекодировка значений в метки
Y_train = np_utils.to_categorical(y_train, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)
```

Сеть будет состоять из двух слоев Convolution_2D и слоев MaxPooling2D после второй и четвертой сверток. После первого слоя подвыборки удваивается количество ядер. После этого выходное изображение слоя подвыборки трансформируется в одномерный вектор (слоем Flatten) и проходит два полносвязных слоя (Dense). На всех слоях, кроме выходного полносвязного слоя, используется функция активации ReLU, последний же слой использует softmax.

Для регуляризации модели после каждого слоя подвыборки и первого полносвязного слоя применяется слой Dropout.

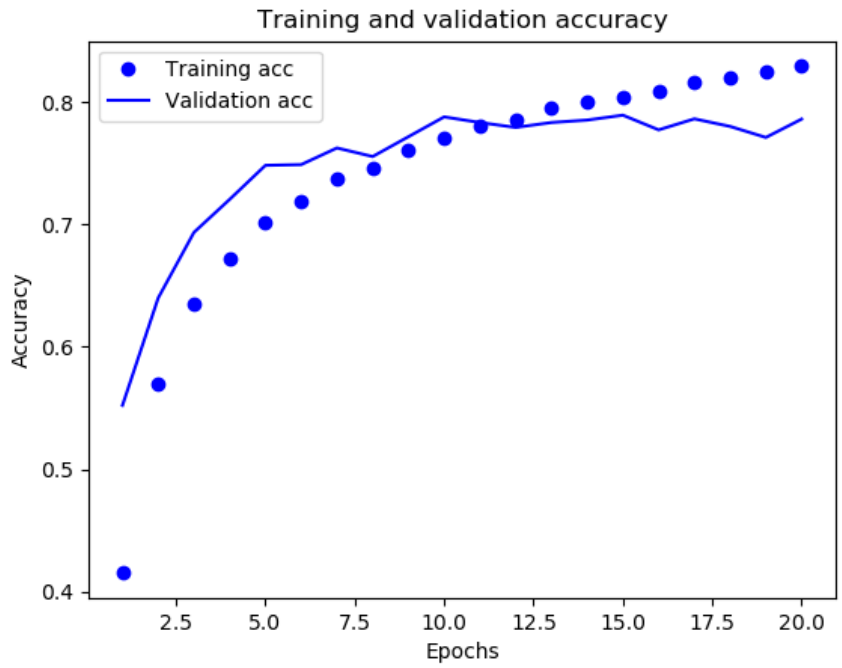
```
#создание слоев
inp = Input(shape=(depth, height, width))


# Conv [32] -> Conv [32] -> Pool (with dropout on the pooling layer)
conv_1 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='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 [64] -> Conv [64] -> Pool (with dropout on the pooling layer)
conv_3 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_4)
drop_2 = Dropout(drop_prob_1)(pool_2)


# Now flatten to 1D, apply Dense -> ReLU (with dropout) -> softmax
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)


#построение модели
model = Model(input=inp, output=out)
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])


#обучение
model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=num_epochs,
          verbose=1, validation_split=0.1)
```

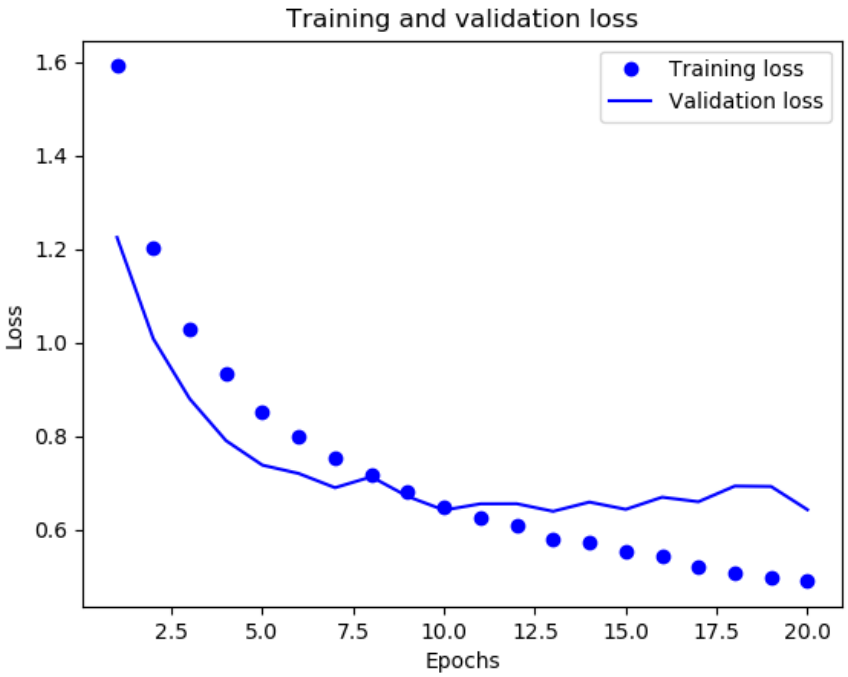
3. Результаты работы

Запустим обучение нейронной сети при стандартной конфигурации (заменив количество эпох на 20 вместо 200 для экономии времени и размер пакета(batch_size) с 32 до 100).

Результат: Accuracy: 79%

Графики точности и потерь:

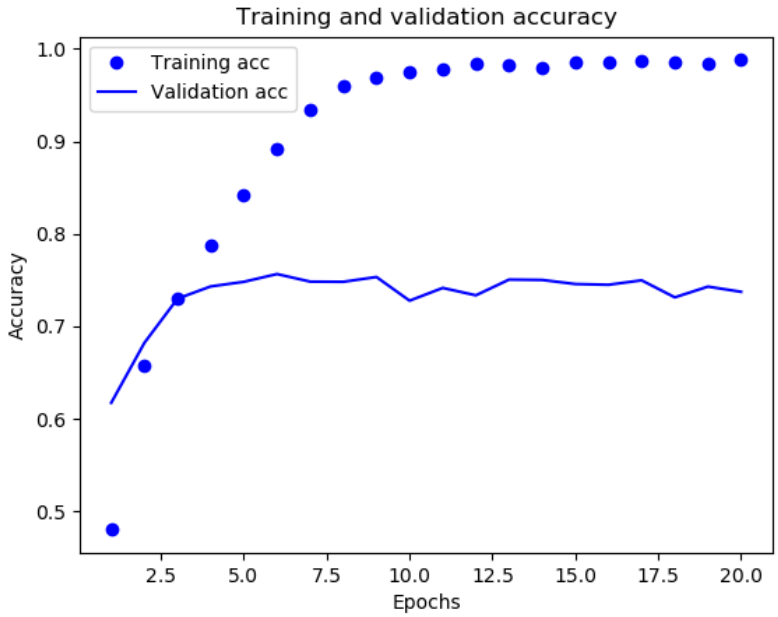


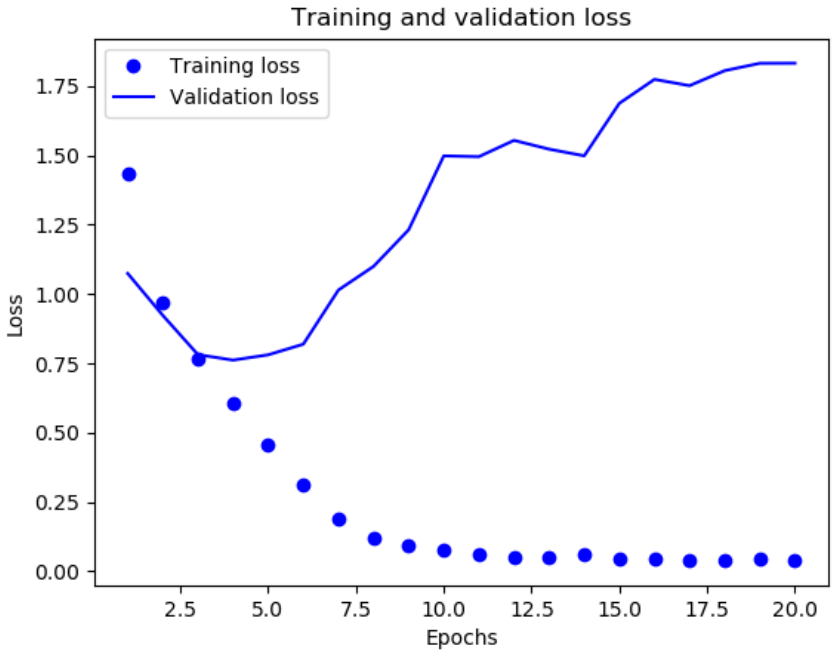


Уберем слои dropout и запустим обучение. Ожидается, что без данных слоев переобучение наступит гораздо раньше, т.к. сеть будет больше обращать внимания на «шум».

Результат: Accuracy: 74%

Графики точности и потерь:





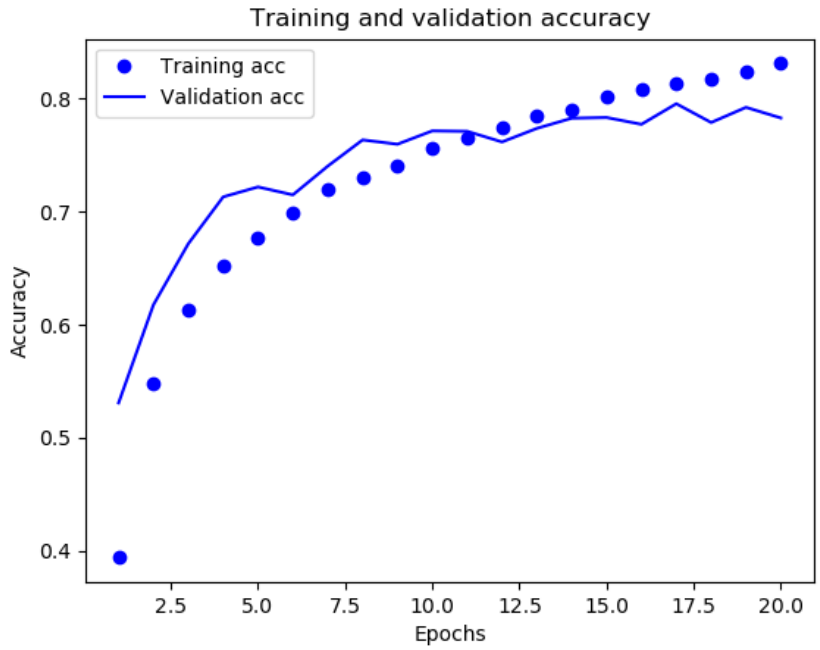
Действительно – переобучение произошло уже примерно на 3 эпохе. Причем к 20 эпохе сеть уже очень сильно переобучилась.

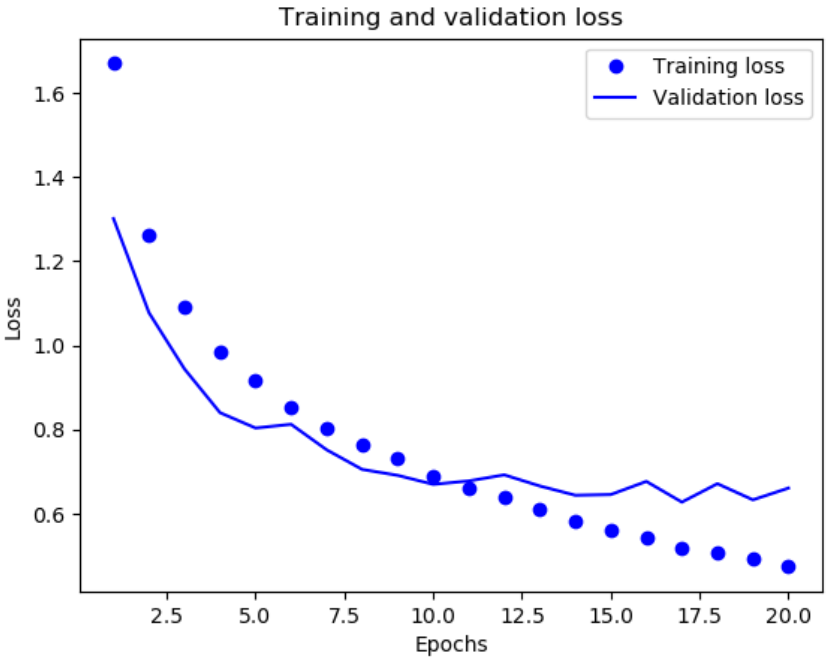
Теперь попробуем изменить размерность ядра с 3x3 на 2x2 и 4x4. Ожидается, что с ядром 2x2 уменьшится точность, а с 4x4 наоборот.

2x2

Accuracy: 78%

Графики точности и потерь:

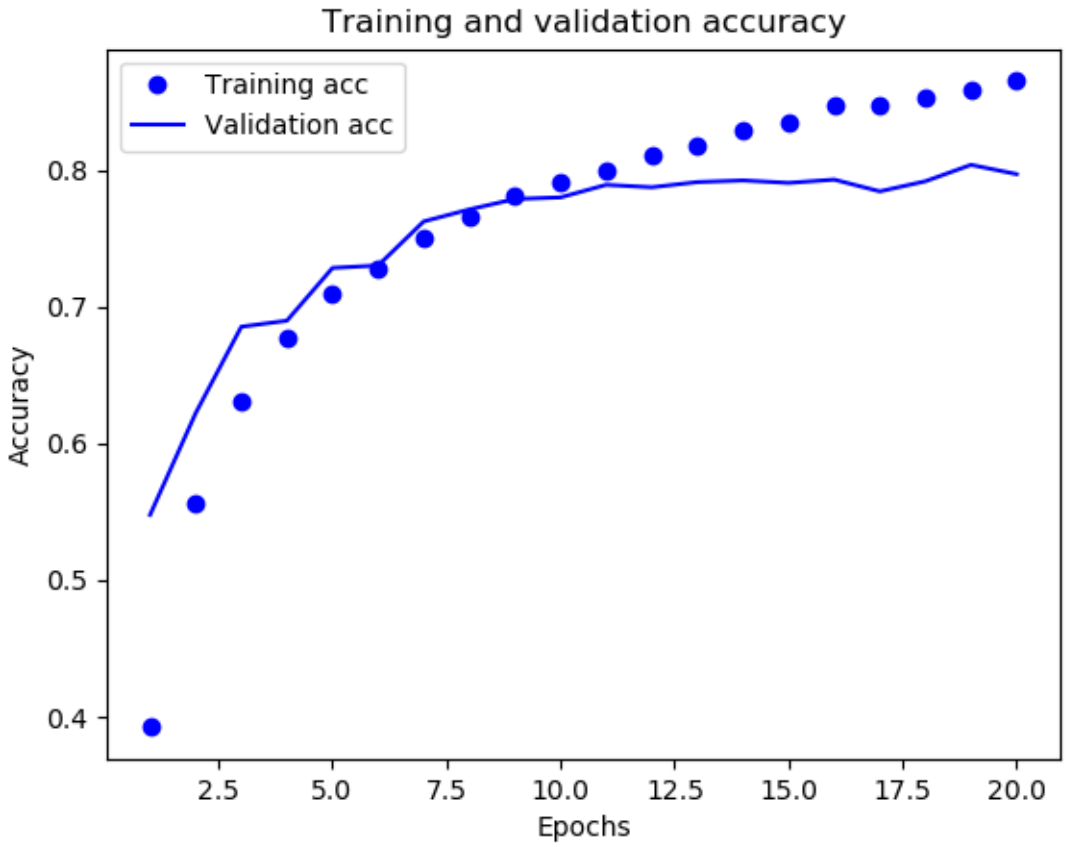


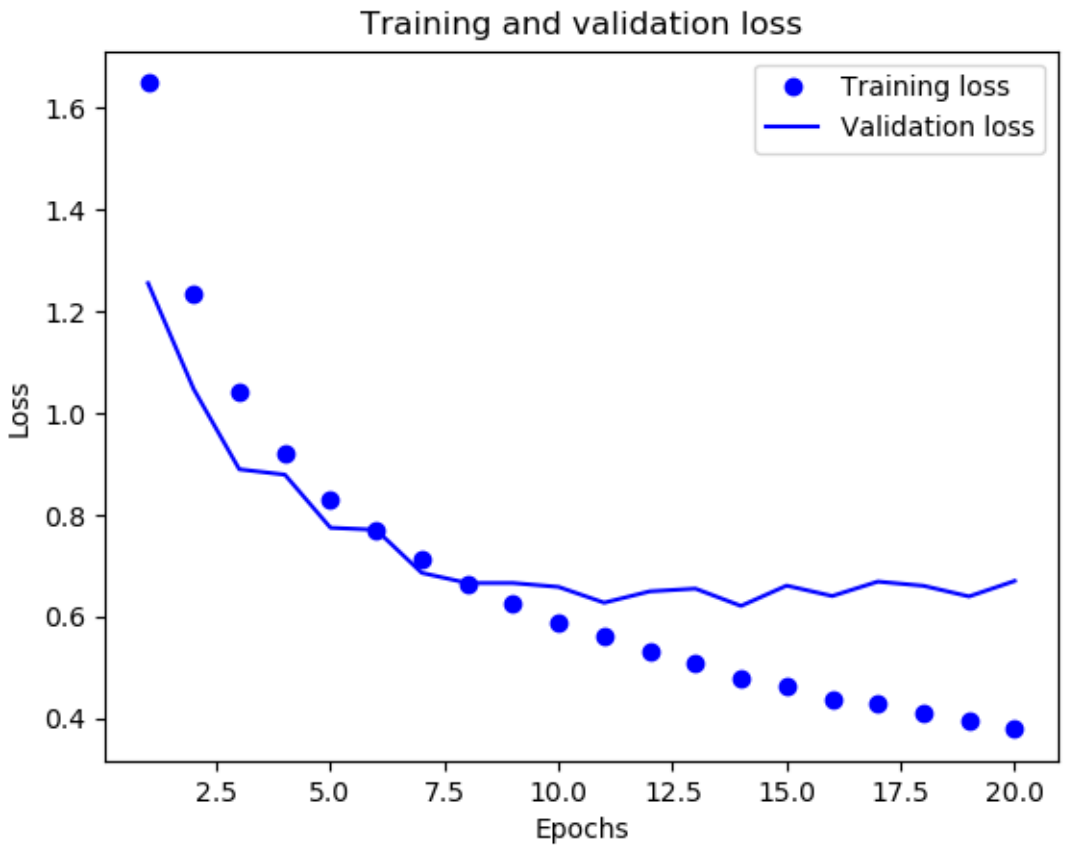


4x4

Результат: 80%

Графики точности и потерь:





Предположения подтвердились – действительно, при меньшей размерности – точность получается хуже. Но переобучение быстрее наступает при большей размерности – исходя из чего можно сделать вывод, что дальнейшее увеличение размерности ядра приведет к уменьшению точности и ухудшению результата.

Вывод.

В ходе выполнения работы было реализовано распознавание объектов на фотографиях CIFAR-10, а именно классификация небольших изображений по десяти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик. Для этого создали искусственную нейронную сеть с помощью библиотеки keras в python, протестировали получившуюся модель, затем, проведя ряд экспериментов, проверили влияние слоев dropout и размерности ядра свертки на результат.