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

Студент гр. 8383 _____ Бессуднов Г. И.

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).

Основные теоретические положения

В лабораторной 4 архитектуру “многослойный перцептрон” (MLP) применили к MNIST. Но все же полносвязный перцептрон обычно не выбирают для задач, связанных с распознаванием изображений — в этом случае намного чаще пользуются преимуществами сверточных нейронных сетей (Convolutional Neural Networks, CNN).

Посмотрим, что происходит с количеством параметров (весов) в модели MLP, когда ей на вход поступают необработанные данные. Например, CIFAR-10 содержит 32 x 32 x 3 пикселей, и если мы будем считать каждый канал каждого пикселя независимым входным параметром для MLP, каждый нейрон в первом скрытом слое добавляет к модели около 3000 новых параметров. И с ростом размера изображений ситуация быстро выходит из-под контроля, причем происходит это намного раньше, чем изображения достигают того размера, с которыми обычно работают пользователи реальных приложений.

Одно из популярных решений — понижать разрешение изображений до той степени, когда MLP становится применим. Тем не менее, когда мы просто понижаем разрешение, мы рискуем потерять большое количество информации, и было бы здорово, если бы можно было осуществлять полезную первичную обработку информации еще до применения понижения качества, не вызывая при этом взрывного роста количества параметров модели.

Существует весьма эффективный способ решения этой задачи, который обращает в нашу пользу саму структуру изображения: предполагается, что пиксели, находящиеся близко друг к другу, теснее “взаимодействуют” при формировании интересующего нас признака, чем пиксели, расположенные в

противоположных углах. Кроме того, если в процессе классификации изображения небольшая черта считается очень важной, не будет иметь значения, на каком участке изображения эта черта обнаружена.

Введем понятие оператора свертки. Имея двумерное изображение I и небольшую матрицу K размерности h\times w (так называемое ядро свертки), построенная таким образом, что графически кодирует какой-либо признак, мы вычисляем свернутое изображение I * K, накладывая ядро на изображение всеми возможными способами и записывая сумму произведений элементов исходного изображения и ядра.

Заметим также, что оператор свертки вовсе не ограничен двухмерными данными: большинство фреймворков глубокого обучения (включая Keras) предоставляют слои для одномерной или трехмерной свертки прямо “из коробки”.

Стоит также отметить, что хотя сверточный слой сокращает количество параметров по сравнению с полносвязным слоем, он использует больше гиперпараметров — параметров, выбираемых до начала обучения. В частности, выбираются следующие гиперпараметры:

- 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. Обычно его принимают равным 1, и чем больше его значение, тем меньше размер выходного изображения;
- Отступ (padding): заметим, что свертка любым ядром размерности более, чем 1x1 уменьшит размер выходного изображения. Так как в общем случае желательно сохранять размер исходного изображения, рисунок дополняется нулями по краям.

Операции свертки — не единственные операции в CNN (хотя существуют многообещающие исследования на тему “чисто-сверхточных” сетей); они чаще применяются для выделения наиболее полезных признаков перед субдискретизацией (downsampling) и последующей обработкой с помощью MLP.

Популярный способ субдискретизации изображения — слой подвыборки (также называемый слоем субдискретизации, по-английски downsampling или pooling layer), который получает на вход маленькие отдельные фрагменты

изображения (обычно 2x2) и объединяет каждый фрагмент в одно значение. Существует несколько возможных способов агрегации, наиболее часто из четырех пикселей выбирается максимальный.

Обычную архитектуру CNN для распределения изображений по k классам можно разделить на две части: цепочка чередующихся слоев свертки/подвыборки Conv -> Pool (иногда с несколькими слоями свертки подряд) и несколько полносвязных слоев (принимающих каждый пиксель как независимое значение) с слоем softmax в качестве завершающего. Я не говорю здесь о функциях активации, чтобы наша схема стала проще, но не забывайте, что обычно после каждого сверточного или полносвязного слоя ко всем выходным значениям применяется функция активации, например, ReLU.

Один проход Conv -> Pool влияет на изображение следующим образом: он сокращает длину и ширину определенного канала, но увеличивает его значение (глубину).

Softmax и перекрестная энтропия более подробно рассмотрены на предыдущем уроке. Напомним, что функция softmax превращает вектор действительных чисел в вектор вероятностей (неотрицательные действительные числа, не превышающие 1). В нашем контексте выходные значения являются вероятностями попадания изображения в определённый класс. Минимизация потерь перекрестной энтропии обеспечивает уверенность в определении принадлежности изображения определенному классу, не принимая во внимание вероятность остальных классов, таким образом, для вероятностных задач softmax предпочтительней, чем, например, метод квадратичной ошибки.

У глубоких сверточных нейронных сетей масса разнообразных параметров, особенно это касается полносвязных слоев. Переобучение может проявить себя в следующей форме: если у нас недостаточно обучающих примеров, маленькая группа нейронов может стать ответственной за большинство вычислений, а остальные нейроны станут избыточны; или наоборот, некоторые нейроны могут нанести ущерб производительности, при этом другие нейроны из их слоя не будут заниматься ничем, кроме исправления их ошибок.

Чтобы помочь нашей сети не утратить способности к обобщению в этих обстоятельствах, мы вводим приемы регуляризации: вместо сокращения количества параметров, мы накладываем ограничения на параметры модели во время обучения, не позволяя нейронам изучать шум обучающих данных. Здесь я опишу прием dropout, который сначала может показаться “черной магией”, но на деле помогает исключить ситуации, описанные выше. В частности, dropout с параметром p за одну итерацию обучения проходит по всем нейронам определенного слоя и с вероятностью p полностью исключает их из сети на

время итерации. Это заставит сеть обрабатывать ошибки и не полагаться на существование определенного нейрона (или группы нейронов), а полагаться на “единое мнение” (consensus) нейронов внутри одного слоя. Это довольно простой метод, который эффективно борется с проблемой переобучения сам, без необходимости вводить другие регуляризаторы.

Выполнение работы

По теоретическим сведениям была построена модель и проведена ее тренировка. Код программы представлен в Приложении А. Стандартные настройки показали следующие результаты:

- Тестовая точность – 0.5554
- Точность тренировки – 0.8075
- Точность валидации – 0.7918

На рис. 1 показан график точности и потерь. Количество эпох было выбрано 10 ввиду ограничения вычислительных мощностей.

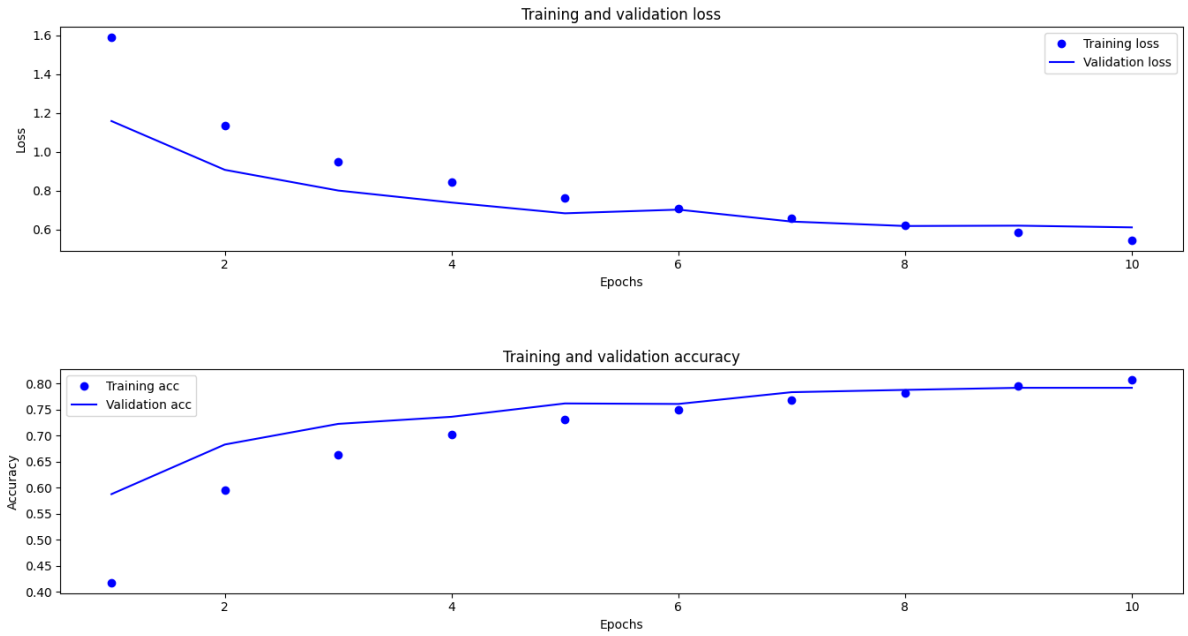

Рисунок 1- Результаты стандартной модели

Далее были убраны слои Dropout и проведена тренировка 2-ой модели. На рис. 2 показан график точности и потерь для этой модели, получившиеся результаты были следующими:

- Тестовая точность – 0. 6321
- Точность тренировки – 0.9035
- Точность валидации – 0.7830

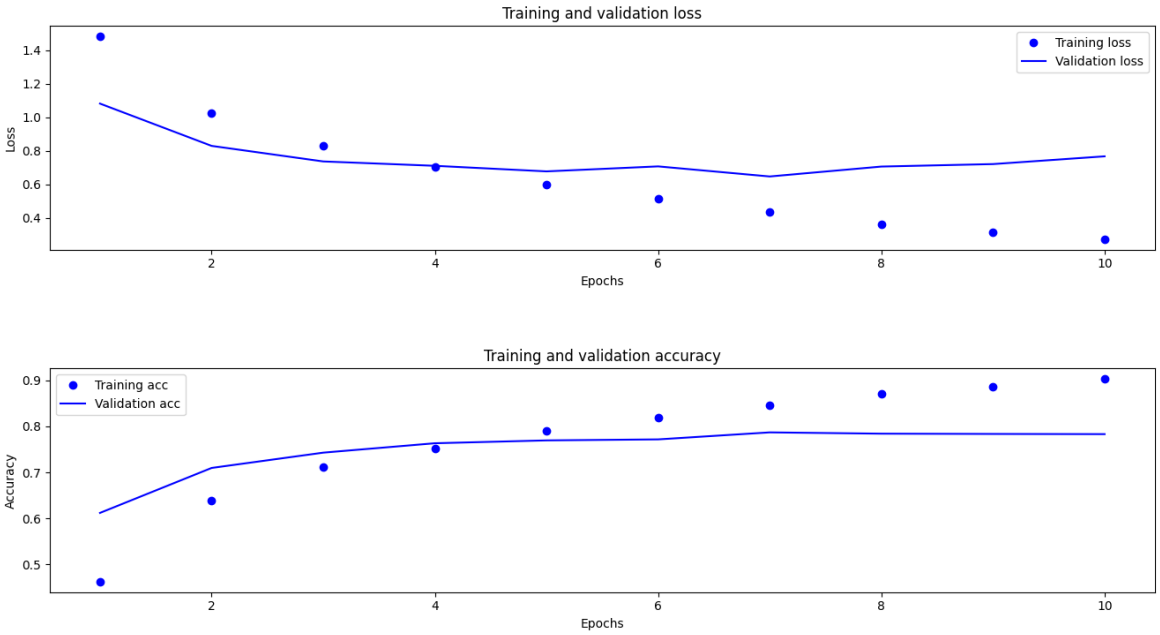

Рисунок 2 - Результаты 2-ой модели

Хотя из графика видно, что точность тестовых данных растет, но точность валидации с 7-ой эпохи почти перестает меняться. На большем количество эпох это бы скорее всего привело к переобучению, так как модель концентрировалась бы на каком-то одном признаке.

Далее была снова взята за основу стандартная модель и уменьшена размерность ядра свертки до 2x2. На рис. 3 показан график точности и потерь для 3-ей модели, получившиеся результаты были следующими:

- Тестовая точность – 0.4769
- Точность тренировки – 0.7667
- Точность валидации – 0.7684

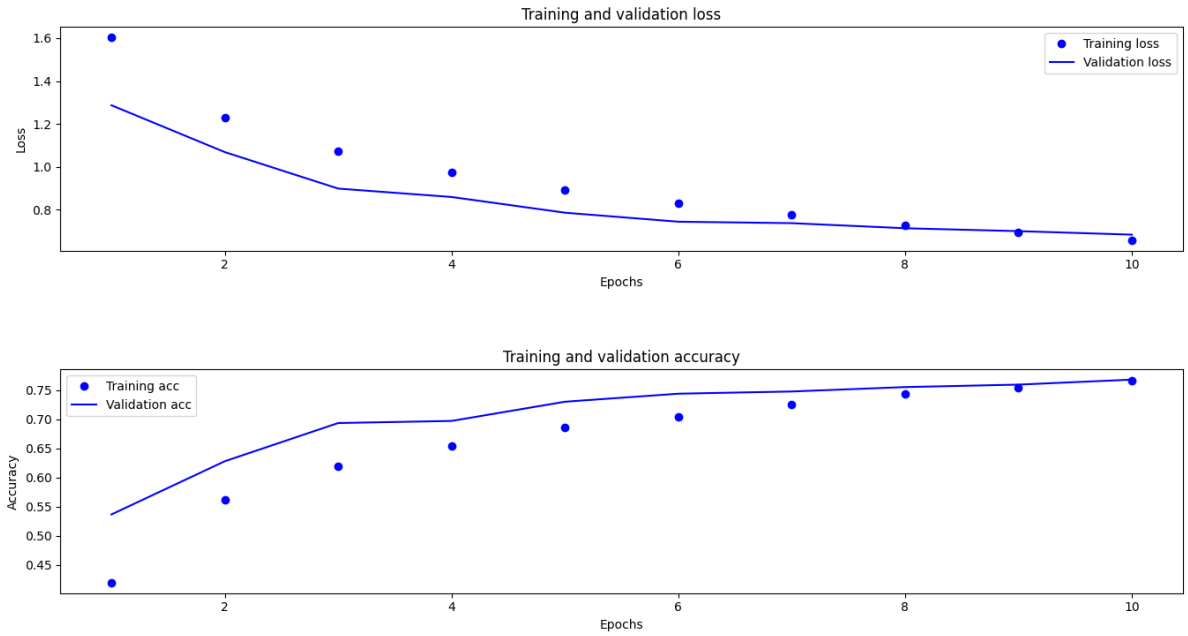

Рисунок 3 - Результаты 3-ей модели

Модель показывает примерно такое же поведение как и модель со стандартными настройками, но точность тестовых данных меньше.

Далее для 4-ой модели размерность ядра свертки была увеличена до 5x5. На рис. 4 показан график точности и потерь для 4-ой модели, получившиеся результаты были следующими:

- Тестовая точность – 0.5808
- Точность тренировки – 0.7798
- Точность валидации – 0.7604

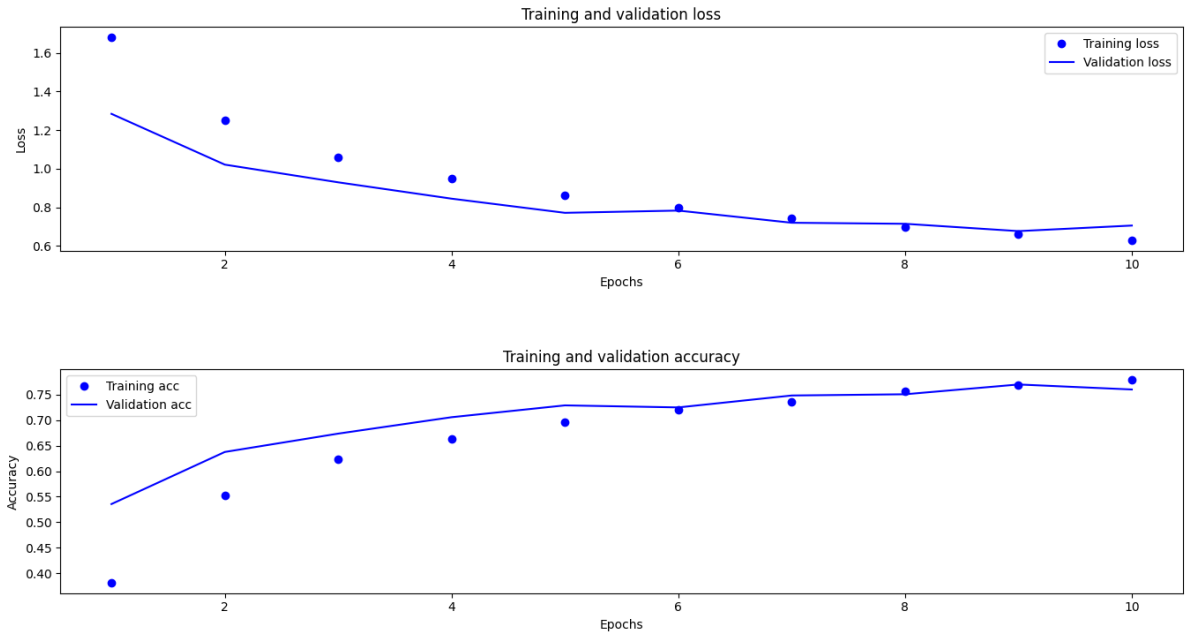

Рисунок 4 - Результаты 4-ой модели

Видно, что результаты практически соответствуют стандартной модели, но тренировка этой модели затратила больше времени.

Далее для 5-ой модели размерность ядра свертки была увеличена до 7x7. На рис. 5 показан график точности и потерь для 5-ой модели, получившиеся результаты были следующими:

- Тестовая точность – 0.5172
- Точность тренировки – 0.7242
- Точность валидации – 0.7086

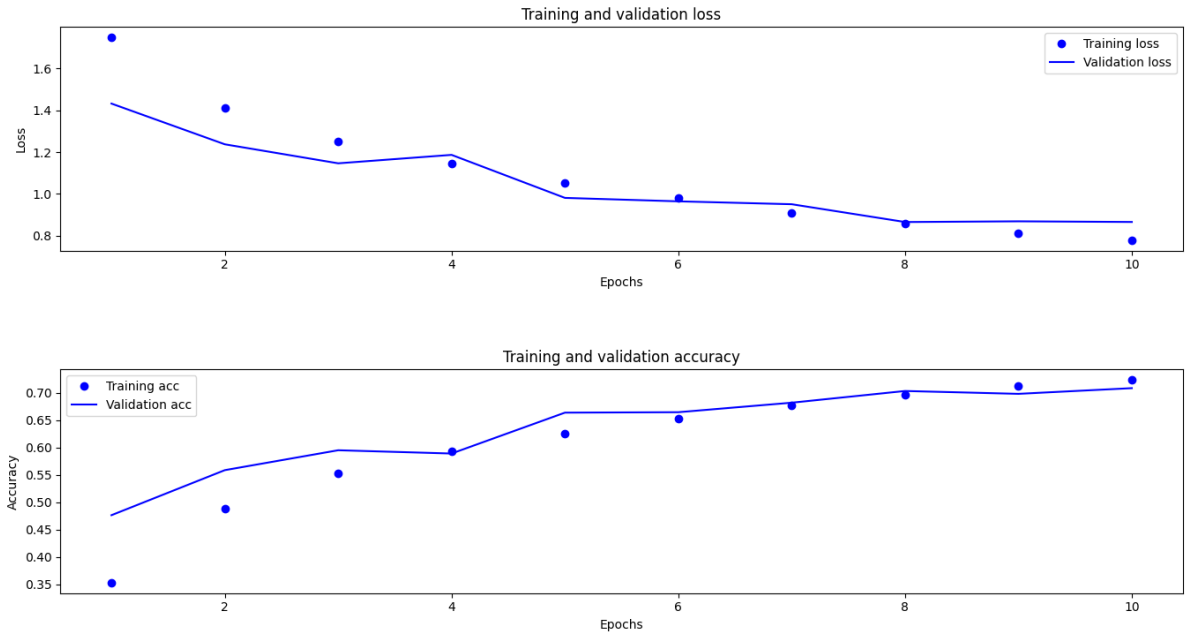

Рисунок 5 - Результаты 5-ой модели

Видно, что по сравнению со стандартной моделью точность упала, помимо этого время тренировки модели значительно увеличилось.

Выводы

В ходе работы была реализована сверточная нейронная сеть для распознавания объектов на фотографиях. Были проведены исследования зависимости работы модели от наличия слоя Dropout и от разных размерностей ядра свертки. Можно установить, что оптимальной моделью являются стандартная модель, реализованная в данной работе, имеющая размерность ядра свертки равной 3 и слой Dropout.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Convolution2D, MaxPooling2D, Dense, Dropout, Flatten
from tensorflow.python.keras import utils

# Глобальные параметры
batch_size = 75
num_epochs = 10
kernel_size = 7
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.25
drop_prob_2 = 0.5
hidden_size = 512

# Данные для сети
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
num_train, height, width, depth= X_train.shape
num_test = X_test.shape[0]
num_classes = np.unique(y_train).shape[0]
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= np.max(X_train)
X_test /= np.max(X_train)
Y_train = utils.np_utils.to_categorical(y_train, num_classes)
Y_test = utils.np_utils.to_categorical(y_test, num_classes)
```

```
   # Входной слой
   inp = Input(shape=(height, width, depth))

   # Слои Свертки и пуллинга
   conv_1 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
padding='same', activation='relu')(inp)
   conv_2 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
padding='same', activation='relu')(conv_1)
   pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)

   # Слой Dropout
   drop_1 = Dropout(drop_prob_1)(pool_1)

   # Еще слои свертки и пуллинга
   conv_3 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
padding='same', activation='relu')(drop_1)
   conv_4 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
padding='same', activation='relu')(conv_3)
   pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_4)

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

   H = model.fit(X_train, Y_train,
```

```
              batch_size=batch_size, epochs=num_epochs,
              verbose=1, validation_split=0.1)


model.evaluate(X_test, Y_test, verbose=1)

#Получение ошибки и точности в процессе обучения
loss = H.history['loss']
val_loss = H.history['val_loss']
acc = H.history['accuracy']
val_acc = H.history['val_accuracy']
epochs = range(1, len(loss) + 1)

#Построение графика ошибки
plt.subplot(2, 1, 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

#Построение графика точности
plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```