НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Лабораторная работа № 3

Классификация цветов с помощью свёрточных нейронных сетей.

Выполнил студент

Шамсулов Руслан Наилевич

Группа № R3235

Преподаватель: Mikhail Kakanov,

Oleg Evstafev

г. Санкт-Петербург

2021

**Текст задания**

В работе необходимо познакомится с различными архитектурами сверхточных нейронных сетей и их обучением на GPU (англ. graphics processing, графический процессор) на языке программирования Python 3 и фреймворка Torch (PyTorch). Для этого предлагается использовать ресурсы Google Colab - Colaboratory, для выполнения вычислений на GPU. После с ознакомления, выполнить практическое задане в конце данной тетради (notebook)

**Ход работы**

**Изображение выглядит как текст

Автоматически созданное описание**

GoogLeNet(

(conv1): BasicConv2d(

(conv): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(maxpool1): MaxPool2d(kernel\_size=3, stride=2, padding=0, dilation=1, ceil\_mode=True)

(conv2): BasicConv2d(

(conv): Conv2d(64, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(conv3): BasicConv2d(

(conv): Conv2d(64, 192, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(maxpool2): MaxPool2d(kernel\_size=3, stride=2, padding=0, dilation=1, ceil\_mode=True)

(inception3a): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(192, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(192, 96, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(96, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(192, 16, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(16, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(192, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(inception3b): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(256, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(256, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(128, 192, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(256, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(32, 96, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(256, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(maxpool3): MaxPool2d(kernel\_size=3, stride=2, padding=0, dilation=1, ceil\_mode=True)

(inception4a): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(480, 192, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(480, 96, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(96, 208, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(208, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(480, 16, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(16, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(16, 48, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(480, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(inception4b): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(512, 160, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(512, 112, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(112, 224, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(224, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(512, 24, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(24, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(512, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(inception4c): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(512, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(512, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(512, 24, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(24, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(24, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(512, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(inception4d): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(512, 112, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(112, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(512, 144, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(144, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(144, 288, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(288, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(512, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(32, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(512, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(inception4e): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(528, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(528, 160, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(160, 320, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(528, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(32, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(528, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(maxpool4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=True)

(inception5a): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(832, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(832, 160, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(160, 320, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(832, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(32, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(832, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(inception5b): Inception(

(branch1): BasicConv2d(

(conv): Conv2d(832, 384, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(branch2): Sequential(

(0): BasicConv2d(

(conv): Conv2d(832, 192, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(192, 384, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch3): Sequential(

(0): BasicConv2d(

(conv): Conv2d(832, 48, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

(1): BasicConv2d(

(conv): Conv2d(48, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(branch4): Sequential(

(0): MaxPool2d(kernel\_size=3, stride=1, padding=1, dilation=1, ceil\_mode=True)

(1): BasicConv2d(

(conv): Conv2d(832, 128, kernel\_size=(1, 1), stride=(1, 1), bias=False)

(bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

)

(aux1): None

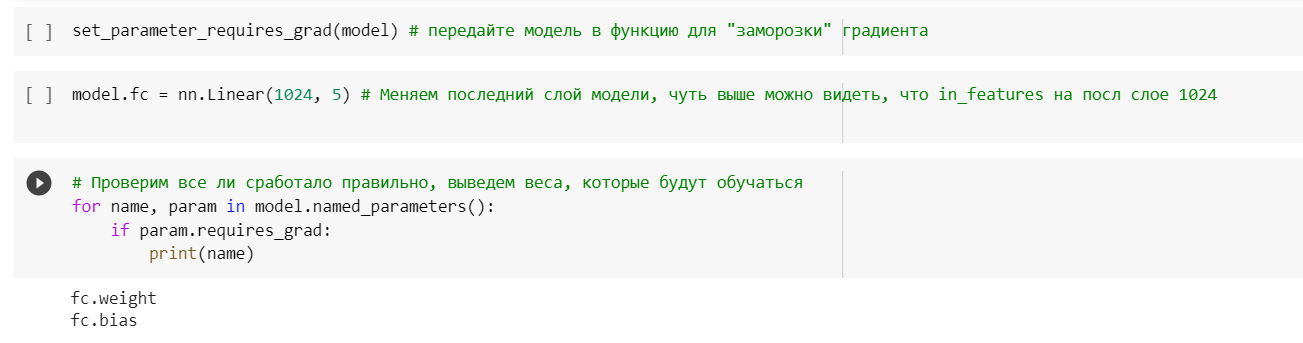
(aux2): None

(avgpool): AdaptiveAvgPool2d(output\_size=(1, 1))

(dropout): Dropout(p=0.2, inplace=False)

(fc): Linear(in\_features=1024, out\_features=1000, bias=True)

)

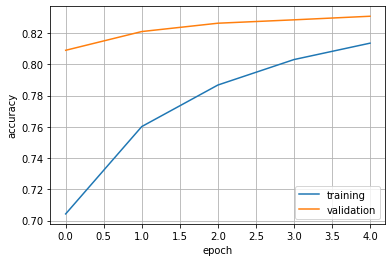
****

**Изображение выглядит как текст

Автоматически созданное описание**

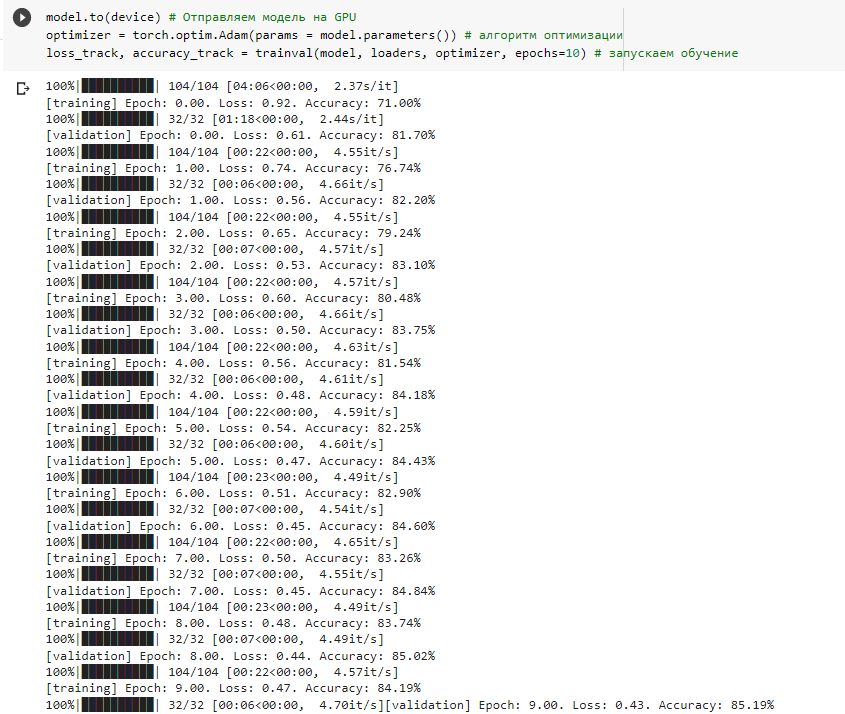
**Изображение выглядит как текст

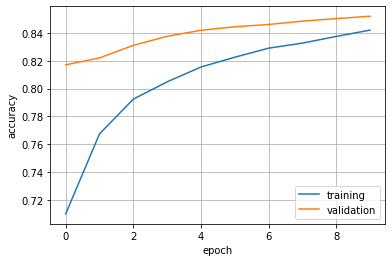
Автоматически созданное описание**

****

Исходя из графика можно сделать вывод о том, что существует необходимость в увеличении количества эпох для получения более высокого показателя accuracy.

Увеличим количество эпох до 10.

****

****

**Изображение выглядит как текст

Автоматически созданное описание**

Изображение выглядит как текст, дерево, цветок, растение

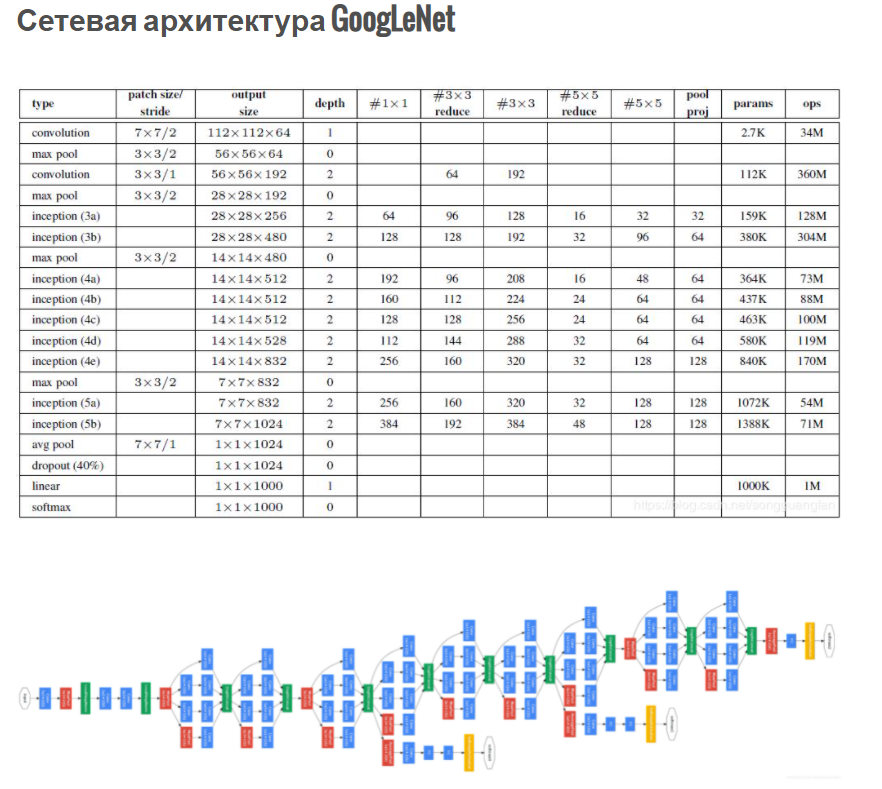
Автоматически созданное описание 

Изображение выглядит как текст, цветок, растение, подсолнечник

Автоматически созданное описание 

**Ответы на вопросы:**

* Как работает выбранная вами модель сверточной нейронной сети? Какие параметры?

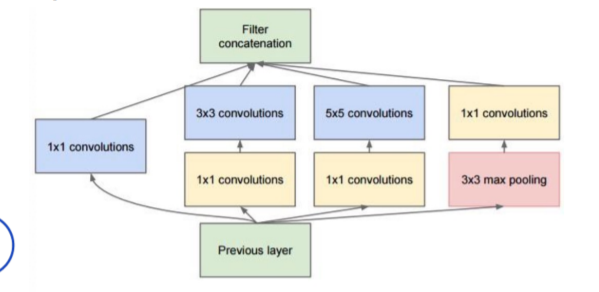


Идея создания googleNet основана на понятии inception blocks.

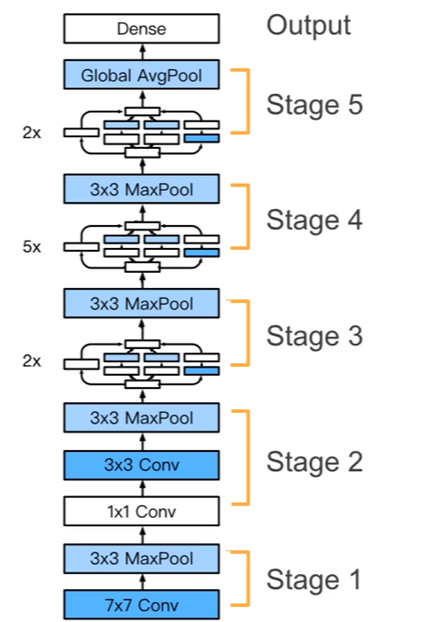
Inception block использует несколько ядер свертки разных размеров, также добавляет пулы, а затем объединяет результаты свертки и объединения( нужно найти оптимальную локальную разреженную структуру и покрыть ее как приблизительный плотный компонент, а именно объединить различные локальные структуры).Таким образом, учитываются характеристики разных размеров на каждом слое.

Основано на принципе Heb и многомасштабной обработке.  
Принцип Хеба: постоянная и повторная стимуляция пресинаптических нейронов в постсинаптические нейроны может привести к повышению эффективности синаптической передачи.

**Inception block**



**Схема в более приятном глазу виде:**



**Параметры**

Изображение выглядит как текст

Автоматически созданное описание

* **В чем основные отличия между сверточной нейронной сетью и "обычной" полносвязной нейронной сетью?**

­­­­С одной стороны, свёрточные нейронные сети похожи на обычные нейронные сети: они состоят из нейронов, которые, в свою очередь, содержат изменяемые веса и смещения. Каждый нейрон получает какие-то входные данные, вычисляет скалярное произведение и, опционально, использует нелинейную функцию активации. Вся сеть по-прежнему представляет собой единственную дифференциируемую функцию оценки: из исходного набора пикселей на одном конце до распределения вероятностей принадлежности к определённому классу на другом конце. У этих сетей по-прежнему есть функция потерь.

Проблема в том, что обычные нейронные сети плохо масштабируются для больших изображений.  Например, изображение большего размера — 200х200х3, приведёт к тому, что количество весов станет равным 200х200х3 = 120 000. Более того, нам понадобится не один подобный нейрон, поэтому общее количество весов быстро начнёт расти. Становится очевидным тот факт, что полносвязность чрезмерна и большое количество параметров быстро приведут сеть к переобученности.

Отличие заключается в том, что архитектура свёрточной нейронной сети, в своём простейшем представлении, представляет собой упорядоченный набор слоёв преобразующий представление изображения в другое представление. Это позволяет сильно уменьшить количество весов.

* **Что такое transfer learning?**

Transfer Learning (трансферное обучение) — это подраздел машинного обучения, целью которого является применение знаний, полученные из одной задачи, к другой целевой задаче.

* **Что такое функция для заморозки весов модели?**

В лабораторной работе стояла задача отнести изображение к одному из 5 классов, в предобученной же модели in\_features на последнем слое 1024. Нам не подходит такая модель «из коробки», следовательно нам необходимо изменить последний слой нашей свёрточной нейронной сети, чтобы он содержал то количество выходов, которое бы соответствовало количеству классов в новом наборе. Так же нам необходимо убедиться, что мы не изменим заранее обученную модель во время процесса тренировки. Решение заключается в отключении переменных предобученной модели — мы просто запретим алгоритму обновляющему значения при прямом и обратно распространении их менять.

"Замораживая" параметры предобученной модели мы позволяем обучаться только последнему слою сети классификации, значения переменных предобученной остаются неизменными.

Если мы не "заморозим" переменные предобученной модели, то в процессе обучения на новом наборе данных значения переменных будут меняться. Это происходит потому, что значения переменных на последнем слое классификации будут заполненны случайными значениями. Из-за случайных значений на последнем слое наша модель будет допускать большие ошибки в классификации, что, в свою очередь, повлечёт сильные изменения исходных весов в предобученной модели, что крайне нежелательно для нас.

**Вывод:**

В ходе работы я познакомился с различными архитектурами сверхточных нейронных сетей и их обучением на GPU с помощью Google Colab. Мною было рассмотрен код, реализующий сверточную нейронную сеть полностью с нуля.

В качестве практического задания я реализовал сверточную нейронную сеть GoogleNet для классификации цветов посредством transfer – learning.

Полученные метрики:

[validation] Loss: 0.43. Accuracy: 85.19%

[training] Loss: 0.47. Accuracy: 84.19%