МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

**Кафедра дифференциальных уравнений и системного анализа**

# Классификация грибов при помощи нейронных сетей

Курсовая работа

Легушева Дмитрия

студента 3-го курса
специальность 1-31 03 09
Компьютерная математика и
системный анализ

Научный руководитель:
ассистент А.П.Тишуров

Минск, 2018

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

Сверточные нейронные сети (Convolution neural networks) стали повсеместно популеризироваться в системах компьютерного зрении после того как архитектура AlexNet выиграла ImageNet Challenge: ILSVRC 2012 в 2012 году, снизив рекорд ошибок классификации с 26 до 15%, что тогда стало прорывом.

На сегодняшний день нейросети лежат в основе услуг многих компаний: Facebook использует нейронные сети для алгоритмов автоматического проставления тегов, Google — для поиска среди фотографий пользователя, Amazon — для генерации рекомендаций товаров, Pinterest — для персонализации домашней страницы пользователя, а Instagram — для поисковой инфраструктуры.

В первой части своей работы я расскажу про основы сверточных нейронных сетей, их происхождение, а также об основных их архитектурах. Во второй речь будет идти непосредственно о моем ходе работы, результатах, выводах и планах на будущее.

# ГЛАВА 1

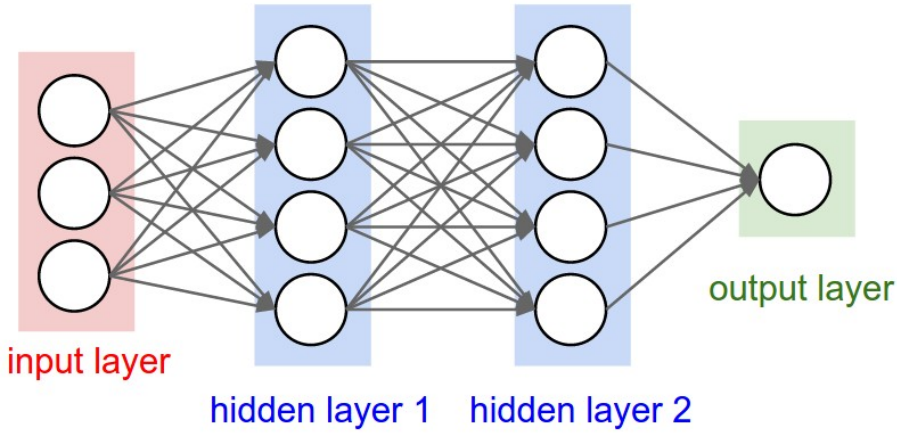
# Теория сверточных нейронных сетей

## 1.1 Принцип работы сверточных нейронных сетей.

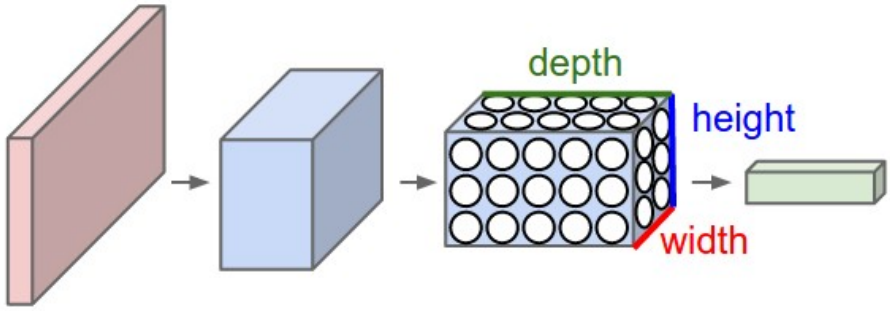
Прототипом для создания свертоных нейронных сетей послужили биологические нейронные сети. Две трети всей сенсорной информации, которая к нам попадает, приходит с зрительных органов восприятия. Изображение попадает к нам из сетчатки глаза, проходит череду зрительных зон и заканчивается в височной зоне. После проведеняи большого количетсва экспериментов в 60-ых годах было установлено два важнейших свойства — это увеличение рецептивных полей наших клеток по мере продвижения от первичных зрительных зон к более поздним зрительным зонам (височным долям), а также увелечение сложности объектов, которые мы распознаем. Рецептивное поле — это та часть изображения, которую обрабатывает каждая клетка (нейрон) головного мозга. Как оказалось, все эти свойства можно перенести в нейронную сеть.

Сверточные нейронные сети очень похожи на обычные. Они состоят из нейронов, которые содержат обучаемые веса и сдвиги (biase). Каждый нейрон содержит вход, сумматор и определенную нелинейную функцию. Сама сеть в целом представляется как единая дифференцируемая функция, которой на вход поступает тензор пикселей картинки, а на выходе получается оценка по классам. Веса в ConvNet обучаются при помощи градиентного спуска таким образом, чтобы результат выхода сети вычислялся в соответствии с правильным значением каждого изображении в тренировочной выборке.

Особенностью сверточных нейронных сетей является тот факт, что входом является изображение, поэтому архитектура строится более рациональным способом. В частности, в отличии от обычных нейронных сетей, слои ConvNets содержат нейроны (другое название **фильтры**) размерности 3 (ширина, высота, глубина), которые, в процессе перехода от одного слоя к последующему, сокращают размерность изображения до момента вывода вектора предсказаний.




(a) Классическая трехслойная нейронная сеть




(b) ConvNet с трехразмерными нейронами

Рис. 1.1: Сравнение классической и сверточной нейронных сетей

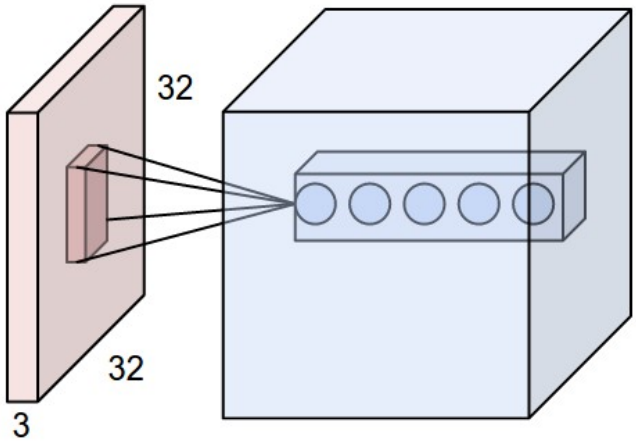
Чаще всего используется три основных типа слоев, которые конструируют ConvNet архитектуру:

- Слой свертки (convolution layer). Слой свертки вычисляет выходы фильтров, которые пространственно присоединены к локальным областям входа, образуя из него так называемые features maps, которые стэкаются друг на друга . Каждый фильтр вычисляет скалярное произведение между своими весами и малой областью входа, при это переходя с некоторым шагом (stride) по данной области. Также данная операция может сопровождаться определенной операцией падинга (padding).

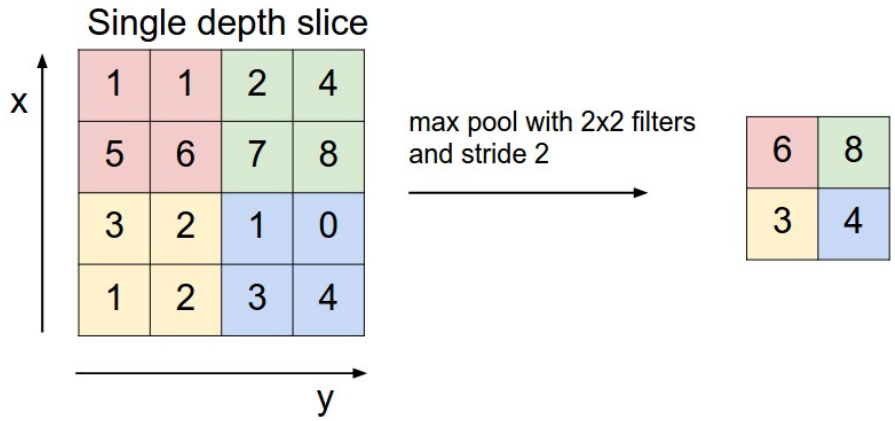
- Слой пулинга (pooling layer). Слой пулинга выполняет операцию сокращения размерности вдоль пространственных размерностей (высота, ширина). Данной сокращение может выполняться различными способами (max, average).

- Полносвязный слой (Fully-Connected layer).Полносвязный слой вычисляет оценки по каждому классу. Каждый нейрон данного слоя будет будет

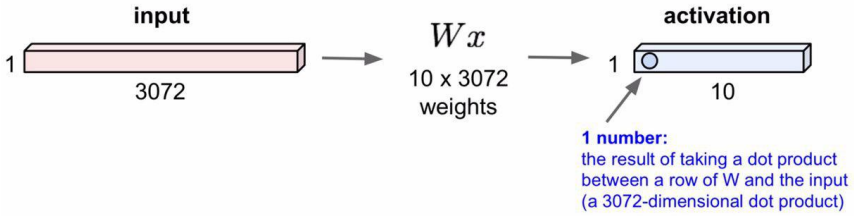
связан со всеми значениями выхода предыдущего слоя. Аналогичен полносвязному слою в классической нейронной сети.




(a) Сверточный слой(5 нейронов)




(b) Принцип работы pooling слоя




(c) Полносвязный слой

Рис. 1.2: Типы слоев в сверточных нейросетях

## 1.2 Основные архитектуры нейронных сетей. Архитектура MobileNet.

Существует некоторые основные архитектуры сверточных нейронных сетей. У них есть собственное имя и некоторые особенности. Наиболее распространенные это:

1. LeNet. Первое успешное применение ConvNets было разработано Yann LeCun в 1990 году. В основном данная архитектура прославилась тем, что использовалась для чтения сжатых кодов, данных и так далее.

2. AlexNet. Первая архитектура, которая положила начало в изучении и исследовании в области сверточных нейронных сетей массово. Разработана Alex Krizhevsky, Ilya Sutskever и Geoff Hinton. AlexNet выиграла ImageNet ILSVRC challenge в 2012 с результатом (top 5 error of 16%) Архитектура повторяла LeNet, но была глубже, больше и в ней использовалось больше слоев со свертками, расположенных последовательно.

3. ZF Net. Победитель ILSVRC challenge в 2013. Разработана Matthew Zeiler и Rob Fergus (откуда и название по первым буквам фамилии ZFNet). Была улучшена выводом более оптимальных параметров архитектуры и обучения. Также был расширен размер средних сверточных слоев и сокращен шаг свертки по изображению и размер фильтра (свертки) на первом слое.

4. GoogLeNet. Победитель ILSVRC challenge в 2014. Google разработка. Главная особенность архитектуры стало введение Inception Models, который значительно сократили количетсво параметров в сети (4 миллиона в сравнении с AlexNet с 60 миллионами). Также данная сеть впервые использует Global Average Pooling слой вместо полносвязного слоя, что также значительно сократило количество параметров. Есть несколько модификация данной модели. Одна из них – Inception-v4.

5. VGGNet. Победитель ILSVRC challenge в 2014. Разработана Karen Simonyan и Andrew Zisserman. Данная архитектура показала, что особенно важной компонентой для лучшего результата работы ConvNet является ее глубина. Однако данная модель содержит колоссально большое количество параметров в сравнении с предыдущими (около 140 миллионов).

6. ResNet (Residual Network). Победитель ILSVRC challenge в 2015. Разработана Kaiming He. Особенностью данной сети является использование специальной skip связи слоев и обильного использования батч нормализации (batch normalization). Архитектура также не использует полносвязных слоев в конце сети. На данный момент является канонической архитектурой. На практике обычно применяется данный тип сети.

Модель, которую я использовал для своих исследований, называется MobileNet. Класс моделей MobileNet используется для мобильных или встроенных приложений компьютерного зрения. Ключевой особенностью архитектуру является использование в ней специального вида свертки depthwise separable convolution, которая значительно сокращает количество параметров, а следовательно вес модели и время вычисления. Depthwise separable convolution – это форма факторизированной свертки, которая разлагает обычную свертку на две, depthwise и $1 \times 1$ pointwise. Depthwise свертка применяет фильтр к каждому входному каналу. Pointwise применяет $1 \times 1$ фильтр, чтобы скомбинировать поканально выход depthwise. В обычной операции свертки единственный фильтр комбинирует вход в выход за один шаг. Depthwise separable convolution разбивает эту операцию на две, образуя из одного слоя два, слой фильтрации и слой комбинирования. Данная факторизация имеет эффект значительного сокращения количества параметров модели. Если размер входа имеет размерность $D_F \times D_F \times M$, а обычный слой свертки имеет размер $D_K \times D_K \times M \times N$, где $D_K$ – пространственный размер фильтра (допускается что это квадрат), M – число входных каналов, N – число каналов выхода,то количество операций для свертки :

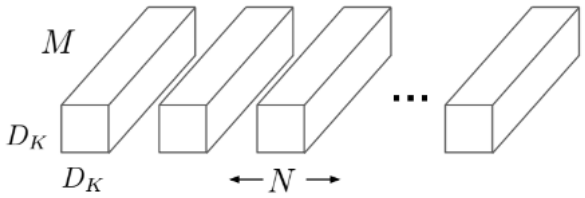
$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

В свою очередь depthwise convolution будет иметь размерность $D_K \times D_K \times M$, а separable convolution $1 \times 1 \times N$, следовательно количетсво операций:

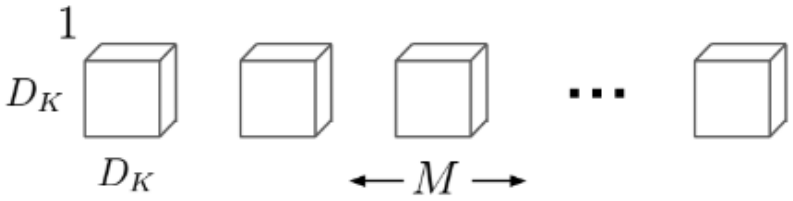
$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

При разбиении обычной операции свертки на две мы получаем сокращение количества вычислений на:

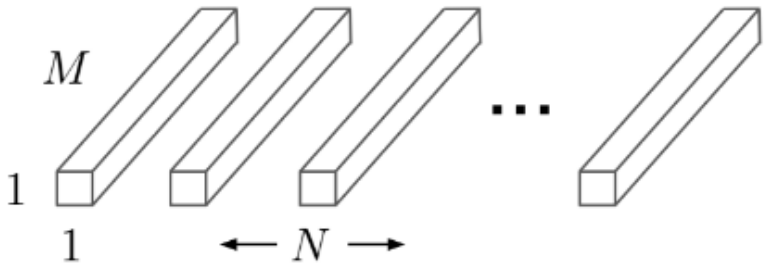
$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$




(a) Обычная свертка




(b) Depthwise свертка




(c) Pointwise свертка

Рис. 1.3: Типы сверток и их размерности

MobileNet использует $3 \times 3$ depthwise separable convolution, который требует примерно в 8 – 9 раз меньше вычислений, чем обычная операция свертки, при это потеря в точности является незначительной. Также количество параметров значительно сокращено (около 4 млн) в сравннении с основными архитектурами (AlexNet – 60 млн, VGG – 140 млн). Полная архитектура сети имеет вид:

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | $3 \times 3 \times 3 \times 32$ | $224 \times 224 \times 3$ |
| Conv dw / s1 | $3 \times 3 \times 32$ dw | $112 \times 112 \times 32$ |
| Conv / s1 | $1 \times 1 \times 32 \times 64$ | $112 \times 112 \times 32$ |
| Conv dw / s2 | $3 \times 3 \times 64$ dw | $112 \times 112 \times 64$ |
| Conv / s1 | $1 \times 1 \times 64 \times 128$ | $56 \times 56 \times 64$ |
| Conv dw / s1 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 128$ | $56 \times 56 \times 128$ |
| Conv dw / s2 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 256$ | $28 \times 28 \times 128$ |
| Conv dw / s1 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 256$ | $28 \times 28 \times 256$ |
| Conv dw / s2 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 512$ | $14 \times 14 \times 256$ |
| $5\times$ Conv dw / s1 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| $5\times$ Conv / s1 | $1 \times 1 \times 512 \times 512$ | $14 \times 14 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| Conv / s1 | $1 \times 1 \times 512 \times 1024$ | $7 \times 7 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 1024$ dw | $7 \times 7 \times 1024$ |
| Conv / s1 | $1 \times 1 \times 1024 \times 1024$ | $7 \times 7 \times 1024$ |
| Avg Pool / s1 | Pool $7 \times 7$ | $7 \times 7 \times 1024$ |
| FC / s1 | $1024 \times 1000$ | $1 \times 1 \times 1024$ |
| Softmax / s1 | Classifier | $1 \times 1 \times 1000$ |

Рис. 1.4: Архитектура MobileNet

# ГЛАВА 2
# Классификация грибов

В рамках второй главы речь зайдет о непосредственной задаче курсовой работы – классификации грибов по различным характеристикам при помощи сверточных нейронных сетей.

## 2.1 Подготовка датасета

База данных **ImageNet** — проект по созданию и сопровождению массивной базы данных аннотированных изображений, предназначенная для отработки и тестирования методов распознавания образов и машинного зрения. Для сбора данных под задачу,в основном, использовался данный источник. По данным от августа 2017 года в ImageNet 14197122 URL с изображениями, разбитых на 21841 категорию. Около 10 миллионов URL с изображениями прошли ручную аннотацию для ImageNet. В аннотациях перечислялись объекты, попавшие на изображение, и прямоугольники с их координатами. База данных с аннотацией и URL изображений от третьих лиц доступна непосредственно через ImageNet, но при этом сами изображения не принадлежат проекту.

В процессе сбора данных были выкачены текстовые файлы с URL с изображениями для 39 основных типов грибов. В общей сложности получилось 23672 выкаченных, отсортированных изображений с грибами. Также для работы пригодилось фотографии предметов различной категории с доминантными объектами для задачи классификации гриб/не гриб. В этом случае использовались как фото от ImageNet, так и сторонние. В общей сложности получилось 22317 выкаченных, отсортированных изображений.

Для загрузки самих изображений из URL использовалась Python библиотека **urllib** – специализированынй модуль для работы с URL.

В дальнейшем датасет подвергался разбиению на классы по типу гриба и на съедобный/несъедобный.

## 2.2 Бинарная классификация (гриб/не гриб)

Во время поиска похожих приложений для классификации грибов было замечено, что они не способны отличить гриб от какого-либо стороннего предмета. По этой причине встала задача построить классификатор, который будет отличать гриб от не гриба. Для реализации задачи использовался язык программирования Python и фрэймворк TensorFlow под Keras оболочкой. Архитектура MobileNet была развернута в keras.applications, поэтому необходимости в повторной реализации не было.

В датасете содержались грибы из всех 39 классов, а также сторонние изображения класса "не гриб в который входили различные типы фруктов, грибы от ядерного взрыва, часы, самолеты, мотоциклы, автомобили и т.д. Из-за большой величины датасета появилась необходимость не загружать весь его в оперативную память компьютера, поэтому использовалось специализированные keras средства называемые ImageGenerator (содержится в keras.preprocessing.image), который подгружал данные с жесткого диска.

Размер входной картинки установил $200 \times 200$. При обучении использовался тип оптимизации Adam, который включает в себя различные другие типы (Momentum, AdaGrad/RMSProp) Метрика - accuracy, функция потерь - логистическая кроссэнтропия (logistic crossentropy), которая для бинарной классификации имеет вид:

$$H(y, p) = -(y \ln(p) + (1 - y) \ln(1 - p))$$

где

- y - истинная метка;
- p - предсказанная вероятность.

Обучение проходило на видеокарте Nvidia GeForce GTX 1800, 70000 эпох, размер батча – 10. В результате на train/val датасете получил точность в 90 %, на тесте 75 - 80 %. Причиной в низкой точности на тестовой выборке , по моему мнению, скорее всего является в сортировке тренировочной выборке от низкокачественных, путающих нейросеть изображениях. (изображениях с низким качеством, с большим количетсвом объектов одинаковой и разной природы, с объектами которые не относятся к нужному классу и т.д.)


(a) Гриб–0.002,Не гриб – 0.99 (b) Гриб–0.0008,Не гриб – 0.99 (c) Гриб–0.99,Не гриб – 0.0024

Рис. 2.1: Хорошие результаты


(a) Гриб–0.7726,Не гриб – 0.2274 (b) Гриб–0.0022,Не гриб – 0.99

Рис. 2.2: Плохие результаты

## 2.3 Многоклассовая классификация (название гриба)

В рамках работы над курсовым проектом была поставлена задача реализовать возможность многоклассовой классификации по типам грибов. Из 39 классов датасета было выбрано 17, изображений которых было больше всего и которые больше присуще для белорусской местности. Основные из них:

- Опенок зимний (Winter mushroom, количество в train – 674)
- Боровик (Bolete, количество в train – 1986)
- Лисичка (Chanterelle, количество в train – 693)
- Опенок (Honey mushroom, количество в train – 512)
- Бледная поганка (Death Cap, количество в train – 861)

- Мухомор (Fly Agaric, количество в train – 2500)
- Гриб-зонтик (Parasol, количество в train – 578)

Архитектура, параметры обучения, вид подкачки датасета в память, входной размер изображения, метрика, количество эпох осталось такое же что и в бинарной (гриб/ не гриб) классификации. Изменился только размер батча – 32 и вид функции потерь (многоклассовая логистическая кроссэнтропия):

$$H(y, p) = -\sum_{x} y(x) \ln(p(x))$$

Как результат на train/val датасете получил точность в 90 %, на тесте 55 - 60 %. Было замечено, что классификатор хорошо распознал изображения, количество которых преобладало в датасете.


(a) Мухомор–0.9798


(b) Лисичка – 0.8537


(c) Боровик – 0.9510


(d) Опенок зимний – 0.6181

Рис. 2.3: Хорошие результаты


(a) Опенок–0.6534 (b) Бледная поганка–0.6034 (c) Лисичка–0.9994

Рис. 2.4: Плохие результаты

## 2.4 Бинарная классификация с элементами многоклассовой (съедобный/несъедобный)

На последнем этапе работы была поставлена цель – обучить нейросеть бинарно разделять грибы на ядовитый/неядовитый, но при этом попробовать заставить ее как можно хуже различать конкретные классы. Для реализации этой идеи использовался классификатор с несколькими выходами (ядовитый-/неядовитый и многоклассовый) и комбинированная функция потерь:

- Логистическая кроссэнтропия (для выхода ядовитый/неядовитый)
- Дивергенция Кульбака-Лейблера между равномерным распределением и предсказаниями на конкретные классы грибов(для многоклассового выхода)

Дивергенция Кульбака-Лейблера является лишь некоторой модификацией формулы для кроссэнтропии, которая показывает меру удаленность друг от друга двух вероятностных распределений. В нашем случае мы используем это расстояние как функцию потерь, где в роли первого распределения играет результат, который вычисляет нейронная сеть, а второго – равномерное распределение на 17 классов (равна количеству классов в сокращенном датасете):

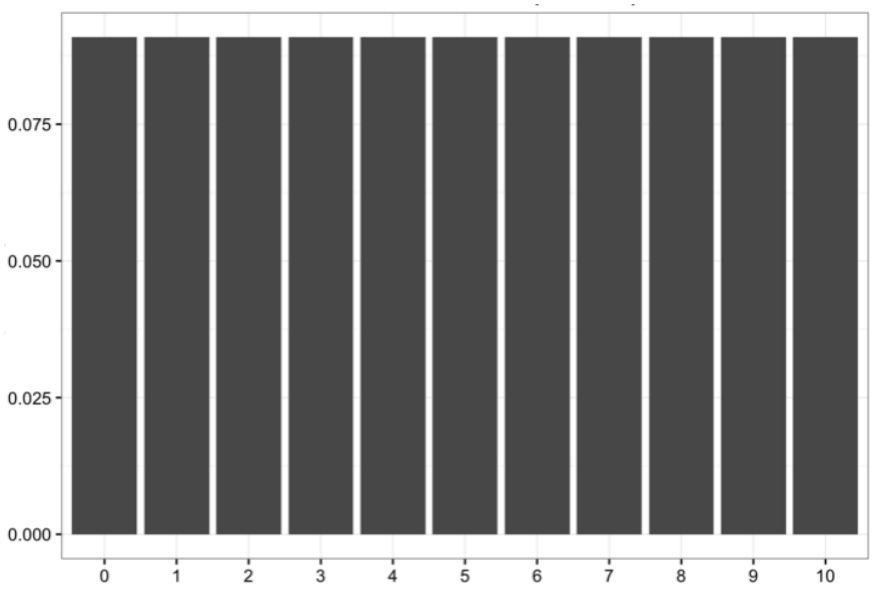



Рис. 2.5: Равномерное распределение

Формула дивергенции имеет вид:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i)(\ln p(x_i) - \ln q(x_i))$$

Таким образом мы пытаемся заставить нейронную сеть размыть представление о классах и при этом выучить разделять грибы на ядовитый/неядовитый.

Размер входной картинки 200 × 200. При обучении использовался тип оптимизации Adam. Метрика - accuracy. В данном случае пришлось уменьшать датасет для того, чтобы полностью загрузить его в оперативную память. Обычный ImageGenerator в данном случае не сработал.

В результате на train/val датасете получил точность в 90 %, на тесте 70% на классификации ядовитый/неядовитый, при этом с довольно неплохим равномерном распределении на второй голове.

```
[array([[ 0.36097854,  0.6390214 ]], dtype=float32),
 array([[ 0.05882356,  0.05882357,  0.05882367,  0.05882354,  0.05882344,
          0.05882368,  0.05882364,  0.05882308,  0.05882353,  0.05882345,
          0.05882353,  0.05882404,  0.0588231 ,  0.05882357,  0.05882349,
          0.05882348,  0.05882358]], dtype=float32)]
```

Рис. 2.6: Вывода некоторого результата

Как показано на Рис.2.6 первый элемент списка это результат первого выхода на бинарную классификацию ядовитый/неядовитый, следовательно

второй элемент результат второго выхода с ярковыраженным равномерным распределением.


(a) Несъедобный–0.6390 (b) Съедобный – 0.9370 (c) Съедобный – 0.9823

Рис. 2.7: Хорошие результаты


(a) Съедобный–0.6362

Рис. 2.8: Плохие результаты

# ЗАКЛЮЧЕНИЕ

В результате работы были обучены три сверточных нейросети одинаковой архитектуры для различных задач классификации:

- Бинарная классификация
- Многоклассовая классификация
- Бинарная классификация с элементами многоклассовой

Точность всех трех ConvNets не оказалось достаточно высокой, по моему мнению, из-за недостаточно хорошо отсортированного датасета от путающих нейросеть изображениях. (изображениях с низким качеством, с большим количетсвом объектов одинаковой и разной природы, с объектами которые не относятся к нужному классу и т.д.) При сборе своего датасета нужно это внимательно учитывать (особенно в случае распознавания относительно похожих объектов).

В перспективе ставятся задачи соединить классификацию и детекцию грибов на изображении, а также попробовать оптимизировать это под приложение на смартфоне с относительно слабыми типами процессоров.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ


[1] CS231n: Convolutional Neural Networks for Visual Recognition. Lecture course /Fei-Fei Li, Justin Johnson, Serena Yeung

[2] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications /Andrew G. Howard, Menglong Zhu, Bo Chen

[3] [Электронный ресурс] https://habr.com/post/322392/

[4] [Электронный ресурс] https://ru.wikipedia.org/wiki/ImageNet

[5] [Электронный ресурс] https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

[6] [Электронный ресурс] https://keras.io/

[7] [Электронный ресурс] https://docs.opencv.org/3.0-beta/doc/

[8] [Электронный ресурс] https://www.tensorflow.org/api_docs/

# ПРИЛОЖЕНИЕ А

## Код программы

```
#load images
import urllib
import os
import sys
import cv2
import numpy as np

import matplotlib.pyplot as plt

filepath = "/home/dl/dl/mushroom/synset/no_mushr/"

files = os.listdir(filepath)
print(files)

# open file to read
def load(filepath, files):
    flag = False
    if type(files) != list:
        files = [files]

    for file in files:
        name = file.split('.')[0]
        print('---' + name + '---')
        with open("{}".format(filepath + file), 'r') as csvfile:
            # iterate on all lines
            lines = list(csvfile)
            i = 0
            for line in lines:
                try:
                    # check if we have an image URL
                    #urllib.request.urlretrieve(line[:-1], "dataset/mushrooms/" +
                    name + "_" + str(i) + ".jpg")
                    response = urllib.request.urlopen(line[:-1], timeout = 30.0)
                    with open("dataset/not_mushrooms/" + name + "_" + str(i) +
                    ".jpg", 'wb') as img:
                        resource = response.read()
                        img.write(resource)
                    print("Image saved for {0} as {1}".format(line[:-1], name
                    + "_" + str(i) + ".jpg"))
                    i += 1
                except urllib.error.HTTPError as e:
                    # Return code error (e.g. 404, 501, ...)
                    print('---HTTPError: {}'.format(e.code))
                    print('Reasone: {}'.format(e.reason))
                    print(line, "\n")
                    i += 1
                except urllib.error.URLError as e:
                    # Not an HTTP-specific error (e.g. connection refused)
                    print('---Reasone: {}'.format(e.reason))
                    print(line, "\n")
                    i += 1
                except KeyboardInterrupt:
                    print("Interrupt")
```

```
                    raise
                except:
                    print("Unexpected error:", sys.exc_info()[0])
                    print(line, "\n")
                    i += 1
                else:
                    # 200
                    print('good', "\n")
load(filepath, files)

#binary/multy classfication
# BINARY
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "1"

import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import keras
import pydot
import cv2

from keras.applications.inception_v3 import InceptionV3
from keras.applications.vgg16 import VGG16
from keras.applications.mobilenet import MobileNet
from keras.models import Model, model_from_json
from keras.layers import Dense, Dropout, Flatten,
AveragePooling2D, Input, GlobalAveragePooling2D
from keras.optimizers import SGD,Adam
from keras.callbacks import EarlyStopping, LearningRateScheduler
from keras.preprocessing.image import ImageDataGenerator
from keras import regularizers
from keras.callbacks import ModelCheckpoint
from keras.utils import plot_model
from keras.backend.tensorflow_backend import set_session

from tqdm import tqdm

#from sklearn.model_selection import train_test_split,
# StratifiedShuffleSplit

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, \
zoom_range=0.2, horizontal_flip=True, vertical_flip=True)
test_datagen  = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('ssd-tensorflow/CW/
train/', target_size=(200, 200), batch_size=10)
val_generator   = train_datagen.flow_from_directory('ssd-tensorflow/CW/
val/', target_size=(200, 200), batch_size=10)

# mobile_net
def num_epoch_decay(epoch):
    if epoch <= 3000:
        lrate = 0.001
    elif 3000 < epoch <= 5000:
        lrate = 0.0001
    else:
        lrate = 0.00001
    print(lrate)
```

```
    return lrate

im_size = 200
num_class = 2

model = MobileNet(input_shape=(im_size, im_size, 3),weights=None,
 classes=num_class)

adam = Adam(decay=1e-6)
model.compile(optimizer=adam, loss='categorical_crossentropy',
 metrics=['accuracy'])

model.summary()

# Training
checkpoints = ModelCheckpoint(filepath='ssd-tensorflow/CW/checkpoints/
weights.{epoch:02d}-{val_loss:.2f}.hdf5', verbose=2, save_best_only=True,
 period=1500)
lrate = LearningRateScheduler(num_epoch_decay)

callbacks_list = [checkpoints, lrate]

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
set_session(sess)

model.fit_generator(train_generator, steps_per_epoch=10, epochs=7000, \
                    validation_data=val_generator, validation_steps=5,
                     callbacks=callbacks_list)
# MULTY
# Using flow_from_directory
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2, \
                                   horizontal_flip=True, vertical_flip=True)

test_datagen  = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('ssd-tensorflow/CW/
train/', target_size=(200, 200), batch_size=32)
val_generator   = train_datagen.flow_from_directory('ssd-tensorflow/CW/
val/', target_size=(200, 200), batch_size=32)

im_size = 200
num_class = 17

model = MobileNet(input_shape=(im_size, im_size, 3),weights=None,
classes=num_class)

adam = Adam(decay=1e-6)
model.compile(optimizer=adam, loss='categorical_crossentropy',
metrics=['accuracy'])

model.summary()

#training
checkpoints = ModelCheckpoint(filepath='ssd-tensorflow/CW/checkpoints/
weights.{epoch:02d}-{val_loss:.2f}.hdf5', verbose=2, save_best_only=True,
period=1500)
#early_stop = EarlyStopping(monitor='val_acc', patience=3, verbose=2)
```

```
lrate = LearningRateScheduler(num_epoch_decay)

callbacks_list = [ checkpoints, lrate]

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
set_session(sess)

model.fit_generator(train_generator, steps_per_epoch=10, epochs=7000, \
                    validation_data=val_generator, validation_steps=5,
                    callbacks=callbacks_list)
# predict for multy and binary (go down)

# KULLBACK-LEIBLER
import os

import numpy as np
import os
import tensorflow as tf
import keras
import cv2
import pandas
import matplotlib.pyplot as plt

from keras.layers import Dense, Dropout, Flatten, AveragePooling2D, Input,
 GlobalAveragePooling2D
from keras.callbacks import EarlyStopping, LearningRateScheduler
from keras.backend.tensorflow_backend import set_session
from keras.applications.mobilenet import MobileNet
from keras.callbacks import ModelCheckpoint
from keras.optimizers import SGD,Adam
from keras.models import Model
from keras.utils import to_categorical
from keras import regularizers
from keras.utils.generic_utils import CustomObjectScope

from sklearn.utils import shuffle

%matplotlib inline

im_size = 200
num_class_1 = 2
num_class_2 = 17

def to_list(lst_str):
    lst = []
    for i in lst_str:
        if len(i) == 1:
            lst.append(int(i))
        elif i[0] == '[':
            lst.append(int(i[1]))
        elif i[1] == ']':
            lst.append(int(i[0]))

    return lst
to_list(['[0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
 '0', '0', '0', '1]\n'])

# train
```

```
x_y_tr_1 = {}
with open('dataset_eadible/train.txt', 'r') as f:
    for i in f:
        line = i.split(" ")
        x_y_tr_1[line[0]] = to_list(line[1:])
        print line[0], to_list(line[1:])

# validation
x_y_val_1 = {}
with open('dataset_eadible/val.txt', 'r') as f:
    for i in f:
        line = i.split(" ")
        x_y_val_1[line[0]] = to_list(line[1:])
        print line[0], to_list(line[1:])

un_distr = [1. / num_class_2  for i in range(num_class_2)]

# train
x_y_tr_2 = {}
with open('dataset_classes/train.txt', 'r') as f:
    for i in f:
        line = i.split(" ")
        x_y_tr_2[line[0]] = un_distr
        print line[0], un_distr

# validation
x_y_val_2 = {}
with open('dataset_classes/val.txt', 'r') as f:
    for i in f:
        line = i.split(" ")
        x_y_val_2[line[0]] = un_distr
        print line[0], un_distr

# train
x_names_tr = []
y_tr = [[], []]
k = 0
for i in x_y_tr_2.keys():
    try:
        y_tr[0].append(np.array(x_y_tr_1[i]))
        y_tr[1].append(np.array(x_y_tr_2[i]))
        x_names_tr.append(i)
    except KeyError:
        k += 1

print k

# validation
x_names_val = []
y_val = [[], []]
k = 0
for i in x_y_val_2.keys():
    try:
        y_val[0].append(np.array(x_y_val_1[i]))
        y_val[1].append(np.array(x_y_val_2[i]))
        x_names_val.append(i)
    except KeyError:
        k += 1

print k
```

```
def num_epoch_decay(epoch):
    if epoch <= 3000:
        lrate = 0.001
    elif 3000 < epoch <= 5000:
        lrate = 0.0001
    else:
        lrate = 0.00001
    print(lrate)
    return lrate

# train
X_tr = []
Y_tr = [[], []]
for i in range(len(x_names_tr)):
    X_tr.append(cv2.imread('dataset_classes/train/' + x_names_tr[i]))
    Y_tr[0].append(y_tr[0][i])
    Y_tr[1].append(y_tr[1][i])

# validation
X_val = []
Y_val = [[], []]
for i in range(len(x_names_val)):
    X_val.append(cv2.imread('dataset_classes/val/' + x_names_val[i]))
    Y_val[0].append(y_val[0][i])
    Y_val[1].append(y_val[1][i])

Y_tr[0] = np.array(Y_tr[0])
Y_tr[1] = np.array(Y_tr[1])

Y_val[0] = np.array(Y_val[0])
Y_val[1] = np.array(Y_val[1])

X_tr = np.array(X_tr)
#Y_tr = np.array(Y_tr)

X_val = np.array(X_val)
#Y_val = np.array(Y_val)

model = MobileNet(input_shape=(im_size, im_size, 3), weights=None, include_top=False)

x = model.output

x_newfc = GlobalAveragePooling2D()(x)
predictions_1 = Dense(num_class_1, activation='softmax',
name='predictions_1', \
                      kernel_regularizer=regularizers.l2(0.01),
                      activity_regularizer=regularizers.l1(0.01))(x_newfc)
predictions_2 = Dense(num_class_2, activation='softmax',
name='predictions_2', \
                      kernel_regularizer=regularizers.l2(0.01),
                      activity_regularizer=regularizers.l1(0.01))(x_newfc)

model = Model(inputs=model.input, outputs=[predictions_1, predictions_2])

adam = Adam(decay=1e-6)
model.compile(optimizer=adam, loss=['categorical_crossentropy',
'kullback_leibler_divergence'], metrics=['accuracy'], \
              loss_weights=[0.5, 0.5])
```

```
model.summary()

checkpoints = ModelCheckpoint(filepath='checkpoints/
weights.{epoch:02d}-{val_loss:.2f}.hdf5', verbose=2, save_best_only=True,
period=10)
#early_stop = EarlyStopping(monitor='val_acc', patience=3, verbose=2)
lrate = LearningRateScheduler(num_epoch_decay)

callbacks_list = [ checkpoints, lrate]

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
set_session(sess)

model.fit(X_tr, Y_tr, batch_size=10, epochs=40, callbacks=callbacks_list,
validation_data=(X_val, Y_val), \
          shuffle=True)

# predict
img = cv2.imread('??Destroying_angel_720.jpg')
img = cv2.resize(img, (im_size, im_size))

model = MobileNet(input_shape=(im_size, im_size, 3), weights=None,
include_top=False)

x = model.output

x_newfc = GlobalAveragePooling2D()(x)
predictions_1 = Dense(num_class_1, activation='softmax',
name='predictions_1', \
                      kernel_regularizer=regularizers.l2(0.01),
                      activity_regularizer=regularizers.l1(0.01))(x_newfc)
predictions_2 = Dense(num_class_2, activation='softmax',
name='predictions_2', \
                      kernel_regularizer=regularizers.l2(0.01),
                      activity_regularizer=regularizers.l1(0.01))(x_newfc)

model = Model(inputs=model.input, outputs=[predictions_1, predictions_2])

adam = Adam(decay=1e-6)
model.compile(optimizer=adam, loss=['categorical_crossentropy',
'kullback_leibler_divergence'], metrics=['accuracy'], \
              loss_weights=[0.5, 0.5])

model.summary()

with CustomObjectScope({'relu6': keras.applications.mobilenet.relu6,
                        'DepthwiseConv2D':
                        keras.applications.mobilenet.DepthwiseConv2D}):
    model.load_weights('checkpoints_kullback/weights.10-0.53.hdf5')

img = np.array([img])
result = model.predict(img)

plt.imshow(cv2.cvtColor(img[0], cv2.COLOR_BGR2RGB))

img = img[0]
text_1 = "Eadible:      {:.4f}".format(result[0][0][0])
text_2 = "Not Eadible: {:.4f}".format(result[0][0][1])
```

```
img = cv2.putText(img, text_1,(5, 175),  cv2.FONT_HERSHEY_SIMPLEX, 0.4,
(255, 255, 0), 1)
img = cv2.putText(img, text_2,(5, 190),  cv2.FONT_HERSHEY_SIMPLEX, 0.4,
(255, 255, 0), 1)
cv2.imwrite("11.jpg", img)

print result
```