Санкт–Петербургский государственный университет

Дусмухамбетов Айдар Александрович

Выпускная квалификационная работа

Распознавание породы собак с помощью сверточных нейронных сетей

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2016 «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:
доцент, кафедра технологии программирования
к.т.н. Блеканов Иван Станиславович

Рецензент:
доцент, кафедра теории систем управления электрофизической аппаратурой и систем
к.ф.-м.н. Козынченко Владимир Александрович

Санкт-Петербург
2020 г.

Содержание

Введение

Темой данной работы является создание модели для распознавания породы собак на изображении с помощью сверточных нейронных сетей. В современном мире уже нельзя не брать во внимание важность задачи распознавания изображений, ведь возможность автоматического распознавания изображений компьютером дает новые возможности для современной техники, а также вносит вклад в развитие науки. Самоуправляемые машины, системы обнаружения лиц и других объектов, распознавание рукописного текста - все это возможно благодаря автоматическому распознаванию изображений, и с решением таких задач отлично справляются нейронные сети.

Сверточные нейронные сети в последние несколько лет получают все большее развитие в связи с колоссальными успехами на ежегодном соревновании по распознаванию изображений ImageNet[1]. Часто выходят статьи о классификации различных объектов и о компьютерном зрении в целом. Это повышает интерес к данной области и эта область не стоит на месте. Также интерес к нейронным сетям повышается вследствии значительного повышения вычислительных мощностей существующих компьютеров и использования графических карт для вычислений. Именно они ускоряют вычисления и позволяют обучать нейронные сети сложной структуры и большой глубины. Они же и показывают высокие резльтаты в различных задачах. Такое направление и получило название глубокое обучение(англ. deep learning).

Задача распознавания изображений часто требует отдельного подхода в зависимости от типов изображений и является очень обширной, что не позволяет рассмотреть ее полностью в одном исследовании. Следовательно, было принято решение рассмотреть на примере отдельную подзадачу распознавания изображений — распознавание породы собак. Это исследование **важно и актуально**, потому что его можно применить к решению схожих задач, где исследуется один объект, но нужно отличать много классов по мелким признакам на картинке. Также это исследование имеет **практическое применение** в современном мире для безопасности

людей. С помощью такой системы можно различать собак на бойцовые породы и нет. По закону к выгулу бойцовых пород предъявляются особые требования(намордник, поводок, бирка). Во многих городах на улицах расположены камеры и их можно использовать во благо. То есть, система могла бы находить нарушителей и делать уведомление в полицию.

Постановка задачи

Цель данной работы — создание эффективной модели для распознавания породы собак по двумерному изображению при помощи сверточных нейронных сетей, написание собственной архитектуры, сравнение имеющихся архитектур и оценка качества моделей. Для достижения цели поставлены следующие задачи:

- Сделать обзор предметной области
- Выбрать и подготовить набор данных для обучения
- Выбрать архитектуру сверточной нейронной сети и реализовать
- Обучить сеть и проверить ее работу на тестовом множестве

Обзор литературы

В процессе написания дипломной работы были использованы различные источники: научно-исследовательские статьи, учебная, а также научная литература, доклады с конференций и видеоматериалы. Все эти источники помогли и для изложения теоретического материала и для применения нейросетей на практике.

Курс «Deep Learning на пальцах»[2] дал базовое представление об области в целом и хорошо помог разобраться в сверточных нейронных сетях. Вместе с этим курсом также полезным было несколько глав из Стэнфордского курса «CS231n»[3]. Описание архитектур было взято из оригинальных публикаций от авторов этих архитектур. В таких статьях подробно описываются идеи, как это реализовывали и какой результат был достигнут.

Для понимания теоретических основ нейронных сетей и машинного обучения использовалась книга «Глубокое обучение» под редацкцией И. Бенджио, Я. Гудфеллоу, А. Курвилля[4]. Книга содержит математическую базу для понимания области, приемы, применяемые на практике, алгоритмы оптимизации и применение нейросетей в различных задачах.

Для реализации архитектур сверточных нейронных сетей использовался современный фреймворк глубокого обучения PyTorch. На официальном сайте[5] содержится подробная документация, инструкция по установке, принипы работы, а также демонстрируются важные функции API и особенности использования фреймворка.

Глава 1. Нейронные сети

1.1 Введение в нейронные сети

Нейронная сеть — это вычислительная система с взаимодействующими узлами, которая работает подобно нейронам в человеческом мозге. Используя некоторые алгоритмы, нейросеть способна распознавать скрытые шаблоны и зависимости в данных, классифицировать их, и, с течением времени, сеть непрерывно обучается и совершенствуется.

Термин «нейронная сеть» появился не в нашем веке, а еще в середине XX века. Первая нейронная сеть была задумана Уорреном Маккалохом и Уолтером Питтсом в 1943 году[6]. Они написали фундаментальную работу о том, как могут работать нейроны, и смоделировали свои идеи, создав простую нейронную сеть с использованием электрических цепей. Еще тогда они считали, что данная модель может обобщать, обучаться, то есть имеет схожесть с человеческим мозгом.

Идея структуры нейронной сети появилась в мире программирования благодаря биологии. Биологические нейронные сети устроены весьма сложно и человек без теоретической подготовки может не понять как работает естественная нейросеть.

Математическая модель биологического нейрона и по сей день еще не утверждена. Но представлено множество моделей, схожих с реальным нейроном и различающиеся вычислительной сложностью. На рисунке 1 изображена модель искусственного нейрона.

Такой нейрон имеет конечное количество входов($x_1, x_2, ..., x_n$), которые имеют веса($w_1, w_2, ..., w_n$). Изначально веса задаются случайными числами, а затем меняются в процессе обучения. Сумматор вычисляет взвешенную сумму входов и весов($\sum_{i=1}^{n} w_i * x_i$). Затем функция активации осуществляет некоторое преобразование над суммой и передает результат(одно число) на выход.

Так устроен один нейрон, и он уже может решать задачу классификации. Но на практике он малоэффективен и поэтому несколько нейронов объединяют в слои, а слои связывают между собой. Также входом для

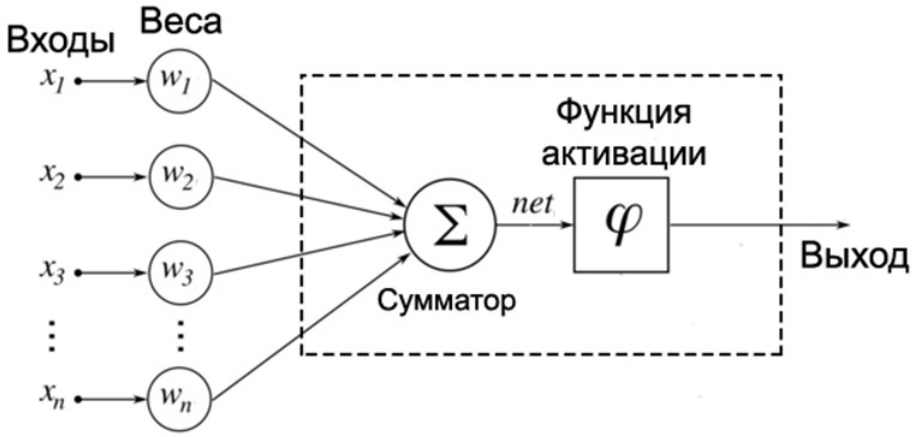

Рис. 1: Модель искусственного нейрона

одного нейрона может служить выход другого и наоборот.

Однослойная нейронная сеть — нейронная сеть, у которой входные сигналы сразу подаются на выходной слой, преобразующий сигналы и сразу же выдает ответ.

Многослойная нейронная сеть — нейронная сеть, которая состоит из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Такие сети имеют гораздо большие возможности, чем однослойные нейросети, но также требуют больше вычислительных ресурсов.

1.2 Функции активации

Функция активации предназначена для вычисления выходного сигнала нейрона и применяется к взвешенной сумме входов(s).

На данный момент существует весьма большое количество функций активации, и выбор конкретной функции зависит от типа нейрона и от поставленной задачи. Ниже представлен список одних из самых популярных:

- Ступенчатая функция: если $s > threshold$, то $f(s) = 1$, а иначе 0;
- Сигмоидная функция: $f(s) = \frac{1}{1+e^{-s}}$;
- Функция гиперболического тангенса: $\frac{e^s - e^{-s}}{e^s + e^{-s}}$;
- Функция ReLU: $f(s) = max(0, s)$;

1.3 Обучение

Обучение нейронной сети представляет собой подбор наилучшего множества весов сети таким образом, что входной сигнал протекает по сети и превращается в правильный ответ на задачу.

Это определение обучения нейронной сети подходит и для биологических нейросетей. Человеческий мозг состоит из большого количества нейросетей, которые связаны друг с другом, и каждая из нейросетей в отдельности состоит из нейронов. Мозг обучается благодаря изменению синапсов. Синапсы могут усиливать или ослаблять входной сигнал.

Если обучать сеть только на одном входном сигнале, то она просто «выучит» правильный ответ и всегда будет правильно на него реагировать, но как только подать ей на вход немного измененный сигнал, то можем получить что-то совсем неожиданное. Нам бы хотелось, чтобы сеть обобщала какие-то признаки и решала задачу для различных входных данных. Обучающие выборки созданы именно для этой цели.

Обучающая выборка — конечный набор входных сигналов(это могут быть события, объекты) иногда вместе с правильными ответами(выходными сигналами), на которых и происходит обучение сети.

Когда сеть выдает правильные результаты для необходимого количества примеров из обучающей выборки, то есть когда заканчивается процесс обучения, то ее уже можно использовать для решения задачи. Однако перед использованием нейронной сети, следует произвести оценку качества ее работы на тестовом множестве.

Тестовое множество — конечный набор данных(входных сигналов), который используется для оценки качества работы нейросети.

Обучение нейросети обычно можно разделить на 2 метода: *обучение с учителем* и *обучение без учителя*.

Обучение с учителем подразумевает под собой наличие известных входных и выходных данных(векторов сети). Каждому примеру из датасета соответствует правильный ответ. На этапе обучения сеть учится давать правильные ответы и подбирает лучшие параметры. Из-за способности к обобщению нейросеть может получать новые результаты для входных век-

торов, не встречающихся в обучающей выборке.

В алгоритме *обучения без учителя* имеются только входные вектора, но нет правильных ответов или решений. Сеть должна сама обучиться и понять, как интерпретировать данные. Хотелось бы здесь тоже получить лучший результат, но что является «лучшим», решает алгоритм обучения.

1.4 Обратное распространение ошибки

Алгоритм обратного распространения ошибки обычно состоит из трех основных фаз:

1. Прямой проход или распространение входного сигнала по сети до последнего слоя и получение ответа
2. Обратный проход или распространение ошибки сети от последнего слоя к первому
3. Настройка весов сети

Разберем данный алгоритм для многослойной нейронной сети.

Прямой проход.

1. Слою k на вход подается вектор x^k и далее применяется функция активация($func$) для всех нейронов i слоя k:

$$y_i^k = func(x_i^k) \tag{1}$$

2. Вычислить входной вектор для следующего слоя $k+1$:

$$x_i^{k+1} = \sum_j w_{ji}^k * y_j^k \tag{2}$$

где w_{ji}^k - вес связи между j-ым нейроном слоя k и i-ым нейроном слоя $k+1$

3. Шаги 1,2 и составляют прямой проход по сети, их нужно выполнять до последнего слоя(K). Таким образом, мы получим выходной вектор.

4. Рассчитать ошибку сети $L(y^K)$ для выбранной функции потерь(англ. Loss function).

Обратный проход.

Когда мы посчитали ошибку выхода(последнего слоя) сети, то необходимо настроить параметры всей сети, чтобы на следующей итерации ошибка уменьшилась, то есть сеть выдала ответ, максимально близкий к эталонному. Для достижения данной цели нужно вычислить производную функции потерь по всем весам сети. Здесь мы можем применить *правило дифференцирования сложной функции*, также известное как *цепное правило*, которое имеет следующий вид:

$$\frac{dy}{dt} = \frac{dy}{dx} * \frac{dx}{dt} \tag{3}$$

где $y = y(x)$, а $x = x(t)$.

Используя (3) и учитывая (2):

$$\frac{dL}{dw_{ij}^k} = \frac{dL}{dx_j^{k+1}} \frac{dx_j^{k+1}}{dw_{ij}^k} = \frac{dL}{dx_j^{k+1}} * y_i^k \tag{4}$$

Все y_i^k известны из прямого прохода, так как мы считали выходы для каждого слоя. Значит нам остается посчитать произоводные функции потерь по входным сигналам(векторам) слоев.

Применяя (1) и цепное правило, имеем:

$$\frac{dL}{dx_j^k} = \frac{dL}{dy_j^k} \frac{dy_j^k}{dx_j^k} = \frac{dL}{dy_j^k} \frac{d}{dx_j^k} func(x_j^k) = \frac{dL}{dy_j^k} func'(x_j^k) \tag{5}$$

Следовательно, для последнего слоя сети K:

$$\frac{dL}{dy_i^K} = \frac{d}{dy_i^K} L(y^K) \tag{6}$$

А для произвольного слоя k необходимо применить правило диффе-

ренцирования сложной функции еще раз:

$$\frac{dL}{dy_i^k} = \sum_j \frac{dL}{dx_j^{k+1}} \frac{dx_j^{k+1}}{dy_i^k} = \sum_j \frac{dL}{dx_j^{k+1}} w_{ij} \tag{7}$$

Учитывая вышесказанное, можно сформулировать алгоритм для обратного прохода.

1. Вычислить производную функции потерь по выходу сети на последнем слое, используя (6);
2. Вычислить производные функции потерь по входным векторам слоя k(слой с известной ошибкой) по (5);
3. Передать ошибку слою $k-1$ по (7);
4. Повторить шаги 2, 3 для всех слоев от выходного до входного;
5. Вычислить производные функции потерь по всем весам сети используя (4);

Таким образом, зная производные функции потерь по всем весам сети, можно переходить к их настройке, используя подходящий алгоритм обучения.

Глава 2. Сверточные нейронные сети

Свёрточная нейронная сеть (СНС, англ. Convolutional Neural Network) — важный класс среди глубоких нейронных сетей. Впервые сеть была предложена Яном Лекуном и нацелена на эффективную классификация и распознавание изображений. Свёрточные сети весьма точно распознают объекты на изображениях, к тому же они устойчивы к изменениям масштаба, небольшим смещениям и поворотам объектов на входных данных. И на сегодняшний день для извлечения признаков из изображений и видео эффективным методом является использование СНС. Также на ежегодном соревновании по распознаванию изображений ImageNet, сверточные нейросети занимают лидирующие позиции.

2.1 Структура

Сверточные нейронные сети обычно имеют слои следующих типов[11]:

- Сверточный слой (англ. Convolution layer): все нейроны слоя связаны только с частью нейронов предыдущего слоя.
- Пулинговый слой (англ. Pooling, Subsampling layer): уменьшение размерности предыдущего слоя сети и выделение наиболее значимых признаков.
- Полносвязный слой (англ. Fully-connected layer): может являться последним слоем в нейронной сети или представлять собой скрытый слой.

Обычно после сверточных и полносвязных слоев применяется функция активации. Она преобразует полученный сигнал нейрона в выходной сигнал.

На вход сверточной нейросети часто подается цветное(трехканальное) изображение. То есть сети подаются три слоя изображения (R-, G-, B-каналы изображения). В компьютере изображения представляются в виде пикселей, а, в свою очередь, каждый пиксель – это значения интенсивности

для соответствующих каналов. Интенсивность каждого канала определяется целым числом от 0 до 255. На рисунке 2 показано как изображение представляется в компьютере.

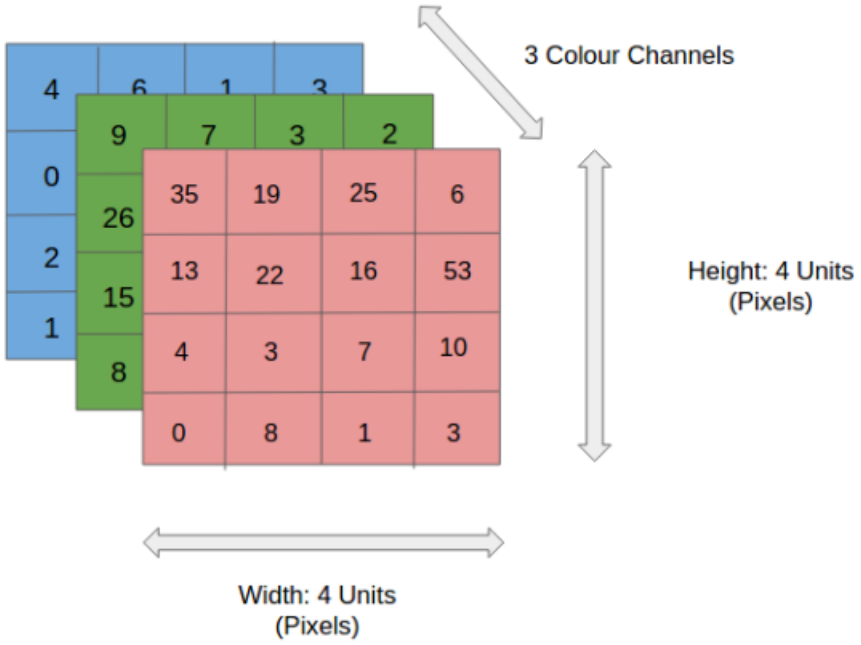

Рис. 2: Представление изображения для сверточного слоя

Ниже более подробно описываются все перечисленные типы слоев.

Сверточный слой. Сверточный слой нейронной сети является самым важным и предназначен для выделения признаков из изображения, которые в дальнейшем на более глубоких слоях, используются для понимания более сложных признаков и, в итоге, определяют класс распознаваемого объекта. Главная характеристика данного слоя это так называемые фильтры – обычно двухмерные или трехмерные матрицы весов. Операция свертки - это получение выходного сигнала нейрона сверточного слоя. Она имеет схожесть с операцией фильтрации изображения: каждое значение сигнала нейрона предыдущего слоя, которое располагается в соответствующей ядру фильтра конкретной области, умножается на соответствующее значение ядра фильтра (в СНС значения ядра фильтра называются весами связей нейронов сверточного слоя); и результатом фильтрации является сумма всех полученных произведений. Операция свертки сверточного слоя показана на рисунке 3.

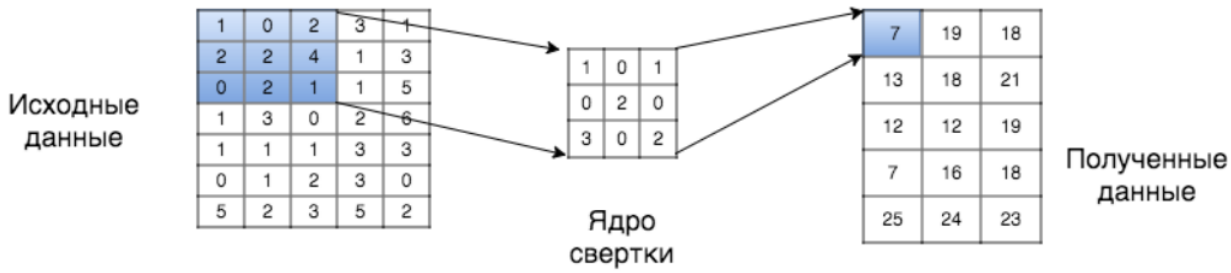

Рис. 3: Работа сверточного слоя

Другие важные параметры свёрточного слоя сети это количество и тип используемых фильтров. Тип фильтра используется для выделения определенных признаков изображения, к примеру, выделение прямых линий под конкретным углом (в процессе обучения сети веса фильтров настраиваются, тем самым влияя на характер выделяемых признаков). Обычно используются квадратные фильтры, размером 3x3 или 5x5. Глубина сверточного слоя определяется количеством используемых фильтров. Также у сверточного слоя есть и другие параметры:

S – шаг свертки, смещение (stride) – определяет на сколько позиций перемещается фильтр для формирования сигнала следующего нейрона этого слоя.

P – добавление нулей (padding) – добавление нулей по краям предыдущего слоя для управления размерностью свёрточного слоя. С помощью этого параметра размер предыдущего и текущего слоя можно сделать равными.

Размерность сверточного слоя определяется его параметрами и размерами предыдущего слоя сети по формулам 8 и 9:

$$H = \frac{H_p - F + 2 * P}{S} + 1 \tag{8}$$

$$W = \frac{W_p - F + 2 * P}{S} + 1 \tag{9}$$

H и W - высота и ширина текущего слоя соответственно

H_p и W_p - высота и ширина предыдущего слоя соответственно

F - размер ядра свертки(фильтра)

P - размер падинга

S - шаг свертки или смещение фильтра

Пулинговый слой(подвыборочный слой, субдискретизирующий слой). Пулинговый слой нужен для снижения размерности изображения и для выделение наиболее значимых признаков. Исходное изображение делится на квадратные блоки размером MxM и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (англ. max pooling) или среднего (англ. average pooling). Этот слой не имеет обучаемых параметров. Основные цели пулингового слоя:

- уменьшение изображения;
- извлечения доминирующих признаков;
- ускорение вычислений.

Процесс пулингового слоя, а именно операции max pooling показан на рисунке 4.

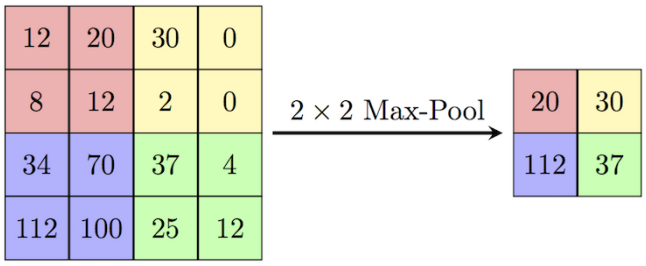

Рис. 4: Пулинговый слой с операцией максимума

Размерность пулингового слоя определяется размером блока, смещением и размерами предыдущего слоя сети по формулам 10 и 11:

$$H = \frac{H_p - M}{S} + 1 \tag{10}$$

$$W = \frac{W_p - M}{S} + 1 \tag{11}$$

H и W - высота и ширина текущего слоя соответственно

H_p и W_p - высота и ширина предыдущего слоя соответственно

M - размер квадратного блока

S - смещение(с каким шагом формируются блоки)

Полносвязный слой. После чередующихся слоев свертки и пулинга стоит один или несколько полносвязных слоев. Данные слои выполняют роль классификаторов, то есть, после выделения различных признаков предыдущими слоями, ее результаты передаются на полносвязные слои, а на них уже происходит предсказание результата. Так мы переходим от нескольких двумерных матриц к одномерному вектору(каждая матрица растягивается в вектор и все вектора конкатенируются). Обычно результатом данного слоя является вероятность принадлежности входного изображения конкретному классу. Схема работы данного слоя представлена на рисунке 5.

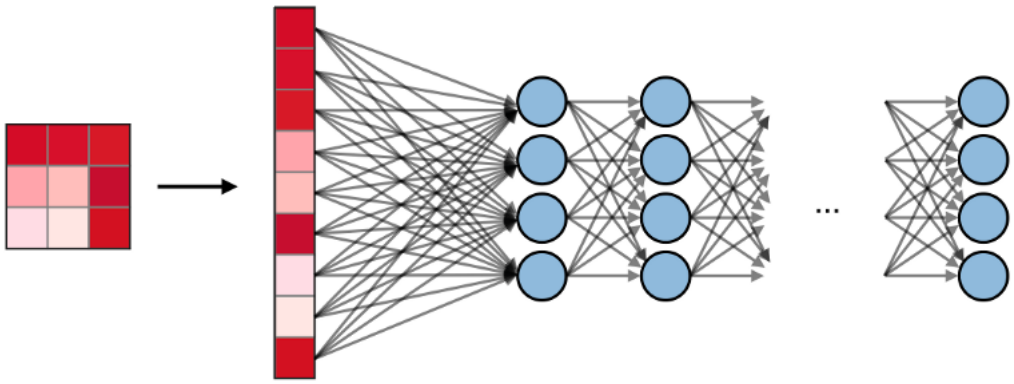

Рис. 5: Схема работы полносвязного слоя

Общая схема построения сверточной сети.

Имея в наборе готовые строительные блоки, можно собирать различные архитектуры, добавляя слой за слоем. Для классификации изображений чаще всего используется следующая схема(рис. 6).

2.2 LeNet

Ян Лекун впервые предложил сверточную нейронную сеть в виде модели LeNet[7], схема которой представлена на рисунке 7. Архитектура LeNet стала фундаментальной для глубокого обучения, Она была важна с

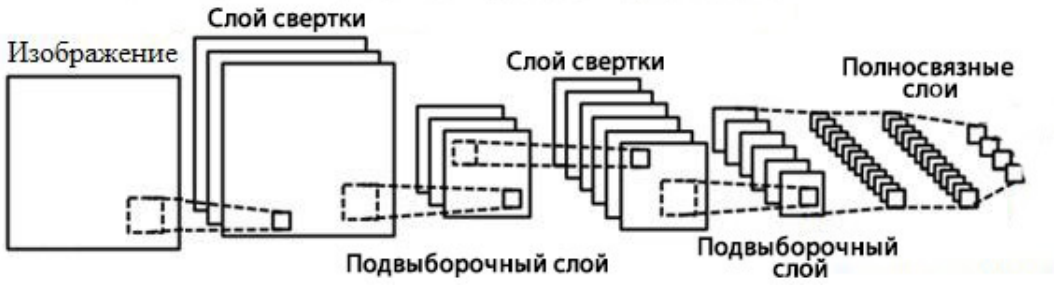

Рис. 6: Общая схема сверточной сети

точки зрения распределения свойств на всем изображении. Свёртки с обучаемыми параметрами позволяли хорошо извлекать одинаковые свойства из разных мест. К тому же эта сеть имела небольшое количество параметров. Это была первая модель, которая содержала чередующиеся дважды сверточные слои и слои пулинга, а также три полносвязных слоя.

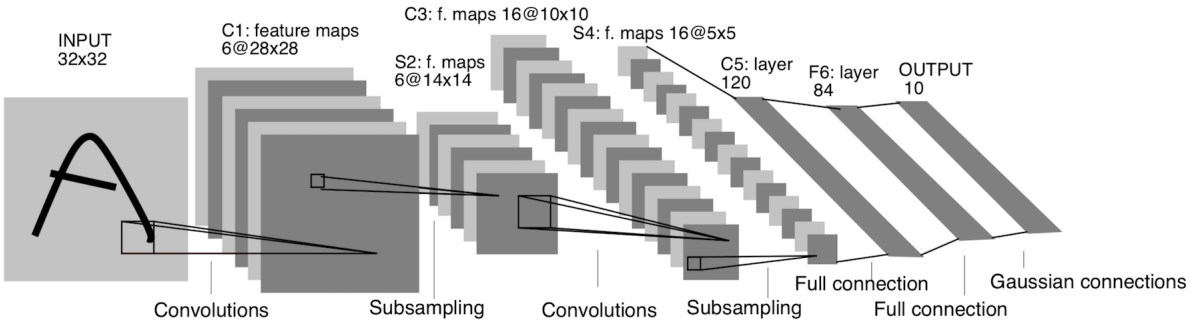

Рис. 7: Архитектура LeNet5

Особенности LeNet:

- На вход поступает черно-белое изображение 32x32
- Размер свертки 5x5
- Пулинг среднего значения(average pooling) 2x2
- Функция активации - гиперболический тангенс(tanh(x))
- Финальная классификация в виде многослойной нейронной сети

2.3 Советы при обучении

Дропаут

Одной из проблем обучения глубоких нейросетей является переобучение. Термин «dropout»[13] обозначает исключение конкретного процента (например 20%) случайных нейронов на разных эпохах во время обучения нейросети. Следовательно, на каждой эпохе мы обучаем подсеть с меньшим количеством нейронов и тем самым исключаем сильные зависимости между некоторыми нейронами. То есть, веса нейронов изменяются, несмотря на ошибки других нейронов. Таким образом, эта техника позволяет улучшить качество предсказания, а также ускоряет процесс обучения.

Аугментация

При глубоком обучении можно столкнуться с ситуацией, когда набор данных имеет небольшой размер. Но чтобы модель имела хорошую обобщающую способность, необходимо иметь больше данных, а именно хорошо помогают различные вариации. Значит мы хотим увеличить размер исходного датасета искусственным образом, и для этого на помощь приходит аугментация данных. Следующие способы аугментации картинок являются самыми популярными: поворот на определенный угол, отображение по вертикали или горизонтали, вырезание части изображения, добавление шума, изменение контрастности и яркости. Это хорошо помогает в данной задаче, так как нам не важно, в какой части изображения или в каком положении находится собака. На рисунке 8 можно увидеть несколько видов аугментаций, примененных к нашему датасету.

2.4 Архитектуры

2.4.1 AlexNet

В 2012 году на свет появилась архитектура AlexNet[8], созданная Алексом Крижевским. Это была важная работа для развития СНС, в которой авторы впервые на тот момент использовали глубокие сверточные нейросети с общей глубиной в 8 слоев (5 сверточных и 3 полносвязных слоя). Архитектура Крижевского чем-то может напоминать сеть LeNet, но

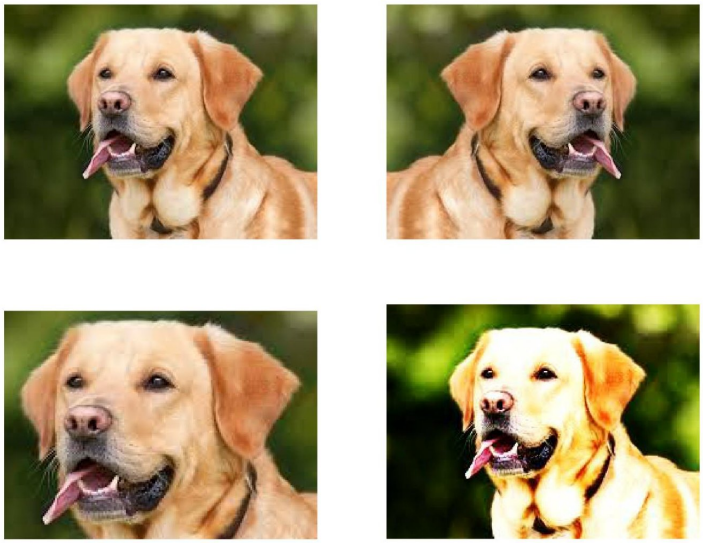

Рис. 8: Примеры аугментации

у AlexNet больше фильтров и вложенных сверточных слоев. Сеть включала в себя, помимо сверток и пулингов, аугментацию данных, дропаут и функцию активации ReLU. Сеть обучалась на двух графических процессорах в течение недели. Архитектура AlexNet представлена на рисунке 9.

Данная сеть в 2012 году выиграла конкурс по распознаванию изображений ImageNet с большим отрывом(с количеством ошибок 15,3%, а у второго места 26,2%)[12].

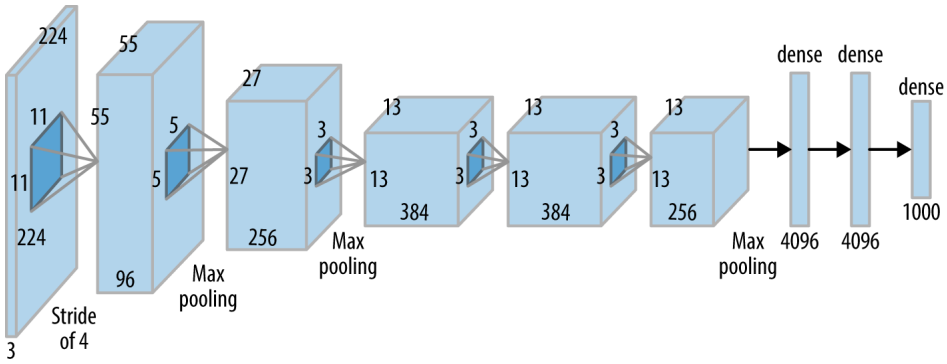

Рис. 9: Архитектура AlexNet

2.4.2 ResNet

В 2015 году произошла очередная революция — на свет появилась глубокая сверточная нейронная сеть ResNet[9]. Почти все нейронные сети

в качестве метода обучения используют метод обратного распространения ошибки. При обратном проходе применяется градиентный спуск с целью уменьшения функции ошибки. При увеличении количества слоев, то есть глубины нейросети, частая проблема — это затухание градиента при обратном проходе. То есть, если градиент имеет небольшое значение в конце сети, то он может принять очень малое значение, когда достигнет начала сети. Для примера $0.5^{50} \approx 8 * 10^{-16}$. Сеть перестает обучаться. То есть не имеет смысла просто увеличивать количество слоев. В основе данной архитектуры лежат блоки с соединениями быстрого доступа(англ. shortcut connections), изображенные на рисунке 10.

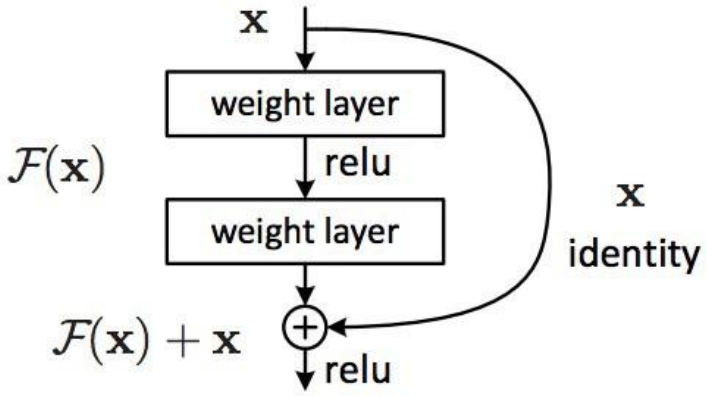

Рис. 10: Блок с shortcut connections

Схему данного блока можно объяснить так: теперь целевая функция имеет вид $G(x) = F(x) + x$. Такие блоки добавляют тождественное преобразование в сеть. Следовательно, при обратном проходе получаем $\frac{dG(x)}{dx} = \frac{dF(x)}{dx} + \frac{dx}{dx} = \frac{dF(x)}{dx} + 1$. Теперь затухание градиента не произойдет, потому что слагаемое единица позволяет «протекать» градиенту вплоть до первых слоев.

Ниже на рисунке 11 представлена сеть ResNet18 - одна из вариаций архитектуры ResNet с 18 слоями и применение блоков с shortcut connections в сети.

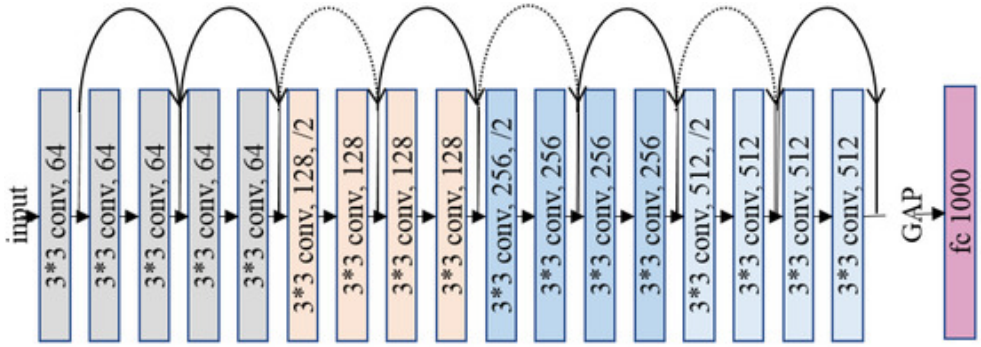

Рис. 11: Схема сети ResNet18

Глава 3. Практика

3.1 Описание базы данных

Чтобы протестировать архитектуры сверточных нейронных сетей была использована база данных **Stanford Dogs Dataset**[10], содержащая изображения разных пород собак и соответствующие им метки. Датасет состоит из 20580 изображений для 120 пород собак. Размер каждого изображения нефиксирован и для обучения его нужно привести к одному размеру(224x224).

3.2 PyTorch

PyTorch — современный фреймворк глубокого обучения для языка Python. PyTorch позволяет гибко работать с нейронными сетями. Для реализации обучения нейросети необходимо написать код тренировки модели. Создание модели в PyTorch не требует больших временных затрат. Также важными преимуществами является легкая интеграция с графическим процессором для вычислений, наличие предварительно обученных моделей и готовых модульных частей, которые легко комбинировать.

Код обучения модели выглядит так:

```
loss_train_history = []
loss_val_history = []
acc_train_history = []
acc_val_history = []

has_gpu = torch.cuda.is_available()
dataloaders = {'train': train_dataloader, 'val': val_dataloader}
def train_model(model, optimizer, criterion, num_epochs=25):
    best_acc = 0.0

    for epoch in range(num_epochs):
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train(True)  # model in training mode
            else:
                model.train(False)  # model in evaluate mode
            current_loss = 0.0
            current_corrects = 0
```

```
19
20              data_loader = dataloaders[phase]
21              for data in data_loader:
22                  # using dataset
23                  inputs, labels = data
24                  if has_gpu:
25                      inputs = Variable(inputs.cuda(),)
26                      labels = Variable(labels.cuda())
27                  else:
28                      inputs = Variable(inputs)
29                      labels = Variable(labels)
30
31                  optimizer.zero_grad()
32                  # forward pass
33                  outputs = model(inputs)
34                  _, preds = torch.max(outputs.data, 1)
35                  loss = criterion(outputs, labels)
36                  # backward pass
37                  if phase == 'train':
38                      loss.backward()
39                      optimizer.step()
40                  # data for graphics
41                  current_loss += loss.data[0] * inputs.size(0)
42                  current_corrects += torch.sum(preds == labels.data)
43
44              epoch_loss = current_loss / len(data_loader.dataset)
45              epoch_acc = current_corrects / len(data_loader.dataset)
46              if phase == 'train':
47                  loss_train_history.append(epoch_loss)
48                  acc_train_history.append(epoch_acc)
49              else:
50                  loss_val_history.append(epoch_loss)
51                  acc_val_history.append(epoch_acc)
52
53              if phase == 'val' and epoch_acc > best_acc:
54                  best_acc = epoch_acc
55
56      return model
```

3.3 Transfer Learning

Для того чтобы воспользоваться мощными и уже предобученными моделями для нашей задачи, применим Transfer Learning. Transfer Learning — это одна из техник обучения нейронных сетей, которая позволяет ис-

пользовать возможности некоторой модели, обученной для одной задачи, для решения похожей задачи. Эти самые предобученные модели обучались на огромном количестве данных и накопили некоторый опыт и обучились находить важные признаки. Чтобы использовать такую мощную модель, нам предстоит дообучить ее на нашем датасете. Добавляем еще один полносвязный слой с выходными 120 нейронами. Это и будут предсказанные вероятности для каждой породы.

Глава 4. Результаты

ResNet18. Базовая вариация сети ResNet с 18 слоями. За 15 эпох достиг точности 0.669. На рисунках 12, 13 представлены графики ошибки и точности на тренировочном и валидационном множествах.

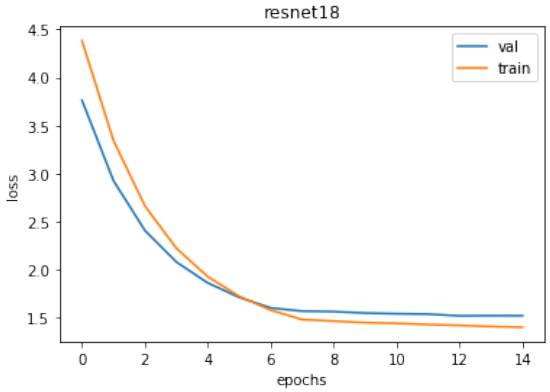

Рис. 12: Ошибка модели

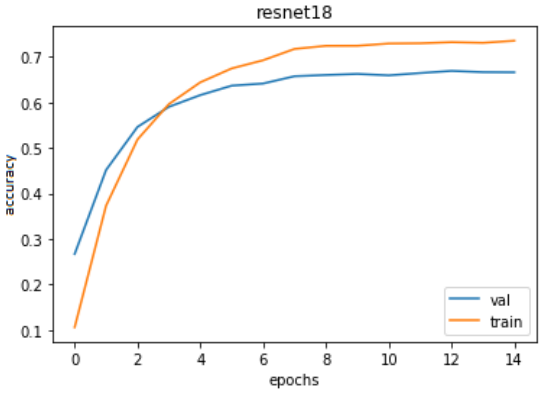

Рис. 13: Точность предсказания

MyNet. В процессе работы также была написана своя архитектура, обученная с нуля. Архитектура построена наподобие AlexNet, но не такой большой глубины из-за нехватки вычислительных ресурсов. Следуя популярным советам, здесь мы чередуем свертки и пулинги, тем самым сжимаем изображение и увеличиваем количество каналов. После этого дропаут с вероятностью 20% и полносвязный слой с выходными 120 нейронами.

- nn.Conv2d(3, 16, 3) → nn.MaxPool2d(2, 2)

  nn.Conv2d(16, 32, 3) → nn.MaxPool2d(2, 2)

  nn.Conv2d(32, 64, 3) → nn.MaxPool2d(2, 2)

  nn.Conv2d(64, 128, 3) → nn.MaxPool2d(2, 2)

  nn.Conv2d(128, 256, 3) → nn.MaxPool2d(2, 2)

- nn.Dropout(0.2)

- nn.Linear(256 * 6 * 6, 120)

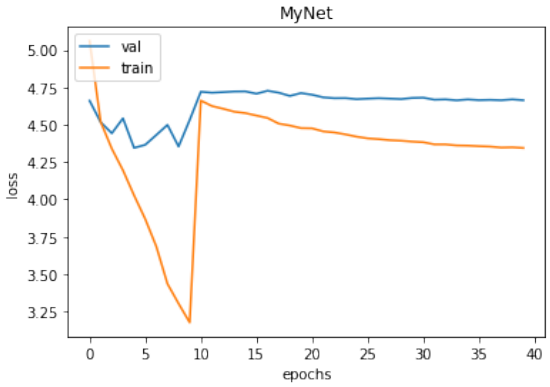

Рис. 14: Ошибка модели

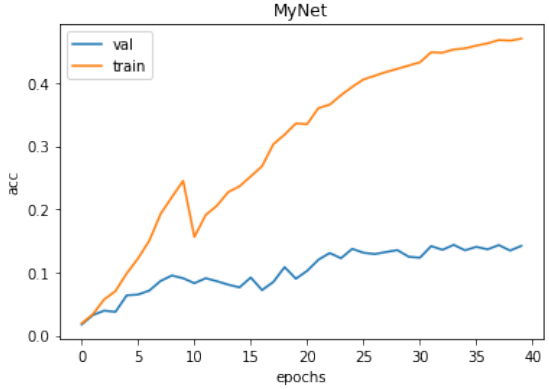

Рис. 15: Точность предсказания

Точность после обучения=0.134. На рисунках 14, 15 представлены графики ошибки и точности на тренировочном и валидационном множествах. Точность на тренировочной выборке растет быстрее, чем на валидационной. Сеть имеет малую обобщающую способность и предсказывает хуже предобученной на другой задаче сети ResNet18.

ResNet152. А теперь применим более глубокую вариацию сети ResNet. За 15 эпох достиг точности 0.829. На рисунках 16, 17 представлены графики ошибки и точности на тренировочном и валидационном множествах.

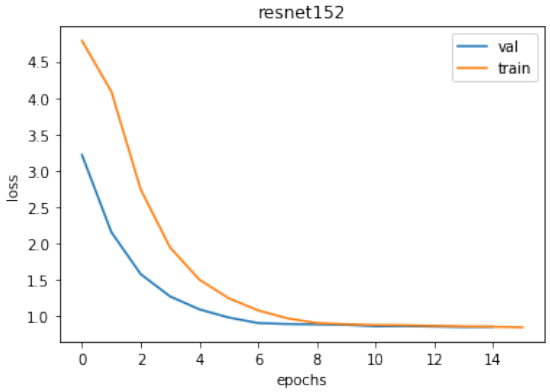

Рис. 16: Ошибка модели

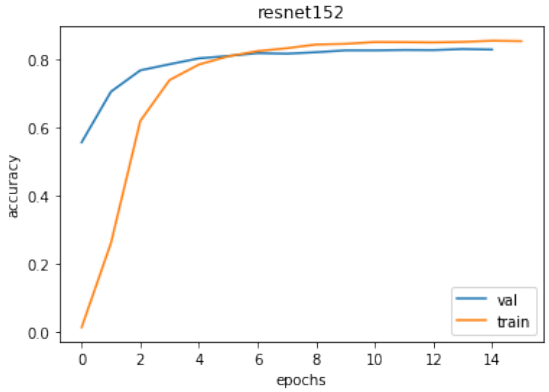

Рис. 17: Точность предсказания

Из графиков видно, что сеть обучается и точность на тренировочном и валидационном множестве не сильно отличается. Сеть может хорошо распознавать породы.

Выводы

Сеть, написанная с нуля, обучается плохо и долго. Популярные методики построения архитектур сверточных нейросетей не помогли создать модель с высокой точностью предсказания. Модель с такой точностью неприменима в данной задаче.

Предобученная же сеть уже имела представления о каких-то признаках на картинке. Обучая ее на нашей задаче, она понимает новые признаки, она научилась различать мелкие детали в различных породах. Человек отличает породу по следующим признакам: окрас и длина шерсти; форма морды, носа, хвоста, ушей, лап и т.д. Вероятно, лучшая модель и научилась понимать некоторые из перечисленных признаков и дала такой хороший результат. Ее точность составила 82%. Безусловно, точность этой модели хотелось бы повысить для применения на практике, но такой результат уже можно принимать во внимание и тестировать модель в реальных условиях.

Ниже на рисунках 18, 19 изображены примеры, на которых модель дает неверное предсказание. На рисунке 18 модель предсказала породу ши-тцу, а это был керн-терьер. На рисунке 19 модель предсказала породу тибетский терьер, а это был ши-тцу.

Рис. 18: Керн-терьер

Рис. 19: Ши-тцу

Заключение

В данной выпускной квалификационной работе были достигнуты следующие результаты:

- Сделан обзор предметной области
- Сделан обзор на используемые архитектуры
- Выбран и подготовлен набор данных для обучения
- Выбрана архитектура сверточной нейронной сети и реализована
- Сеть обучена и ее работа проверена на тестовом множестве
- Представлены графики для сравнения результов используемых архитектур

Для решения задачи распознавания породы собак была предложена модель, основанная на ResNet152. Модель была дообучена на базе данных Stanford Dogs Dataset с использованием графического процессора. Сверточная нейронная сеть достигла хорошей точности, учитывая, что решалась задача многоклассовой классификации. Ее можно пробовать улучшать и применять для решения схожих задач, где исследуется один объект, но имеется много разных видов. Также можно задуматься о применении данной модели в реальной жизни для разделения собак на бойцовые и не бойцовые породы и сделать окружающий мир безопаснее.

Список литературы

[1] ImageNet Overview. http://image-net.org/about-overview

[2] Семен Козлов. Deep Learning на пальцах. https://dlcourse.ai/

[3] Stanford University. CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/

[4] Bengio Y., Goodfellow I., Courville A . Deep Learning. - MIT Press, book in preparation

[5] PyTorch. Official site. https://pytorch.org/

[6] McCulloch W. S., Pitts W. A logical calculus of the ideas immanent in nervous activity //The bulletin of mathematical biophysics. - 1943. - T. 5. - №. 4. - pp. 115-133

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. // Proceedings of the IEEE, November 1998

[8] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, June 2017, pp. 84–90

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition // ArXiv. —2015. https://arxiv.org/pdf/1512.03385.pdf

[10] Stanford Dogs Dataset by Stanford University. http://vision.stanford.edu/aditya86/ImageNetDogs

[11] Convolutional Neural Networks (CNNs / ConvNets). http://cs231n.github.io/convolutional-networks/#conv

[12] A. Krizhevsky, I. Sutskever, G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Proceedings of Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012, Pp. 1097-1105

[13] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929–1958