ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики

Лапин Александр, группа БИВ204

Разработка сверточной нейронной сети для распознавания изображений героев мультсериала “Симпсоны”.

Курсовая работа по дисциплине «Алгоритмизация и программирование»

по направлению 09.03.01 Информатика и вычислительная техника

студентов образовательной программы бакалавриата «Информатика и вычислительная техника»

Студент		**Руководитель**	
/	А.А. Лапин	/	ассист. Е.В. Лежнев
подпись	*И.О. Фамилия*	*подпись*	*должность, звание, И.О. Фамилия*

Оглавление

Аннотация

Предметом изучения курсовой работы является сверточная нейронная сеть, способная распознавать и классифицировать изображения, а также техники для её оптимизации.

Цель курсовой работы – разработка приложения для распознавания изображений, на базе обученной и оптимизированной модели.

Annotation

The subject of the course work is a convolutional neural network capable of recognizing and classifying images, as well as techniques for its optimization.

The purpose of the course work is to develop an application for image recognition, based on a trained and optimized model.

Введение

Что такое искусственный интеллект? Большинство людей, когда слышит данное словосочетание представляет себе антропоморфного робота, который может разговаривать, думать, как человек. Давайте попробуем глубже разобраться, что же это такое. Что же называют искусственным интеллектом, в наиболее близких к технологиям и науке сообществам. С одной стороны, мы можем рассматривать искусственный интеллект как некоторые фундаментальные в научном направлении концепции и целью этого научного направления мы можем видеть, что такая цель совсем далёкое это понять природу интеллекта. Одновременно получается понять природу человека, потому что, если мы понимаем, мы можем создать машину, которая думает, чувствует, как человек, то мы можем детальнее разобраться, что человек представляет из себя. Это часть фундаментальных исследований, которые носят своеобразно философский характер. С другой стороны, это прикладная задача, человечество хочет создать помощника, который будет помогать выполнять разные задачи. И, наконец, это набор технологий, который позволяет решать задачи, выполнение которых ранее требовало человеческого интеллекта, например, распознавание лиц, машинный перевод или умный дом.

Подходы к построению искусственного интеллекта

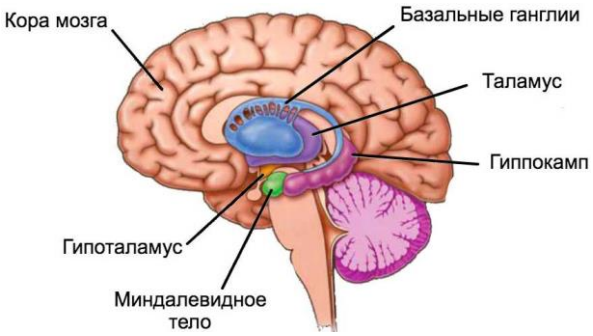
Выделяют два основных подхода. Первый подход базируется на природе мозга человека, а именно на строении мозга, в состав которого входит сложная система нейронов. И эта система позволяет решать сложные задачи. Другой подход идет от поведения, мы можем исследовать, как человек будет решать те или иные задачи и смоделировать при помощи модели рассуждений.

История развития искусственного интеллекта

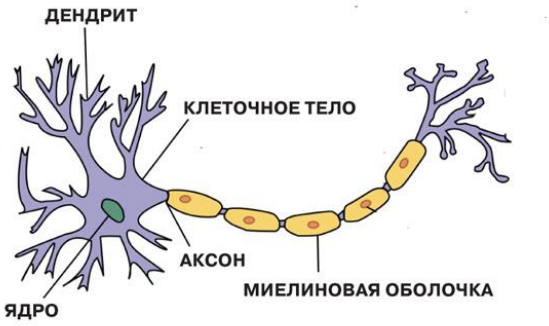
Искусственный интеллект как направление исследований возник в середине прошлого века, когда начали появляться компьютеры, позволяющие строить и решать сложные математические модели. В то же время ученые имели уже достаточно представлений, как устроен мозг и начали проводиться исследования подходов. В 1956 году появилось понятие искусственного интеллекта, как объекта исследования.

Искусственные нейронные сети и глубокое обучение

Чтобы разобраться, что такое нейронные сети, нужно понять, как работает мозг существа. Рассмотрим часть мозга, которая называется Гиппокамп, эта часть мозга связана с эпизодической памятью, то есть памятью о конкретных событиях.



В Гиппокампе можно выделить такую часть, которая называется Зубчатые Фазцы. Это область и является нейросетью, потому что состоит из нейронов.

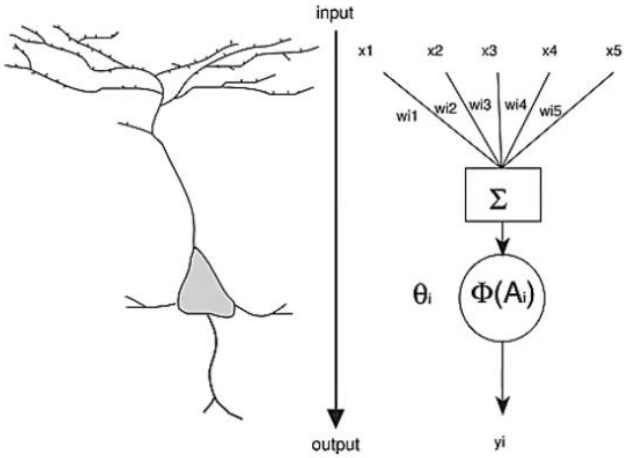


Нейрон состоит из 3 основных частей: дендритного дерева, которое крепится ко второй важной в нейроне части – ядру. От ядра уходит аксон, отросток, при помощи которого нейрон посылает свои сигналы, в то время как при помощи дендритного дерева эти сигналы принимаются. Нейроны активируются посредством впрыскивания специального вещества в синаптическую щель.

- Активность нейрона определяется преобразованием взвешенного суммарного воздействия на него.
- Воздействия могут быть активирующими (положительные веса) или тормозными (отрицательные веса).

$$a_j = \sum_{j=1}^{N} w_{ij} x_j$$

$$y_i = \Phi(a_i) = \Phi(\sum_{j=1}^{N} w_{ij}x_j - \vartheta_i)$$



На рисунке изображено дендритное дерево, от которого вниз отходит аксон. Из этого можно сформулировать простейшую математическую модель, также изображенную на рисунке. X1, x2, x3, x4, x5 – концентрация тех веществ, которые активируют нейрон, тогда обозначим концентрацию рецепторов в синаптической щели (воспринимающих вещество для активации), как w1, w2, w3, w4, w5. Это значит, что при перемножении этих коэффициентов мы можем получить оценку влияния, которую данный вход вносит в активацию нейрона. После сложения всех этих воздействий появляется возможность сравнить эту сумму с пороговой величиной активации нейрона.

Построение и обучение нейронной сети

Введем понятия для дальнейшего использования:

- Слой – структурная единица нейронной сети, которая как правило включает в себя некоторое линейное преобразование, а также функцию активацию.
    - Линейное преобразование f(x) = wx + b
    - Нелинейное преобразование f(x) = σ(x)
    - Входной слой (input layer) и выходной слой (output layer)
- Функция активации – функции, которые применяются к выходным значениям с линейных преобразований.

Чтобы сформулировать из нейронов сеть, необходимо соединить нейроны весами. Однако изначально, мы не знаем, какие веса соответствуют каждому соединению (если бы мы их знали, задача уже была бы решена). Выходит, данные веса нужно каким-либо образом настроить. Задача обучения в

нейронных сетях подразумевает создание архитектуры расположения и связей нейронов. А также нахождение и корректировка весов.

- Нейросеть учится на примерах правильных ответов – обучающая выборка.
- Обучение нейронной сети происходит за счет изменения весов.
- Для проверки того, насколько хорошо прошло обучение, используются примеры, которые не были использованы при обучении – тестовая выборка.

Концепция ошибки прямого распространения

Прямое распространение – это вычисление ответа нейронной сети посредством последовательного вычисления значений каждого нейрона, на базе входных данных, с заранее известными весами. Конечный ответ нейронной сети используется в функции ошибки, для последующей корректировки весов нейронов. Рассмотрим наиболее распространенные функции ошибки:
где $y^{(i)}$- ответ нейронной сети (значение выходного слоя); $\hat{y}^{(i)}$- значение метки, истинного предполагаемого значения предсказания;
1) Cross-Entropy – энтропия

$$CE = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} * \log\left(\hat{y}^{(i)}\right) = -(y * \log(\hat{y}) + (1-y) * \log(1-\hat{y})$$

2) MSE (Mean Squared Error) – квадрат средней ошибки

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$$

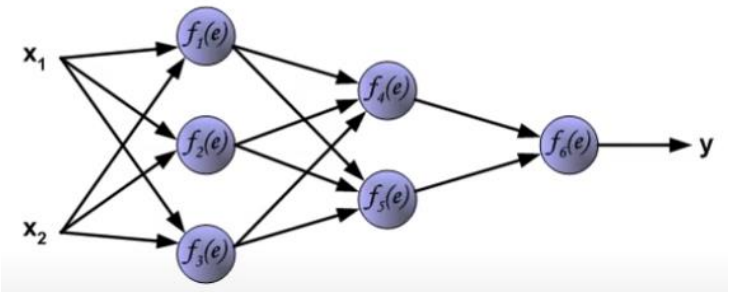
3) Root MSE (Mean Squared Error) – корень квадрата средней ошибки

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2}$$

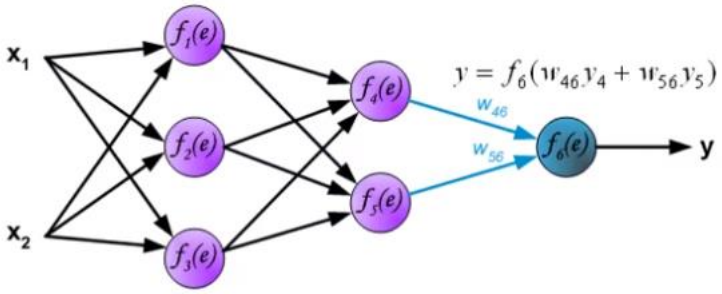
В дальнейшем эти функции будут минимизироваться для поиска оптимальных весов нейронов.

Метод обратного распространения ошибки

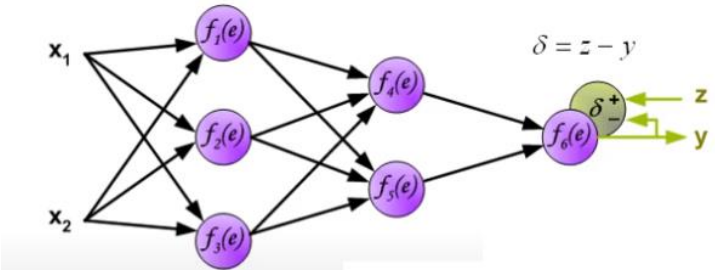
Основной алгоритм обучения нейросетей, эквивалентный дифференцированию сложной функции – это метод распространения обратной ошибки. Рассмотрим нейросеть:



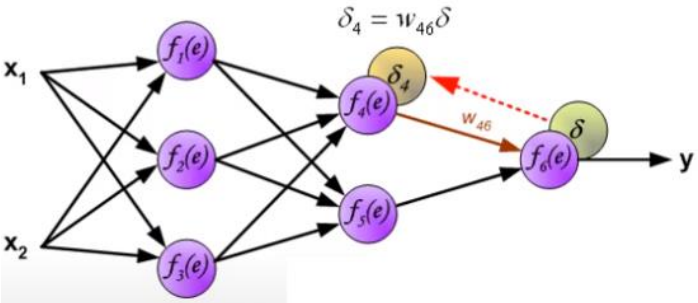
Для начала подаем на вход какие-то значение x1 и x2, чтобы посчитать нейроны:



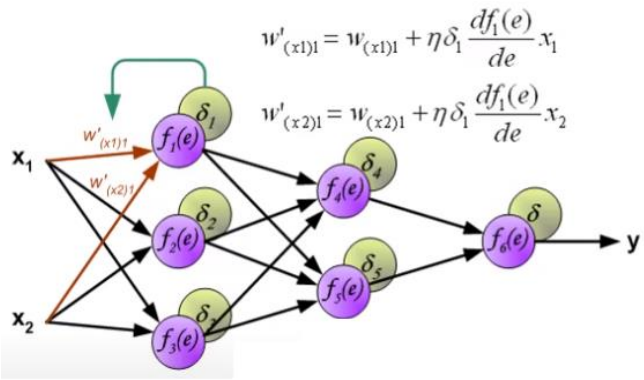
После подсчета значения 6 нейрона, нужно сравнить его с тем значением, которое мы ожидали, тем самым вычислив ошибку предсказания δ.



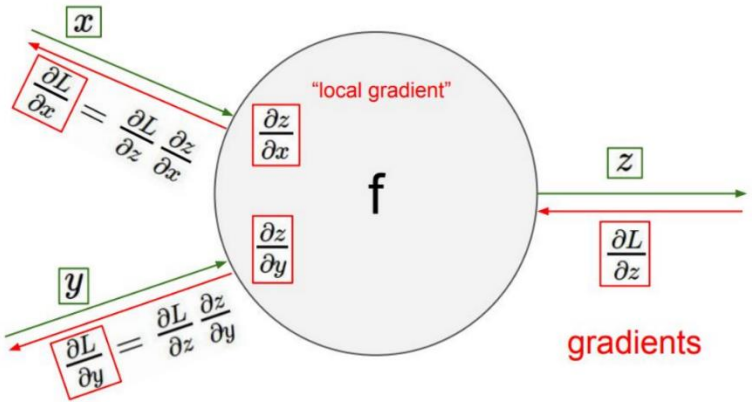
Затем нужно скорректировать веса для каждого нейрона. Для этого посчитаем ошибку и для последующих нейронов:



После нахождения ошибки для каждого нейрона (насколько он повлиял на конечное значение) займемся корректировкой весов. Для этого возьмем предыдущее значение веса и прибавляем к нему произведение коэффициента обучения (как быстро у нас будут изменяться веса) и производную ошибки активации нейрона.



Принципиальная схема расчета обратного распространения ошибки на примере 1 абстрактного нейрона такова:



При расчете производных будем руководствоваться очевидным правилом следующим из математического анализа:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Важно отметить, что после каждого слоя следует функция активация, которая фигурирует в контексте модели, как нелинейное преобразование. Известно, линейная комбинация линейных комбинаций – это линейная комбинация, значит без функций активации вся нейронная сеть будет равносильна 1 нейрону.

$$w_1(w_2(w_3x + b_3) + b_2) + b_1 = (w_1w_2w_3)x + (w_1w_2b_3 + w_1b_2 + b_1)$$
$$\equiv \hat{w}x + \hat{b}$$

Тогда линейная комбинация нелинейных комбинаций – нелинейная комбинация. Последовательность нелинейных преобразований способствует изучению моделью более глубоких закономерностей и извлечения признаков.

Функции активации

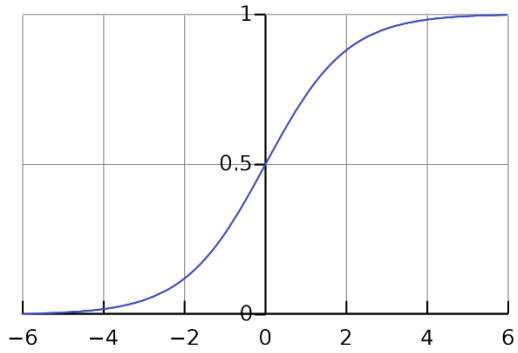
Нейронная сеть шаг за шагом выстраивает новое признаковое представление исходных данных. Изначально модель работает с исходными признаками, а

после первого слоя модель получает новое признаковое представление того объекта, который пришел на вход, затем после второго слоя у модели снова появляется новое признаковое представление изначального объекта и так далее. Для того чтобы признаковое представление не линейно менялись (не были линейной комбинацией исходных признаков) используются нелинейные функции активации.

Рассмотрим основные функции активации:

1) Сигмоидальная функция активации (sigmoid)

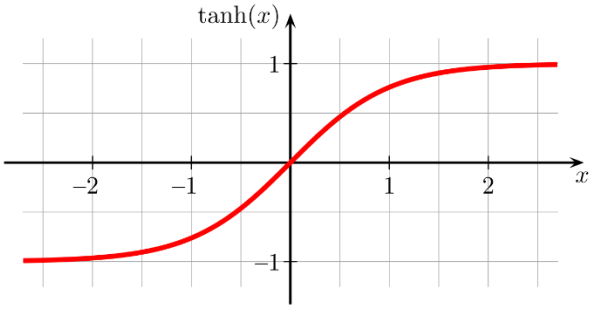
$$\sigma(x) = \frac{1}{1 + e^x}$$



- переводит числа в промежуток [0, 1]
- исторически популярна, так как отлично интерпретируется как “пороговое значение активации” нейрона
- затухание градиентов из-за того, что “хвосты” плоскости
- вывод не нормированный, идет постоянный сдвиг значения

2) Гиперболический тангенс (tanh)

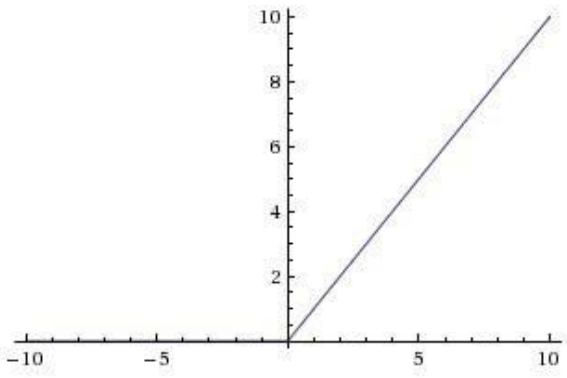
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



- переводит числа в промежуток [-1, 1] (вывод нормированный)
- в отличие от сигмоидальной функции значения центрированный
- проблема затухания градиентов все еще актуальна

3) Функция ReLu (Rectified Linear Unit)

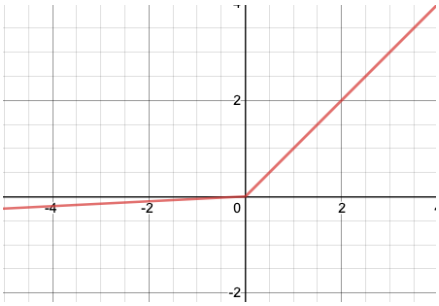
$$ReLu(x) = \max(0, a)$$



- решена проблема затухаещего градиента – справа градиент всегда равен 1
- эффективна с точки зрения вычислений
- однако, вывод не нормирован
- проблема с градиентов в левой части, когда x<0

4) Функция Leaky ReLu

$f(x) = \max(0.01x, x)$ или $f(x) = \max(\alpha x, x)$



- решена проблема затухания градиентов с обоих сторон
- эффективна с точки зрения вычислений
- имеет оптимизируемый параметр

Сверточные нейронные сети (Convolutional Neural Networks)

Обработка изображений нейронными сетями – достаточно тривиальная задача, поэтому на фоне явных трудностей перед полносвязными нейронными сетями, была сформулирована концепция сверточных нейронных сетей. Рассмотрим недостатки с которыми сталкиваются обычные нейронные сети, а затем техники, которые предоставили сверточные нейронные сети для борьбы с этими недостатками.

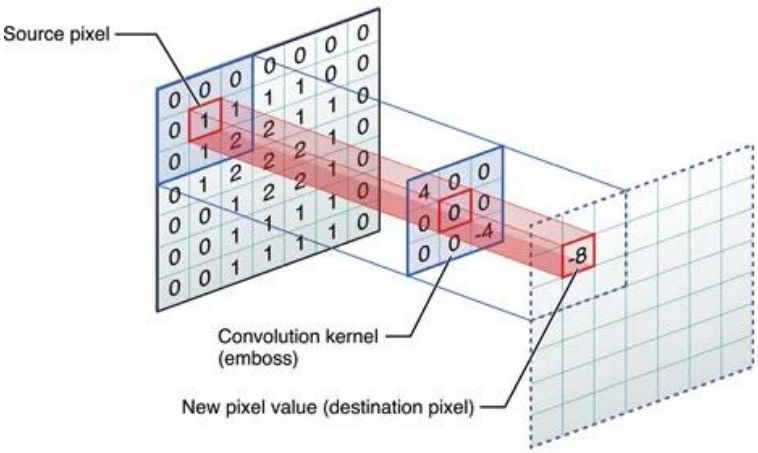
1) Сложность вычислений

Изображение – набор пикселей, тогда входными значениями от изображения

(n, m) будет вектор размера (n*m*3, 1), тройка появилась из соображений цветовой палитры RGB. Затем эти данные нужно будет обработать и если рассматривать форматы высокого разрешения, то количество вычислений для 1 слоя будет более 3 миллионов, что достаточно тяжело для современных вычислительных машин.
2) Потеря информации, получаемой из взаимного расположения пикселей
При использовании вектора из пикселей, обычная нейронная сеть будет рассматривать как набор неких значений, не выделяя характерные для изображения паттерны и свойства. Поэтому полносвязные нейронные сети будут склонны к переобучению.

Свертки и структура сверточных нейронных сетей

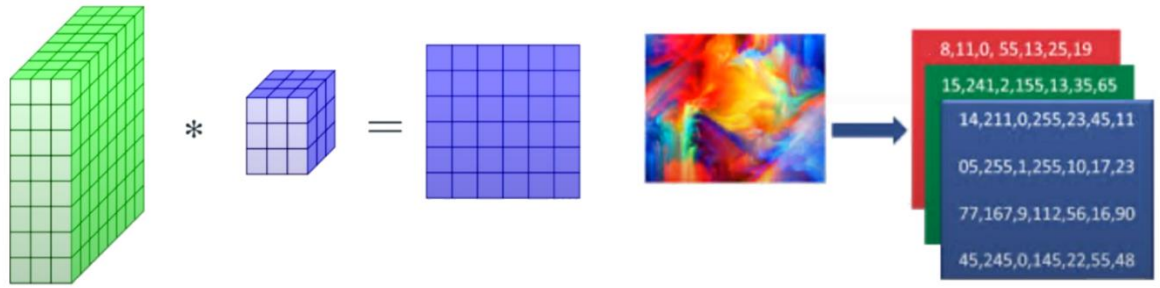
Сверточные нейронные сети используют свертки при помощи ядер свертки иначе называемые фильтр. Фильтр – это некая квадратная матрица размера (n, n), при помощи которого происходит свертка изображения:



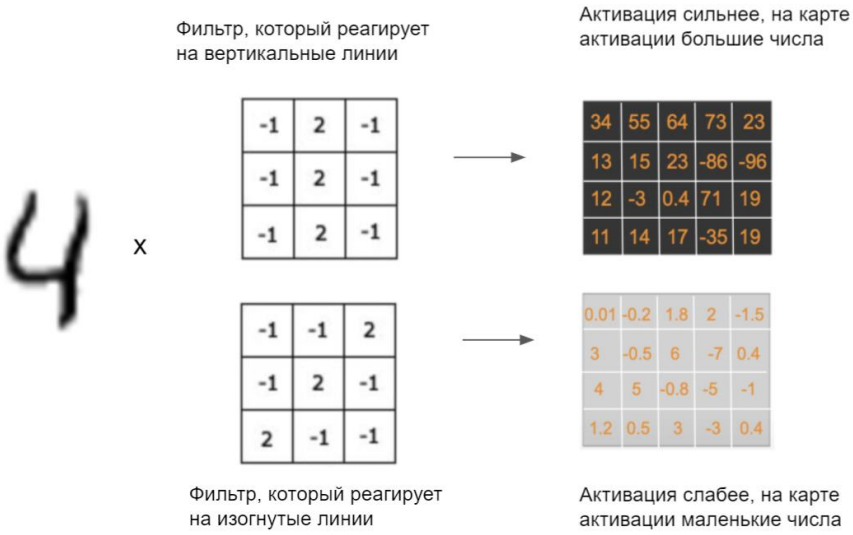
Свертка происходит суммой поэлементного перемножения ядра и области, охватываемого этим ядром из исходного изображения. На рисунке это:

(4 * 0) + (0 * 0) + (0 * 0) + (0 * 0) + (0 * 1) + (0 * 1) + (0 * 0) + (0 * 1) + (-4 * 2) = -8

Затем ядро будет сдвигаться вправо и вниз на величину называемую stride, пока все изображение не будет свернуто. Как можно заметить, полученная свертка будет меньше исходного изображения. Так как изображения цветные и передача трехканальная, то для таких изображений используются так называемые 3d свертки, которые в результате дадут не обьемные матрицы в качестве результата.



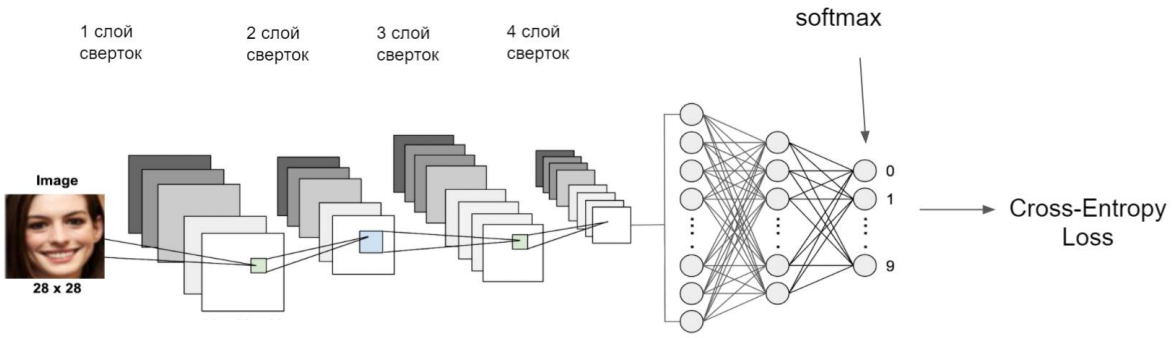
Полученная свертка называется картой активации. Фильтры “реагируют” на паттерны на изображении. Если паттерн присутствует на изображении, то карта активации после соответствующего фильтра будет содержать большие числа. Эта реакция по сути и решает проблему использования информации взаимного расположения пикселей на изображении. Разные фильтры реагируют на соответствующие паттерны, в конечном результате, если числа на карте активации большие – произошла активация. Именно поэтому результат свертки называется картой активации.



Разные ядра способны различать разные паттерны, начиная от горизональных или вертикальных линий и заканчивая, например, чертами лица человека. Стоит отметить, что ядро является обучаемым параметром сети – это значит, что человеку не нужно самому формулировать фильтры распознающие определенные фильтры, нейронная сеть по мере углубления будет корректировать ядра самостоятельно, настраивая их на более сложные паттерны, характерные для конкретной задачи.

После получения карт активаций, происходит их разворачиваение в векторы и последующая конкатенация, чтобы подать на вход полносвязной сети.

Итого структура сверточной сети представлена на рисунке ниже, как можно заметить количество сверток достаточно большое – это сделано для выявления особенных паттернов.



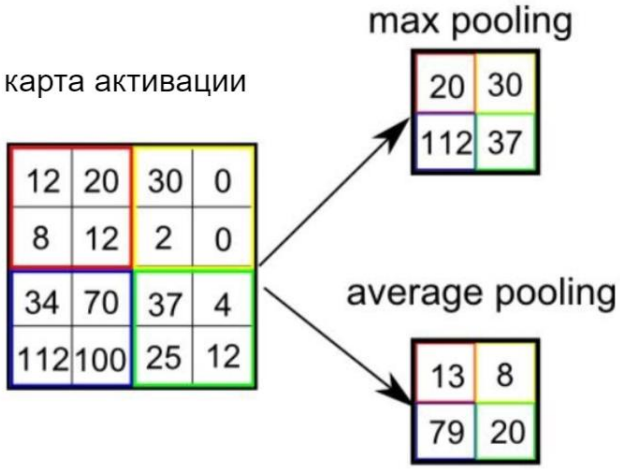
После сверточных слоев, как и после полносвязных, используется функция активации. Самая популярная и хорошо работающая функция активации промежуточных слоев – ReLu.

Pooling

Рассмотрим операцию, которая часто используется в сверточных нейронных сетях для улучшения результатов. Техника уменьшения размерности (downsampling'a) – pooling. Используется для:

- уменьшения размерности очень больших изображений
- уменьшения чувствительности сверток к положению объектов на изображении

Выделяют два основных типа pooling'a – max pooling и avarage pooling. При max pooling = n из каждого квадрата покрываемого ядром pooling выделяется максимальное число, а если используется average, то среднее.



Полезные техники

Рассмотрим некоторые полезные техники для улучшения качества и точности работы нейронной сети:

Optimizers

Для вычисления новых весов нейронов используют градиентный спуск со стандартной оптимизацией SGD (stochastic gradient descent – стохастический градиентный спуск). В SGD фигурирует лишь один изменяемый параметр – lr (learning rate – скорость обучения), который определяет размер шага при поиске минимума функции. Однако SGD имеет и недостатки, например, плохая реакция на шумные данные, а также опасность попасть в ловушку локального минимума и так и не найти глобальный.

SGD: $x_{t+1} = x_t - learning\ rate * \nabla f(x_i)$

Избежать ловушки попадания в локальный минимум можно используя momentum. Концепция momentum'а заключается в накоплении импульса или движения по инерции.

SGD with momentum:

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Где α – learning rate, v – параметр импульса, некая накопительная переменная.

Однако есть возможность улучшить и SGD с momentum – это Nesterov momentum. Позволим нашей модели "заглядывать в будущее", особенность метода заключается в том, что сначала происходит сдвиг по инерции, а только потом расчет градиента.

SGD with Nesterov momentum:

$$v_{t+1} = \rho v_t + \nabla f(x_t + \ \rho v_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Для еще большей оптимизации работы модели встает задача создания адаптивного learning rate для каждого признака. Эту проблему решает техника Adagrad: SGD with cache, которая характеризуется сохранением старых градиентов.

$$cache_{t+1} = \ cache_t \ + (\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{cache_{t+1} + \varepsilon}$$

Где cache – переменная для накапливания, что-то похожее на v в momentum, а ε – очень маленькое число, чтобы исключить случаи деления на 0.

Однако в случае Adagrad появляется проблема затухания градиента, когда знаменатель становится очень большой. Эту проблему решает RMSprop, концепция которого заключается в модификации Adagrad экспоненциальным сглаживанием.

$$cache_{t+1} = \beta cache_t + (1-\beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - \alpha \frac{\nabla f(x_t)}{cache_{t+1} + \varepsilon}$$

Где β – сглаживающий коэффициент <1.

Подытожим, есть идея накапливать некий импульс и двигаться по инерции (momentum), а также нормировать каждую компоненту градиента на какое-то число (RMSprop). Тогда исходя из отсутствия противоречия, можно объединить эти две идеи. Adam (adaptive learning rate optimization) – комбинация двух ранее показанных подходов.

$$v_{t+1} = \rho v_t + (1-\gamma)\nabla f(x_t + \rho v_t)$$

$$cache_{t+1} = \beta cache_t + (1-\beta)(\nabla f(x_t))^2$$

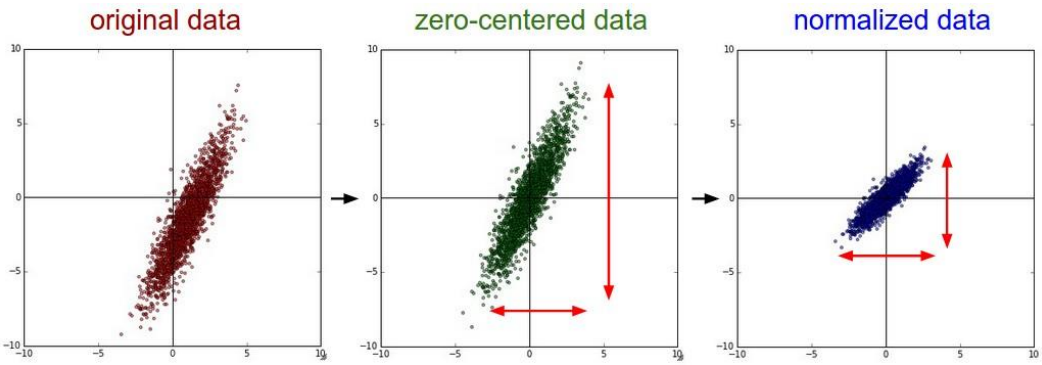
$$x_{t+1} = x_t - \alpha \frac{v_{t+1}}{cache_{t+1} + \varepsilon}$$

Data normalization

Рассмотрим пример некоторой выборки данных в двумерном пространстве с признаками по осям. Заметим, что изначальная выборка смещена относительно центра и имеет разброс, это может привести к уменьшению качества работы модели и как следствие точности. Это происходит, потому что разделяющаей плоскости будет сложно отделять классы из-за большого скопления объектов в одном направлении и небольшого в другом. Тогда следует отнормируем данные путем их ценрализации – графики посередине и справа. Для этого вычтем среднее и поделим на дисперсию.

$x_i = \frac{x_i - u_i}{\sigma_i}$, где $u_i -$ среднее значение всех объектов x, а $\sigma_i -$ дисперсия выборки данных.

Теперь провести точную разделяющую повехность гораздно проще.



Batch normalization

Введем понятие пакета, набора – batch. Для эффективной работы с большими наборами старовых данных, которые также называют датасетами, чтобы не подавать в нейронную сеть по 1 элементу (так как это очень долго), была придумана концепция batch'ей, где в нейронную сеть подаются объединенные в небольшие пакеты элементы.
Тогда появляется проблема обработки поступающих значений слоям из середины. Например, второй, при прохождении батча будет использовать для формулировки ответа данные, пришедшие из первого слоя. Тогда во время обратного прохода, когда веса будут корректироваться, 2 слой изменит веса так, чтобы они соответствовали выходам из 1 слоя. Однако веса меняет не только 2 слой, но и 1, а это значит, что корректировка весов 2 слоя будет основана на неправильных данных, эту проблему решают нормализацией данных поступающих в слой:

$$x_i = \frac{x_i - u_i}{\sigma_i}, \text{где } x_i - \text{ элемент пакета}, u_i - \text{среднее значение всех элементов}, \sigma_i - \text{дисперсия}$$

Regulariztion

Для нейросетей существует проблема переобучения (overfitting), она характеризуется слишком большими весами для определенных нейронов сети и как следствие вклад этих нейронов в результат будет гораздо больше – это искажает результат. Данную проблему решает регуляризация:

Ограничение на веса (weight decay) – добавляем дополнительное слагаемое которое будет изменяться и минимизировать итоговую функцию.

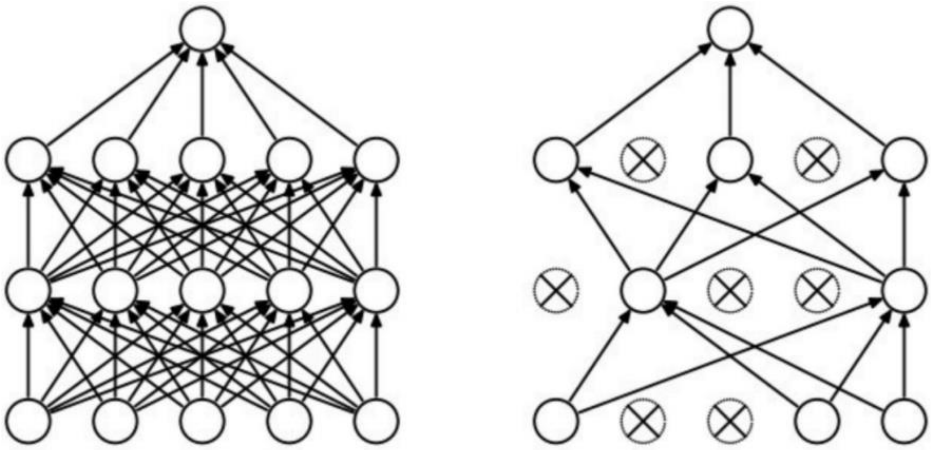
$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=y_j}\max\left(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1\right) + \lambda R(W)$$

Например для разных видов регуляризации:

- При L1 регуляризации $R(w) = \left|\left|w\right|\right|_1$
- При L2 регуляризации $R(w) = \left|\left|w\right|\right|_2^2$
- Elastic net это сумма L1 и L2 $R(w) = \left|\left|w\right|\right|_1 + \beta\left|\left|w\right|\right|_2^2$

Dropout

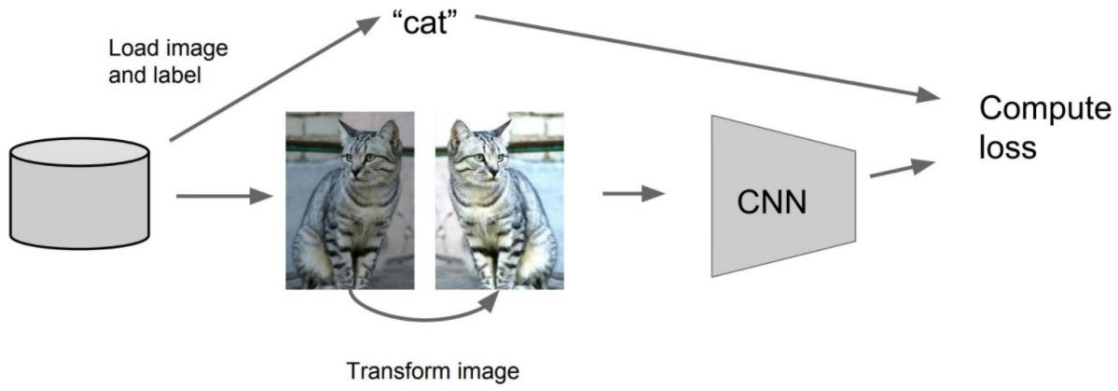
Помимо регуляризации для борьбы с переобучением также используется технология Dropout. Она заключается в случайном “выключении” нейронов, выключение подрузамевает обнуление весов на 1 проход. Особенность данного подхода в том, что нейронная сеть будет делать предсказания опираясь на вывод из всех нейронов, а не только каких-то определенных, потому что они могут быть выключены. Таким образом остальные нейроны слоя будут корректировать свое значение, пытаясь заменить выключенный нейрон, что гарантирует защиту от переобучения.



(a) Standard Neural Net (b) After applying dropout.

Аугментация данных

Порой случается, что данных для обучения мало, из-за малочисленности данных, нейронная сеть может переобучиться. Для этого можно сгенерировать новые данные путем преобразованиях исходных данных, причем эти преобразования будут сохранять смысл и реалистичность. Таким образом можно улучшить качество работы нейронной сети даже не изменяя закономерности непосредственно внутри сети.

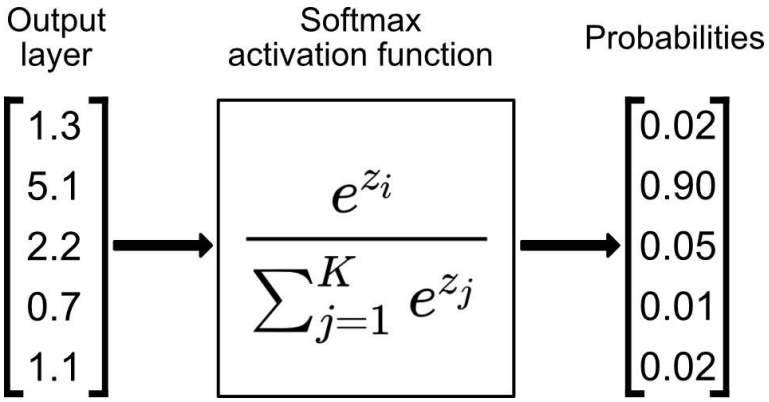


Softmax

Ранее было показано, что в нейронной сети есть функция ошибки, которая регулирует корректировку весов. Однако приведенные выше функции ошибки не предусматривают случай, когда результирующих классов много. Рассмотрим функцию softmax, которая позволяет работать с большим кромчеством классов.

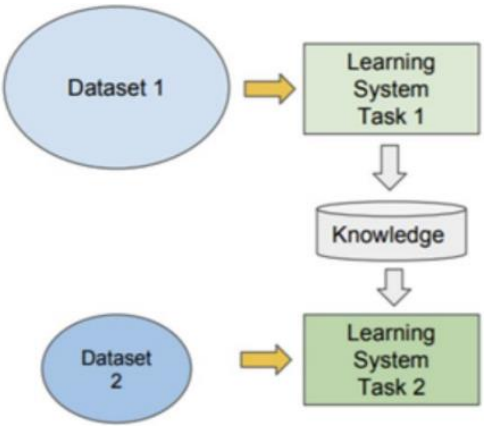
$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Где K – количество классов на которые могут подразделяться объекты.
В конечном итоге для всех элементов выходного вектора значений применяется softmax. Эта функция отражает принадлежность элемента к каждому классу. В конечном итоге оъект относится к тому классу, вероятность принадлежности к которому максимальна.



Предобученные нейросети

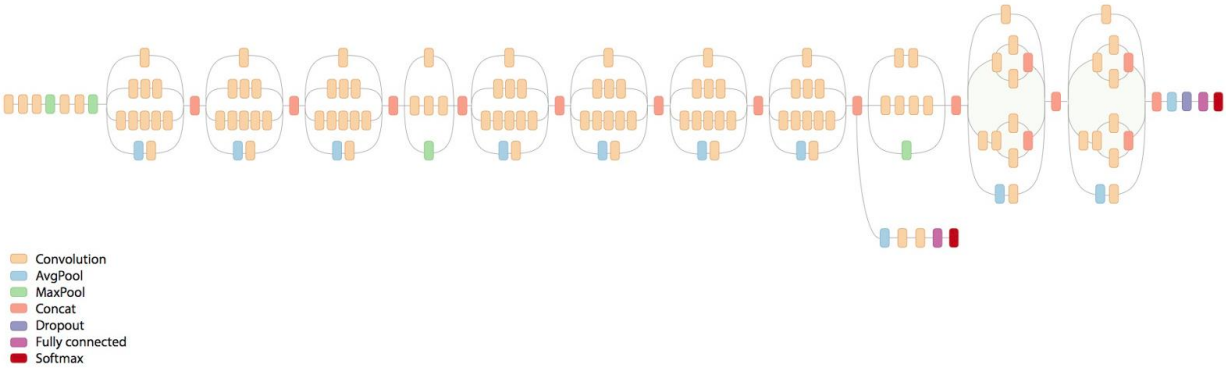
Часто датасет под какую-либо задачу для обучения сети содержит мало объектов. И если обучать сеть на этом датасете с нуля, то сеть переобучится. Например, машинный перевод с малораспространенных языков, таких так татарский. Тогда можно использовать знания, полученные другими сетями на похожих задачах.



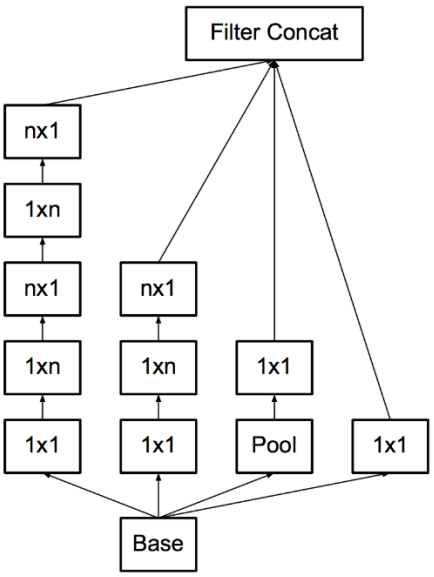
Inception v3

В работе будет использоваться предобученная сеть Inception v3.

Вот как выглядит вся модель:



Отличием этой модели является особенная архитектура, которая содержит в себе слой из параллельных сверток. Эти четыре слоя выдадут совершенно разную информацию, которую выделят из пришедшей ранее информации. Таким образом за один слой будут выделяться сразу разные признаки, затем информация конкатенируется и подается следующему слою.



Также в данной модели используются техники и методы, рассмотренные ранее.

Используемые данные

В качестве датасета был выбран набор картинок с персонажами из популярного западного мультфильма “Симпсоны”

Обучение модели

В приложенном notebook, выделены и обьяснены все основные функции и действия. Рассмотрим сам процесс обучения, после 24 эпох обучения нейронная сеть способна выдать точность 97.94% на валидационной выборке.

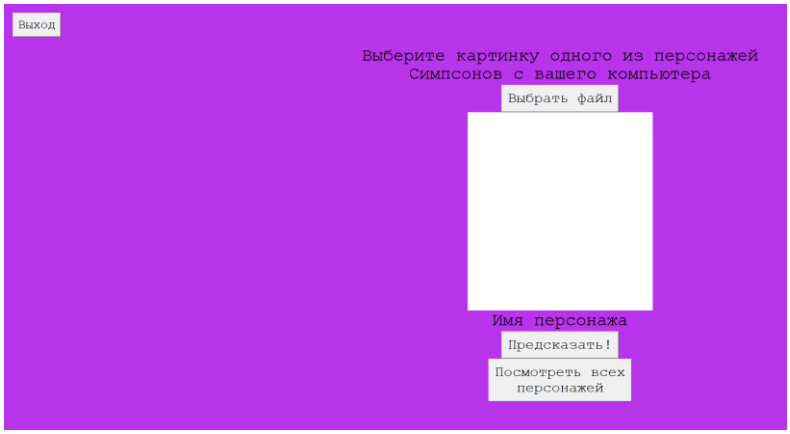
```
epoch:  88%|          | 21/24 [3:24:22<29:07, 582.47s/it]
Epoch 021 train_loss: 0.0012     val_loss 0.1188 train_acc 0.9996 val_acc 0.9794
loss 0.0009980097134653738
epoch:  92%|          | 22/24 [3:34:05<19:25, 582.55s/it]
Epoch 022 train_loss: 0.0010     val_loss 0.1189 train_acc 0.9997 val_acc 0.9794
loss 0.0010359290267036739
epoch:  96%|          | 23/24 [3:43:47<09:42, 582.55s/it]
Epoch 023 train_loss: 0.0010     val_loss 0.1190 train_acc 0.9996 val_acc 0.9794
loss 0.0009721070564980247
epoch: 100%|          | 24/24 [3:53:29<00:00, 583.74s/it]
Epoch 024 train_loss: 0.0010     val_loss 0.1191 train_acc 0.9998 val_acc 0.9794
```

Приложение

Сохраним обученную модель для последующего использования в локальном приложении.

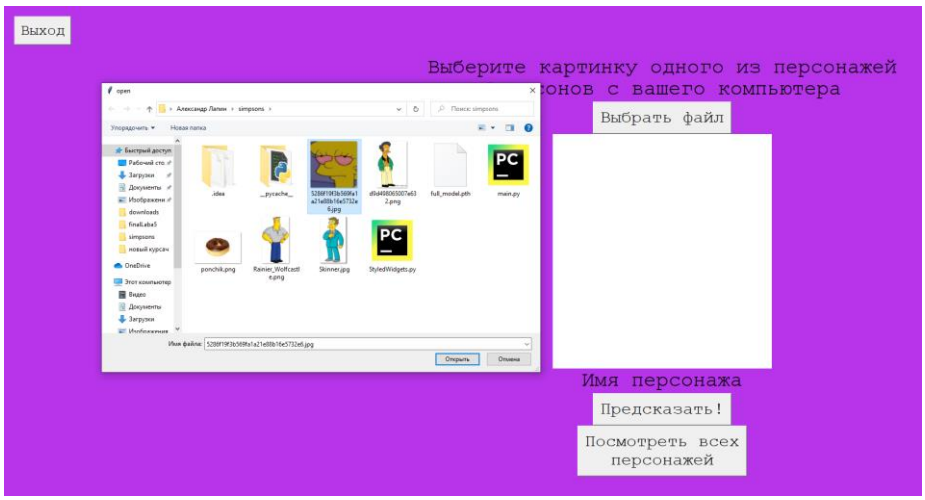
```
1 torch.save(model_incep, 'gdrive/My Drive/full_model.pth')
```

Теперь создадим интерфейс приложения и логику.



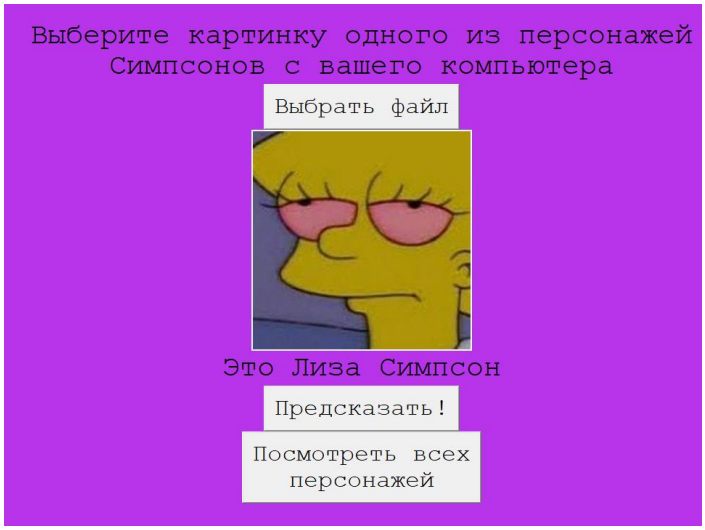
Функции и процесс предсказания описаны в коде. Так что посмотрим на процесс работы приложения.

Приложение позволяет выбрать изображение непосредственно с компьютера пользователя и предсказать имя персонажа с изображения.



Выберем изображение и сделаем предсказание.

Как можно заметить, модель успешно справилась.



Заключение

Цель, создать нейронную сеть, способную решать задачу по распознаванию изображений и разделению их на классы была достигнута. В процессе реализации были изучены нейронные сети, а также техники и методы, оптимизирующие и улучшающие их работу. В качестве практической части работы была разработана и обучена нейронная сеть, которая впоследствии была интегрирована в приложение.

Список используемых источников

1. Эволюция нейросетей для распознавания изображений в Google: Inception-v3 https://habr.com/ru/post/302242/
2. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество. https://habr.com/ru/post/348000/
3. Официальная документация по модулю nn библиотеки PyTorch. https://pytorch.org/docs/stable/nn.html
4. Официальная документация по библиотеке matplotlib https://matplotlib.org/
5. Стохастический градиентный спуск. http://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D1%82%D0%BE%D1%85%D0%B0%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA
6. Нейронные сети для начинающих. Часть 1. https://habr.com/ru/post/312450/
7. Нейронные сети для начинающих. Часть 2. https://habr.com/ru/post/313216/
8. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество. https://habr.com/ru/post/348000/
9. Введение в Tkinter https://habr.com/ru/post/133337/
10. Книга «Программируем с PyTorch: Создание приложений глубокого обучения» https://habr.com/ru/company/piter/blog/512158/