Софийски университет

„Св. Климент Охридски"

Дипломна работа

Катедра „Софтуерни технологии"

Автоматично разделяне на големи количества изображения на класове с използване на дълбоко учене (deep learning)

Дипломант:

Валентин Змийчаров

Специалност:

Извличане на информация и откриване на знания

Факултетен номер:

24952

Научен ръководител:

Д-р Милен Чечев

София 2017г.

Съдържание

1 Увод

1.1 Актуалност на проблема и мотивация

Разпознаването на изображения е способност на човека, която не е тривиално да бъде решена от компютър. Класифициране на изображения означава причисляване на изображение към един или повече класове с определена точност. Нека например имаме непресичащи се класове за превозни средства: автомобил, мотор, самолет, кораб, подовдница. Нека имаме изображенията, показани на 1.1.

Фигура 1.1: Примерни изображения

Целта е първата картинка да се причисли към класа автомобили, втората към класа мотоциклети и третата към кораби. Това е елементарно за човешкото око, но компютърът вижда само последователност от нули и единици. Освен това при дадения пример разликите са големи, но може да се налага да се класифицират изображения, които са трудно различими дори с просто око.

Проблемът е особено актуален, поради непрестанното използване на социални мрежи и облачни услуги в световен мащаб. Всеки ден над 350 милиона снимки се качват само във Facebook [13]. С използване на модерните технологии заснемаме гигантско количество изображения и много често след това просто забравяме за тях. Непосилна задача е ръчно да обозначим съдържанието

на всяка една от снимките. Тук на помощ идва автоматичната обработка и анализ на съдържанието на изображения.

С намирането на добро решение на проблема са се захванали някои от най-големите компании в сферата на информационните технологии:

- **Google photos** [23] използва автоматично класифициране на изображения, за да групира снимките, които качваме по категории и папки. Огромна публичност доби скандал с автоматичното категоризиране, при който двойка тъмнокожи бяха категоризирани като горили [20]. Това повдигна редица етични въпроси относно проблема. Google също предоставя и API за категоризация на изображения: Vision API [31], което позволява интегрирането на софтуер за разпознаване на изображения в собствена система.

- **Microsoft** също предоставя подобна услуга за своите облачни услуги. Те имат и аналогично API [9].

- Amazon, Dropbox също поддържат подобни услуги, което говори за потенциала на разпознаването на изображения.

Бизнесът на някои компании изцяло лежи на разпознаването на изображения. Такива са Imagga [16], Clarifai [8].

Предоставянето на тази функционалност като услуга за бизнесите предразполага раждането на много оригинални идеи. Такива са вече споменатите облачни услуги. С наличието на API за разпознаване на изображения всеки доставчик на облачни услуги може да направи автоматична категоризация на изображенията на потребителите. Друг интересен случай са сайтове и приложения за споделяне на авторски изображения. При тях обемът на данните е огромен и в повечето случаи няма никаква конкретна информация.

Разпознаването на обекти в изображението би могло да бъде от полза и по отношение на сигурността. При голямо количество съдържание може лесно да се намерят например оръжия или снимки с нецензурирано съдържание. Обемът на снимките расте с гигантски размери и е единствено въпрос на въображение от страна на компаниите какви ползи могат да се извлекат от автоматичното категоризиране на изображения.

1.2 Цел и задачи на дипломната работа

Целта на дипломната работа е да проучи съвременните подходи за класифицирането на изображения, като се дават детайли за използването на конволюционни невронни мрежи с дълбоко обучение.

Задачите биха могли да се обобщят по следния начин:

1. Обзор на проблемната област. Да се направи проучване на различните софтуерни платформи, които може да се използват за класифициране на изображения в големи мащаби. Какви са предимствата и недостатъците на всяка една от тях. Какво използват най-големите компании в тази сфера.

2. Какви са условията (финансови, хардуерни и др.), за да построим класификатор с добра точност. Какво е добра точност и каква точност постигат готовите решения.

3. Подбор на данни за обучение. Намиране на достатъчно голям обем категоризирани изображения с добро качество с цел обучение.

4. Реализация на прототип за класификация на изображения.

5. Експерименти за оценка на точността на класификацията и възможностите за работа с големи данни.

6. Анализ на получените резултати.

1.3 Структура на дипломната работа

Глава 1 е въведение в проблемната област. Тя цели да покаже кратко каква е целта на дипломната работа и мотивацията да се разреши точно този проблем. Обсъдени са резултатите, които трябва да се постигнат.

Глава 2 представя в детайли използваните алгоритми и подходи. Освен това в нея се аргументира изборът на подход с конволюционни невронни мрежи и се прави подробен анализ на най-доброто решение до момента на писане на дипломната работа.

Глава 3 мотивира избора на технологии, платформи, услуги и библиотеки, които са необходими при разработването на

конволюционната невронна мрежа за класифициране на изображения. Направено е въведение в спецификите на Python и Tensorflow като са анализирани различните функционалности и спрямо изискванията на конкретния проблем е избрано подмножество от тях.

Глава 4 дава подробен поглед върху процеса по реализация на конволюционната невронна мрежа за класифициране на изображения и избора на архитектура. Вниманието е насочено към направените експерименти и анализ на резултатите от тях.

Глава 5 е заключителната част, даваща насоки за бъдещо развитие на системата и обобщаващо в каква степен са изпълнени първоначалните поставени цели и задачи.

2 Преглед на предметната област

2.1 Въведение

През 1959 г. Артър Самюел определя машинното самообучение като „Поле на проучване, което дава на компютрите възможността да учат, без да бъдат експлицитно програмирани“. То изследва проучването и изграждането на алгоритми, които да правят прогнози чрез данни и от които да може да се учи. Машинното самообучение използва събрани исторически данни за изграждането на модел, който в последствие да може да се използва върху нови данни.

Машинното самообучение се използва в широка гама от изчислителни задачи, където програмирането на експлицитни алгоритми е неосъществимо. Примерните приложения включват филтриране на спам, оптично разпознаване на символи, търсачки и компютърно зрение.

В областта на аналитични данни, машинното самообучение се използва за разработване на сложни модели и алгоритми, които се поддават на прогнозиране. Тези аналитични модели позволяват изследователи, учени и инженери да получават надеждни, повтаряеми решения и резултати.

2.2 Разпознаване на изображения: мотивация и история

През 1963 година в докторската си теза Л. Робъртс [26] разглежда в детайли как машините могат да възприемат дадени триизмерни обекти от изображение. Той анализира очертанията на обектите и различните перспективи на заснемане. През 1987 година учени от Кеймбридж (Масачузетс) описват разпознаване на обекти с

използване на изкуствен интелект [14]. Намират се изключително много приложения на разпознаването на обекти в изображения и все повече и повече данни започват да бъдат налични.

В края на 70-те и началото на 80-те невронните мрежи започват да добиват все по-голяма популярност и започват да се прилагат в множество области. Конволюционните невронни мрежи са вдъхновени от зрението на животните в природата. В началото на 21 век се случват няколко много важни неща, които предразполагат развитието на пазара в сферата на автоматичното класифициране на изображения.

Едно от тях е, че се намират начини за използване на видео картата за изчисления в невронните мрежи, което значително подобрява времето на изпълнение. Със създаването и популяризирането на социалните мрежи броят на картинките е огромен: всеки ден над 350 милиона снимки се качват само във Facebook [13].

Активното използване на автоматично разчитане на изображения се прилага активно в днешни дни. То служи за разпознаване на номера от автомобили, пощенски кодове и чекове. Все с по-голяма сила навлиза разпознаването на пръстови отпечатъци за идентификация [17].

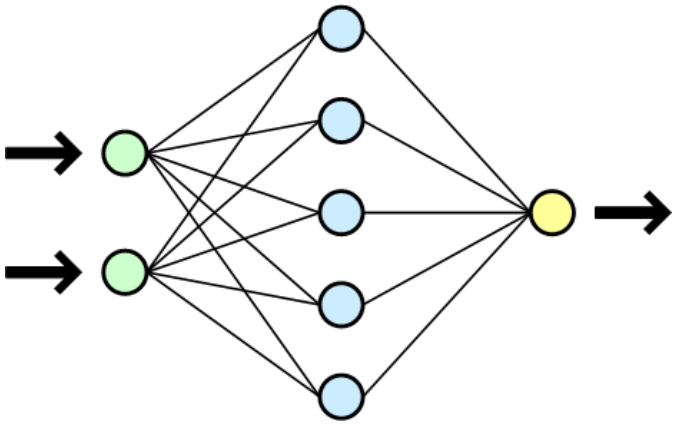
Със особено влияние са алгоритмите за локализиране и разпознаване на лице на хора. При наличието на много улични камери такъв алгоритъм разполага с възможност да търси в гигантско количество информация.

С увеличаването на интернет потребителите се увеличават и заплахите в електронния свят. Публично се разпространяват изображения, свързани с насилие, детска порнография и други нецензурирани материали. Автоматичното разпознаване на изображения идва на помощ при засичането на подобни нередности.

2.3 Невронни мрежи

Невронната мрежа е модел за обработка на информация, вдъхновен от изучаването на биоелектричните мрежи в мозъка на човека и животните, образувани от неврони и техните синапси. В наши дни учените често наричат изкуствените невронни мрежи просто невронни мрежи.

Математическият аналог на биологичната невронна мрежа представлява множество от взаимносвързани прости изчислителни елементи (неврони). Всеки неврон приема сигнали от другите (под формата на числа), сумира ги, като сумата минава през активационна функция, и така определя своята активация (степен на възбуда), която се предава по изходящите връзки към другите неврони. Всяка връзка има тегло, което, умножавайки се със сигнала, определя неговата значимост (сила). Теглата на връзките са аналогични на силата на синаптичните импулси, предавани между биологичните неврони. Отрицателна стойност на теглото съответства на потискащ импулс, а положителна – на възбуждащ. 2.1 илюстрира примерна невронна мрежа.



Фигура 2.1: Невронна мрежа

В невронната мрежа обикновено винаги съществуват входен и изходен слой от неврони, във входния се въвежда информацията към мрежата, след това сигналите от входните неврони преминават през един или няколко слоя от междинни (скрити) неврони, според топологията на невронната мрежа, като сигналите накрая стигат до изходния слой, откъдето се чете получената информация.

Теглата на връзките между невроните определят функционалността и поведението на невронната мрежа. За да бъде една невронна мрежа използваема и приложима към даден проблем, тя трябва да бъде предварително обучена.

Обучаването на една невронна мрежа се осъществявя с използване на тренировъчни данни, като в процеса на обучение се променят теглата на връзките между невроните търсейки

се най-оптимални тегла спрямо тренировъчните данни. Най-разпространеното правило за това е метода на обратното разпространение на сигнал за грешка (back-propagation), където за всеки изходен неврон се изчислява разликата от желаното му поведение, като се формира сигнал за грешка, който се движи назад към входния слой и по пътя си променя теглата на връзките така, че при следващата активация на мрежата грешката да бъде по-малка от сегашната.

Невронните мрежи много се доближават до нашите мозъци по начина си на функциониране. Изчисленията не се извършват на едно централно място, а са разпределени по цялата мрежа. Това прави мрежата изключително гъвкава, като тя продължава да работи дори когато от нея умишлено са премахнати изчислителни елементи. За сравнение, фон Нойманова архитектура престава да функционира ако от нея липсва дори един елемент.

За съжаление, дори този модел на човешкия ум не е перфектен. Повечето съществуващи към момента изкуствени невронни мрежи не могат да моделират ефекта на хормоните върху мозъка. Освен това се оказва, че за да можем математически да опишем една изкуствена мрежа, в нея трябва да се спазват определени правила, например сигналите да се движат от входния към изходния слой, но не и обратно. (Съществуват и изкуствени мрежи, които са циклични, но поради повишената им сложност все още не са добре изучени.) Може би най-големият недостатък на невронните мрежи е, че тяхното обучение изисква множество повторения, а в много случаи човек научава дадена информация след като е бил изложен на нея само веднъж.

2.4 Задълбочено обучение (deep learning)

Задълбоченото обучение е направление в машинното самообучение, който се основава на алгоритми, които целят да изградят абстракция от високо ниво на данните. Ако разгледаме един много прост случай може да имаме две множества неврони: едни, които получават входен сигнал и втори, които изпращат изходящ сигнал. Когато първият слой получи вход, той предава модифицирана версия на входа към следващия слой. При по-дълбока мрежа, има много слоя между входния и изходния, което

позволява на алгоритъма за използва множество обработващи слоя, състоящи се от много трансформации. [10] [2]

Дълбокото учение е част от по-голяма група методи в машинното самообучение, които се основават на обучение върху изгледа на данни. Наблюдение (например картинка) може да се представи по много начини като например вектор от интензивността на пикселите или по по-абстрактен начин като множество от ъгли, региони с определена форма и още много други. Някои изгледи са по-добри от други по отношение на опростяване на обучението. Една от основните дейности на задълбоченото обучение е, че се премахва нуждата от ръчно изваждане на модели от хора. На тяхно място се използват ефективни алгоритми за обучение и йерархично автоматично изграждане на модели.

Разработките в тази насока се стремят да се намерят тези модели в огромен обем необработени данни. Някои модели са вдъхновени от напредъка в неврологията и се основават на интерпретации на обработката на информация и начините на комуникация в нервната система.

Различни видове архитектури като Невронни мрежи, Конволюционни невронни мрежи, Deep belief мрежи и рекурентни невронни мрежи са прилагани в области като разпознаване на реч, обработка на естесвен език, разпознаване на звуци, биоинформатика, където те достигат най-точни резултати от всички използвани методи.

2.5 Конволюционни Невронни мрежи

Този вид невронни мрежи за вдъхновени от визуалния кортекс при животните. Отделни неврони отговарят за стимулирането на ограничен район на действие, който се нарича област на възприемане. Когато полето на възприемане на даден неврон бъде задействано, неговият отговор може да бъде изчислен математически чрез конволюционна операция (convolutional). Вдъхновени са от биологични процеси и са вариация на многопластов перцептрон, който използва минимално количество предварителна обработка. Те са най-приложими в обработката на изображения и видеа, препоръчващите системи и обработката на естествен език. [19] [7]

2.5.1 Област (поле) на възприемане

През 50-те и 60-те години на 20 век труд на Хубел и Визел показва, че визуалния кортекс на котките и маймуните съдържа неврони, които индивидуално реагират на малки региони от визуалното поле. Регионът от визуалното поле, в който съдържанието въздейства на специфични неврони се нарича област (поле) на възприемане. Съседните клетки имат подобни и припокриващи се полета на възприемане. Размерът и мястото на областта на възприемане варира из корпуса, за да формира пълна карта на визуалното пространство. [21]

2.5.2 Специфични особености

Въпреки, че стандартните невронни мрежи могат успешно да бъдат приложени за разпознаване на изображения, те имат сериозен недостатък при уголемяване на размерите на изображенията. Например при изображение с размери 32x32 пиксела и 3 цветови канала един пълно свързан неврон би имал 32*32*3=3072 тегла. При изображение с размери 500x500 броят им би бил 750 000. При наличието на следващ слой със същата големина броят на връзките би бил 562 500 000 000. Освен това тази архитектура третира пиксели, които са раздалечени по един и същи начин като пиксели, които са един до друг. Очевидно пълната свързаност на невроните не е добро решение при задачата за разпознаване на изображения. Като следствие огромният брой параметри води до прекомерно нагаждане (overfitting). Конволюционните невронни мрежи имат следните специфични осоености

1. **3 измерения на невроните**. Слоевете имат неврони, организирани в 3 измерения: ширина, височина и дълбочина. Невроните във всеки слой са свързани само с малък брой от невроните в предишния слой (област на възприемане). Различни видове слоеве (както локално така и напълно свързани) се обединяват за да организират конволюционна невронна мрежа.

2. **Локална свързаност**: следвайки концепцията на полето на възприемане, този вид невронни мрежи използват локална свързаност като принуждават невроните от последователните

слоеве да комуникират само с близките до тях съседи. По този начин архитектурата осигурява това, че научените "филтри" продуцират най-силна връзка при взаимодействие между близки неврони. Нареждането на много подобни връзки води до нелинейни филтри, които стават все по-глобални. Това позволява на мрежата първо да намери отличителни черти на малка част от входа и след това да види как те са приложени в по-голямо изображение.

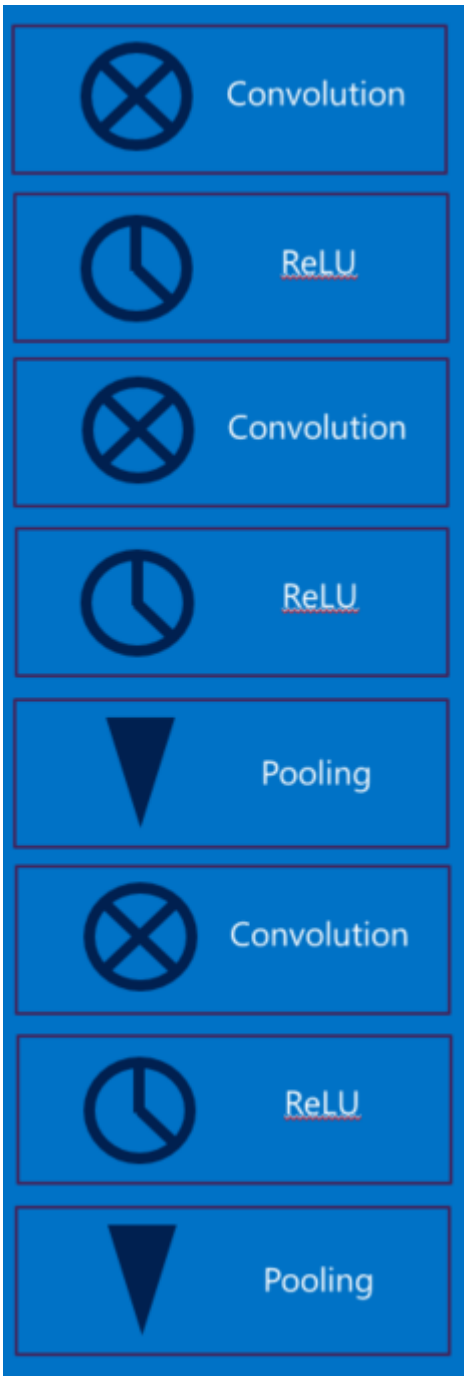
3. **Споделени тегла**: Всеки филтър се прилага върху цялото визуално поле. Където се използват, тези единици споделят едни и същи тегла. Това означава, че всички неврони в даден слой засичат точно един и същи модел на свързаност. По този начин се позволява моделът да бъде намерен независимо къде се намира върху визуалното поле.

Заедно тези свойства позволяват на конволюционните невронни мрежи да постигнат по-добра генерализация при визуалните проблеми. Споделянето на теглата драстично намалява броя на параметрите, които трябва да се обучат. По този начин се намаляват хардуерните изисквания за използване на невронната мрежа. Това от своя страна позволява тренирането на по-големи и по-мощни мрежи.

2.5.3 Видове слоеве

Конволюционните невронни мрежи използват някои различни слоеве в сравнение със стандартните невронни мрежи. Такива са конволюционният слой, обединяващият слой и слоят на ректифицираните линейни единици (ReLU).

2.2 показва примерна архитектура на конволюционна невронна мрежа.



Фигура 2.2: Архитектура на конволюционна невронна мрежа [5]

Конволюционен слой

Това е основата на този тип невронни мрежи. Параметрите на този слой са множество от самообучаващи се филтри, които имат малко поле на възприемане, но се разширяват до пълната дълбочина на входните параметри. При всяко предаване едно ниво напред всеки

филтър създава своя карта в зависимост от това дали се среща или не по цялата ширина и височина на входа. Това означава, че се търси къде този филтър е наличен в подадения вход и се връща като резултат двуизмерна активационна карта, указваща местата на филтъра. Като резултат мрежата научава филтри, които се активират при специфичен модел на някоя позиция от входа. Като се обединят активационните карти на всички филтри по измерението за дълбочина се оформя пълният изход от конволюционния слой. Всяка единица от изхода може да се интерпретира като резултат от неврон, който отговаря за малък регион от визуалното поле и споделя параметри с неврони от същата активационна карта.

Когато входните данни са от много високо измерение (както изображенията) не е практично да се свързват неврони с всички неврони от предходния слой, защото такава архитектура не взима предвид близостта на данните (например дали 2 пиксела са един до друг или раздалечени). Всеки неврон е свързан само с малък регион от входните данни. Големината на тези региони е хиперпараметър, който се нарича област на възприемане на неврона. Връзките са локални по отношение на ширина и височина, но винаги са по цялата дълбочина на входните данни. Такава архитектура гарантира, че научените филтри имат най-силно въздействие върху резултата.

Три хиперпараметъра контролират размера на изхода на конволюционния слой: дълбочина, стъпка на алокиране и брой колони с добавени нули. Дълбочината на изхода контролира броя неврони в слоя, които са свързани със същия регион от входния слой. Всички тези неврони ще се научат да се активират за различни модели на входа. Например, ако първият конволюционен слой приема необработено изображение като вход, различните неврони разположени в дълбочина може да се активират при наличие на различни ъгли, цветове или други форми. Стъпката (stride) се грижи за това как се оформят колоните в дълбочина около измеренията ширина и височина. Когато съпката е 1 се образува нова колона неврони, които са свързани само до единиците на разстояние 1. Това води до пренатоварване на полетата на възприемане между колоните и също така до огромни размери на изхода. Понякога е удобно да изместим входните данни като добавим нули на граничните стойности. Колко колони с нули ще добавим е третият хиперпараметър. Той ни дава контрол да определеми

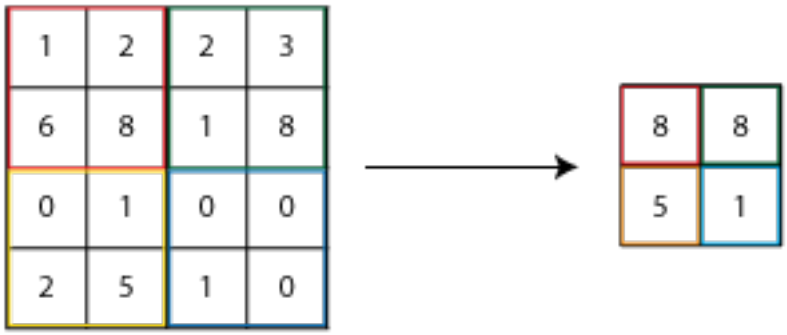
размера на изхода.

В конволюционните слоеве се използва схема на споделяне на параметри за контрол на броя на свободните параметри. То се основава на следното: ако даден модел е полезен в определен регион, то той би бил полезен дори да е намерен и на друга позиция. Тъй като всички неврони в един изрязък по отношение на дълбочина споделят един и същи параметри, изходът от всяка част от конволюционния слой може да се изчисли като конволюция от теглата на невроните към входните данни. Следователно, често срещано е да наричаме множество от тегла филтър (или ядро), който е свързан с вхдоните данни. Резултатът от конволюцията е активационна карта, и множеството такива карти за всеки филтър се напасват заедно по отношение на дълбочината за да генерират изходните данни. Важно е да се отбележи, че понякога споделянето на параметри може да не е подходящо. Такъв пример е когато входните изображения за невронната мрежа имат нещо специфично в центъра на изображенията и ние очакваме филтрите да бъдат намерени точно в централната част. Пример на практика е разпознаване на лица, където лицата са в центъра на изображението. В този случай избягваме споделянето на параметри и просто разчитаме на локално свързани слоеве.

Обединяващ (Pooling) слой

Друга важна концепция в конволюционните невронни мрежи е обединяването (pooling), което е вид нелинейно намаляване на пространство. Има няколко нелинейни функции, които имплементират обединяването. Най-популярната от тях е max pooling 2.3. При нея изображението се разделя на множество от непресичащи се правоъгълници и за всеки такъв регион връща максимума. Интиутивното обяснение е, че щом даден модел е бил намерен, неговото точно местоположение не е от толкова голямо значение колкото относителното му местоположение спрямо други модели. Основната роля на този слой е прогресивно да намали размера на данните в мрежата и следователно да се избегне прекомерното нагаждане. Този слой се изпълнява независимо за всяка дълбочина на входа и променя размерите независимо. Най-често срещаната форма в прилагане на pooling функция с филтри с

размери 2x2 като по този начин се изоставят 75% от активациите. По този начин всяка операция за максимизиране ще търси най-голямото от 4 числа. Измерението за дълбочина остава непроменено. Други видове функции, които биха могли да бъдат приложени са средно аритметично намаляване и L2-нормализация. До скоро средно аритметичното намаляване е било предпочитано, но в последните години максимизирането дава по-добри резултати в практиката.



Фигура 2.3: Обединяващ (pooling) слой

Слой на ректифицираните линейни единици (ReLU - Rectified Linear Units)

Това е слой от неврони, който прилага активационната функция f(x)=max(0,x). Той повишава нелинейните свойства на функцията за взимане на решения и като цяло на мрежата без да оказва влияние на областите на възприемане на конволюционния слой. Използват се и други функции за повишаване на нелинейността. Например f(x)=tanh(x) и f(x)=|tanh(x)|. ReLU е най-предпочитана в сравнение с другите функции, защото прави процеса на обучение на мрежата много по-кратък и няма големи разлики по отношение на точността.

Напълно свързан слой

Накрая след няколко конволюционни и обединяващи слоя взимането на решения се осъществява от напълно свързаните слоеве. Невроните в тях имат връзки със всички активации в предишния слой както при стандартните невронни мрежи. Тяхната активация би могла да се пресметне чрез матрица на умноженията, последвана от контролирано отклонение.

Слой на наказанията

Този слой указва как мрежата наказва разликата между предсказаните и истинските стойности и обикновено е последният слой в мрежата. Има различни функции за наказание в зависимост от конкретната задача. Softmax е една от тях и се използва, когато трябва да се предскаже 1 от няколко взаимно изключващи се класа. Това е и конкретният случай. Крос ентропичен сигмоид се използва при предсказване на K независими вероятности в диапазона [0,1].

2.5.4 Избор на хиперпараметри

Брой филтри

Тъй като с навлизане в невронната мрежа размерът на данните се намалява е удачно да имаме по-малко филтри в началото и повече в края. За да се уеднакви изчислителното време във всеки слой произведението на броя пиксели и броя модели е почти константно из всичките слоеве. За да се запази входната информация трябва да се осигури, че броят активации не трябва да намалява от един слой към следващия.

Форма на филтрите

Има много различни форми на филтрите и обикновено се избират в зависимост от входните данни.

Максимални размери на формата за обединяване

Обикновено обединяването се извършва върху 2x2 пиксела. При огромни входни данни може да бъде 4x4. Това обаче може да доведе до премахването на твърде много важна информация.

2.5.5 Методи за регуларизация

В сферата на машинното самообучение, под регуларизация се разбира процесът на добавяне на информация с цел да се реши проблемът с прекомерното нагаждане.

Ранно прекратяване (Early stopping)

Това е един от най-простите методи за предотвратяване на прекомерното нагаждане. Той предлага просто да прекратим тренирането преди нагаждането да може да се случи. Недостатъкът е, че обучението се прекратява преждевременно. Един от начините за прилагане на ранното прекратяване е да се следи грешката върху тренировъчното и валидационното множество и когато се види, че грешката върху валидационното спира да пада се спира обучението.

Ограничаване на броя параметри

Друг много прост подход е да се ограничат броят параметри в невронната мрежа. Обикновено се ограничава броят на скритите единици във всеки слой или дълбочината на невронната мрежа. За конволюционни невронни мрежи, размерът на филтрите също оказва влияние на броя на параметрите. Ограничаване на броя параметри намалява директно възможността на мрежата да предскаже резултат. Намалява се сложността на операциите, които се извършват върху данните и по този начин се намалява възможността за прекомерно нагаждане.

Изкуствено добавяне на тренировъчни данни (Artificial data)

Тъй като прекомерното нагаждане на модела се определя от неговата специфика и количеството тренировъчни данни, генерирането на допълнителни тренировъчни примери може значително да намали overfitting-а. Един от популярните подходи за генериране на нови примери е да се променят настоящите и да се подадат като нови примери. Например дадено изображение може да бъде завъртяно или изрязано на произволен принцип. Основна идея на този метод е моделът да не се преспецифицира към конкретното място, на което е показан обектът, но да е готов да го срещне и на друго място.

Разпадане на теглото (Weight decay)

При този подход ръчно добавяме допълнителна грешка към обучителните данни. Има различни начини за избиране на грешката

- сума на всички тегла (L1 нормализация) или корен квадратен от сбора на квадратите на теглата (L2 нормализация). Може да се намали сложността на големи модели като се уголеми наложеното наказание върху векторите.

Метод на отпадане (Dropout)

Тъй като напълно свързаният слой има достъп до повечето параметри, той е податлив на прекомерно нагаждане (overfitting). Методът на отпадане е емпиричен и се грижи това да не се случи. На всяка итерация на трениране всяка нишка или отпада с вероятност 1-p или остава с вероятност p. По този начин остава редуцирана мрежа като входящите и изходящите нишки също се премахват. След това само редуцираната мрежа се тренира. След това премахнатите елементи се добавят на ново в мрежата с първоначалните им тегла. По време на етапите на трениране, вероятността даден елемент да се запази е обикновено 0.5. За входящи нишки вероятността трябва да е много по-висока, защото информацията просто изчезва, когато игнорираме входящи данни. Като не тренираме върху всички връзки от тренировъчните данни, методът на отпадане намалява прекомерното нагаждане в невронните мрежи. Освен това той значително подобрява времето за трениране. Това прави комбинациите на модела подходящи дори за много дълбоки невронни мрежи. Тази техника намалява сложността на взаимодействието между слоевете и спомага за научаването на по-генерални модели. Използва се за подобряване на продуктивността на невронни мрежи при задачи като визуално и гласово разпознаване, класификация на документи и биологични проблеми.

Метод на премахване на връзките (DropConnect)

Този метод е генерализация на Dropout, при която всяка връзка (вместо всяка изходяща единица) може да отпадне с вероятност 1-p. По този начин всеки неврон получава данни от произволни единици от предишния слой. Този метод е много подобен на Dropout, защото позволява динамично разреждане на модела, но се различава по това, че премахването на данни е от теглата, а не от изходящите вектори на слоя. С други думи, при този подход напълно свързаният слой

става частично свързан, като връзките, които остават са избрани на произволен принцип.

Стохастично обединяване (Stochastic pooling)

При стохастичното обединяване се оформят региони на действие. За всеки регион се избира активация според полиномиално разпределение, което е следствие от действията в този регион. Този подход не зависи от външни параметри и може да бъде комбиниран с други методи за регуларизация. Друг начин на представяне на този метод е, че той е същият като максимално обединение (max pooling), но върху много копия на едно и също изображение, като всяко от копията има малки локални промени. Методът е подобен на предварителна обработка на входните изображения, което води до много добро представяне в някои случаи. Използването на метода при архитектура с многослоен модел води до експоненциален брой деформации, защото промените в началните слоеве са независими от тези в последващите.

2.6 Състезанието на ImageNet

2.6.1 Въведение

Състезанието на ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [27] оценява алгоритми за разпознаване на обекти и класифициране на изображения в огромни мащаби. Една от основните цели е да позволи на изследователите в тази област да сравнят постигнатите от тях резултати при наличието на голямо разнообразие от данни, възползвайки се от ръчно обработени картинки. Друга полза е да се види докъде е стигнал глобалният прогрес по разпознаване на изображения и автоматичната им анотация.

Състезанието е ежегодно и в него се включват най-големите компании в областта. За всяко издание се организира конференция, на която се споделят най-добрите методи и резултатите от текущото издание.

2.6.2 Подробен преглед на победителя в ILSVRC 2015

Победителите са от Microsoft Research Asia (MSRA). Те предоставят две архитектури: директна (plain) и остатъчна (residual) [12]. При директната конволюционните слоеве имат филтри с размери 3x3 и следват 2 основни правила:

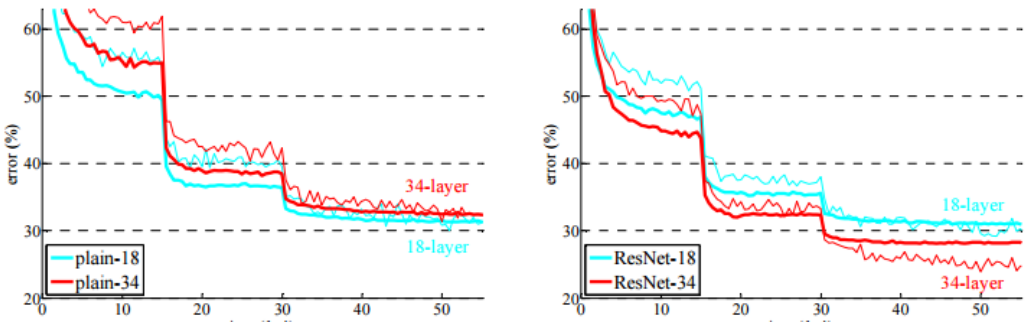
1. За даден размер на изхода, слоевете имат същия брой филтри.
2. Ако размерът на функция на картата е намалена наполовина, броят на филтрите се удвоява, така че да се запази сложността.

Стъпката (stride) е 2. Мрежата свършва с напълно свързан слой с 1000 неврона и softmax активационна функция. Общият брой слоеве са 34. Остатъчната мрежа е базирана на първата като се добавят кратки пътища 2.5.

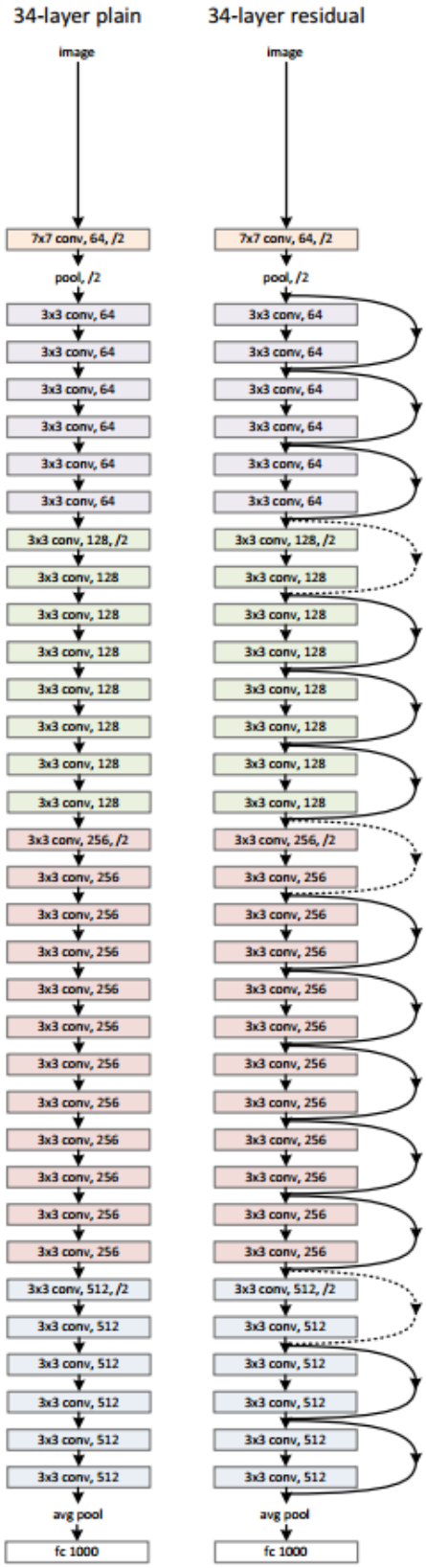
Те могат да се използват, когато входът и изходът са с едни и същи размери. Когато са с различни размери има два подхода:

1. Добавят се нули, за да се уеднакви размерът.
2. Използва се проекция, за да се уеднакви размерът.

Авторите сравняват мрежи с различен брой слоеве, като на 2.4 са показани резултатите при 16 и 34 слоя за стандартни и остатъчни мрежи:



Фигура 2.4: Графики на резултатите при 16 и 34 слоя за стандартни и остатъчни мрежи [12]



Фигура 2.5: Сравнение на архитектурата на стандартна и остатъчна мрежа [12]

Първоначално изображенията се намаляват до размери [256, 480], след което се взима произволен участък с размери [224, 224]. За класификационната задача тестовете са извършени върху 1000 класа с 1 280 000 тренировъчни изображения и 50 000 тестови.

Изпробвани са остатъчни невронни мрежи с 16, 34, 50, 101 и 152 слоя. Има и 2 вида мерки за точност: при първия (top 1) се връща само 1 клас и точността е процентът изображения, за които правилният клас е избран. При втория (top 5) един опит се счита за успешен ако търсеният клас е един от първите 5 върнати класа от невронната мрежа. Най-добрите постигнати резултати са с мрежа със 152 слоя (19.38% и 4.49% грешка съответно за top 1 и top 5) 2.1.

Модел	**top 1 грешка**	**top 5 грешка**
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Таблица 2.1: Резултати при различните архитектури на невронни мрежи [12]

3 Използвани технологии и платформи

3.1 Обучителни данни

ImageNet е огромна база от данни с изображения, предназначена за използване от разработчици на софтуери за разпознаване на изображения. От 2016 година, повече от 10 милиона изображения са анотирани ръчно от ImageNet, за да покажат какви обекти са показани. Очертаващи линии на обектите в изображенията са дадени за над 1 милион картинки [15]. Базата с изображения, както и техните анотации е свободно достъпна в ImageNet, но те не са тяхно притежание.

Процесът на анотация се извършва чрез външни хора. Анотациите на ниво изображение индикират наличието или липсата на даден клас в картинката. Анотациите, които са за конкретен обект от картинката предоставят и ограждащи граници на обекта (видимата му част върху изображението). ImageNet използва йерархична словесна схема за организация като поддържа клас куче и над 120 под-класа, които са различни породи кучета. Това позволява избор на разработчиците колко прецизно и на какво ниво на абстракция искат да е класифицирането.

С цел обучение бяха изтеглени 57 класа животни, съдържащи между 1500 и 2000 изображения за клас. Те са с различни размери и обхващат разнообразни фонове и среди. Изтеглените класове са показани на Фигура 3.1.

Номер	**Име**	**Идентификатор**
0	alligator	n01698434
1	antelope	n02419796
2	bat	n02139199
3	bear	n02131653
4	bison	n02410509

5	butterfly	n02274259
6	camel	n02437136
7	canary	n01533339
8	cat	n02121808
9	cattle	n04052442
10	chicken	n01791625
11	cockroach	n02233338
12	deer	n02430045
13	dog	n02084071
14	donkey	n02389559
15	duck	n01846331
16	eagle	n01613294
17	elephant	n02503517
18	fly	n02190166
19	fox	n02118333
20	frog	n01641739
21	giraffe	n02439033
22	goat	n02416880
23	goldfish	n01443537
24	horse	n02374451
25	hyena	n02116738
26	iguana	n01677366
27	jaguar	n02128925
28	kangaroo	n01877134
29	lion	n02129165
30	llama	n02437616
31	monkey	n02484322
32	mosquito	n02200198
33	mouse	n02330245
34	octopus	n01970164
35	ostrich	n01518878
36	owl	n01621127
37	panda	n02510455
38	parrot	n01817346
39	penguin	n02055803
40	pig	n02395406

41	pigeon	n01814921
42	piranha	n02584449
43	rabbit	n02324587
44	raven	n01579260
45	scorpion	n01770393
46	shark	n01482330
47	sheep	n10588074
48	snail	n01944390
49	snake	n01726692
50	spider	n01772222
51	swan	n01858441
52	tiger	n02129604
53	turkey	n01794344
54	whale	n02062744
55	worm	n01922303
56	zebra	n02391049

Таблица 3.1: Обучително множество от класове

3.2 Технологии, платформи и методологии

Една от основните причини разпознаването на изображение да бъде възможно е напредъкът на хардуера и възможността да използваме мощността и бързия достъп до паметта на видео картите (GPU). Един от най-удобните начини това да се случи се предоставя от CUDA [22]. Като програмен език е избран Python [25], който е най-популярното решение в тази сфера, защото предоставя лесна интеграция с интерфейси за достъп до видео картата и има много популярни и добри библиотеки за изкуствен интелект. За изграждане на конволюционните невронни мрежи се използва TensorFlow [29]. За управление на версиите се разчита на скрито Git хранилище в Bitbucket [4].

3.2.1 CUDA

CUDA е платформа за паралелна обработка и програмен интерфейс (API), създаден от Nvidia. Той позволява на софтуерните разработчици да използват видео карти, които позволяват CUDA за обработка с каквато и да е цел - този подход се нарича GPGPU (General-Purpose computing on Graphics Processing Units). CUDA платформата е софтуерен слой, който позволява директен достъп до видео картата и елементите за паралелна обработка за всякакви изчислителни цели. [28]

CUDA е предназначен за работа с програмни езици като C,C++ и Фортран. Тази достъпност прави по-лесен за специалистите по паралелна обработка да използват ресурсите на видео картата за разлика от други услуги като Direct3D и OpenGL, които изискват добри умения в графичното програмиране. Също така CUDA поддържа програмни среди като OpenACC и OpenCL. Когато за първи път е представен от Nvidia, CUDA е бил абревиатура на Compute Unified Device Architecture, но впоследствие Nvidia се отказва от съкращението.

CUDA предоставя услуги както на ниско, така и на високо ниво на абстракция. Първата услуга CUDA пуска в публичното пространство на 15.02.2007г. с поддръжка на Windows и Linux. Mac OS X поддръжка се добавя във версия 2.0. CUDA работи с всички видео карти на Nvidia и повечето стандартни операционни системи.

3.2.2 Python

Python е глобално използван динамичен език за програмиране от високо ниво [30]. Философията на неговия дизайн набляга на четимост на кода и неговият синтаксис позволява на програмистите да имплементират концепции с по-малко линии код в сравнение с други езици като C++ и Java. Езикът предоставя конструкции, които са предназначени за писане на чист код както за малки, така и за големи програми. [32]

Python поддържа множество програмни парадигми като обектно-ориентирано, императивно, функционално програмиране и процедурни стилове. При него има динамично типизиране и автоматично управление на паметта. Има много и качествени налични библиотеки.

Python интерпретатори са налични за много операционни системи, което позволява да се изпълнява Python код при почти всякакви системи. CPython, както и повечето различни имплементации на Python са с отворен код.

Структура и функционалност

Python предлага добра структура и поддръжка за разработка на големи приложения. Той притежава вградени сложни типове данни като гъвкави масиви и речници, за които биха били необходими дни, за да се напишат ефикасно на C.

Python позволява разделянето на една програма на модули, които могат да се използват отново в други програми. Също така притежава голям набор от стандартни модули, които да се използват като основа на програмите. Съществуват и вградени модули, които обезпечават такива неща като файлов вход/изход (I/O), различни системни функции, сокети (sockets), програмни интерфейси към GUI-библиотеки като Tk, както и много други.

Тъй като Python е език, който се интерпретира, се спестява значително време за разработка, тъй като не са необходими компилиране и свързване (linking) за тестването на дадено приложение. Освен това, бидейки интерпретируем език с идеология сходна с тази на Java, приложение, написано на него, е сравнително лесно преносимо на множеството от останали платформи (или операционни системи).

Програмите, написани на Python, са доста компактни и четими, като често те са и по-кратки от еквивалентните им, написани на C/C++. Това е така, тъй като:

- наличните сложни типове данни позволяват изразяването на сложни действия с един-единствен оператор;
- групирането на изразите се извършва чрез отстъп, вместо чрез начални и крайни скоби или някакви други ключови думи (друг език, използващ такъв начин на подредба, е Haskell);
- не са необходими декларации на променливи или аргументи.
- Python съдържа прости конструкции, характерни за функционалния стил на програмиране, които му придават допълнителна гъвкавост

Всеки модул на Python се компилира преди изпълнение до код за съответната виртуална машина. Този код се записва за повторна употреба като .pyc файл.

Програмите написани на Python представляват съвкупност от файлове с изходен код. При първото си изпълнение този код се компилира до байткод, а при всяко следващо се използва кеширана версия. Байткодът се изпълнява от интерпретатор на Python.

- Строго типизиран (strong typing) – При несъответствие между типовете е необходимо изрично конвертиране.
- Динамично типизиран (dynamic typing) – Типовете на данните се определят по време на изпълнението. Работи на принципа duck typing – Оценява типа на обектите според техните свойства.
- Използва garbage collector – вътрешната реализация на езика се грижи за управлението на паметта.
- Блоковете се формират посредством отстъп. Като разграничител между програмните фрагменти използва нов ред.

3.2.3 Tensorflow

Tensorflow е софтуерна библиотека с отворен код за числови изчисления с помощта на графи за предаване на данните (data flow graphs). Всеки връх в графа представлява математически операции, докато пътищата са многомерни масиви (тензори - tensors), които служат за комуникация между тях. Архитектурата позволява да се извършват изчисления на повече от един процесор или видео карта както на десктоп, така и на сървър или мобилно устройство използвайки един единствен програмен интерфейс (API). Tensorflow първоначално е разработен от изследователи и инженери, работещи в Google по проекта Google Brain [11] по разработка за изследователската организация Google's Machine Intelligence. Първоначалната им цел е била изследване в сферите на машинното самообучение и невронните мрежи, но системата е достатъчно генерална, за да се прилага свободно и в други области. [29]

Въведение

Проектът Google Brain стартира през 2011, за да изследва използването на дълбоки невронни мрежи с много големи мащаби както с изследователска цел така и за да се използва от Google. Като част от началната работа по този проект първо е построен DistBelief, система от първа генерация за скалируемо трениране. Благодарение на DistBelief са извършени множество изследователски дейности като обучение без надзор, репрецентация на езици, модели за класифициране на изображения, разпознаване на обекти, класификация на видеа, разпознаване на реч, игра на Go, разпознаване на граждани и множество други сфери [1]. Освен това често след взаимодействие с екипът Google Brain, повече от 50 други екипи от Google и други комнпании са използвали дълбоки невронни мрежи с помощта на DistBelief в много продукти, сред които Google Search, рекламни продукти, системи за разпознаване на реч, Google Photos, Google Maps и StreetView, Google Translate, YouTube и много други.

На база на опита, придобит от DistBelief и след по-добро разбиране на това какво е нужно да има такава система е създаден Tensorflow - система от втора генерация за имплементиране и използване на модели за машинно самообучение от големи мащаби. Tensorflow чете изчисленията с модел, който използва поток от данни и ги нагажда спрямо голямо разнообразие от хардуерни платформи. Той може да се използва както от мобилни устройства с Android или iOS, така и от отделни машини с една или повече видео карти. Налична е поддръжката и на паралелна работа върху много машини. Разполагайки с една система, която може да се изпълнява върху голям набор от хардуерни платформи значително опростява използването на системи за машинно самообучение. Изчисленията в Tensorflow са представени като графи на данните със състояния. Той позволява бързи експерименти с нови модели за изследователски цели и предоставя много висока производителност. С цел да се скалира тренирането на невронни мрежи за по-големи системи, Tensorflow позволява лесно да се изградят паралелни операции на различни машини, които обновяват множество споделени параметри и състояния. Лесният за употреба интерфейс позволява да се постигат различни техники за паралелна обработка с много малко

усилия. Някои употреби на Tensorflow позволяват гъвкавост по отношение на цялостност при обновяването на параметрите и това много лесно може да се използва при големи системи. В сравнение с DistBelief, програмният модел на Tensorflow е по-гъвкав, неговата производителност е значително по-добра и поддържа използване на по-голямо множество от модели върху повече хардуерни платформи.

Програмен модел и основни концепции

Изчисленията в Tensorflow представляват насочен граф, който е изграден от множество върхове. Графът представлява изчисление на поток от данни с разширения, които позволяват някои пътища да не се променят и дават контрол вътху разклоненията и циклите на пътищата в графа. Най-честите употреби на Tensorflow са чрез езиците за програмиране C++ и Python.

В графа на TensorFlow всеки връх има 0 или повече входа и 0 или повече изхода и представлява инстанция на една операция. Стойности, които се пренасят по стандартните пътища в графа се наричат тензори. Те са масиви с променливи измерения, при които типът на данните се определя при създаване на графа. Има и специални пътища, които контролират зависимостите. През тях не се предават данни, но те синхронизират работата на останалите пътища и определят кога дадена операция да започне да се изпълнява. Tensorflow понякога автоматично добавя такива специални пътища, за да въведе ред над независими до преди този момент операции. Това се прави с различни цели като например оптимизирано използване на паметта.

Операции и ядра

Всяка операция има име и представлява абстракция на изчисление (например умножение на матрици или събиране). Операциите могат да имат атрибути и всички атрибути трябва да бъдат дефинирани по време на изграждане на графа, за да се създаде връх в него, който да отговаря за операцията. Една от честите употреби на атрибутите е една операция да се направи полиморфна върху различни типове данни.

Под ядро се разбира конкретна имплементация на операция, която може да се изпълни върху конкретен вид устройство

(например процесор или видео карта). Tensorflow дефинира видовете операции и ядра посредством механизъм за регистрация. Това множество може да се разширява чрез свързване на допълнителни дефиниции на операции и/или ядра.

Сесии

Всяка клиентска програма взаимодейства със системата на Tensorflow като създава нова сесия. С цел да създаде граф на изчисленията, интерфейсът на сесията поддържа метод за разширение, за да разшири графа с допълнителни върхове и пътища (при създаване на сесията графът е празен). Другата основна операция, която се поддържа от интерфейса на сесията е "Стартирай" (Run), която приема множество изходи, които трябва да се изчислят, както и множество тензори, които могат да се добавят към графа. В повечето случаи на употреба на Tensorflow след като веднъж се създаде сесия с граф, върху нея се извършват хиляди или милиони стартирания.

Операция Variable

При повечето изчисления един граф се използва много пъти. Повечето тензори не оцеляват по време на едно изпълнение на графа. Variable обаче е специален вид операция, която генерира връзка към постоянен непроменящ се тензор, който се запазва след няколко изпълнения върху графа. Тези връзки мгоат да бъдат предадени към специални операции като Assign или AssignAdd, които изменят съответния тензор. За приложенията на Tensorflow, свързани с машинно самообучение, параметрите на модела обикновено се пазят в тензори, които се пазят в променливи (Variable) и се обновяват, когато се стартира тренировъчния граф за модела.

Алтернативи

Theano също е библиотека в Python за цифрови изчисления. При нея те са представени с помощта на Numpy синтаксис [3] и могат да се използват както върху процесор, така и върху видео карта. Theano е с отворен код и е основно разработен от група за машинно самообучение от Университета на Монреал.

Caffe е библиотека, написана на C++. Първо е започната като докторантски проект в UC Berkeley, след което става с отворен код и за развитието ѝ допринасят много програмисти.

Има и други алтернативи на Tensorflow като Torch, CNTK (Computational Network Toolkit) на Microsoft, DSSTNE (Deep Scalable Sparse Tensor Network Engine) на Amazon.

3.2.4 Keras

Keras е библиотека за невронни мрежи от високо ниво. Написана е на Python и може да използва Tensorflow или Theano. Първоначалната ѝ цел е била да позволи бързи и лесни експерименти. Философията на създателите е, че възможността да стигнеш от идея до резултат за възможно най-кратко време е ключът към успешните изследователски дейности.

Сред основните предимства на библиотеката са:

- Модулярност: Слоевете на невронната мрежа, оптимизаторите, активационните функции, схемите за регуларизация са независими модули, които могат да се комбинират, за да създадат нов модул.
- Опростена имплементация: Всеки модул е кратък и лесен за разбиране. Всеки ред код е интуитивен за четене още при първия преглед на имплементацията.
- Възможност за разширяване: Много е лесно да се добавят нови модули, а за вече съществуващите са предоставени множество примери. Това прави Keras подходящ и за по-сложни изследователски цели.
- Директна работа с Python. Няма нужда да се добавят допълнителни модули или конфигурационни файлове.

3.2.5 Git

Git е система за управление на версиите, следене на промените на компютърни файлове и координиране на работата по тези файлове с други хора. Той основно се използва за софтуерна разработка, но може да се използва и за други цели. Основните му приоритети са

скорост, интегритет на данните и поддръжка на дистрибутирани, нелинейни процеси.

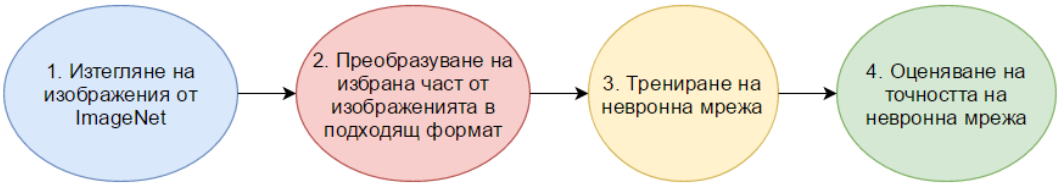
Git е създаден от Линус Торвалдс през 2005 година, за да се разработи ядрото на Линукс заедно с други разработчици, които допринасят за първоначалния му напредък.

Както и повечето други системи за управление на версиите (и за разлика от повечето системи с начин на комуникация клиент-сървър), всяка Git директория на всеки компютър съдържа в себе си цялата информация и история за хранилището. Това позволява пълно следене на версиите, независимо от наличието на мрежа или централен сървър.

4 Реализация, експерименти и анализ на резултатите

4.1 Реализация

Системата за класификация на изображения е разработена на Python, използвайки PyCharm [24]. 4.1 показва процеса на работа на системата.



Фигура 4.1: Начин на работа на системата

4.1.1 Събиране на данни и подготовка за обработка

Подготвени са 57 класа животни, показани в 3.1. **imagenet_metadata.py** съдържа информация за тях във формат:

<индекс, наименование, идентификатор в ImageNet>

image_crawler.py изтегля всички снимки от изваденитe класове в директория, която е релативно разположена до основната. Създава се папка _DATA, в нея папка ImageNet и в нея по една папка за всеки клас изображения. В нея съответно са изображенията.

Файлът **bin_generator.py** създава файлове с разширение .bin, които представляват изображенията в удобен за четене формат. От картинките се взимат толкова класа, колкото е посочено в конфигурацията. Преди да се запишат картинките се преоразмеряват отново в зависимост от конфигурацията. За

тестовете, извършени на локална машина, картинките се представят в размер 64x64 пиксела.

4.1.2 Трениране и оценка на конволюционни невронни мрежи

Файловете в директорията main отговарят за тренирането и оценка на невронни мрежи върху изтеглените и обработени изображения. **common.py** съдържа общи за системата функционалности. В **models.py** са дефинирани използваните архитектури. **train.py** зарежда избраната невронна мрежа и посочени от потребителя класове изображения и тренира върху тях. След това тя запазва модела, за да може да се преизползва в папката на генерираните .bin файлове. **eval.py** използва този модел и дава подробна оценка за точността разпределена по обучаваните класове. Имплементирана е крос валидация, която помага за по-точната оценка на моделите. **retrain.py** взима съществуващ модел, налага промени върху него (замразява слоеве, добавя нови, премахва ги) и продължава тренирането на избраната мрежа.

4.1.3 Конфигурация и основни дефиниции

Стартовата точка на системата е **main.py**. Нейната единствена роля е да предостави избор на потребителя какво иска да направи в текущата итерация на програмата. Предоставят се 4 възможности:

1. Изтегляне на данни
2. Превръщане на изображенията в подходящ за четене от програмата формат
3. Трениране на конволюционна невронна мрежа
4. Оценяване на вече трениран модел на конволюционна невронна мрежа

За всички пътища в системата се грижи **definitions.py**. Той е в основната директория на програмата и съдържа всички нужни пътища в системата като например къде да се запазят изтеглените изображения, къде да се запазят натренираните модели и други.

Всички конфигуриации на системата се контролират от **src/config.py**. Параметрите, които подлежат на конфигурация са следните:

1. **IMAGE_SIZE**: Размер на изображението в пиксели след обработка
2. **NUM_CLASSES**: Брой класове, върху които да се тренира невронната мрежа
3. **EPOCHS**: Брой итерации, които се извършват при трениране на невронна мрежа
4. Други конфигурации на невронната мрежа
5. **CROSS_VALIDATION_ENABLED**: Флаг, указващ дали крос валидацията е позволена
6. **N_FOLD_CROSS_VALIDATION**: На колко части да се разделят данните при използване на крос валидация
7. Имена на файлове, използвани в системата
8. Брой и размер на филтрите в конволюционните невронни мрежи

4.1.4 Тестове върху програмния код на победителя в победителя в ILSVRC 2015

Директорията **pretrained** съдържа кода на едно от най-добрите решения до момента в областта [12]. Той е публичен [18] и предоставя функционалности за класифициране на изображения, търсене на отличителни черти (features) в изображения и изкарване на отличителни черти от междинен слой на невронната мрежа. В **test_imagenet.py** са имплементирани тестове върху готовите модели.

4.2 Направени експерименти

4.2.1 Избор на архитектура

Извършените опити за трениране на невронна мрежа се основават на 3 архитектури и техни разновидности с добавката

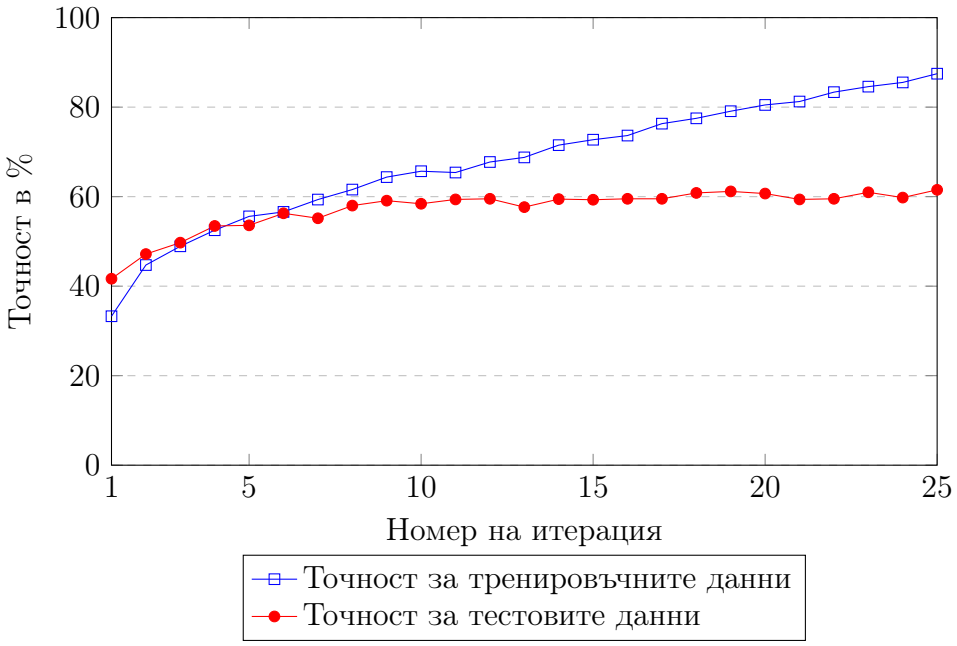
на няколко параметъра. Първата е базова архитектура на конволюционна невронна мрежа, а другите 2 са на база на опити върху други данни с изображения CIFAR10 [6]. Техните разновидности са направени с цел подобряване на резултатите и избягване на прекомерното нагаждане.

Архитектура 0 (4.1) е базова и съдържа само основни слоеве.

№	**Архитектура 0** (8 слоя)
1	Конволюционен
2	Max pooling
3	Конволюционен
4	Max pooling
5	Преминаване към 1 измерение
6	Напълно свързан (softmax активация)

Таблица 4.1: Архитектура 0 на конволюционни невронни мрежи и нейните разновидности

4.2 показва резултатите при базовата Архитектура 0. Върху тестовите данни се забелязва растеж само до 7-8 итерация.



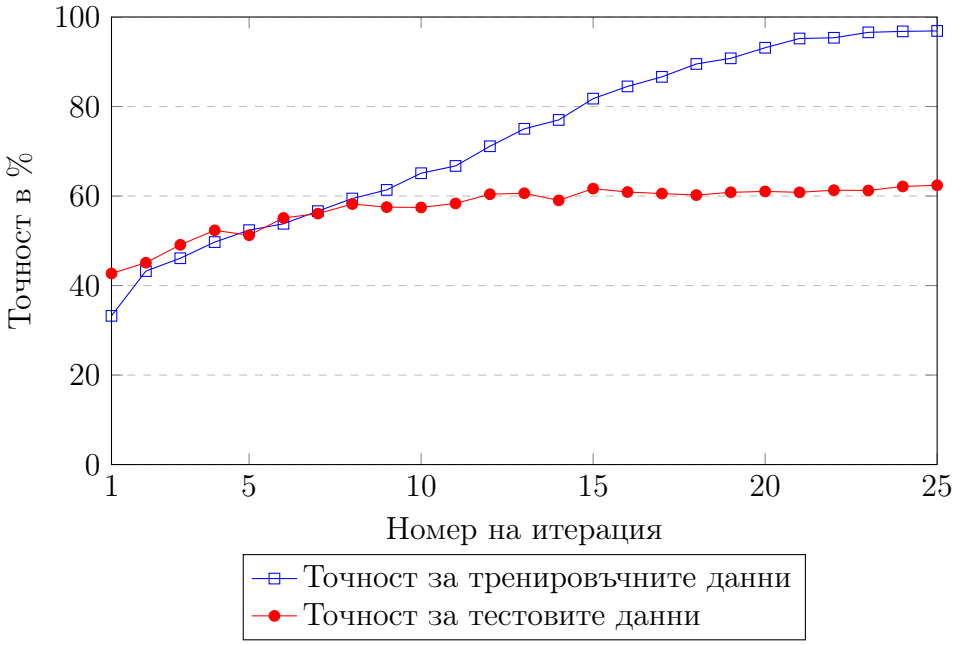
Фигура 4.2: Точност на **Архитектура 0** при **5 класа**

Архитектура 1 (4.2) е кратка и се състои от 8 слоя. Нейната разновидност 1.1 добавя 2 слоя: напълно свързан и слой на отпадане основно с цел да да се избегне прекомерното нагаждане.

№	**Архитектура 1** (8 слоя)	**Архитектура 1.1** (10 слоя)
1	Конволюционен	Конволюционен
2	Dropout 0.2	Dropout 0.2
3	Конволюционен	Конволюционен
4	Max pooling	Max pooling
5	Преминаване към 1 измерение	Преминаване към 1 измерение
6	Напълно свързан (512 неврона)	Напълно свързан (512 неврона)
7	Dropout 0.5	Dropout 0.5
8	Напълно свързан (softmax активация)	**Напълно свързан (256 неврона)**
9	-	**Dropout 0.3**
10	-	Напълно свързан (softmax активация)

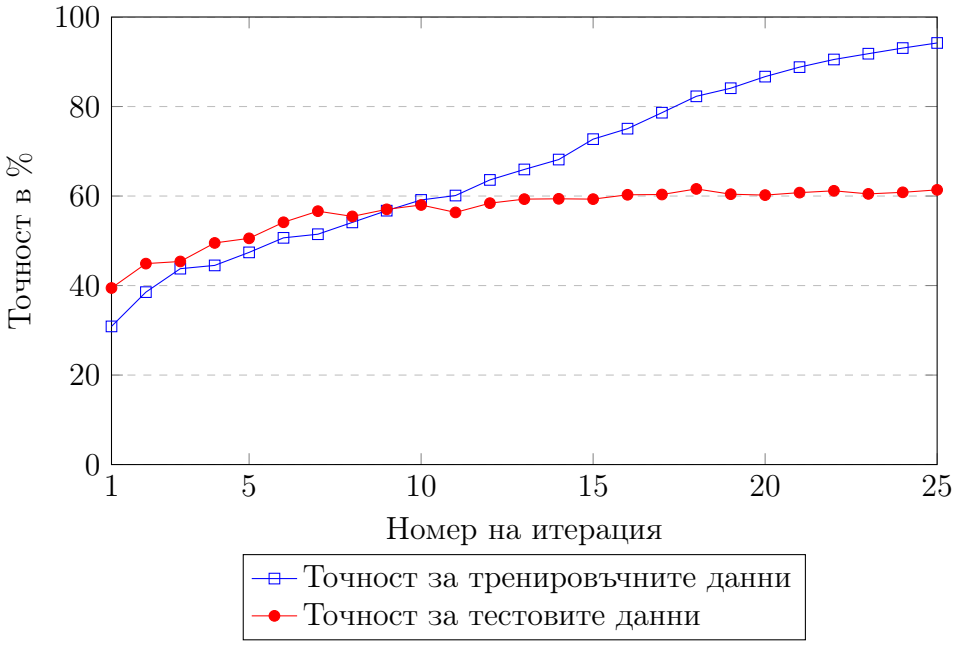
Таблица 4.2: Архитектура 1 на конволюционни невронни мрежи и нейните разновидности

4.3 показва резултатите при Архитектура 1. Значителен растеж се наблюдава до десетата итерация на трениране. След това спира растежа върху тестовите данни и значитено нараства този при тренировъчните. Финалните резултати са 96.89% точност за тренировъчните и 62.41% за тестовите.



Фигура 4.3: Точност на **Архитектура 1** при **5 класа**

4.4 показва резултатите при Архитектура 1.1. Тя е алтернатива на Архитектура 1, основната цел на която е да се избегне прекомерното нагаждане. Опитът е неуспешен, защото се намалява точността при тестовите данни и не се намалява прекомерното нагаждане. Финалните постигнати резултати са съответно 94.20% и 61.38%.



Фигура 4.4: Точност на **Архитектура 1.1** при **5 класа**

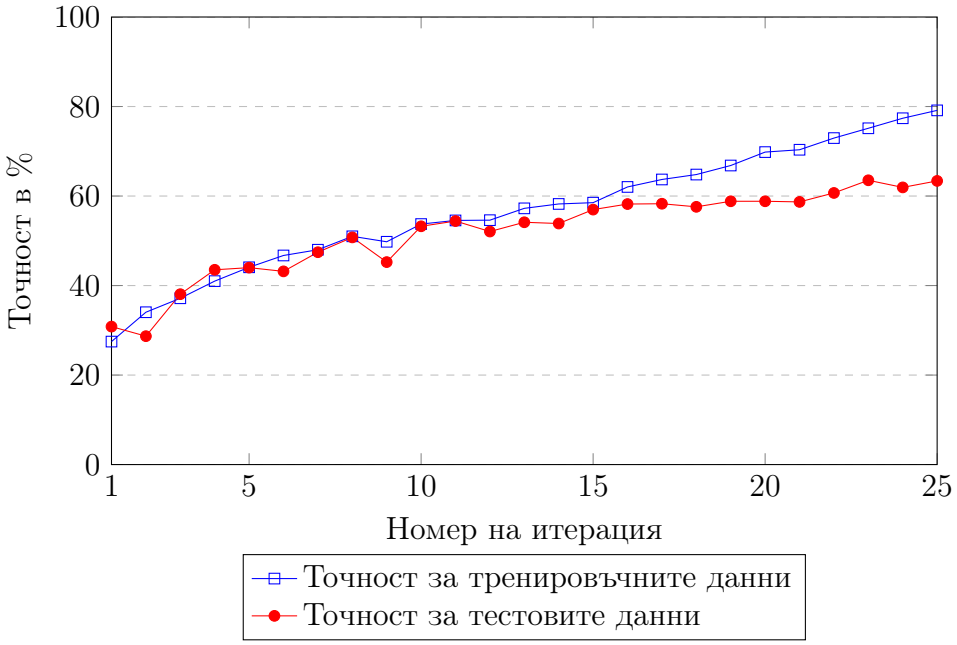
Архитектура 2 (4.3) е по-дълбока: 19 слоя. Тя включва 6 конволюционни слоя, а големината ѝ прави по-трудно запомнянето на голямо количество информация за изображенията и съответно прекомерното нагаждане е в по-малки размери от Архитектура 1. Тя има 2 разновидности: архитектури 2.1 и 2.2, които целят по-малко нагаждане спрямо тренировъчните данни и постигане на по-високи резултати.

№	**Архитектура 2** (19 слоя)	**Архитектура 2.1** (19 слоя)	**Архитектура 2.2** (21 слоя)
1	Конволюционен	Конволюционен	Конволюционен
2	Dropout 0.2	Dropout 0.2	Dropout 0.2
3	Конволюционен	Конволюционен	Конволюционен
4	Max pooling	Max pooling	Max pooling
5	Конволюционен	Конволюционен	Конволюционен
6	Dropout 0.2	Dropout 0.2	Dropout 0.2
7	Конволюционен	Конволюционен	Конволюционен
8	Max pooling	Max pooling	Max pooling
9	Конволюционен	Конволюционен	Конволюционен
10	Dropout 0.2	Dropout 0.2	Dropout 0.2
11	Конволюционен	Конволюционен	Конволюционен

12	Max pooling	Max pooling	Max pooling
13	Преминаване към 1 измерение	Преминаване към 1 измерение	Преминаване към 1 измерение
14	Dropout 0.2	Dropout 0.2	Dropout 0.2
15	Напълно свързан (1024 неврона)	Напълно свързан (1024 неврона)	Напълно свързан (1024 неврона)
16	Dropout 0.2	Dropout 0.2	Dropout 0.2
17	Напълно свързан (512 неврона)	Напълно свързан (512 неврона) **L2 нормализация**	Напълно свързан (512 неврона)
18	Dropout 0.2	**Dropout 0.5**	Dropout 0.2
19	Напълно свързан (softmax активация)	Напълно свързан (softmax активация)	**Напълно свързан (256 неврона)**
20	-	-	**Dropout 0.3**
21	-	-	Напълно свързан (softmax активация)

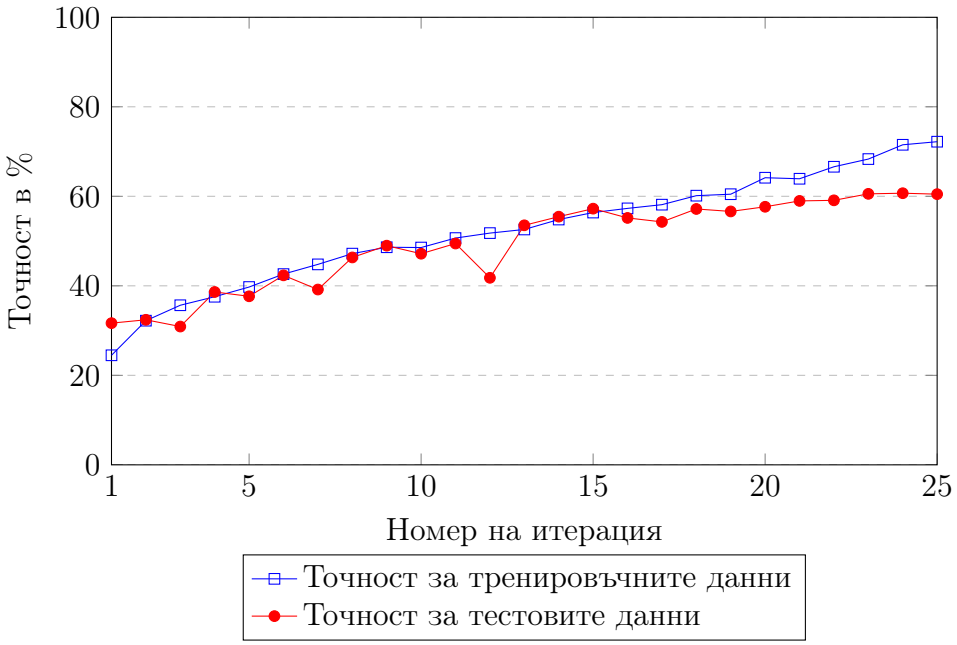
Таблица 4.3: Архитектура 2 на конволюционни невронни мрежи и нейните разновидности

4.5 показва резултатите при Архитектура 2. Тук наблюдаваме растеж на точността върху тестовите данни почти до последната итерация. Тя е малко по-добра от тази при Архитектура 1 и тук прекомерно нагаждане е много по-слабо (разликата между точностите е около 15%). Постигнатите резултати са 79.14% върху тренировъчните данни и 63.38% върху тестовите.

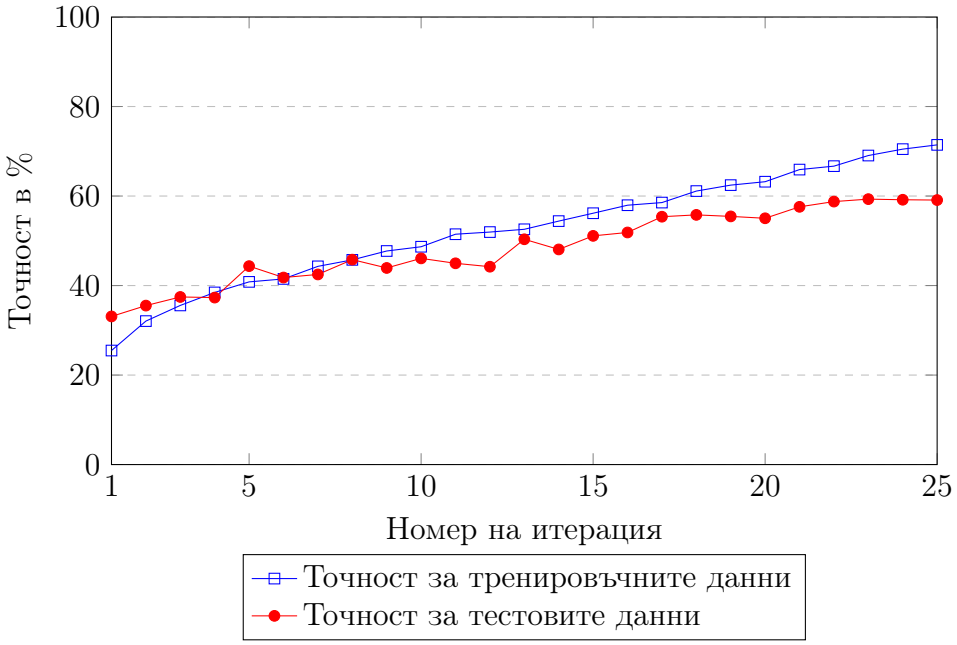


Фигура 4.5: Точност на **Архитектура 2** при **5 класа**

С цел постигане на по-добри резултати и още по-малко прекомерно нагаждане са предложени две алтернативи на Архитектура 2. Резултатите са показани на диаграми 4.6 и 4.7. Разликата между резултатите на последната итерация е около 11%, но на цената на по-лоша точност като цяло.



Фигура 4.6: Точност на **Архитектура 2.1** при **5 класа**



Фигура 4.7: Точност на **Архитектура 2.2** при **5 класа**

4.2.2 Експерименти върху локална машина

Разработката и по-голямата част от експериментите се изпълняват върху локална машина със следните параметри:

- Процесор: Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz 2.60 GHz
- RAM памет: 8GB

За тези експерименти се използва процесор, а не видео карта. В противен случай видео картата е най-важният ресурс. При трениране върху **5 класа** на **Архитектура 1** и преоразмеряване на изображенията на 64x64 пиксела една итерация отнема **362** секунди. Резултатите, получени по време на експериментите са при 25 итерации, което означава, че пълен цикъл на трениране с тези параметри отнема 9050 секунди (приблизително 2 часа 30 минути).

При трениране върху **5 класа** на **Архитектура 2** и преоразмеряване на изображенията на 64x64 пиксела една итерация отнема **650** секунди. При 25 итерации цялото изпълнение отнема около 4 часа и 30 минути.

4.2.3 Експерименти върху Amazon cloud инстанция

Amazon е лидер в предоставянето на облачни услуги. В момента благодарение на тях е достъпно и лесно да се вземат ресурси и да се реши подобен проблем. Предлагат се множество различни видове машини в зависимост от конкретните нужди. Заплаща се само времето, в което реално е използвана машината.

С експериментални цели бе наета Amazon cloud инстанция p2.xlarge, цената на която варираше между 0.25 и 0.30 щатски долара за час: Видео карта: NVIDIA K80 GPU (1) - 12GB (RAM памет: 61GB).

Благодарение на използването на видео картата се постигна оптимизация от около 28 пъти в сравнение с локалната машина (една итерация на трениране върху **Архитектура 1** отнема 13 секунди). Поради по-бързото изпълнение на наетата инстанция на Amazon бяха стартирани множество тестове с различни промени на параметрите като: увеличаване на брой класове, увеличаване на размера на изображенията, увеличаване на размера и броя на филтрите. Резултатите са подробно разгледани в секция 4.3 Анализ на резултатите.

4.3 Анализ на резултатите

4.3.1 Експерименти върху локална машина

По време на тестовете върху локалната машина всички пускания на системата са върху изображения **64x64** пиксела с параметри на входния конволюционен слой: **32** филтъра с размери **3x3 пиксела**. Пробите са извършени върху 5 класа, което означава 5796 изображения за трениране и 1450 за валидиране (тестовете са извършени върху 20% от наличните изображения). 4.8 показва как изглежда едно и също изображение на алигатор преоразмерено съответно на 256x256 128x128 64x64 и 32x32 пиксела.

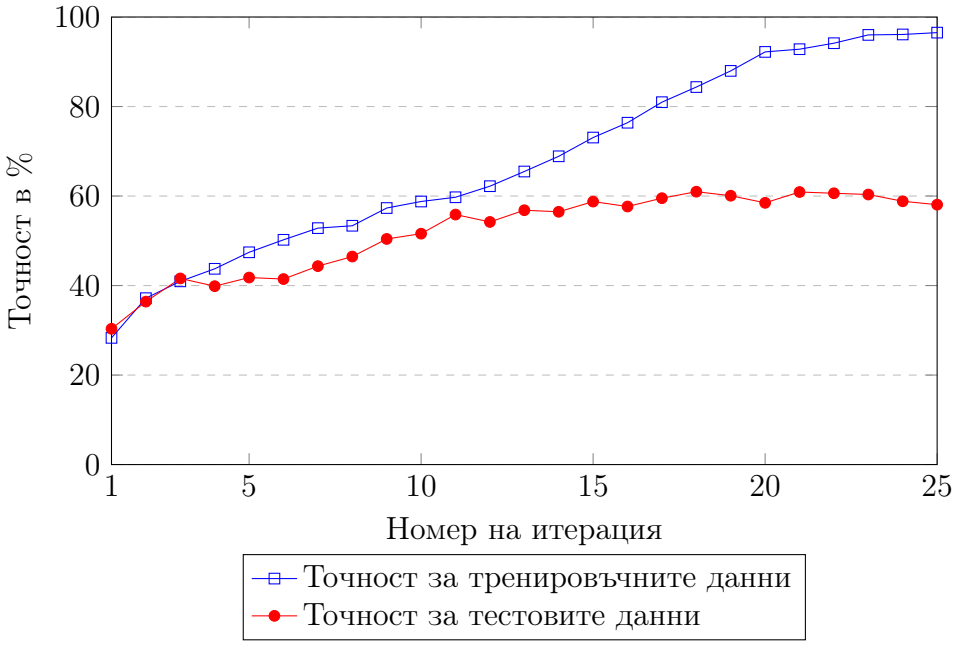
Фигура 4.8: Едно и също изображение преоразмерено съответно на 256x256 128x128 64x64 и 32x32 пиксела

Архитектура	**Точност трен. данни**	**Точност тестови данни**
Архитектура 0	87.47%	61.52%
Архитектура 1	96.89%	62.41%
Архитектура 1.1	94.20%	61.38%
Архитектура 2	79.14%	63.38%
Архитектура 2.1	72.20%	60.48%
Архитектура 2.2	71.43%	59.10%

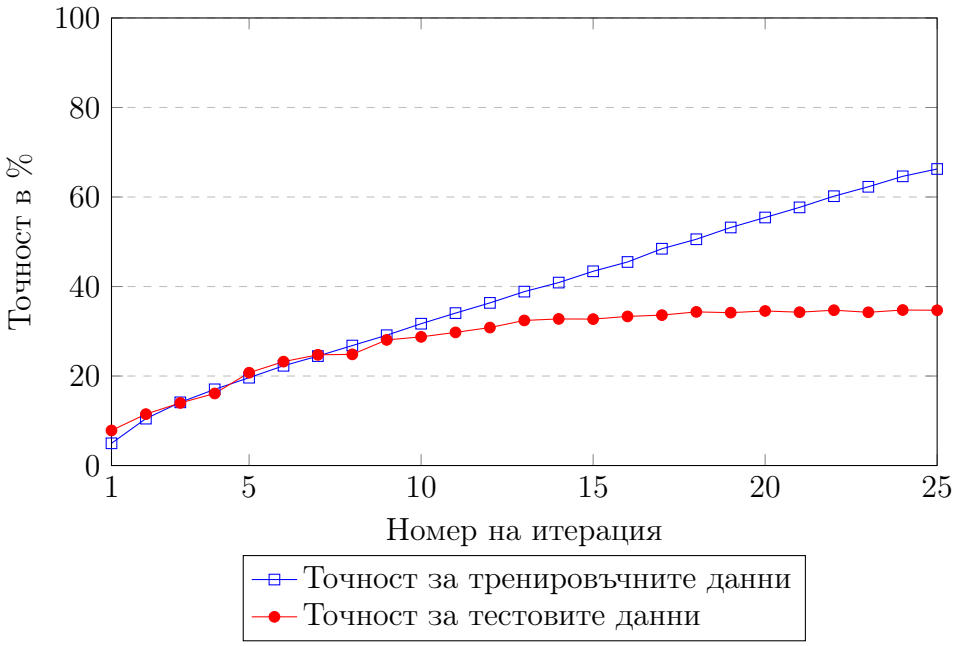
Таблица 4.4: Постигнати резултати при различните архитектури

4.3.2 Експерименти върху Amazon cloud инстанция

Поради много по-бързото изпълнение имаше възможност да се експериментира за повече класове, по-големи изображения и повече филтри. Използването на същите архитектури, но с по-големи изображения или повече филтри не доведе до увеличаване на точността. 4.9 Показва графика на точността по итерации. Постигнатите 60% са по-малко от 63.38% при размер на изображенията 64x64px.



Фигура 4.9: Точност на **Архитектура 2** при **5 класа** и размер на изображенията **168x168px**



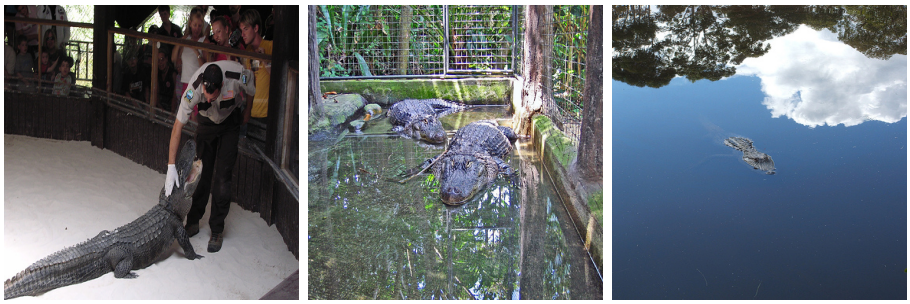
Фигура 4.10: Точност на **Архитектура 2** при **57 класа** и размер на изображенията **64x64px**

При опит за трениране върху всичките 57 класа с **Архитектура 2** една итерация отнема 243 секунди, което означава, че цялото изпълнение на 25 итерации отнема 1 час и 40 минути на инстанцията на Амазон. Тестът е направен с размер на изображенията 64x64

пиксела. За трениране за използвани 62920 изображения, а за валидиране - 15731. Постигнатата точност върху тренировъчните изображения е **66.26%**, а върху тестовите **34.69%**. 4.10 показва прогреса на резултатите за всяка итерация.

4.3.3 Проблемът с прекомерното нагаждане

Както се вижда ясно на графиките с резултатите, при повечето архитектури е налице проблемът с прекомерното нагаждане (overfitting). При някои експерименти разликата в точността между тренировъчните и тестовите данни е над 30%. Чрез леки промени в архитектурата и параметрите на конволюционните невронни мрежи тази разлика падна до 10-12% (Архитектура 2.2), но на цената на по-лоша точност като цяло. Има и друго обяснение за този проблем и то е голямото разнообразие на изображенията. 4.11 показва 3 изображения от един и същи клас - алигатор, които се различават драстично. Именно огромните разлики в това какво точно е показано на картинките довежда до големите разлики между точността на тренировъчните и тестовите данни. Невронната мрежа успява успешно да класифицира първото и второто изображение, но не и третото.



Фигура 4.11: 3 различни изображения от един и същи клас

5 Заключение

5.1 Обобщение на изпълнението на началните цели

Следва списък на началните цели и подробен анализ за постигнатите резултати относно всяка една от тях:

1. **Обзор на проблемната област**.
   Областта е анализирана в детайли. Разгледани са различни приложения на класифицирането на изображения и бъдещето на тази сфера. В детайли са споменати някои от компаниите, които се занимават с това, както и по какво тяхното решение се отличава от останалите.

2. **Какви са условията (финансови, хардуерни и др.), за да построим класификатор с добра точност? Какво е добра точност и каква точност постигат готовите решения?**
   Поради големият обем изображения, нужен за да се тренира добра конволюционна невронна мрежа хардуерът е от основно значение. Дори без да разполагаме с такъв можем да наемем cloud инстанция. С цел анализ, ускоряване на работата и постигане на по-добри резултати беше наета инстанция от Амазон. Тя успя да ускори работата 27-28 пъти в сравнение с локалната машина. Цената на услугата е в зависимост от ползването и варира: 25-40 цента на час за единично стартиране на инстанция и 90 цента на час за постоянна инстанция с гарантирана наличност. Амазон и други доставчици предлагат и по-мощни машини, за да удовлетворят всякакви цели.

   Какво е добра точност е много относителен въпрос и зависи от начина, по който е зададен проблемът. Най-доброто известно решение до момента постига малко над 80% точност за първият

върнат клас (top 1). Ако проблемът се сведе до връщане на редица тагове за дадено изображение, то бихме могли да представим други мерки за точност, като например колко от върнатите тагове са адекватни за изображението и дали най-важните тагове са върнати от системата.

3. **Подбор на данни за обучение.**
   Благодарение на ImageNet подборът на обучителни данни за целите на дипломната работа беше много бърз и лесен. Не така стоят нещата обаче, когато решението трябва да е глобално и да е обучено за всякакви теми и обекти. Някои компании изискват от клиентите си да предоставят тренировъчни изображения ако проблемът, който искат да се разреши е много специфичен.

4. **Реализация на прототип за класификация на изображения.**
   Успешно беше реализиран прототип за класификация на изображения. За целите на дипломната работа са избрани класове, съставени от различни видове животни. Бяха изпробвани различни архитектури на конволюционни невронни мрежи и накрая трениран класификатор за разпонаване на 57 класа животни с 62920 тренировъчни изображения и 15731 тестови.

5. **Експерименти за оценка на точността на класификацията и възможностите за работа с големи данни.**
   Направени са множество експерименти с 2 различни архитектури и техни разновидности (общо 5). Точността е измервана при всеки опит като 20% от изображенията се заделят като тестови. С увеличение на броя на изображенията и с увеличение на техния размер времето за трениране и оценка нараства пропорционално. На наетата от Амазон инстанция при трениране на над 62000 изображения една итерация отнема 4 минути. На локалната машина тя би отнела над 1 час.

6. **Анализ на получените резултати.**
   Резултатите са подробно анализирани. При 5 класа се постига точност над 63% за тестовите и над 96% за тренировъчните

данни. При трениране върху 57 класа се постига точност над 34% за тестовите и над 66% за тренировъчните. При повечето опити е наличен проблемът за прекомерното нагаждане, който може да се обясни както с архитектурите и естеството на конволюционните невронни мрежи, така и с много разнообразните изображения от ImageNet.

5.2 Насоки за бъдещо развитие и усъвършенстване

Изработената система може да бъде развивана и усъвършенствана в много насоки:

1. С цел постигане на по-добри резултати биха могли да се направят още множество експерименти, включително изпробване на други архитектури на невронната мрежа, промяна на хиперпараметрите, избор на нови тренировъчни изображения.

2. За по-бърза работа би могло да се наеме още по-добър хардуер и да се оптимизира процесът да използва повече от една видео карта. Това се позволява от интерфейса на библиотеката Tensorflow.

3. Системата би могла да се специализира в конкретна насока като например разпознаване на лица, идентифициране на туристически обекти или други.

6 Референции

[1] Martın Abadi et al. “Large-Scale Machine Learning on Heterogeneous Distributed Systems”. In: *Google Research* (2015).

[2] Bengio and Yoshua. “Learning Deep Architectures for AI”. In: *Foundations and Trends in Machine Learning* (2009). DOI: 10.1561/2200000006.

[3] J. Bergstra et al. “Theano: A CPU and GPU Math Expression Compiler”. In: *Proceedings of the Python for Scientific Computing Conference (SciPy) 2010* (2010).

[4] *Bitbucket - The Git solution for professional teams*. https://bitbucket.org/. Accessed: 2016-12-28.

[5] Brandon. “How do Convolutional Neural Networks work?” In: *Data Science and Robots Blog* (2016). URL: http://brohrer.github.io/how_convolutional_neural_networks_work.html.

[6] Jason Brownlee. *Object Recognition with Convolutional Neural Networks in the Keras Deep Learning Library*. http://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/. Accessed: 2017-01-12. 2016.

[7] Le Callet and Patrick; Christian Viard-Gaudin; Dominique Barba. “A Convolutional Neural Network Approach for Objective Video Quality Assessment”. In: *IEEE Transactions on Neural Networks* (2006).

[8] *Clarifai - Image & Video Recognition API*. https://www.clarifai.com/. Accessed: 2016-12-27.

[9] *Computer Vision API*. https://www.microsoft.com/cognitive-services/en-us/computer-vision-api. Accessed: 2016-12-27.

[10] Deng, L.; Yu, and D. “Deep Learning: Methods and Applications”. In: *Foundations and Trends in Signal Processing* (2014). DOI: 10.1561/2000000039.

[11] *Google Brain team members set their own research agenda, with the team as a whole maintaining a portfolio of projects across different time horizons and levels of risk.* https://research.google.com/teams/brain/. Accessed: 2016-12-30.

[12] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

[13] Kevin Ho. "41 Up-to-Date Facebook Facts and Stats". In: *Wishpond* (2014). URL: http://blog.wishpond.com/post/115675435109/40-up-to-date-facebook-facts-and-stats.

[14] Daniel P. Hutternlocher and Shimon Ullman. *Object Recognition Using Alignment.* https://www.cse.unr.edu/~bebis/CS773C/ObjectRecognition/Papers/Huttenlocher87.pdf. Accessed: 2017-01-12. 1963.

[15] *ImageNet Summary and Statistics.* http://image-net.org/about-stats. Accessed: 2016-12-28.

[16] *Imagga - powerful image recognition APIs for automated categorization & tagging.* https://imagga.com/. Accessed: 2016-12-27.

[17] Nitin Kaushal and Purnima Kaushal. "Human Identification and Fingerprints: A Review". In: *J Biomet Biostat* (2011). URL: https://www.omicsonline.org/human-identification-and-fingerprints-a-review-2155-6180.1000123.php?aid=2581.

[18] *Keras code and weights files for popular deep learning models.* https://github.com/fchollet/deep-learning-models. Accessed: 2016-12-30.

[19] *LeNet-5, convolutional neural networks.* http://yann.lecun.com/exdb/lenet/. Accessed: 2016-12-29.

[20] Richard Gray for MailOnline. "Google apologizes after Photos app tags black couple as gorillas: Fault in image recognition software mislabeled picture". In: (2015). URL: http://www.dailymail.co.uk/sciencetech/article-3145887/Google-apologises-Photos-app-tags-black-people-gorillas-Fault-image-recognition-software-mislabelled-picture.html.

[21] Matusugu and Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda. "Subject independent facial expression recognition with robust face detection using a convolutional neural network". In: *Neural Networks* (2003). URL: http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/sparse/matsugo_etal_face_expression_conv_nnet.pdf.

[22] *Parallel Programming and Computing Platform - NVIDIA CUDA*. http://www.nvidia.com/object/cuda_home_new.html. Accessed: 2016-12-28.

[23] *Photos - Google Photos*. https://photos.google.com/. Accessed: 2016-12-27.

[24] *Python IDE for Professional Developers*. https://www.jetbrains.com/pycharm/. Accessed: 2016-12-28.

[25] *Python programming language*. https://www.python.org/. Accessed: 2016-12-28.

[26] L. G. Roberts. *Machine Perception of Three-Dimensional Solids*. https://dspace.mit.edu/bitstream/handle/1721.1/11589/33959125-MIT.pdf?sequence=2. Accessed: 2017-01-12. 1963.

[27] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[28] Silberstein et al. "Efficient computation of sum-products on GPUs through software-managed cache." In: *Proceedings of the 22nd annual international conference on Supercomputing – ICS '08. pp. 309–318.* (2008). DOI: 10.1145/1375527.1375572.

[29] *Tensorflow - an open source software library for numerical computation using data flow graphs*. https://www.tensorflow.org/. Accessed: 2016-12-28.

[30] *TIOBE Programming Community Index Python*. http://www.tiobe.com/documentation//tpci/Python.html. Accessed: 2016-12-29. 2015.

[31] *Vision API - Image Content Analysis | Google Cloud Platform*. https://cloud.google.com/vision/. Accessed: 2016-12-27.

[32] *Why was Python created in the first place?* `https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place`. Accessed: 2016-12-29.

7 Приложения

7.1 Приложение 1: Указание за инсталиране

Предварителни изисквания:

- Linux базирана операционна система (софтуерът е тестван под Ubuntu 14.04 и Ubuntu 16.04).
- Python версия 3 или по-нова.
- Видео карта NVIDIA - интерфейсът CUDA разпознава само видео карти на NVIDIA. Ако видео картата не е NVIDIA софтуерът ще използва мощността на процесора и системата ще се забави значително.

Софтуерът използва Tensorflow, който от своя страна достъпва видео картата чрез CUDA. За целта трябва да се инсталират (Подробно описание: https://www.tensorflow.org/versions/r0.11/get_started/os_setup):

- CUDA toolkit 8.0
- CuDNN v5
- Tensorflow

Повечето модули, от които зависят програмите могат да се инсталират чрез pip. В ubuntu по подразбиране са инсталирани 2 версии на python: 2 и 3. За да се инсталира пакет с помощта на pip за python3 се използва pip3. Някои модули изискват повече права от други - трябва да се добави sudo пред pip3 за работа като администратор (su=super user). Python имплементацията зависи от следните модули:

- **os**: Използва се за записване и четене на файлове. Наличен е по подразбиране в python.

- **random**: Използва се за генериране на произволни числа. Наличен е по подразбиране в python.
- **pickle**: Използва се за сериализация (запазване на текущото състояние на обекти във файл). Наличен е по подразбиране в python.
- **time**: Използва се за работа с дати и часове. Наличен е по подразбиране в python.
- **tarfile**: Използва се за работа с tar файлове. В такъв формат се свалят първоначално снимките от ImageNet. Наличен е по подразбиране в python.
- **multiprocessing**: Използва се паралелна работа в повече от една нишка. Ускорява изтеглянето на изображенията и други процеси. Наличен е по подразбиране в python.
- **pprint**: Използва се за принтиране на сложни структури от данни като речници и списъци. Наличен е по подразбиране в python.
- **numpy**: Този модул е в основата на всички научни разработки, написани на python. Поддържа оптимизирани многомерни масиви, обработката им чрез функции от високо ниво и полезни практики от линейната алгебра. Ако не е наличен се инсталира с "pip install numpy".
- **PIL**: Абревиатура на Python Imaging Library. Използва се за обработка на изображения. В контекста на програмата помага за преоразмеряването на всяко изображение в нужния формат. Инсталира се с "pip install pillow".
- **h5py**: Използва се за запазване на вече обучен модел на невронна мрежа. Инсталира се последователно със следните 2 команди:

  sudo apt-get install libhdf5
  sudo pip3 install h5py

- **keras**: Абстракция на работата с невронни мрежи. Използва Tensorflow. Инсталира се с "pip install keras". За повече информация: https://keras.io/#installation.

7.2 Приложение 2: Наръчник на потребителя

Файловата организация е както следва:

- src
  - data
    - bin_generator.py
    - image_crawler.py
    - imagenet_metadata.py
  - main
    - common.py
    - eval.py
    - models.py
    - retrain.py
    - train.py
  - pretrained
    - audio_conv_utils.py
    - imagenet_utils.py
    - inception_v3.py
    - resnet50.py
    - test_imagenet.py
    - vgg16.py
    - vgg19.py
    - xception.py
  - config.py
- definitions.py
- main.py

Единствената изходна точка на програмата е файлът **main.py** в директорията /image-recognition. След успешно стартиране на програмата, тя предоставя 4 опции на потребителя в следния диалог:
1. Image crawler

2. Bin Generator
3. Train
4. Eval

Choose what to do:

При избор на първата опция (**1. Image crawler**), програмата изтегля изображенията за 57-те класа в папка _DATA в основната директория. Процесът може да отнеме време в зависимост от връзката към интернет и ядрата на компютъра. След изтегляне на всички изображения папката заема около 8.5 GB дисково пространство.

При избор на втората опция (**2. Bin Generator**), програмата създава папка в директорията _MODELS с име текущата дата и час. Там добавя файлове с разширение .bin. Всеки един от тях съдържа информация за до 1000 изображения в удобен за четене формат. Броят на изображенията, за които е генерирана информация зависи от конфигурационния файл image-recognition/src/config.py.

При избор на третата опция (**3. Train**), програмата предоставя опция за избор на директория с генерирани .bin файлове. След избор, програмата тренира конволюционна невронна мрежа и запазва модела в същата папка в два файла: _model.h5 и _model.json. Освен това програмата принтира подробна информация за архитектурата на мрежата и точността при всяка итерация.

При избор на четвъртата опция (**4. Eval**), програмата отново предоставя опция за избор на директория с генериран модел. След избор, програмата оценява обучената невронна мрежа върху всеки един от класовете, за които е тренирана и изкарва статистика за точността.