МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

_____ П.О. Приставка

" ___ " _____ 2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

“ МАГІСТР ”

Тема: «*Аналіз та розпізнавання дерев та лісових насаджень за даними аерофотознімків»*

Виконавець: Гнатюк В.А.

Керівник: д.т.н., проф. Приставка П.О.

Нормоконтролер:

Київ 2020

Добавлено примечание ([ВГ1]):

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально Науковий інститут ІДС

Кафедра Прикладної математики

Спеціальність 113 “ Прикладна математика ”

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ П.О. Приставка

" ___ " _____2020 р.

ЗАВДАННЯ

на виконання дипломної роботи

Гнатюка Владислава Анатолійовича

1. Тема дипломної роботи: аналіз та розпізнавання дерев та лісових насаджень за даними аерофотознімків затверджена наказом ректора від «___» _____ 2020р. №_____
2. Термін виконання роботи: з _____р. по _____р.
3. Вихідні дані до роботи: *цифрові зображення реалістичних сцен які отримано з камери безпілотного повітряного судна.*
4. Зміст пояснювальної записки:
   - *аналіз методів для первинної обробки цифрових зображень, сегментація типів місцевості;*
   - *розробка методу та ІТ для класифікації та розпізнавання типів місцевості за даними аерофотозйомки;*
   - *реалізація даної ІТ в програмному забезпеченні.*

5. Календарний план-графік

№ п/п	Завдання	Термін виконання	Відмітка про виконання
1.	*Одержати індивідуальне завдання*		*Виконано*
2.	*Уточнення постановки задачі*		*Виконано*
3.	*Провести аналіз літератури*		*Виконано*
4.	*Провести аналіз методів розпізнавання*		*Виконано*
5.	*Розробити програмне забезпечення*		*Виконано*
6.	*Протестувати програмне забезпечення*		*Виконано*
7.	*Оформити пояснювальну записку*		*Виконано*
8.	*Створити презентації*		*Виконано*
9.	*Здача дипломної роботи*		*Виконано*
10.	*Захист в ДЕК*		*Виконано*

6. Дата видачі завдання: " ___ " _____ 2020 р.

Керівник дипломної роботи П.О. Приставка.

Завдання прийняв до виконання Гнатюк В.А.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи на тему: «*Інформаційна технологія класифікації типів місцевості та пошук об'єктів за даними аерофотозйомки*»: 71 сторінки, 28 рисунків, 32 таблиць, використаних джерел.

ЦИФРОВЕ ВІДЕО, ЦИФРОВЕ ЗОБРАЖЕННЯ, ПЕРВИННА ОБРОБКА ЦИФРОВИХ ЗОБРАЖЕНЬ, ФІЛЬТРАЦІЯ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, МОДЕЛЬ ЦИФРОВОГО ЗОБРАЖЕННЯ, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, АВТОМАТИЗОВАНА СИСТЕМА.

Об'єкт дослідження: процес сегментації та класифікації зображень в системах комп'ютерного зору.

Предмет дослідження: методи обробки цифрових відео та цифрових зображень і алгоритми сегментації та класифікації об'єктів.

Програмне середовище, що використовувалося в процесі виконання дипломної роботи *Python 3.6*.

Мета: розробити метод та інформаційну технологію класифікації типів місцевості, який є стійким до масштабування, часткового зашумлення, неповного представлення та інваріантним до обертань.

Результатом виконання дипломної роботи є автоматизована система сегментації та класифікації типів місцевості яка працює на операційній системі *Windows*.

Галузь застосування: результат дипломної роботи рекомендується використовувати при проведені наукових досліджень, у навчальному процесі, практичній діяльності фахівців у сфері прикладної математики та в військовій галузі для класифікації типів місцевості та пошуку особливих та підозрілих зон на цифровому зображенні або відео що отримано з камери безпілотного повітряного судна.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АС – автоматизована система;

ЛА – літальний апарат;

ІТ – інформаційна технологія;

ПЗ – програмне забезпечення;

ЦЗ – цифрове зображення;

БПС – безпілотне повітряне судно;

ЕОМ – електронна обчислювальна машина.

ЗНМ – згорткова нейронна мережа.

БД – база даних.

ПОСТАНОВКА ЗАДАЧІ

Нехай маємо дані з аерофотознімків з БПЛА. Необхідно розробити інформаційну технологію, яка дозволяла на даних аерофотознмків дерева та лісові насадження, здійснювати підрахунок площ під даними типами місцевості, відповідала вимогам роботи в реальному часом.

Для досягнення даної мети необхідно реалізувати розпізнавання на основі ЗНМ та роз’язувати наступні задачі:

1. На основі існуючі набори даних аерофотознімки підготувати датасет.
2. На основі існуючих архітектур розробити свою.
3. Дослідити різні параметри навчання та вибрати оптимальні.
4. Розробити автоматичну систему та провести тестування та дослідити якість розробленої моделі.
5. Оформити результати у вигляді дипломної записки дотримуючись ДСТУ.

Галузь застосування: результат дипломної роботи рекомендується використовувати при проведені наукових досліджень, у навчальному процесі, практичній діяльності фахівців у сфері прикладної математики та в військовій галузі для відокремлення лісу від особливих та підозрілих ідентифікованих зон на цифровому зображенні або відео що отримано з камери безпілотного повітряного судна.

ВСТУП

Здатність людини взаємодіяти з навколишнім світом за допомогою зору дозволяє сприймати кольори, світло та зовнішню структуру оточуючого середовища у вигляді зображення, базується на здатності до розпізнавання. Ця вроджена здатність легко визначати та розпізнавати об'єкти, навіть якщо вони спотворені, спонукала до виникнення досліджень про те, як людський мозок обробляє ці зображення.

Вміння розпізнавати об’єкти є досить надійним, не зважаючи на зміни, пов'язані з умовами спостереження, емоційними виразами, старінням або навіть обставинами, які дозволяють бачити тільки частину об’єктів. Крім того, ми можемо пам’ятати мільйони різноманітних об’єктів протягом свого життя.

Розуміння людського механізму класифікації, на додаток до когнітивних аспектів, сприяє побудові систем автоматичної ідентифікації об’єкту машиною. Тим не менш, розпізнавання об’єктів є областю активного дослідження, оскільки ще не було запропоновано абсолютно успішного підходу або моделі для вирішення задачі ідентифікації особи за обличчям. Розпізнавання візуальних образів є одним з найважливіших компонентів систем управління та обробки інформації, АС і систем прийняття рішень[1, 2]. Задачі, пов’язані з класифікацією та ідентифікацією предметів, явищ та сигналів, які характеризуються скінченним набором деяких властивостей та ознак, виникають в таких областях як робототехніка, інформаційний пошук, моніторинг, трекінг об’єкту на відео, фотограмметрії та аналіз візуальних даних, дослідження штучного інтелекту. Алгоритмічна обробка і класифікація зображень застосовуються в системах безпеки, контролю та управління доступом, в системах відеоспостереження, системах віртуальної реальності та інформаційних пошукових системах у військовому та цивільному напрямі. В даний момент у виробництві широко використовуються системи розпізнавання рукописного тексту, автомобільних номерів, відбитків пальців або людських облич, що знаходять застосування в інтерфейсах програмних продуктів, системах безпеки та ідентифікації особистості, а також в інших прикладних цілях[3].

Кількість інформації, що підлягає обробці, постійно зростає, тому існує потреба в розробці методів, які дозволяють проводити розпізнавання об’єктів в режимі реального часу.

Метою дипломної роботи є розробка методу та інформаційної технології класифікації типів місцевості, який є стійким до масштабування, часткового зашумлення, неповного представлення та інваріантним до обертань.

Об’єктом дослідження дипломної роботи є процес сегментації та класифікації зображень в системах комп’ютерного зору.

Предметом дослідження є методи обробки цифрових відео та цифрових зображень і алгоритми сегментації та класифікації об’єктів.

Методи дослідження. Для рішення поставлених задач використовувались методи комп’ютерного зору, теорії оптимізації, математичної статистики, теорії розпізнавання, теорії апроксимації на основі *B*-сплайнів.

Наукова новизна дипломної роботи полягає в наступному: Вперше було створено інформаційну технологію для розпізнавання дерев та лісових насаджень де використовуються ЗНМ.

ВВЕДЕННЯ В ПРОБЛЕМУ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ. МЕТОДИ РОЗПІЗНАВАННЯ

Матеріали даного розділу розглядають проблему розпізнавання об’єктів дана теоретична частина використовувалась для побудови власної системи розпізнавання за допомогою якої було вирішено задачу класифікації типів місцевості.

В першому підрозділі розглядаються проблеми розпізнавання, в якому описано завдання класифікації на мові ознак. Також описані цілі розпізнавання.

В другому підрозділі розглядаються загальні характеристики задачі розпізнавання. Типи задач розпізнавання. Види інформації в задачах розпізнавання і прогнозування.

1.1. Проблеми розпізнавання. Цілі розпізнавання.

З огляду на суто історичних причин (термін «pattern recognition», перекладений як «розпізнавання образів», був запозичений, як і багато інших термінів інформатики, з англомовних робіт) цей клас завдань виявився пов'язаний з поняттям «образа» («pattern»). На жаль, свого часу не звернули уваги на його багатозначність - термін «pattern», крім значення «образ», має ще і значення «модель», «стиль», «режим», «закономірність», «образ дій». В
сучасному розпізнаванні і особливо штучному інтелекті його вживають в найширшому сенсі, маючи на увазі деяке структуроване наближене (часткове) опис (ескіз) досліджуваного об'єкта або явища, причому часткова визначеність
опису є важливою властивістю образу. Образ допускає рекурсивне визначення: символ є образом, список символів є образом, образами є тільки ті вираження, які побудовані відповідно до двох зазначених умов. Обліковий запис дозволяє використовувати одне і те ж уявлення для опису способу довільного типу незалежно від його «змісту». Додатковою перевагою запису цього типу служить можливість користуватися одними і тими ж

алгоритмами для роботи з образами з різними депонтами. Природно також допускати, що образ складається з двох груп символів, що представляють відповідно змінні і постійні характеристики об'єкта опису.

Основне призначенців описів (образів) - це їх використання в процесі встановлення відповідності об'єктів, тобто при доказі їх ідентичності, аналогічності, подібності, і тому подібному, здійснюваному шляхом порівняння (зіставлення). Два образи вважаються подібними, якщо вдається встановити їх відповідність. Можна, зокрема, вважати, що має місце відповідність двох образів, якщо можна досягти їх ідентичності, підставляючи замість змінних будь-які вирази.

Зіставлення образів являє собою основну задачу розпізнавання і грає істотну роль в інформатиці в цілому. Це завдання виникає, зокрема, в різних розділах штучного інтелекту, наприклад в розумінні природного мови, символьної обробки виразів алгебри, експертних системах, перетворенні і синтезі програм ЕОМ. Процедура зіставлення виявилася настільки суттєвою для штучного інтелекту, що в багато мов програмування, які використовуються в штучний інтелект, вона входить в якості примітиву.

Відзначимо, що в різних завданнях поняттю способу надається різний зміст. Так, скажімо, в розпізнаванні (у класичних моделях) образ зазвичай описується вектором ознак, кожен елемент якого являє числове значення одною з ознак, що характеризують відповідний об'єкт. У структурній моделі розпізнавання як образу виступає деякий вислів, що породжується тієї граматикою, яка характеризує клас, якому даний образ належить. У завданнях обробки тексту роль образу виконує деяка ланцюжок - в результаті процедура встановлення відповідностей зводиться до пошуку входжень цього ланцюжка (образа) в тексті.

Термін «розпізнавання» в рівній мірі відноситься як до процесів сприйняття і пізнання, властивим людині і живим організмам в цілому, так і до спроб реалізувати і використовувати «механічні» аналоги (по функції і результату) цих процесів, дослідження і синтез яких складають предмет розпізнавання як розділу інформатики. У цьому розділі будуть розглядатися лише останні. Отже, метою створення автоматизованих систем розпізнавання

є автоматизація групи процесів сприйняття і пізнання, пов'язаних з пошуком, виділенням, ідентифікацією, класифікацією та описом образів на основі аналізу реальних даних, отриманих тим чи іншим способом. Зазвичай пошук і
виділення образів здійснюються на початковому етапі аналізу в процесі обробки вихідних даних і виконуються для того, щоб отримати деякі проміжні результати (тобто перетворити вихідні дані в деяку іншу форму), «краще» що представляють образи з точки зору вирішення відповідної завдання. Наступний етап - розробка «класифікатора» - зазвичай включає аналіз вибіркових (перетворених) даних, синтез моделі, що враховує мінливість образів, що належать певного класу, вибору із заданого набору характеристик
деякого їх підмножини, адекватно характеризує окремі класи об'єктів, визначення методів виділення зазначеного підмножини і розробку власне алгоритму розпізнавання (класифікації).

Центральна задача розпізнавання - побудова на основі
систематичних теоретичних і експериментальних досліджень ефективних обчислювальних засобів для віднесення формалізованих описів ситуацій і об'єктів до відповідних класів. В основі такого віднесення (розпізнавання, класифікації) лежить отримання деякої агрегованої оцінки ситуації, виходячи з її опису. За умови встановлення відповідності між класами еквівалентності, які ви зробили на безлічі рішень і безлічі об'єктів розпізнавання (ситуацій),
автоматизація процедур розпізнавання стає елементом автоматизації процесів прийняття рішень. завдання розпізнавання являють собою, по суті, дискретні аналоги задач пошуку оптимальних рішень. До них відноситься широкий клас
задач, в яких за деякою, зазвичай вельми різнорідної, бути може, неповної, нечіткої, перекрученої і непрямої інформації потрібно встановити, чи мають досліджувані (вельми складні, в певному сенсі «комплексні») ситуації (об'єкти, явища) фіксованим кінцевим набором властивостей, що дозволяє віднести їх до певного класу, - завдання розпізнавання і класифікації, або по аналогічного роду інформації про кінцевий безлічі досить однотипних процесів слід з'ясувати, в якій області з кінцевого числа

областей перебуватимуть ці процеси через певний період часу – завдання прогнозування. До завдань цього виду зводяться завдання технічної
і медичної діагностики, геологічного прогнозування (зокрема, відновлення геофізичних полів), прогнозування властивостей хімічних сполук, сплавів та нових матеріалів, розпізнавання і характеризації властивостей динамічних і статичних об'єктів в складній фонової обстановці і при наявності активних і
пасивних перешкод по зображеннях, що отримуються за допомогою різноманітних технічних засобів, прогнозування ходу будівництва великих об'єктів, обробки даних дистанційного дослідження природних ресурсів, прогнозування врожаю, виявлення лісових пожеж, управління виробничими
процесами (прогнозування можливостей входу значень параметрів швидкоплинних процесів в критичні області) і інше.

1.2. Загальна характеристика задачі розпізнавання. Типи задач розпізнавання. Види інформації в задачах розпізнавання і прогнозування.

Практичні завдання, для вирішення яких доцільно застосовувати методи розпізнавання, відрізняються рядом специфічних особливостей.

Інформаційні завдання, вирішення яких здійснюється за допомогою застосування до доступним вихідними даними системи перетворень, що включає в загальному випадку два основних етапи: а) приведення вихідних даних до деякого стандартного виду, зручному для розпізнавання - синтез формалізованого опису. ситуації (об'єкта) на основі наявної різнорідної інформації (емпіричних даних, результатів вимірювань, знань про логічних аспектах досліджуваних явищ (процесів), відомостей про конструкції, призначення і експлуатаційних характеристиках (Можливо, передбачуваних) об'єкта, експертних даних, наявної апріорної семантичної і синтаксичної інформації); б) власне розпізнавання - перетворення формалізованого опису в стандартизовану матрицю відповідей, відповідну вибору в якості відповіді (класифікаційного рішення) однієї з деякого кінцевого фіксованого набору можливостей (вказівка приналежності ситуації (об'єкта) певного класу).

У завданнях є можливість вводити поняття деякого подібності між об'єктами (ситуаціями), точніше, між їх описами - формулювати узагальнене поняття близькості в якості підстави для зарахування ситуацій (об'єктів) в один і той ж клас або різні класи.

Є можливість оперувати певним напором прецедентів - прикладів, класифікація яких (в сенсі розв'язуваної задачі) відома і які (у вигляді стандартних формалізованих описів) можуть бути пред'явлені алгоритмом розпізнавання для налаштування на завдання в процесі навчання.

Завдання, для яких важко будувати формальні теорії і застосовувати класичні математичні методи, оскільки в ситуаціях, в яких вони виникають, має місце один з двох таких випадків: а) рівень формалізації відповідної предметної області та / або доступна інформація такі, що ні можуть скласти основу для синтезу математичної моделі, відповідає класичним математичним або математично-фізичним канонам і допускає вивчення класичними аналітичними або чисельними методами; б) математична модель в принципі може бути побудована, проте її синтез або вивчення пов'язані з такими витратами (збір необхідної інформації, обчислювальні ресурси, час), що вони істотно перевищують виграш, принесений шуканим рішенням, або виходять за межі існуючих технічних можливостей, або роблять рішення завдання просто безглуздим.

У завданнях «за визначенням» існує «погана» вихідна інформація, що характеризує складну в семантичному і структурному відношенні ситуацію (об'єкт в деякому середовищі) – це обмежена, неповна (з пропусками), різнорідна, непряма (характеристики зовнішніх проявів функціонування процесу, причому не завжди пов'язані з принциповим особливостям лежачого в його основі механізму), нечітка, неоднозначна, імовірнісна. В цілому це завдання, в яких відомо дуже мало, щоб можна було користуватися класичними методами рішення (моделями), але все-таки відомо досить, щоб рішення було можливо.

Типи задач розпізнавання:

1. Віднесення пред'явленого об'єкта (ситуації) по його формалізованого опису до одного з заданих класів - завдання розпізнавання (навчання з учителем).

2. Розбиття множини ситуацій (об'єктів) з їх формалізованим описами на систему непересічних підмножин (класів) -задача автоматичної класифікації (таксономія, кластер-аналіз, навчання без вчителя).

3. Визначення інформативного набору ознак для побудови формалізованого опису об'єкта розпізнавання; оцінка інформативності окремих ознак і їх поєднань – завдання вибору інформативного набору ознак при розпізнаванні.

4. Побудова формалізованого опису об'єкта розпізнавання -задача приведення вихідних даних до виду, зручного для розпізнавання.

5. *Завдання 1* з урахуванням динамічності об'єкта (ситуації).

6. *Завдання 2* з урахуванням динамічності об'єктів (ситуацій).

7. *Завдання 5, 6*, в яких рішення має ставитися до деякого моменту часу в майбутньому - задача прогнозування.

Всі перераховані вище завдання можуть вирішуватися при завданні вихідних даних в одній з наступних форм або в їх поєднаннях:

- зображення, отримані в різних частинах спектра випромінювань (оптичні, інфрачервоні, ультразвукові та т. д.) різними способами (телевізійні, фотографічні, лазерні, радіолокаційні, радіаційні і т. д.) і перетворені в цифрову форму;
- сигнали (довгі числові послідовності);
- експертні дані, числові та інші види символьного інформації загального вигляду;
- серії зображень («фільми») будь-якого виду;

1.3. Висновки

В матеріалах 1 розділу було проаналізовано основні проблеми розпізнавання. Поставлено цілі розпізнавання та значущість розпізнавання в різних прикладних задачах. Описано постановку задачі системи розпізнавання та способи побудови системи розпізнавання. Описана загальна характеристика задачі розпізнавання. Типи задач розпізнавання. Види інформації в задачах розпізнавання і прогнозування.

РОЗПІЗНАВАННЯ ОБРАЗІВ ЗА ДОПОМОГОЮ ЗГОРТКОВИХ НЕЙРОНИХ МЕРЕЖ

Матеріали даного розділу розглядають такий метод розпізнавання об’єктів як згорткова нейронна мережа, робиться огляд бібліотек *keras* та *tensorflow*, а також описується процес аугментації даних.

В першому підрозділі розглядаються згорткові нейронні мережі, як метод розпізнавання образів, їх основні частини та біологічні процеси які стали основою для їх створення.

В другому підрозділі розглядаються бібліотеки *keras* та *tensorflow*, та їх можливості.

В третьому підрозділі розглядаються сучасні архітектури ЗНМ їх переваги та недоліки.

В четвертому підрозділі розглядається процедура аугментації, як засіб штучного збільшення вибірки.

2.1. Згорткові нейроні мережі

Згорткова мережа (LeCun, 1989), вона ж згорткова нейронна мережа (ЗНМ), - це спеціальний вид нейронної мережі для обробки даних з сітковою топологією. Призкладами можуть служити часові ряди, які можна розглядати як одновимірну сітку прикладів, які обирають через регулярні проміжки часу, а також зображення, що розглядаються як двовимірна сітка пікселів. Згорткові мережі добилися колосального успіху в практичних додатках. Своєю назвою вони обов'язані використанню математичної операції згортки. Згортка - це особливий вид лінійної операції. Згорткові мережі - це просто нейронні мережі, в яких замість спільної операції множення на матрицю, по крайній мірі в одному шарі, використовується згортка.

У найзагальнішому вигляді згортка - це операція над двома функціями аргументу. Щоб обґрунтувати визначення згортки, почнемо з прикладів можливих функцій. Припустимо, що ми стежимо за положенням космічного корабля за допомогою лазер ного датчика. Наш датчик видає єдине значення, x(t) положення

корабля в момент t. Змінні x і t приймають дійсні значення, тобто свідчення датчика в будь-які два моменти часу можуть відрізнятися.

Тепер припустимо, що датчик схильний перешкод. Щоб отримати менш зашумлену оцінку положення корабля, необхідно усереднити кілька результатів вимірювань. Зрозуміло, недавні вимірювання важливіші, тому ми хочемо обчислювати зважене середнє, надаючи недавнім вимірам більший вага. Для цього можна скористатися ваговою функцією ω(a), де a - давність виміру. Застосувавши таку операцію усереднення в кожен момент часу, ми отримаємо нову функцію, яка дає згладжену оцінку положення космічного корабля:

$$s(t) = \int x(a)\omega(t-a)da. \qquad (2.1)$$

Ця операція називається згорткою і зазвичай позначається зірочкою:

$$s(t) = (x * \omega)(t) \qquad (2.2)$$

У нашому прикладі w повинна бути функцією щільності ймовірності, інакше усереднення не вийде. Крім того, ω повинна бути дорівнює 0 для всіх негативних значень аргументу, інакше вона буде здатна заглядати в майбутнє, що навряд чи в межах наших можливостей. Але ці обмеження характерні тільки для даного прикладу. У загальному випадку згортку можна визначити для будь-яких функцій, для яких визначено показаний вище інтеграл (2.1), і використовувати не тільки для отримання зваженого середнього.

У термінології згорткових мереж перший аргумент (в нашому прикладі функція x) називається входом, а другий (функція ω) - ядром. Вихід іноді називають картою ознак.

Лазерних датчиків, здатних видавати результати вимірювань в будь-який момент часу, в дійсності не буває. Зазвичай при роботі з даними в комп'ютері час дискретизований, і наш датчик буде видавати дані через регулярні інтервали. Мабуть, було б реалістичніше припустити, що лазер виробляє виміру раз в секунду. Індекс моментів часу t може приймати тільки цілі значення. Якщо тепер припустити, що x і ω визначені тільки для цілих t, то ми отримаємо визначення дискретної згортки:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\omega(t-a) \qquad (2.3)$$

У додатках машинного навчання входом зазвичай є багатовимірний масив даних, а ядром - багатовимірний масив параметрів, адаптованих алгоритмом навчання. Будемо називати ці масиви тензорами. Оскільки кожен елемент входу і ядра повинен зберігатися окремо в явному вигляді, зазвичай передбачається, що ці функції дорівнюють нулю всюди, крім кінцевого безлічі точок, для яких ми зберігаємо значення. На практиці це означає, що суму від мінус до плюс нескінченості можна замінити підсумовуванням за кінцевим числом елементів масиву.

Нарешті, ми часто використовуємо згортки відразу по декількох осях. Наприклад, якщо входом є двовимірне зображення I, то і ядро має бути двовимірним:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i-m, j-n). \qquad (2.4)$$

Операція згортки комутативна, тому формулу можна записати і так:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i-m, j-n)K(m, n). \qquad (2.5)$$

Зазвичай другу формулу простіше реалізувати в бібліотеці машинного навчання, оскільки діапазон допустимих значень m і n менше.

Властивість комутативності згортки має місце, тому що ми відобразили ядро щодо входу, т. е. при збільшенні m індекс входу збільшується, а індекс ядра зменшується. Єдина причина такого відображення - забезпечити комутативність. І хоча комутативність корисна для доведення теорем, в реалізації нейронних мереж вона зазвичай ролі не грає. Натомість у багатьох бібліотеках реалізована споріднена функція - перехресна кореляція - та ж згортка, тільки без відображення ядра:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i+m, j+n)K(m, n). \qquad (2.6)$$

У багатьох бібліотеках машинного навчання реалізована саме перехресна кореляція, але називається вона згорткою. Далі ми будемо дотримуватися цього угоди і називати згорткою обидві операції, а в тих місцях, де відображення ядра має

значення, будемо це явно обумовлювати. В контексті машинного навчання алгоритм навчання ставить знайдені значення ядра в правильну позицію, тому якщо алгоритм заснований на згортці з відображенням ядра, то він навчить ядро, відбите щодо того, яке навчено алгоритмом зі згорткою без відображення. Рідко буває так, щоб згортка використовувалася в машинному навчанні сама по собі; частіше вона комбінується з іншими функціями, і такі комбінація не комутативні поза залежно від того, використовується в ядрі-згортка відображення чи ні.

Дискретну згортку можна розглядати як множення на матрицю, на елементи якої накладені деякі обмеження. Наприклад, в разі одновимірної дис-
конкретної згортки кожен рядок матриці повинний дорівнювати попередньому, зсунутий на один елемент. Це твердження називається теоремою Теплиця. У двовимірному випадку згортка відповідає двічі блочно-циркуляній матриці. Крім обмежений на рівність деяких елементів, згортку зазвичай відповідає сильно розріджена матриця (в якій більшість елементів дорівнює нулю). пов'язано це з тим, що ядро, як правило, набагато менше вхідного зображення. будь алгоритм нейронної мережі, що базується на збільшенні матриць і не залежить від особливостей структури цих матриць, повинен працювати і з згорткою без будь-яких модифікацій самої мережі. У типових згорткових нейронних мережах все ж використовуються особливості структури, щоб організувати ефективну обробку великих входів, але з теоретичної точки зору це не є обов'язковим.

З згорткою пов'язані три важливі ідеї, які допомагають поліпшити систему машинного навчання: розріджені взаємодії, поділ параметрів і еквіваріантні представлення. Крім того, згортка надає засоби для роботи з
входами змінного розміру. Наведемо ці ідеї по черзі.

У шарах традиційної нейронної мережі застосовується множення на матрицю параметрів, в якій взаємодія між кожним вхідним і кожним вихідним
блоками описується окремим параметром. Це означає, що кожен вихідний
блок взаємодіє с кожним вхідним блоком. Навпаки, в згорткових мережах вза-ємодії зазвичай розріджені (це властивість називають ще розрідженій звяз-

ністю, або розрідженими вагами). Досягається це за рахунок того, що ядро менше входу. Наприклад, вхідне зображення може містити тисячі або мільйони пікселів, але невеликі значимі ознаки, наприклад кордону, можна виявити за допомогою ядра, що охоплює всього десятки або сотні пікселів. Отже, потрібно зберігати менше параметрів, а це знижує вимоги моделі до обсягу пам'яті і підвищує її статистичну ефективність. Крім того, для обчислення виходу потрібно менше операцій. Всі разом зазвичай набагато підвищує ефективність мережі. Якщо є *m* входів і *n* виходів, то для множення матриць необхідно m × n параметрів, і складність практично використовуваних алгоритмів становить *O(m × n)* (в розрахунку на один приклад). Якщо обмежити число з'єднань з кожним виходом величиною *k*, то буде потрібно тільки *k × n* параметрів, і складність складе *O (k × n)*. У багатьох практичних додатках можна отримати хорошу якість на задачі машинного навчання, коли *k* на кілька порядків менше m. Графічно розріджена зв'язність ілюструється на рис. 2.1 і 2.2.

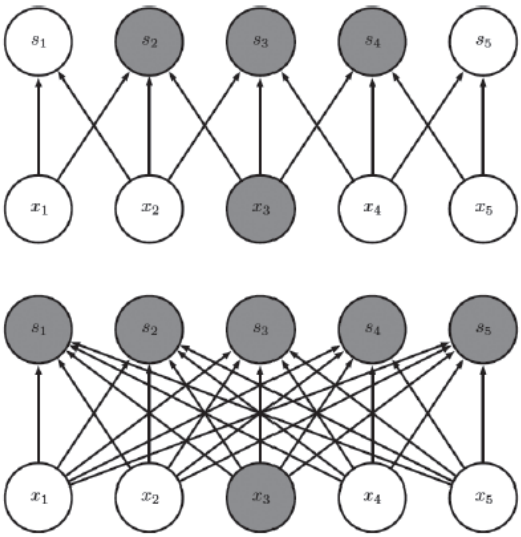


Рис. 2.1. Розріджена зв'язність при погляді знизу.

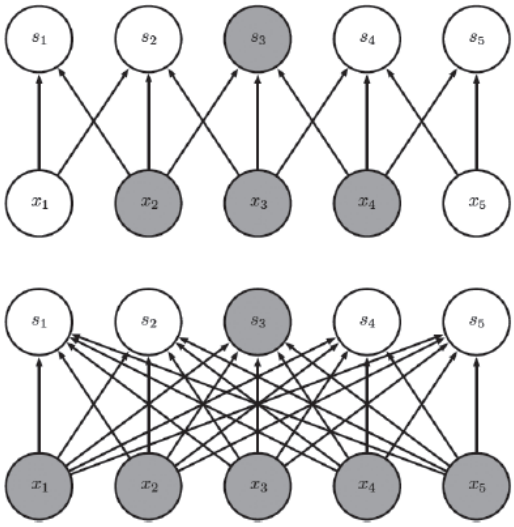


Рис. 2.2. Розріджена зв'язність при погляді зверху.

У глибокій згорковій мережі блоки нижніх рівнів можуть побічно взаємодіяти з більшою частиною мережі, як показано на рис 2.3.

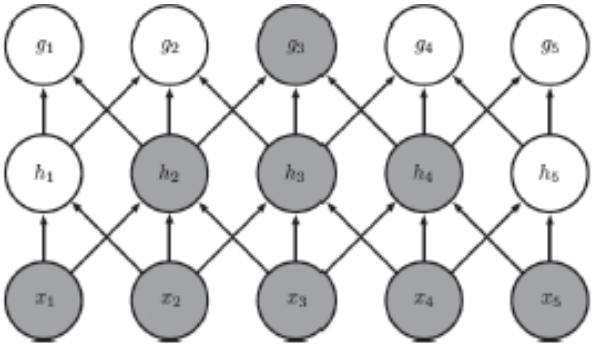


Рис. 2.3. Рецептивне поле блоків в глибоких шарах згорткової мережі

більше рецептивного поля в шарах, близьких до поверхності.

Під поділом параметрів розуміють, що один і той же параметр використовується в декількох функціях моделі. У традиційній нейронної мережі кожен елемент матриці ваг використовується рівно один раз при обчисленні виходу шару. він примножує на один елемент входу, і більше ми до нього ніколи не повертаємося. Замість вживання

терміну «поділ параметрів» можна сказати, що в мережі присутні звязувальні ваги, оскільки значення ваги, застосованого до одного входу, пов'язаний із значенням ваги, застосованого десь ще.

У згоркової нейронної мережі кожен елемент ядра застосовується до кожної позиції входу (за винятком, бути може, деяких граничних пікселів - у залежності від того, як вирішено обробляти кордон). Поділ параметрів означає, що замість навчання окремого набору параметрів для кожної точки ми повинні навчити тільки один набір. Це не впливає на час прямого поширення - воно як і раніше має порядок O (k × n), - але додатково зменшує вимоги до обсягу пам'яті: досить зберігати k параметрів. Нагадаємо, що k зазвичай на кілька порядків менше m. Оскільки величини m і n приблизно рівні, то k практично несуттєво в порівнянні з m × n. Таким чином, згортка багаторазово ефективніше множення матриць з точки зору вимог до пам'яті та статистичної ефективності.

У випадку згортки спеціальний вид поділу параметрів наділяє шар властивістю, яке називається еквіваріантнісь щодо паралельного переносу. Кажуть, що функція еквіваріантна, якщо при зміні входу вихід змінюється точно так же. Точніше, функція f (x) еквіваріантна щодо функції g, якщо
$f(g(x)) = g(f(x))$. У разі згортки, якщо g - паралельний перенос, або зсув входу, то функція згортки еквіваріантна щодо g. Нехай, наприклад, функція I визначає яскравість в точках з цілими координатами, і нехай g - функція, відображає одну функцію зображення в іншу функцію зображення, так що $I' = g(I)$ - функція зображення, для якої $I'(x, y) = I(x - 1, y)$. Це зрушення кожного пікселя I на одну позицію вправо. Якщо застосувати це перетворення до I, а потім виконати згортку, то результат буде таким же, як якщо б ми спочатку застосували згортку до I', а потім перетворення g до результату. При обробці часових рядів це означає, що згортка породжує свого роду тимчасову шкалу, на якій показаний час появи різних ознак у вхідних даних.

Якщо перенести подія на більш пізній момент часу у вхідних даних, то на виході з'явиться точно таке ж його уявлення, тільки пізніше. А при роботі з зображеннями згортка створює двовимірну карту появи певних

ознак у вхідному зображенні. Якщо перемістити об'єкт у вхідному зображенні, то його подання на виході переміститься на таку ж величину. Це буває
потрібно, коли ми знаємо, що деяка функція від невеликого числа пікселів по
корисними при застосуванні до декількох дільницях входу. Наприклад, в разі обробки ображень корисно виявляти кордону в першому шарі згорткової мережі. Одні і ті ж межі зустрічаються більш-менш всюди в зображенні, тому має сенс розділяти параметри по всьому зображенню. Але в деяких випадках такий глобальний поділ параметрів небажаний. Наприклад, якщо ми обробляємо зображення, які були кадровані, так щоб в центрі виявилося обличчя людини, то напевно, хочемо виділяти різні ознаки в різних точках - частина мережі буде обробляє верхню частину обличчя в пошуках брів, а інша частина - шукати підборіддя в нижній частині обличчя.

Згортка не еквіваріантна щодо деяких інших перетворень, наприклад масштабування або повороту. Для обробки таких перетворень необхідні інші механізми.

Типовий шар згорткової мережі складається з трьох стадій. На першій стадії шар паралельно виконує кілька згорток і породжує безліч лінійних
активацій. На другій стадії кожна лінійна активація пропускається через нелінійну функцію активації, наприклад функцію лінійної ректифікації. цю стадію часто називають детекторной. На третій стадії використовується функція **пулінг** для подальшої модифікації виходу шару.

Функція пулінг замінює вихід мережі в певній точці зведеної статистикою прилеглих виходів. Наприклад, операція **max- пулінг** (Zhou and Chellappa, 1988) повертає максимальний вихід в прямокутної околиці. З інших функцій пулінгу відзначимо усереднення по прямокутної околиці, L2-норму в прямокутної околиці і зважене середнє з вагами, залежними від відстані до центрального пікселя.

У будь-якому випадку пулінг дозволяє зробити уявлення приблизно інваріантним щодо малих паралельних переносів входу. Інваріантність відносно паралельного перенесення означає, що якщо зрушити вхід на невелику величину, то значення більшості підданих пулінгу виходів не зміняться.

Локальна інваріантність відносно паралельного перенесення корисна, якщо нас більше цікавить сам факт існування певної ознаки, а не його точне місцезнаходження. наприклад, коли ми хочемо визначити, чи присутній в зображенні обличчя, нам не важливо положенння очей з точністю до пікселя, потрібно тільки знати, чи є очей зліва і очей праворуч. В інших ситуаціях важливіше зберегти місце розташування ознаки. Наприклад, якщо ми шукаємо кутову точку, утворену перетином двох кордонів, орієнтованих певним чином, то необхідно зберегти положення кордонів настільки точно, щоб можна було перевірити, чи перетинаються вони.

Пулінг можна розглядати як додавання нескінченно сильного апріорного припущення, що навчається шаром функція повинна бути інваріантна до малих паралельним перенесенням. Якщо це припущення правильно, то воно може істотно поліпшити статистичну ефективність мережі.

Пулінг по просторовим областям породжує інваріантність до паралельних переносів, але якщо він проводиться по виходах згорток з різними параметрами, то ознаки можуть навчитися, до яких перетворень стати інваріантними.

У більшості завдань пулінг необхідний для обробки входів змінного розміра. Наприклад, якщо ми хочемо класифікувати зображення різного розміру, то код шару класифікації повинен мати фіксований розмір. Зазвичай це достігається за рахунок варіювання величини кроку між областями пулінгу, так, щоб шар класифікації завжди отримував однаковий обсяг зведеної статистики незалежно від розміру входу. Так, можна визначити фінальний шар пулінгу в мережі, так щоб він виводив чотири зведених статистичних показника, по одному на кожен квадрант зображення, незалежно від розміру самого зображення.

У сучасних додатках згорткових мереж часто беруть участь мережі, що містять більше мільйона блоків. Для роботи з ними необхідні ефективні реалізації, задіють кошти розпаралелювання обчислень. Але у багатьох випадках роботу можна прискорити, вибравши відповідний алгоритм згортки.

Згортка еквівалентна перекладу входу і ядра в частотну область за допомогою перетворення Фур'є, по-точковому перемножування двох сигналів і поверненню у

часову область за допомогою зворотного перетворення Фур'є. При певному розмірі завдання це може виявитися швидше наївною реалізації дискретної згортки в лоб.

Якщо d-мірне ядро можна представити у вигляді зовнішнього твори d векторів, по одному на кожне вимір, то ядро називається сеперабельним. для сепарабельних ядер наївна згортка неефективна. Вона еквівалентна композиції *d*-вимірних згорток з кожним з цих векторів. Це набагато швидше обчислення *d*-вимірної згортки з їх зовнішнім твором. Крім того, для подання ядра у вигляді векторів потрібно менше параметрів. Якщо ядро складається з *w* елементів в кожному напрямку, то для наївною багатовимірної згортки потрібен час *O (wd)* і стільки ж місця в пам'яті для зберігання параметрів, тоді як для сеперабельной згортки потрібен час і пам'ять порядку *O (w × d)*. Зрозуміло, не всяку згортку можна передставити подібним чином.

Пошук більш швидких способів обчислити згортку точно або наближено, що не приносячи в жертву вірності моделі, - область активних досліджень. Навіть методи, поліпшують ефективність одного лише прямого поширення, вже корисні, тому що в комерційних системах зазвичай більше ресурсів витрачається не на навчання, а на розгортання мережі.

Згорткові мережі - мабуть, найяскравіший приклад успішного застосування біо-технологічного штучного інтелекту. Хоча на них вплинули і багато інші наукові дисципліни, деякі ключові принципи були почерпнуті з
нейро-біології.

Історія згортальних мереж починається з нейро-біологічних експериментів, поставлених задовго до створення відповідних комп'ютерних моделей. Нейро-фізіологи *Давид Хубель* і *Торстен Візель* протягом декількох років спільно встановили більшість основних фактів, що стосуються роботи зорової системи ссавців (*Hubel and Wiesel, 1959, 1962, 1968*). За свої досягнення вони були відзначені Нобелівською премією. Їх відкриття, що зробили величезний вплив на сучасні моделі глибокого навчання, були засновані на реєстрації активності окремих нейронів в мозку кішок. Вони спостерігали, як нейрони реагують на зображення, що проектуються точно на певні ділянки екрана, розташованої ного перед кішкою. Найважливіше відкриття полягало в тому, що нейрони первинних зорових центрів сильніше реагують на дуже

специфічні зорові паттерни, наприклад точно орієнтовані смужки, і набагато слабкіше - на всі інші патерни. Їх робота допомогла охарактеризувати багато аспектів функціонування мозку, виходять за рамки цієї книги. З точки зору глибокого навчання, нас цікавить в основному спрощена, схематична картина.

І в цій спрощеної картині ми зосередимося в області мозку яка називається зоною *V1*, або первинної зорової корою. Це перша область мозку, яка починає значиму обробку зорової інформації. Не вдаючись в деталі,
скажімо, що зображення формується завдяки попаданню в око світла, який
стимулює сітківку, світлочутливий орган, що становить внутрішню оболонку ока. Нейрони сітківки виконують просту попередню обробку зображення, але не дуже сильно змінюють спосіб його уявлення. Потім зображення по
зоровому нерву поступає в область мозку, яка називається латеральним ко
лінчастим тілом. Головне завдання цих анатомічних утворень що нас цікавить - передати сигнал з ока в зону *V1*, розташовану на потилиці.

Шар згорткової мережі вловлює три властивості зони V1:

- зона V1 організована у вигляді просторової карти. Вона має двовимірну структуру, що повторює структуру зображення на сітківці. Так, світло, що падає на верхню половину сітківки, впливає тільки на відповідну половину зони V1. Згорткова мережа вловлює цю властивість, оскільки її ознаки визначені в термінах двовимірних карт.
- зона V1 складається з великого числа простих клітин. Активність клітини можна до певної міри охарактеризувати лінійною функцією зображення в малому просторово локалізованому рецептивної поле. Детекторні блоки згорткової мережі покликані імітувати саме ці властивості простих клітин;
- в зоні V1 є також багато складних клітин. Вони реагують на ознаки, схожі на знайдені простими клітинами, але інваріантні щодо невеликих змін в положенні ознак. Звідси беруть початок пулінгові блоки згорткових мереж. Складні клітини інваріантні також відносно деяких змін освітлення, які неможливо вловити з допомогою простого агрегування по просторовим областям. Ці види

інваріантності стали причиною деяких стратегій міжканального пулінгу в згорткових мережах, наприклад *maxout-блоків* (Goodfellow et al., 2013a).

2.2. Огляд біблотек keras та tensorflow

Tensorflow (далі - TF) - досить молодий фреймворк для глибокого машинного навчання, що розробляється в Google Brain. Довгий час фреймворк розроблявся в закритому режимі під назвою DistBelief, але після глобального рефакторінга 9 листопада 2015 року було випущено в open source. За рік з невеликим TF доріс до версії 1.0, знайшов інтеграцію з keras, став значно швидше і отримав підтримку мобільних платформ. Останнім часом фреймворк розвивається ще й в сторону класичних методів, і в деяких частинах інтерфейсу вже чимось нагадує scikit-learn. До поточної версії інтерфейс змінювався активно і часто, але розробники пообіцяли заморозити зміни в API. Ми будемо розглядати тільки Python API, хоча це не єдиний варіант - також існують інтерфейси для C ++ і мобільних платформ.

Робота с TF будується навколо побудови і виконання графа обчислень. Граф обчислень - це конструкція, яка описує те, яким чином будуть проводитися обчислення. У класичному імперативний програмуванні ми пишемо код, який виконується за рядком. В TF звичний імперативний підхід до програмування потрібен тільки для якихось допоміжних цілей. Основа TF - це створення структури, яка задає порядок обчислень. Програми природним чином структуруються на дві частини - складання графа обчислень і виконання обчислень в створених структурах.

Граф обчислень в TF за змістом не відрізняється від такого в Theano. У попередній статті циклу дано відмінне опис цієї сутності.

В TF граф складається з плейсхолдеров, змінних і операцій. З цих елементів можна зібрати граф, в якому будуть обчислюватися тензори. Тензори - багатовимірні масиви, вони служать «паливом» для графа. Тензором може бути як окреме число, вектор ознак з розв'язуваної задачі або зображення, так і цілий Батч описів об'єктів або масив із зображень. Замість одного об'єкта ми можемо передати в граф масив об'єктів і для нього буде вирахувано масив відповідей. Робота TF з тензорами схожа

на те, як обробляє масиви numpy, у функціях якого можна вказати вісь масиву, щодо якої буде виконуватися обчислення.

Обчислювальні графи виконуються в сесіях. Об'єкт сесії (tf.Session) приховує в собі контекст виконання графа - необхідні ресурси, допоміжні класи, адресні простори. Існує два типи сесій – звичайні tf.Session, які реалізовані і інтерактивні (tf.InteractiveSession). Різниця між ними в тому, що інтерактивна сесія більше підходить для виконання в консолі і відразу визначає себе як сесія за замовчуванням. Основний ефект - об'єкт сесії не потрібно передавати у функції обчислення як параметр. У прикладах далі я буду вважати, що в даний момент працює інтерактивна сесія, яку ми оголосили в першому прикладі, і коли знадобиться звернення до сесії, буду звертатися до об'єкта sess.

Створимо, наприклад, тензор, заповнений нулями.

```
zeros_tensor = tf.zeros([3, 3])
```

Взагалі, API в TF багато в чому нагадуватиме numpy і tf.zeros()- далеко не єдина функція, що має прямий аналог в numpy. Щоб побачити значення тензора, його потрібно виконати. Детальніше про виконання графа трохи нижче, поки що обійдемося тим, що виведемо значення тензора і сам тензор.

```
print(zeros_tensor.eval())
print(zeros_tensor)
>>> [[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
>>> Tensor("zeros_1:0", shape=(3, 3), dtype=float32)
```

Різниця між рядками полягає в тому, що в першому рядку відбувається обчислення тензора, а у другому рядку ми просто друкуємо уявлення об'єкта.

Опис тензора показує нам кілька важливих речей:

У тензорів є імена. У нашого воно zeros: 0. Існує поняття форми тензора, воно схоже на розмірність масиву з numpy. Тензори типізовані і типи для них задаються з бібліотеки.

Над тензорами можна здійснювати різноманітні операції:

```
a = tf.truncated_normal([2, 2])
b = tf.fill([2, 2], 0.5)
print(sess.run(a + b))
print(sess.run(a - b))
print(sess.run(a * b))
print(sess.run(tf.matmul(a, b)))

>>> [[-1.12130964 -1.02217746]
 [ 0.85684788  0.5425666 ]]
>>> [[ 0.35249496  0.96118248]
 [-1.55395389 -1.18111515]]
>>> [[-0.06559008 -0.11100233]
 [ 0.51474923 -0.27813852]]
>>> [[-0.16202734 -0.16202734]
 [-0.8864761  -0.8864761 ]]
```

В наведеному вище прикладі ми використовуємо конструкцію sess.run- це метод виконання операцій графа в сесії. У першому рядку я створив тензор з усіченого нормального розподілу. Для нього використовується стандартна генерація нормального розподілу, але з нього виключається все, що випадає за межі двох стандартних відхилень. Крім цього генератора є рівномірний, просте нормальне, гамма і ще кілька інших розподілів. Дуже характерна для TF штука - вже реалізовано більшість популярних варіантів виконання операції і, можливо, перед винаходом велосипеда варто поглянути на документацію. Другий тензор - це заповнений значенням 0.5 багатовимірний масив розміру 2x2 і це щось схоже на numpy і його функції створення багатовимірних масивів.

Створимо тепер змінну на основі тензора:

```
v = tf.Variable(zeros_tensor)
```

Мінлива бере участь в обчисленнях як вузол обчислювального графа, зберігає стан, і їй потрібна якась ініціалізація. Так, якщо в наступному прикладі обійтися без першого рядка, то TF викине виняток.

```
sess.run(v.initializer)
v.eval()

>>> array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]], dtype=float32)
```

Операції над змінними створюють обчислювальний граф, який можна потім виконати. Ще є плейсхолдери - об'єкти, які параметрізуется граф і відзначають місця для підстановки зовнішніх значень. Як написано в офіційній документації, плейсхолдер - це обіцянка підставити значення потім. Створимо плейсхолдер і призначаємо йому тип даних і розмір:

```
x = tf.placeholder(tf.float32, shape=(4, 4))
```

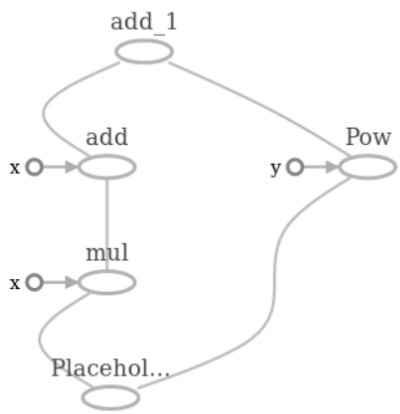
Ще такий приклад використання. Тут два плейсхолдера служать вхідними вузлами для суматора:

```
a = tf.placeholder("float")
b = tf.placeholder("float")
y = tf.multiply(a, b)
print(sess.run(y, feed_dict={a:100, b:500}))
>>> 50000.0
```

Як приклад створимо і обчислимо кілька виразів.

```
x = tf.placeholder(tf.float32)
f =  1 + 2 * x + tf.pow(x, 2)
sess.run(f, feed_dict={x: 10})
>>> 121.0
```

І граф обчислення:

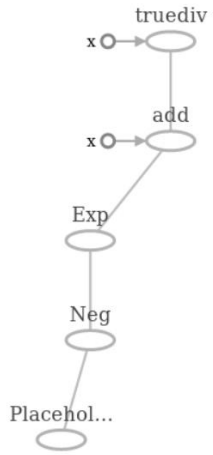


x і *y*, що вказують на операції в цій схемі - це додаткові параметри, замість яких могли б бути ребра графа, але ми підставили в *f* скалярні значення 1 і 2 і це просто позначення в графі для чисел. У цьому прикладі ми створюємо плейсхолдер і на його основі - граф вираження*1+2x+x2*, А після цього виконуємо обчислення графа в контексті поточної сесії. Я не вказав форму в параметрах плейсхолдера і це означає, що можна подавати на вхід тензори будь-яких розмірів. Єдине, що необхідно вказати - це тип тензора. Параметри при обчисленні всередину сесії передаються через *feed_dict*- словник з усім, що необхідно для обчислень.

Спробуємо зібрати що-небудь більш практично значуще. Ось, наприклад, сигмоїда :*σ(x)=1/1+e-x*

```
x = tf.placeholder(dtype=tf.float32)
sigma = 1 / (1 + tf.exp(-x))
sigma.eval(feed_dict={x: np.linspace(-5, 5) })
```

І ось такий граф для неї.



У фрагменті з запуском обчислення функції є один момент, який відрізняє цей приклад від попередніх. Справа в тому, що в плейсхолдер замість одного скалярного значення ми передаємо цілий масив. TF обробляє всі значення масиву разом, в рамках одного тензора (пам'ятаємо, що масив == тензор). Точно таким же чином ми можемо передавати в граф об'єкти цілими Батче і поставляти нейронної мережі картинки цілком.

В цілому робота з тензорами нагадує роботу з масивами в *numpy*. Однак, є деякі відмінності. Коли ми хочемо знизити розмірність, будь-яким чином об'єднавши значення в тензор за певним виміру, ми користуємося тими функціями, які починаються з *reduce_*.Якщо порівняти с *API Theano* - в *TF* немає поділу на вектори і матриці, але замість цього доводиться стежити за размерностями тензорів в графі і є механізм виведення форми тензора, який дозволяє отримати розмірності ще до runtime.

Спочатку *Keras* виріс як зручна надбудова над *Theano*. Звідси і його грецьке ім'я - κέρας, що означає "ріг" по-грецьки, що, в свою чергу, є відсиланням до Одіссеї Гомера. Хоча, з тих пір спливло багато води, і *Keras* став спочатку підтримувати

Tensorflow, а потім і зовсім став його частиною. Втім, наша розповідь буде присвячений не складну долю цього фреймворка, а його можливостям. Якщо вам цікаво, ласкаво просимо під кат.

Бекенд - це те, через що *Keras* став відомий і популярний (крім інших достоїнств, які ми розберемо нижче). *Keras* дозволяє використовувати в якості бекенд різні інші фреймворки. При цьому написаний вами код буде виконуватися незалежно від використовуваного бекенд. Починалася розробка, як ми вже говорили, з *Theano*, але з часом додався *Tensorflow*. Зараз *Keras* за замовчуванням працює саме з ним, але якщо ви хочете використовувати *Theano*, то є два варіанти, як це зробити:

Відредагувати файл конфігурації *keras.json*, який лежить по дорозі *$HOME/.keras/keras.json* (або *%USERPROFILE%\.keras\keras.json* в разі операційних систем сімейства *Windows*). Нам потрібно поле *backend*:

```
{
"image_data_format": "channels_last",
"epsilon": 1e-07,
"floatx": "float32",
"backend": "theano"
}
```

Другий шлях - це задати змінну оточення KERAS_BACKEND, наприклад, так:

```
KERAS_BACKEND=theano python -c "from keras import backend"
Using Theano backend.
```

Варто відзначити, що зараз ведеться робота з написання біндінг для *CNTK* від *Microsoft*, так що через деякий час з'явиться ще один доступний бекенд. Стежити за цим можна тут .

Також існує *MXNet Keras backend* , який поки не володіє всією функціональністю, але якщо ви використовуєте *MXNet*, ви можете звернути увагу на таку можливість.

Ще існує цікавий проект *Keras.js* , що дає можливість запускати натреновані моделі *Keras* з браузера на машинах, де є *GPU*.

Так що бекенд *Keras* ширяться і з часом захоплять світ! (Але це не точно.)

Навчання будь-якої моделі в машинному навчанні починається з даних. *Keras* містить всередині кілька навчальних датасета, але вони вже приведені в зручну для роботи форму і не дозволяють показати всю міць *Keras*. Тому ми візьмемо більш сирої датасета. Це буде датасета 20 *newsgroups* - 20 тисяч новинних повідомлень з груп *Usenet* (це така система обміну поштою родом з 1990-х, споріднена *FIDO*, який, може бути, трохи краще знаком читачеві) приблизно порівну розподілені по 20 категоріям.

Keras містить в собі інструменти для зручного препроцессінга текстів, картинок і часових рядів, іншими словами, найпоширеніших типів даних. Сьогодні ми працюємо з текстами, тому нам потрібно розбити їх на маркери і привести в матричну форму.

2.3. Огляд популярних архітектур

Архітектура CNN є ключовим фактором у визначенні його продуктивності та ефективності. Спосіб структурування шарів, які елементи використовуються в кожному шарі та як вони сконструйовані, часто впливатиме на швидкість та точність, з якою він може виконувати різні завдання.

Проект ImageNet - це візуальна база даних, призначена для використання при дослідженні програмного забезпечення для розпізнавання візуальних об'єктів. Проект ImageNet налічує понад 14 мільйонів зображень, спеціально розроблених для навчання CNN в області виявлення об'єктів, один мільйон з яких також забезпечує «обмежувальні коробки» для використання таких мереж, як YOLO .

З 2010 року проект проводить щорічний конкурс під назвою ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Учасники змагань будують програмні програми, які намагаються правильно виявити та класифікувати об’єкти та сцени в межах заданих зображень. В даний час у виклику використовується скорочений список із тисячі окремих класів.

Коли розпочалися щорічні змагання ILSVRC, хороший показник класифікації становив 25%, перший серйозний стрибок у виконанні досягнута мережею під назвою AlexNet у 2012 році, яка знизила рівень класифікації на 10%. У наступні роки рівень помилок знизився до нижчих відсотків і, нарешті, перевищив можливості людини.

LeNet-5 (1998)

Цей 7-шаровий CNN класифікує цифри, оцифровані 32 × 32 пікселів вхідних зображень із сірим кольором. його використовували кілька банків для розпізнавання рукописних номерів на чеках.

AlexNet (2012)

AlexNet розроблений групою SuperVision, що має схожу архітектуру з LeNet, але глибше— вона має більше фільтрів на шар, а також укладені згорткові шари. Він складається з п’яти згорткових шарів, за якими слідують три повністю пов'язані шари.

Однією з найбільш значущих відмінностей між AlexNet та іншими алгоритмами виявлення об'єктів є використання ReLU для нелінійної частини замість функції Sigmond або Tanh, як традиційні нейронні мережі. AlexNet використовує швидше навчання ReLU, щоб зробити свій алгоритм швидшим.

Творці AlexNet розділити свою мережу на два трубопроводи , тому що вони використовували два Nvidia Geforce GTX 580 графічних процесорів (GPU) для підготовки їх CNN.

GoogleNet (2014)

Створена за допомогою CNN, натхненна LetNet, мережа GoogleNet, яка також називається Inception V1, була створена командою Google. GoogleNet став переможцем ILSVRC 2014 та досяг 5-відсоткових помилок менше 7%, що наближається до рівня продуктивності людини.

Архітектура GoogleNet складалася з 22-х шарового CNN, який використовував модуль, заснований на невеликих згортках, який називається "модулем початку", який використовував пакетну нормалізацію, RMSprop та зображення, щоб зменшити кількість параметрів з 60 мільйонів, як в AlexNet, до всього 4 мільйонів.

VGGNet (2014)

VGGNet, що займав підготовку до ILSVRC 2014, складався з 16 згорткових шарів. Подібно до AlexNet, він використовував лише 3 × 3 згортки, але додав більше фільтрів. VGGNet тренувався на 4 графічних процесорах більше двох тижнів, щоб досягти його продуктивності.

Проблема з VGGNet полягає в тому, що він складається з 138 мільйонів параметрів, що в 34,5 разів більше, ніж GoogleNet, що робить його складним для запуску.

ResNet (2015)

ResNet - це скорочена назва для Residual Network (дослівно - «залишкова мережу»), але що таке residual learning («залишкове навчання»)?

Глибокі згорткові нейронні мережі перевершили людський рівень класифікації зображень в 2015 році. Глибокі мережі витягають низько-, середньо-і високорівневі ознаки наскрізним багатошаровим способом, а збільшення кількості stacked layers може збагатити «рівні» ознак. Коли глибша мережу починає згортатися, виникає проблема: зі збільшенням глибини мережі точність спочатку збільшується, а потім швидко погіршується. Зниження точності навчання показує, що не всі мережі легко оптимізувати.

Щоб подолати цю проблему, Microsoft ввела глибоку «залишкову» структуру навчання. Замість того, щоб сподіватися на те, що кожні кілька stacked layers безпосередньо відповідають бажаному основному поданням, вони явно дозволяють цим верствам відповідати «залишковим». Формулювання F (x) + x може бути реалізована за допомогою нейронних мереж з сполуками для швидкого доступу.

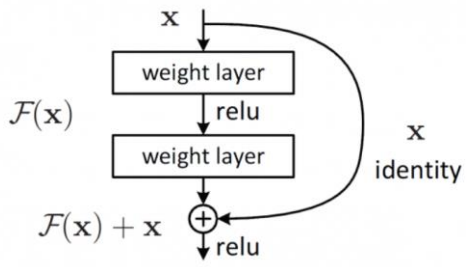


Рис. 2.1.Приклад сполуки для швидкого доступу.

З'єднання швидкого доступу (shortcut connections) пропускають один або кілька шарів і виконують зіставлення ідентифікаторів. Їх виходи додаються до виходів stacked layers. Використовуючи ResNet, можна вирішити безліч проблем, таких як:

ResNet відносно легко оптимізувати: «прості» мережі (які просто складають шари) показують велику помилку навчання, коли глибина збільшується.

ResNet дозволяє відносно легко збільшити точність завдяки збільшенню глибини, чого з іншими мережами домогтися складніше.

Проста мережа : Прості базові лінії в основному натхненні філософією мереж VGG. Згорткові шари в основному мають фільтри 3 × 3 і слідують двох простих правил:

Для однієї і тієї ж вихідний карти об'єктів шари мають однакову кількість фільтрів;

Якщо розмір карти об'єктів зменшується вдвічі, число фільтрів подвоюється, щоб зберегти тимчасову складність кожного шару.

Варто відзначити, що модель ResNet має менше фільтрів і складність менше, ніж мережі VGG.

ResNet : на основі описаної вище простий мережі додано швидке з'єднання, яке перетворює мережу в її залишкову версію. Ідентифікаційні швидкі з'єднання F (x {W} + x) можуть використовуватися безпосередньо, коли вхід і вихід мають однакові розмірності. Коли розмірності збільшуються, він розглядає два варіанти:

Швидке з'єднання виконує зіставлення ідентифікаторів з додатковими нулями, доданими для збільшення розмірності. Ця опція не вводить ніяких додаткових параметрів.

Проекція швидкого з'єднання в F (x {W} + x) використовується для зіставлення розмірності (виконано за допомогою 1 × 1 згорток).

Для будь-якої з опцій, якщо швидкі з'єднання йдуть по картах об'єктів двох розмірностей, вони виконуються з кроком 2.

Кожен блок ResNet має два рівня глибини (використовується в невеликих мережах, таких як ResNet 18, 34) або 3 рівня (ResNet 50, 101, 152).

50-шарова ResNet : кожен 3-шаровий блок замінюється в 34-шарової мережі цим 3-шаровим вузьким місцем, в результаті виходить 50-шарова ResNet (див. Таблицю вище). Вони використовують варіант 2 для збільшення розмірності. Ця модель має 3,8 мільярда FLOPs.

ResNet з 101 і 152 шарами : вони створюють ResNet з 101 і 152 шарами, використовуючи більше 3-шарових блоків (див. Таблицю вище). Навіть після збільшення глибини 152-шарова ResNet (11,3 мільярда FLOP) має меншу складність, ніж мережі VGG-16/19 (15,3 / 19,6 мільярда FLOPs).

Мережа ResNet сходиться швидше, ніж її простий аналог. Малюнок 2.1 показує, що більш глибокі ResNet досягають кращих результатів навчання в порівнянні з неглибокою мережею.

ResNet-152 досягає 4,49% в top-5 помилок валідації. Комбінація з 6 моделей з різною глибиною досягає 3,57% в top-5 помилок валідації.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
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**

Рис. 2.1. Частота помилок (%) для однієї моделі в наборі валідації ImageNet (крім випадків, зазначених в тестовому наборі).

2.4. Аугментація даних

Аугментація даних (data augmentation) - це методика створення додаткових навчальних даних з наявних даних. Для досягнення хороших результатів глибокі мережі повинні навчатися на дуже великому обсязі даних. Отже, якщо вихідний навчальний набір містить обмежену кількість зображень, необхідно виконати аугментацію, щоб поліпшити результати моделі.

Бібліотека глибокого навчання Keras надає можливість автоматично збільшувати дані при навчанні моделі. Це досягається за допомогою класу ImageDataGenerator .

По-перше, клас може бути екземпляром, а конфігурація типів збільшення даних визначається аргументами до конструктора класів.

Підтримується цілий ряд методів, а також методи масштабування пікселів. Ми зосередимо увагу на п’яти основних типах примноження даних:

- Зміщення зображення через аргумент *width_shift_range* та *height_shift_range*.
- Розворот зображення через параметри *horizontal_flip* та *vertical_flip*.
- Поворот зображень за допомогою аргументу *rotation_range*
- Зміна яскравості зображення за допомогою аргумента *brightness_range*.
- Збільшення зображення за допомогою аргументу *zoom_range* .

Наприклад, може бути побудований екземпляр класу ImageDataGenerator.

```
...
# create data generator
datagen = ImageDataGenerator()
```

Після побудови ітератора можна створити для набору даних зображення. Ітератор поверне одну групу доповнених зображень для кожної ітерації.
Ітератор може бути створений із набору даних, завантажених у пам'ять за допомогою функції *flow;* наприклад:

```
...
# load image dataset
X, y = ...
# create iterator
it = datagen.flow(X, y)
```

Крім того, ітератор може бути створений для набору даних зображень, розташованих на диску у визначеному каталозі, де зображення в цьому каталозі організовані в підкаталоги відповідно до їх класу.

```
...
# create iterator
it = datagen.flow_from_directory(X, y, ...)
```

Після створення ітератора його можна використовувати для тренування моделі нейронної мережі, викликавши функцію *fit_generator ()* .

Аргумент *steps_per_epoch* повинен визначати кількість партій зразків, що складаються з однієї епохи. Наприклад, якщо ваш початковий набір даних має 10 000 зображень, а розмір вашої партії - 32, то розумним значенням для *steps_per_epoch* при встановленні моделі на доповнені дані може бути *ceil (10000/32)* або 313 партій.

```
# define model
model = ...
# fit model on the augmented dataset
model.fit_generator(it, steps_per_epoch=313, ...)
```

Зображення в наборі даних не використовуються безпосередньо. Натомість моделі надаються лише доповнені зображення. Оскільки аугментації виконуються випадковим чином, це дозволяє генерувати та використовувати під час тренування як модифіковані зображення, так і близькі до оригінальних зображень (наприклад, майже без збільшення).

Генератор даних також може бути використаний для визначення даних валідації та тестового набору даних. Часто використовується окремий екземпляр *ImageDataGenerator* , який може мати таку ж конфігурацію масштабування пікселів (не розглянуту в цьому підручнику), як екземпляр *ImageDataGenerator, який* використовується для навчального набору даних, але не використовує збільшення даних. Це пояснюється тим, що збільшення даних використовується лише як техніка штучного розширення навчального набору даних з метою поліпшення продуктивності моделі на незаангажованому наборі даних.

Горизонтальний та вертикальний зсув

Перехід до зображення означає переміщення всіх пікселів зображення в одному напрямку, наприклад горизонтально або вертикально, зберігаючи однакові розміри зображення.

Це означає, що деякі пікселі будуть відрізані від зображення, і буде область зображення, де потрібно буде вказати нові значення пікселів.

Аргументи *width_shift_range* і *висота_shift_range* для конструктора *ImageData Generator* керують величиною горизонтального та вертикального зсуву відповідно.

Ці аргументи можуть задавати значення з плаваючою комою, яке вказує на відсоток (від 0 до 1) від ширини або висоти зображення для зміщення. Крім того, для зміщення зображення може бути задана кількість пікселів.

Зокрема, значення в діапазоні між відсутністю зсуву та відстотковою чи піксельною величиною будуть вибіркові для кожного зображення та виконаного зсуву, наприклад [0, значення]. Крім того, ви можете вказати кортеж або масив діапазону min та max, з якого буде відібрано зсув; наприклад: [-100, 100] або [-0,5, 0,5].

Наведений нижче приклад демонструє зсув горизонталі з аргументом *width_shift_range* між [-200,200] пікселями та генерує графік згенерованих зображень для демонстрації ефекту.

```
# example of horizontal shift image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(width_shift_range=[-200,200])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

Запустивши приклад, створюється екземпляр класу ImageDataGenerator, налаштований для збільшення зображення, а потім створює ітератор. Потім ітератор викликається дев'ять разів у циклі, і кожне збільшене зображення виводиться.

На рисунку (рис. 2.1.) можна побачити, що виконувались діапазон різних випадково вибраних позитивних та негативних горизонтальних зсувів, а значення пікселів на краю зображення дублюються для заповнення порожньої частини зображення, створеної зсувом.

Нижче наведено той самий приклад, оновлений для виконання вертикальних зрушень зображення за допомогою аргументу *height_shift_range*, в цьому випадку вказуючи відсоток зображення, який зміщується на 0,5 від висоти зображення.

```
# example of vertical shift image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(height_shift_range=0.5)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

Наведений приклад створює графік зображень, доповнених випадковими позитивними та негативними вертикальними зрушеннями.

Можна побачити на рисунку (рис 2.2.), що як горизонтальний, так і вертикальний позитивні та негативні зрушення, ймовірно, мають сенс для обраної фотографії, але в деяких випадках повторювані пікселі на краю зображення можуть не мати сенсу для моделі.

Слід зауважити, що інші режими заповнення можуть бути визначені за допомогою аргументу " *fill_mode* ".

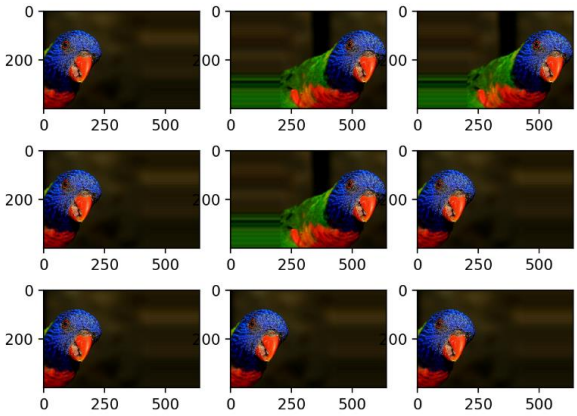


Рис. 2.1. Приклад горизонтального зсуву

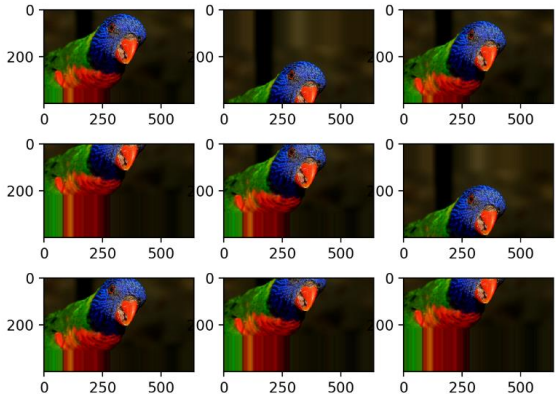


Рис. 2.2.Приклад вертикального зсуву

Горизонтальний та вертикальний розворот зображення

Перевернути зображення означає обертання рядків або стовпців пікселів у разі вертикального або горизонтального перевороту відповідно.

Задається булевим аргументом *vertical_flip* або *horizontal_flip* класу *ImageDataGenerator*

Для типів зображень, таких як аерофотознімки, космологічні фотографії та мікроскопічні фотографії, можливо, вертикальні фліпи мають сенс.

Наведений нижче приклад демонструє доповнення обраної фотографії горизонтальним переворотом через аргумент *horizontal_flip* .

```
# example of horizontal flip image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(horizontal_flip=True)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

Наведений приклад створює сюжет із дев'яти доповнених зображень. На рисунку (рис 2.3) можена бачити, що горизонтальний розворот застосовується випадковим чином до деяких зображень, а не до інших.

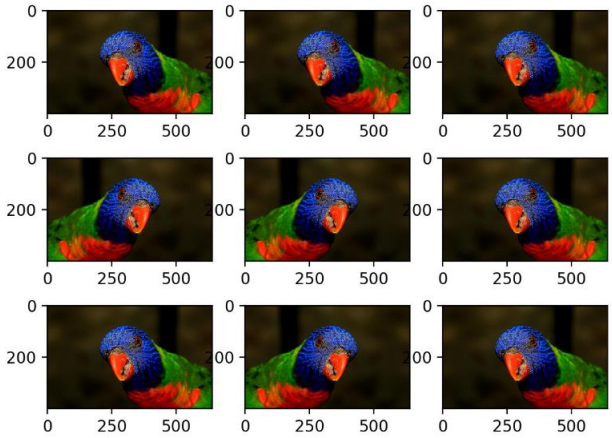


Рис. 2.3. Приклад горизонтального розвороту зображення

Випадкове обертання

Обертання випадковим чином обертає зображення за годинниковою стрілкою на задану кількість градусів від 0 до 360.

Обертання, ймовірно, обертає пікселі з кадру зображення та залишає ділянки кадру без даних про пікселі, які слід заповнити.

Наведений нижче приклад демонструє випадкові обертання через аргумент *rotation_range* , з обертаннями до зображення між 0 і 90 градусами.

Запуск прикладу генерує приклади обертового зображення (рис. 2.4.), показуючи в деяких випадках пікселі, обернені з кадру та заливку найближчим сусідом.

```
# example of random rotation image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(rotation_range=90)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

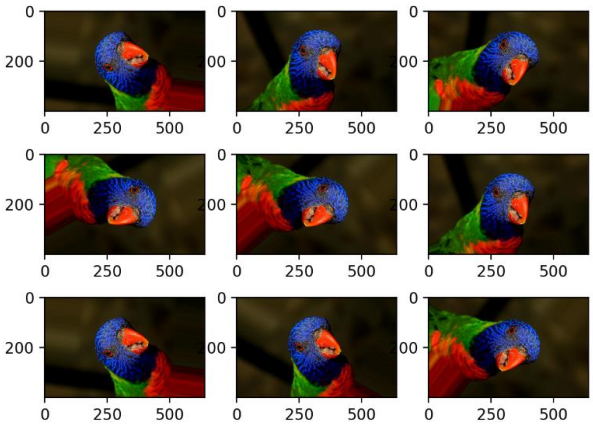


Рис. 2.4. Приклад довільного обертання.

Випадкове збільшення яскравості

Яскравість зображення можна збільшити або зменшити випадковою зміною яскравості.

Намір полягає в тому, щоб дозволити моделі узагальнювати зображення, що навчаються на різних рівнях освітлення.

Цього можна досягти, вказавши аргумент *brightness_range* який задає мінімум та максимум діапазон як поплавок, що представляє відсоток для вибору яскравішої суми.

Значення менше 1,0 затемнюють зображення, наприклад [0,5, 1,0], тоді як значення, що перевищують 1,0, освітлюють зображення, наприклад [1,0, 1,5], де 1,0 не впливає на яскравість.

Наведений нижче рисунок (рис. 2.5.) демонструє збільшення зображення яскравості, дозволяючи генератору випадково затемнити зображення між 1,0 (без змін) і 0,2 або 20%.

```
# example of brighting image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(brightness_range=[0.2,1.0])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

Наведений приклад показує доповнені зображення із застосованою різною кількістю затемнення.

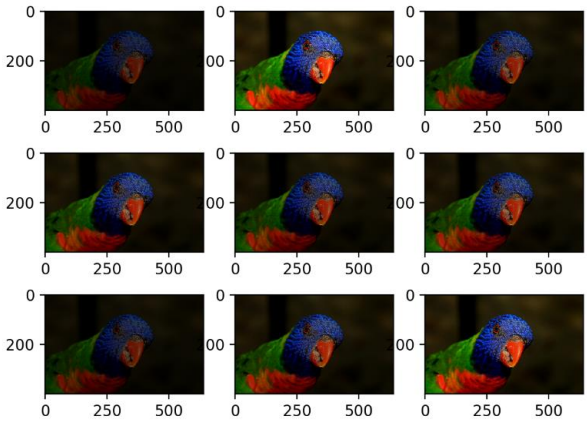


Рис. 2.5. Приклад довільної зміни яскравості

Випадкове збільшення масштабу

Збільшення масштабу випадковим чином збільшує зображення і або додає нові значення пікселів навколо зображення, або інтерполює значення пікселів відповідно.

Масштабування зображення може бути налаштовано аргументом *zoom_range* конструктора *ImageDataGenerator* . Ви можете вказати відсоток масштабу як масив чи кортеж.

Наприклад, якщо вказати 0,3, то діапазон буде [0,7, 1,3], або між 70% (збільшення) і 130% (зменшення масштабу).

Наведений нижче приклад демонструє збільшення зображення, наприклад збільшення об'єкта на фотографії.

```python
# example of zoom image augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
# load the image
img = load_img('bird.jpg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(zoom_range=[0.5,1.0])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

Наведений приклад створює приклад збільшення зображення, показуючи випадкове збільшення, яке відрізняється як за розмірами ширини, так і висоти, що також випадковим чином змінює співвідношення сторін об'єкта на зображенні.

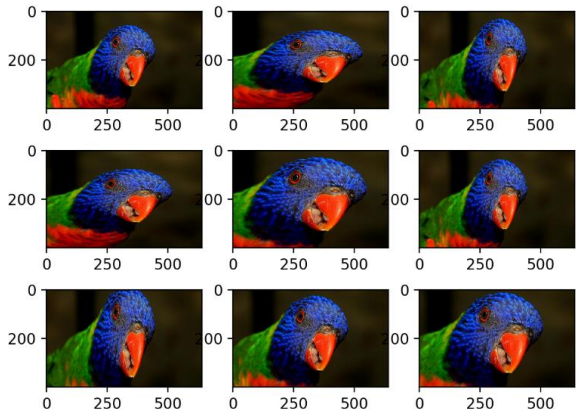


Рис. 2.6. Приклад довільного збільшення.

2.5.Висновки

В матеріалах 2 розділу було розглянута ЗНМ як метод розпізнавання образів. Розглянуто можливості бібліотек *keras* та *tensorflow*. Був проведений аналіз переваг

та недоліків архітектур сучасних ЗНМ. Також була розгянута та продемонстрована процедура штучного збільшення вибірки – аугментація.

АВТОМАТИЗОВАНА СИСТЕМА РОЗПІЗНАВАННЯ ДЕРЕВ ТА ЛІСОВИХ НАСАДЖЕНЬ

В матеріалах даного розділу детально описано за допомогою словника ознак кожний тип місцевості, що будуть розпізнаватися. Наведено алгоритми процедур розпізнавання дерев та лісових насаджень, фільтрації патчів.

В першому підрозділі було описано використання методу ЗНМ для знаходження дерев та лісових насаджень на основі даних з аерофотознімків, програмне забезпечення.

В другому підрозділі було проведено тестування розробленого програмного забезпечення.

3.1. Опис програмного забезпечення.

Інформаційна технологія - цілеспрямована організована сукупність інформаційних процесів з використанням засобів обчислювальної техніки, що забезпечують високу швидкість обробки даних, сукупність методів та засобів для обробки та переказу даних задля отримання інформації нової якості про стан об’єкту, процесу або явища.

Алгоритм розпізнавання дерев та лісових насаджень включає в себе наступні етапи:

- Завантаження навченої ЗНМ до програми.
- Завантаження аерофотознімка до програми.
- Поділ вхідного знімка на рівні частини та підготовка кожної частини до подання його на вхід ЗНМ.
- За допомогою ЗНМ для кожної частини знімка визначається степінь впевненості належності до певного класу.
- Якщо максимальна степінь впевненості для певної частини аерофотознімка є меншою ніж деякий поріг, то слід вважати цю частину, як невизначений клас.
- Підрахувати кількість отриманих частин по кожному класу окремо.

- Виділити всі частини знімку які відносяться до класу ліс.
- Підрахувати відносну площу для кожного класу.
- Вивести аерофотознімок та відносну площу по кожному класу.

Узагальнену блок-схему роботи розробленого алгоритму розпізнавання дерев та лісових насаджень можна зобразити наступним чином рисунок (рис. 3.14.).

Архітектуру ЗНМ наведено на рисунках (рис. 3.15. а) – б)). Для навчання ЗНМ використовувалась БД яка була отримана під час сегментації кадрів, дану архітектуру було написано на високоріневій мові програмування python з використанням бібліотеки keras та tensorflow. В якості попередньої обробки було використано бібліотеку OpenCV.

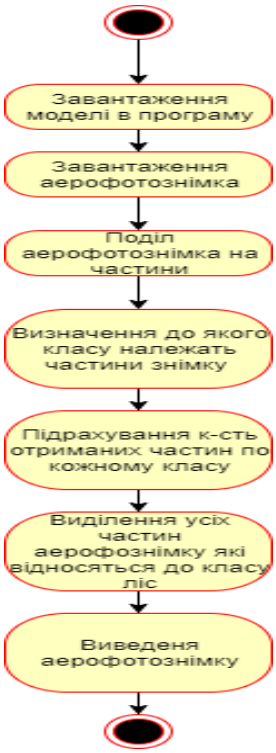


Рис. 3.14. Узагальнена схема роботи алгоритму знаходження дерев та лісових насаджень на аерофотознімках.

Розроблена автоматизована система здатна за допомогою вже навченої ЗНМ може проводити фільтрацію частин зображень за допомогою виведених порогових значень для кожного класу.

На вхід подаються патчі серед яких можуть бути «цільові об’єкти».

Завдяки тому, що ЗНМ була навчена за допомогою бібліотеки keras та на вихідному шарі як функція активації використовується softmax та тренування проводилися на даних серед яких немає «цільових об’єктів», будемо вважати що їх аномальними. Оскільки ЗНМ після процедури класифікації зображення повертає степені впевненості належності до кожного класу окремо ми можемо вивести оптимальні пороги для кожного класу при яких частка відфільтрованих даних буде максимальною а частка відфільтрованих «цільових об’єктів» буде мінімальною. Якщо модель віднесла патч до певного класу, але з степеню впевненості нижчою ніж порогова для даного класу, то слід вважати що дане зображення як хибно класифіковане. Такі об’єкти можуть подаватися оператору для подальшого перегляду вручну, та подальшого віднесення до певного існуючого класу.

Узагальнену блок-схему роботи розробленого алгоритму фільтрації зображень можна зобразити наступним чином рисунок (рис. 3.15.).

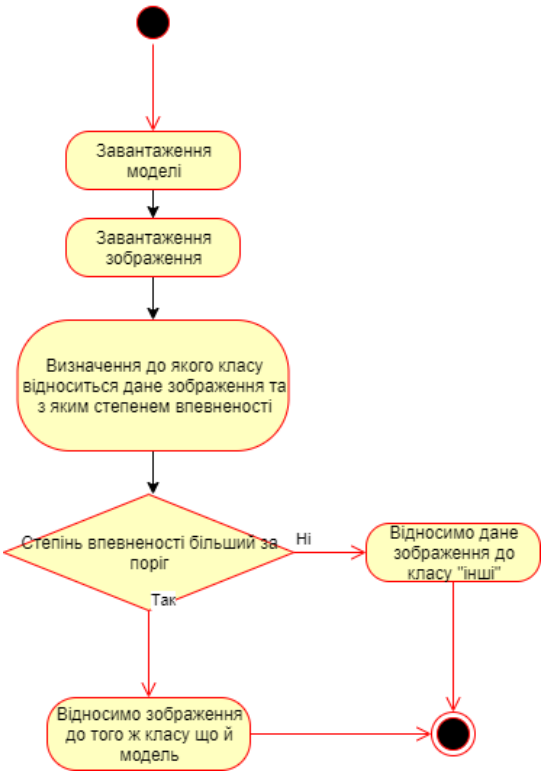


Рис. 3.15. Узагальнена схема роботи алгоритму фільтрації зображень.

Звичайно перед використанням даних алгоритмів потрібно завантажити деякі додаткові бібліотеки а саме:

- *Opencv* – використовується для роботи з ЦЗ.
- *Tensorflow* – дане пакетне забезпечення дозволяє трансформувати данні у форматі який приймає на вхід *keras*, цю бібліотеку використовують як *background*.
- *Keras* – основна та найпоширеніша бібліотека для машинного навчання на мові програмування *python*.
- *Numpy* – було використано для обробки даних.
- *Pandas* - було використано для обробки даних.

- *Matplotlib* – дана бібліотека використовувалась для візуалізації та представлення даних.

3.2. Підготування даних для навчання

Для навчання ЗНМ використовувалась БД, що складається з 6932 зображень, які поділяються на наступні класи:

- 0 – клас ліс.
- 1 – клас ліс та будинки.
- 2 – клас ліс та поле.
- 3 – клас ліс та дорога.
- 4 – клас будинки.
- 5 – клас поле.
- 6 – клас розоране поле.

Приклади місцевостей зображено на рисинках (рис. 4.1 а) - е).

Рис. 4.1 а) Місцевість класу ліс

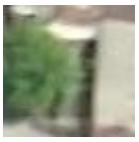


Рис. 4.1 б) Місцевість класу ліс і будинки

Рис. 4.1 в) Місцевість класу ліс і поле

Рис. 4.1 г) Місцевість класу ліс і дорога

Рис. 4.1 ґ) Місцевість класу будинки

Рис. 4.1 д) Місцевість класу поле

Рис. 4.1 е) Місцевість класу розоране поле

Для навчання ЗНМ необхідно велика кількість даних, тому було вирішено провести процедуру штучного збільшення даних – аугментацію.

Дану процедуру було реалізовано за допомогою бібліотеки keras, а саме функції ImageDataGenerator у якій реалізовано велика кількість можливих операцій над вхідним зображенням таких як:

- Вертикальний та горизонтальний зсув.
- Вертикальний та горизонтальний розворот.
- Випадкове обертання на довільний кут.
- Збільшення або зменшення яскравості.
- Збільшення масштабу.

Оскільки, всі вище перераховані процедури, крім вертикального та горизонтального розвороту, з певною степеню спотворюють вхідне зображення, для процедури аугментації було вирішено використовувати тільки розворот

3.3. Тестування ЗНМ.

Параметри нейронної мережі:

- Розмір партії 32.
- В якості оптимізатора градієнтного спуску було використано оптимізатор *rmsprop* з швидкістю навчання (*learn rate*) *0.0002* параметром розпаду *0.000004*. Це допомагає не застрявати градієнтному спуску в локальних мінімумах.
- В якості функції помилок використано категоріальну кроссєнтропію.
- А в якості метрики використано *accuracy* – відсоток правильно класифікованих типів місцевості.

Вибір показників ефективності системи розпізнавання та оцінка їх значень. На таблиці (табл.4.1) показано схему для визначення метрики правильно класифікованих об’єктів.

Таблиця 4.1.

Визначення істинності результату класифікаційної системи

Категорія		Експертні данні	
		«Вірно»	«Невірно»
Оцінка системи	«Вірно»	TP	FP
	«Невірно»	FN	TN

де:

- TP – істинне «вірне» значення.
- TN – істинне «невірне» значення.
- FP – брехливе «вірне» значення.
- FN – брехливе «невірне» значення.

Точність та повнота визначається:

$$recall = \frac{TP}{TP + FN},$$

$$precision = \frac{TN}{TN + FP}.$$

Точність методу визначається за формулою:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

На таблицях (табл. 4.1 – 4.2) зображено результат навчання ЗНМ процес навчання якої був зображений на рисунку (рис. 4.8.) здатна прогнозувати класи на всій БД. Обсяг тестових зображень якої складає 1336. На перетині класів зображено кількість вірно та помилково класифікованих класів.

Таблиця 4.1.

Матриця помилок та правильних рішень (*confusionMatrix*).

Класи	0	1	2	3	4	5	6
0	**421**	0	5	2	0	1	1
1	3	**91**	0	16	12	0	0
2	17	2	**103**	12	1	5	1
3	7	4	5	**110**	3	0	0
4	2	11	0	14	**145**	0	0
5	3	0	0	0	0	**144**	23
6	2	0	6	2	0	7	**155**

Таблиця 4.2.

Статистики класифікаці.

Класи	0	1	2	3	4	5	6
presision	0.93	0.84	0.87	0.71	0.90	0.92	0.86
recoll	0.98	0.75	0.73	0.85	0.84	0.85	0.90
F1-score	0.95	0.79	0.79	0.77	0.87	0.88	0.88

Аналізуючи матрицю помилок та правильних рішень з таблиці (табл. 4.1) можна зробити висновки про те, що звичайно є незначні помилки *1*-го та *2*-го роду, але вони незначні в порівнянні з кількістю правильно класифікованих класів.

Наступним кроком є тестування роботи алгоритму аналізу та розпізнавання дерев та лісових насаджень за даними аерофотознімків та підрахування відносної площі лісових насаджень. Для цього були взяті кадри з відео знятого за допомогою БПЛА. На рисунках (рис. 4.2) наведено приклад роботи даного алгоритму.

ліс:50.0%
ліс_і_будинки:10.0%
ліс_і_поле:2.0%
ліс_і_дорога:2.0%
будинки:0.0%
поле:33.0%
розоране_поле:0.0%
інше:2.0%

Рис. 4.2. а) Приклад роботи алгоритму розпізнавання дерев та лісових насаджень

ліс:32.0%
ліс_і_будинки:24.0%
ліс_і_поле:21.0%
ліс_і_дорога:8.0%
будинки:3.0%
поле:4.0%
розоране_поле:5.0%
інше:2.0%

Рис. 4.2. б) Приклад роботи алгоритму розпізнавання дерев та лісових насаджень

ліс:22.0%
ліс_і_будинки:14.000000000000002%
ліс_і_поле:24.0%
ліс_і_дорога:20.0%
будинки:1.0%
поле:11.0%
розоране_поле:6.0%
інше:1.0%

Рис. 4.2. в) Приклад роботи алгоритму розпізнавання дерев та лісових насаджень

Рис. 4.2. г) Приклад роботи алгоритму розпізнавання дерев та лісових насаджень

З рисунків (рис. 4.2. а) – г)) видно, що ліс був виділений за допомогою сірого кольору та алгоритм працює добре та з невеликими помилками знаходить лісові насадження. Підрахування відносної площі лісових масивів та інших класів відбувається коректно.

3.4. Тестування роботи процедури фільтрації патчів.

Після того, як було проведено тестування роботи ЗНМ щодо правильної класифікації місцевості наступним кроком було тестування алгоритму фільтрації патчів.

Для цього було проведено аналіз степені впевненості для кожного патча та виведення оптимальних порогів для проведення фільтрації, виведення яких для кожного класу окремо зображено на рисунках ()

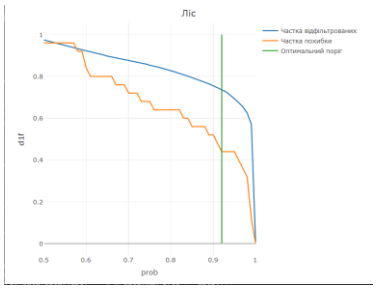


Рис. 4.3. а) Виведення порогу для класу ліс

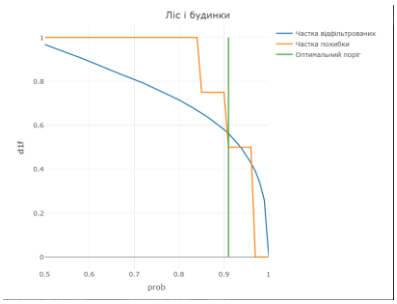
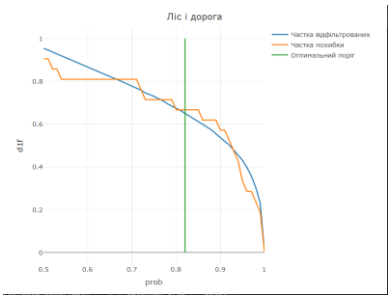


Рис. 4.3. б) Виведення порогу для класу ліс і будинки



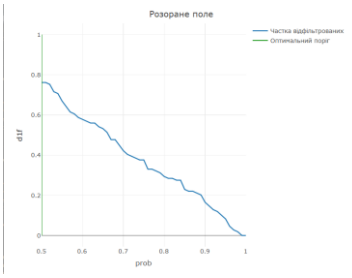


Рис. 4.2. ґ) Виведення порогу для класу розоране поле

Рис. 4.3. в) Виведення порогу для класу ліс і поле

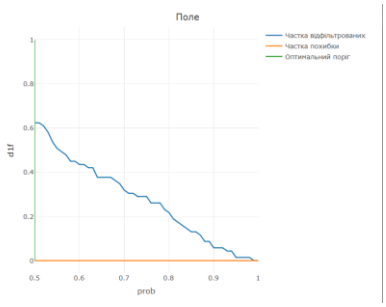


Рис. 4.2. г) Виведення порогу для класу поле

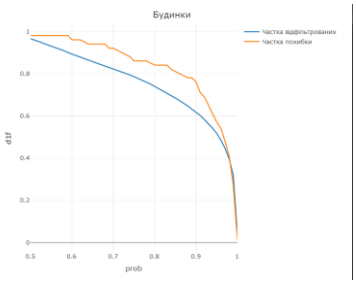


Рис. 4.2. д) Виведення порогу для класу будинку

Для класу будинки (рис. 4.2.) не вдалось вивести оптимальний поріг, оскільки «цільові об’єкт» в основному дуже похожі на екземпляри класу будинки. Тому для оптимальної фільтрації, всі патчі які були віднесені до класу будинки ми будемо вважати як цільові.

Тестування роботи даного алгоритму проводились на вибірки з 55824 патчів серед яких 156 «цільових об’єктів».

В таблиці 4.3. наведено результат роботи даного алгоритму.

Таблиця 4.3.

Результат роботи алгоритму фільтрації патчів.

Класи	**Поріг**	**К-сть зображень які відкидаються після фільтрації**	**К-сть помилково відфільтрованих цільових об’єктів**
Ліс	0.92	4860	11
Ліс і будинки	0.91	9703	2
Ліс і поле	0.91	2489	2
Ліс і дороги	0,95	5351	7
Поле	0,5	43	0
Розоране поле	0,5	83	0
Сума		22529	22
Показники відносно загальної к-сті зображень		40.3%	14%

Також були виведені альтернативні пороги по кожному класу для збільшення частки фільтрації, які наведені в таблиці 4.4.

Таблиця 4.4.

Результат роботи алгоритму фільтрації патчів.

Класи	Поріг	К-сть зображень які відкидаються після фільтрації	К-сть помилково відфільтрованих цільових об’єктів
Ліс	0.75	4860	11
Ліс і будинки	0.76	9703	2
Ліс і поле	0.75	2489	2
Ліс і дороги	0,73	5351	7
Будинки	0,999	2084	9
Поле	0,5	43	0
Розоране поле	0,5	83	0
Сума		33666	47
Показники відносно загальної к-сті зображень		60.3%	30,1%

3.5. Висновок

В даному розділі реалізовано побудову класифікаційної системи розпізнавання, та вирішено дві задачі для досягнення мети:

- Розроблено алгоритм керування роботою системи. Процес функціонування даної системи (ЗНМ) був у певному сенсі оптимальним та вибраний критерій правильно класифікованих об’єктів кожного класу досягав екстремального значення.

- Обрано показник ефективності системи - відсоток правильно класифікованих об’єктів розпізнавання. Та оцінено її значення.

Перевірено однорідність патчів в БД які сформовані в результаті сегментації. Підтвердження того, що класи однорідні наведено у таблиці (табл. 4.1) тестування ЗНМ на сформованій БД.

Навчено ЗНМ на *7*-ми класах, зображено процес навчання, тестування ЗНМ на сформованій БД. В якості тестування якості процесу навчання підраховано матрицю помилок та правильних рішень, та відповідних статистик які можна отримати виходячи з цієї матриці. Проведено тестування ЗНМ на відеорядах які відрізняються від відео з яких було сформовано БД, тестування було проведено для кожного класу де було підраховано відсоток правильно класифікованих патчів з ЦЗ.

Наведено результат пошуку підозрілих не ідентифікованих об’єктів на відеоряді який не входить до БД. Зображено результат автоматичного наповнення БД.

ВИСНОВКИ

- Було проаналізовано основні проблеми розпізнавання. Наведено якісний опис завдання розпізнавання за допомогою побудови словника ознак, який використано під час побудови системи сегментації та класифікації різних типів місцевості. Описано постановку задачі системи розпізнавання та способи побудови системи розпізнавання. Наведено властивості системи розпізнавання та проведена класифікація систем розпізнавання.
- Проаналізовано метод ЗНМ, а в якості потужного класифікатора.
- Запропонована та навчена власна архітектура ЗНМ на підготовленій БД. Детально описано на мові словника ознак всі типи місцевості а саме:
  1. *Складено повний словник ознак.*
  2. *Проведено первісну класифікацію розпізнаваних об'єктів. На основі апріорної інформації виконано початкове розбиття об'єктів на класи, складено апріорний алфавіт класів T, K, F, та S.*
  3. *Розроблено апріорний словник ознак для кожного класу.*
  4. *Розбито апріорний просторів ознак на області, які відповідають класам апріорного алфавіту класів.*
  5. *Вибрано алгоритм розпізнавання (сегментації), що забезпечив віднесення об'єкта, що розпізнається до конкретного класу.*
  6. *Визначено робочий алфавіт класів та робочого словника ознак системи розпізнавання.*
- Описано в блок-схемах роботу автоматизованої системи, зображено архітектуру загорткової нейронної мережі, що була використана в якості класифікатора, зображено *UML* діаграми та діаграми варіантів використання АС, проведено опис інформаційної технології.
- Перевірено однорідність патчів в БД які сформовані в результаті сегментації. Підтвердження того, що класи однорідні наведено у таблиці (табл. 4.1) тестування ЗНМ на сформованій БД.

- Навчено ЗНМ на *7*-ми класах, зображено процес навчання, тестування ЗНМ на сформованій БД. В якості тестування якості процесу навчання підраховано матрицю помилок та правильних рішень, та відповідних статистик які можна отримати виходячи з цієї матриці. Проведено тестування ЗНМ на відеорядах які відрізняються від відео з яких було сформовано БД, тестування було проведено для кожного класу де було підраховано відсоток правильно класифікованих патчів з ЦЗ.
- Наведено результат пошуку підозрілих не ідентифікованих об’єктів на відеоряді який не входить до БД. Зображено результат автоматичного наповнення БД.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Хуршудов А.А. Обнаружение локальных пространственных структур для распознавания изображений / А.А. Хуршудов // Научно-технические ведомости СпбГУ. Информатика. Телекоммуникации. Управление.,» с. 72-82, 2014.
2. Wang S. Learning to Extract Parameterized Features by Predicting Transformed Images / S. Wang, с. 53, 2011.
3. Viola P. Rapid object detection using a boosted cascade of simple features / P. Viola, M. Jones // Computer Vision and Pattern Recognition,» с. 1-511, 2001.
4. Горелик А.Л., Скрипкин В.А. Методы роспознавания. М., «Высшая школа», 1977.
5. Кононюк А.Е. «ОБЩАЯ ТЕОРИЯ РАСПОЗНАВАНИЯ». - 2012.
6. Вентцель Е.С. Теория вероятностей. Физматгиз, 1964.
7. Митропольский А.К. Техника статистическиї вычислений. Физматгиз, 1961.
8. Горелик А.Л., Скрипкин В.А. Об одном методе решения задачи класификации обьектов или явлений. «Техническая кибернетика», 1965, №1.
9. Berger, James O. (1985). Statistical decision theory and Bayesian analysis. Berlin: Springer-Verlag.
10. Batenburg, K. J.; Sijbers, J. (2009-10-01). Adaptive thresholding of tomograms by projection distance minimization. Pattern Recognition 42 (10). с. 2297–2305.
11. Batenburg, K. J.; Sijbers, J. (2009-05-01). Optimal Threshold Selection for Tomogram Segmentation by Projection Distance Minimization. IEEE Transactions on Medical Imaging 28 (5). с. 676–686.

12. Mobahi, Hossein; Rao, Shankar R.; Yang, Allen Y.; Sastry, Shankar S.; Ma, Yi (2011-04-08). Segmentation of Natural Images by Texture and Boundary Compression. International Journal of Computer Vision (en) 95(1). c. 86–98.
13. Shankar Rao, Hossein Mobahi, Allen Yang, Shankar Sastry and Yi Ma Natural Image Segmentation with Adaptive Texture and Boundary Encoding, Proceedings of the Asian Conference on Computer Vision (ACCV) 2009, H. Zha, R.-i. Taniguchi, and S. Maybank (Eds.), Part I, LNCS 5994, pp. 135-146, Springer.
14. Linda G. Shapiro and George C. Stockman (2001): «Computer Vision», pp 279–325.
15. Ron Ohlander, Keith Price, and D. Raj Reddy (1978): «Picture Segmentation Using a Recursive Region Splitting Method», Computer Graphics and Image Processing, volume 8, pp 313–333.
16. Caselles, V.; Kimmel, R.; Sapiro, G. (1997). Geodesic active contours (PDF). International Journal of Computer Vision 22 (1): 61–79.
17. Jianbo Shi and Jitendra Malik (2000): «Normalized Cuts and Image Segmentation», IEEE Transactions on pattern analysis and machine intelligence, pp 888–905, Vol. 22, No. 8.
18. Leo Grady (2006): «Random Walks for Image Segmentation», IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1768–1783, Vol. 28, No. 11.
19. Z. Wu and R. Leahy (1993): «An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation», IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1101–1113, Vol. 15, No. 11.
20. Leo Grady and Eric L. Schwartz (2006): «Isoperimetric Graph Partitioning for Image Segmentation», IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 469–475, Vol. 28, No. 3.

21. C. T. Zahn (1971): «Graph-theoretical methods for detecting and describing gestalt clusters», IEEE Transactions on Computers, pp. 68-86, Vol. 20, No. 1.
22. Manisha Bhagwat, R. K. Krishna & Vivek Pise: «GSimplified Watershed Transformation», International Journal of Computer Science & Communication, Vol. 1, No. 1, January-June 2010, pp. 175–177.
23. Bailey, Ken (1994). Numerical Taxonomy and Cluster Analysis. Typologies and Taxonomies. c. 34.
24. Tryon, Robert C. (1939). Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality. Edwards Brothers.
25. Cattell, R. B. (1943). The description of personality: Basic traits resolved into clusters. Journal of Abnormal and Social Psychology 38 (4): 476–506.
26. Estivill-Castro, Vladimir (20 June 2002). Why so many clustering algorithms – A Position Paper. ACM SIGKDD Explorations Newsletter 4 (1): 65–75.
27. LeCun, Yann. LeNet-5, convolutional neural networks. Процитовано 16 November 2013.
28. Zhang, Wei (1988). Shift-invariant pattern recognition neural network and its optical architecture. Proceedings of annual conference of the Japan Society of Applied Physics.
29. Zhang, Wei (1990). Parallel distributed processing model with local space-invariant interconnections and its optical architecture. Applied Optics 29 (32) .
30. Habibi,, Aghdam, Hamed. Guide to convolutional neural networks : a practical application to traffic-sign detection and classification. Heravi, Elnaz Jahani,. Cham, Switzerland.
31. Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. Proceedings of the Twenty-Second

international joint conference on Artificial Intelligence-Volume Volume Two 2: 1237–1242.

32. Krizhevsky, Alex. ImageNet Classification with Deep Convolutional Neural Networks.