Київський Національний Університет імені Тараса Шевченка

Механіко-математичний факультет

Курсова робота
Студентки 1 курсу Магістратури
Спеціальності Математика

Томіної Маргарити Вадимівни

на тему

Методи глибокого навчання в задачах комп'ютерного бачення

Науковий керівник
доцент кафедри теорії ймовірностей, статистики
та актуарної математики

Ямненко Ростислав Євгенійович

28 травня 2022 р.

Зміст

1 Про машинне навчання

Ідея штучного інтелекту з'явилася в 1950-х, коли група ентузіастів з області інформатики, що тільки зароджується, задалися питанням, чи можна змусити комп'ютери «думати», — питанням, наслідки якого ми вивчаємо досі. Коротко цю область можна визначити так: автоматизація інтелектуальних завдань, які зазвичай виконуються людьми. Відповідно,ШІ - це область, що охоплює машинне навчання і глибоке навчання, а також включає багато підходів, не пов'язані з навчанням.

Наприклад, перші програми для гри в шахи діяли за жорстко визначеними правилами, заданими програмістами, і не могли кваліфікуватися як такі, що здійснюють машинне навчання. Довгий час багато експертів вважали, що ШІ рівня людини можна створити, якщо дати програмісту достатній набір явних правил для маніпулювання знаннями. Цей підхід, відомий як символічний ШІ, і був домінуючою парадигмою ШІ з 1950-х до кінця 1980-х. Пік його популярності припав на бум експертних систем у 1980-х. Символічний ШІ чудово справлявся з вирішенням чітко визначених логічних завдань, таких як гра в шахи, але, як виявилося, неможливо задати суворі правила для вирішення складніших, нечітких завдань, таких як класифікація зображень, розпізнавання мови та переклад іншими мовами.

На зміну символічному ШІ виник новий підхід: машинне навчання.

Область машинного навчання виникла з питання: чи може комп'ютер вийти за межі того, «що ми й самі знаємо, як виконувати», і самостійно навчитися вирішувати певне завдання? Чи може комп'ютер здивувати нас? Чи може комп'ютер без допомоги програміста, що задає правила обробки даних, автоматично визначити ці правила, досліджуючи дані? Це питання відкрило двері в нову парадигму програмування.

У класичному програмуванні, у парадигмі символічного ШІ, люди вводять правила (програму) та дані для обробки відповідно до цих правил та отримують відповіді. У машинному навчанні люди вводять дані та відповіді, що відповідають цим даним, а на виході одержують правила. Ці правила можна застосувати до нових даних для отримання оригінальних відповідей.

Розквіт машинного навчання розпочався лише у 1990-х, але ця область швидко перетворилася на найбільш популярний та успішний розділ ШІ, і ця тенденція була підкріплена появою більш швидкодіючої апаратури та величезних наборів даних. Машинне навчання тісно пов'язане з математичною статистикою, але має кілька важливих відмінностей. На відміну від статистики, машинне навчання зазвичай має справу з великими та складними наборами даних (наприклад, що складаються з мільйонів фотографій, кожна з яких складається з десятків тисяч пікселів), до яких практично неможливо застосувати класичні методи статистичного аналізу, такі як методи байесівські. Як результат, машинне і особливо глибоке навчання не мають потужної математичної платформи і ґрунтуються майже виключно на інженерних рішеннях. Це практична дисципліна, у якій ідеї найчастіше доводяться емпірично, а не теоретично.

2 Мета роботи

Основною метою моєї курсової роботи є практичне використання раніше теоретично освоїних знань, повязаних із використанням методів глибокого навчання в задачах комп'ютерного бачення.

Конкретно кажучи, моя робота полягає у наступному:

- створити особистий датасет із фото/картинок;

- навчити нейронну мережу на цьому датасеті розрізняти певні класи і досягти найкращих можливих результатів, використовуючи різні техніки та прийоми;

- дослідити, як впливає використання певних прийомів на точність результатів на моєму датасеті.

3 Технічні уточнення

У своїй роботі я буду використовувати мову програмування Python. Для роботи з нейронними мережами я використовуватиму відкриту бібліотеку KERAS, написану на мові Python.

Працювати я буду у хмарному сервісі Google Colab, адже він дає змогу безкоштовно використовувати процесори CPU, TPU , що дозволяє підвищити швидкість при обробці великих обсягів обчислювальних задач.

4 Алгоритм дій

Узагальнений процес вирішення задач машинного навчання включає в себе наступні кроки:

1. Визначення задачі та створення набору данних;
2. Вибір міри успіху;
3. Вибір протоколу оцінки;
4. Попередня підготовка данних;
5. Розробка моделі досконалішої, аніж у базовому випадку з проблемою перенавчання;
6. Регуляризація моделі та налаштування гіперпараметрів.

4.1 Визначення задачі та створення набору данних

Насамперед у цьому пункті, необхідно визначити завдання, а для цього відповісти на наступні запитання. Який вигляд матимуть вхідні дані? Що потрібно передбачити? До якого типу належить завдання, що стоїть перед нами?

Також потрібно пам'ятати, що ми ставимо наступні гіпотези:

- гіпотеза про те, що вихідні дані можна передбачити за вхідними даними;
- гіпотеза про те, що доступні дані досить інформативні для вивчення відносин між вхідними та вихідними даними.

Я вирішила зробити датасет з фотографіями людських рук на світлому фоні, які показують певну кількість пальців: 1, 2, 3, 4 або 5 і натренувати нейронну мережу, щоб вона навчилась розрізняти, скільки пальців показує рука на фотографії. Фотографії здебільшого були зібрані з інтернету або зроблені мною та моїми знайомими.

Мій набір данних HAND складається з двох папок TRAIN та TEST, кожна з яких містить 5 класів: one, two, three, four, five; у кожному класі по 280 зображень, які розбиті на тренувальні та тестові - 80% і 20% від усіх зображень відповідно; усі фото мають назву 'class.num' і зображають певні положення руки. Тобто, датасет HAND має наступний вигляд:

HAND/
- TRAIN/
  - one/
    - one.0
    - one.1
    - ...
  - two/
    - two.0
    - ...
  - ...
  - five/
    - five.0
    - ...
- /TEST
  - one/
    - one.0
    - ...
  - ...
  - five/
    - five.0
    - ...

Потрібно відмітити, що k пальців можна показати C_5^k способами кожною рукою, але

я брала лише ті конфігурації правої і лівої руки, які Ви бачите на малюнках вище. Це пов'язано з тим, що мій набір данних є дуже маленьким. І якщо навчити нейронну мережу розрізняти якийсь один жест, що показує, наприклад 3 пальці, на такому невеликому датасеті з відносно великою точністю можливо, то задача розрізнити 10 різних жестів, що показують 3 пальці зі збереженням розміру набору данних, звучить майже неможливо.
Отже, відповідаючи на поставлені на початку питання, кажемо, що наші вхідно данні - це RGB зображення положень руки на білому фоні. Задача нейронної мережі - визначити кількість пальців, які показує рука. Завдання належить до однозначної багатокласової класифікації.
Отож, стискаємо наш набір данних та переходимо до програмування.

4.2 Вибір міри успіху

Щоб тримати ситуацію під контролем, потрібно мати можливість спостерігати за нею. Для досягенння успіху, потрібно зрозуміти, що розуміється під успіхом. Показник успіху буде визначати вібір функції втрат - що саме повинна оптимізувати наша модель.
Функція втрат, яку я обрала для даної задачі має назву sparse_categorical_crossentropy (або логарифмічна функція втрат - log loss). Крос-ентропія вимірює розбіжність між двома імовірнісними розподілами. Якщо крос-ентропія велика, це означає, що різниця між двома розподілами велика, а якщо крос-ентропія мала, то розподіли схожі один на одного.
Крос-ентропія визначається як:

$$H(P,Q) = -\sum_{x} P(x)log(Q(x))$$

де P - розподіл істинних відповідей, Q - розподіл ймовірностей прогнозів моделі. Наприклад, для бінарної класифікації формула крос-ентропії матиме наступний вигляд:

$$CE = -\sum_{i}(y_i log(p_i) + (1-y_i)log(1-p_i))$$

де y - двійковий ідентифікатор (0 або 1) того, чи є мітка певного класу правильною класифікацією для даного спостереження, а p - прогнозована ймовірність моделі, а i - номер об'єкту.
При двійковій класифікації кожна передбачена ймовірність порівнюється з фактичним значенням класу (0 або 1) і обчислюється оцінка, яка штрафує ймовірність на основі відстані від очікуваного значення.
В нашому випадку ми маємо багатокласову класифікацію, тому ми будемо брати суму значень логарифмічних функцій втрат для кожного прогнозу класів(а їх в нас 5), що спостерігаються.

$$CE = -\frac{1}{q}\sum_{i=1}^{q}\sum_{j=1}^{l} y_{ij} log(p_{ij})$$

де q - кількість елементів у вибірці, l - число класів.
Дана функція втрат sparse_categorical_crossentropy є оптимальною при використанні задач багатокласової класифікації з мітками - цілими числами.

4.3 Вибір протоколу оцінки

Після визначення цілі, потрібно також зазначити, як вимірювати рух до неї. Існує, якнайменьше, три основних протоколи оцінки:

- виділення з усієї вибірки окремий перевірочний набір данних;
- перехресна перевірка по K блокам;
- ітераціна перевірка по K блокам з перемішуванням

Не дивлячись на те, що в мене достатньо малий набір данних і для таких випадків рекомендують використовувати 2 і 3 протоколи оцінки, для початку я використаю саме виділення з усієї вибірки окремий перевірочний набір данних, адже в більшості випадків він дає можливість отримати достатньо надійну оцінку.

4.4 Попередня підготовка данних

Перед передачею у мережу данні мають бути перетворені у тензори з дійсними числами в діапазоні [0,1]. У данний момент ми маємо файли JPG, тому їх потрібно підготувати до передачі у нейронну мережу, виконавши наступні кроки:

1. Прочитати файли із зображеннями;
2. Декодувати зміст зображень і перетворити їх у тензори з дійсними числами;
3. Масштабувати значення пікселів із діапазону [0,255] в діапазон [0,1].

Завантажуємо файл у Google Colab та використовуємо наступну команду для його розпакування.

```
!unzip HAND.zip
```

У фреймворці KERAS є інструменти, які дозволяють ці кроки робити автоматично. Наприклад, модуль keras.preprocessing.image з інструментами обробки зображень, зокрема клас ImageDataGenerator в ньому, який дозволить швидко налаштувати генератори Python для автоматичного перетворення файлів із зображеннями у пакети готових тензорів.

Створюємо тренувальний та перевірочний набори данних, попередньо домноживши коефіцієнти зображень на 1/255 для встановлення значень між 0-1. Це полегшить роботу нейронній мережі та зменшить обчислення. Також, використовуючи команду validation_split, "розбиваємо"данні на два набори: 80% даних - тренувальний та 20% - перевірочний. Оскільки нейронні мережі приймають тензори виключно однакових розмірів, також змінюємо розміри усіх зображень на розміри, вказані в параметрі target_size. Параметр class_mode відповідає за перетворення наших міток у цілі числа. В нашому випадку мітки перетворяться в 1D - вектор міток.

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
      rescale = 1./255,
      validation_split = 0.2)

train_dataset = train_datagen.flow_from_directory(
      '/content/HAND/TRAIN',
      target_size=(150, 150),
      batch_size=32,
      class_mode='binary',
      subset = 'training')

 validation_datagen = ImageDataGenerator(
      rescale=1./255,
      validation_split = 0.2)

 validation_dataset = validation_datagen.flow_from_directory(
      '/content/HAND/TRAIN',
       target_size=(150, 150),
       batch_size=32,
       class_mode='binary',
       subset = 'validation')
```

Аналогічно створюємо набір данних для тестування. Створюємо тренувальний набір данних train_dataset , взявши зображення із '/content/HAND/TRAIN'.

```
test_datagen = ImageDataGenerator(rescale=1./255)

test_dataset = test_datagen.flow_from_directory(
    '/content/HAND/TEST',
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

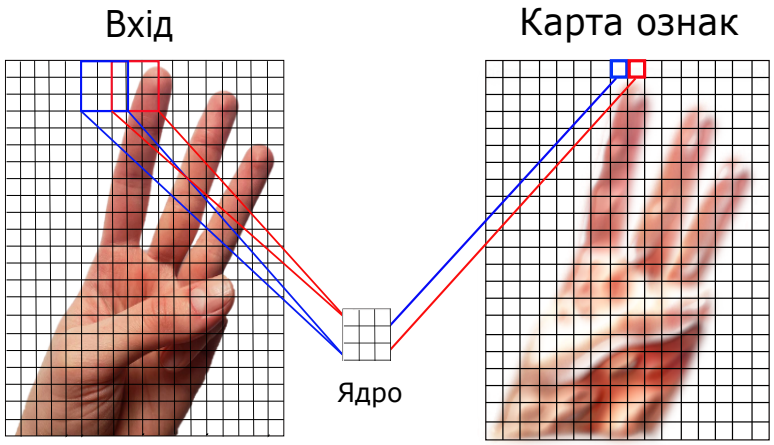
4.5 Розробка моделі досконалішої, аніж у базовому випадку з проблемою перенавчання

Після підготовки данних ми приступаємо до побудови нейронної мережі. Для того, щоб створити першу робочу модель, яка буде надалі вдосконалюватися, ми маємо створити таку, яка видавала б результат кращий, аніж для "базового"випадку. Базовий випадок в данному формулюванні - це випадковий вибір(тобто 20% для нашого випадку).
Ідеальна модель - це модель між перенавчанням та недонавчанням. Щоб знайти цю грань, потрібно створити таку модель, яка перенавчиться для подальшої регуляризації та її налаштування.
Щоб зробити перенавчену модель, потрібно додати більше шарів, зробити великою кількість параметрів у шарах та навчати модель на великій кількості епох.
Для задачі багатокласової класифікації я буду використовувати згорткову архітектуру нейронних мереж, які показують дуже хороші результати у задачах для розпізнавання образів.
Американський вчений французького походження, Ян ЛеКун, натхненний роботами нобелевских лауреатів в області медицини перший запропонував використовувати згорткові нейронні мережі. Ідея згорткових нейронних мереж полягає в чергуванні згорткових шарів (C-layers), субдискретизуючих шарів (S-layers) та наявності повнозв'язних (F-layers) шарів на виході.
Така архітектура містить у собі 3 основні парадигми:

- Локальне сприйняття;
- Роздільні вагові коефіцієнти;
- Субдискретизація.

Локальне сприйняття передбачає, що у вхід одного нейрона подається в повному обсязі зображення (чи виходи попереднього шару), лише деяка його область. Такий підхід дозволив зберігати топологію зображення від шару до шару. Концепція вагових коефіцієнтів, що розділяються, передбачає, що для великої кількості зв'язків використовується дуже невеликий набір вагових коефіцієнтів. Тобто. якщо у нас є на вході зображення розмірами 32x32 пікселя, то кожен з нейронів наступного шару прийме на вхід тільки невелику ділянку цього зображення розміром, наприклад, 5x5, причому кожен із фрагментів буде оброблений одним і тим самим набором. Важливо розуміти, що самих наборів вагів може бути багато, але кожен з них буде застосований до всього зображення. Такі набори часто називають ядрами.
А як же це позначиться на розпізнаванні образів ? Як не дивно, на краще. Справа в тому, що таке штучно введене обмеження на вагові коефіцієнти покращує узагальнюючі властивості мережі (generalization), що в результаті позитивно позначається на можливості мережі знаходити інваріанти у зображенні та реагувати головним чином на них, не звертаючи уваги на інший шум. Можна подивитися на цей підхід з іншого боку. Ті, хто займався класикою розпізнавання зображень і знає,

як це працює на практиці (наприклад у військовій техніці) знають, що більшість таких систем будуються на основі двовимірних фільтрів. Фільтр є матрицею коефіцієнтів, зазвичай задану вручну. Ця матриця застосовується до зображення за допомогою математичної операції, яка називається згорткою. Суть цієї операції в тому, що кожен фрагмент зображення множиться на матрицю (ядро) згортки поелементно і результат підсумовується та записується в аналогічну позицію вихідного зображення. Основна властивість таких фільтрів полягає в тому, що значення їх виходу тим більше, чим більше фрагмент зображення схожий на сам фільтр. Таким чином зображення згорнуте з якимсь ядром дасть нам інше зображення, кожен піксел якого означатиме ступінь схожості фрагмента зображення на фільтр. Іншими словами, це буде карта ознак. Процес поширення сигналу в С-шарі я спробувала зобразити на малюнку:



Кожен фрагмент зображення поелементно домножається на невелику матрицю вагових коефіцієнтів (ядро), результат сумується. Ця сума є пікселем вихідного зображення, яке називається картою ознак. Ще одна особливість згорткових нейронних мереж у тому, що вони трохи зменшують зображення за рахунок крайових ефектів.
Суть субдескритизації S - шарів заключається у зменшенні просторової розмірності зображення, тобто зображення грубо (усередненням) зменшується у задану кількість разів. Субдескритизація потрібна для забеспечення інваріантності до масштабу. Послідовність шарів дозволяє робити карти ознак з карт ознак, що на практиці означає можливість розпізнавання складних ієрархій ознак.
Згортки визначаються двома ключовими параметрами:

- Розмір шаблонів, що витягуються з вхідних даних, зазвичай 3×3 або 5×5.
- Глибина вихідної карти ознак - кількість фільтрів, що обчислюються згорткою.

Код нижче - наша початкова модель. Реалізуємо загальну структуру: згорткова

нейронна мережа буде організована як стек з 4 шарів Conv2D і MaxPooling2D. Використаємо функцією активації ReLu.

```
from keras import layers
from keras import models

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
                                    input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

Наступний крок - передача останнього вихідного тензора на вхід повнозв'язного класифікатора: стека шарів Dense. Ці класифікатори обробляють вектори-одномірні масиви, тоді як поточний вихід є тривимірним тензором. Тому ми повинні перш за все перетворити тривимірний вивід в одномірний шаром Flatten і потім додати зверху кілька шарів Dense.
Так як перед нами стоїть задача багатокласової класифікації, модель закінчується шаром Dense з розміром 5 (що рівне кількості класів) та функцією активації sofmax.

```
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dense(5, activation='softmax'))
```

Компілюємо модель. Для цього нам потрібно налаштувати ще три параметри: функція втрат, оптимізатор та метрики для моніторингу на етапах навчання.
Як вже було зазначено раніше, функція втрат sparse_categorical_crossentropy, яка визначає, як мережа повинна оцінювати якість своєї роботи на навчальних даних та, відповідно, як коригувати її у правильному напрямку обрана нами, адже вона є підходящою для багатокласової однозначної класифікації і виводить індекс категорії з найбільшою відповідністю.
Оптимізатор - механізм, за допомогою якого мережа буде оновлювати себе, спираючись на дані та функцію втрат.

Метод оптимізації Adam був представлений Дідерік Кінгма від OpenAI та Джиммі Ба з Університету Торонто у їхній 2015 року статті під назвою «Адам: метод стохастичної оптимізації». Adam — один із найефективніших алгоритмів оптимізації у навчанні нейронних мереж. Він поєднує в собі ідеї RMSProp та оптимізатора імпульсу. Адам відрізняється від класичного стохастичного градієнтного спуску. Стохастичний градієнтний спуск підтримує єдину швидкість навчання (звану альфа) для всіх оновлень вагових коефіцієнтів і швидкість навчання не змінюється під час тренування. Швидкість навчання в Adam підтримується для кожного вагового коефіцієнту мережі (параметра) і окремо адаптується по мірі розвитку навчання. Метод обчислює індивідуальні адаптивні швидкості навчання для різних параметрів з оцінок першого та другого моментів градієнтів.
Його переваги:

- Проста реалізація.
- Обчислювальна ефективність.
- Невеликі вимоги до пам'яті.
- Інваріант до діагонального масштабування градієнтів.
- Добре підходить для великих задач з точки зору даних і параметрів.
- Підходить для нестаціонарних цілей.
- Підходить для задач з дуже "шумними"або розрідженими градієнтами.
- Гіперпараметри мають наочну інтерпретацію і зазвичай потребують невеликих налаштувань.

Метрики для моніторингу на етапах навчання та тестування ми оберемо ті, які найчастіше використовуються ("acc") і подивимось на результат.

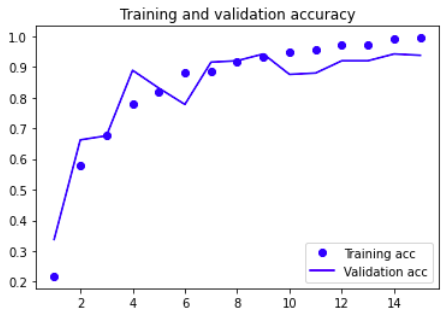
```
model.compile(loss = "sparse_categorical_crossentropy",
                        optimizer = 'adam',
                        metrics = ["acc"])
```

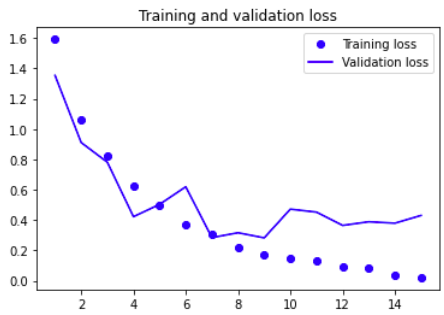
Навчаємо модель, передавши їй дані для тренування та перевірки.

```
history = model.fit_generator(train_dataset,
                                    validation_data = validation_dataset,
                                    epochs = 15)
```

Та будуємо графічне зображення наших результатів, використовуючи пакет matplotlib.

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Добре видно, що на 8 кроці навчання наша модель перенавчилася і почала виділяти специфічні шаблони, які не є релевантними у загальному сенсі. Іншими словами, модель починає вивчати шаблони, характерні виключно для тренувальних данних, але не характерні для нових, раніше не "бачених".
Таким чином, дана модель дає точність 86.79% на перевірочних данних.

4.6 Регуляризація моделі та налаштування гіперпараметрів.

Цей етап займає найбільшу кількість часу, адже потрібно багаторазово змінювати модель, навчати, оцінювати якість на перевірочних данних, знову змінювати і повторювати цей цикл до досягнення точності, яка буде влаштовувати.

4.6.1 Розширення данних

Найкращий спосіб запобігти вивченню моделлю специфічних або нерелевантних шаблонів, що мають місце в тренувальних даних, - збільшити обсяг тренувальних даних. Модель, навчена на більшому обсязі даних матиме більшу узагальненість.

Спробуємо запобігти перенавчанню, використовуючи такий прийом, як розширення данних.
Причиною перенавчання є недостатня кількість зразків для навчання моделі, здатної узагальнювати нові дані. Маючи нескінченний обсяг даних, можна було б отримати модель, що враховує всі аспекти розподілу даних: ефект перенавчання ніколи не настав би. Прийом розширення даних (data augmentation) реалізує підхід створення додаткових навчальних даних з наявних шляхом трансформації зразків безліччю випадкових перетворень, що дають нові правдоподібні зображення. Мета полягає в тому, щоб на етапі навчання модель ніколи не побачила те саме зображення двічі. Це допоможе моделі виявити більше особливостей даних та досягти кращого ступеня узагальнення.
Отже, позитивні сторони використання прийому розширення данних:

- Покращує продуктивність моделей ML. Розширені дані покращують продуктивність і результати моделей глибокого навчання, створюючи нові та різноманітні екземпляри для наборів навчальних даних.
- Зменшує експлуатаційні витрати, пов'язані зі збором даних. Збір даних і маркування можуть бути трудомісткими та дорогими процесами для моделей глибокого навчання. Компанії можуть скоротити операційні витрати, трансформуючи набори даних за допомогою методів збільшення даних.

Звичайно, цей метод також має свої проблеми, зокрема:

- Вартість забезпечення якості розширених наборів даних.
- Дослідження та розробки для створення синтетичних даних із передовими додатками.
- Перевірка методів збільшення зображення, таких як GAN, є складною задачею.
- Знайти оптимальну стратегію збільшення даних нетривіально.
- Властива вихідним даним упередженість зберігається в доповнених даних.

У KERAS це може бути зроблене шляхом використання вищезгаданного класу ImageDataGenerator. Він також дозволяє налаштувати випадкові перетворення та операції нормалізації, які будуть виконуватися з даними зображення під час навчання.
Застосуємо прийом розширення данних тільки до тренувальних данних, адже для перевірочних та тестових у цьому немає необхідності.

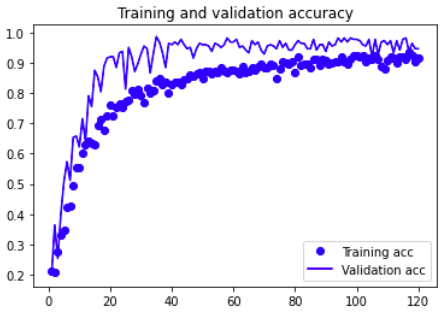
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
        rescale=1./255,
```

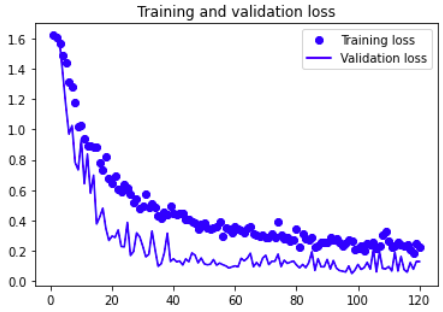
```
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split = 0.2,)

train_dataset = train_datagen.flow_from_directory(
        '/content/HAND/TRAIN',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary',
        subset = 'training')
```

Пояснимо, що означає кожен рядок у команді ImageDataGenerator:

- rescale значення, на яке помножиться дата. В нашому випадку зображення складаються з RGB коефіцієнтів 0-255. Оскільки такі числа є дуже великими та потенційно ускладнять роботу нейронним мережам, множимо на 1/255 щоб встановити значення між 0-1;
- rotation_range у градусах(0-180) значення, у діапазоні якого ми повертаємо зображення;
- width_shift_range та height_shift_range це діапазони (як частка загальної ширини або висоти), у межах яких можна випадковим чином переміщати зображення по вертикалі або горизонталі;
- shear_range призначений для випадкового застосування перетворень зсуву;
- zoom_range довільне масштабування всередині зображень;
- horizontal_flip призначений для випадкового перегортання половини зображень по горизонталі;
- validation_split розбиває дані з папки на тренувальний та перевірочний набори данних у співвідношенні 0.9 до 0.1 відповідно.

Також збільшуємо кількість епох до 120.
Також після побудови графіка, бачимо, що явного перенавчання моделі вже не спостерігається.
Отже, прийом Data augmentation є дуже дієвим для боротьби з перенавчанням нейронної мережі при відсутності великої кількості зображень для навчання. Він дозволяє нейронній мережі навчатись на зовсім різних данних шляхом розтягувань, поворотів, переміщенню зображень і не бачити одного й того ж зображення двічі, що запобігає виділенню нерелевантних шаблонів.





4.6.2 Додавання проріджування

Проріджування (dropout) - один з найефективніших і найпоширеніших прийомів регуляризації для нейронних мереж, розроблений Джеффом Хінтоном (Geoff Hinton) та його студентами в Університеті Торонто. Проріджування, яке застосовується до шару, полягає в привласненні нуля ознакам, що випадково вибираються, на етапі навчання.
Цей прийом може здатися дивним та необґрунтованим. Як він допоможе впоратися з перенавчанням? За словами Хінтона, основою для цього прийому, крім іншого, став механізм, який використовується банками для запобігання шахрайству. Ось його слова: «Відвідуючи свій банк, я помітив, що операційні, які обслуговують мене, часто змінюються. Я спитав одного з них, чому так відбувається. Він сказав, що не знає, але часто доводиться переходити з місця на місце. Я припустив, що це робиться для виключення шахрайської змови клієнта зі співробітником банку. Це навело мене на думку, що видалення випадково обраної підмножини нейронів з кожного прикладу може допомогти запобігти змові моделі з вихідними даними і тим самим послабити ефект перенавчання». Основна ідея полягає в тому, що введення шуму у вихідні значення може розбити випадково складені шаблони, що не мають великого значення, які модель починає запам'ятовувати при відсутності шуму. У Keras додати проріджування в мережу можна за допомогою рівня Dropout, який обробляє результати роботи шару безпосередньо перед ним.
Отож, повнозв'язний класифікатор перепишемо так:

```
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))

model.add(layers.Dense(5, activation='softmax'))
```

Після додавання розширення данних та проріджування, модель показала точність 93.93% на незнайомих перевірочних данних.

4.6.3 Інші експерименти

Є ще кілька прийомів, які я випробувала на своїй задачі для покращення результатів, але вони не дали позитивного результату. Це такі методи, як:

- L1 та L2 регуляризація;
- зміна архітектури(додавання та прибирання шарів);
- зміна гіперпараметрів.

Можливо, ви знайомі з принципом бритви Оккама: якщо якомусь явищу можна дати два пояснення, правильним, швидше за все, буде простіше, що має меншу кількість припущень. Ця ідея застосовна також до моделей, отриманих за допомогою нейронних мереж: для тих самих наборів тренувальних даних та архітектури мережі існує безліч наборів вагових значень (моделей), що пояснюють дані. Простіші моделі менш схильні до перенавчання, ніж складні. Проста модель в даному контексті - це модель, в якій розподіл значень параметрів має меншу ентропію (або модель з меншою кількістю параметрів). Тобто типовий спосіб пом'якшення проблеми перенавчання полягає у зменшенні складності мережі шляхом обмеження значень її вагових коефіцієнтів, що робить їх розподіл більш рівномірним. Цей прийом називається регуляризацією ваг, він реалізується додаванням у функцію втрат нейронної мережі штрафу за збільшення значень вагових коефіцієнтів і має два різновиди:

- L1-регуляризація (L1 regularization) - штраф, що додається, прямо пропорційний абсолютним значенням вагових коефіцієнтів (L1-норма вагових коефіцієнтів).
- L2-регуляризація (L2 regularization) - штраф, що додається, пропорційний квадратам значень вагових коефіцієнтів (L2-норма вагових коефіцієнтів). У контексті нейронних мереж L2 регуляризація також називається скороченням вагів (weight decay). Це дві різні назви одного і того ж явища: скорочення вагів з математичної точки зору суть те ж саме, що L2-регуляризація.

При використанні L1 або L2 регуляризації моя модель застрягала і показувала результати приблизно рівні базовому випадку.
Обрана архітектура, а саме стек з 4 шарів згортки, та гіперпараметри показали свою оптимальність у багаторазових дослідах.

5 Висновок

Отож, ми досягли поставлених на початку роботи цілей, а саме створили свій датасет, навчили нейронну мережу на цьому наборі данних розрізняти різні класи зображень та покращили точність моделі на небачених раніше данних майже до 94%.

Ми дослідили, як вплинули різні прийоми боротьби з перенавчанням і покращенням результатів точності нейронної мережі і прийшли до наступних висновків.
Згорткові нейронні мережі - дійсно дуже ефективна архітектура для розпізнавання образів. Але в той же час при невеликому наборі данних сама по собі вона дуже швидко перенавчається та починає виділяти нерелевантні шаблони. В таких випадках на допомогу приходять різні прийоми боротьби з перенавчанням. В моєму випадку дуже гарно спрацював прийом розширення данних, який майже одразу "поборовся"з перенавчанням та проріджування, який покращив результат майже до 94%.
В той же час такі прийоми, як L1, L2, L1-L2 - регуляризація в моєму випадку зовсім не підійшли.

Використані матеріали

1. Francois Chollet. Deep Learning with Python, Second Edition (2018).
2. Офіційна документація KERAS

   https://keras.io/
3. Tutorial on using Keras flow_from_directory and generators

   https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720
4. Yann LeCun, J. S. Denker, S. Solla, R. E. Howard and L. D. Jackel: Optimal Brain Damage, in Touretzky, David (Eds), Advances in Neural Information Processing Systems 2 (NIPS*89), Morgan Kaufman, Denver, CO, 1990
5. Y. LeCun and Y. Bengio: Convolutional Networks for Images, Speech, and Time-Series, in Arbib, M. A. (Eds), The Handbook of Brain Theory and Neural Networks, MIT Press, 1995
6. Daniel Godoy. Understanding binary cross-entropy / log loss: a visual explanation

   https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
7. Офіційна документація Tensor Flow

   https://www.tensorflow.org/
8. Francois Chollet. Tutorials(2016). Building powerful image classification models using very little data

   https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
9. Jason Brownlee. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning

   https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
10. Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019).

    https://doi.org/10.1186/s40537-019-0197-0