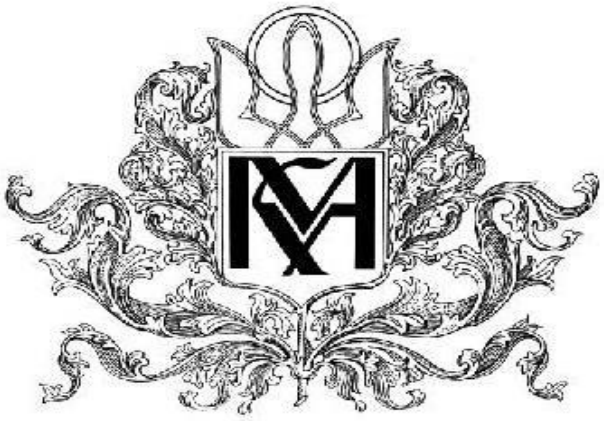
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики



# Методи обрізки нейронних мереж

**Текстова частина до курсової роботи**
**за спеціальністю „Комп'ютерні науки" - 122**

**Керівник курсової роботи**
Кандидат фізико-математичних наук,
старший викладач Швай Н. О.

_____
(підпис)
"____" _____ 2021 р.

**Виконав студент КН-4**
Дубчак О. Б.
"____" _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

Доцент., к. ф.-м. н. С.С.Гороховський

(підпис)

„____”_____2021 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на курсову роботу

студенту Дубчаку Олександру Богдановичу факультету інформатики 4-го курсу

ТЕМА Методи обрізки нейронних мереж

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Вступ
4. Дослідження алгоритмів обрізки нейронних мереж
5. Реалізовані алгоритми
6. Висновки
7. Список літератури

Дата видачі „___” _____ 2021 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання роботи:

**Тема: Методи обрізки нейронних мереж**

| № | Назва етапу | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Отримання теми курсової | 28.10.2020 | |
| 2. | Пошук тематичної наукової літератури | 16.11.2020 | |
| 3. | Ознайомлення з науковою літературою | 11.01.2021 | |
| 4. | Створення і тренування моделей | 16.02.2021 | |
| 5. | Реалізація алгоритму l1 норми | 25.02.2021 | |
| 6. | Реалізація алгоритму функціонального обрізання | 19.03.2021 | |
| 7. | Написання теоретичної частини | 29.03.2021 | |
| 8. | Створення презентації | 1.04.2021 | |
| 9. | Внесення змін до роботи | 10.04.2021 | |
| 10. | Перевірка роботи на плагіат | 12.04.2021 | |
| 11. | Захист роботи | 18.04.2021 | |

Зміст

# Вступ

Останнім часом глибокі нейронні мережі зробили величезний крок вперед у напрямку розв'язання задач класифікації, регресії, знаходження об'єктів і порівняння. Вони надають можливості фактично замінювати людей у тих сферах, які раніше не були притаманні комп'ютерним програмам. Проте така потужність не дається просто так. Для того, щоб досягти таких високих результатів, модель нейронної мережі повинна бути натренована на великій кількості інформації, а також під час передбачення нейронною мережею відбувається надзвичайно велика кількість обчислень. Це може спричиняти уповільнення роботи моделі, або взагалі нездатність моделі до передбачення у адекватно відведений для цього час. Для збереження великої кількості параметрів(мільйонів і мільярдів одиниць інформації) використовується велика кількість пам'яті, що ,наприклад, є критичним для мобільних телефонів.

Дослідниками було запропоновано велика кількість архітектурних рішень нейронних мереж таких як повноз'єднані, згорткові, рекурентні, мережі Хопфілда, та багато інших. Зазвичай нейронні мережі з великою кількістю прихованих шарів можуть швидко навчатись та обходити локальні мінімуми під час навчання, оскільки з ростом кількості параметрів зростає і швидкість обходу локальних мінімумів. Проте, якщо модель виростає занадто сильно, це означатиме, що більше фільтрів, більше вузлів, більше обчислювальних потужностей будуть використані без необхідності. Також такі моделі зазвичай більш схильні до перетренування, коли модель показує відмінні результати на тренувальній вибірці і втрачає точність на тестових даних. Здатністю нейронної мережі до сприйняття невідомих даних називають генералізацією. Чим краще модель справляється з невідомими даними, тим більша її здатність до генералізації. Найкращу здатність до генералізації показують саме невеликі моделі, адже вони менш вразливі до перетренування. Проте їхнє тренування може зайняти деякий час, адже у таких моделей може бути не достатньо параметрів для обробки даних. Таким чином, оптимальна архітектура нейронної мережі – це та, яка достатньо велика для повного вивчення задачі, та достатньо мала для того, щоб не втратити здатність до генералізації. У цій роботі були

досліджено методи обрізання нейронних мереж, за допомогою яких можна знизити кількість параметрів нейронної мережі без значного падіння точності моделі.

# 1 Опис існуючих методів та підходів для зменшення розмірів моделей нейронних мереж

Серед існуючих методів обрізання нейронних мереж виділяють наступні типи: методи, засновані на штрафуванні значень, кросвалідаційні методи, методи, засновані на величинах, методи, засновані на важливості параметрів та інші.

Мета методів, заснованих на штрафуванні значень, полягає у тому, щоб створити штрафуючу функцію, яку надалі мінімізують таким чином, щоб ваги з меншими значеннями поступово наближались до нуля. У методі запропонованому в звіті «Structure Learning by Pruning in Independent Component Analysis» [1] було визначення штрафуючої функції, яка обмежує використання непотрібних зв’язків і не допускає того, щоб значення параметрів були надто великикими. У цьому документі також було запропоновано алгоритм видалення надлишкових параметрів для мінімізації штрафуючої функції, проте цей підхід може також видаляти і важливі для моделі параметри.

У методах крос-валідації критерії обрізання також залежать від значень ваг, проте додано додатковий крок у вигляді валідації обрізаної моделі. Дані поділяються на тренувальну і валідаційну вибірку. На кожному етапі обрізання валідаційна вибірка дозволяє зрозуміти точність обрізаної моделі. Якщо модифікована модель показує себе краще за початкову, то результати обрізання приймаються, і процес може бути продовжено. Інакше модель повертається до стану на попередньому кроці.

Методи, засновані на величинах(Magnitude based pruning) припускають, що малі значення ваг є неважливими. У роботі [2] запропоновано три параметри, які називаються сила параметрів ваг, витрачена енергія, та фактор корисності. Ці метрики є досить простими для реалізації і вимагають небагато обчислювальних

потужностей. Проте такі методи часто видаляють важливі частини моделі. Одним з прикладів алгоритмів, заснованих на величинах є алгоритм [3] обрізання нейронних мереж згідно з нормами ядер фільтрів, який був реалізований у цій роботі.

Методи, засновані на важливості параметрів обчислюють значення важливості завдяки процесу отримання вихідних даних з кожного шару моделі при заданих вхідних даних. Один із таких алгоритмів був реалізований у розділі 2.3.2

# 2 Реалізація алгоритмів обрізки нейронних мереж

## 2.1 Нейронні мережі, на яких проводились досліди

Згорткові нейронні мережі є одними з найпопулярніших типів нейронних мереж. Головна їхня ідея полягає у тому, що для повного розуміння зображення, чи будь-якої іншої неперервної інформації не достатньо «близького» погляду на речі – потрібен також загальний вигляд об’єкта. Часто такі нейронні мережі використовують для задач класифікації, хоча вони також можуть вирішувати задачі знаходження об’єкта, задачі порівняння та багато інших. Ефективність згорткових нейронних мереж показала великий успіх у вирішенні цих задач не тільки в наукових експериментах, а й в завданнях для бізнесу. У даній роботі було проведено дослідження методів обрізки згорткових нейронних мереж. Для цього було обрано моделі AlexNet [4], VGG16 [5], та створена спеціально для тестів модель SimpleBirdsNet. Всі експерименти проводилися на датасеті birds-200 [6].

### 2.1.1 Модель AlexNet

Модель AlexNet була представлена у 2012 році Алексом Крижевським. Вона вважається однією з основоположників архітектур сучасних згорткових нейронних мереж. і отримала призове місце на змаганнях ImageNet LSVRC-2012, де розробники створювали моделі для розпізнавання класів у дата сеті ImageNet. [7], який складається з 1.2 мільйонів тренувальних зображень і 100 тисяч тестових.

Модель складається з п'яти згорткових, та трьох повноз'єднаних шарів. Загальна кількість параметрів – 62 мільйони.

### 2.1.2 Модель VGG-16

Модель VGG-16 можна вважати правонаступницею моделі AlexNet. Вона була представлена у 2014 році за змаганнях ImageNet LSVRC-2014. І досі, ця архітектура широко застосовуються у вирішенні задач класифікації. Однією з нововведень моделі було фокусування на більшій кількості згорткових шарів, аніж на повноз'єднаних шарах. Модель складається з 16 шарів, 13 з яких є згортковими, та 3 повноз'єднані шари. Кількість параметрів складає 134 мільйони. Через це її досить довго тренувати, а також через це можна показати наскільки така велика кількість параметрів дійсно потрібна для задач розпізнавання, через процес обрізання моделі.

### 2.1.3 Модель SimpleBirdsNet

Для отримання інформації про те, як себе поводитимуть моделі з низькою кількість параметрів було вирішено створити модель SimpleBirdsNet. Модель складається з семи блоків та трьох повно з'єднаних шарів. Кожен блок складається з згорткового шару, шару нормалізації, та максимізаційного агрегувального шару. Архітектуру запропонованої моделі можна детальніше роздивитись у додатку 1.Загальна кількість параметрів складає 5 мільйонів. На тестовому датасеті модель показала точність у 88%, що є хорошим показником для моделі з такою низькою кількістю параметрів

Модель було треновано на датасеті birds200 [6]. Датасет складається з 11000 зображень 200 видів птахів.

## 2.2 Методи квантування нейронних мереж

Квантування нейронних мереж – це техніка оптимізації нейронних мереж, згідно якої для зберігання параметрів використовуються цілі числа замість чисел з плаваючою точкою. Квантована нейронна мережа виконує операції з цілими числами, що дозволяє створювати більш компактні копії моделі. Також варто зазначити, що використання векторизованих операцій на цілих числах є більш ефективним на багатьох обчислювальних пристроях. Таким чином цей підхід

дозволяє знизити і розмір, який займає модель у пам’яті, і вартість обчислень, проте знизивши кінцеву точність моделі.

Є два підходи до квантування нейронних мереж – квантування після тренування і квантуванні під час тренування.

Квантування після тренування виконується після того, як було досягнуто оптимальної точності нейронної мережі. Одним з найпростіших методів у цьому підході є квантування на динамічному діапазоні(Dynamic range quantization). Згідно з ним, тільки параметри ваг підлягають квантуванню з типів плаваючої точки до восьми бітних цілочисельних типів. Проте для того, щоб повністю перейти до цілочисельних операцій, потрібно також перетворити параметри входу та параметри виходу моделі на цілочисельні типи. Бібліотека Tensorflow надає такі можливості [8].

Другим підходом є квантування під час навчання нейронної мережі. Такий підхід зазвичай залишає кращу точність моделі, порівнюючи з квантуванням тренованих моделей. Згідно з ним, під час тренування створюються емульовані квантовані вузли, за допомогою яких створюється справжня квантована модель.

## 2.3 Методи обрізки нейронних мереж

### 2.3.1 Алгоритм, заснований на механізмі l1 нормалізації

#### 2.3.1.1 Вступ

Якщо після тренування нейронної мережі подивитись на значення ваг фільтрів у згорткових шарах, то можна помітити, що багато з них насправді дуже близькі до нуля.

```
[[ 0.02322666,  0.01293909, -0.09539729, ...,  0.00505567,
   0.03751447,  0.04293165],
 [-0.02554734, -0.01724515, -0.07621191, ..., -0.02270777,
   0.0040784 ,  0.03535814],
 [ 0.02018581,  0.01378967, -0.04618791, ..., -0.04711549,
   0.09202967,  0.05460544],
```

*Рисунок 1 приклади параметрів у згортковому шарі*

Це вказує на те, що ці параметри насправді не важливі для кінцевої точності моделі, і тому їх можна ігнорувати. Якщо уявити нейронну мережу як мозок, то

можна побачити, що при вирішенні конкретної задачі працює не весь мозок, а тільки його частина. Таким чином, чим більший мозок, і конкретніша задача, тим менший відсоток задіяних нейронів потрібен для виконання цієї задачі.

Згорткові нейронні мережі зазвичай мають деяку надлишковість серед фільтрів у шарах. Це означає, що фільтри можуть виявляти одні і ті самі ознаки на зображенні. Таким чином, мета алгоритму, описаного у роботі pruning filters for efficient convnets [3] є знайти ці надлишкові фільтри у моделі. Основа цього алгоритму полягає у знайденні норм фільтрів, і видаленні фільтрів з найменшою нормою. Оскільки при зменшенні кількості фільтрів зменшується і кількість операцій з матрицями, отримуємо пряму відповідність між кількістю видалених фільтрів і швидкістю роботи моделі.

#### 2.3.1.2 Опис алгоритму

Алгоритм працює лише з фільтрами згорткової нейронної мережі. Він має бути задіяний після навчання моделі, і працює над всіма згортковими шарами одразу. Запропонований підхід - використання $l_1$ норми для виявлення неважливих фільтрів і подальше їх видалення. Підхід не включає в себе ніяких інших регуляризаторів, окрім як відсотка видалення фільтрів. Для кожного шару цей відсоток є однаковим.

Алгоритм працює наступним чином:

1) Для кожного фільтра $F_i^{(l)}$, де i – індекс фільтра, l – індекс шару, обчислити суму абсолютних параметрів ядра фільтра $s_l = \sum_{m=1}^{n_i} \sum |\mathrm{K}_m|$, де $\mathrm{K} \in \mathbb{R}^{k\,x\,k}$ k = розміру ядра фільтра.
2) Відсортувати фільтри по $s_l$
3) Знайти фільтри з найменшими значеннями $s_l$ , та вибрати з них відсоток тих, які підлягають видаленню
4) Знайти відповідні цим фільтрам підрозмірності матриць у наступних шарах моделі
5) Видалити фільтри, та відповідні їм підрозмірності у всіх наступних шарах моделі, поки не дійдем до наступного згорткового шару.

6) Повторити кроки для всіх згорткових шарів у моделі.

#### 2.3.1.3 Процес знаходження найменш значущих фільтрів

Реалізація цього алгоритму розроблена у класі `Prunning`. Для того, щоб розпочати процес обрізання, потрібно викликати метод `prune_model`. Він для кожного згорткового шару запускає метод `process_conv_layer`, який відповідає за знаходження фільтрів з найменшою нормою. Норма фільтрів реалізована за допомогою методів `sum_of_absolute_kernel_weights`, який обраховує суму абсолютних значень ядра, і методу `sum_of_filter`, який знаходить суму параметрів у фільтрі. Після в конфігурації обрізаної моделі записуються необрізані фільтри, та їхні байеси.

#### 2.3.1.4 Процес видалення фільтрів в моделі

У реалізованому алгоритмі процес обрізання діє на одразу на всю модель, без попереднього огляду на значення фільтрів для точності моделі. Було реалізовано незалежний механізм обрізання, де для кожного фільтра окремо визначається його важливість, без урахування обрізаних фільтрів у попередньому згортковому шарі.

Для видалення відповідних до обрізаних фільтрів підрозмірностей у наступних шарах застосовуються метод `remove_corresponding_channels_until_next_conv`, який з огляду на тип шару, який слідує за згортковим, видаляє потрібні елементи. Оскільки для різних типів шарів існує різна кількість параметрів, то типи шарів окрема вказано в коді для забезпечення цілісності моделі.

#### 2.3.1.5 Результати роботи

Модель SimpleBirdsNet продемонструвала, що модель показує найточніші результати при 5% обрізаних параметрів(92% проти 88% до обрізання). Така аномалія може пояснюватись тим, що були видалені перетреновані фільтри, які псували загальний результат. Після видалення 20% параметрів похибка моделі збільшується на 10%. Таким чином оптимальна кількість видалених параметрів

лежить між значеннями 10% і 15%, тоді модель показує точність у 88% - 84%. Проте при використанні дотренування відсоток видалених фільтрів вдалось збільшити до 25, отримавши точність моделі 83.7%

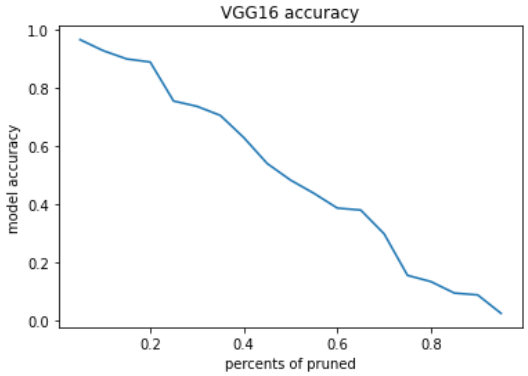



*Рисунок 2 Точнінсть моделі VGG16 відносно відсотків видалених фільтрів*

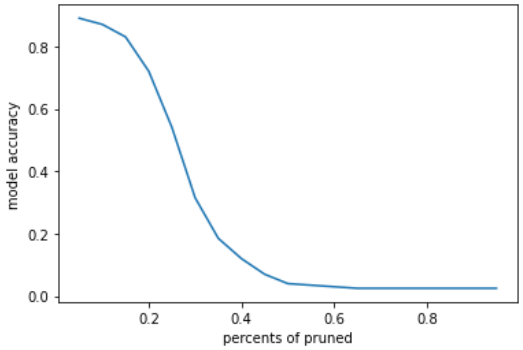



*Рисунок 3Точність моделі SumpleBirdsNet відносно від відсотка видалених фільтрів*

| Модель | Кількість параметрів | Відсоток видалення | Початкова точність | Точність після обрізання | Точність після дотренування |
|---|---|---|---|---|---|
| VGG16 | 134млн. | 46 | 95.6% | 42% | 90.5% |
| AlexNet | 62млн. | 29 | 93.2% | 54% | 92.8% |
| SimpleBirdsNet | 5млн. | 25 | 88.8% | 68% | 83.7% |

*Рисунок 4 Таблиця результатів обрізання за даним алгоритмом*

У таблиці наведено результати найуспішніших експериментів. Можна побачити, що з ростом числа параметрів моделі, росте і кількість зайвих параметрів. Оскільки експерименти проводились на моделях, тренованих за допомогою порівняно малого дата сету [6], виявилось що усі моделі мали великий надлишок параметрів. Також можна побачити, що якщо дотреновувати модель після її обрізання, то можна видалити набагато більше параметрів з порівняно невеликою втратою точності.

### 2.3.2 Алгоритм заснований на функціональній залежності фільтрів

#### 2.3.2.1 Вступ

Багато існуючих алгоритмів для обрізання нейронних мереж працюють саме з параметрами фільтрів у згорткових шарах. Такий підхід дозволяє узагальнити процес оптимізації мережі, а також спростити процес визначення надлишкових фільтрів. Проте у таких алгоритмах існує один важливий недолік – поки не було теоретично доведено стовідсоткової кореляції між параметрами фільтрів і їхньою важливістю для моделі. [9] [3] Адже, наприклад при видаленні фільтрів з відносно малими величинами параметрів існує вірогідність також видалити фільтри, які знаходять унікальні контури і патерни на зображені, що може призвести до втрати точності всієї моделі. Таким чином у роботі [10] було запропоновано алгоритм видалення фільтрів, який базується на функціональній надлишковості фільтрів у згортковому шарі нейронної мережі. Цей підхід використовує цілий ряд методик та інших алгоритмів, таких як метод максимізації матриць активації, алгоритм k-means, і багато інших для того, щоб виявити надлишкові фільтри не тільки на рівні значень їх параметрів, а й базуючись на надлишковості патернів і зображень, які ці фільтри знаходять. Цей метод дозволяє знаходити найоптимальніше відношення видалення фільтрів для кожного шару для того, щоб зменшити надмірну структурну надлишковість моделі.

#### 2.3.2.2 Опис алгоритму

Запропонований алгоритм працює у декілька етапів:

1) Знаходження функціональної візуалізації для кожного фільтра
2) Кластеризація фільтрів, яка базується на їхній функціональній візуалізації
3) Знаходження значення важливості кожного фільтра
4) Обрізання моделі і дотренування

### 2.3.2.3 Детальний опис алгоритму

#### 2.3.2.3.1 Фунціональна візуалізація фільтрів

**Опис і значення функціональної візуалізації фільтрів**

Останнім часом було зроблено величезний стрибок у сфері машинного навчання і глибоких нейронних мереж. Моделі ставали все більшими, а кількість їхніх параметрів невпинно зростала. Проте, наше розуміння того, що відбувається всередині прихованих шарів було досить низьким. Ці шари слугували як так звана «чорна коробка»(black box), тобто насправді було досить важко прослідкувати які саме обчислення проводяться всередині прихованих шарів. Для того, щоб ддізнатись які саме ознаки вибирає кожен фільтр було запропоновано алгоритми, основані на градієнтному підйомі. Одним з них є алгоритм описаний у [11]. Метою цього алгоритму є знаходження найоптимальнішою матриці активації для кожного з фільтрів. Матриця активації - це зображення, яке найбільше відповідає тим характеристикам, які розпізнає фільтр. Наприклад відомий фільтр Собеля визначає приблизне значення градієнту яскравості зображення.

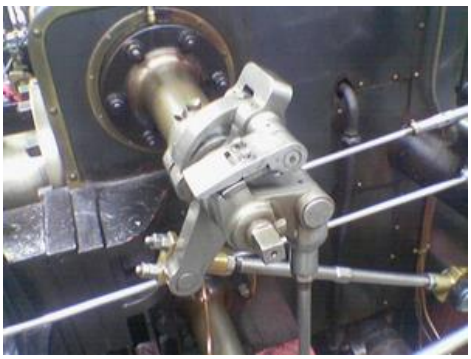


*Рисунок 5Вхідне зображення*

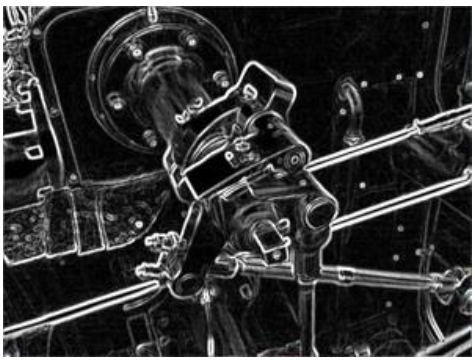


*Рисунок 6Матриця активації*

На першому рисунку зображено вхідне зображення, на другому – зображення після застосування фільтра Собеля. Оскільки фільтр знаходить контури предметів на зображенні, тоді зображення активації цього фільтра буде зображення з лініями, які визначають ці контури. Прикладом є друге зображення, де показано активація цього фільтру.

**Отримання матриць активацій**

Нехай дано зображення $X \in R^{H \times W \times C}$ , де C – кількість каналів у вхідному зображенні, H – це висота зображення, W – ширина. Тоді $A_i^l(X)$ — це матриця активації цього зображення у l шарі та фільтрі i, якщо дати його на вхід до нейронної мережі. Таким чином процес знаходження оптимальної матриці активацій можна описати за допомогою формули

$$X^* = arg\ \max\left(\mathrm{A_i^l(X)} - \mathrm{R_\Theta(X)}\right)$$

Де $R_\Theta(X)$ – це функція регуляризації зображення. Зображення X спочатку ініціалізується випадковим чином, далі для отримання найрепрезентативніших матриць активацій потрібно застосувати метод градієнтного підйому, за допомогою якого вибудовується найоптимальніша матриця активації, точність якої регулюється параметрами функцій регуляризації.

$$X \leftarrow r_\Theta\left(X + \eta \frac{\partial A_i^l(X)}{\partial X}\right)$$

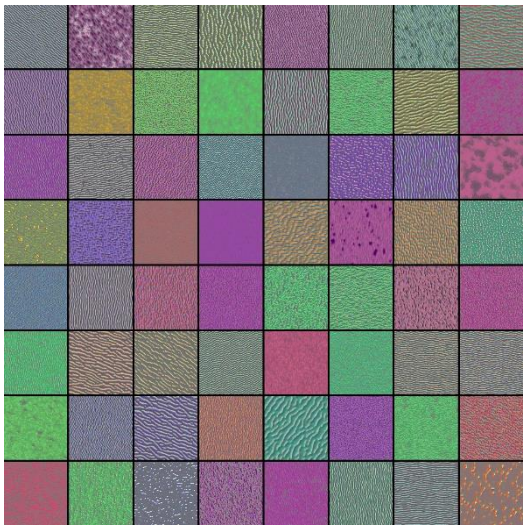


*Рисунок 8Активації першого шару моделі*

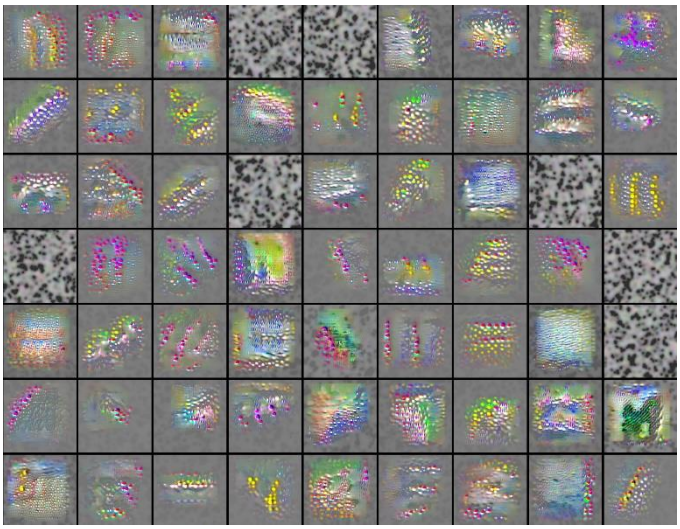


*Рисунок 9Активації третього шару моделі*

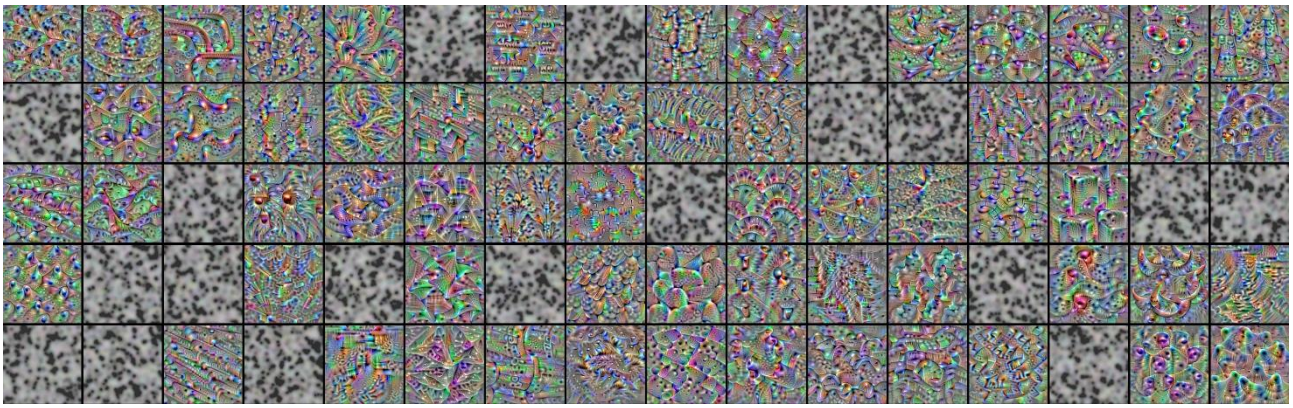


*Рисунок 7Активації останнього шару моделі*

Таким чином під час кожного кроку до зображення додається його градієнт, помножений на розмір кроку $\eta$, і застосовуються функції регуляризації.

З кожним шаром ускладнюютсья характеристики зображення. Якщо на перших шарах нейронна мережа розрізняє прості параметри, такі як колір чи проста форма, останні шари розпізнають набагато складніші форми і особливості на зображенні. На рисунках показано матриці активації фільтрів нейронної мережі, описаної раніше у розділі SimpleBirdsNet. За допомогою них можна зробити два висновки – по-перше зі зростанням глибини шарів зростає і складність зображення для активації фільтра, і по-друге – можна одразу побачити, що деякі зображення дуже схожі один на одне і демонструють схожу функціональність,

отже якщо видалити один з фільтрів, які відповідають цим зображенням – то кінцева точність моделі не має різко впасти.

**Застосування регуляризаторів**

Як можна побачити з рисунків активацій фільтрів – для деяких фільтрів матриця активацій відрізняється тим, що вона складається з шумів сірого та чорного кольорів. Ці шуми означають, що при заданих параметрах не було знайдено найоптимальнішої активації. Для того, щоб збільшити кількість та якість матриць активацій було застосовано функції регуляризатори. Ці функції застосовуються до зображення після його сумування з градієнтом. Вони призначені для зниження загального рівня шумів, збільшення кількості якісних активацій, покращення відображення активації для сприйняття.

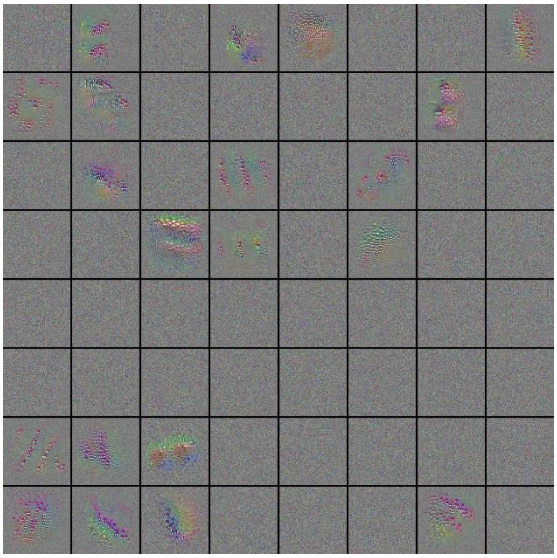


*Рисунок 10 Матриці активацій без застосування регуляризаторів*

На рисунку вище можна побачити якими виглядають матриці активацій третього шару моделі SimpleBirdsNet у випадку застосування лише методу градієнтного підйому. Порівнюючи його з прикладом, де використовувались функції регуляризації, можна помітити, що це зображення набагато більш схильне до шумів, було отримано гірші за якістю активації, а також їх кількість набагато менша. Під час реалізації алгоритму було випробувано багато різних підходів для регуляризації, а також їх параметрів. Найефективнішими з них виявились L2 розклад, фільтр Гауса та видалення пікселів з малою нормою.

**1) L2 розклад**

Ця функція призначена для штрафування великих значень. Вона застосовується для того, щоб невелика кількість пікселів з відносно великими значеннями не псували процес знаходження активації. Такі пікселі зазвичай не виникають великими групами, також вони не є корисними для знаходження кінцевої матриці активації

$$r_\Theta(X) = \left(1 - \Theta_{decay}\right)X$$

**2) Фільтр Гауса**

Для отримання зображень, менш схильних до шумів, було обрано застосування функції Гауса. Під час градієнтного підйому отримуються зображення з великою кількістю зайвих шумів, які спровоковують високий рівень активації на цих ділянках. Оскільки ці шуми не несуть корисної інформації, їх потрібно штрафувати. Для цього фільтр Гауса застосовувався як функція регуляризації.

$$r_\Theta(X) = Gauss(X, \Theta_{width})$$

Де Gauss – це фільтр Гауса, застосований до всіх каналів зображення X, з шириною ядра $\Theta_{width}$. Оскільки розмивання зображень з малою шириною ядра на кожному кроці побудови матриці активації є еквівалентним до розмивання зображення з більшою шириною ядра, але з меншою частотністю операцій, то було вирішено застосовувати фільтр Гауса кожні $\Theta_{every}$ кроки під час градієнтного підйому. Таким чином можна знизити кількість обчислень і прискорити процес знаходження оптимальної матриці активації.

**3) Видалення пікселів з малою нормою**

Описані вище функції регуляризації штрафують високо амплітудні і високочастотні пікселі на зображенні, після застосування яких зображення стає менш різким, більш згладженим, та з меншою кількістю шумів. Проте для того, щоб зображення активації не мало артефактів градієнтного підйому(пікселі, які не є головним об’єктом матриці активації, проте продовжують вносити свій внесок у градієнтний підйом) потрібно щоб такі елементи були нульовими, або

зливались з загальним фоном. Для цього потрібно застосувати функцію регуляризації для видалення пікселів з малою нормою. Ця функція обчислює норму для кожного пікселя на кожному каналі і прирівнює $\Theta_{pct}$ відсоток пікселів з найменшою нормою до нуля. Таким чином зображення матиме у собі лише головний об'єкт активації.

**4) Видалення пікселів за їхньою значимістю**

**Процес знаходження матриць активації фільтрів**

У коді за процес знаходження матриць активацій відповідає клас `LayerFiltersMaxActivation` який у конструкторі приймає параметри моделі, шару, який є згортковим і належить цій моделі, а також параметри для функцій регуляризації, яким вже виставлено значення за замовчуванням. Для того, щоб розпочати процес знаходження найоптимальнішої матриць активації для заданого згорткового шару, потрібно викликати метод `compute_layer_filters_max_activations`, який розпочне процес знаходження активацій для кожного фільтра у шарі. Для доступу до активацій існує атрибут `filters_activation_maps` , який є списком довжиною n, де n це кількість фільтрів у шарі. Кожен елемент списку – це numpy.ndarray массив розміром h,w,c. Для того, щоб отримати зображення матриць активацій потрібно викликати метод `show_images`, який виведе на екран зображення з активаціями. Якщо кількість фільтрів більша за 256, то метод виведе тільки 256 фільтрів, оскільки таке велике зображення потребує багато ресурсів для їх обробки.

### 2.3.2.3.2 Кластеризація фільтрів

Для того, щоб виявити функціональну надлишковість активацій потрібно об'єднати фільтри у групи. Для цього було обрано використання методу k середніх за допомогою методу головних компонент та методу силуету для знаходження оптимальної кількості кластерів.

**Алгоритм k means**

Одним з найпопулярніших підходів до кластеризації є застосування алгоритму k середніх. Цей алгоритм враховує набір даних у вигляді векторів і може створювати кластери на основі відстаней між ними. Він послідовно

переміщує центри кластерів, а потім групує вектори відповідно до центрів цих кластерів.

Основною його перевагою є те, що метод не потребує наперед заданих класів, а також його відносна простота і швидкість виконання. Метод було реалізовано за допомогою бібліотеки Scikit learn та класу KMeans. Цей метод доволі чутливий до високо розмірних масивів, тому за допомогою методу головних компонент виділяються головні ознаки на зображенні, а також зменшується розмірність вхідної матриці для кластеризації.

**Застосування методу головних компонент**

Цей метод є одним з основних у машинному навчанні. Він дозволяє зменшити розмірність даних, при цьому втративши мінімум корисної інформації. Обрахування головних компонент зводиться до обрахування власних векторів і власних значень коваріативної матриці вхідних даних. У роботі було використано клас PCA з бібліотеки Scikit learn.

При розмірах вхідного зображення 224x224x3, кількість параметрів, за якими проводиться кластеризація складає 150528. Якщо застосувати метод головних компонент до цього зображення з відсотком 0.99, то загальна кількість параметрів впаде до 561, що допоможе значно пришвидшити процес кластеризації. При цьому кількість кластерів, у яких знаходиться лише один фільтр також впаде.

**Застосування методу силуету для знаходження оптимальної кількості кластерів**

Метод силуету вимірює подібність елемнту до свого кластеру, а також до інших кластерів. Для кожного $k \in [1, filters\_num/2]$ потрібно знайти значення методу силуету. Чим більше це значення, тим вірогідніше, що елемент знаходиться у правильному кластері.

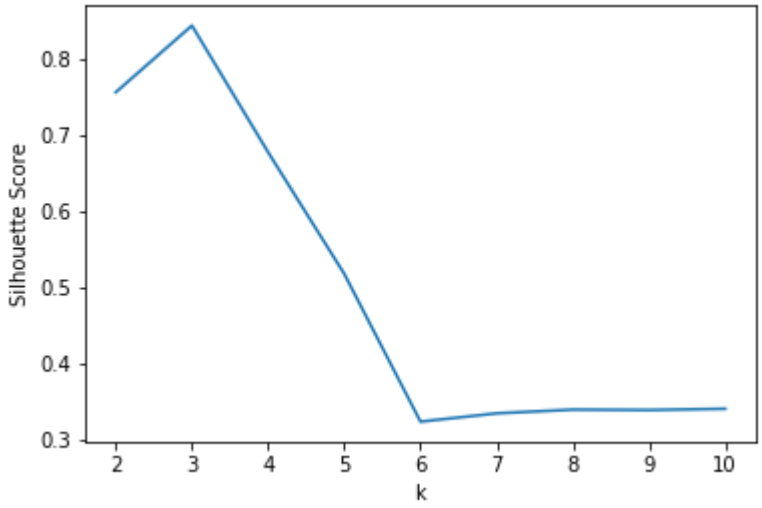



*Рисунок 113начення методу силуету при різних k кластерах*

У кінці значення порівнюються, і та кількість кластерів, де загальне значення силуету досягає глобального максимуму обирається як оптимальне k.

**Результати кластеризації**

За реалізацію кластеризації відповідає клас `LayerFiltersClusterization`. У конструкторі вказується модель та екземпляр класу `LayerFiltersMaxActivation`, де знаходятся матриці активацій фільтрів. Розроблено два методи для виконання кластеризації: `clasterize_activation_maps`, який кластеризує активації для заданого k, та `perform_k_search`, який шукає найоптимальнішу кількість кластерів і повертає індекси активацій відповідно до кластерів. Для перегляду матриць активацій відповідно до їх кластерів є метод `show_images_in_clusters` який створює зображення кластерів. Для отримання індексів фільтрів відповідно до їх кластерів є атрибут `images_indices_in_clusters` який складається зі списку масивів, де знаходяться індекси фільтрів

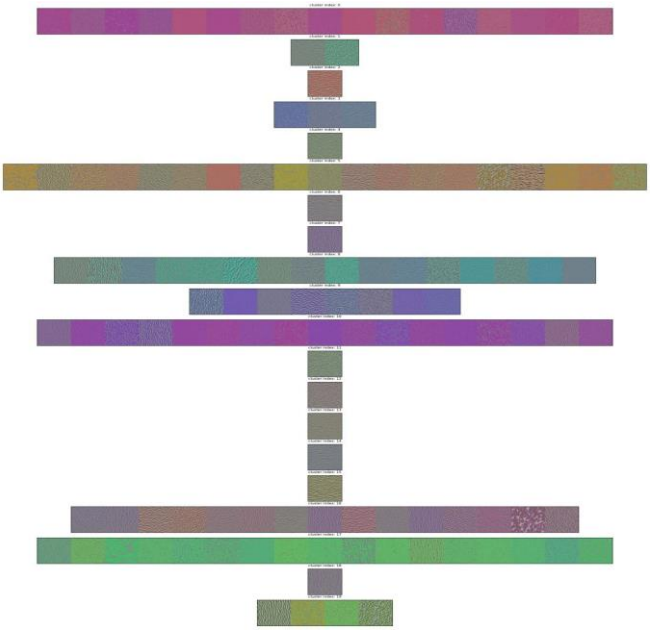


*Рисунок 12 Результат кластеризації активацій першого шару*

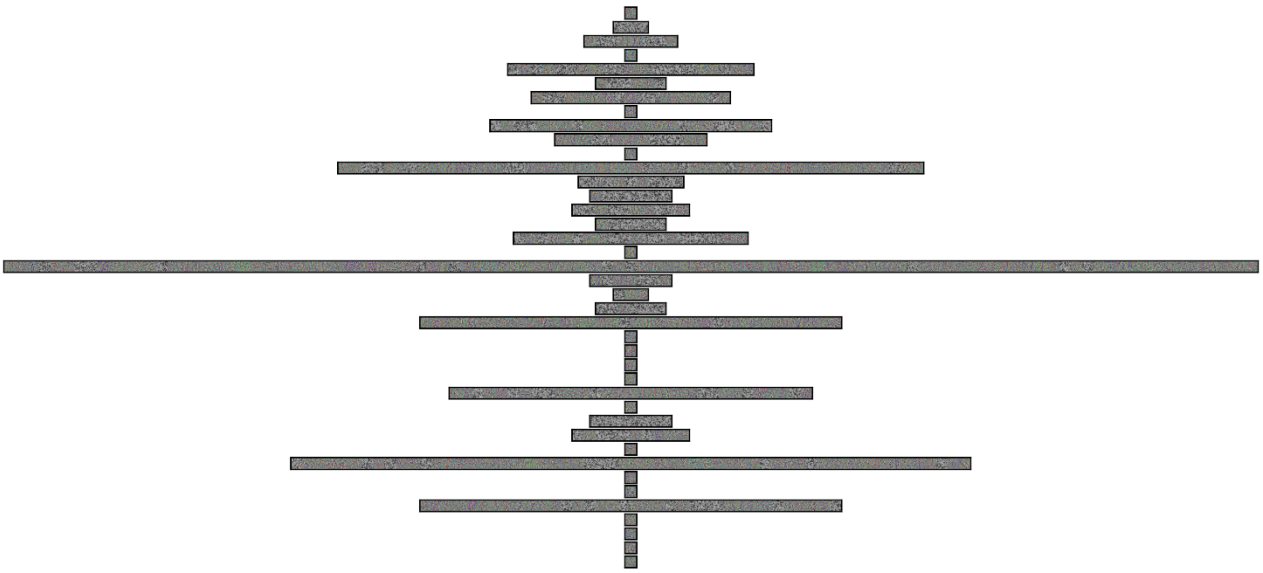


*Рисунок 13 Результат кластеризації останнього шару*

З ростом кількості кластерів, все менші відмінності у зображеннях грають більшу роль, спричиняючи те, що багато кластерів складаються лише з одної матриці активації. Ці фільтри визначають унікальні властивості на зображенні, через що їхнє видалення може призвести до втрати точності моделі. Такі кластери об’єднуються у один захищений кластер, фільтри у якому не будуть підлягати обрізанню.

#### 2.3.2.3.3 Знаходження значимості фільтрів

Фільтри зі схожою функціональністю можуть заміняти один одного. Проте вони все рівно можуть мати внесок у загальну точність моделі. Для того, щоб визначити внесок кожного фільтра у модель, використовується наступна рівність:

$$I\left(F_i^{(c,l)}\right) = \frac{1}{N}\sum_{n=1}^{N}\left\|\frac{\partial Z(F, A^l)}{\partial A_i^l(x_n)}\right\|$$

Тут кожний фільтр $F_i^{(c,l)}$ у кластера c, шару l з індексом і спочатку оцінюється за індексом внеску, який обчислюється за допомогою середніх значень градієнтів. Тут $Z(F, A^l)$ це значення функції втрат на виході всієї моделі на тестовому зображенні $x_n$ а $A_i^l(x_n)$ це матриця активації фільтра для кожного тестового зображення. У коді за цей процес відповідає клас `FilterSignificance`. У конструкторі він приймає чотири параметри – модель, тестові зображення, та відповідно класи цих зображень. Для того щоб під час цього процесу не закінчувалась оперативна пам'ять – адже він дуже ресурсозатратний – в обчисленні береться тільки 50 перших зображень. Атрибут `image_losses` повертає значення функції втрат моделі для кожного зображення у вигляді масиву, де і-тий елемент відповідає і-те зображенню. Клас обраховує значимість кожного фільтра одразу для всієї моделі, адже значення функцій втрат моделі залишаються одними і тими ж для кожного фільтру. Для початку обрахування значимості фільтрів потрібно викликати метод `compute_significance` , який спочатку обраховує одразу всі матриці активацій для кожного шару, а тоді використовує ці дані для обрахування значимості кожного фільтра. Результати записуються у атрибут `layers_filters_significance` який є словником, де ключами є назви шарів, а значеннями – масиви, де і-тий елемент масиву відповідає коефіцієнту значимості і-того фільтра у шарі.

#### 2.3.2.3.4 Обрізання моделі

Після всіх пройдених етапів було отримано потрібні дані для ідентифікації надлишкових фільтрів, отже можна починати працювати над обрізанням моделі. Для цього потрібно вказати глобальний параметр R, а також знайти локальні пропорції обрізання $r_l$ для кожного шару. Для кожного шару $r_l$ обчислюється виходячи з значимості кластеризованих фільтрів у шарі, і за замовчуванням дорівнює 1. Перед початком процесу обрізання потрібно вказати пропорції $r_l$ для кожного шару. Пропорція обрізання дорівнює R * $r_l$.

У коді процесом обрізання керує клас `FunctionalPrunning`. У конструкторі він приймає модель, коефіцієнт глобального обрізання, та масив зображень для проведення аналізу значимості фільтрів. Для початку процесу обрізання потрібно викликати метод start_prunning, який запустить всі вищезгадані методи і алгоритми, та поверне нову обрізану модель. Після цього її треба дотренувати для підвищення її точності. Обрізання було реалізовано за допомогою бібліотеки keras surgeon, яка надає зручні методи для видалення фільтрів з нейронної мережі.

#### 2.3.2.3.5 Результати

*Таблиця 1 Результати експериментів функціонального обрізання*

| Модель | Кількість параметрів | Відсоток видалення | Початкова точність | Точність після обрізання | Точність після дотренування |
|---|---|---|---|---|---|
| VGG16 | 134млн. | 51 | 95.6% | 33% | 91.9% |
| AlexNet | 62млн. | 47 | 93.2% | 45% | 91.5% |
| SimpleBirdsNet | 5млн. | 28 | 88.8% | 55% | 84.7% |

У таблиці наведено результати експериментів функціонального обрізання на трьох моделях. Для простоти і чистоти експерименту, пропорції обрізання для кожного шару складали 1. Якщо порівнювати ці результати з результатами попереднього експерименту, то можна побачити, що у загальному випадку вдалось обрізати більше параметрів та з меншою втратою точності, ніж за допомогою алгоритму l1 норми.

## Висновок

Для того, щоб створювати ефективні моделі нейронних мереж потрібно не тільки розуміти процес побудови архітектури та тренування, а й застосовувати методи оптимізації та зниження кількості обчислень моделей. У цій роботі було розглянуто алгоритми, за допомогою яких було досягнуто зниження розмірів моделі при порівняно невисоких втратах точності.

## Список літератури


[1] Andreas Brinch Nielsen та Lars Kai Hansen, «Structure Learning by Pruning in Independent Component Analysis,» 2008.

[2] W. Wan, S. Mabu, K. Shimada та K. Hirasawa, «Enhancing the generalization ability of neural networks through controlling the hidden layers,» 2009.

[3] H. Li, A. Kadav, I. Durdanovic, H. Samet and H. P. Graf, "PRUNING FILTERS FOR EFFICIENT CONVNETS," 2016.

[4] Alex Krizhevsky, Ilya Sutskever та Geoffrey E. Hinton, «ImageNet Classification with Deep Convolutional».

[5] K. Simonyan та Andrew Zisserman, «Very deep convolutional networks for large-scale image recognition,» 2014.

[6] «Birds 200 dataset,» [Онлайновий]. Available: https://www.kaggle.com/dmcgow/birds-200.

[7] «ImageNet,» [Онлайновий]. Available: http://image-net.org/index.

[8] «Quantization in tensorflow,» [Онлайновий]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization.

[9] Z. Liu, M. Sun, T. Zhou, G. Huang та T. Darrel, « Rethinking the value of network pruning,» 2018.

[10] Zhuwei Qin, Fuxun Yu, Chenchen Liu та Xiang Chen, «Functionality-Oriented Convolutional Filter Pruning».

[11] Jason Yosinski , Jeff Clune, Anh Nguyen, Thomas Fuchs та Hod Lipson , «Understanding Neural Networks Through Deep Visualization».

# Додаток 1

## Архітектура моделі SimpleBirdsNet

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      1792
_________________________________________________________________
batch_normalization (BatchNo (None, 224, 224, 64)      256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 220, 220, 128)     204928
_________________________________________________________________
batch_normalization_1 (Batch (None, 220, 220, 128)     512
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 110, 110, 128)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 108, 108, 256)     295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 108, 108, 256)     1024
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 128)       32896
_________________________________________________________________
batch_normalization_3 (Batch (None, 54, 54, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 27, 27, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 27, 27, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 512)       1638912
_________________________________________________________________
batch_normalization_4 (Batch (None, 23, 23, 512)       2048
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 11, 11, 512)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 9, 9, 256)         1179904
_________________________________________________________________
batch_normalization_5 (Batch (None, 9, 9, 256)         1024
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 128)         295040
_________________________________________________________________
batch_normalization_6 (Batch (None, 2, 2, 128)         512
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              132096
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_2 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_3 (Dense)              (None, 40)                10280
=================================================================
Total params: 5,108,904
Trainable params: 5,105,960
Non-trainable params: 2,944
```