

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

Факультет інформатики

Кафедра математики

**Кваліфікаційна робота**  
освітній ступінь – бакалавр

на тему: **"ПОРІВНЯННЯ ОПЕРАЦІЙ ОБ'ЄДНАННЯ У  
ЗГОРТКОВИХ  
НЕЙРОННИХ МЕРЕЖАХ"**

Виконала: студентка 4-го року навчання  
освітньої програми "Прикладна математика",  
спеціальності 113 Прикладна математика  
*Шульга Віра Костянтинівна*

Керівник: *Швай Н.О.*,  
кандидат фіз.-мат. наук, ст. викладач

Рецензент \_\_\_\_\_  
(*прізвище та ініціали*)

Кваліфікаційна робота захищена з  
оцінкою \_\_\_\_\_

Секретар ЕК \_\_\_\_\_  
(*підпис*)

" \_\_\_\_\_ " \_\_\_\_\_ 2022 р.

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

Факультет інформатики

Кафедра математики

ЗАТВЕРДЖУЮ

Зав. кафедри математики,  
проф., д.ф.-м.н., *Олійник Б.В.*

\_\_\_\_\_  
(підпис)

“ \_\_\_\_ ” \_\_\_\_\_ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ  
на кваліфікаційну роботу  
студентці Шульзі Вірі Костянтинівні факультету інформатики 4 курсу

**Тема:** Порівняння операцій об'єднання у згорткових нейронних мережах.

**Вихідні дані:** Порівняння операцій об'єднання на прикладі кількох згорткових нейронних мереж.

**Зміст текстової частини до кваліфікаційної роботи:**

Індивідуальне завдання

Анотація

Вступ

1 Теоретична частина згорткових нейронних мереж і операцій об'єднання

2 Реалізація операцій об'єднання та їх порівняння

Висновки

Список літератури

Дата видачі “ \_\_\_\_ ” \_\_\_\_\_ 2021 р. Керівник \_\_\_\_\_

(підпис)

Завдання отримав \_\_\_\_\_

(підпис)

**Тема:** "Порівняння операцій об'єднання у згорткових нейронних мережах".

**Календарний план виконання роботи:**

№ п/п	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Вибір і затвердження теми.	19.10.2021	
2.	Аналіз літератури.	01.01.2022	
3.	Аналіз теоретичного підґрунтя згорткових нейронних мереж.	01.02.2021	
4.	Написання першого теоретичного розділу.	15.05.2022	
5.	Реалізація класів операцій об'єднання.	30.05.2022	
6.	Порівняння операцій об'єднання на різних згорткових нейронних мережах.	06.06.2022	
7.	Внесення правок.	01.06.2022	
8.	Передзахист кваліфікаційної роботи.	15.06.2022	
9.	Фінальна корекція роботи.	27.06.2022	
10.	Захист кваліфікаційної роботи.	05.07.2022	

# Зміст

Анотація	5
Вступ	6
<b>1 Теоретична частина згорткових нейронних мереж і операцій об'єднання</b>	<b>8</b>
1.1 Загальний вигляд нейронних мереж . . . . .	8
1.2 Згорткова нейронна мережа: згортка і операція об'єднання . . . . .	10
1.3 Середнє об'єднання (Average pooling) . . . . .	11
1.4 Максимальне об'єднання (Max pooling) . . . . .	12
1.5 Змішане об'єднання максимального і середнього (Mixed max-average)	13
1.6 Маскове об'єднання змішування максимального та середнього (Gated max-average) . . . . .	14
1.7 Двовимірне\одновимірне пірамідне об'єднання (Spatial\temporal pyramid pooling) . . . . .	15
1.8 Стохастичне об'єднання (Stochastic pooling) . . . . .	16
1.9 Об'єднання із застосуванням рангу (Rank-based pooling) . . . . .	17
Середнє рангове об'єднання (Rank-based average pooling, RAP) . .	18
Зважене рангове об'єднання (Rank-based weighted pooling, RWP) .	18
Рангове стохастичне об'єднання (Rank-based stochastic pooling, RSP) . . . . .	19
<b>2 Реалізація операцій об'єднання та їх порівняння</b>	<b>19</b>
2.1 Клас змішаного об'єднання . . . . .	19
2.2 Клас маскового об'єднування змішування максимального та середнього . . . . .	20
2.3 Клас стохастичного об'єднання . . . . .	22
2.4 Клас середнього рангового об'єднання . . . . .	23
2.5 Клас зваженого рангового об'єднання . . . . .	24
2.6 Клас рангового стохастичного об'єднання . . . . .	25
2.7 Порівняння застосування об'єднань . . . . .	26
Згорткова нейронна мережа для MNIST . . . . .	26
Згорткова нейронна мережа для CIFAR10 . . . . .	32
Згорткова нейронна мережа для fashion MNIST . . . . .	38
2.8 Загальне порівняння . . . . .	42
<b>Висновки</b>	<b>44</b>
<b>Список літератури</b>	<b>45</b>

## Анотація

У даній роботі розглянуто такі методи об'єднання у згорткових нейронних мережах, як-от: максимальний, середній, змішаний, масковий змішаний, пірамідальний, стохастичний, середній ранговий, зважений ранговий та стохастичний ранговий. Здійснено реалізацію даних методів за допомогою мови програмування Python і бібліотек pytorch. Розглянуто роботу об'єднань на прикладі з згорткових нейронних мереж для класифікації датасетів MNIST, CIFAR10 та fashion MNIST.

## Вступ

Вже в 1997 році суперкомп'ютер Deep Blue від IBM зміг перемогти всесвітнього чемпіона з шахів Гаррі Каспарова, але задача знаходження на малюнку милого котика все ще була непосильною. Розпізнавання для людини є набагато простішим, аніж гра в шахи. Людина використовує багато органів чуття, які допомагають вирішувати таку купу незрозумілих для комп'ютера завдань. Також ми маємо мозок із великою кількістю нейронів, які обробляють усі ці сигнали. Виходить, що вже в той час існували нейронні мережі, але їх потужності недостатньо для обробки таких великих даних як зображення (текст). Це відбувається через те, що кожний піксель інформації стає змінною і втрачається певний зв'язок між значеннями, що знаходяться поряд. Це утворює ситуацію, в якій отримуємо не тільки велику кількість даних, а ще й втрату сенсу. [1, Geron]

Тож стає зрозумілим, що людина розподіляє малюнок таким чином, щоб виділити певні деталі, які є спрощеним варіантом отриманої інформації, але без втрати сенсу. Таку здатність мають саме згорткові нейронні мережі. Це стає реальним завдяки застосуванню (у загальному випадку) двох додаткових шарів: згортка і об'єднання. Задача згортки полягає в тому, щоб виділити певні важливі патерни, а в об'єднання – обрізати певний шум, який залишився після згортки, без втрати сенсу для зменшення обчислювального навантаження, пам'яті та кількості параметрів (зменшує ризик перенавчання). Таким чином вибір методу об'єднання має великий вплив на результат роботи мережі.

Робота складається з двох розділів.

Перший розділ розпочинається з розбору роботи згорткових нейронних мереж: їх архітектури та процесу навчання. Далі детально розглянуто теоретичне підґрунтя та особливості таких методів об'єднання, як-от: максимальний, середній, змішаний, масковий змішаний, пірамідальний, стохастичний, середній ранговий, зважений ранговий та стохастичний ранговий.

Другий розділ присвячено реалізації даних методів за допомогою мови програмування Python і бібліотеки pytorch. Згадуються особливості та складності реалізації цих класів об'єднань. Далі розглянуто роботу даних методів об'єднання на прикладі згорткових нейронних мереж із нескладною архітектурою. Робота мереж здійснюється для задач класифікації таких датасетів: MNIST (малюнки з цифрами від 0 до 0), CIFAR10 (фотографії 10 типів об'єктів) та fashion MNIST (малюнки 10 типів одягу). Порівняння роботи об'єднань відбувається за допомогою графіків функції втрат під час навчання та точності під час тестування.

Метою даної роботи є:

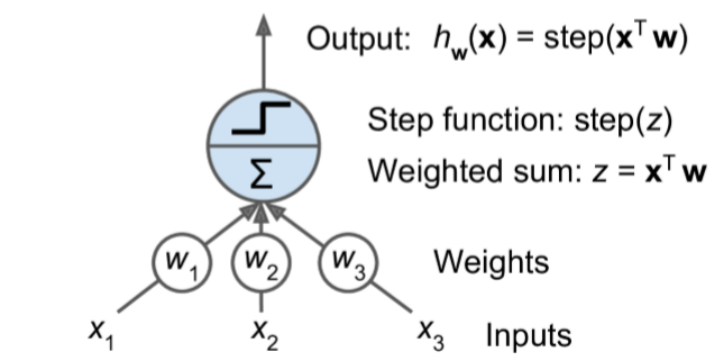
- Розглянути теоретичне підґрунтя деяких методів об'єднань у згорткових нейронних мережах.
- Реалізувати ті з них, які ще не імплементовані в загальновідомих бібліотеках, за допомогою засобів Python і бібліотеки pytorch або tensorflow.

- Порівняти роботу об'єднань на прикладі деяких згорткових нейронних мереж.

# 1 Теоретична частина згорткових нейронних мереж і операцій об'єднання

## 1.1 Загальний вигляд нейронних мереж

Звичайна нейронна мережа має схожий на мозку вигляд: тобто є нейрони, які мають між собою зв'язок. Головною частиною, яка утворює всю мережу, є нейрон. Він являє собою лінійний пороговий юніт (linear threshold unit, LTU) або пороговий логічний юніт (threshold logic unit, TLU). Головний принцип його роботи полягає в тому, що на вхід подаються дані у вигляді матриці. Кожен екземпляр розділяють на окремі значення. Далі кожному з них надається певний коефіцієнт, який передає "вплив" цієї інформації на вихідний результат нейрону (також зазвичай є вага для зміщення (bias), яка є додатковим параметром). Далі проводиться сумування добутків коефіцієнтів на вхідні дані із зміщенням, до цього добутку застосовується певна функція активації. Таким чином виходом із нейрону є значення функції (схему роботи подано на рис. 1) [1, Geron]. Отже, загальна нейронна мережа складається з таких нейронів, причому вони йдуть шарами і вхід наступного шару є виходом минулого. Зв'язки між шарами можуть мати довільну схему.

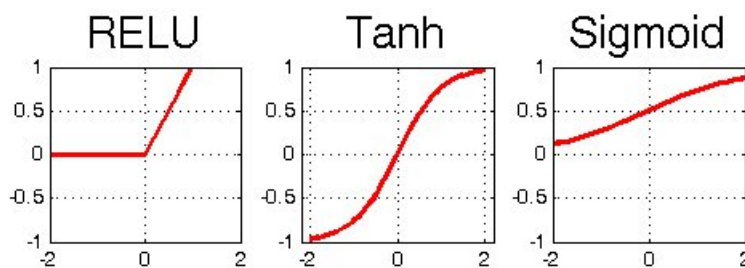


**Рис. 1:** Схема роботи лінійного порогового юніту (LTU) [1, Geron]

Якщо сенс використання коефіцієнтів для вхідних даних є досить зрозумілим, то виникає питання щодо використання функцій активації. Саме вони допомагають враховувати, чи має значення даний нейрон для виходу всієї нейронної мережі. Тобто вони "перевіряють" нейрон за допомогою функції і далі він активується або ні [1, Geron], [2, Hamdan]. Найпоширенішими серед таких функцій є ReLU (Rectified Linear Activation), сигмоїда (логістична функція) і гіперболічний тангенс ( $\tanh$ ), їх вигляд подано на рис. 2.

Навчання мережі полягає в підборі коефіцієнтів для кожного нейрону. Спочатку ці коефіцієнти визначаються випадковим чином, далі на вхід подаються значення, які проходять через мережу (forward). На виході отримуємо значення, яке треба певним чином порівняти з тим, яке б мало бути в залежної змінної. Це порівняння здійснюється за допомогою функції втрат. Її вибір повністю залежить від того, яку задачу ми маємо: класифікувати (наприклад, чи на малюнку





**Рис. 2:** Схема роботи лінійного порогового юніту (LTU) [2, Hamdan]

кіт чи собака) або регресійну (наприклад, визначити ціну на апельсиновий сік). [3, towardsdatascience], [1, Geron]

Основні функції втрат (loss functions) для задач регресії:

- Середньоквадратична помилка:  $MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$
- Абсолютна середня помилка:  $MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$
- Середня зміщена помилка:  $MBE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)}{n}$

Найпоширенішою функцією втрат для задачі класифікації є крос-ентропія (негативна логарифмічна правдоподібність):

$$\text{CrossEntropyLoss} = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (1)$$

Після отримання певної оцінки помилки необхідно оцінити зміну коефіцієнтів таким чином, щоб мінімізувати функцію втрат. Для цього використовується градієнтний спуск, який знаходить певну «яму», де функція втрат набуває найменшого значення. Цей пошук можливо здійснювати завдяки тому, що градієнт в точці показує нахил графіку і напрямок збільшення функції. Для пошуку мінімуму треба використовувати негативний градієнт. Далі спуск полягає в тому, що обчислення наступних коефіцієнтів відбувається завдяки відніманню від минулих добуток градієнту на розмір кроку (швидкість навчання), що показано на (2) [1, Geron].

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta) \quad (2)$$

Існує декілька алгоритмів градієнтного спуску, але зараз найбільшого поширення набув саме Adam, завдяки швидкій роботі для різних нейронних мереж. Це реалізується за допомогою експоненційного зважування середнього градієнтів, що прискорює збіг до мінімуму. Роботу алгоритму показано на (3). [4, geeksforgeeks]

$$\begin{aligned}\omega_{t+1} &= \omega_t - \alpha \cdot m_t, \\ m_t &= \beta \cdot m_{t-1} + (1 - \beta) \cdot \left[ \frac{\delta L}{\delta \omega_t} \right],\end{aligned}\tag{3}$$

де  $m_t$  — поточна сукупність градієнтів (у час  $t$ ),

$m_{t-1}$  — минула сукупність градієнтів (у час  $t-1$ ),

$\omega_t$  — поточне значення вагів (у час  $t$ ),

$\omega_{t+1}$  — наступне значення вагів (у час  $t + 1$ ),

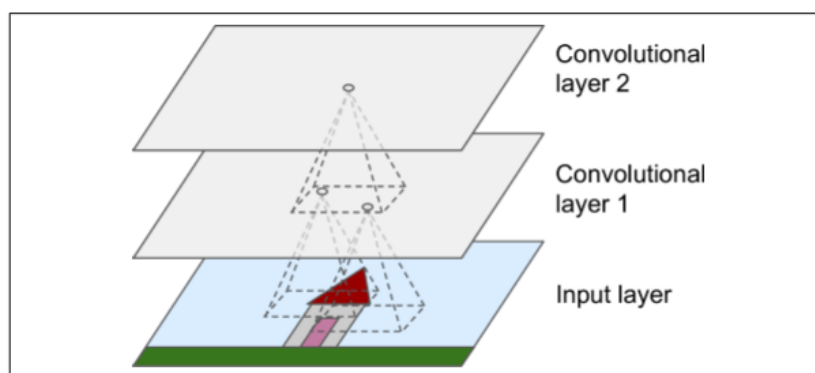
$\alpha$  — швидкість навчання,

$\frac{\delta L}{\delta \omega_t}$  — поточний градієнт (у час  $t$ ),

$\beta$  — параметр ковзного середнього (зазвичай дорівнює 0.9).

## 1.2 Згорткова нейронна мережа: згортка і операція об'єднання

У згортковій нейронній мережі данні спочатку проходять додатково шари зі згорток і об'єднань. Саме згортка розділяє зображення на певні важливі патерни. Це відбувається завдяки застосуванню фільтрів. Перевагою є те, що кожний наступний шар пов'язаний не з одним пікселем, а з цілим регіоном об'єднання, що дозволяє зберегти просторові особливості зображення, запобігти перенавчанню і зменшити кількість пам'яті, що використовується. Тобто процес, який відбувається, показано на рис. 3 [1, Geron].



**Рис. 3:** Схема роботи згорткових шарів [1, Geron]

Таким чином значення, які набуває матриця фільтру, впливають на те, які патерни буде знайдено. Наприклад, якщо б нейронна мережа мала більш схожу до людей логіку, то задача згортання обличч відбувалася б наступним чином: на першому шарі виділялися б певні горизонтальні та вертикальні лінії, далі частини обличчя (ніс, очі, рот) і в кінці отримаємо вже обличчя. Але якщо перегля-

нути зображення після застосування фільтрів, то ми не отримаємо зрозумілий для людського ока результат.

Шари об'єднання (pooling) застосовуються між згортковими. Їх задача полягає в тому, щоб зменшити навантаження для обрахунків, кількість пам'яті, що використовується і кількість параметрів (запобігання перенаванчання). Так само як і в згортковому шарі, тут наступний шар пов'язаний лише з певною областю, а не одним пікселем, що допомагає позбутися певного шуму, який заважає якісному навчанню моделі. Подібно до згортки, даний шар при застосуванні має певні параметри: розмір ядра (kernel size), кроку (stride) і додаткова рамка з нулів (padding). Принцип роботи максимального об'єднання показано на рис. 4. Тож у наступних главах буде розглянуто саме види шарів об'єднання [1, Geron].

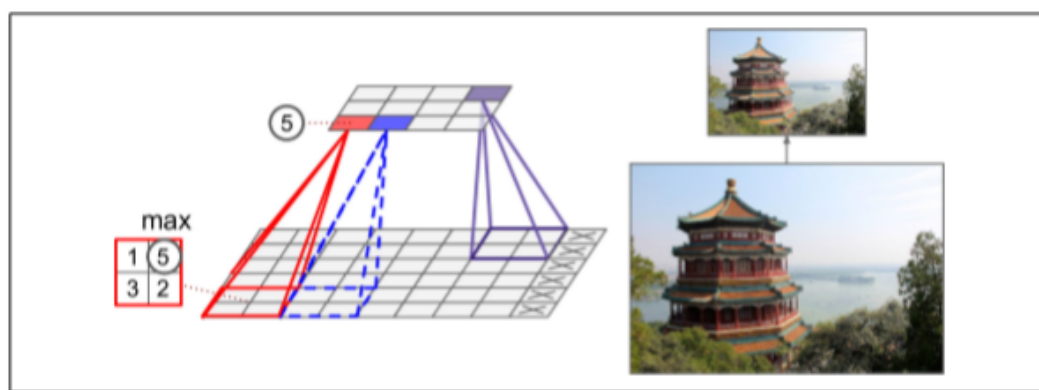


Рис. 4: Схема роботи шарів об'єднання [1, Geron]

### 1.3 Середнє об'єднання (Average pooling)

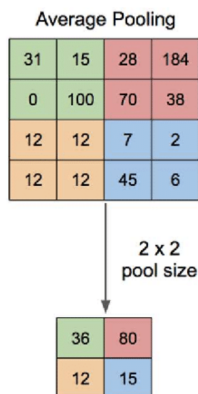
За допомогою цього методу матриця розподіляється на рівні за розміром прямокутники (квадрати) і в кожному з них обчислюється середнє значення. Далі отримаємо матрицю, яка за розмірністю довжини (ширини) рівна кількості прямокутників, які накладалися за довжиною (шириною) в початкову матрицю. Загальний вигляд обчислення подано на (4) [1, Geron], [5, Sharma, Mehra], [6, pytorch documentation], [7, Gholamalinezhad, Khosravi].

$$f_{average}(x) = \frac{1}{N} \sum_{i=1}^N x_i, \quad (4)$$

де  $x_i$  – значення з матриці, яке отримано зі згортки,

$N$  – площа ядра.

Наприклад, розглянемо застосування цього об'єднання на рис. 5 з ядром розмірності  $2 \times 2$  для даних, що мають розмірність  $4 \times 4$ .



**Рис. 5:** Приклад застосування average pooling розмірності  $2 \times 2$  зі stride=2 [8, Yani, Irawan, Setianingsih]

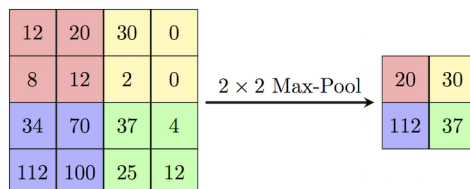
Цей метод не є досить поширеним через суттєвий недолік. Він полягає в тому, що через пошук середнього втрачається багато інформації. Тобто маємо найгіршу ситуацію у такому випадку: ненульове значення має лише один елемент у регіоні та взяття середнього його суттєво зменшує.

## 1.4 Максимальне об'єднання (Max pooling)

Із назви зрозуміло, що за цим методом також, як і за average pooling, дані розподіляються на рівні регіони (прямокутники) і визначається максимальне значення в цій області. Дана стратегія часто використовується на початку для скорочення обчислень (за рахунок зменшення розмірності) без втрати важливих патернів. Звісно певні важливі деталі втрачаються, але набагато менше, ніж за застосування average pooling, бо зберігається контраст. Тобто до даних застосовується зміна, яку подано на (5) [1, Geron], [5, Sharma, Mehra], [7, Gholamalizhad, Khosravi], [9, pytorch documentation].

$$f_{max}(x) = \max_i(x_i) \quad (5)$$

Приклад застосування max pooling розмірності  $2 \times 2$  для даних із розмірністю  $4 \times 4$  показано на рис.6.



**Рис. 6:** Приклад застосування max pooling розмірності  $2 \times 2$  зі stride=2 [10, link]

## 1.5 Змішане об'єднання максимального і середнього (Mixed max-average)

Даний метод змішує використання max і average pooling, що зрозуміло з його назви. Змішування відбувається за допомогою параметру  $\alpha_l$ , який (які) визначаємо під час визначення архітектури [5, Sharma, Mehra], [11, Lee, Gallagher, Tu]. Його визначення може відбуватися для:

- всієї нейронної мережі
- кожного шару нейронної мережі
- кожного шару і регіону, до якого застосовується об'єднання (у такому випадку навчання однакове для всіх каналів у одній області)
- кожного шару і каналу (у такому випадку немає розподілу по областях)
- шару, каналу, області у поєднанні (за цим методом отримаємо найбільше всього параметрів  $\alpha_l$ )

Зміни, які застосовує цей pooling подано на (6).

$$f_{mix}(x) = \alpha_l \cdot f_{max}(x) + (1 - \alpha_l) \cdot f_{average}(x), \quad (6)$$

де  $\alpha_l$  – параметри (один або декілька), що задаються,

$f_{max}(x)$  – максимальне об'єднання,

$f_{average}(x)$  – середнє об'єднання.

Значення  $\alpha_l$  знаходяться в проміжку від 0 до 1, що реалізує пропорційне змішування середнього і максимального об'єднання у кожному регіоні. Для навчання необхідно визначити функцію втрат  $E$  та вплив параметру  $\alpha$  на неї ( $\frac{\partial E}{\partial \alpha}$ ). Для того, щоб це вирішити, необхідно помножити чисельник і знаменник на  $\partial f_{mix}(x)$ . Таким чином отримаємо:

$$\frac{\partial E}{\partial \alpha} = \frac{\partial E}{\partial f_{mix}(x)} \cdot \frac{\partial f_{mix}(x)}{\partial \alpha} \quad (7)$$

Вираз  $\frac{\partial E}{\partial f_{mix}(x)}$  позначає помилку зворотного поширення на вищі шари й дорівнює  $\delta$ . Наступний вираз по суті є похідною для даного об'єднання за різних параметрів  $\alpha$ . Отже, отримаємо:

$$\frac{\partial E}{\partial f_{mix}(x)} \cdot \frac{\partial f_{mix}(x)}{\partial \alpha} = \delta \cdot (max_i(x_i) - \frac{1}{N} \cdot \sum_{i=1}^N x_i) \quad (8)$$

Для успішного навчання треба визначити градієнт для даного шару. Тому треба обчислити похідну для функції втрат за  $x_i$ :  $\frac{\partial E}{\partial x_i}$ . Застосуємо знов множення функції, яка використано раніше, на чисельник і знаменник. Отримаємо:

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial f_{mix}(x_i)} \cdot \frac{\partial f_{mix}(x_i)}{\partial x_i} = \delta \cdot [\alpha \cdot \mathbb{1}[x_i = \max_i x_i] + (1 - \alpha) \cdot \frac{1}{N}] \quad (9)$$

Недолік даного методу полягає в нечутливості до різних змінних через фіксований єдиний параметр, хоча і може відрізнятися для шару/області/каналу. Але даний метод дає кращі результати, ніж max і average. [5, Sharma, Mehra], [11, Lee, Gallagher, Tu]

## 1.6 Маскове об'єднання змішування максимального та середнього (Gated max-average)

Даний метод є досить схожим на минулий через поєднання двох стратегій: середнього та максимального. Gated max-average теж може мати по окремому параметру для області/регіону/шару чи один на всю мережу. Але це об'єднання відрізняється виглядом параметром змішування. У даному методі це спеціальна маска (gated mask). Дана маска має ту ж розмірність, що і регіон об'єднання. Отримаємо скаляр за допомогою множення регіону на маску, а потім для цього значення застосовується сигмоїда. Таким чином отримаємо параметр, який має ту ж саму область значень, що й в згаданому змішаному об'єднанні. Тобто обчислення відбувається за (10) [5, Sharma, Mehra], [11, Lee, Gallagher, Tu].

$$f_{gated} = \sigma(\omega^T x) \cdot f_{max}(x) + (1 - \sigma(\omega^T x)) \quad (10)$$

Визначимо формулу (11) для обчислення впливу значення маски на функцію втрат й використаємо множення чисельника та знаменника на  $\partial f_{gated}(x)$ .

$$\frac{\partial E}{\partial \omega} = \frac{\partial E}{\partial f_{gated}(x)} \cdot \frac{\partial f_{gated}(x)}{\partial \omega} \quad (11)$$

Обчислимо похідну для сигмоїди в загальному випадку на (12).

$$(\sigma(x))'_x = \left( \frac{1}{1 + e^{-x}} \right)'_x = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} = \sigma(x)(1 - \sigma(x)) \quad (12)$$

Таким чином визначено вираз на (13) для градієнту, за допомогою підстановки значення похідної для сигмоїди і помилки зворотного поширення  $\delta$ .

$$\frac{\partial E}{\partial f_{gated}(x)} \cdot \frac{\partial f_{gated}(x)}{\partial \omega} = \delta \cdot \sigma(\omega^T x)(1 - \sigma(\omega^T x)) \cdot x \cdot (\max_i(x_i) - \frac{1}{N} \sum_{i=1}^N x_i) \quad (13)$$

Відповідно також знайдемо, як впливають змінні на функцію втрат, бо поширення помилки відбувається всередині. Застосуємо вже використані способи для обчислення та отримаємо результат на (14) [5, Sharma, Mehra], [11, Lee, Gallagher, Tu].

$$\begin{aligned} \frac{\partial E}{\partial x_i} &= \frac{\partial E}{\partial f_{gated}(x)} \cdot \frac{\partial f_{gated}(x)}{\partial x_i} = \delta \cdot [\sigma(\omega^T x)(1 - \sigma(\omega^T x)) \cdot \omega_i \cdot \\ &\cdot (\max_i(x_i) - \frac{1}{N} \sum_{i=1}^N x_i) + \sigma(\omega^T x) \cdot \mathbb{1}[x_i = \max_i x_i] + (1 - \sigma(\omega^T x)) \cdot \frac{1}{N}] \end{aligned} \quad (14)$$

## 1.7 Двовимірне\одновимірне пірамідне об'єднання (Spatial\temporal pyramid pooling)

Як відомо, мапи об'єктів з усіх каналів на виході зі згорткового шару повинні мати обмежений розмір. До створення пірамідального об'єднання застосовували розрізання (cropping) і об'єднання (wrapping) даних, перед до повноз'язного шару (наприклад, застосування Flattering). Таке звичайне обрізання та об'єднування в одну мапу сприяє до втрати певного вмісту і просторових орієнтацій. Через це було винайдено даний pooling [5, Sharma, Mehra], [12, Grauman, Darrell], [13, He, Zhang, Ren, Sun], [14, tensorflow documentation].

Основна ідея даного методу полягає в тому, що мапа об'єктів розподіляється на певну кількість регіонів у двовимірному випадку (на вектор-рядки для одновимірного), які мають пропорційний розмір до загального. Кількість регіонів відрізняється у рівнів піраміди. Далі для кожного регіону (рядку) застосовується додаткове максимальне, середнє чи інше об'єднання для надання кожному шматку одного значення. Далі спочатку відбувається конкатенація значень для однакових рівнів піраміди, а потім для всіх разом, у єдиний вектор, який і є результатом роботи даного методу. Для прикладу розглянемо застосування двовимірного пірамідного об'єднання (рис. 7).

Мапа об'єктів, що подається на вхід, має розмір  $256 \times 256$ , кількість рівнів піраміди рівна 3. Таким чином отримаємо на рівні 0 один шматок, який повністю складається із всіх значень мапи. До нього застосовуємо один із згаданих раніше методів об'єднання (наприклад, максимальне), і отримаємо одне значення. На рівні 1 пропорційно розділяємо дану мапу на 4 частини, кожна з яких має розмір  $64 \times 64$ . Далі застосовуємо певне об'єднання для кожної частини на отримаємо 4 значення. Рівень 2: розподіляємо мапу на 16 частин розміром  $16 \times 16$ , визначаємо 16 значень після об'єднання. Далі конкатенуємо значення на кожному рівні, а потім всі рівні разом. Отримаємо  $16 + 4 + 1 = 21$  значення, які передаються наступному рівню згорткової нейронної мережі. Відповідно, якщо було б застосовано одновимірний спосіб, то отримали б ту ж саму кількість елементів після останньої конкатенації, але застосування проміжних об'єднань відбувалося б для рядків, і отримані значення можуть відрізнятися.

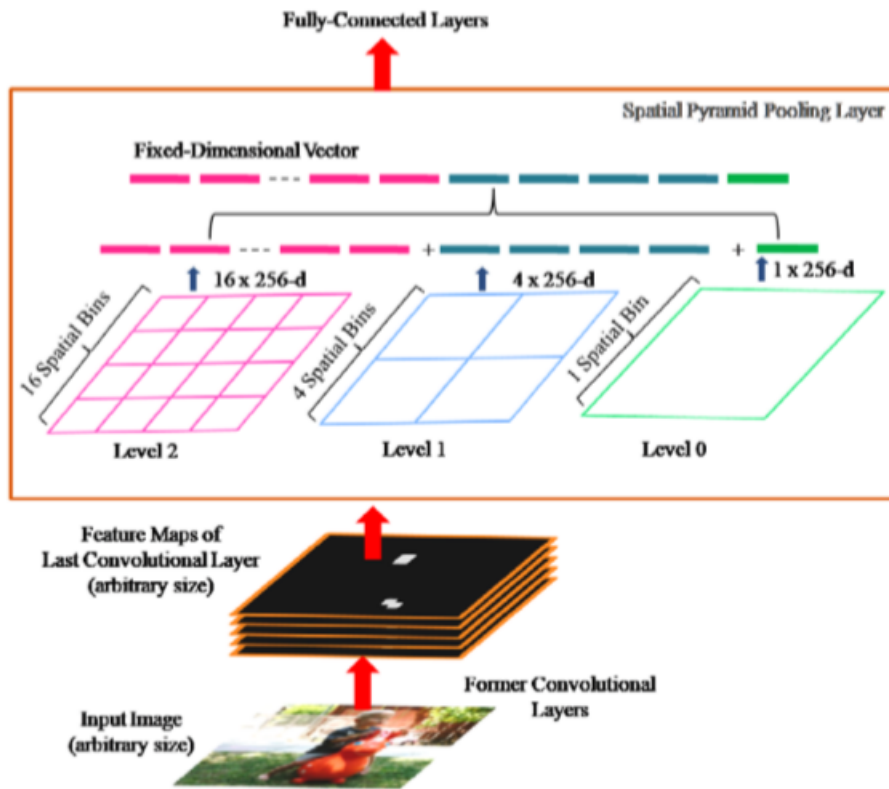


Рис. 7: Приклад застосування spatial pyramid pooling для мапи розміром  $256 \times 256$  з рівнями 0-2

## 1.8 Стохастичне об'єднання (Stochastic pooling)

Головна ідея цього методу полягає у стохастичному процесі, який забезпечує меншу ймовірність перенавчання моделі. Головна проблема середнього об'єднання полягає в тому, що значення, які мають високу активацію після згорткового шару, зменшуються за рахунок неактивованих і аналогічна ситуація відбувається навпаки. Таким чином даний метод об'єднання допомагає вирішити проблему постійно однозначного вибору, як у максимального та середнього об'єднання. Алгоритм полягає у розподілі мапи змінних на регіони, далі застосовується ReLU (через те, що необхідні додатні значення для майбутньої нормалізації, яка визначає ймовірнісний простір для даного регіону). Далі кожне значення ділиться на суму всіх (15), і таким чином область значень після застосування цієї процедури визначається на відрізьку  $[0, 1]$ . [5, Sharma, Mehra], [15, Zeiler, Fergus]

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k} \quad (15)$$

Тож даний ймовірнісний простір задає мультиноміальний розподіл, за допомогою якого визначається остаточне значення для даного регіону об'єднання. (16).

$$s_j = a_l \text{ where } l \sim P(p_1, \dots, p_{|R_j|}) \quad (16)$$

Тобто чим більше значення, тим вище ймовірність, що його буде визначе-



но, але залишається випадкова складова, через яку може бути визначено й не найбільше значення. Приклад застосування даного методу зображено на рис. 8. Головний недолік цього об'єднання полягає в тому, що вибираються лише не негативні значення через процедуру нормалізації. Визначимо проблему масштабування, яка полягає в тому, що на якість роботи моделі дуже сильно впливає розмір навчальної вибірки. За застосування даного методу об'єднання з'являється ця проблема через:

- врахування лише позитивних активованих значень у малі змінних,
- рахування ймовірностей за допомогою нормалізації, що робить ці значення неконтрольованими

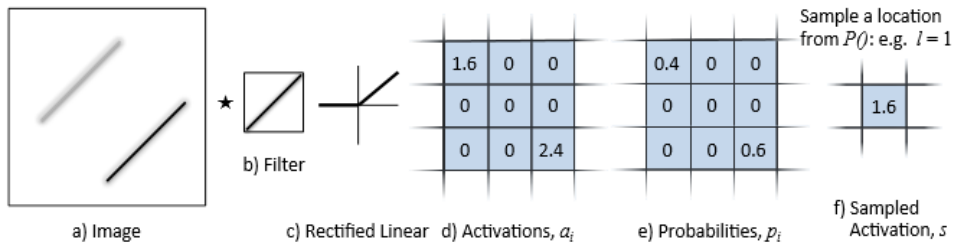


Рис. 8: Приклад застосування stochastic pooling, [15, Zeiler, Fergus]

## 1.9 Об'єднання із застосуванням рангу (Rank-based pooling)

Для вирішення проблем масштабу (про яку згадано у стохастичному об'єднанні) було створено категорію рангових методів об'єднання. Перед визначенням їх роботи відбувається надання рангу для кожного елементу із мапи змінних. Спочатку записуються елементи в порядку спадання значення, тобто отримаємо множину:  $T \rightarrow \{a_{max}, \dots, a_{min}\}$ . Далі визначається множина рангів:  $T \rightarrow \{1, \dots, n\}$ , де  $n$  - це площа ядра об'єднання. Таким чином чим менше значення рангу, тим більшим є елемент, що показано на (17) для елементів на  $i$ -ому і  $j$ -ому місці [5, Sharma, Mehra], [16, Shi, Ye, Wu].

$$a(i) > a(j) \implies r(i) < r(j) \quad (17)$$

У випадку рівних значень більший ранг має той елемент, який знаходиться раніше, тобто це показано на (18).

$$a(i) = a(j) \wedge i < j \implies r(i) < r(j) \quad (18)$$

Коли ранги визначено, то далі застосовується один із наступних методів.

## Середнє рангове об'єднання (Rank-based average pooling, RAP)

Середнє рангове об'єднання створене, щоб не втрачати важливу інформацію, як наприклад, це відбувається в максимальному і середньому об'єднанні. Таким чином в даному методі також використовується середнє значення, але для елементів, що мають ранг, який менше або дорівнює  $t$ . Тобто зважується кожне з перших  $t$  значень, а інші вважаються рівними 0. Формально показано як відбувається об'єднання для  $R_j$  регіону для значень, що маю ранг не більше  $t$  на (19). Тож якщо покладемо  $t = 1$ , тоді отримаємо лише максимальне значення, а у випадку  $t = n$  (де  $n$  - це площа регіону об'єднання) отримаємо середнє значення всієї області. Оптимальним значенням для  $t$ , яке об'єднує середнє і максимальне об'єднання, є медіана (визначено у [16, Shi, Ye, Wu]). Приклад застосування показано на рис. 9 [5, Sharma, Mehra], [16, Shi, Ye, Wu].

$$s_j = \frac{1}{t} \sum_{i \in R_j, r_i \leq t} a_i \quad (19)$$

## Зважене рангове об'єднання (Rank-based weighted pooling, RWP)

У ранговому середньому об'єднанні для кожного із значень, які мають ранг не більше за  $t$ , визначено однакову вагу. Але очевидно, що чим менше ранг, тим більшу вагу повинен мати елемент регіону. Визначимо зваження елементів за допомогою експоненційної функції. Таким чином вага для кожного значення буде обчислюватися за (20) [5, Sharma, Mehra], [16, Shi, Ye, Wu].

$$p_r = \alpha(1 - \alpha)^{r-1}, \quad r = 1, \dots, n \quad (20)$$

Тобто маємо параметр  $\alpha$ , значення якого знаходяться на  $[0, 1]$ . Отже, легко довести, що за таких значень  $\alpha$  сума  $p_r$ , при  $n$  наближається до нескінченності, буде рівна 1, що показано на (21).

$$\begin{aligned} \sum_{r=1}^n &= 1 - (1 - \alpha)^n \\ \lim_{n \rightarrow +\infty} \sum_{r=1}^n p_r &= 1 \end{aligned} \quad (21)$$

Після отримання "вагів" для кожного елементу регіону, обчислюється сума з добутків із знайдених значень і елементів регіону, що показано на (22). Завдяки тому, що зваження враховує на скільки значення є великим, це сприяє покращенню продуктивності об'єднання. Приклад застосування показано на рис. 9.

$$s_j = \sum_{i \in R_j} p_i \cdot a_i \quad (22)$$

## Рангове стохастичне об'єднання (Rank-based stochastic pooling, RSP)

Даний метод є досить близьким до згаданого стохастичного, бо тут також використовується мультиноміальний розподіл (показано на (16)), завдяки чому виникає стохастичний простір. Але обчислення ймовірностей відбувається за допомогою тої ж процедури, що і в зваженому ранговому об'єднанні (20). Завдяки цьому обчисленню ймовірностей маємо більшу ймовірність для визначення не найбільшого елементу. Значення  $\alpha = 0,5$  призводить до якісної роботи моделі завдяки високому рівню випадковості. Рангове стохастичне об'єднання зберігає значно більше інформації, порівняно із звичайним стохастичним, завдяки тому, що ймовірності обчислюються за допомогою застосування експоненційної функції до рангів. Таким чином через врахування від'ємних значень цей метод має потенціал позбутися проблеми масштабування. Приклад застосування показано на рис. 9 [5, Sharma, Mehra], [16, Shi, Ye, Wu].

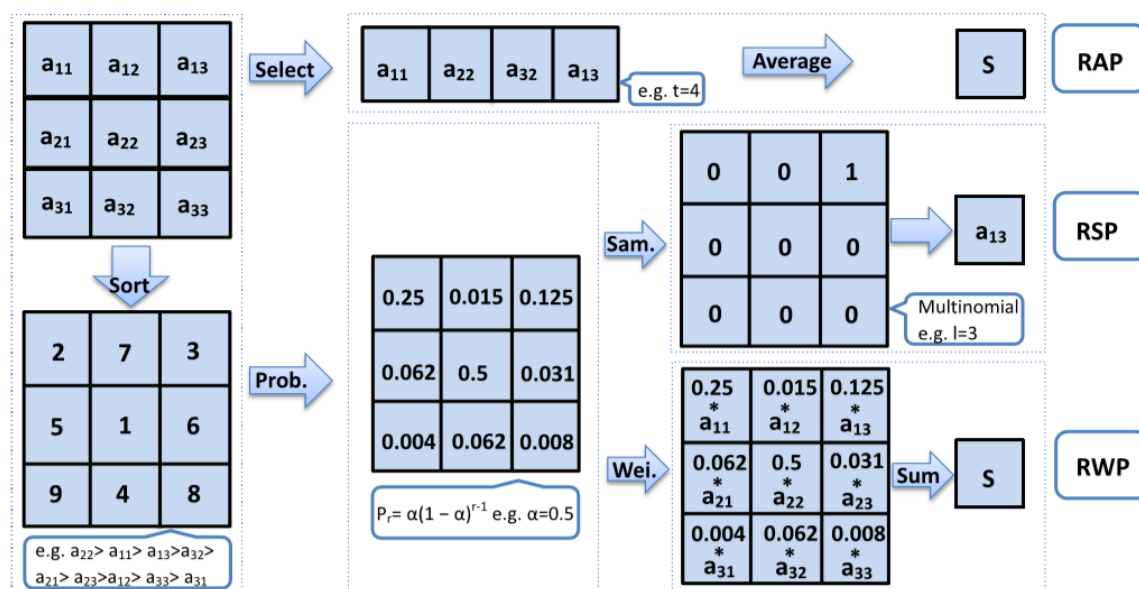


Рис. 9: Приклад застосування rank based pooling

## 2 Реалізація операцій об'єднання та їх порівняння

### 2.1 Клас змішаного об'єднання

Для створення згорткової нейронної мережі, об'єднань і виконання навчання та тестування будемо використовувати бібліотеки torch і torchvision (Лістинг 1)

Лістинг 1: Завантаження необхідних бібліотек

```
import torch
import torchvision
import torchvision.transforms as transforms
```

```
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
```

Для реалізації даного методу об'єднання, треба додатково у клас додати параметр альфа, а далі модель повертатиме пропорційне змішування максимального і середнього об'єднання (Лістинг 2).

**Лістинг 2:** Клас mixed pooling

```
class mixedPool(nn.Module):
    def __init__(self, kernel_size, stride, alpha = 0.5, padding = 0):
        super(mixedPool, self).__init__()
        alpha = torch.FloatTensor([alpha])
        self.alpha = nn.Parameter(alpha)
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

    def forward(self, x):
        x = self.alpha * F.max_pool2d(x,
                                       self.kernel_size,
                                       self.stride,
                                       self.padding) + (1 - self.alpha) * F.avg_pool2d(x,
                                       self.kernel_size,
                                       self.stride,
                                       self.padding)

        return x
```

## 2.2 Клас маскового об'єднування змішування максимального та середнього

Реалізуємо даний метод об'єднання, де навчання буде відбуватися для кожного шару чи для шару і каналу. Для цього створимо один клас, де будуть дві окремі функції, в залежності від того, скільки параметрів для навчання потрібно. У випадку, коли потрібна одна маска для всього шару, матимемо одну маску, яка має розмірність ядра, а у випадку коли для кожного каналу, отримаємо, що розмірність маски рівна розмірності ядра на канал і канал. Далі у випадку навчання для шару пробігаємося по каналам і застосовуємо одну й ту саму згортку, а якщо для каналу і шару, то застосовуємо цю велику маску для всіх даних. Тобто маємо реалізацію класу на Лістинг 3.

**Лістинг 3:** Клас gated pooling

```
class gatedPool(nn.Module):
    def __init__(self, in_channel, kernel_size, stride, padding = 0,
                 learn_option='l/c'):
        super(gatedPool, self).__init__()
```

```

if learn_option == 'l/c':
    self.mask = nn.Parameter(torch.randn(in_channel,
                                          in_channel,
                                          kernel_size,
                                          kernel_size).float())

elif learn_option == 'l':
    self.mask = nn.Parameter(torch.randn(1,
                                          1,
                                          kernel_size,
                                          kernel_size).float())

else:
    raise NameError(learn_option)

self.learn_option = learn_option
self.kernel_size = kernel_size
self.stride = stride
self.padding = padding

def layer(self, x):
    size = list(x.size())[1]
    channels_gated = []

    for ch in range(size):
        a = x[:,ch,:,:]
        a = torch.unsqueeze(a,1)
        a = F.conv2d(a,self.mask,stride = self.stride)
        channels_gated.append(a)
    gated = channels_gated[0]
    for channel_gated in channels_gated[1:]:
        gated = torch.cat((gated,channel_gated),1)

    alpha = F.sigmoid(gated)

    x = alpha * F.max_pool2d(x,
                            self.kernel_size,
                            self.stride,
                            self.padding) + (1 - alpha)*F.avg_pool2d(x,
                            self.kernel_size,
                            self.stride,
                            self.padding)

    return x

def layer_channel(self, x):
    mask_channels = F.conv2d(x,self.mask,stride = self.stride)
    alpha = F.sigmoid(mask_channels)
    x = alpha * F.max_pool2d(x,
                            self.kernel_size,
                            self.stride,
                            self.padding) + (1-alpha) * F.avg_pool2d(x,
                            self.kernel_size,

```

```

        self.stride,
        self.padding)

    return x

```

## 2.3 Клас стохастичного об'єднання

Для реалізації цього методу необхідно, щоб після подання батчу, дані розподілялися на регіони об'єднання. Далі проходимо по кожному регіону і застосовуємо ReLU. Після цього вже можна зробити нормування на суму значень, таким чином знайшовши ймовірність. Далі застосовуємо функцію з мультиноміальним розподілом, яка є в pytorch. Так отримавши індекс, можемо записати значення. Отже, матимемо даний результат для кожного регіону. Потім знов збираємо тензори у батч (ця процедура є досить схожою на ту, що була на початку). Тобто маємо реалізацію цього класу на Лістинг 4.

Лістинг 4: Клас stochastic pooling

```

class stochasticPool(nn.Module):
    def __init__(self, kernel_size, stride, padding = 0):
        super(stochasticPool, self).__init__()
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

    def forward(self, x):
        x = F.pad(x, (self.padding, self.padding, self.padding, self.padding))
        n_imgs = x.shape[0]
        n_channels = x.shape[1]
        n_height = x.shape[2]
        n_width = x.shape[3]

        x = F.relu(x)
        x_unfold = F.unfold(x, kernel_size=self.kernel_size, stride=self.stride)

        n_regions = x_unfold.shape[-1]
        n_regions_side = int(np.sqrt(n_regions))

        x_unfold = x_unfold.view(n_imgs, n_channels, self.kernel_size,
                                self.kernel_size, n_regions).permute(0, 4, 1, 2, 3)

        norm = torch.sum(x_unfold, dim=(-1, -2)).view(n_imgs, n_regions,
                                                       n_channels, 1, 1)

        x_normed = torch.nan_to_num(x_unfold / norm).view(n_imgs, n_regions,
                                                           n_channels, self.kernel_size ** 2)

        output = torch.zeros((n_imgs, n_channels, n_height // self.stride, n_width
                               // self.stride))

        for idx_i, img in enumerate(x_normed):

```

```

for idx_r, region in enumerate(img):
    for idx_c, channel in enumerate(region):
        if torch.sum(channel) == 0:
            output[idx_i][idx_c][idx_r // n_regions_side][idx_r %
                n_regions_side] = 0
            continue
        idx = torch.multinomial(channel, 1)
        val = channel[idx] * norm[idx_i][idx_r][idx_c][0][0]
        output[idx_i][idx_c][idx_r // n_regions_side][idx_r %
            n_regions_side] = val

return output

```

Загалом навчання нейронної мережі з даним об'єднанням займає більше часу через нешвидку роботу функції з мультиноміальним розподілом. Спочатку було реалізовано таким чином, що для кожної зони виконувалося нормування, але в такому випадку модель навчалася занадто довго. Обчислення норми попередньо до визначення стохастичного значення збільшило швидкість навчання приблизно в 3 рази.

## 2.4 Клас середнього рангового об'єднання

Щоб реалізувати даний метод об'єднання, також використовуємо розподіл на регіони, як і в стохастичному. Далі відсортовуємо значення за спаданням і беремо середнє для перших  $t$  значень. Таким чином отримаємо клас на Лістинг 5

Лістинг 5: Клас rank-based average pooling

```

class rankAvgPool(nn.Module):
    def __init__(self, kernel_size, stride, t=-1, padding = 0):
        super(rankAvgPool, self).__init__()
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding
        if t == -1:
            self.t = kernel_size // 2
        else:
            self.t = t

    def forward(self, x):
        x = F.pad(x, (self.padding, self.padding, self.padding, self.padding))
        n_imgs = x.shape[0]
        n_channels = x.shape[1]
        n_height = x.shape[2]
        n_width = x.shape[3]

        x_unfold = F.unfold(x, kernel_size=self.kernel_size, stride=self.stride)

```

```

n_regions = x_unfold.shape[-1]
n_regions_side = int(np.sqrt(n_regions))

x_unfold = x_unfold.view(n_imgs,
                          n_channels,
                          self.kernel_size,
                          self.kernel_size,
                          n_regions).permute(0, 4, 1, 2, 3).view(n_imgs,
                                                                    n_regions,
                                                                    n_channels,
                                                                    self.kernel_size ** 2)

t_sorted = torch.sort(x_unfold, descending=True, dim=-1).values[:, :, :, :
                                                                (self.t + 1)]

output = torch.mean(t_sorted, dim=-1).permute(0, 2, 1).view(n_imgs,
                                                             n_channels, n_regions_side, n_regions_side)

return output

```

Спочатку було реалізовано за допомогою проходження по кожній зоні, але в цьому випадку навчання було дуже довгим. Тому для вирішення цієї проблеми попередньо дані сортуються за допомогою `sort` і обчислюється середнє з використанням `mean` (функція з бібліотеки `pytorch`).

## 2.5 Клас зваженого рангового об'єднання

Щоб реалізувати цей метод об'єднання, необхідно подібно до рангового середнього, розподілити мапу змінних на регіони. Ймовірності обчислюються ще на рівні ініціалізації, бо для них достатньо знати площу ядра. Далі створюємо тензор, що має ймовірність ту ж саму, що і поданий батч. Зважені значення обраховано за допомогою множення. Потім відбувається сумування, яке дає остаточні значення для кожного регіону (Лістинг 6).

**Лістинг 6:** Клас rank-based weighted pooling

```

class rankWeightedPool(nn.Module):
    def __init__(self, kernel_size, stride, padding = 0, alpha = 0.5):
        super(rankWeightedPool, self).__init__()
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding
        probas = []
        for rank in range(kernel_size ** 2):
            probas.append(alpha * (1 - alpha) ** rank)
        self.probas = torch.Tensor(probas)

    def forward(self, x):
        x = F.pad(x, (self.padding, self.padding, self.padding, self.padding))
        n_imgs = x.shape[0]

```



```

n_channels = x.shape[1]
n_height = x.shape[2]
n_width = x.shape[3]

x_unfold = F.unfold(x, kernel_size=self.kernel_size, stride=self.stride)

n_regions = x_unfold.shape[-1]
n_regions_side = int(np.sqrt(n_regions))

probas = torch.cat([self.probas] * (n_imgs * n_channels *
    n_regions)).view(n_imgs,
    n_regions,
    n_channels,
    self.kernel_size ** 2)
x_unfold = x_unfold.view(n_imgs,
    n_channels,
    self.kernel_size,
    self.kernel_size,
    n_regions).permute(0, 4, 1, 2, 3).view(n_imgs,
    n_regions,
    n_channels,
    self.kernel_size ** 2)

x_sorted = torch.sort(x_unfold, descending=True, dim=-1).values
mul = self.probas * x_sorted
output = torch.sum(mul, dim=-1).permute(0, 2, 1).view(n_imgs, n_channels,
    n_regions_side, n_regions_side)

return output

```

## 2.6 Клас рангового стохастичного об'єднання

Цей метод реалізовано дуже подібно до минулих об'єднань, що базуються на ранзі. Ймовірності також обраховуються на етапі ініціалізації, а далі створюємо збільшений варіант з ймовірностями із розмірністю, яку має батч. Потім подібно до стохастичного об'єднання визначаємо кожне значення, пройшовшись циклами (Лістинг 7).

**Лістинг 7:** Клас rank-based stochastic pooling

```

class rankStochasticPool(nn.Module):
    def __init__(self, kernel_size, stride, padding = 0, alpha = 0.5):
        super(rankStochasticPool, self).__init__()
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding
        probas = []
        for rank in range(kernel_size ** 2):
            probas.append(alpha * (1 - alpha) ** rank)
        self.probas = torch.Tensor(probas)

```

```

def forward(self, x):
    x = F.pad(x, (self.padding, self.padding, self.padding, self.padding))
    n_imgs = x.shape[0]
    n_channels = x.shape[1]
    n_height = x.shape[2]
    n_width = x.shape[3]

    x_unfold = F.unfold(x, kernel_size=self.kernel_size, stride=self.stride)

    n_regions = x_unfold.shape[-1]
    n_regions_side = int(np.sqrt(n_regions))

    x_unfold = x_unfold.view(n_imgs,
                              n_channels,
                              self.kernel_size,
                              self.kernel_size,
                              n_regions).permute(0, 4, 1, 2, 3).view(n_imgs,
                              n_regions,
                              n_channels,
                              self.kernel_size ** 2)

    probas = torch.cat([self.probas] * (n_imgs * n_channels *
                                         n_regions)).view(n_imgs,
                                                         n_regions,
                                                         n_channels,
                                                         self.kernel_size ** 2)

    output = torch.zeros((n_imgs, n_channels, n_height // self.stride, n_width
                          // self.stride))

    for idx_i, img in enumerate(x_unfold):
        for idx_r, region in enumerate(img):
            for idx_c, channel in enumerate(region):
                idx = torch.multinomial(probas[idx_i][idx_r][idx_c], 1)
                val = channel[int(idx)]
                output[idx_i][idx_c][idx_r // n_regions_side][idx_r %
                    n_regions_side] = val

    return output

```

## 2.7 Порівняння застосування об'єднань

### Згорткова нейронна мережа для MNIST

Для порівняння роботи методів об'єднання для першої мережі використаємо MNIST. Даний датасет складається з малюнків цифр від 0 до 9 розміром 28 на 28 пікселів. Кожний малюнок подано у вигляді тензора розмірністю 28 на 28, де кожне значення пікселю є числом від 0 до 1 (тобто присутнє нормування). Загальний вигляд екземплярів подано на рис. 10. Навчатися модель буде

на 6400 екземплярах через нешвидку роботу стохастичних об'єднань. Для їх навчання на великій кількості даних необхідно мати більш потужний процесор та додатково реалізовувати їх на іншій мові програмування (наприклад, C++). Відповідно через ту саму проблему тестування також відбувається не на цілій вибірці, а 1000 екземплярах.

Завантажуємо дані і створюємо DataLoader з 4-ма екземплярами у кожному батчі за допомогою бібліотек torch і torchvision (Лістинг 8)

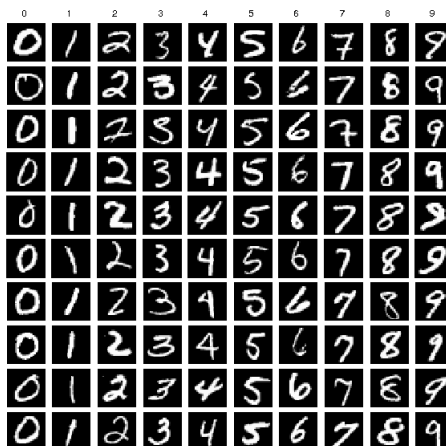


Рис. 10: Приклад датасету MNIST [17, Lim, Young, Patton]

**Лістинг 8:** Завантаження MNIST і створення DataLoader з 4 екземплярами у кожному батчі

```
transform = transforms.Compose(
    [transforms.ToTensor()])

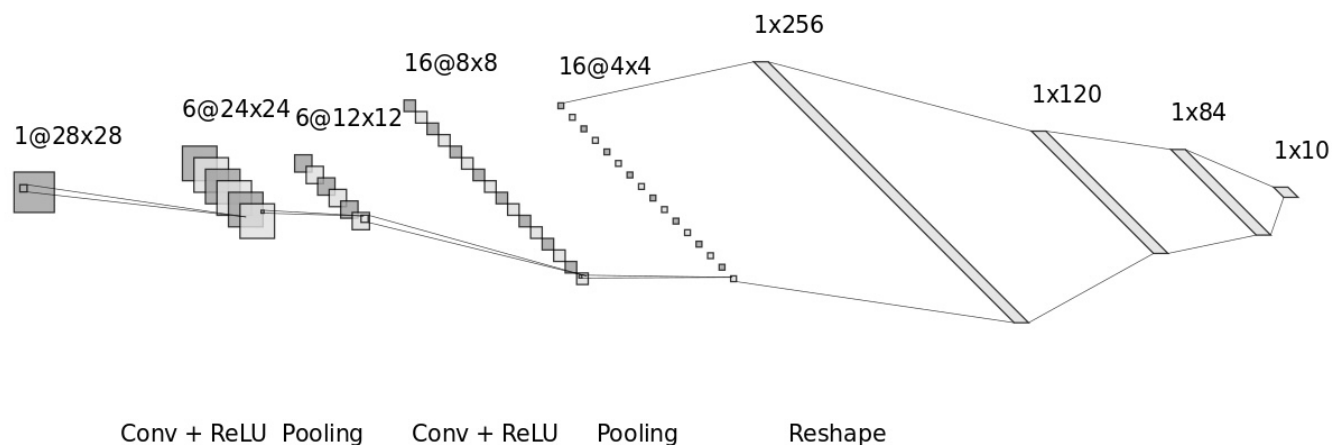
trainset_MNIST = torchvision.datasets.MNIST(root='./data', train=True,
download=True, transform=transform)

testset_MNIST = torchvision.datasets.MNIST(root='./data', train=False,
download=True, transform=transform)

trainset_sub = Subset(trainset, indices=range(6400))
trainloader = torch.utils.data.DataLoader(trainset_sub,
batch_size=4, shuffle=True, num_workers=2)

testset_sub = Subset(trainset, indices=range(1000))
testloader = torch.utils.data.DataLoader(testset_sub,
batch_size=4,
shuffle=True, num_workers=2)
```

Для класифікації використаємо ЗНМ з 2 згортковими шарами між якими будемо використовувати різні pooling, з 3 повнозв'язними шарами, у якості функції активації використаємо ReLU, з CrossEntropy як функцією втрат, а градієнтний спуск буде Adam (будову мережі подано на рис. 11). Через достатньо довгу роботу стохастичних об'єднань будемо навчатися на одній епосі. Побудуємо загальний клас мережі для MNIST, який потім будемо наслідувати для майбутнього порівняння з різними об'єднаннями (Лістинг 9).



**Рис. 11:** Архітектура згорткової нейронної мережі для класифікування MNIST

**Лістинг 9:** Загальний вигляд ЗНМ для MNIST: class MNIST

```
class model_MNIST(nn.Module):
    def __init__(self):
        super(model_MNIST, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(4 * 4 * 16, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.loss_fn = nn.CrossEntropyLoss()
        self.optimizer = torch.optim.Adam(self.parameters(), lr=1e-4)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.reshape(x.shape[0], 4 * 4 * 16)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def train(self, batches):
        losses = []
        running_loss = 0.0
        for i, batch in enumerate(batches):
            X_batch, y_batch = batch

            X_batch = X_batch
            y_batch = y_batch

            self.optimizer.zero_grad()

            y_pred = self.forward(X_batch)
            loss = self.loss_fn(y_pred, y_batch)
```

```

loss.backward()
self.optimizer.step()

running_loss += loss.item()

if i % 200 == 199:
    print('{} loss: {}'.format(i + 1,
                                round(running_loss / 200,
                                      3)))
    losses.append(running_loss)
    running_loss = 0.0

plt.plot(np.arange(len(losses)), losses)
plt.show()
print('train ended')

def test(self, data):
    class_correct = list(0 for i in range(10))
    class_total = list(0 for i in range(10))
    classes = np.arange(10)

    with torch.no_grad():
        for i, batch in enumerate(data):
            images, labels = batch
            y_pred = self.forward(images)
            _, predicted = torch.max(y_pred, 1)

            checker = (predicted.detach() == labels)

            for i in range(4):
                label = labels[i]
                class_correct[label] += checker[i].item()
                class_total[label] += 1

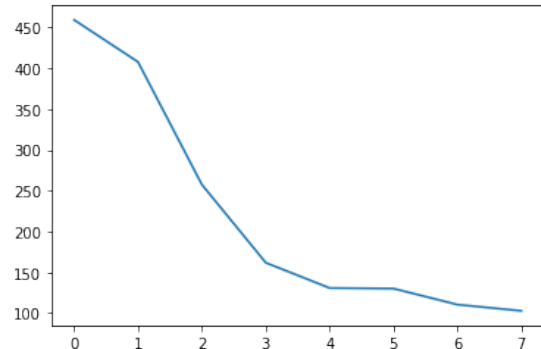
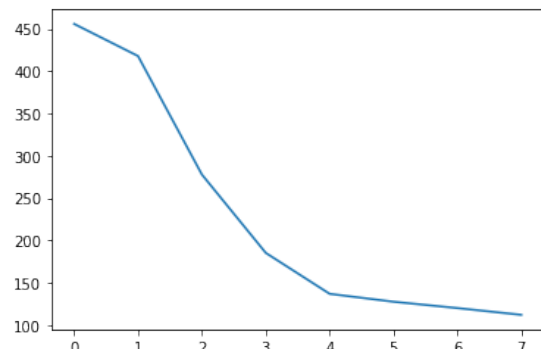
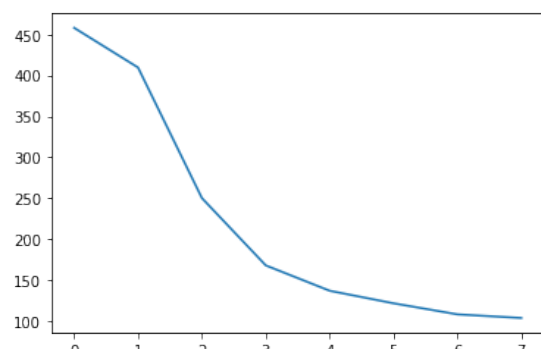
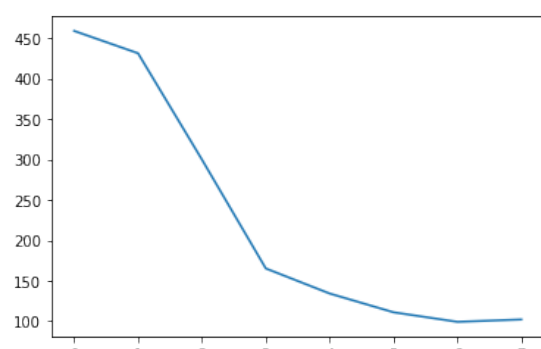
            for idx, num_class in enumerate(classes):
                print('Accuracy of {}: {}'.format(num_class,
                                                  round(class_correct[idx] / class_total[idx] * 100, 3)))

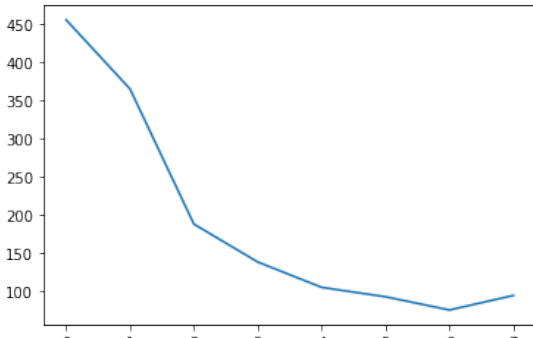
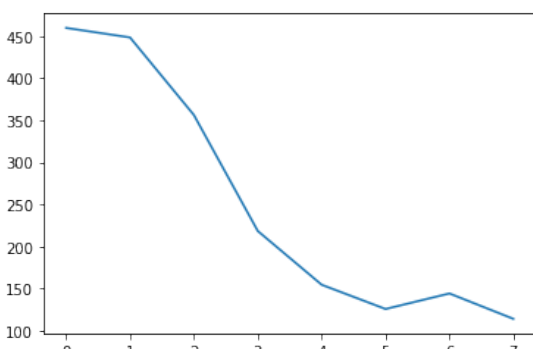
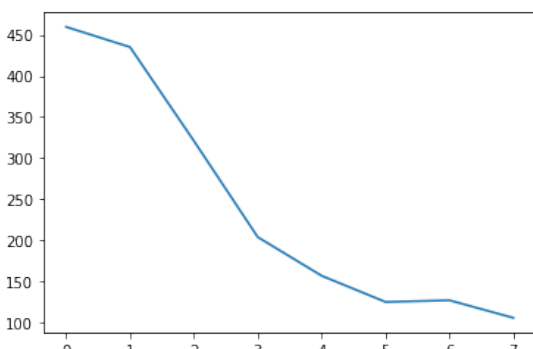
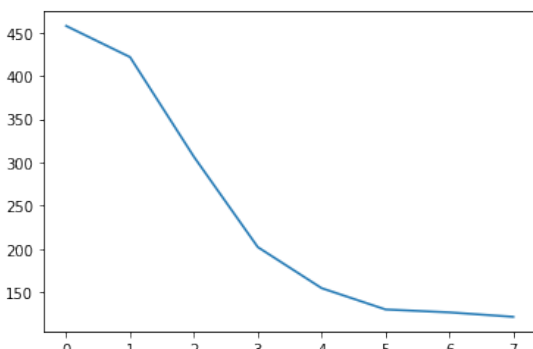
    print('Mean accuracy = {}'.format(sum(class_correct) / sum(class_total) *
                                       100))

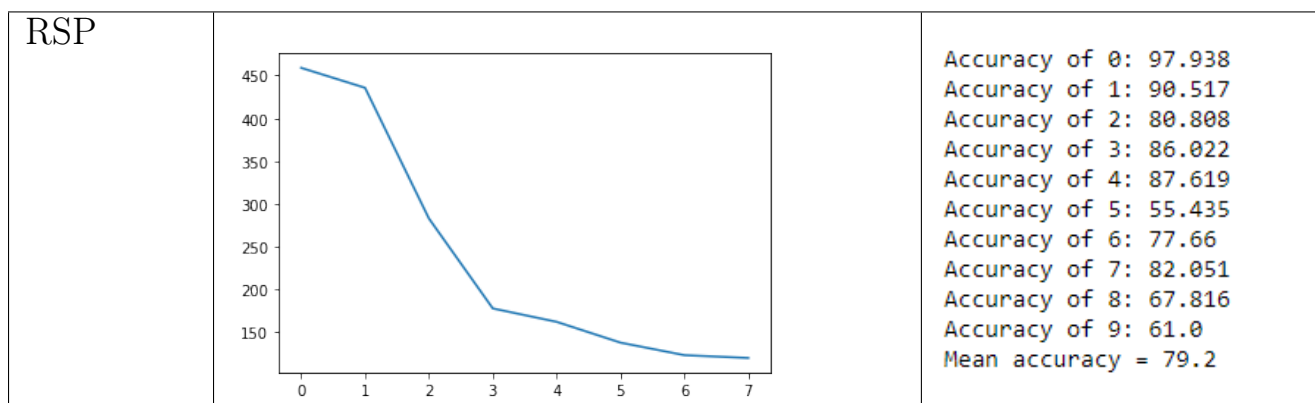
```

Отримаємо такі результати роботи мережі із різними об'єднаннями на таблиці 1.

Табл. 1: Порівняння роботи різних об'єднань із однаковою архітектурою мережі для MNIST

Назва об'єднання	Графік функції втрат (навчання)	Точність моделі для кожного класу (тестування)
Average		Accuracy of 0: 96.907 Accuracy of 1: 93.103 Accuracy of 2: 74.747 Accuracy of 3: 83.871 Accuracy of 4: 80.0 Accuracy of 5: 52.174 Accuracy of 6: 91.489 Accuracy of 7: 86.325 Accuracy of 8: 87.356 Accuracy of 9: 84.0 Mean accuracy = 83.3
Max		Accuracy of 0: 96.907 Accuracy of 1: 93.103 Accuracy of 2: 77.778 Accuracy of 3: 77.419 Accuracy of 4: 92.381 Accuracy of 5: 70.652 Accuracy of 6: 78.723 Accuracy of 7: 88.034 Accuracy of 8: 88.506 Accuracy of 9: 58.0 Mean accuracy = 82.5
Mixed		Accuracy of 0: 96.907 Accuracy of 1: 94.828 Accuracy of 2: 79.798 Accuracy of 3: 86.022 Accuracy of 4: 87.619 Accuracy of 5: 71.739 Accuracy of 6: 92.553 Accuracy of 7: 89.744 Accuracy of 8: 70.115 Accuracy of 9: 61.0 Mean accuracy = 83.5
Gated		Accuracy of 0: 96.907 Accuracy of 1: 94.828 Accuracy of 2: 75.758 Accuracy of 3: 89.247 Accuracy of 4: 91.429 Accuracy of 5: 70.652 Accuracy of 6: 94.681 Accuracy of 7: 87.179 Accuracy of 8: 62.069 Accuracy of 9: 78.0 Mean accuracy = 84.6

Spatial pyramid + max		<p>           Accuracy of 0: 100.0            Accuracy of 1: 94.828            Accuracy of 2: 81.818            Accuracy of 3: 82.796            Accuracy of 4: 94.286            Accuracy of 5: 82.609            Accuracy of 6: 91.489            Accuracy of 7: 94.017            Accuracy of 8: 82.759            Accuracy of 9: 84.0            Mean accuracy = 89.2         </p>
Stochastic		<p>           Accuracy of 0: 95.876            Accuracy of 1: 93.103            Accuracy of 2: 64.646            Accuracy of 3: 84.946            Accuracy of 4: 84.762            Accuracy of 5: 48.913            Accuracy of 6: 92.553            Accuracy of 7: 84.615            Accuracy of 8: 70.115            Accuracy of 9: 79.0            Mean accuracy = 80.4         </p>
RAP		<p>           Accuracy of 0: 95.876            Accuracy of 1: 93.966            Accuracy of 2: 77.778            Accuracy of 3: 73.118            Accuracy of 4: 84.762            Accuracy of 5: 57.609            Accuracy of 6: 90.426            Accuracy of 7: 78.632            Accuracy of 8: 80.46            Accuracy of 9: 84.0            Mean accuracy = 82.0         </p>
RWP		<p>           Accuracy of 0: 94.845            Accuracy of 1: 93.103            Accuracy of 2: 78.788            Accuracy of 3: 78.495            Accuracy of 4: 86.667            Accuracy of 5: 59.783            Accuracy of 6: 88.298            Accuracy of 7: 86.325            Accuracy of 8: 86.207            Accuracy of 9: 54.0            Mean accuracy = 81.0         </p>



Бачимо, що найкращий результат на достатньо невеликій кількості даних має spatial pyramid у поєднанні з max. На мою думку, серед досліджених об'єднань, цей є найстійкішим до проблеми масштабування. Також досить гарний результат роботи мають gated і mixed. Високий відсоток точності має саме gated завдяки тому, що є параметр, який навчається. Через це якість роботи моделі не суттєво змінюється за збільшення тренувальних даних на відміну від стохастичних (stochastic і RSP). Тобто у випадкових об'єднань бачимо ту саму проблему масштабування, що виникає при недостатньо великому датасеті. Загалом pooling з випадковою складовою дають набагато кращі результати і навіть можуть бути на рівні з mixed і gated, якщо навчання відбувається на великих даних і з великою кількістю епох, що бачимо з [15, Zeiler, Fergus]. Подивившись на графіки mixed і gated, зазначимо, що вони мають досить схожі хвости наприкінці. З чого випливає, що вони дають близькі результати під час навчання.

## Згортова нейронна мережа для CIFAR10

Друга мережа буде навчатися класифікувати CIFAR10. Цей датасет складається із кольорових малюнків (розміром 32 на 32 пікселі) різних об'єктів 10 типів, як-от: літак, машина, птах, кіт, олень, собака, жаба, кінь, корабель, вантажівка. Загальний вигляд екземплярів зображено на рис. 12. Відповідно кожний малюнок подано у вигляді тензору розмірністю 3 (RGB – ref green blue) на 32 на 32, де кожне значення є числом від 0 до 1 (тобто є нормування). Через довгу роботу об'єднань з випадковою складовою навчання буде відбуватися на 3200 екземплярах, бо в цьому випадку робота відбувається ще довше, через те, що для моделі необхідне збільшення кількості каналів, бо малюнок є кольоровим.

Завантажуємо дані і створюємо DataLoader з 4-ма екземплярами у кожному батчі за допомогою бібліотек torch і torchvision (Лістинг 10).

**Лістинг 10:** Завантаження CIFAR10 і створення DataLoader з 4 екземплярами у кожному батчі

```

transform = transforms.Compose(
    [transforms.ToTensor()])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
```



```

download=True, transform=transform)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)

trainset_sub = Subset(trainset, indices=range(3200))
testset_sub = Subset(testset, indices=range(600))

trainloader = torch.utils.data.DataLoader(trainset_sub, batch_size=4,
shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
shuffle=True, num_workers=2)

```



**Рис. 12:** Приклад датасету CIFAR10 [18, github]

Архітектура мережі складається з 2 згорткових шарів між якими будемо використовувати різні pooling, з 2 повнозв'язних шарів, у якості функції активації також використаємо ReLU, CrossEntropy як функцію втрат, а градієнтний спуск буде здійснюватися за допомогою Adam (схему архітектури подано на рис. 13). Після декількох спроб було вирішено збільшити кількість епох від 1 до 2, бо на даному датасеті моделі набагато важче вчитися на такій малій кількості даних, але через довгу роботу стохастичних об'єднань неможливо перевірити на більшому розмірі датасету за використання засобів python. Створимо загальний клас мережі для CIFAR10, який потім буде наслідуватися для порівняння pooling (Лістинг 11). Реалізація навчання і тестування майже не відрізняється від класу для MNIST. Різниця лише в назвах класів зображень.

**Лістинг 11:** Загальний вигляд ЗНМ для CIFAR10: class CIFAR10

```

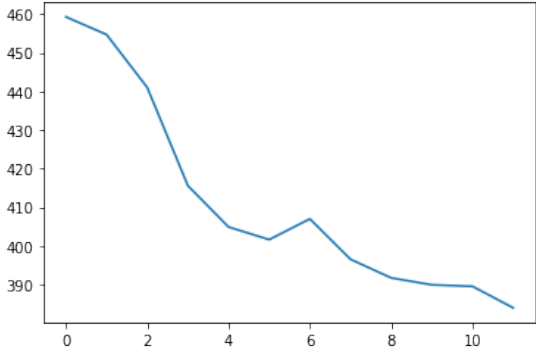
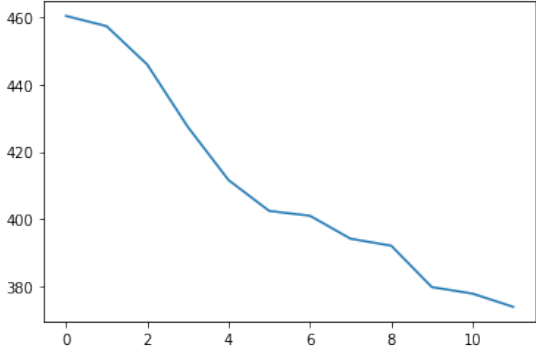
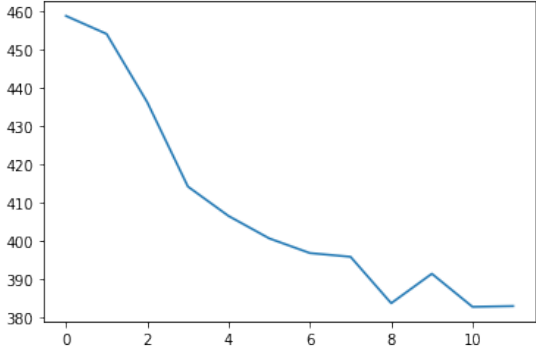
class model_CIFAR10(nn.Module):
    def __init__(self):
        super(model_CIFAR10, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3)
        self.fc1 = nn.Linear(6 * 6 * 32, 64)
        self.fc2 = nn.Linear(64, 10)
        self.loss_fn = nn.CrossEntropyLoss()

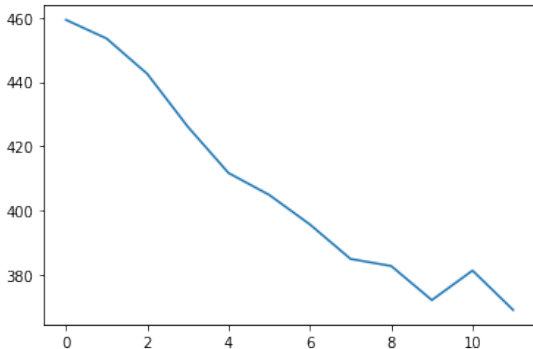
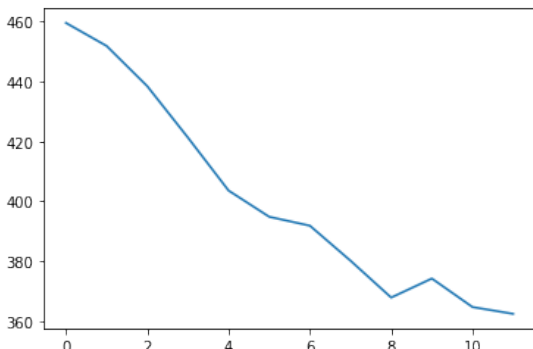
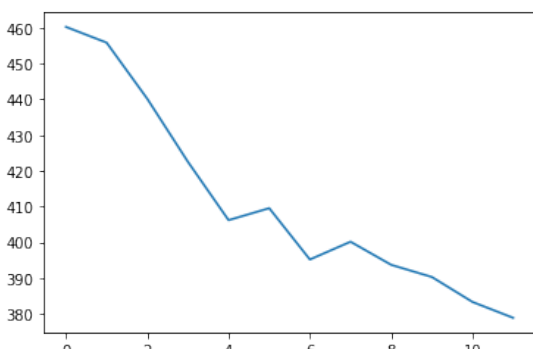
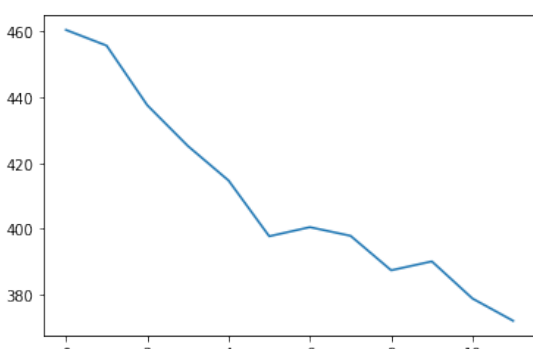
```

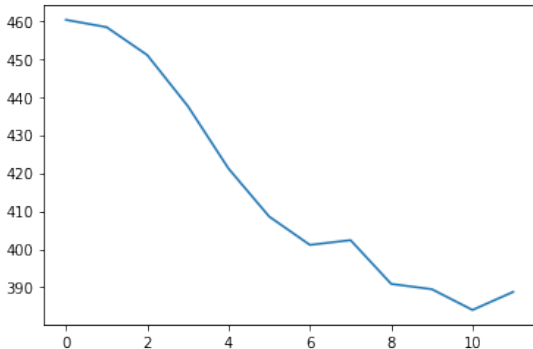
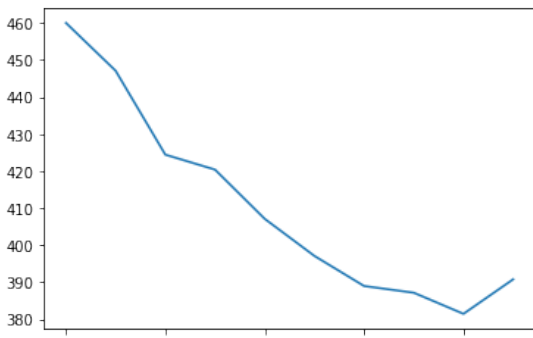
```
self.optimizer = torch.optim.Adam(self.parameters(), lr=1e-4)
```

Отримаємо такі результати при навчанні класифікування CIFAR10 з даною архітектурою на таблиці 2.

**Табл. 2:** Порівняння роботи різних об'єднань із однаковою архітектурою мережі для CIFAR10

Назва об'єднання	Графік функції втрат (навчання)	Точність моделі для кожного класу (тестування)
Average		Accuracy of plane: 27.273 Accuracy of car: 43.478 Accuracy of bird: 22.414 Accuracy of cat: 1.786 Accuracy of deer: 13.725 Accuracy of dog: 29.032 Accuracy of frog: 75.714 Accuracy of horse: 7.018 Accuracy of ship: 17.46 Accuracy of truck: 50.704 Mean accuracy = 30.166666
Max		Accuracy of plane: 28.788 Accuracy of car: 23.913 Accuracy of bird: 29.31 Accuracy of cat: 3.571 Accuracy of deer: 1.961 Accuracy of dog: 40.323 Accuracy of frog: 71.429 Accuracy of horse: 7.018 Accuracy of ship: 61.905 Accuracy of truck: 46.479 Mean accuracy = 33.5
Mixed		Accuracy of plane: 40.909 Accuracy of car: 50.0 Accuracy of bird: 8.621 Accuracy of cat: 0.0 Accuracy of deer: 1.961 Accuracy of dog: 46.774 Accuracy of frog: 67.143 Accuracy of horse: 42.105 Accuracy of ship: 31.746 Accuracy of truck: 43.662 Mean accuracy = 34.5

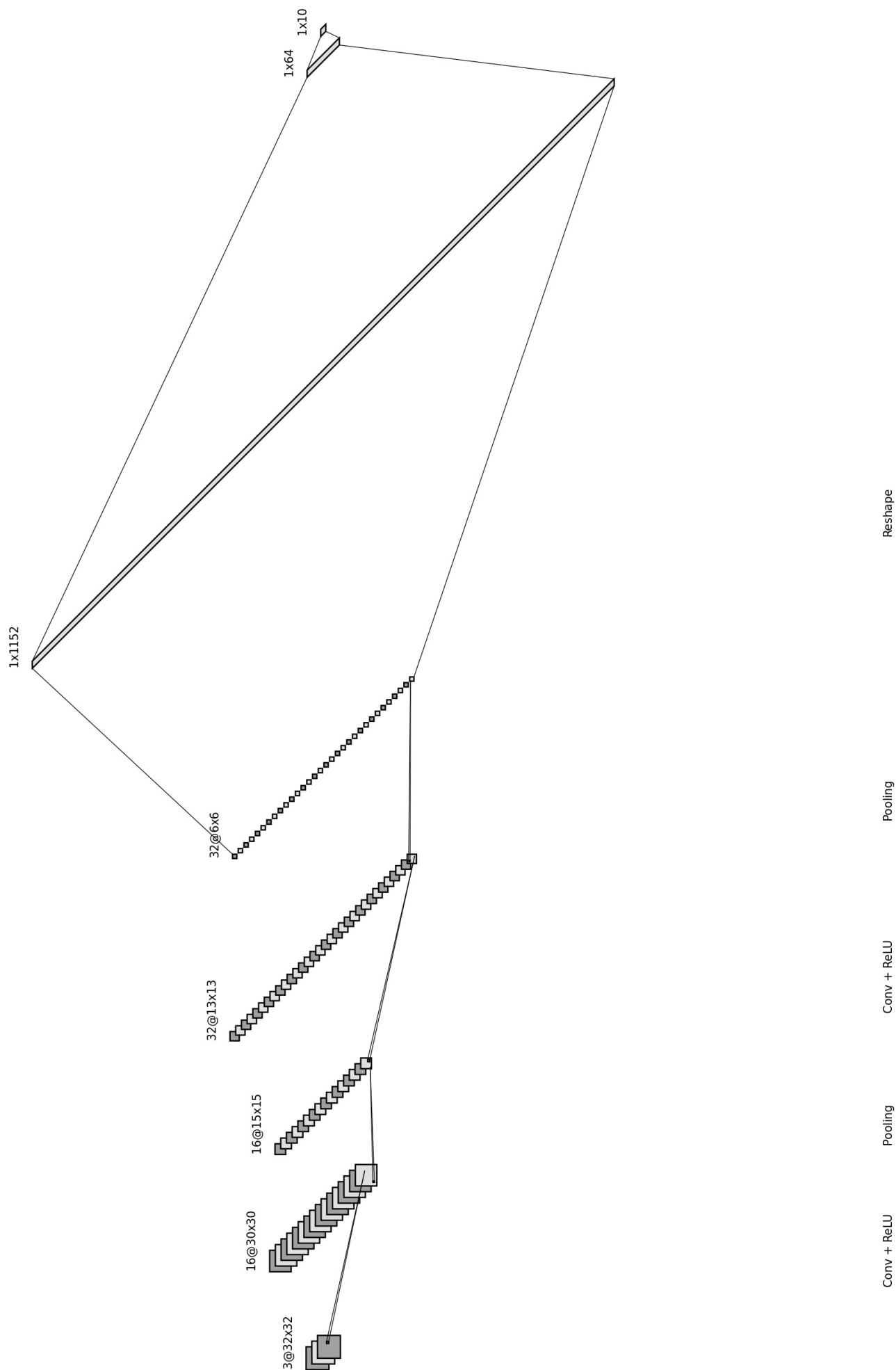
Gated		<p>             Accuracy of plane: 45.455              Accuracy of car: 26.087              Accuracy of bird: 29.31              Accuracy of cat: 16.071              Accuracy of deer: 1.961              Accuracy of dog: 33.871              Accuracy of frog: 74.286              Accuracy of horse: 12.281              Accuracy of ship: 26.984              Accuracy of truck: 74.648              Mean accuracy = 36.5           </p>
Spatial Pyramid + max		<p>             Accuracy of plane: 36.364              Accuracy of car: 45.652              Accuracy of bird: 13.793              Accuracy of cat: 5.357              Accuracy of deer: 39.216              Accuracy of dog: 35.484              Accuracy of frog: 54.286              Accuracy of horse: 47.368              Accuracy of ship: 36.508              Accuracy of truck: 45.07              Mean accuracy = 36.33333333           </p>
Stochastic		<p>             Accuracy of plane: 37.879              Accuracy of car: 36.957              Accuracy of bird: 15.517              Accuracy of cat: 1.786              Accuracy of deer: 1.961              Accuracy of dog: 43.548              Accuracy of frog: 78.571              Accuracy of horse: 24.561              Accuracy of ship: 42.857              Accuracy of truck: 53.521              Mean accuracy = 35.66666666           </p>
RAP		<p>             Accuracy of plane: 51.515              Accuracy of car: 30.435              Accuracy of bird: 12.069              Accuracy of cat: 0.0              Accuracy of deer: 19.608              Accuracy of dog: 37.097              Accuracy of frog: 61.429              Accuracy of horse: 15.789              Accuracy of ship: 42.857              Accuracy of truck: 54.93              Mean accuracy = 34.33333333           </p>

RWP		Accuracy of plane: 46.97 Accuracy of car: 43.478 Accuracy of bird: 15.517 Accuracy of cat: 0.0 Accuracy of deer: 0.0 Accuracy of dog: 35.484 Accuracy of frog: 65.714 Accuracy of horse: 35.088 Accuracy of ship: 17.46 Accuracy of truck: 52.113 Mean accuracy = 32.666666
RSP		Accuracy of plane: 42.424 Accuracy of car: 19.565 Accuracy of bird: 3.448 Accuracy of cat: 19.643 Accuracy of deer: 11.765 Accuracy of dog: 14.516 Accuracy of frog: 70.0 Accuracy of horse: 5.263 Accuracy of ship: 66.667 Accuracy of truck: 49.296 Mean accuracy = 32.333333

Бачимо, що навчання класифікувати CIFAR10 на невеликій кількості даних дає досить поганий результат загалом. Цікаво, що на цьому датасеті найкраще всіх навчається мережа із gated pooling. Тобто можемо зробити припущення, що gated і spatial pyramid є найбільш стійкими до проблеми масштабування. Зазначимо, що порівнюючи RAP і звичайне середнє об'єднання, робота рангового середнього є кращою у точності класифікування об'єктів. Порівнюючи роботу навчання мережі із stochastic pooling на MNIST і CIFAR10, несподіваним є те, що отримаємо кращий відносний результат під час навчання саме для CIFAR10. У цьому випадку стохастичне об'єднання навіть випереджає за точністю mixed. Цікаво, що графіки функції втрат під час навчання у RAP і average достатньо відрізняються.

Додатково проведено навчання моделей без стохастичної складової на всьому датасеті (таблиця 3). Так само як і під час навчання для MNIST, найкращий результат дає пірамідалне об'єднання, але не на стільки суттєвий. Несподіваним стає те, що gated дає трошки гірший результат, ніж mixed. Це означає, що mixed є більш залежним від кількості даних і дає кращий результат під час навчання для великого тренувального датасету. Зазначимо, що найгірші результати мають average і max, що підтверджує необхідність винайдення всіх додаткових pooling, які було розглянуто.

Рис. 13: Архітектура згорткової нейронної мережі для класифікування CIFAR10



**Табл. 3:** Порівняння роботи різних об'єднань із однаковою архітектурою мережі для повного датасету CIFAR10

Average	Max	Mixed
<hr/> Accuracy of plane: 45.3 Accuracy of car: 61.8 Accuracy of bird: 33.5 Accuracy of cat: 16.6 Accuracy of deer: 33.0 Accuracy of dog: 39.0 Accuracy of frog: 82.0 Accuracy of horse: 53.6 Accuracy of ship: 75.0 Accuracy of truck: 59.0 Mean accuracy = 49.88	Accuracy of plane: 43.5 Accuracy of car: 65.0 Accuracy of bird: 42.9 Accuracy of cat: 12.6 Accuracy of deer: 50.7 Accuracy of dog: 50.2 Accuracy of frog: 61.0 Accuracy of horse: 63.6 Accuracy of ship: 53.7 Accuracy of truck: 59.8 Mean accuracy = 50.3	Accuracy of plane: 64.3 Accuracy of car: 65.9 Accuracy of bird: 31.7 Accuracy of cat: 20.4 Accuracy of deer: 53.2 Accuracy of dog: 44.7 Accuracy of frog: 66.3 Accuracy of horse: 61.0 Accuracy of ship: 62.4 Accuracy of truck: 60.1 Mean accuracy = 53.0
Gated	Pyramid	RAP
<hr/> Accuracy of plane: 48.0 Accuracy of car: 61.9 Accuracy of bird: 39.2 Accuracy of cat: 25.4 Accuracy of deer: 38.0 Accuracy of dog: 48.9 Accuracy of frog: 63.4 Accuracy of horse: 53.8 Accuracy of ship: 64.9 Accuracy of truck: 56.8 Mean accuracy = 50.02999999999999	Accuracy of plane: 62.5 Accuracy of car: 72.8 Accuracy of bird: 28.0 Accuracy of cat: 38.6 Accuracy of deer: 53.3 Accuracy of dog: 47.1 Accuracy of frog: 46.1 Accuracy of horse: 60.2 Accuracy of ship: 64.7 Accuracy of truck: 67.6 Mean accuracy = 54.09	<hr/> Accuracy of plane: 52.0 Accuracy of car: 65.4 Accuracy of bird: 32.3 Accuracy of cat: 32.6 Accuracy of deer: 41.5 Accuracy of dog: 44.1 Accuracy of frog: 50.0 Accuracy of horse: 63.7 Accuracy of ship: 62.0 Accuracy of truck: 60.7 Mean accuracy = 50.43
RWP		
<hr/> Accuracy of plane: 55.6 Accuracy of car: 65.0 Accuracy of bird: 34.3 Accuracy of cat: 16.3 Accuracy of deer: 27.3 Accuracy of dog: 57.8 Accuracy of frog: 73.8 Accuracy of horse: 62.9 Accuracy of ship: 51.3 Accuracy of truck: 63.1 Mean accuracy = 50.739999		

## Згорткова нейронна мережа для fashion MNIST

Остання мережа навчатиметься класифікувати fashion MNIST. Даний датасет складається із чорно-білих малюнків одягу 10 типів (топ, штани, пуловер, плаття, пальто, босоніжки, сорочка, кросівки, сумка, ботильйони) розміром 28 на 28 пікселів. Загальний вигляд екземплярів зображено на рис. 14. Кожний малюнок подано у вигляді тензору розміру 28 на 28, де кожен елемент набуває значення від 0 до 1 завдяки нормуванню. Навчання відбуватиметься для 6400 екземплярів.

Завантажимо дані та створимо DataLoader, в якому кожний батч складатиметься з 4-ох екземплярів (Лістинг 12).

**Лістинг 12:** Завантаження fashion MNIST і створення DataLoader

```

transform = transforms.Compose(
    [transforms.ToTensor()])

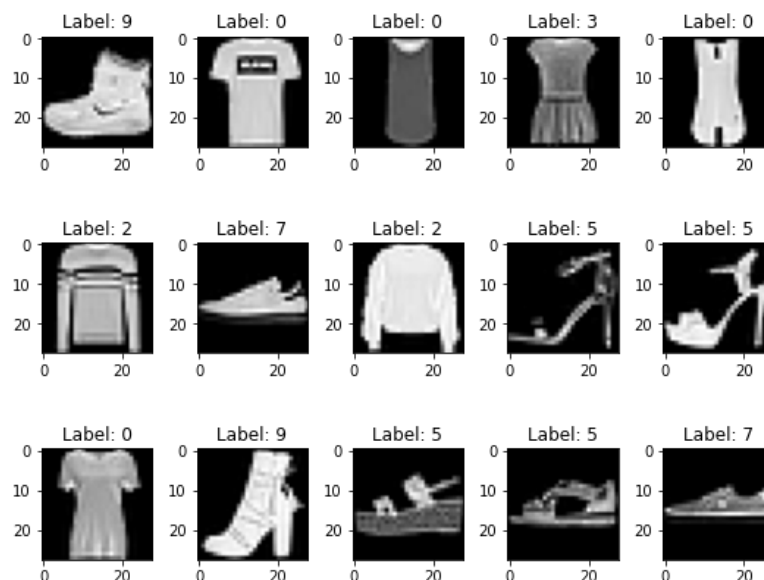
trainset = torchvision.datasets.FashionMNIST(root='./data', train=True,
download=True, transform=transform)

testset = torchvision.datasets.FashionMNIST(root='./data', train=False,
download=True, transform=transform)

trainset_sub = Subset(trainset, indices=range(6400))
testset_sub = Subset(testset, indices=range(1000))

trainloader = torch.utils.data.DataLoader(trainset_sub, batch_size=4,
shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset_sub, batch_size=4,
shuffle=True, num_workers=2)

```

**Рис. 14:** Приклад датасету fashion MNIST, [19, medium]

Дана мережа для класифікації fashion MNIST складається з 2 згорткових шарів, між якими будуть застосовані різні об'єднання, з 3 повнозв'язних лінійних шарів, з функцією активації ReLU, з функцією втрат CrossEntropy та з градієнтним спуском Adam (має таку ж архітектуру як і MNIST, рис. 11). Кількість епох дорівнюватиме 2. Отже, створимо загальний клас ЗНМ для fashion MNIST, який буде наслідуватися для порівняння об'єднань (Лістинг 13). Функції навчання і тестування відбуваються подібним чином як і в MNIST. Різниця лише в назвах класів при тестуванні.

**Лістинг 13:** Загальний вигляд ЗНМ для fashion MNIST: class fashionMnist

```

class model_fashionMNIST(nn.Module):
    def __init__(self):
        super(model_fashionMNIST, self).__init__()

```



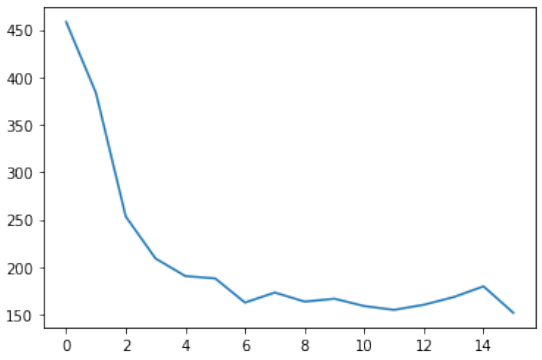
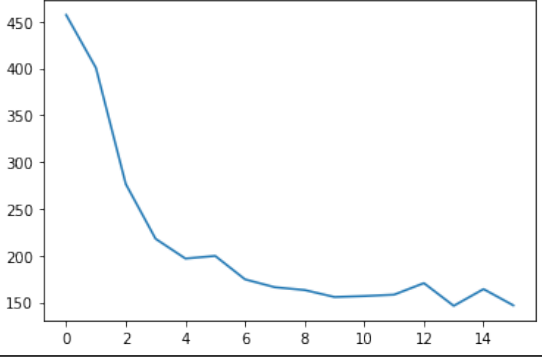
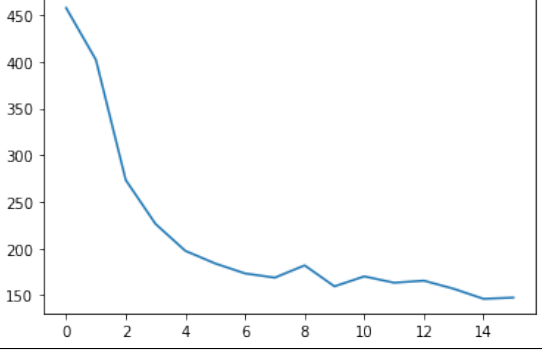
```

self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
self.fc1 = nn.Linear(4 * 4 * 16, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10)
self.loss_fn = nn.CrossEntropyLoss()
self.optimizer = torch.optim.Adam(self.parameters(), lr=1e-4)

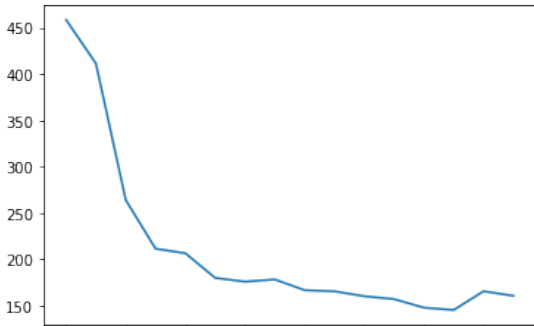
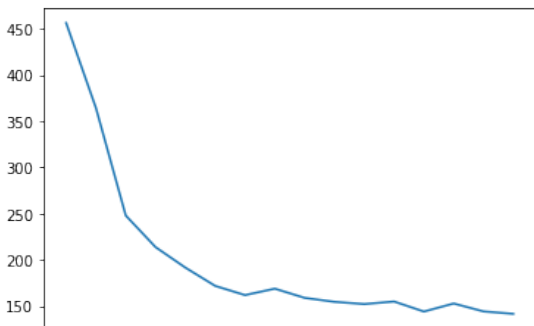
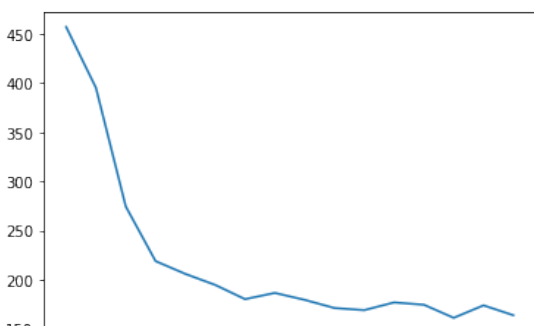
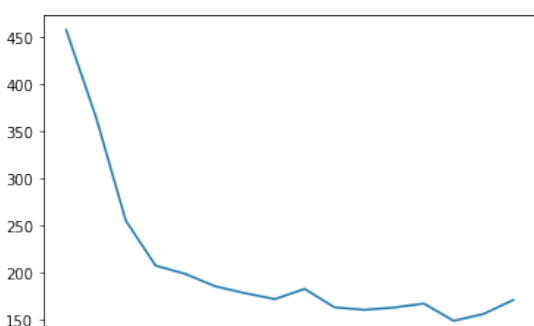
```

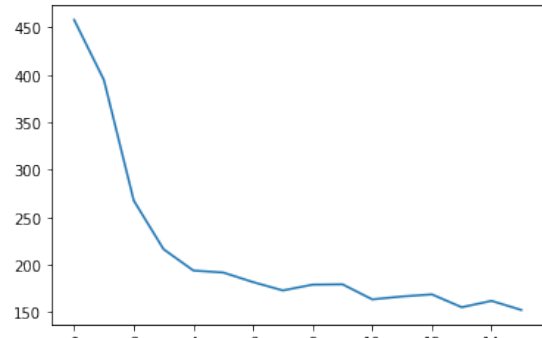
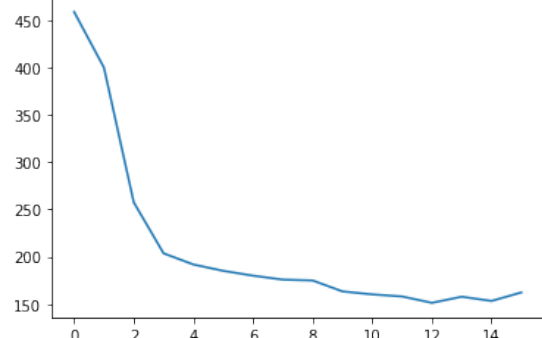
Отже, після навчання мережі з різними pooling, отримаємо результат на таблиці 4.

**Табл. 4:** Порівняння роботи різних об'єднань із однаковою архітектурою мережі для

Назва об'єднання	Графік функції втрат (навчання)	Точність моделі для кожного класу (тестування)
Average		Accuracy of T-shirt/top: 74.766 Accuracy of Trouser: 89.524 Accuracy of Pullover: 4.505 Accuracy of Dress: 64.516 Accuracy of Coat: 80.87 Accuracy of Sandal: 72.414 Accuracy of Shirt: 24.742 Accuracy of Sneaker: 74.737 Accuracy of Bag: 87.368 Accuracy of Ankle boot: 98.947 Mean accuracy = 66.7
Max		Accuracy of T-shirt/top: 80.374 Accuracy of Trouser: 94.286 Accuracy of Pullover: 54.054 Accuracy of Dress: 61.29 Accuracy of Coat: 70.435 Accuracy of Sandal: 74.713 Accuracy of Shirt: 12.371 Accuracy of Sneaker: 87.368 Accuracy of Bag: 88.421 Accuracy of Ankle boot: 91.579 Mean accuracy = 71.39999999999999
Mixed		Accuracy of T-shirt/top: 75.701 Accuracy of Trouser: 97.143 Accuracy of Pullover: 49.55 Accuracy of Dress: 67.742 Accuracy of Coat: 67.826 Accuracy of Sandal: 77.011 Accuracy of Shirt: 12.371 Accuracy of Sneaker: 87.368 Accuracy of Bag: 84.211 Accuracy of Ankle boot: 83.158 Mean accuracy = 70.0



Gated		<p>             Accuracy of T-shirt/top: 69.159              Accuracy of Trouser: 97.143              Accuracy of Pullover: 58.559              Accuracy of Dress: 79.57              Accuracy of Coat: 66.957              Accuracy of Sandal: 83.908              Accuracy of Shirt: 8.247              Accuracy of Sneaker: 85.263              Accuracy of Bag: 84.211              Accuracy of Ankle boot: 90.526              Mean accuracy = 72.0           </p>
Spatial Pyramid + max		<p>             Accuracy of T-shirt/top: 77.57              Accuracy of Trouser: 94.286              Accuracy of Pullover: 83.784              Accuracy of Dress: 66.667              Accuracy of Coat: 60.87              Accuracy of Sandal: 77.011              Accuracy of Shirt: 2.062              Accuracy of Sneaker: 80.0              Accuracy of Bag: 90.526              Accuracy of Ankle boot: 90.526              Mean accuracy = 72.39999999999999           </p>
Stochastic		<p>             Accuracy of T-shirt/top: 72.897              Accuracy of Trouser: 90.476              Accuracy of Pullover: 50.45              Accuracy of Dress: 53.763              Accuracy of Coat: 66.957              Accuracy of Sandal: 62.069              Accuracy of Shirt: 10.309              Accuracy of Sneaker: 86.316              Accuracy of Bag: 88.421              Accuracy of Ankle boot: 89.474              Mean accuracy = 67.10000000000001           </p>
RAP		<p>             Accuracy of T-shirt/top: 65.421              Accuracy of Trouser: 89.524              Accuracy of Pullover: 52.252              Accuracy of Dress: 78.495              Accuracy of Coat: 78.261              Accuracy of Sandal: 72.414              Accuracy of Shirt: 8.247              Accuracy of Sneaker: 85.263              Accuracy of Bag: 93.684              Accuracy of Ankle boot: 91.579              Mean accuracy = 71.3           </p>

RWP		Accuracy of T-shirt/top: 69.159 Accuracy of Trouser: 93.333 Accuracy of Pullover: 47.748 Accuracy of Dress: 63.441 Accuracy of Coat: 79.13 Accuracy of Sandal: 75.862 Accuracy of Shirt: 12.371 Accuracy of Sneaker: 86.316 Accuracy of Bag: 90.526 Accuracy of Ankle boot: 85.263 Mean accuracy = 70.19999999999999
RSP		Accuracy of T-shirt/top: 74.766 Accuracy of Trouser: 94.286 Accuracy of Pullover: 42.342 Accuracy of Dress: 61.29 Accuracy of Coat: 71.304 Accuracy of Sandal: 72.414 Accuracy of Shirt: 27.835 Accuracy of Sneaker: 83.158 Accuracy of Bag: 87.368 Accuracy of Ankle boot: 92.632 Mean accuracy = 70.5

Найкращий результат отримали мережа із gated і spatial pyramid pooling. Важливо зазначити, що звичайне стохастичне об'єднання дає гірші результати, аніж рангове. Але так відбувається лише під час навчання для fashion MNIST. З чого можемо зробити припущення, що проблема масштабування не завжди вирішується за допомогою обчислення ймовірностей за рангом. Порівняно з минулими мережами, в цій досить поганий результат дає mixed. На мою думку, це відбувається через те, що середнє об'єднання дає найгірший результат, а як відомо, mixed pooling об'єднує max і average. Досить суттєва різниця між роботою звичайного та рангового середнього об'єднання.

## 2.8 Загальне порівняння

Назва об'єднання	MNIST	CIFAR10	fashion MNIST
Max	82.5	33.5	71.4
Average	83.3	30.17	66.7
Mixed	83.5	34.5	70
Gated	84.6	36.5	72
Spatial pyramid + max	89.2	36.3	72.4
Stochastic	80.4	35.7	67
Rank-based average	82	34.3	71.3
Rank-based weighted	81	32.7	71.3

Rank-based stochastic	79.2	32.3	70.5
-----------------------	------	------	------

**Табл. 5:** Порівняння точності згорткових нейронних мереж із різними об'єднаннями під час тестування

Загалом бачимо на таблиці 5, що через високу чутливість до проблеми масштабування стохастичне об'єднання не дає високих результатів. Навіть рішення обраховування ймовірностей за допомогою рангів не дає бажаного результату на такій невеликій вибірці та на декількох епохах. З чого можна зробити висновок, що для використання рангового стохастичного об'єднання також необхідна більша кількість даних. Різниця чутливості цих методів до розміру вибірки потребує додаткових досліджень.

Маскове і звичайне змішане об'єднання загалом дають високий результат серед усіх моделей. Цікаво зазначити, що чим меншою є вибірка, тим більше gated випереджає максимальне і середнє за точністю. Також хочу згадати, що на більшій вибірці даний метод не випереджає навіть максимальне об'єднання. Із чого випливає, що цей метод завдяки параметру, що навчається, є менш чутливим до проблеми масштабування, але не досконалим для вирішення багатьох задач.

Серед рангових об'єднань випереджає для всіх вибірок саме середній. Можна зробити припущення, що цей метод є менш чутливим до масштабування, але не кращим у випадку великої вибірки та кількості епох. Проте точне визначення потребує додаткових досліджень.

Найкращий результат на великій та малій вибірці серед усіх має пірамідальне об'єднання. Загалом цей метод дає, як мінімум, одні з кращих результатів серед усіх моделей. Тому стає зрозумілим те, що це об'єднання набуває ширшого використання у згорткових нейронних мережах.

## Висновки

У роботі розглянуто як теоретично працюють mixed, gated, spatial/temporal pyramid, stochastic, rank-based average, rank-based weighted, rank-based stochastic і їх певні особливості. Також реалізовано ці методи на мові програмування Python з використанням бібліотеки pytorch.

Розглянуто 3 приклади застосування цих методів для згорткових нейронних мереж, що класифікують зображення (датасети: MNIST, CIFAR10, fashion MNIST). Також порівняно їх роботу в залежності від кількості даних для навчання. Перевірено на цих прикладах їх чутливість до проблеми масштабування.

## Список літератури

1. Geron, Aurelien. (2017). Hands-On Machine Learning with Scikit-Learn & TensorFlow : concepts, tools, and techniques to build intelligent systems . Beijing: O'Reilly.
2. Hamdan, Muhammad. (2018). VHDL auto-generation tool for optimized hardware acceleration of convolutional neural networks on FPGA (VGT).
3. Common Loss functions in machine learning, towardsdatascience
4. Adam optimizer, geeksforgeeks
5. Sharma, Shallu and Mehra, Rajesh. "Implications of Pooling Strategies in Convolutional Neural Networks: A Deep Insight" Foundations of Computing and Decision Sciences, vol.44, no.3, 2019, pp.303-330.
6. Average pooling, pytorch documentation
7. Gholamalinezhad, Hossein and Khosravi, Hossein. Pooling Methods in Deep Neural Networks, a Review. 2020, arXiv.
8. Yani M., Irawan S., Setianingsih C. (2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. Journal of Physics: Conference Series.
9. Max pooling, pytorch documentation
10. Example of the max pooling
11. Lee C.-Y., Gallagher P. W., Tu Z., Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree, in Artificial Intelligence and Statistics, 2016, 464-472.
12. Grauman K., Darrell T., The pyramid match kernel: Discriminative classification with sets of image features, in Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, 2005, 1458-1465.
13. He K., Zhang X., Ren S., Sun J., Spatial pyramid pooling in deep convolutional networks for visual recognition, in European conference on computer vision, 2014, 346-361.
14. Spatial pyramid pooling, tensorflow documentation

15. Zeiler M. D., Fergus R., Stochastic pooling for regularization of deep convolutional neural networks, arXiv preprint arXiv:1301.3557, 2013, 1-9.
16. Shi Z., Ye Y., Wu Y., Rank-based pooling for deep convolutional neural networks, Neural Networks, 83, 2016, 21-31.
17. Lim, Seung-Hwan & Young, Steven & Patton, Robert. (2016). An analysis of image storage systems for scalable training of deep neural networks.
18. Example image from CIFAR10 dataset
19. Example image from fashion MNIST dataset