Київський національний університет імені Тараса Шевченка

Факультет радіофізики, електроніки та комп’ютерних систем

Кафедра комп’ютерної інженерії

Класифікація картин українських митців за допомогою нейронних мереж

Кваліфікаційна робота бакалавра
студента напряму підготовки
123 «Комп'ютерна інженерія»

ЯВКІНА ДАНИЛА

_____(підпис)

Науковий керівник,
кандидат технічних наук,
асистент

ГАЛИНА СТРІЛЬЧУК

_____(підпис)

Рецензент

_____(підпис)

До захисту допускаю

Протокол засідання кафедри від

“___” _____2022р. №_____

Завідувач кафедри
к.ф.-м.н., доцент
ЮРІЙ БОЙКО

КИЇВ 2022

РЕФЕРАТ

Кваліфікаційна робота містить 33 с., 19 рис., 1 табл., 25 джерел.

Ключові слова: нейронна мережа, навчання нейронних мереж, класифікація нейронних мереж, конволюційні нейронні мережі, розробка нейронних мереж

Мета роботи:

1. Розробити підхід до аналізу зображень, заснований на навчанні нейронної мережі, враховуючи особливість зображень.

2. Зібрати відповідний набір даних для нейромережі, котрий містить картини різних відомих українських художників

3. Розробити мережу, яка може класифікувати твори українських художників за їх автором.

Інструменти розроблення: мова програмування Python, зокрема його бібліотека Tensorflow, інтерактивне середовище розробки Jupyter Notebook, програмно-апаратна архітектура паралельних обчислень для відеокарт Nvidia – CUDA, пакетний менеджер середовища розробки Conda.

Результати роботи: розглянуто різні підходи до аналізу зображень, обрано найбільш ефективний, зібрано 2 379 картин 12 українських художників, реалізовано мережу, яка з середньою точністю в 88% класифікує картину відповідно до її автора.

Зміст

ВСТУП

Обчислювальні системи, натхненні біологічними нейронними мережами для виконання різних завдань з величезною кількістю даних, називаються штучними нейронними мережами.

Нейронні мережі все більше й більше набувають популярність, їх все частіше й частіше обговорюють та прагнуть застосувати у своїх цілях, у своїх проектах. Хоча, не дивлячись на те, що така область досить відома, не зовсім зрозуміло що це таке, як ці мережі працюють та де їх можна використати.

Почнімо з того, у яких випадках нейронні мережі можуть бути корисними. Нейронні мережі чудово себе зарекомендували у вирішенні складних проблем в реальних ситуаціях. Вони можуть вивчати та моделювати нелінійні та складні зв’язки між входами та виходами; робити узагальнення та висновки; виявити приховані зв'язки, закономірності та передбачення; і моделювати дуже нестабільні дані і відхилення, необхідні для прогнозування рідкісних подій (наприклад, виявлення шахрайства). В результаті нейронні мережі можуть покращити процеси прийняття рішень у таких областях, як:

- Виявлення шахрайства з кредитними картками
- Оптимізація логістики транспортних мереж
- Розпізнавання символів і голосу, також відоме як обробка природної мови
- Лікарська діагностика та діагностика захворювань
- Цільовий маркетинг
- Фінансові прогнози щодо цін на акції, валюти, опціонів, ф'ючерсів, банкрутства та рейтингів облігацій

Отже нейронні мережі також можуть бути корисними і у випадку аналізу картин, їх класифікації по авторам, жанрам, періоду створення. За допомогою такого аналізу, наприклад, в подальшому можна автоматизовано створювати нові картини у певному жанрі чи стилі автора.

Розділ 1. Нейронні мережі. Конволюційна нейронна мережа

1.1 Що таке нейронні мережі?

Нейронна мережа — це такий тип машинного навчання, який моделює себе подібно до людського мозку. Він створює штучну нейронну мережу, яка дозволяє навчатися комп'ютеру за допомогою певних алгоритмів, включаючи нові дані.

Незважаючи на велику кількість алгоритмів штучного інтелекту, нейронні мережі можуть виконувати також і глибоке навчання (deep learning). У той час як основною одиницею мозку є нейрон, персептрон є основним будівельним блоком нейронної мережі. Він виконує просту обробку сигналів, а потім вони з'єднуються у велику сітчасту мережу.

Комп'ютер із нейронною мережею навчається виконувати завдання, тренуючись на навчальних даних. Досить популярним для нейронної мережі завданням є розпізнавання об'єктів, де нейронній мережі дають велику кількістю об'єктів певного типу, наприклад, собака, кішка або вуличний знак, і мережа, аналізуючи повторювані закономірності в представлених зображеннях, вчиться класифікувати нові зображення.

1.2 Конволюційна нейронна мережа (CNN)

CNN є найзрілішою формою глибоких нейронних мереж для отримання найточніших, тобто кращих результатів комп'ютерного зору. CNN складаються з шарів згорток, котрі створені за допомогою сканування пікселів зображень у наборі даних. Коли дані апроксимуються шар за шаром, CNN починає розпізнавати шаблони і, таким чином, розпізнавати об'єкти на зображеннях. Потім такі об'єкти застосовуються в різних програмах для ідентифікації, класифікації тощо.

Останні практики, такі як навчання з перенесенням у CNN, призвели до значного покращення неточності моделей. Перекладач Google і Google Lens – найсучасніші приклади CNN. Застосування CNN має експоненційний характер, тому що вони навіть застовуються для вирішення питань, які в першу чергу не стосуються комп'ютерного зору.

Згорткові нейронні мережі — це багатошарові нейронні мережі, які дійсно добре отримують функції з даних. Вони добре працюють із зображеннями і не потребують багатої попередньої обробки. Використання згортки та об'єднання для зведення зображення до його основних функцій, дає змогу правильно ідентифікувати зображення.

Легше навчати моделі CNN з меншою кількістю початкових параметрів, ніж з іншими видами нейронних мереж. Для цього не

знадобиться величезна кількість прихованих шарів, тому що згортки зможуть обробляти велику кількість прихованих шарів. Однією з цікавих речей CNN є кількість складних проблем, до яких вони можуть бути застосовані. Від самокерованих автомобілів до виявлення діабету, CNN можуть обробляти такі дані та надавати точні прогнози.

1.3 Архітектура конволюційних нейронних мереж (CNN)

Конволюційна нейронна мережа (CNN, рис 1.3.1) — це алгоритм глибокого навчання, котрий отримує вхідне зображення, визначає важливість для різних аспектів/об’єктів зображення та може відрізняти їх один від одного. Попередня обробка даних, котра необхідна для CNN, набагато нижча в порівнянні з іншими алгоритмами класифікації. На відміну від примітивних меодів, де фільтри створюються вручну, CNN мають можливість вивчати ці фільтри/характеристики.

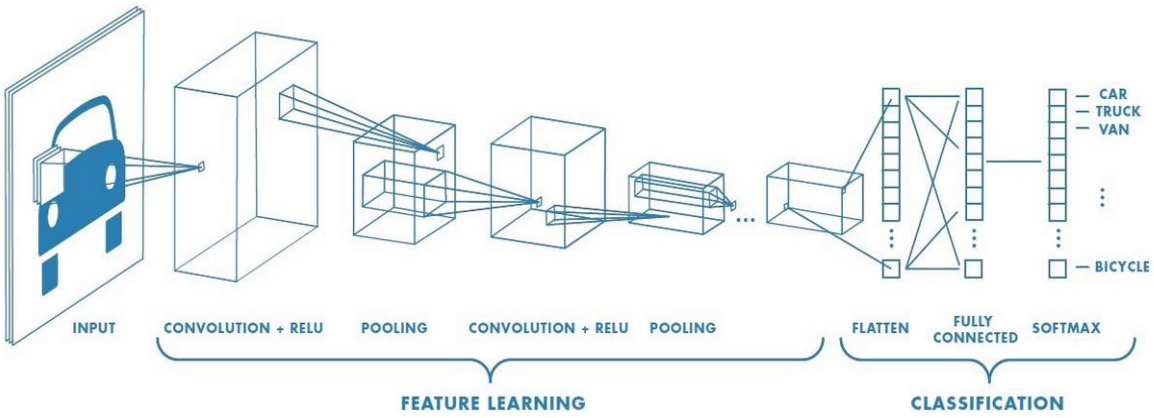
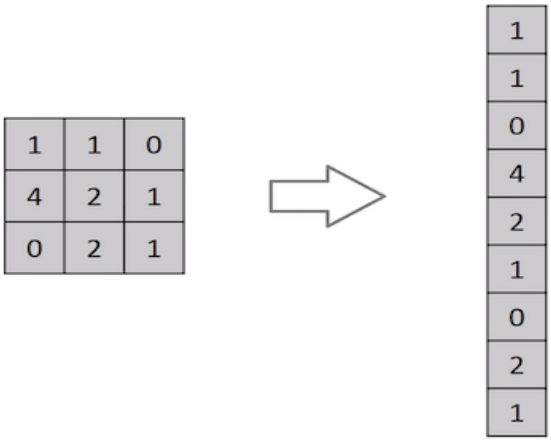


Рис 1.3.1. Архітектура Конволюційних нейроних мереж

Архітектура CNN аналогічна моделі зв'язку нейронів у людському мозку. Розробники була натхненні організацією зорової кори, де окремі нейрони реагують на подразники лише в обмеженій області поля зору. Набір таких полів перекривається, щоб охопити всю візуальну область.

Чому CNN замість нейронних мереж із прямим переміщенням?



Зображення - це матриця значень пікселів. Таким чином можна просто зрівняти зображення (наприклад перетворити 3x3 матрицю у вектор 9x1, рис 1.2.2) і, для класифікації, передати його багаторівневому персептрону.

Рис.1.3.2 Перетворення 3x3 матриці у 9x1 вектор

У випадках надзвичайно простих бінарних зображень метод може демонструвати середню оцінку точності при прогнозуванні класів, але не матиме практичної точності у випадках складних зображень, залежних від пікселів.

CNN може успішно фіксувати просторово-часові залежності в зображенні за умови застосування відповідних фільтрів. Завдяки зменшенню кількості задіяних параметрів і повторному використання ваг, архітектура краще відповідає набору даних зображення, тобто мережу навчається ліпше розуміти витонченість зображення.

1.3.1 Вхідне зображення

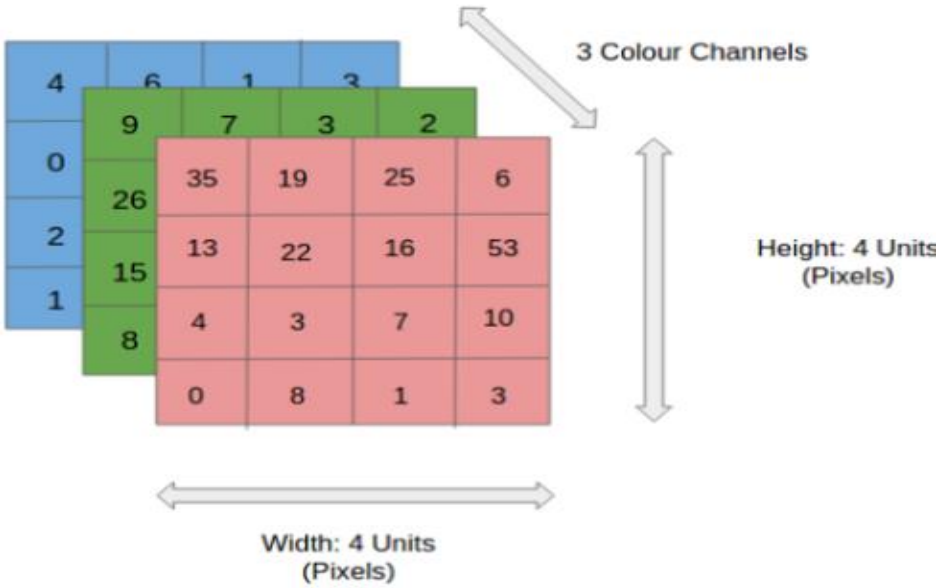


Рис 1.3.1.1 RGB зображення, з трьома площинами кольорів — синім, зеленим і червоним

На рисунку 1.3.1.1 маємо RGB зображення, яке розділене трьома площинами кольорів — синім, зеленим і червоним. Існує декілька просторів кольорів, де існують зображення, наприклад відтінки сірого, CMYK, HSV, RGB, тощо. Важко уявити, наскільки інтенсивними будуть обчислення, коли зображення досягнуть розмірів, скажімо, 8K (7680×4320). Коли ми хочемо розробити архітектуру, яка не тільки добре вивчає особливості, але й масштабується до масивних наборів даних, важливо зменшити зображення до форми, котру легше обробляти нейронній мережі без втрати критичних функцій отримання правильного прогнозу.

1.3.2 Шар згортки — ядро (Convolution Layer)

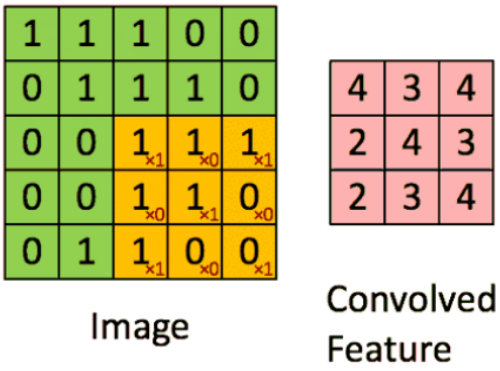


Рис1.3.2.1 Згортання зображення 5x5x1 з ядром 3x3x1, щоб отримати згорнуту функцію 3x3x1
Розміри зображення = 5(висота)x5 (ширина)x1 (кількість каналів, наприклад, RGB)

У наведеній вище демонстрації (рис 1.3.2.1) зелений розділ відображає вхідне зображення 5x5x1. Жовтим кольором позначено ядро/фільтр K, тобто елемент, котрий приймає участь у виконанні операції згортки в першій частині шару згортки.
K обрано як матрицю 3x3x1 (рис 1.3.2.2).

```
Kernel/Filter, K =

1   0   1
0   1   0
1   0   1
```

Рис1.3.2.2 Обрана матриця 3x3x1

Ядро зміщується 9 разів через довжину кроку = 1 (без стрибків). Кожної ітерації виконується операція множення матриці між K та частиною P зображення, над якою наведено ядро. Процес зміщення показаний на рис 1.3.2.3

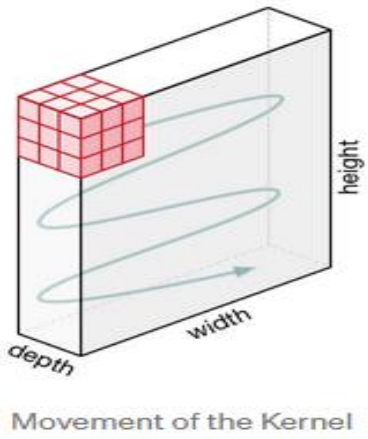


Рис1.3.2.3 Рух ядра по зображенню

Фільтр зміщується вправо з певним заданим кроком, поки не проаналізує повну ширину. Просуваючись далі, він стрибає вниз до початку (ліворуч) зображення з тим же заданим кроком і повторює той самий процес, до повного проходження всього зображення.

При зображені із декількома каналами (наприклад, RGB), ядро матиме таку ж глибину, що й у вхідного зображення. Між стеками Kn і In ([K1, I1]; [K2, I2]; [K3, I3]), відбувається операція множення матриці і всі результати підсумовуються зі зміщенням, щоб отримати здавлений вихід згорнутої характеристики каналу на одну глибину (рис 1.3.2.4).

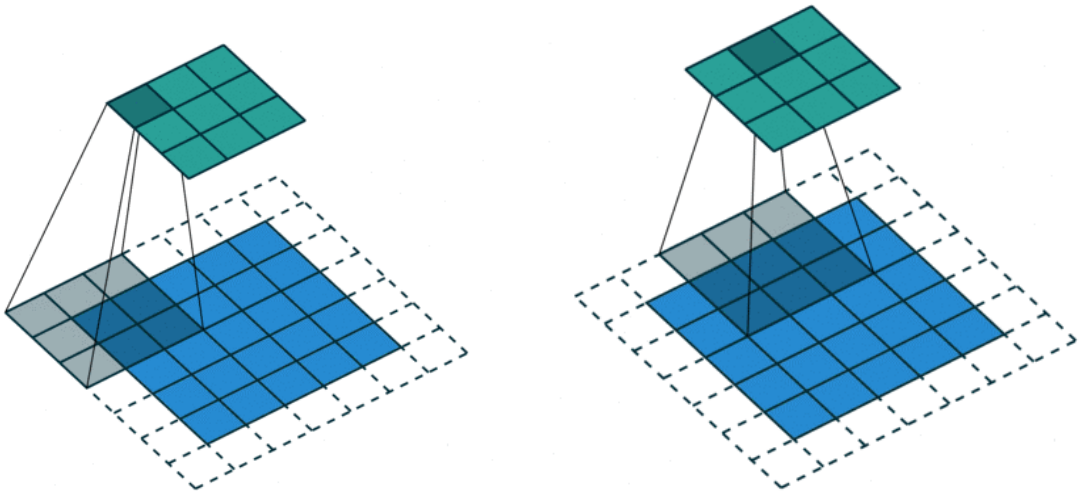


Рис 1.3.2.4 Операція згортки зі зміщенням = 2

Операцію згортки використовують для вилучення з вхідного зображення високорівневих функцій, наприклад краї. CNN не обов’язково обмежуються лише одним згортковим шаром.

Зазвичай перший шар згортки відповідає за захоплення низькорівневих функцій, тобто градація, колір і т.д. За допомогою доданих шарів архітектура також адаптується до функцій високого рівня, надаючи мережу, яка повністю розуміє зображення у заданому наборі даних.

Існує два типи результатів операції — один, у якому розмірність згорнутого елемента зменшується порівняно з вхідним, а в іншому збільшується або залишається незмінною. Для цього застосовуються Valid Padding або Same Padding у випадку зменшення та збільшення відповідно. Збільшуючи зображення 5x5x1 до 6x6x1, а потім застосовуючи до нього ядро 3x3x1, знаходимо, що згорнута матриця виявляється розміром 5x5x1. Звідси назва — Same Padding. Виконуючи таку ж операцію без заповнення, буде представлена матриця розміру ядра (3x3x1) — Valid Padding.

1.3.3 Об’єднаний шар (Pooling Layer)

Подібно до шару згортки, шар об’єднання (рис1.3.3.1) застосовується для зменшення розміру згорнутого елемента. Це зменшує обчислювальне навантаження, необхідне для обробки даних. Також він корисний для визначення домінуючих ознак, які є обертальними та позиційними інваріантами, таким чином він покращує процес навчання моделі.

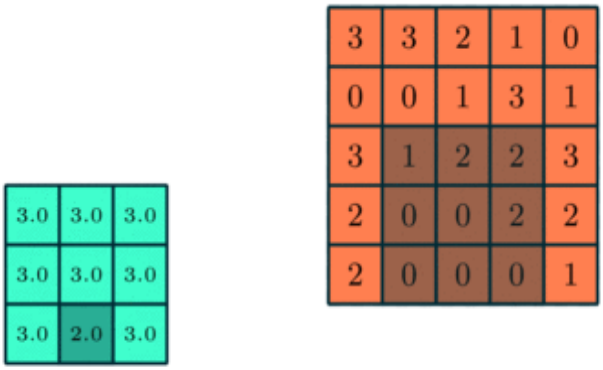


Рис 1.3.3.1 Об’єднання 3x3 через згорнуту функцію 5x5

Існує два типи об’єднання: максимальне об’єднання(Max Pooling) та середнє об’єднання(Average Pooling) (рис1.3.3.2). Max Pooling повертає матрицю максимальне значеннь з частин зображення, які були охоплені ядром. Тим часом, Average Pooling повертає матрицю середніх значеннь з частин зображення, які були охоплені ядром.

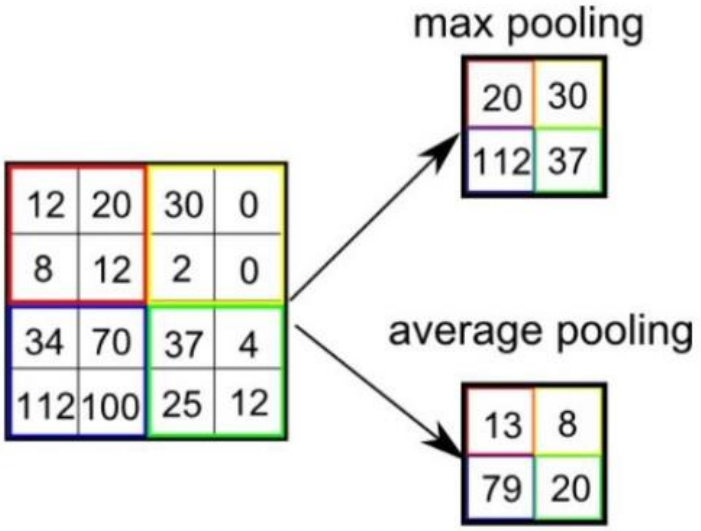


Рис 1.3.3.2 Максимальне та середнє об’єднання

Max Pooling повністю відкидає шумні активації, шляхом видалення шумів та зменшуючи розмірності. Тобто, Max Pooling заглушує шуми зображеня. Тим часом, Average Pooling просто зменшує розмірності для придушення шуму. Як результат, Max Pooling працює набагато краще.

Разом шар об’єднання і згортковий шар утворюють і-й шар CNN. Для ще більшого захоплення низькорівневих деталей кількість таких шарів може бути збільшена, залежно від складності зображень. Однак такі операції потребують більшу обчислювальну потужність. Отже, модель успішно включена для розуміння функцій.

1.3.4 Класифікація — повністю підключений рівень (FC шар, FC Layer)

Додавання повністю підключеного шару — це дешевий спосіб для вивчення нелінійних комбінацій високорівневих функцій, які показані результатом шару згортки. Повністю підключений рівень вивчає нелінійну функцію в цьому просторі.

Коли вхідне зображення перетворено у відповідну форму для багаторівневого персептрона, зрівняємо зображення у вектор стовпця. Зведений вихід подається в нейронну мережу і зворотне поширення застосовується до кожної ітерації навчання. Після ряду тренувань модель може розрізняти домінуючі ознаки в зображеннях і класифікувати їх за допомогою техніки Softmax Classification.

Розділ 2. Збір набору даних для нейронної мережі

2.1 Митці та їх твори

Обрано 12 митців і відповідно зібрано 2 379 картин.
У наступній таблиці можна побачити список митців та кількість їх картин у наборі даних.

	Митець	Кількість картин
0	Григорій Гавриленко	366
1	Іван Айвазовський	295
2	Давид Бурлюк	294
3	Олег Голосій	282
4	Казимир Малевич	238
5	Олександр Ройтбурд	231
6	Архип Куінджі	170
7	Марія Приймаченко	116
8	Дмитро Левицький	113
9	Соня Делоне	99
10	Євгенія Гапчинська	91
11	Тетяна Яблонська	84

Табл 2.1.1 Митці та кількість їх картин у наборі даних

Митців обрано таким чином, щоб у них як мінімум було 80 картин, щоб нейронна мережа мала достатню кількість даних для тренування. Таким чином, наприклад, такі митці, як Тарас Шевченко, Надія Білокінь, Іван Марчук, Микола Івасюк, на жаль, не потрапили у список.

2.2 Файлова структура зібраного набору даних

Файлова структура набору даних показана на рисунку 2.2.1:

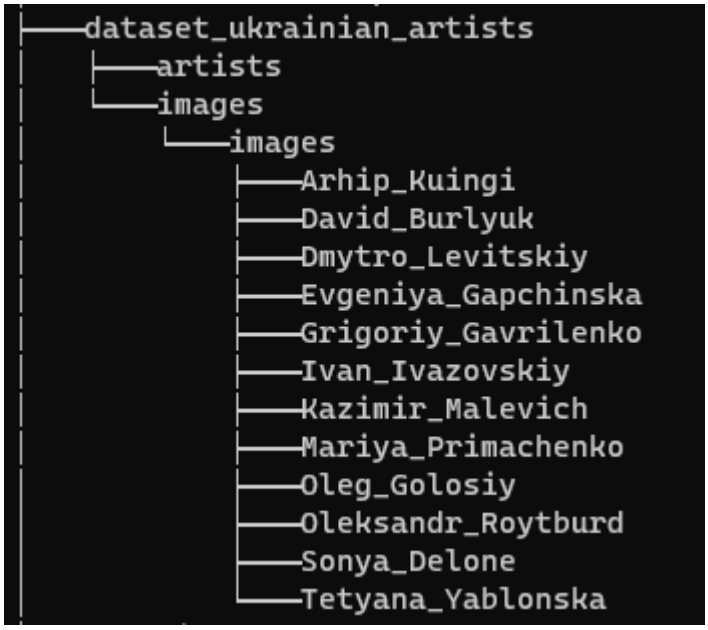


Рис 2.2.1 Файлова структура набору даних

У файлі artists зберігається таблиця у форматі .csv з переліченими митцями та кількістю їх творів.
У директорії images зберігаються файли з картинами у директоріях відповідних авторам.

Розділ 3. Розробка нейронної мережі

Заплановано розробити нейромережу, котра буде класифікувати картини відповідно до їх авторів. Подальшу реалізацію конволюційної нейронної мережі можна поділити на такі етапи:

1. Читання та обробка вхідних даних
2. Збільшення кількості вхідних даних (Data augmentation)
3. Побудова та тренування моделі
4. Тестування моделі

3.1 Читання та обробка вхідних даних

Зібрано 2 379 картин для 12 митців, так як у них кількість творів відрізняється, треба створити такий інструмент, який би урівнював митців між собою. Для вирішення цієї проблеми створений словник class_weights, який обраховується за наступною формулою:

$$\frac{\text{Кількість всіх картин}}{\text{Кількість митців} \times \text{Кількість картин митця}}$$

Значення class_weights для кожного митця показані на рисунку 3.1.1:

	name	paintings	class_weight
0	Grigoriy Gavrilenko	366	0.541667
1	Ivan Ivazovskiy	295	0.672034
2	David Burlyuk	294	0.674320
3	Oleg Golosiy	282	0.703014
4	Kazimir Malevich	238	0.832983
5	Oleksandr Roytburd	231	0.858225
6	Arhip Kuingi	170	1.166176
7	Mariya Primachenko	116	1.709052
8	Dmytro Levitskiy	113	1.754425
9	Sonya Delone	99	2.002525
10	Evgeniya Gapchinska	91	2.178571
11	Tetyana Yablonska	84	2.360119

Рис 3.1.1 Цінність митців для нейронної мережі відносно кількості їх картин

Прочитаємо вхідні дані та виведемо частину з них, дивитись рисунок 3.1.2:

Рис.3.1.2 Вхідні дані для нейронної мережі

3.2 Збільшення кількості вхідних даних (Data augmentation)

Коли є невелика кількість навчальних прикладів, модель іноді вчиться на шумах або небажаних деталях із навчальних прикладів — до такої міри, що це починає негативно впливати на точність моделі на нових прикладах. Це явище відоме як перетренування (overfitting). Це означає, що моделі буде важко узагальнювати новий набір даних.

Перетренування зазвичай відбувається при малій кількості початкових даних. Для вирішення цієї проблеми використано підхід збільшення даних, який називається data augmentation. Збільшення даних генерує додаткові тренувальні дані змінюючи кут нахилу зображень з початкового набору даних.

У випадку картин зміна кута нахилу на невелике значення погано вплине на зображення та може зменшити точність нейронної мережі. Тому збільшення набору даних здійснено за допомогою перевернутого зеркального зображення картин (рис 3.2.1).

Рис 3.2.1 Збільшення набору даних

Інший метод подолання вирішення явища перетренування полягає у тому, що для тренування за одну епоху не беруться усі нейрони відразу, а вилучається випадкова певна їх частина (10%,20%,40%), котра під час проходження цієї епохи не навчається. Такий підхід називається Dropout.

3.3 Побудова та тренування моделі

Для побудови моделі використана конволюційна нейронна мережа ResNet50, яка в своїй архітектурі має 50 рівнів. Також додано декілька рівнів поверх цієї моделі. Реалізація показана у Додатку А, Cell 8-11, с 26-30

Натренована модель має тренувальну точність 99,58 %, та перевірочну точність 88%. (див рис 3.3.1)

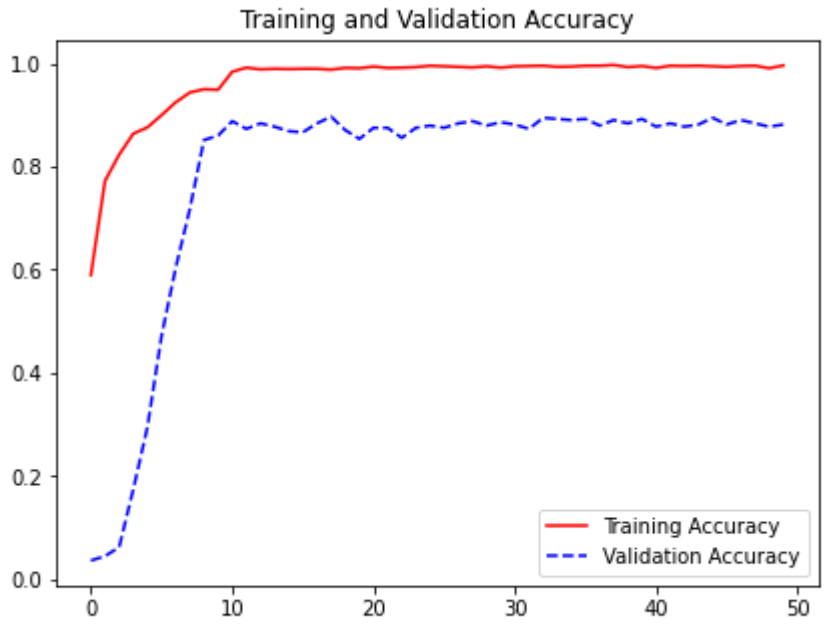


Рис 3.3.1 Графік тренувальної та перевірочної точності

Відсоток тренувальних та перевірочних втрат становлять 0.1725 та 0.4858 відповідно. (див рис 3.3.2)

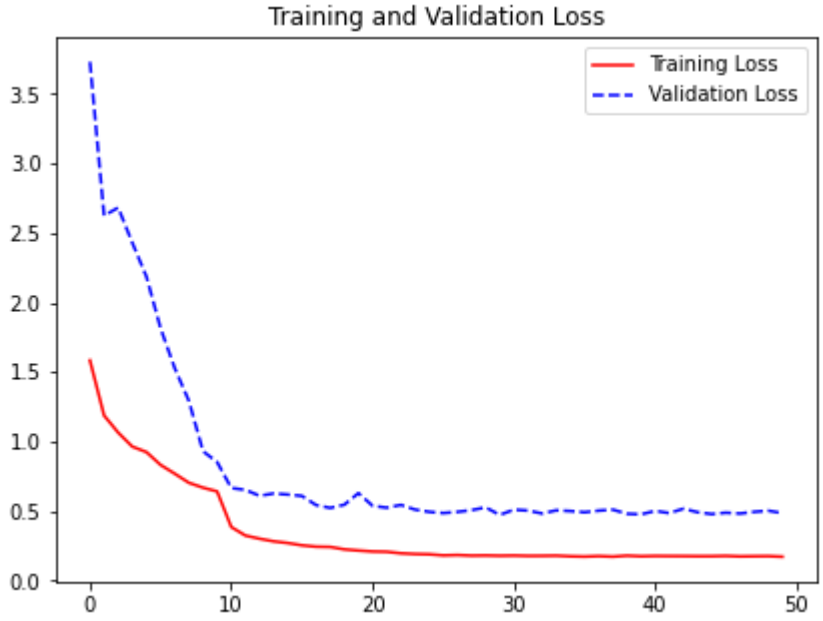


Рис 3.3.2 Графік тренувальних та перевірочних втрат

3.4 Тестування моделі

Побудована мережа має середнє значення точності 88%. Побудовану матрицю, де показана точність передбачення для кожного з митців, показано на рисунку 3.4.1.

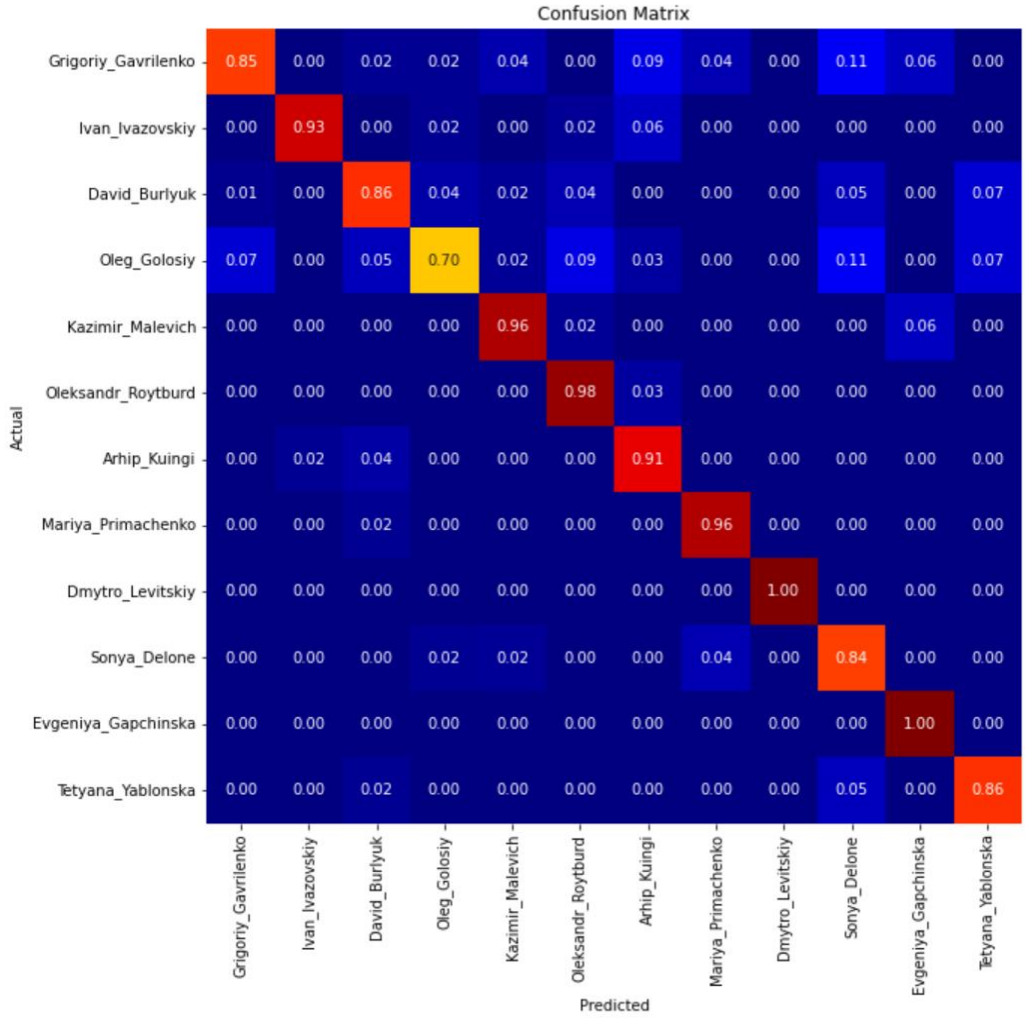


Рис 3.4.1 Матриця точності передбачення для кожного митця

Як видно з результатів, найменша точність виявилась для картин Олега Голосія, найбільша – для Євгенії Гапчинської та Дмитра Левицького.

Також взято випадкові картини з набору даних та перевірено у нейронній мережі:

Рис 3.4.2 Приклад роботи нейронної мережі для картин Архипа Куінджі та Євгенії Гапчінської

Рис 3.4.3 Приклад роботи нейронної мережі для картин Євгенії Гапчінської та Івана Айвазовського

Також нейронна мережа може обробляти зображення по посиланню з інтернету.

Взято зображення за посиланням:

https://uploads4.wikiart.org/images/ivan-aivazovsky/the-tempest-1899-1.jpg!PinterestSmall.jpg

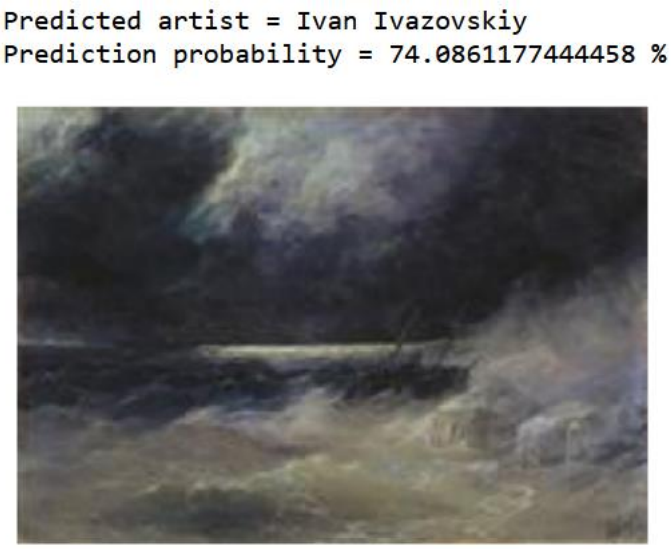


Рис 3.4.4 Приклад роботи нейронної мережі для картини Івана Айвазовського

Висновок

Розглянуто різні підходи до аналізу зображень за допомогою нейронних мереж. Як найбільш ефективний підхід, для подальшої розробки обрано конволюційні нейронні мережі. Описано архітектуру конволюційних мереж.

Зібрано набір даних, який включає в себе 2 379 картини 12 українських митців.

Розроблено нейронну мережу, яка з середньою точністю 88% класифікує вхідні дані. Як вхідні данні мережа може приймати випадкові картини з зібранного набору даних або посилання на картину в інтернеті.

Загалом, точність 88% - це високий показник як для класифікації картин. Менша точність в деяких митців зумовлена тим, що вони мають більшу кількість картин без яскраво вираженого унікального стилю. Такими можуть бути пейзажі, портрети і т.д.. Досить схожими є тематики та стилі картин Давида Бурлюка та Тетяни Яблонської. Передивлюючись набори даних для обох митців вирізняється те, що вони мають подібні погляди на відображення природи, натюрмортів і портретів.

В той же час, митці з особливим стилем, в яких картини одразу вирізняються з-поміж інших, мають більшу точність. Наприклад, картини Євгенії Гапчинської визначаються нейромережою з точністю у 100%, адже їй характерно малювати маленьку дитину з округлим лицем у різних експозиціях. Але ідеальна точність класифікації виявилась для робіт Дмитра Левицького, адже всі його роботи – це портрети на однаковому фоні, в однакових умовах. На відміну від інших, у цьому випадку нейромережа жодного разу не помилилась класифікувавши картину іншого митця як картину Левицького.

ПЕРЕЛІК ПОСИЛАНЬ

1. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.
2. A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks.” in Advances in Neural Information Processing Systems 25, 2012, pp. 1097–1105.
3. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” ArXiv: 1512.03385, vol. 7, no. 3, pp. 171–180, dec 2015.
4. Y. Gong, L. Wang, R. Guo, and S. Lazebnik, “Multi-scale Orderless Pooling of Deep Convolutional Activation Features,” ArXiv:1403.1840, pp. 1–17, mar 2014.
5. Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” The handbook of brain theory and neural networks, vol. 3361, pp. 255–258, 1995.
6. S. R. Kheradpisheh, M. Ghodrati, M. Ganjtabesh, and T. Masquelier, “Deep Networks Resemble Human Feed-forward Vision in Invariant Object Recognition,” ArXiv: 1508.03929, 2015.
7. Y. Xu, T. Xiao, J. Zhang, K. Yang, and Z. Zhang, “Scale-Invariant Convolutional Neural Networks,” ArXiv: 1411.6369, 2014.
8. C. Garcia and M. Delakis, “Convolutional face finder: A neural architecture for fast and robust face detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 11, pp. 1408–1423, 2004.
9. P. Sermanet and Y. Lecun, “Traffic sign recognition with multi-scale convolutional networks,” Proceedings of the International Joint Conference on Neural Networks, no. SEPTEMBER 2011, pp. 2809–2813,2011.
10. S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell,A. Hertzmann, and H. Winnemoeller, “Recognizing Image Style,”ArXiv: 1311.3715, pp. 1–20, nov 2013.
11. B. Saleh and A. Elgammal, “Large-scale Classification of Fine-Art Paintings: Learning The Right Metric on The Right Feature,” arXiv preprint arXiv:1505.00855, p. 21, 2015.
12. https://www.educba.com/what-is-neural-networks
13. https://www.techradar.com/news/what-is-a-neural-network
14. https://medium.com/fintechexplained/understanding-neural-networks-98e94251fb97
15. https://www.educba.com/classification-of-neural-network/
16. https://www.freecodecamp.org/news/convolutional-neural-network-tutorial-for-beginners/
17. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
18. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
19. https://www.tensorflow.org/tutorials/images/classification
20. https://www.linkedin.com/learning/deep-learning-image-recognition/
21. https://keras.io/api/applications/resnet/#resnet50-function
22. https://www.mathworks.com/help/deeplearning/ref/resnet50.html;jsessionid=d54ccbc21139bea8ea6fc4e14784
23. https://www.wikiart.org/uk/
24. https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time
25. https://drive.google.com/file/d/1jjGri9gywWvumD1Iwhxf_W5qvGtwyRM8/view?usp=sharing

Додаток

Додаток А

Код реалізації нейронної мережі:

Cell 1:

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
import os
from tqdm import tqdm, tqdm_notebook
import random

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.applications import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.initializers import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from numpy.random import seed
seed(1)
from tensorflow import random as rand
rand.set_seed(1)
```

Cell 2:

```
print(os.listdir("dataset_ukrainian_artists"))
artists = pd.read_csv('dataset_ukrainian_artists/artists.csv')
artists.shape
```

Cell 3:

```
# Sort artists by number of paintings

artists = artists.sort_values(by=['paintings'], ascending=False)


# Create a dataframe with artists

artists_top = artists[artists['paintings']].reset_index()

artists_top = artists_top[['name', 'paintings']]

artists_top['class_weight'] = artists_top.paintings.sum() / (artists_top.shape[0] *
artists_top.paintings)

artists_top
```

Cell 4:

```
class_weights = artists_top['class_weight'].to_dict()

class_weights
```

Cell 5:

```
# Print few random paintings

n = 5

fig, axes = plt.subplots(1, n, figsize=(20,10))


for i in range(n):

    random_artist = random.choice(artists_top_name)

    random_image = random.choice(os.listdir(os.path.join(images_dir, random_artist)))

    random_image_file = os.path.join(images_dir, random_artist, random_image)

    image = plt.imread(random_image_file)

    axes[i].imshow(image)

    axes[i].set_title("Artist: " + random_artist.replace('_', ' '))

    axes[i].axis('off')


plt.show()
```

Cell 6:

```
# Augment data
batch_size = 16
train_input_shape = (224, 224, 3)
n_classes = artists_top.shape[0]

train_datagen = ImageDataGenerator(validation_split=0.2,
                                   rescale=1./255.,
                                   #rotation_range=45,
                                   #width_shift_range=0.5,
                                   #height_shift_range=0.5,
                                   shear_range=5,
                                   #zoom_range=0.7,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                  )

train_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                   target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    subset="training",
                                                    shuffle=True,
                                                    classes=artists_top_name.tolist()
                                                   )

valid_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                   target_size=train_input_shape[0:2],
                                                    batch_size=batch_size,
                                                    subset="validation",
                                                    shuffle=True,
                                                    classes=artists_top_name.tolist()
                                                   )
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
```

```
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print("Total number of batches =", STEP_SIZE_TRAIN, "and", STEP_SIZE_VALID)
```

Cell 7:

```
# Print a random paintings and it's random augmented version
fig, axes = plt.subplots(1, 2, figsize=(20,10))

random_artist = random.choice(artists_top_name)
random_image = random.choice(os.listdir(os.path.join(images_dir, random_artist)))
random_image_file = os.path.join(images_dir, random_artist, random_image)

# Original image
image = plt.imread(random_image_file)
axes[0].imshow(image)
axes[0].set_title("An original Image of " + random_artist.replace('_', ' '))
axes[0].axis('off')

# Transformed image
aug_image = train_datagen.random_transform(image)
axes[1].imshow(aug_image)
axes[1].set_title("A transformed Image of " + random_artist.replace('_', ' '))
axes[1].axis('off')

plt.show()
```

Cell 8:

```
# Load pre-trained model
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=train_input_shape)

for layer in base_model.layers:
    layer.trainable = True

# Add layers at the end
```

```
X = base_model.output
X = Flatten()(X)

X = Dense(512, kernel_initializer='he_uniform')(X)
X = Dropout(0.5)(X)
X = BatchNormalization()(X)
X = Activation('relu')(X)

X = Dense(16, kernel_initializer='he_uniform')(X)
X = Dropout(0.5)(X)
X = BatchNormalization()(X)
X = Activation('relu')(X)

output = Dense(n_classes, activation='softmax')(X)

model = Model(inputs=base_model.input, outputs=output)

optimizer = Adam(lr=0.0001)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

n_epoch = 10

early_stop = EarlyStopping(monitor='val_loss', patience=20, verbose=1,
                           mode='auto', restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,
                              verbose=1, mode='auto')
```

Cell 9:

```
# Train the model - all layers
history1 = model.fit(train_generator, steps_per_epoch=STEP_SIZE_TRAIN,
                     validation_data=valid_generator,
validation_steps=STEP_SIZE_VALID,
                     epochs=n_epoch,
                     shuffle=True,
                     verbose=1,
                     callbacks=[reduce_lr],
                     use_multiprocessing=False,
                     workers=16,
                     class_weight=class_weights
                    )
```

Cell 10:

```
#train
for layer in model.layers:
    layer.trainable = False

for layer in model.layers[:50]:
    layer.trainable = True

optimizer = Adam(lr=0.0001)

model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

n_epoch = 50
history2 = model.fit(train_generator, steps_per_epoch=STEP_SIZE_TRAIN,
                              validation_data=valid_generator,
validation_steps=STEP_SIZE_VALID,
                              epochs=n_epoch,
```

```
    shuffle=True,
    verbose=1,
    callbacks=[reduce_lr, early_stop],
    use_multiprocessing=False,
    workers=16,
    class_weight=class_weights
)
```

Cell 11:

```
# Merge history1 and history2
history = {}
history['loss'] = history1.history['loss'] + history2.history['loss']
history['acc'] = history1.history['accuracy'] + history2.history['accuracy']
history['val_loss'] = history1.history['val_loss'] + history2.history['val_loss']
history['val_acc'] = history1.history['val_accuracy'] +
history2.history['val_accuracy']
history['lr'] = history1.history['lr'] + history2.history['lr']

# Plot the training graph
def plot_training(history):
    acc = history['acc']
    val_acc = history['val_acc']
    loss = history['loss']
    val_loss = history['val_loss']
    epochs = range(len(acc))

    fig, axes = plt.subplots(1, 2, figsize=(15,5))

    axes[0].plot(epochs, acc, 'r-', label='Training Accuracy')
```

```
    axes[0].plot(epochs, val_acc, 'b--', label='Validation Accuracy')
    axes[0].set_title('Training and Validation Accuracy')
    axes[0].legend(loc='best')

    axes[1].plot(epochs, loss, 'r-', label='Training Loss')
    axes[1].plot(epochs, val_loss, 'b--', label='Validation Loss')
    axes[1].set_title('Training and Validation Loss')
    axes[1].legend(loc='best')

    plt.show()

plot_training(history)
```

Cell 12:

```
# Prediction accuracy on train data
score = model.evaluate(train_generator, verbose=1)
print("Prediction accuracy on train data =", score[1])

# Prediction accuracy on CV data
score = model.evaluate(valid_generator, verbose=1)
print("Prediction accuracy on CV data =", score[1])
```

Cell 13:

```
# Classification report and confusion matrix
from sklearn.metrics import *
import seaborn as sns

tick_labels = artists_top_name.tolist()

def showClassficationReport_Generator(model, valid_generator, STEP_SIZE_VALID):
```

```
    # Loop on each generator batch and predict
    y_pred, y_true = [], []
    for i in range(STEP_SIZE_VALID):
        (X,y) = next(valid_generator)
        y_pred.append(model.predict(X))
        y_true.append(y)

    # Create a flat list for y_true and y_pred
    y_pred = [subresult for result in y_pred for subresult in result]
    y_true = [subresult for result in y_true for subresult in result]

    # Update Truth vector based on argmax
    y_true = np.argmax(y_true, axis=1)
    y_true = np.asarray(y_true).ravel()

    # Update Prediction vector based on argmax
    y_pred = np.argmax(y_pred, axis=1)
    y_pred = np.asarray(y_pred).ravel()

    # Confusion Matrix
    fig, ax = plt.subplots(figsize=(10,10))
    conf_matrix = confusion_matrix(y_true, y_pred, labels=np.arange(n_classes))
    conf_matrix = conf_matrix/np.sum(conf_matrix, axis=1)
    sns.heatmap(conf_matrix, annot=True, fmt=".2f", square=True, cbar=False,
                cmap=plt.cm.jet, xticklabels=tick_labels, yticklabels=tick_labels,
                ax=ax)
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
    ax.set_title('Confusion Matrix')
    plt.show()

    print('Classification Report:')
    print(classification_report(y_true, y_pred, labels=np.arange(n_classes),
target_names=artists_top_name.tolist()))
```

```
showClassficationReport_Generator(model, valid_generator, STEP_SIZE_VALID)
```

Cell 14:

```
# Prediction
from tensorflow.keras.preprocessing import *

n = 5
fig, axes = plt.subplots(1, n, figsize=(25,10))

for i in range(n):
    random_artist = random.choice(artists_top_name)
    random_image = random.choice(os.listdir(os.path.join(images_dir, random_artist)))
    random_image_file = os.path.join(images_dir, random_artist, random_image)

    # Original image

    test_image = image.load_img(random_image_file,
target_size=(train_input_shape[0:2]))

    # Predict artist
    test_image = image.img_to_array(test_image)
    test_image /= 255.
    test_image = np.expand_dims(test_image, axis=0)

    prediction = model.predict(test_image)
    prediction_probability = np.amax(prediction)
    prediction_idx = np.argmax(prediction)

    labels = train_generator.class_indices
    labels = dict((v,k) for k,v in labels.items())

    title = "Actual artist = {}\nPredicted artist = {}\nPrediction probability =
{:.2f} %" \
                .format(random_artist.replace('_', ' '),
labels[prediction_idx].replace('_', ' '),
```

```
                        prediction_probability*100)

    # Print image
    axes[i].imshow(plt.imread(random_image_file))
    axes[i].set_title(title)
    axes[i].axis('off')

plt.show()
```

Cell 15:

```
# Predict from url
url = 'https://uploads4.wikiart.org/images/ivan-aivazovsky/the-tempest-1899-
1.jpg!PinterestSmall.jpg'

import imageio
import cv2

web_image = imageio.imread(url)
web_image = cv2.resize(web_image, dsize=train_input_shape[0:2], )
web_image = image.img_to_array(web_image)
web_image /= 255.
web_image = np.expand_dims(web_image, axis=0)



prediction = model.predict(web_image)
prediction_probability = np.amax(prediction)
prediction_idx = np.argmax(prediction)

print("Predicted artist =", labels[prediction_idx].replace('_', ' '))
print("Prediction probability =", prediction_probability*100, "%")

plt.imshow(imageio.imread(url))
plt.axis('off')
plt.show()
```