ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

До бакалаврської роботи

На ступінь вищої освіти бакалавр

на тему: **«Технологія обробки зображень з використанням штучного інтелекту»**

Виконав: студент 4 курсу, групи ППЗ–52

спеціальності

121_Інженерії програмного забезпечення_

(шифр і назва спеціальності)

_____Ритов А.А_____

(прізвище та ініціали)

Керівник _Коваленко Д.О._____

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____О.В. Негоденко

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

1.Тема роботи: « **Технологія обробки зображень з використанням штучного інтелекту**».

Керівник роботи: Коваленко Д.О.,

затверджені наказом вищого навчального закладу від “12.03” 2021 року №65 .

2.Строк подання студентом роботи 01.06.2021

3.Вхідні дані для роботи:

3.2 Теорія цифрової обробки зображень

3.2 Методи та алгоритми цифрової обробки зображень

3.3 Проектування програмного рішення.

3.4 Структура та реалізація застосунку.

3.6 Тестування розробленого засобу.

4.1 Перелік обов'язкових слайдів презентації:

4.2 Визначення цифрової обробки зображень

4.3 Основні методи цифрової обробки

4.4 Згорткова нейронна мережа

4.5 Для чого застосовують згорткові нейронні мережі

4.6 Висновок

6. Дата видачі завдання 19.04.2021

6. Календарний план-графік

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.21	Виконано
2	Дослідження положення побудови	25.04.21	Виконано
3	Аналіз методів побудови	01.05.21	Виконано
4	Розробка моделі CNN	15.05.21	Виконано
5	Висновки, оформлення роботи	19.05.21	Виконано
6	Розробка демонстраційних матеріалів	20.05.21	Виконано
7	Здача роботи	24.05.21	Виконано

Студент Ритов А.А.

Керівник роботи Коваленко Д.О.

РЕФЕРАТ

Текстова частина роботи 81 стр., 32 рис., 13 табл., 34 джерел

Технологія обробки зображень з використанням штучного інтелекту

Об'єктом дослідження - є нейронні мережі та алгоритми їх побудови.

Мета роботи – розробка системи CNN.

Предмет дослідження – є існуючі алгоритми та структури даних нейронних мереж, їх ефективність в умовах неповної або відсутньої інформації.

Методи дослідження. У науковій роботі було використані загальнонаукові та спеціальні методи дослідження, а саме: методи аналізу та синтезу, індукції та порівняння під час дослідження досвіду впровадження нейронних мереж та побудови списку рекомендацій, метод єдності історичного та логічного – для відображення етапів впровадження нейронних мереж, методів машинного навчання.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ШІ (AI) – штучний інтелект,

МН (ML) – машинне навчання,

DL – глибоке навчання,

CNN – convolutional neural networks,

GAN – generative adversarial networks,

SOTA – state-of-the-art,

SR – super-resolution,

SISR – single image super-resolution,

SSIM – structural similarity measure,

PNSR – peak signal-to-noise ratio,

UIQM - underwater image quality measure,

RoI – region of interests,

RDB – residual dense blocks,

FENet – feature extraction network,

RGHS - relative global histogram stretching

UCM - unsupervised color correction,

MS-Fusion - multiscale fusion,

MSRetinex - multi-scale Retinex,

SESR - simultaneous enhancement and super-resolution.

ВСТУП

Цифрова фотографія надзвичайно просунулася за останні роки завдяки постійному вдосконаленню датчиків камери та конвеєрів для обробки сигналів зображення. Однак у міру прогресу зроблені фотографії все ще можуть бути неякісними через різні фактори, зокрема стан сцени, погане освітлення або майстерність фотографа. Ретушери людських зображень часто покращують естетичну якість цифрових фотографій за допомогою ручного регулювання. За допомогою професійного програмного забезпечення (наприклад, Photoshop, Lightroom) ви можете вносити різні зміни як за допомогою інтерактивних, так і напівавтоматичних інструментів.

На додаток до загальних загальних налаштувань, таких як посилення контрасту та освітлення, розширена функція редагування також доступна через локальні налаштування зображення. Однак ручне вдосконалення залишається проблемою для непрофесіоналів, які можуть не мати належних навичок, часу чи естетичного судження, щоб ефективно покращити свої образи.

Обробка зображень - це будь-яка форма обробки інформації, де вхідні дані представлені зображенням, а результат також є зображенням.

Завданням обробки може бути як поліпшення зображення за певним критерієм, так і радикальна трансформація та кардинальна зміна зображення. Для оцифрованих зображень (фото, відео) та зображень, отриманих за допомогою комп’ютерної візуалізації, використовуються різні методи. Під час оцифровки ви можете помітити шум на зображенні.

Важливою та нагальною проблемою залишається відновлення фотографій після оцифрування, тобто все ще існує необхідність у видаленні всіляких шумів, подряпин та інших дефектів. завдання відновлення зображення - знайти більш повні риси оригінального об'єкта на спостережуваному зображенні. На жаль, не існує такого універсального методу, який вирішив би цю проблему, і вибір методів рішення безпосередньо залежить від того, що буде включено в завдання обробки.

РОЗДІЛ 1. ЦИФРОВА ОБРОБКА ЗОБРАЖЕНЬ

1.1 Теорія цифрових зображень

Що таке зображення? Відповісти на це питання можна кількома способами. Найпростіше і найширше визначення цього поняття: образ - це те, що ми бачимо. Інше визначення: зображення - це інформація, придатна для зорового сприйняття. Залежно від походження можна умовно розрізнити такі типи зображень:

• Намальовано або надруковано (художник, принтер, принтер).

• Оптичний (розподіл напруженості електромагнітного поля, створюваного оптичним приладом, у певній області простору (області локалізації), наприклад на сітківці, на екрані під час проектування, в площині камери приймач лінз).

• Фотографічне (оптичне зображення, записане на фотоматеріалі в результаті хімічного процесу).

• Електронний або цифровий (оптичне зображення, зафіксоване електронним приймачем, таким як ПЗС, сканер, мікроденситометр). Зображення, що відображається на екрані, також називається електронним.

Цифрове зображення - це не що інше, як двовимірний сигнал. Він визначається математичною функцією f (x, y), де x і y - дві координати, горизонтальна та вертикальна. Значення f (x, y) у будь-якій точці вказує на значення пікселя в цій точці зображення.

Якщо зображення є двовимірним масивом, як воно пов’язане із сигналом? Щоб зрозуміти це, нам слід спочатку зрозуміти, що таке сигнал.

У фізичному світі будь-яка величина, виміряна в часі в просторі або у вищому вимірі, може сприйматися як сигнал. Сигнал є математичною функцією, і він передає деяку інформацію.

Сигнал може бути одновимірним, двовимірним або багатовимірним сигналом. Одновимірний сигнал - це сигнал, який вимірюється в часі. Типовий приклад - мовний сигнал.

Двовимірні сигнали - це сигнали, які вимірюються низкою інших фізичних величин. Прикладом двовимірного сигналу є цифрове зображення. Детальніше ми обговоримо, як формуються та інтерпретуються одновимірні або двовимірні сигнали та вищі сигнали на наступному уроці.

Незалежно від типу зображень, всі різні принципи та методи обробки зображень можна розділити на такі напрямки: - відновлення та вдосконалення зображення; - аналіз зображення (розпізнавання зразків та аналіз сцени); - синтез зображення; - кодування сигналів зображення. У першому напрямку змінюється контраст, придушуються шуми, задаються межі об’єктів і коригуються кольори.

Коли ми говоримо про покращення образу, ми маємо на увазі зміну його властивостей, що призводить до більш комфортного суб'єктивного сприйняття образу, а не до досягнення повної ідентичності з реальним образом.

Другий напрямок визначає об’єкти, що знаходяться на досліджуваній сцені, оцінює взаємозв’язок між фрагментами зображення та визначає характеристики зображуваних об’єктів.

Синтез зображень набув надзвичайного поширення пізно. Методи та прийоми синтезу зображень використовуються в абсолютно різних сферах діяльності. Наприклад, синтез тривимірних зображень виконується на плоских фотографіях поверхні Землі або поверхні інших космічних об’єктів для вивчення властивостей цих об’єктів. Синтез здійснюється під час двовимірного або тривимірного моделювання об'єктів під час автоматизованого проектування будівель, транспортних засобів.

Процес кодування сигналів зображення виконується для зменшення потоку сигналу, необхідного для запису або передачі інформації про зображення.

Реєстрація зображень - це процес перетворення різних наборів даних в єдину систему координат. Даними можуть бути серії фотографій, дані різних датчиків, час, глибина або точки спостереження [4]. Алгоритми реєстрації зображень використовуються в комп'ютерному зорі, медичних техніках візуалізації, у військовій галузі для автоматичного розпізнавання цілей, для впорядкування та

аналізу супутникових знімків. Реєстрація необхідна для того, щоб мати можливість порівняти або інтегрувати дані, отримані з різного обладнання для реєстрації.

У сучасних інформаційних системах зображення в основному передаються, зберігаються та обробляються у цифровій формі, але первинні зображення зазвичай існують у вигляді безперервних двовимірних полів яскравості та розподілу кольорів. Перетворення первинних зображень у цифрові сигнали є обов'язковою дією, якщо ви плануєте використовувати цифрову обробку, передачу та зберігання. Це перетворення складається з двох процедур, які виконуються одночасно.

Перший полягає у заміні безперервного зображення на серію дискретних елементів і називається дискретизацією, а другий замінює безперервний розподіл яскравості та кольору низкою квантованих значень для кожного пікселя і називається квантуванням.

Двовимірна природа зображення у порівнянні зі звичайними одновимірними сигналами надає додаткові можливості для оптимізації цифрового потоку сигналів для зменшення обсягу цифрових даних.

Растр - це структура поля зображення, що утворюється в результаті поелементного розкладання або синтезу зображення. У сучасній термінології елемент зображення називається «піксель» або «піксель», а в англійській літературі ви можете знайти кілька рівнозначних назв, утворених із поєднання слів елемент зображення (елемент зображення)– pictel, pixel, pel.

На практиці застосовується вибірка прямокутної сітки та рівномірне квантування яскравості. Цей підхід використовується через легкість, з якою можна виконувати відповідні операції, а також необхідність виконувати подальші операції, пов'язані з перетворенням зображень.

Коли прямокутна сітка використовується в остаточному вигляді, сигнали оцифрованого зображення подаються у вигляді матриці (монохромне зображення) або серії матриць (кольорове зображення), рядків і стовпців, що містять квантовані значення параметрів відповідні елементи вибіркового зображення.

1.2 Цифрова обробка зображень шляхом поелементних перетворень

У різних інформаційних системах результати обробки даних представлені у вигляді зображень, які формуються на екрані пристрою відображення. Процедура, яка перетворює електричні сигнали в оптичне зображення, називається візуалізацією.

Бажано надати відтвореному зображенню такі властивості, щоб сприйняття окремих, найбільш значущих фрагментів зображення або зображення в цілому було легким для спостерігача з огляду на художні або технологічні властивості цього зображення.

Якість зображення не завжди оцінюється на основі точності вихідного зображення або лінійності зміни параметрів. Наприклад, зображення, отримане в сутінках або тумані, буде характеризуватися низькою контрастністю, розмитими контурами, блідими кольорами.

Зображення, отримане за допомогою інфрачервоного перетворювача, за умови прямого відтворення сигналів може мати невелику кількість ступенів яскравості, що запобігає підкреслення деталей з низькою контрастністю. Часто корисно підкреслити або посилити певні особливості переробленої сцени, щоб поліпшити її суб'єктивне сприйняття.

Суб'єктивність сприйняття зображення у разі візуального аналізу та оцінки значно ускладнює застосування формалізованих підходів до формування властивостей відтвореного зображення. Тому під час реалізації візуалізації зображень широко розповсюджені методи, для яких відсутні суворі математичні критерії оптимальності.

Застосування певних методів, параметрів і режимів часто базується на досвіді, суб’єктивному сприйнятті та художніх навичках. Серед процедур, що використовуються для обробки зображень, можна виділити дві групи [6]:

• поелементна обробка;

• кореляційна обробка.

Коли використовуються методи обробки першої групи, результат обробки будь-якої точки кадру зображення залежить лише від опорного значення характеристичного параметра первинного зображення в тій же точці. Очевидною перевагою таких процедур є простота виконання, вони призводять до значного суб'єктивного покращення якості зору. Поелементна обробка зображень використовується як остання стадія складного процесу обробки зображень.

Друга група процедур базується на тому, що між елементами всього зображення або окремих його фрагментів є взаємні зв’язки – кореляція.

У цьому випадку для отримання результату перетворення для кожного пікселя використовуються дані про параметри певного набору точок первинного зображення навколо обробленої точки. Суть поелементної обробки зображень полягає у встановленні деякої функціональної залежності між кінцевим значенням сигналу зображення та його основним значенням або статистичною характеристикою.

Нехай $x(i, j) = x_{i,\ j}$ та $y(i, j) = y_{i,\ j}$ - значення яскравості вхідного і вихідного сигналів зображень у точці, що має координати відповідно до *i*-го відліку в напрямку вертикальної осі та *j*-го відліку в напрямку горизонтальної осі.

Поелементна обробка базується на тому, що між значеннями яскравості існує однозначна функціональна залежність :

$$y_{i.j} = f_{i,j}(x_{i,j}), \qquad (1.1)$$

що дозволяє на підставі значень первинного сигналу визначити значення кінцевого. Параметри функції, що описує обробку, можуть залежати від поточних координат.

Якщо така залежність існує, виконувана обробка називається неоднорідною. У більшості процедур, які знайшли практичне застосування, виконують однорідну поелементну обробку, в цьому випадку можуть бути відсутні індекси і та j виразу (1.1): $y = f(x)$. (1.2)

Залежність між яскравістю елементів вхідного та вихідного зображень буде визначено функцією (1.2) однаковою для всіх точок зображення.

Процедури обробки на кожен елемент можна розділити на:

• процедури зміни контрастності зображення;

• процедури бінаризації зображень.

Контраст зображення - параметр, рівний відношенню максимальної яскравості до мінімальної в полі зображення.

Контраст зображення - параметр, який визначає різницю в яскравості від середнього рівня і чисельно дорівнює відношенню різниці між максимальною та мінімальною яскравістю та їх сумою [8]. Часто остаточне значення наводиться у відсотках.

1.2.1 Лінійне контрастування зображень

Лінійний контраст використовується для узгодження динамічного діапазону вхідного сигналу зображення та динамічного діапазону яскравості пристрою відображення. Якщо 1 байт (8 біт) пристрою пам'яті виділено для цифрового значення кожного підрахунку зображень, вхідні або вихідні сигнали можуть бути одним із 256 значень. Діапазон можливих значень аналогового сигналу знаходиться в межах від 0 до 255. Зазвичай значення 0 відповідає рівню чорного, а значення 255 - рівню білого.

Припустимо, мінімальна та максимальна яскравість основного зображення дорівнюють x_min та x_max відповідно.

Якщо ці параметри або один із них суттєво відрізняються від меж діапазону сигналів яскравості, відтворене зображення виглядає малоконтрастним, що незручно для спостереження та дослідження.

Під час лінійного контрасту виконується лінійне поелементне перетворення вхідного значення x відповідно до рівняння. $y = a \cdot x + b$, (1.3)

де параметри a та b визначаються бажаними значеннями мінімальної y_{min} та максимальної y_{max} яскравості в кінцевому зображені.

Для знаходження зазначених параметрів потрібно розв’язати систему

рівнянь[8]:

$$\begin{cases} y_{min} = a \cdot x_{min} + b, \\ y = a \cdot x + b \end{cases} \quad (1.4)$$

Після підстановки знайдених значень у (1.3) рівняння для лінійного контрастування набуває остаточного вигляду:

$$y = \frac{x - x_{min}}{x - x} (y_{max} - y_{min}) + y_{min} \quad (1.5)$$

При порівнянні двох зображень (вхідних та вихідних) досягається набагато краща візуальна якість обробленого зображення. Поліпшення відбувається за рахунок узгодження динамічного діапазону вхідного зображення та динамічного діапазону екрану завдяки реалізації лінійного контрасту.

1.2.2 Соляризація зображення

Даний різновид нелінійного контрастування отримав свою назву від слова «солярій» (лат. solarium, від sol – сонце).

Результатом цієї трансформації є освітлення певної частини зображення рівнями яскравості, пов'язаними з центром динамічного діапазону.

Перетворення здійснюють відповідно до співвідношення:

$$y = k \cdot x \cdot (x_{\max} - x), \quad (1.6)$$

де $x_{\max}$ – максимальне значення вихідного сигналу;

k – константа, що управляє динамічним діапазоном перетвореного зображення.

Функція, що описує дане перетворення, є квадратичною.

Внаслідок соляризації ділянки вхідного зображення з рівнем білого (або

близьким до нього рівнем яскравості) набувають чорного рівня. Рівень білого у вихідному зображенні набувають ділянки, що мали на вході середній рівень яскравості (рівень сірого).

1.2.3 Зональне контрастування зображення

Цей тип контрасту називається підготовкою зображення, тобто здійсненням такого перетворення яскравості певної частини динамічного діапазону, що дозволяє дослідити або підкреслити певні властивості зображення.

Розтин - це клас поелементних перетворень зображень, більшість з яких мають власну назву. Амплітудні характеристики процедур розтину, які найчастіше використовуються на практиці [9]:

• бінаризація;

• зріз для наочності;

• зворотний контраст;

• зональний лінійний контраст.

Бінаризація - це перетворення з пороговою характеристикою.

Операція використовується, коли метою обробки є вибір характерних контурів об'єктів зображення. Основною проблемою в реалізації такої обробки є визначення порогового значення x0, рівняння, яке дозволяє визначити значення кінцевого зображення в будь-якій точці, використовуючи яскравість вхідного зображення.

Найбільш виправданим підходом до визначення порогу є використання математичного пристрою ймовірності, довільних процесів та довільних полів для опису зображень. За цих обставин визначення оптимального порогу двійкового квантування є статистичним завданням.

Іноді при обробці зображення доводиться мати справу із зображеннями, збереженими в напівтонах, але по суті вони мало чим відрізняються від двійкових. Сюди входять текст, штрихове зображення, малюнки, зображення відбитків пальців.

Щільність імовірності $w(x)$, що описує розподіл яскравості такого

зображення, може містити два виділених максимуми. Поріг бінарного квантування варто вибирати посередині між цими максимумами.

Заміна вихідного напівтонового зображення бінарним препаратом вирішує дві основні проблеми. По-перше, більша видимість буде досягнута під час візуального спостереження порівняно з вихідним зображенням. По-друге, обсяг пам'яті для зберігання зображення значно зменшиться, оскільки для зберігання кожної точки двійкового зображення двійковому файлу потрібно лише 1 біт пам'яті, тоді як для біття часто потрібно 8 біт.

Іноді яскравість зображення посилюється за рахунок контрасту з пилоподібною характеристикою перетворення. У цьому випадку різні піддіапазони яскравості одночасно піддаються локальному лінійному контрасту. Однак майте на увазі, що ця трансформація, як і деякі інші, може супроводжуватися появою помилкових контурів у кінцевому препараті.

1.2.4 Перетворення гістограм, еквалізація

В результаті всіх поелементних перетворень відбувається зміна закону розподілу ймовірностей яскравості пікселів, що описують зображення. Розглянемо механізм цієї зміни на прикладі випадкового перетворення з монотонною ознакою.

Нехай Δx – довільний малий інтервал значень випадкової величини x, а Δy – відповідний інтервал перетвореної випадкової величини y. Знаходження величини x в інтервалі Δx обумовлює знаходження величини y в інтервалі Δy , що означає імовірнісну еквівалентність цих двох подій.

З огляду на малі розміри обох інтервалів, можна записати таку рівність:

$$w_x(x)|\Delta x| \approx w_y|\Delta y|.$$

Для обчислення щільності ймовірності перетвореної величини, підставимо замість x її вираз через обернену функцію та виконаємо граничний перехід □x □ 0 (що обумовлює □y □ 0).

За допомогою цього виразу можна обчислити щільність ймовірності результату перетворення, яка не збігається з щільністю розподілу вихідної

випадкової величини. Зрозуміло, що форма закону розподілу щільності ймовірностей wy □y□ залежить від характеристики перетворення, оскільки вираз (1.8) містить обернену функцію перетворення та її похідну.

Відносини набувають більш складної форми, коли перетворення описується не унікальною функцією.

В результаті кожного перетворення щільність ймовірності остаточного зображення не буде збігатися з щільністю ймовірності вихідного зображення. Неважко помітити, що за умови лінійного контрастування форма закону розподілу щільності ймовірності буде збережена, але параметри щільності ймовірності перетвореного зображення будуть іншими.

Перетворення щільності ймовірності вимагає знання інтегрального розподілу для первинного зображення. Як правило, достовірних відомостей про нього немає. Використання аналітичних підходів для опису функцій розподілу також мало сенсу, оскільки їх незначні відхилення від фактичних розподілів можуть призвести до значної різниці в кінцевих результатах.

Обробка зображень перетворення розподілу виконується у два етапи. На першому етапі вимірюється гістограма вхідного зображення.

Для цифрового зображення, шкала яскравості якого становить від 0 до 255, гістограма являє собою таблицю з 256 чисел, кожна з яких представляє кількість пікселів заданої яскравості в даному кадрі.

Щоб знайти оцінку розподілу ймовірності яскравості пікселів зображення, розділіть усі числа в цій таблиці на загальний розмір вибірки, рівний загальній кількості пікселів на зображенні.

Позначимо цю оцінку як $w^*(j), 0 \le j \le 255.$

Тоді оцінку інтегрального розподілу буде визначено за формулою:

$$F^*(j) = \sum_{i=0}^{j} w_x^*(i).$$

На другому етапі виконайте пряме нелінійне перетворення (1.2), яке дасть необхідні властивості вихідного зображення. Під час такого перетворення використовується оцінка на основі гістограми замість невідомого інтегрального

розподілу.

Методи поелементного перетворення зображень, метою яких є зміна законів розподілу, називаються методами гістограм. Зокрема, перетворення, яке надає кінцевому зображенню рівномірний розподіл, називається згладжуванням (вирівнюванням) гістограм.

Процедури перетворення гістограм можна застосовувати до зображення в цілому або до окремих фрагментів.

Застосування цієї процедури до окремих фрагментів зображення може бути корисним для обробки нестаціонарних зображень, зміст яких суттєво відрізняється за своїми характеристиками в різних областях. У цьому випадку найкращого ефекту можна досягти, застосувавши обробку гістограми до окремих ділянок зображення.

Використання співвідношень (1.5) - (1.9), які застосовуються до зображень із постійним розподілом яскравості, не зовсім коректне для цифрових зображень.

Зверніть увагу, що в результаті обробки неможливо отримати ідеальний розподіл вихідного зображення, тому корисно перевірити його гістограму.

1.2.4 Застосування табличного методу до поелементного перетворення зображень

Під час поелементних перетворень зображень обчислення згідно (1.2) необхідно виконувати для всіх пікселів вхідного зображення. У тих завданнях, де функція $f(x)$ (1.2) передбачає трудомісткі обчислення (множення, ділення, розрахунки тригонометричних функцій, статистичних та інших функцій), застосування безпосередньо прямого методу перетворення яскравості може виявитись взагалі неприйнятним. Час виконання обробки залежить від її обчислювальної складності. Для запобігання таких незручностей застосовують табличний метод, що отримав широке розповсюдження у практиці цифрової обробки зображень. Сутність табличного методу полягає в тому, що шляхом попереднього

розрахунку створюють таблицю функції $y = f(x)$.

Під час обробки зображення замість обчислень використовують готові результати шляхом звернення до цієї таблиці. Значення вхідної яскравості x використовують для визначення номера стовпчика, з якого треба прочитати значення перетвореного сигналу y.

Виконання цієї нескладної операції у порівнянні з безпосереднім обчисленням значень y дозволяє зробити обробку технологічною, а тривалість обчислень стає незалежною від складності перетворення.

Необхідно взяти до уваги той факт, що всі реальні таблиці, які можна записати в оперативній пам’яті комп’ютера, мають обмежену величину. Якщо множина значень вхідного сигналу перевищує розміри таблиці, то при розташуванні значення x між точками, для яких у таблиці зафіксовано певні значення, доводиться застосовувати інтерполяцію – наближене визначення відсутніх значень функції $y = f(x)$ за наявними сусідніми значеннями.

Часто для забезпечення цих обчислень використовують лінійну інтерполяцію – на проміжку між заданими вузлами невідому функцію замінюють відрізком прямої.

Слід зазначити, що якщо вхідне зображення відображається як ціле число у діапазоні від 0 до 255, розмір усієї таблиці, що містить усі значення функції, є незначним. Значення яскравості вхідного сигналу - це адреса, яка визначає номер стовпця у таблиці функцій. Обробка цим методом дуже зручна і швидка.

Фактичні значення пікселів використовуються при виконанні різних нецілих редагувань та перетворень зображень. Якщо ви хочете зменшити обсяг задіяної пам'яті або збільшити швидкість алгоритмів, замість дисплеїв із плаваючою крапкою використовуються дисплеї з фіксованою крапкою;

• напівтонові зображення з високою роздільною здатністю (цілі числа без підпису). Використовується в тих випадках, коли 256 відтінків сірого не дозволяють відображати все багатство вихідної інформації;

• Позначити зображення - використовується під час автоматичного вибору областей посилань та об'єктів. Кожен піксель такого зображення позначається номером області, до якої воно належить. Байтові зображення тут не підходять,

оскільки вони можуть кодувати лише 255 різних областей, а зображення з високою роздільною здатністю можуть мати десятки і навіть сотні тисяч;

• двовимірні частотні характеристики - складні зображення, що складаються з реальної та уявної частин. Вони утворюються в результаті двовимірного косинусного перетворення, двовимірного перетворення Фур'є, швидкого перетворення Фур'є, перетворення зображення з просторової області в частотну область. На програмному рівні вони реалізуються не як двовимірний масив комплексних чисел (двокомпонентних векторів), а як пара двовимірних масивів, один з яких представляє реальну частину зображення, а інший - уявне;

• кольорові зображення - спеціальний тип даних, формат запису TcolorRef = {Червоний, Зелений, Синій}. Роздільна здатність на кожному з каналів - 8 біт.

Кольорове зображення є системним типом даних і підтримується усіма пристроями введення кольорових зображень. Крім того, стандартний тип даних TRGBBitmap підтримується операційною системою Windows як частина графічного інтерфейсу на рівні системи;

• багатозонові та гіперспектральні зображення - вектор, піксель є масивом цілих чисел. Сформовані спеціальними пристроями введення. Використовується для класифікації за пікселями та сегментації зображень. На програмному рівні вони, як правило, реалізуються не як двовимірний набір векторів, а як набір двовимірних зображень, кожен з яких відповідає одному зональному спектральному компоненту;

• зображення зображень - скаляр і вектор, піксель - це скаляр, масив або список реальних значень, є результатом аналізу ознак зображень. Використовується для класифікації за пікселями та сегментації зображень. На програмному рівні вони зазвичай реалізуються не як двовимірний набір векторів, а як набір двовимірних зображень, кожен з яких відповідає одному типу функції.

1.2.5 Етапи та задачі цифрової обробки зображень

Покращення зображення - це одна з найпростіших та найбільш вражаючих областей цифрової обробки зображень. По суті, методи покращення зображення спрямовані на виявлення погано видимих деталей або просто підкреслення бажаних особливостей оригінального зображення. Відомим прикладом покращення є збільшення контрастності зображення, оскільки воно "виглядає краще". Важливо пам’ятати, що покращення якості - це дуже суб’єктивна сфера обробки зображень.

Відновлення зображення - це також область, пов’язана з поліпшенням візуальної якості зображення, але на відміну від реального покращення, критерії якого є суб’єктивними, відновлення зображення є об’єктивним, оскільки методи відновлення зображення базуються на математичних або ймовірнісних моделях. спотворення (спотворення) зображення.

Обробка кольорових зображень стала особливо важливою через значне розширення використання кольорових зображень в Інтернеті. Обговорюються деякі фундаментальні поняття, пов'язані з кольоровими моделями, та основні типи цифрових кольорових перетворень.

Вейвлети є основою для одночасного показу зображень з різною роздільною здатністю. Цей пристрій в основному використовується для стиснення даних зображення, а також для побудови пірамідального подання, в якому зображення поступово розбивається на менші фрагменти.

Стиснення, як випливає з назви, стосується методів зменшення обсягу пам'яті, необхідного для зберігання зображення, або зменшення пропускної здатності, необхідної для його надсилання.

Морфологічна обробка включає інструменти для отримання таких компонентів зображення, які можуть бути корисними для відображення та опису фігури.

Сегментація поділяє зображення на компоненти або об'єкти. Взагалі, автоматична сегментація є одним із найскладніших завдань цифрових зображень. Надмірна дробова сегментація вимагає складного вирішення проблеми обробки зображень, якщо ви хочете ідентифікувати об’єкти окремо. З іншого боку,

недостатньо детальна або помилкова сегментація майже неминуче призведе до помилок на завершальній стадії обробки. Загалом, чим точніша сегментація, тим більше шансів на успіх у визнанні.

Представлення та описи майже завжди відбуваються відразу після етапу сегментації, який зазвичай містить лише вихідні піксельні дані, які або утворюють межу регіону, або представляють усі точки самих регіонів. У будь-якому випадку ви повинні перетворити дані у форму, необхідну для комп’ютерної обробки. Перше рішення, яке потрібно прийняти, - подати ці дані у вигляді меж областей або областей в цілому.

Розпізнавання - це процес, який присвоює об’єкту ідентифікатор (наприклад, «транспортний засіб») на основі його описів.

1.3 Вивчення різновидів нейронних мереж, їх структуру та призначення

Існує багато видів штучних нейронних мереж, кожна зі своїми унікальними особливостями. Розглянемо найбільш важливі і поширені типи нейронних мереж.

Це один з найпростіших видів штучних нейронних мереж. У цьому типі нейронної мережі дані проходять через кілька вхідних вузлів, поки не досягають вихідного вузла. З цього випливає, що дані рухаються лише в одному напрямку з першого рівня, поки не досягнуть вихідного вузла. Це також відомо як хвиля, що поширюється вперед, що зазвичай досягається за допомогою рейтингу тригера. На відміну від більш складних типів нейронних мереж, вона не має зворотного поширення, а дані рухаються лише в одному напрямку. Нейронна мережа може мати один шар або вона може мати приховані шари, які дозволяють розрахувати суму вхідних продуктів та їх ваги, які потім подаються на вихід. Нейронні мережі прямого поширення використовуються в технологіях розпізнавання обличчя та комп'ютерного зору, і ця мережа обладнана для обробки даних, що містять багато шуму. Ця нейронна мережа досить проста в обслуговуванні.

1.3.1 Мережа радіальних базисних функцій (Radial Basis Function Neural Network)

Радіальна основна функціональна мережа (RBF) - це загальноприйнятий тип штучної нейронної мережі для проблем функціонального підходу. Радіальні базові функціональні мережі відрізняються від інших нейронних мереж своїм універсальним підходом і більшою швидкістю навчання. Мережа RBF - це тип нейронної мережі прямого надходження, що складається з трьох шарів, а саме вхідного рівня, прихованого рівня і вихідного рівня, кожен з яких має різні завдання.

Навчання моделі RBF зупиняється після того, як обчислена похибка досягла бажаних значень (наприклад, 0,01) або кількість ітерацій навчання (наприклад, 500) вже завершена. Вибирається мережа RBF з певною кількістю вузлів (наприклад, 10) у прихованому шарі. Функція Гаусса використовується як функція передачі даних в одиницях обчислення. Залежно від випадку, зазвичай зазначається, що мережі RBF займають менше часу для завершення навчання.

Нейронна мережа радіальних основних функцій широко функціонує в системах рекуперації енергії. За останні десятиліття енергетичні системи стали більшими та складнішими, збільшуючи ризик затемнення. Ця нейронна мережа використовується в системах рекуперації енергії для відновлення енергії в найкоротші терміни.

1.3.2 Перцептрон

Розенблатт спроектував перший нейронний комп’ютер, персептрон, в 1958 році, намагаючись імітувати людське навчання. Дендрити нейрона моделюються вагами, помноженими на вхідні значення. Крім того, додається значення компенсації для моделювання необхідного потенціалу активації нейронів. Потім множинні значення та зміщення додаються в тілі клітини і передаються через функцію активації, щоб отримати вихід, який представляє швидкість вогню в нейроні. Описана архітектура показана на малюнку нижче.

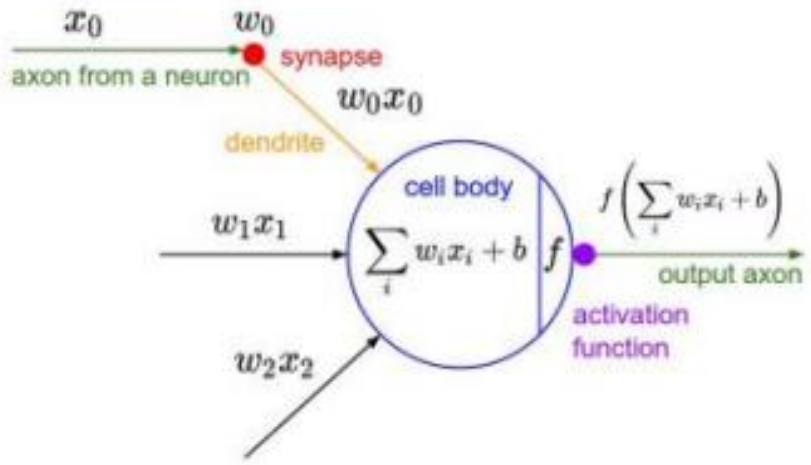


Рис.1.1. Модель персептрона з вагами wi, входами xi та зсувом b.

Персептрон - це лінійний класифікатор, який визначається вагами wi, зміщенням b і функцією активації f. Ви можете поєднати зміщення зі шкалою, використовуючи однорідні координати, тобто розмістити зміщення внизу вектора ваги та додати константу 1 до вхідного вектора x. Можна використовувати кілька тригерних функцій, але оригінальний персептрон використовує функцію важкого ступеня, що веде до двійкового виводу. Отже, персептрон є лінійним класифікатором, і його ваги визначають гіперплощину як лінійну межу роздільної здатності між класами. Тому представницька влада обмежена; наприклад, неможливо імітувати функцію XOR за допомогою персептрона, оскільки жодна лінія не може бути проведена до окремих класів, як показано на малюнку нижче.

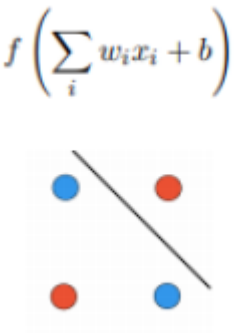


Рис. 1.2. Функція XOR створює нелінійно відокремлений набір. Два класи

червоний і синій не можна розділити однією прямою лінією

1.3.3 Багатошаровий перцептрон

Багатошаровий персептрон має три або більше шарів. Застосовується для класифікації даних, які не можна розділити лінійно. Багатошаровий прецептрон показаний на рисунку 1.3.

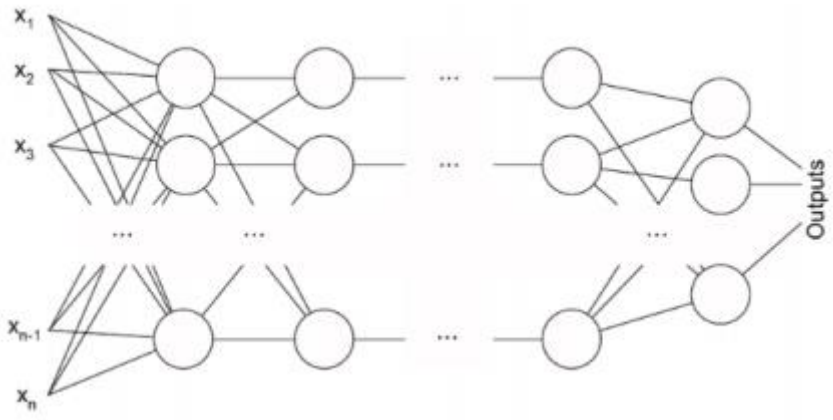


Рис 1.3. Багатошаровий перцептрон

Це свого роду штучна нейронна мережа, яка повністю пов’язана. Це пов’язано з тим, що кожен вузол у шарі пов’язаний із кожним вузлом у наступному шарі. Багатошаровий персептрон використовує нелінійну функцію активації (переважно гіперболічну дотичну або логістичну функцію).

Цей тип нейронних мереж широко використовується в технологіях розпізнавання мови та машинного перекладу.

1.3.4 Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) - це підгрупа алгоритмів глибокого навчання, які зосереджуються на завданнях комп'ютерного зору та обробки зображень. Мережі системи CNN надихаються тим, як мозок обробляє візуальну інформацію. Хубель і Візель першими запропонували глибокі моделі, що нагадують будову візуального котячого кота. Ці моделі ідентифікували прості клітини з місцево сприйнятливими полями, подібними до фільтрів або ядер, та

складні клітини, подібними до шару. Перший CNN був запроваджений в неокогнітроні Фукусіма [5]. Пізніше CNN модернізував LeNet-5 до Янна ЛеКуна, який використовував стохастичні градієнти на основі градієнтів для розпізнавання документів і дуже успішно розпізнавав рукописні завдання. Основним недоліком досліджень і розробок CNN у 1990-х і 2000-х роках була обчислювальна потужність, необхідна для їх широкого застосування до зображень з високою роздільною здатністю. Однак це змінилося з 2010 року. Є три причини, чому глибокі мережі стали успішними:

• збільшена обчислювальна потужність завдяки Закону Мура, особливо для сучасних графічних процесорів;

• більше навчальних даних;

• нові та кращі алгоритми.

Зі збільшенням обчислювальної потужності дослідники описали нові способи ефективнішої підготовки згорткових нейронних мереж, що дозволяє створювати більш глибокі мережі. Останнім часом продуктивність значно зросла завдяки безлічі баз даних зображень, які наближаються або навіть перевершують людські показники, такі як ідентифікація абонента (<0,25 відсотка) та багато інших завдань розпізнавання образів, основні проблеми візуальної класифікації та інші.

Загалом CNN дуже добре класифікує такі об'єкти, як конкретні породи собак і котів, на основі дрібнозернистих частин, тоді як у людей з ними проблеми. Недоліком глибокого навчання є те, що для розпізнавання об’єктів у реалістичному середовищі потрібні дуже великі масиви даних для навчання. В даний час найкращі мережі CNN борються з дрібними або плоскими предметами або спотвореними цифровими фільтрами. Нещодавнє дослідження виявило відмінності в тому, наскільки глибокі нейронні мережі та люди розпізнають предмети. У цьому дослідженні зображення кодуються таким чином, що люди не можуть розпізнати, але глибокі нейронні мережі виявили правильний клас об’єктів майже із 100% точністю [10].

Для навчання високоскладних згорткових нейронних мереж на великих наборах даних останнім часом спостерігається тенденція до збільшення кількості

шарів та розмірів шарів за допомогою скринінгу [34] для вирішення проблеми надмірного обладнання. Крижевський та Сегеди, творці двох найвідоміших мереж, що використовуються у дослідженнях, а саме AlexNet [6] та GoogLeNet [7], наголошують на важливості високого рівня відсіву під час навчання.

Інша тенденція - зробити мережі дуже глибокими, тобто мережі мають багато шарів. Цайлер експериментував із глибиною CNN і дійшов висновку, що продуктивність мережі сильно залежить від кількості шарів. Виміряна продуктивність значно знижується, навіть якщо знімається згортковий шар. Він зазначає, що глибина мережі важливіша за будь-яку іншу архітектурну частину мережі.

Тому вибір базової архітектури є запорукою ефективності мережі. Це також можна побачити на GoogLeNet, який був опублікований у 2014 році та має глибину 22 шари, не враховуючи шарів злиття. У 2012 році AlexNet розпочав рух навколо вдосконалення глибокого навчання, включаючи 8-рівневу мережу. Попри досягнення найсучасніших результатів при публікації, багато інших мереж перевершили його.

Обмежуючим фактором для розміру CNN є обсяг пам'яті, доступний у сучасних графічних процесорах, і портативний час навчання. До широкого використання графічних процесорів для глибокого мережевого навчання загальний час навчання процесів був у 9 разів довшим. Бібліотека cuDNN від Nvidia, яка оптимізована для швидкої обробки зображень та ефективних операцій згинання графічного процесора, ще більше пришвидшує час навчання майже у 9 разів. Тож сучасні графічні процесори з найновішими бібліотеками CUDA та cuDNN пришвидшують обчислювальний час у 17 разів порівняно з сучасними процесорами. Залежно від кількості мережевих параметрів та розміру набору даних, вивчення згорткової нейронної мережі на графічному процесорі із випадково ініційованими вагами іноді може зайняти дні або навіть тижні.

У згортковій нейронній мережі (CNN) використовуються різноманітні багатошарові персептрони. CNN містить один або кілька згорткових шарів. Ці шари можуть бути повністю зв’язані або поєднані між собою.

Перш ніж передавати результат наступному шару, згортковий шар використовує операцію згортання введення. Завдяки цій згортковій роботі мережа може бути набагато глибшою, але з набагато меншими параметрами.

Завдяки цій можливості конвертовані нейронні мережі показують високоефективні результати в розпізнаванні зображень та відео, природній обробці мови та системах перенаправлення.

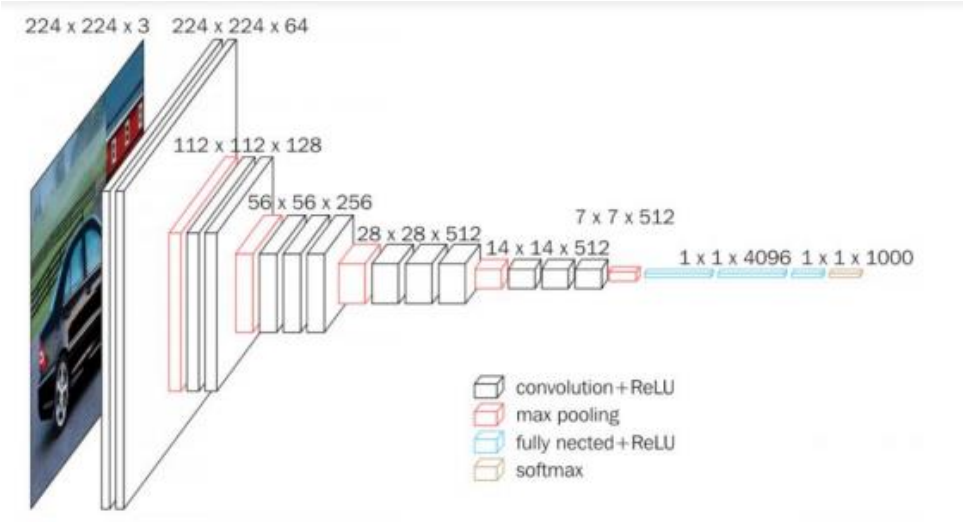


Рис 1.4. Архітектура згорткової нейронної мережі

Згорткові нейронні мережі показують чудові результати в семантичному аналізі та виявленні парафрази. Вони також використовуються в обробці сигналів та класифікації зображень, де вони добре працюють.

1.3.5 Рекурентна нейронна мережа - Довга короткочасна пам'ять

Рекурентна нейронна мережа - це різновид штучної нейронної мережі, в якій вихід певного рівня зберігається і подається назад на вхід. Це допомагає передбачити результат шару.

Перший шар формується так само, як і в електромережі. Тобто добуток суми ваг та характеристик. Однак у наступних шарах починається повторюваний процес нейронної мережі.

З кожного кроку часу до наступного кожен вузол запам'ятовує інформацію, яку мав на попередньому кроці часу. Іншими словами, кожен вузол виконує функцію комірки пам'яті під час обчислення та виконання операцій. Нейронна мережа починає розповсюдження вперед, як зазвичай, але запам'ятовує інформацію, яку, можливо, доведеться використати пізніше.

Якщо передбачення неправильне, система навчається під час відтворення і працює над правильним передбаченням. Цей тип нейронної мережі є дуже ефективним у технології синтезу мовлення. Ось так виглядає періодична нейронна мережа.

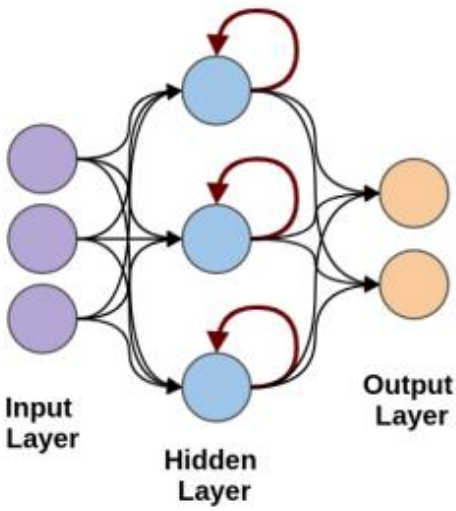


Рис 1.5. Рекурентна нейронна мережа

1.3.6 Модульна нейронна мережа

Модульна нейронна мережа має ряд різних мереж, які функціонують незалежно та виконують підзадачі. Різні мережі насправді не взаємодіють між собою і не передають сигнал під час розрахунку. Вони працюють самостійно для досягнення результату.

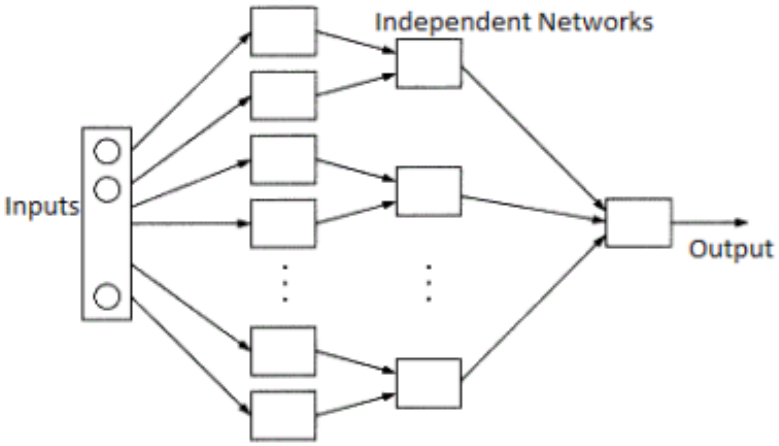


Рис 1.6. Модульна нейронна мережа

Як результат, великий та складний процес обчислення можна виконати набагато швидше, розбивши його на незалежні компоненти. Швидкість комп'ютера зростає, оскільки мережі не взаємодіють і навіть не з'єднуються між собою. Ось візуальна картина модульної нейронної мережі.

РОЗДІЛ 2.
МЕТОДИ ТА АЛГОРИТМИ ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ

2.1. Прості методи обробки зображень

Поліпшення зображення (Image Enhancement) - процес поліпшення якості зображення без втрати інформації для отримання бажаного візуального результату (роздільна здатність, колір і стиль) або підготовка фотографій до подальшого аналізу в різних програмах комп'ютерного зору: розпізнавання об'єктів, класифікація, інтерпретація зображень. Поліпшення якості зображення, як правило, передбачає низку перетворень: зменшення шуму, покращення розмитості фотографії, підвищену роздільну здатність, контрастність, затемнення темних фотографій, усунення оптичних спотворень тощо.

Залежно від розглянутої звукової моделі складність вирішення такої задачі може значно варіюватися. У практичних завданнях зазвичай має сенс використовувати моделі імпульсного та аддитивного гауссового шуму, оскільки вони близькі до найпоширеніших реальних шумів.

Адитивний гауссовий шум заснований на використанні нормально розподіленої випадкової величини без математичних сподівань, значення якої додаються до кожного пікселя зображення. Ця модель описує шум, який виникає природним чином при зйомці зображення за допомогою цифрових датчиків, і для якого існують добре вивчені методи зменшення шуму - однак високою ефективністю користуються класичні лінійні фільтри, такі як фільтр Вінера [13], поряд із фільтрацією шум також є незначними деталями. У контексті цієї проблеми може знадобитися використовувати нелінійні методи, такі як анізотропні алгоритми дифузії, двосторонні та тристоронні фільтри [14]. Вищезазначені методи засновані на локалізованій оцінці градієнта зображення, наявності контурів та тонких деталей, що може додатково зменшити плавність цих областей та заощадити більше деталей.

Імпульсний шум виражається як неправильне (фіксоване або випадкове) значення деяких пікселів зображення. Зазвичай такий шум виникає через помилки при передачі інформації. Для такої моделі ефективні рейтингові фільтри [15], які спрямовані на виявлення та виправлення похибки імпульсу, використовуючи оцінку на основі наявних даних, залишаючи цілими цілі пікселі.

Як більш загальне рішення, придатне для придушення як гауссового, так і імпульсного шуму, використовуються адаптивні алгоритми, які використовують певне піксельне середовище для визначення типу та корекції шуму, властивого центру цього сусідства.

Поліпшення зображення приділяло багато уваги в своїх додатках, починаючи від візуалізації спостережень [5,6], медичної візуалізації [7,8] та розпізнавання об'єктів. Звичайні методи засновані на інтерполяції, такі як білінійна, бікубічна та інтерполяція Ланкросса [11], які є простими, але зазвичай дають занадто гнучкі реконструкції. Щоб подолати цю проблему, методи базуються на прикладах

Було запропоновано використовувати вручну створені функції, починаючи з вивчення словникового запасу, вивчення сусідства з деревом регресії [9].

Недавні досягнення в галузі глибокого навчання досягли значних успіхів у надвисокій роздільній здатності. [7.13]. Донг та ін. [24] вперше представили SRCNN для наскрізного навчання, щоб зіставити низьку якість із високою якістю. Хоча SRCNN - це лише мережа з трьома згортковими рівнями, вона перевершує попередні методи, засновані на ручних функціях. У роботі [25] Shi et al. Надайте субпіксельні модулі, що забезпечують ефективний метод передискретизації для відновлення високоякісних зображень. Виявляється, надвисока роздільна здатність також виграє від дуже глибоких мереж, як і в багатьох інших додатках. 5-рівневий FSRCNN [26], 20-шаровий VDSR [21] та 52-рівневий DRRN [27] демонструють значні покращення з точки зору точності. Lim et al. [23] надають дуже широко модифікований ResNet [28] для досягнення найсучасніших функцій PSNR. Хоча їх підвищення точності не можна заперечувати, обчислювальні вимоги залишають бажати кращого, особливо для використання в мобільних пристроях.

Глобальне покращення зображень: Бичковський та ін. [3] зібрали популярний набір даних MIT-Adobe-5K1, що складається з 5000 фотографій та ретушований п’ятьма різними художниками. Автори пропонують підхід на основі регресії для вивчення фотографічних налаштувань художників кількох зображень. Для автоматизації покращення кольорів [27] пропонує підхід до класифікації тренувань у десяти популярних глобальних програмах управління кольором. У [7] FCN використовується для вивчення підходів різних глобальних операторів обробки зображень, таких як фотографічний стиль, нелокальна дегазація та малювання олівцем.

Пост-обробка фотографії виконується у кадрі [10], який забезпечує глобальні криві ретуші в просторі RGB. Підхід підсилення (RL) забезпечує процес коригування зображення в [10], а глибокий RL використовується для визначення порядку коригувань покращення в [19], дозволяючи застосовувати глобальні налаштування зображення (наприклад, Контраст, Насиченість).

2.2. Огляд технологій обробки зображень на основі нейронних мереж

2.2.1 Аналіз сучасних методів покращення роздільної здатності окремих зображень.

Застосована супер-роздільна здатність одного зображення (SR) має на меті реконструювати втрачені високі частоти (насичені деталі) у зображенні, використовуючи низку попередніх прикладів парних зображень із низькою роздільною здатністю (LR) та високою роздільною здатністю (HR). Ця проблема актуальна, для кожного зображення LR простір правдоподібних відповідних HR-зображень величезний і масштабується квадратично з коефіцієнтом збільшення.

Конкурс NTIRE 2017 [12] став кроком вперед у порівняльному аналізі CP. Це був перший виклик у своєму роді із прикладами стандартної бікубічної деградації та "невідомих" операторів (розмиття та зменшення) на зображеннях із роздільною здатністю 1000 DIVerse 2K із набору даних DIV2K [14].

Завдання NTIRE 2018 базується на NTIRE 2017 і буде продовжуватися. Порівняно з попереднім виданням, NTIRE 2018: (1) використовує той самий набір даних DIV2K; (2) має лише один бікубічний шлях відновлення зі збільшенням × 8; (3) Сприяє реалістичній настройці, емулюючи шлях збору даних камери через три доріжки з поступово зростаючими труднощами.

Цілями конкурсу NTIRE 2018, заснованого на надвисокій роздільній здатності для одного зображення із зразкової бази даних, є: (i) оцінка та сприяння поточному рівню CP; (ii) порівняти різні рішення; та (iii) сприяти реалістичному узгодженню CP. Набір даних DIV2K [1], що використовується викликом NTIRE 2017 SR, також використовується в нашому завданні. DIV2K має 1000 зображень RGB із роздільною здатністю 2K RGB 800 для тренувань, 100 для тестування та 100 для тестування. Високоякісні зображення, зібрані вручну, різняться за змістом.

Доступ до даних та представлення зображень HR вимагає реєстрації на гоночному курсі Codalab.

Трек 1: Класичний Bicubic × 8 використовує бікубічне зменшення масштабу (Matlab imresize, налаштування за замовчуванням), найпоширеніший параметр в останній літературі про SR, із співвідношенням × 8. Він призначений для легкого захоплення останніх запропонованих SR-рішень. впровадити.

Трек 2: Реалістичні несприятливі умови Mild × 4 припускають, що оператори деградації, що імітують процес отримання зображення з цифрової камери, можуть бути оцінені за допомогою навчальних пар зображень LR та HR. Оператори деградації однакові (з однаковими параметрами управління) у кожному просторі зображень та для всіх зображень у наборі поїздів, перевірка та тестування. Розмиття в русі та шум Пуассона насправді залежать від зображення та можуть спричинити зсув та масштабування пікселів. Кожне зображення правди DIV2K (GT) зменшується (× 4) до зображень LR.

Трек 3: Реально важкі × 4 несприятливі умови схожі на Трек 2, тільки погіршення сильніше.

Доріжка 4: Реалістичні несприятливі умови Wild × 4 схожі на смуги 2 і 3, оператори деградації однакові в просторі зображень, але відрізняються від одного

зображення до іншого. Деякі зображення менш погіршуються, ніж інші. Цей параметр найближчий до реальних "диких" умов. Через підвищену складність завдання було створено 4 деградованих зображення LR для кожного навчального зображення HR.

На першому етапі змагань - етапі розробки, учасники отримали пари зображень поїздів LR та HR та зображення перевірки LR із набору даних DIV2K; Інтернет-сервер перевірки з табло надав миттєвий зворотний зв'язок про завантажені результати HR на зображення перевірки LR. На другому етапі тестування учасники отримали тестові зображення від LR і повинні були представити результати суперпоширення HR-зображень, код та інформаційний бюлетень для свого методу.

Протокол оцінки це пікове відношення сигнал / шум (PSNR), виміряне в децибелах [дБ], та індекс структурної схожості (SSIM) [37], як повні контрольні значення, розраховані між результатом HR та зображенням GT. Ми повідомляємо середні значення для наборів зображень. Як і в [31], ми ігноруємо межу пікселів зображення 6 + s (s - коефіцієнт масштабування). Використовуючи зміщення пікселів та масштабування для доріжок 2, 3 та 4, ми розглядаємо всі переклади $\in [-40, 40]$ за обома осями, обчислюємо PSNR та SSIM та повідомляємо про найбільш сприятливі оцінки. Через обмеження в часі для доріжок 2, 3 та 4 ми розрахували PSNR та SSIM із обрізаним зображенням із центром 60 × 60 пікселів на етапі перевірки та обрізаним зображенням із центром 800 × 800 пікселів для кінцевих результатів.

Команда Toyota-TI запропоновані мережі глибокого зворотного проектування (DBPN) [9] (див. рис. 2.1), які використовують зворотний та зворотний зв’язок про помилки для керівництва мережею для досягнення оптимальних результатів. На відміну від попередніх методів із використанням зворотного SR-зображення, DBPN виконує взаємопов’язані кроки вибірки вгору та вниз, щоб генерувати як функції LR, так і HR, і обидві помилки прогнозування накопичуються для прогнозування кінцевих результатів SR. Групові функції LR

спочатку витягуються із вхідного зображення LR. Потім зворотні етапи проекції використовуються для почергового формування LR та HR показників ефективності, які додатково посилюються щільним з'єднанням, де вхід для кожного модуля проекції є комбінацією виходу з усіх попередніх блоків. Нарешті, всі карти ефективності HR використовуються для реконструкції остаточної оцінки SR.

Структура нещодавно введених висхідних і опускаючих проекційних одиниць показана на рисунку 2.1 (b). Для вирішення класичної задачі SR із зменшенням вибірки в бікубічному форматі × 8 DBPN DBPN використовує згортковий шар 12 × 12 з вісьмома кроками та двома заливками в одиницях проекції та формує результат SR.

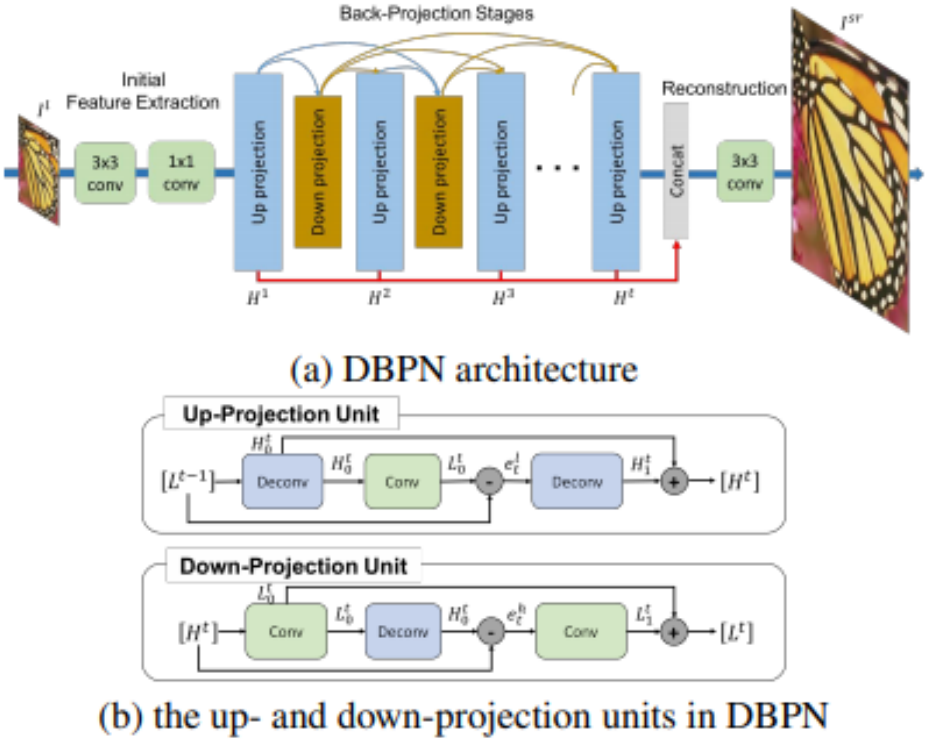


Рисунок 2.1. Структура мережі Toyota-TI

Команда Rainbow запропонував метод, заснований на EDSR [21] та SRDenseNet [11] (рис. 2.2). Вони використовували пірамідальну архітектуру, щоб поступово генерувати HR-зображення. Щоб компенсувати результати та час

висновків, вони дотримувались двоступеневої стратегії розширення. Вони навчили мережу з втратою L1 та адаптували її з втратою L2.

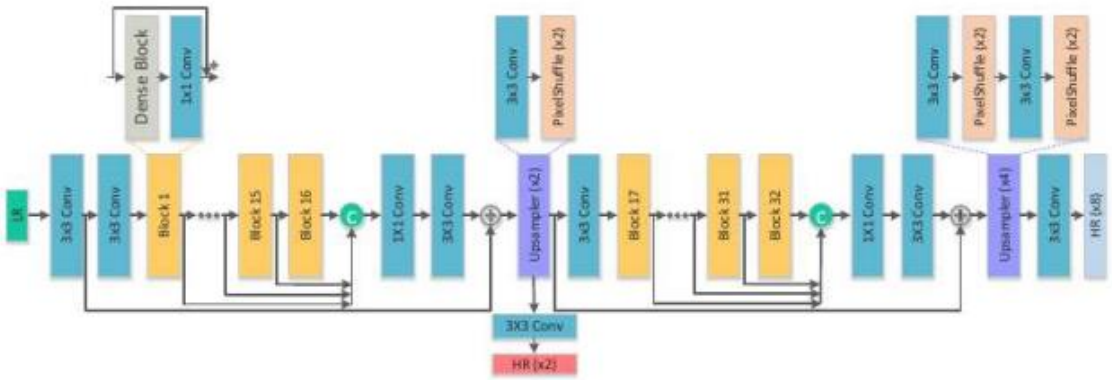


Рис.2.2. Архітектура мережі Rainbow’s

Команда Pixel Overflow використовували ту саму структуру мережі, що і EDSR [21]. Для кращої ефективності CP на етапі навчання приймаються зовнішні дані про навчання. Pixel Overflow використовує фільтр Собеля для видалення джерела та цільових країв зображення, щоб підкреслити периферійні втрати та деталі.

Команда DRZ запропонував асиметричну пірамідальну структуру для SR-зображення [36] (див. рис. 2.3). Кожен рівень піраміди складається з каскаду компактних блоків стиснення (DCU), а шар субпікселів згортки використовується для формування залишкової карти для реконструкції HR-зображення. DCU складається з меншого модифікованого тісно пов'язаного блоку [11], за яким слідує згортка 1 × 1. Порівняно з оригінальним тісно пов'язаним блоком, запропонованим для класифікації, рівень нормалізації партії (BN) був видалений в DCU. На етапі навчання за програмою [5] була прийнята навчальна стратегія для досягнення кращих результатів із CP та скорочення часу навчання. Зокрема, DRZ спочатку тренує 2 × частину мережі, а потім поступово змішує новий рівень піраміди, щоб зменшити вплив на попередньо навчені шари. Навчання за програмою додає в

середньому 0,07 дБ PSNR для набору тестів DIV2K для шкал 2 × / 4 × / 8 ×, порівняно з 0,03 дБ для звичайного багатомасштабного навчання.

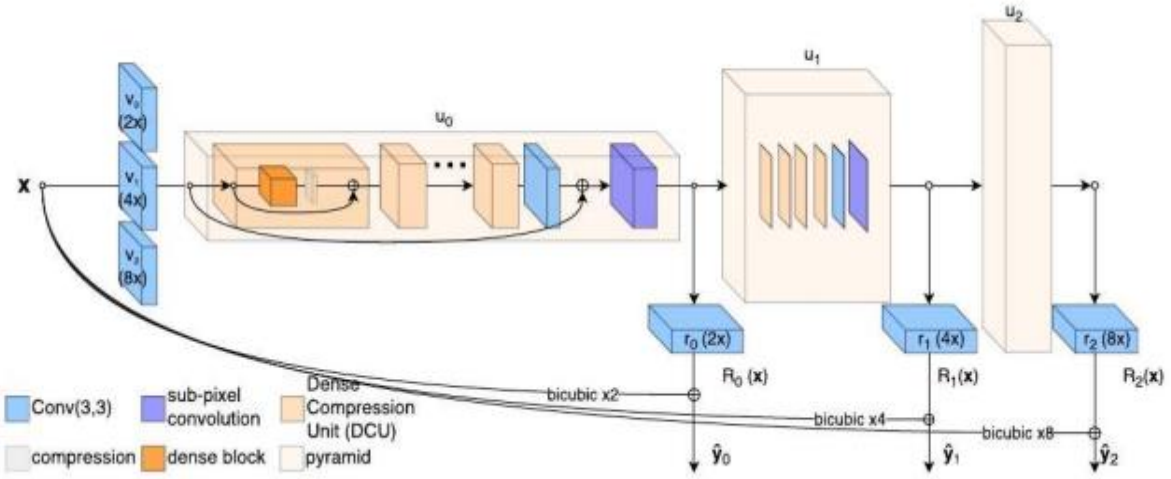


Рис. 2.3. Асиметрична пірамідальна архітектура DRZ з DCU

Команда UIUC-IFP запропонував широку мережу активації SR (WDSR, див. рис. 2.4), яка являє собою глибинну залишкову мережу SR (двошарові залишкові блоки), подібну до базової EDSR [21]. Щоб поліпшити продуктивність SR, WDSR модифікує оригінальний EDSR у трьох аспектах. По-перше, у порівнянні з EDSR, WDSR зменшує ширину шляху відображення ідентичності та збільшує ширину карт об'єктів перед функцією ReLU у кожному залишковому блоці (див. Рис. 2.4 (a)). Їхні експерименти показали, що WDSR надзвичайно ефективно покращує точність. По-друге, UIUCIFP відслідковує нещодавню роботу [8, 21, 31], яка видаляє рівні BN в залишкових блоках та застосовує нормалізацію ваги в їх підході WDSR, хоча впровадження нормалізації ваги в навчальні мережі SR може не сильно допомогти, дозволяють автори. використовувати вищий рівень освіти для навчання мережі. По-третє, WDSR видаляє деякі рівні згортки, що використовуються в EDSR, і безпосередньо генерує змішану оцінку SR (див. Рис. 2.4 (b)), така стратегія може покращити швидкість обробки, не впливаючи на точність мережі SR.

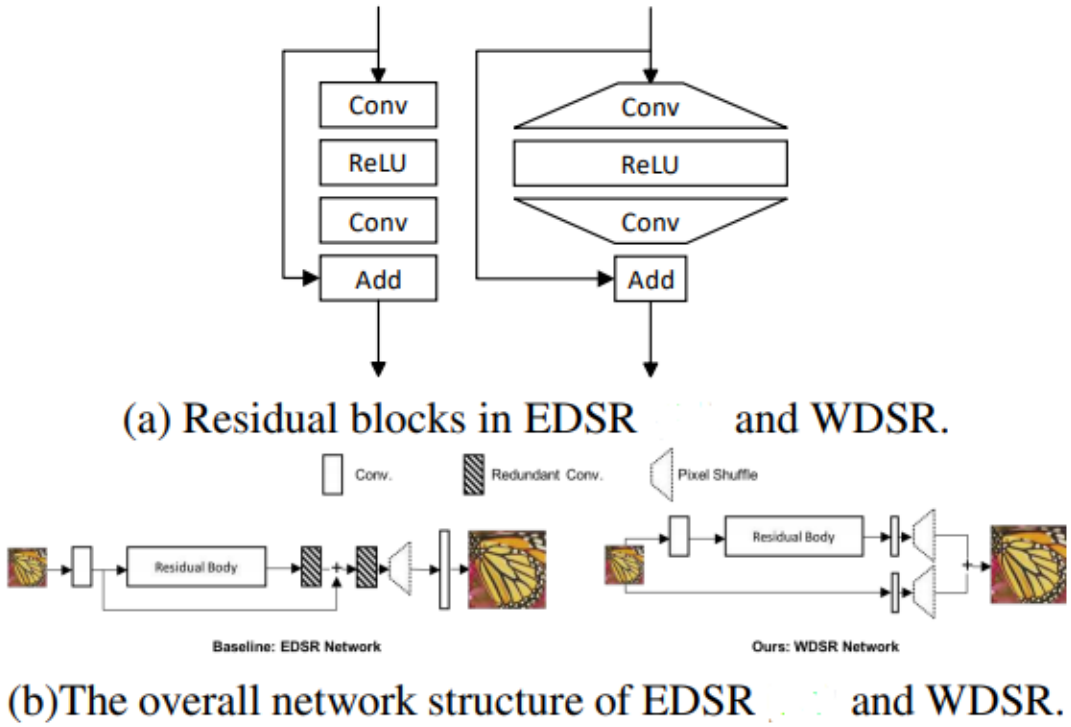


Рис. 2.4. Архітектура блоку WDSR UIUC-IFP

Команда PDN запропонований PolyDenseNet (PDN) (див. малюнок 2.5) для зображення SR. Основним будівельним блоком PDN є PolyDense Block (PDB), мотивований PolyNet [42] та DenseNet [11]. Кожен PDB містить три 5-шарові суцільні блоки і використовує три параметри α1, α2 та α3 для об'єднання вихідних даних суцільного блоку D1, D2 і D3 для отримання вихідних даних. Команда PDN також дослідила варіант PDN, побудувавши смуги пропускання між сусідніми PDB (див. Малюнок 2.5 (b)). Результати двох варіантів збираються під час тестування. На етапі навчання автори беруть вибірку зображень LR і обчислюють найкращі параметри зміщення щодо rt. основна істина на основі PSNR. Для масштабування яскравості автори регулюють середнє значення пікселів для LR-зображень, використовуючи відповідне зображення, яке відповідає оригіналу.

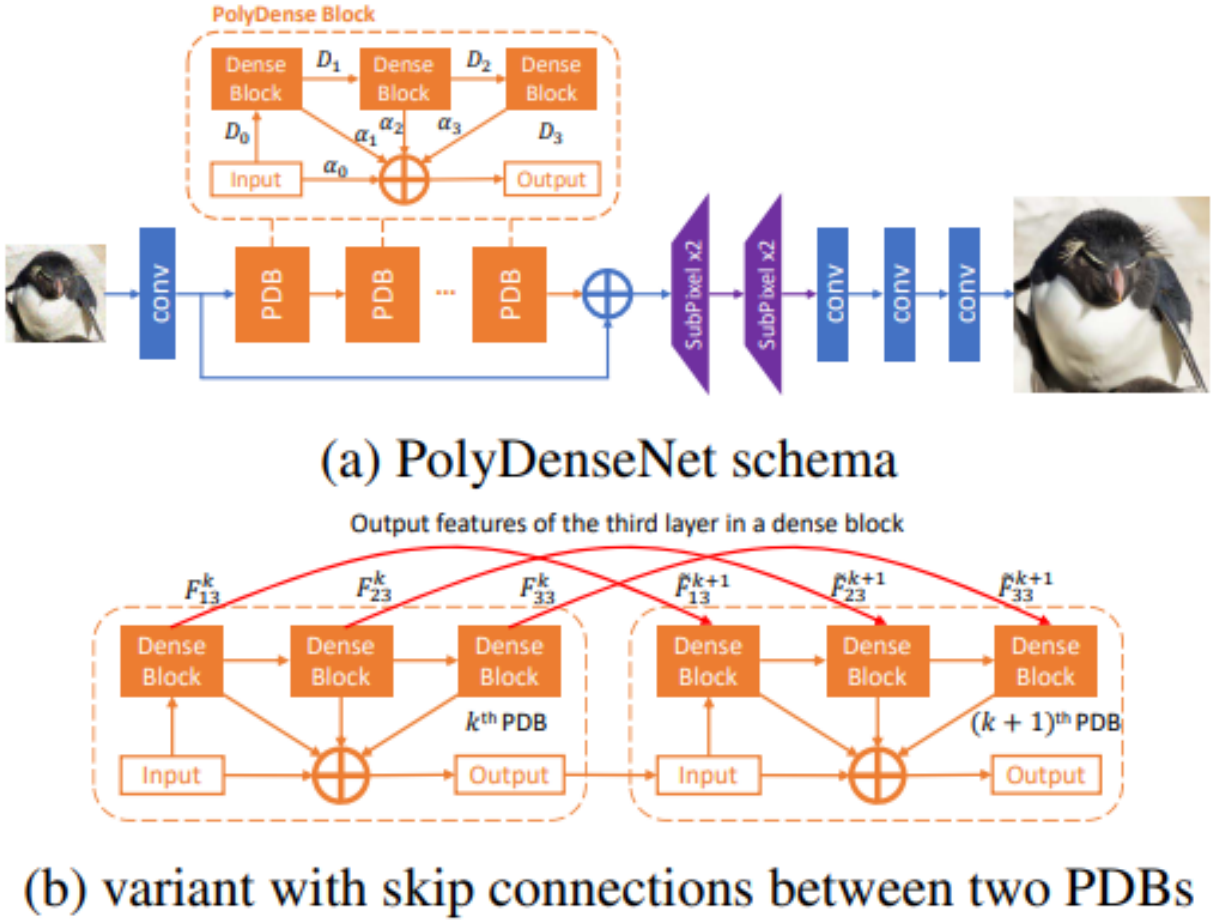


Рис. 2.5. PolyDenseNet, варіант PolyDenseBlock.

Команда BMIPL UNIST розклав початкові проблеми завдання NTIRE 2018 на підзадачі (SR у різних масштабах та відображення / розмиття) та запропонував ефективну мережу SR одного зображення на основі модулів [27] (EMBSR, див. рис. 2.6). Для окремої мережі модулів у SR вони запропонували EDSR-PP, який інтегрував агрегацію пірамід у зразковий шар EDSR [21], щоб краще використовувати як глобальну, так і локальну контекстну інформацію. Для модульної мережі шуму / розмиття вони запропонували мережу залишкової згортки (DnResNet), яка замінила блоки згортки DnCNN [40] на залишкові блоки з BN та масштабуванням. У процесі навчання DnResNet був зроблений етап попередньої обробки, який узгоджує вхідні та цільові зображення, і це, як кажуть, має вирішальне значення для хорошої продуктивності.

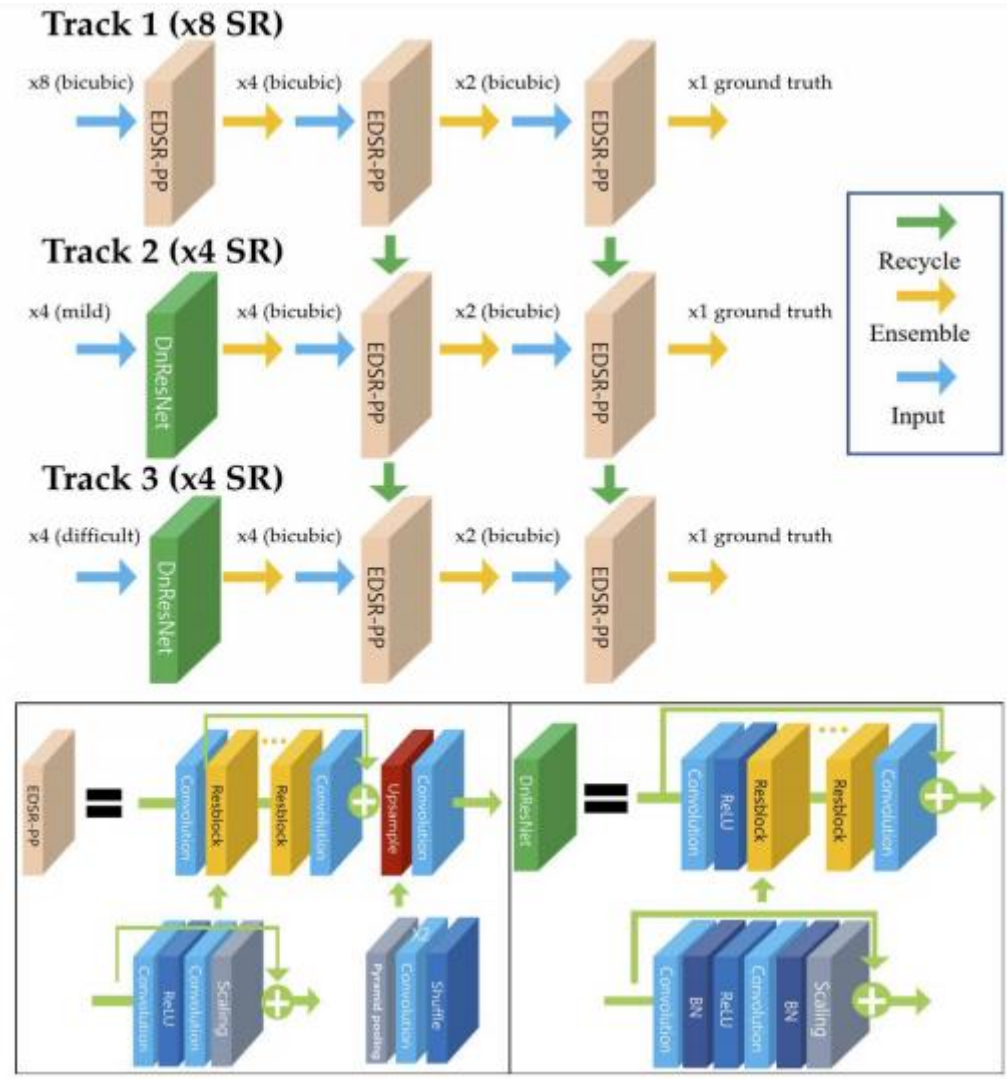


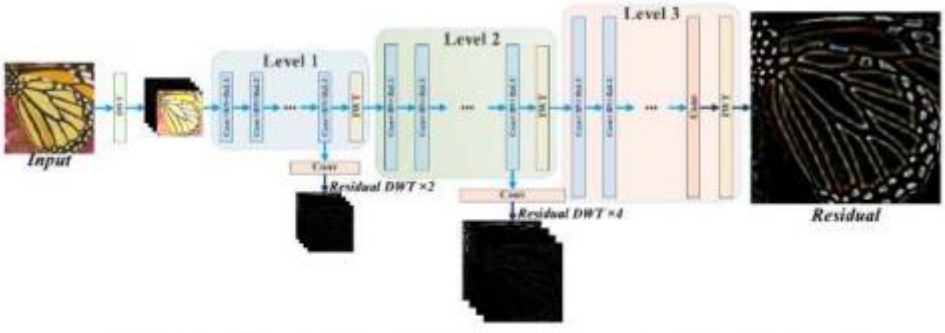
Рис. 2.6. Мережеві структури BMIPL UNIST.

Команда HIT-VPC використовував різні стратегії для вирішення бікубічних та реалістичних експериментальних параметрів. Для смуги 1 «Класичний бікубік» × 8 HIT-VPC запропонував інвертовану багаторівневу вейвлет-згорткову нейронну мережу (iMwCNN). Як показано на рис. 2.7 (a), iMwCNN розроблений як пірамідальна структура з багаторівневим перетворенням вейвлет-пакетів (WPT) [22]. Вхідне зображення LR спочатку бікубічно інтерполюється з коефіцієнтом масштабування 2, а коефіцієнти DWT інтерпольованого зображення є входом мережі. Щоб отримати коефіцієнт масштабування 8, трирівневі мережі використовувались для оцінки зворотних коефіцієнтів DWT. Фіксоване зворотне вейвлет-перетворення застосовується між кожним мережевим шаром для перетворення коефіцієнтів назад у простір зображення. Кожен мережевий рівень

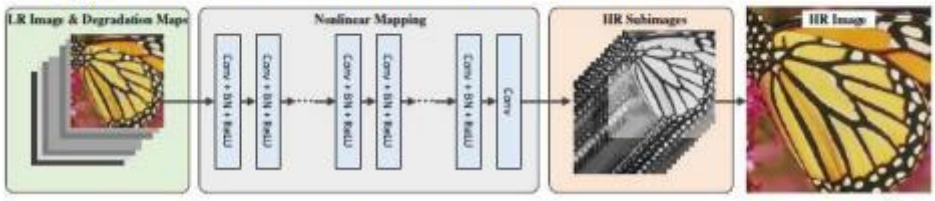
містить 8 згорткових шарів, а номер функціональної картки для трьох рівнів встановлюється відповідно 256, 256 та 128. На етапі навчання втрати визначаються на кожній шкалі оцінок.

Для реалістичних налаштувань (доріжки 2, 3 і 4) HIT-VPC побудовані на нещодавно запропонованій мережі із багатовисокою роздільною здатністю із надвисокою роздільною здатністю (SRMD) [39]. Як показано на рис. 2.7 (b) SRMD приймає параметризовану карту деградації та зображення LR як мережевий вхід і використовує 20 звивин + блоки BN + ReLU для оцінки HR-зображень. Щоб застосувати SRMD до треків 2, 3 та 4, ядро розмиття яких невідоме, HIT-VPC центрує ядра розмиття на основі найбільших значень для вирівнювання зображення LR та зображення HR та обчислює середнє (вирівняне) погіршення карти для кожна доріжка. Потім карти середньої деградації для кожної доріжки використовуються для зображень із надвисокою роздільною здатністю відповідних доріжок.

HIT-VPC ∗ ('lpj008', некласифікований) представив додаткові результати однієї моделі SRMD для треків 3 і 4. Вони використовували зображення Track 1 для оцінки оператора деградації та показали переваги несліпого SRMD: (i) він може відстежувати 3 і 4 в одній моделі, тоді як (ii) дає кращі результати з точним розмиттям серцевини, ніж сліпий SRMD.



(a) iMwCNN for the Track 1 'Classic Bicubic' ×8.



(b) SRMD for the realistic settings: Track 2, 3, and 4.

Рис. 2.7. Рішення HIT-VPC

Архітектура **команди Faceall Xlabs** заснована на EDSR [21]. Номер фільтра для кожного шару згортки змінено на 256 і використано 80 залишкових блоків.

Команда Duke Data Science також прийняв різні стратегії для двобічного та реалістичного середовища. Для бікубічного середовища вони використовували EDSR [21] з іншою стратегією навчання. Запроваджено теплі перезапуски та підхід косинусного світіння, що дозволяє мережі вибити локальні мінімуми. Для реалістичних налаштувань автори спочатку навчали DnCNN [22] та EDSR [21] окремо щодо зменшення шуму та SR, а потім вдосконалили дві мережі разом.

Команда SIA відтворено EDSR [21] та використано втрату Шарбоньє замість втрати L1, як запропоновано в [19]. Процесор використовувався під час тестування, щоб повною мірою скористатися повною згорткою зображення. Для колій 2, 3 та 4 пари поїздів спочатку вирівнювались на основі PSNR.

Команда KAIST-VICLAB [19] розробив 43-шаровий CNN для смуги 1 (див. Рис. 2.8), щоб поступово масштабувати вхідне зображення RGB до кінцевої цільової роздільної здатності. Два шари згортки субпікселів вставляються після 20-го та 40-го шарів згортки, щоб збільшити карти об'єктів на 2 та 4 відповідно. Мережа проходить грубу і делікатну підготовку: спочатку масштабування в 2 рази, потім у 4 рази і, нарешті, у 8 разів. Для треків 2, 3 та 4 компанія KAIST-VICLAB розробила рішення, що складається з: 1) чотири фільтри 5 × 5 вивчені між навчальними підзображеннями LR та HR, які використовуються для створення проміжного продукту з меншим рівнем шуму та регулювання HR яскравості зображення. 2) Вирівняні навчальні зображення з HR створюються шляхом вирівнювання вихідних навчальних зображень із HR із проміжними зображеннями. 3) 58-шаровий CNN з параметрами 2M тренується з використанням шумних навчальних зображень LR та нещодавно вирівняних навчальних зображень HR.

Зокрема, мережа приймає залишкове навчання, залишкові одиниці та два згорткові рівні субпікселів.

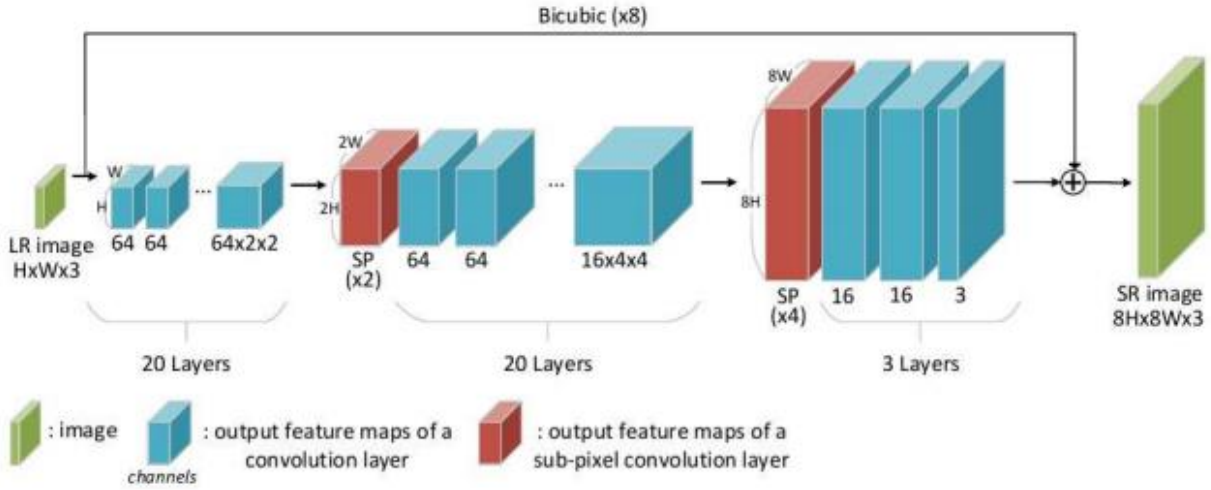


Рис.2.8. Запропонована мережева архітектура KAIST-VICLAB.

Архітектура та головні ідеї. Усі запропоновані методи, крім TSSR UW18, засновані на глибокому навчанні. Архітектура глибоких залишкових мереж (ResNet) [10] та архітектура щільних мереж (DenseNet) [11] є основою для більшості запропонованих методів. Для швидкого завершення, тобто навчання та тестування часу, більшість команд виконують основні операції SR в кімнаті LR. Кілька команд, такі як UIUC-IFP, BMIPL-UNIST, Pixel Overflow, будують свої методи на основі EDSR [21], найсучаснішого підходу та переможця попереднього виклику NTIRE 2017 SR [22]; тоді як інші команди, такі як Toyota-TI, HIT-VPC, DRZ, PDN, запропонували нові архітектури для SR.

Відновлення вірності. 4 найкращих методу "Classic Bicubic" досягли порівнянних значень PSNR (в межах 0,04 дБ). DeepSR, який посів 12 місце, відстає лише на 0,17 дБ від найкращого результату PSNR Toyota-TI. З точки зору реалістичних налаштувань, через наявність шуму та розмитості руху, смуги 2, 3 та 4, стратегія навчання та відтворення архітектури мережі однаково важливі. Хоча UIUC-IFP посідала 7 місце після класичного Bicubic, нижче DRZ і Duke Data

Science, вона зробила крок попереднього вирівнювання до етапу навчання і досягла найкращих результатів на реалістичних трасах 2 і 3, набагато кращих, ніж DRZ і Duke Data Science . PDN посідає перше місце на смузі 4, але, не представляючи результатів для інших смуг, ми не можемо сказати, чи є їх рішення / архітектура кращими, ніж UIUC-IFP.

Ансамблі та злиття. Більшість команд використовують псевдочутливі [23]. Записи інвертуються / обертаються, а результати HR вирівнюються та усереднюються для кращого прогнозування.

Час роботи / ефективність. BOE-SBG повідомив про найнижчий час роботи, 0,15 с для супер-роздільної здатності × 8 одиночних LR-зображень на графічному процесорі, але фінішував 17-м на "Classic Bicubic" на 0,63 дБ нижче, ніж найкращий рейтинговий метод Toyota-TI. З 4 найкращих методів на курсі Classic Bicubic, Rainbow зробила найкращий компроміс між ефективністю та продуктивністю. На графічному процесорі GTX 1080Ti веселка займає 6,75 секунди, тоді як Toyota-TI займає 35 секунд, щоб зображення LR створювало HR-зображення, включаючи самозбірку для обох методів.

Тренувальні данні. Масштабування даних (лише доріжка 1), прокрутка та обертання - інші поширені методи. Лише декілька команд, включаючи Pixel Overflow, використовували додаткові дані для навчання. Pixel Overflow використовував зображення з www.pexels.com, яке також є джерелом багатьох зображень DIV2K. HIT-VPC використовував зображення смуги 1 для оцінки зниження рейтингу операторів на смугах 3 і 4, тому їх запис "lpj008" у Таблиці 1 є лише довідковим і не класифікується в завданні.

У таблиці 1 представлені остаточні результати випробувань та оцінки змагань, тоді як у таблиці 2 - дані про час виконання та основні деталі, про які вони самі повідомили.

Таблиця 1

Результати порівняння та підсумкові рейтинги

а) Трек 1 Classic Bicubic × 8

Team	Author	PSNR	SSIM
Toyota-TI	iim_lab	25.455	0.7088
Pixel_Overflow	McCourt_Hu	25.433	0.7067
rainbow	zheng222	25.428	0.7055
DRZ	yifita	25.415	0.7068
Faceall_Xlabs	xjc_faceall	25.360	0.7031
Duke Data Science	admian98	25.356	0.7037
UIUC-IFP	jhyume	25.347	0.7023
Haiyun_XMU	cr2018	25.338	0.7037
BMIPL_UNIST	BMIPL_UNIST	25.331	0.7026
Ajou-LAMDA-Lab	nmhkahn	25.318	0.7023
SIA	mikigom	25.290	0.7014
DeepSR	enoch	25.288	0.7015
	Mrobot0	25.175	0.6960
reveal.ai	muneebaadil	25.137	0.6942
HIT-VPC	cskzh	25.088	0.6943
MCML	ghgh3269	24.875	0.7025
BOE-SBG	boe_sbg	24.822	0.6817
SRFun	ccook	24.819	0.6829
KAIST-VICLAB	JSChoi	24.817	0.6810
	zeweihe	24.773	0.6813
	jingliting	24.714	0.6913
CEERI	harshakoundinya	24.687	0.6719
APSARA	MingQiu	24.618	0.6817
UW18	zzsmg	24.192	0.6531
Baseline	Bicubic	23.703	0.6387

b) Реалістичні Треки 2,3 та 4 ×8

Team	Author	Track 2 Mild		Track 3 Difficult		Track 4 Wild	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
UIUC-IFP	jhyume	$23.631_{(1)}$	0.6316	$22.329_{(1)}$	0.5721	$23.080_{(2)}$	0.6038
PDN	xixihaha					$23.374_{(1)}$	0.6122
BMIPL_UNIST	BMIPL_UNIST	$23.579_{(2)}$	0.6269	$22.074_{(2)}$	0.5590		
HIT-VPC*	lpj008			22.249	0.5637	22.879	0.5936
HIT-VPC	cskzh	$23.493_{(3)}$	0.6174	$21.450_{(9)}$	0.5339	$22.795_{(3)}$	0.5829
SIA	mikigom	$23.406_{(5)}$	0.6275	$21.899_{(3)}$	0.5623	$22.766_{(4)}$	0.6023
KAIST-VICLAB	jschoi	$23.455_{(4)}$	0.6175	$21.689_{(6)}$	0.5434	$22.732_{(6)}$	0.5844
DRZ	yifita	$23.397_{(6)}$	0.6160	$21.592_{(8)}$	0.5438	$22.745_{(5)}$	0.5881
srFans	yyuan13	$23.218_{(9)}$	0.6222	$21.825_{(4)}$	0.5573	$22.707_{(7)}$	0.5932
Duke Data Science	adamian98	$23.374_{(7)}$	0.6252	$21.658_{(7)}$	0.5400		
	bighead	$23.247_{(8)}$	0.6165				
ISP_Team	hot_milk	$23.098_{(11)}$	0.6167	$21.779_{(5)}$	0.5550	$22.496_{(8)}$	0.5867
BOE-SBG	boe_sbg	$23.123_{(10)}$	0.6008	$21.443_{(10)}$	0.5275	$22.352_{(10)}$	0.5612
MCML	ghgh3269	$22.953_{(12)}$	0.6115	$21.337_{(11)}$	0.5354	$22.472_{(9)}$	0.5842
DeepSR	enoch	$21.742_{(15)}$	0.5572	$20.674_{(16)}$	0.5168	$21.589_{(12)}$	0.5444
	jingliting	$21.710_{(16)}$	0.5384	$20.973_{(12)}$	0.5187	$20.956_{(14)}$	0.5214
Haiyun_XMU	cr2018	$21.519_{(17)}$	0.5313	$20.866_{(13)}$	0.5072	$21.367_{(13)}$	0.5321
Ajou-LAMDA-Lab	nmhkahn	$21.240_{(18)}$	0.5376				
Juanluisgonzales	juanluisgonzales	$22.625_{(13)}$	0.5868				
APSARA	mingqiu			$20.718_{(15)}$	0.4977		
NMH	nmh			$20.645_{(17)}$	0.4890		
	join16	$20.453_{(19)}$	0.4928				
Baseline	Bicubic	$22.391_{(14)}$	0.5336	$20.830_{(14)}$	0.4631	$21.761_{(11)}$	0.4989

Таблиця 2

Звіти про час виконання кожного тестового зображення та деталі з інформаційних листів.

Team	runtime [s] Track 1	Track 2,3,4	Platform	CPU/GPU (at runtime)	Ensemble
Ajou-LAMDA-Lab	13.84	13.84	Pytorch	GTX 1080Ti	flip/rotation (×8)
APSARA	30	30	Tensorflow	GTX 1080Ti	flip/rotation (×8)
BOE-SBG	0.15	1.11	Pytorch	Nivida P100	-
bighead	-	1.5			
BMIPL_UNIST	2.52	4.68	Pytorch	?	flip/rotation (×8)
CEERI	12.23	-	Tensorflow,Keras	GTX 1080	-
DeepSR	9.89	1.83	Tensorflow	Titan X	flip/rotation (×8)
DRZ	11.65	2.91	Pytorch	Titan Xp	Track1: flip/rotation (×8)
Duke Data Science	6.99	18	???	Nivdia P100	flip/rotation (×8)
Faceall_Xlabs	7.31	-	Pytorch	GTX 1080	flip/rotation (×4)
Haiyun_XMU	14.52	2.14	Pytorch	Track 1: Titan X Track 2,3,4: GTX 1080	Track1: flip/rotation (x8)
HIT-VPC	0.26	0.2	Matconvnet	GTX 1080Ti	-
ISP_Team	-	2.1	Tensorflow	Titan X	-
jingliting	1.27	0.72	???	???	-
join16	-	4.12	???	GTX 1080	-
juanluisgonzales	-	0.02	???	???	-
KAIST-VICLAB	0.44	1.60	Track1: Matconvnet, Track2,3,4: Tensorflow	Titan Xp	Track1: - Track2,3,4: flip/rotation (×8)
MCML	5.95	1.08	Tensorflow	GTX 1080	Track1: flip/rotation (×8)
Mrobot0	10	-	???	???	-
NMH	-	3.31	???	???	-
PDN	-	13.07	Pytorch	4 Titan Xp	Ensemble two variations of the proposed methods
Pixel_Overflow	20	-	Tensorflow	Nvidia P100	-
rainbow	6.75	-	Pytorch	GTX 1080Ti	flip/rotation (×8)
reveal.ai	92.95	-	Pytorch	Tesla K80	flip/rotation (×8)
SIA	396.0	396.0	Tensorflow	CPU	flip/rotation (×8)
srFans	-	0.10	Pytorch	Tesla K80	-
SRFun	1	-	Tensorflow	GTX 1080Ti	-
Toyota-TI	35	-	Pytorch	Titan X	flip/rotation (×8)
UIUC-IFP	5.03	7.28	Pytorch	P100	flip/rotation (×8)
UW18	300	-	Matlab	Intel Core i7-6700K CPU @ 4.00GHz	-
zeweihe	1.02	-	???	???	-

2.2.2 Інструменти покращення зображень

Deep Photo Enhancer - поліпшення картинок поганої якості за допомогою глибоких нейронних мереж**.** Deep Photo Enhancer використовує GAN (Generative Adversarial Networks) та дивні алгоритми навчання. Deep Photo Enhancer забезпечує метод покращення зображення, заснований на вивченні фотографій. Нейрони вчаться знаходити загальні риси в серії зразків зображення (наприклад, рівень контрастності, баланс білого, кольорова гама), а потім застосовувати ці функції до покращеного зображення, щоб зберегти значення оригінального зображення. Цей метод вимагає використання високоякісних оригінальних зображень і може бути додатково налаштований.

Архітектура Deep Photo Enhancer складається з генератора (Рисунок 2.9), дискримінатора (Рисунок 2.10), односторонньої вулиці (Рисунок 2.11) та двостороннього GAN (Рисунок 2.12):

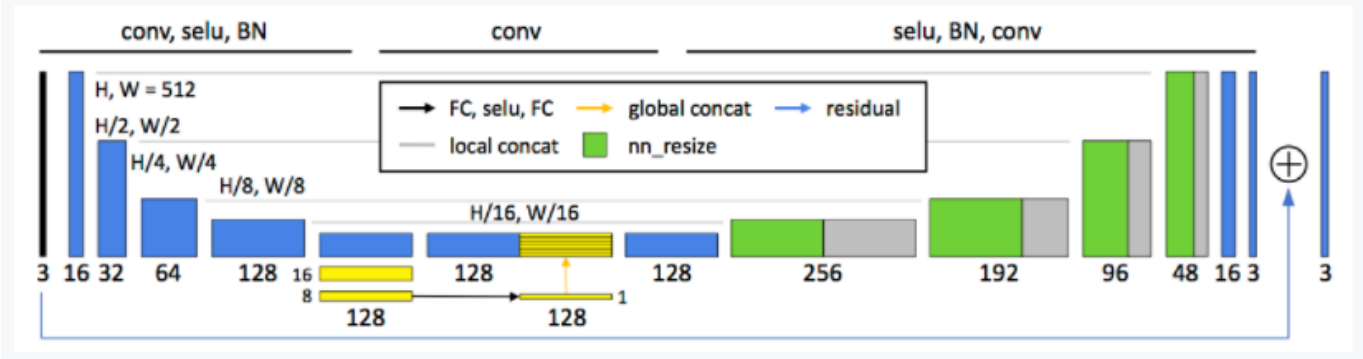


Рисунок 2.9. Генератор

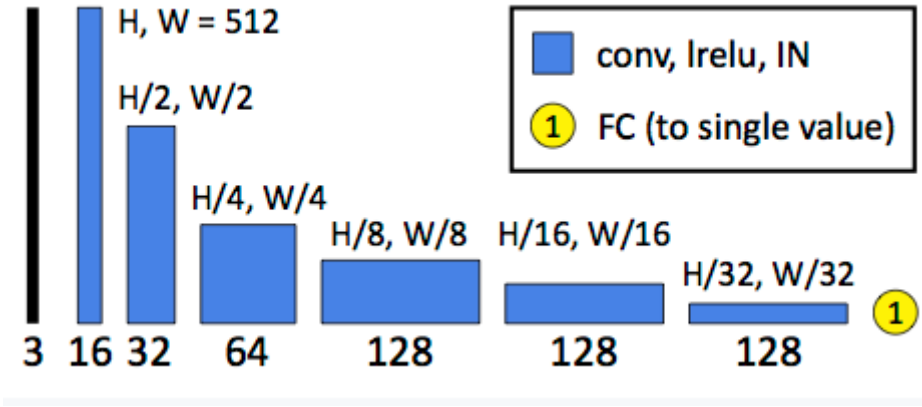


Рисунок 2.10. Дискримінатор

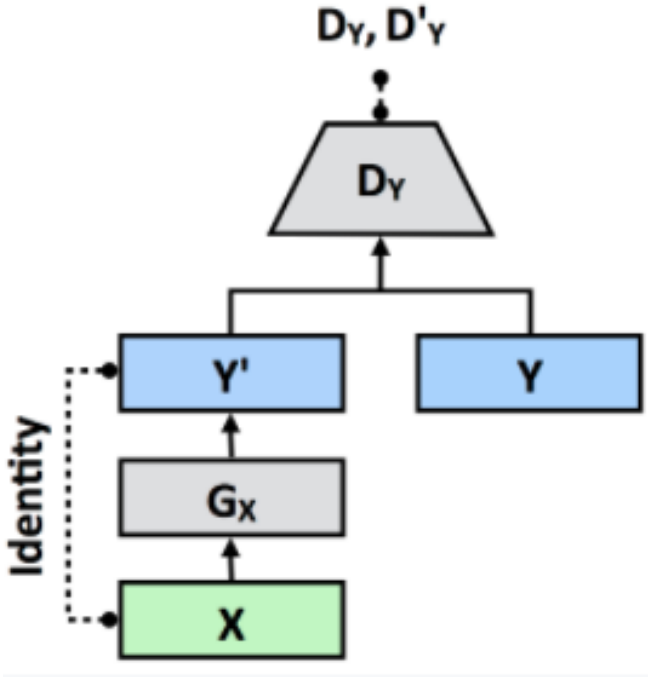


Рисунок 2.11. Одностороння GAN

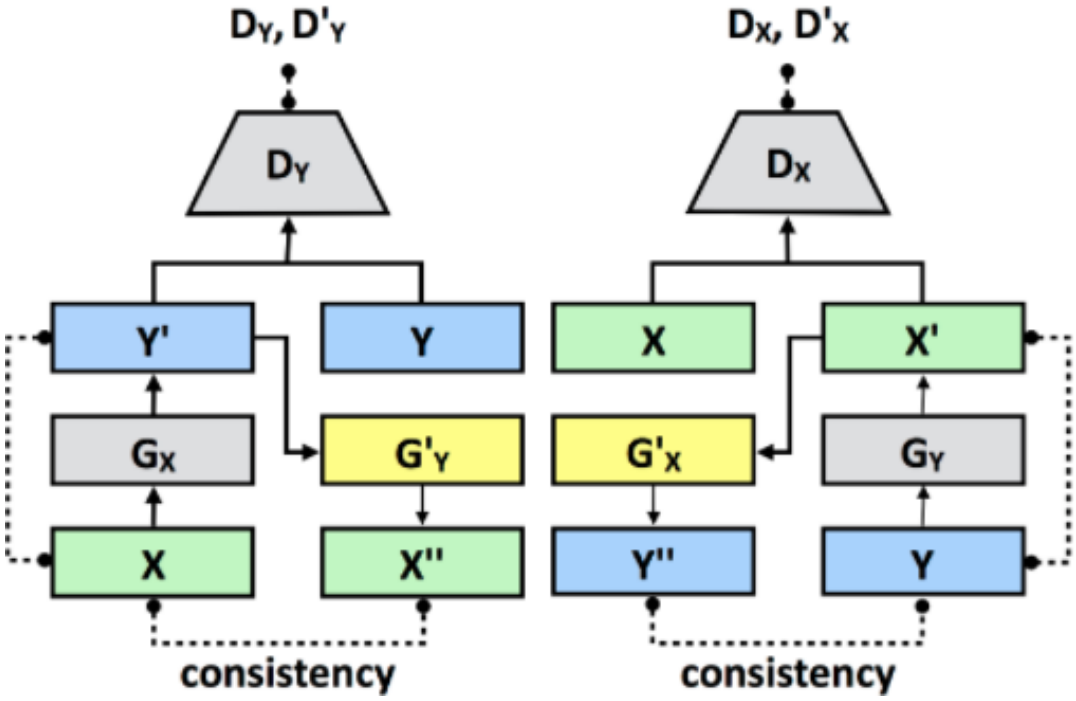


Рисунок 2.12. Двостороння GAN

Генератор заснований на U-Net, який спочатку пропонувався для біомедичної сегментації зображень, але згодом також показав високу ефективність

у багатьох завданнях. Однак UNet не дуже добре виконує наше завдання. Ми припускаємо, що U-Net не містить загальних функцій. Наша система зору зазвичай адаптується до загальних умов освітлення та налаштувань сцени. Так само камери мають налаштування сцени і часто використовують різні типи регулювання залежно від поточного налаштування. Глобальні функції можуть розкривати інформацію високого рівня, таку як категорія сцени, тип об’єкта або загальні умови освітлення, що може бути корисним для окремих пікселів для визначення їх локальних налаштувань. Ось чому ми додаємо глобальні можливості до U-Net.

Щоб підвищити ефективність моделі, глобальне вилучення функцій використовує ту саму контрактну частину U-Net з локальним вилученням функцій для перших п’яти шарів. Кожен етап зменшення складається з 5 × 5 фільтрації з кроком 2 з подальшою активацією SELU [15] та нормалізацією партії [12]. Враховуючи карту об’єктів 32 × 32 × 128 5-го шару, для глобальних об’єктів карта об’єктів додатково зменшується до 16 × 16 × 128, а потім 8 × 8 × 128, виконуючи описаний вище крок зменшення. Потім функціональна карта 8 × 8 × 128 масштабується до повністю зв’язаного шару 1 × 1 × 128, за яким слідує шар активації SELU, а потім ще один повністю зв’язаний шар. Витягнуті глобальні об'єкти розміром 1 × 1 × 128 потім копіюються та об’єднуються на 32 × 32 копії на низькорівневій карті об’єктів 32 × 32 × 128, в результаті чого створюється карта об’єктів 32 × 32 × 256, яка об’єднує як локальні, так і глобальні функції разом. Потім розширений шлях U-Net виконується на об'єднаній карті об'єктів. Нарешті, ідея залишкового навчання була прийнята, оскільки вона виявилася ефективною для завдань з обробки зображень та корисною для конвергенції. Тобто генератор дізнається лише різницю між вхідним зображенням та зображенням мітки.

Глобальні особливості досліджувались за допомогою інших завдань обробки зображень, таких як забарвлення. Однак їх модель вимагає додаткової моніторингової мережі, навченої явними мітками сцен. Для багатьох програм важко визначити чіткі мітки. Новизна моделі полягає у використанні самої U-Net для кодування неявного вектора функцій, який описує глобальні функції, корисні для цільового додатка.

Ми оцінили різні архітектури мережі для генератора. (1) DPED: Оскільки ми оцінюємо лише генератори, ми просто взяли генератор з їхньої архітектури GAN. (2) 8RESBLK [12]: Генератор використовувався в CycleGAN та UNIT. (3) FCN: повністю згорнута мережа фільтрів доступу. (4) CRN: Архітектура була використана для синтезу реалістичних зображень семантичних міток. (5) U-Net. Залишкова освіта доповнюється всіма ними. Через обмеження розміру зображення та обсягу пам'яті кількість функцій у першому шарі обмежена 16. Інакше загальну архітектуру не можна запам'ятати.

	DPED	8RESBLK	FCN	CRN	U-Net	Ours
PSNR	25.50	31.46	31.52	33.52	31.06	**33.93**
SSIM	0.911	0.951	0.952	0.972	0.960	**0.976**

Рис. 2.13 Середня точність різних архітектур мережі щодо наближення швидкої локальної лапласівської фільтрації.

Рисунок 2.13 показує як середні значення PSNR, так і SSIM для всіх порівнянних архітектур з точки зору передбачуваної швидкої локальної фільтрації Лапласа для 500 тестових зображень із набору даних MITAdobe 5K. Додаючи глобальні функції, запропонована архітектура забезпечує майже 3 дБ посилення порівняно зі своїм аналогом без глобальних функцій і перевершує всі порівнянні архітектури. Генератор чудово апроксимує швидкий локальний фільтр Лапласа з PSNR на 33,93 дБ, кращий, ніж FCN, розроблений для таких завдань.

	DPED	8RESBLK	FCN	CRN	U-Net	Ours
PSNR	21.76	23.42	20.66	22.38	22.13	**23.80**
SSIM	0.871	0.875	0.849	0.877	0.879	**0.900**

Рис. 2.14 Середня точність різних мережевих архітектур щодо прогнозування ретушованих зображень фотографами.

Рисунок 2.14 показує ефективність цих архітектур при прогнозуванні ретушованих зображень. Це завдання набагато складніше, оскільки людська ретуш

може бути більш складною та менш суперечливою, ніж алгоритмічні фільтри. Знову ж таки, запропонована глобальна архітектура U-Net перевершує інші.

2.2.3 IBM / MAX Image Resolution Enhancer - нейромережа для відновлення стислих фото

Ця розширювана модель дозволяє збільшити розмір пікселізованого зображення в 4 рази, одночасно створюючи фотореалістичні деталі за допомогою GAN (потік тензора SRGAN), натренованого на 600 000 зображень з OpenImages V4. Ідеальним вхідним зображенням є файл PNG із розміром від 100x100 до 500x500 пікселів, бажано без подальшої обробки. Модель може генерувати деталі з нерівного зображення, але вона не підходить для виправлення розмитих зображень.

GAN використовує методи, описані в дослідженні "Фотореалістичне надвисоке роздільне зображення з використанням генеративно протилежної мережі" [16].Щоб відрізнити реальні HR-зображення від сформованих зразків SR, ми навчаємо дискримінаційну мережу. Архітектура показана на рисунку 2.15. Ми дотримуємося архітектурних рекомендацій, узагальнених Редфордом [24], і використовуємо активацію LeakyReLU ($\alpha = 0,2$), уникаючи максимальної інтеграції мережі. Мережа дискримінатора містить вісім згорткових шарів зі збільшенням кількості 3×3 фільтруючих ядер, що подвоюється з 64 до 512 ядер, як у мережі VGG. Поступова звивка використовується для зменшення роздільної здатності зображення кожного разу, коли кількість функцій подвоюється. Отримані карти функцій 512 супроводжуються двома щільними шарами та остаточною функцією активації сигмовидної системи для отримання ймовірності класифікації вибірки.

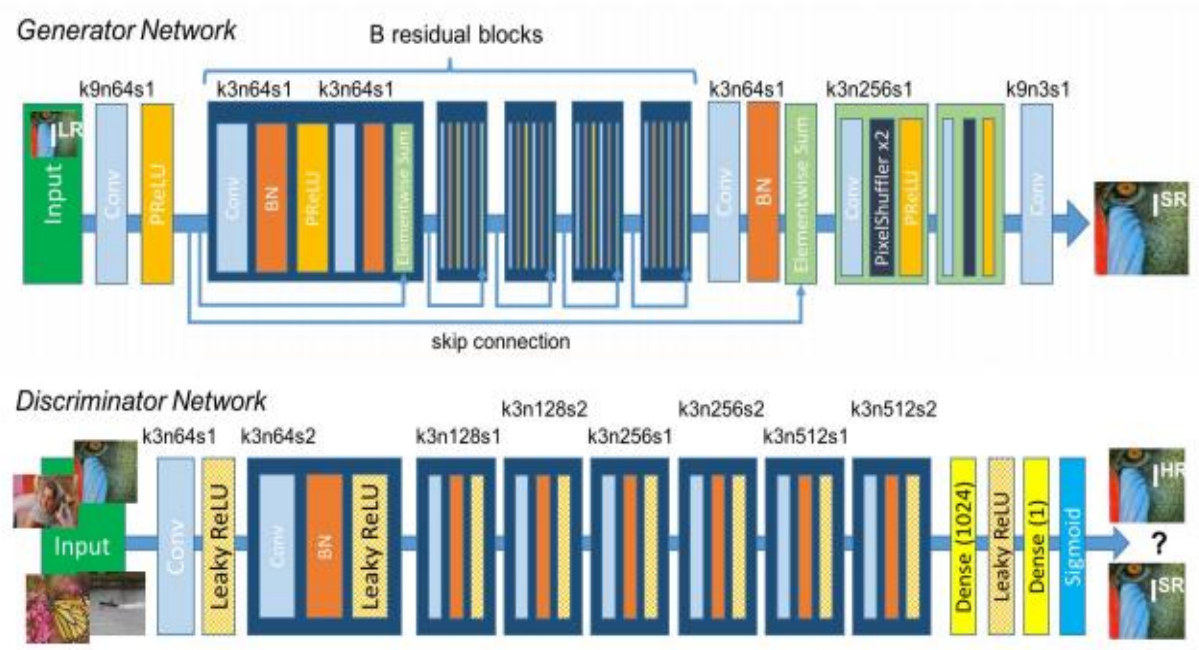


Рис. 2.15. Архітектура мережі генераторів та дискримінаторів з відповідним розміром ядра (k), кількістю карт функцій (n) та кроком (s), вказаними для кожного згорткового шару

Set5	Author's SRGAN	This SRGAN
PSNR	29.40	29.56
SSIM	0.85	0.85

Set14	Author's SRGAN	This SRGAN
PSNR	26.02	26.25
SSIM	0.74	0.72

BSD100	Author's SRGAN	This SRGAN
PSNR	25.16	24.4
SSIM	0.67	0.67

Рис.2.16. Порівняння показників вихідної та використаної GANів.

Ефективність цього впровадження оцінювали за трьома наборами даних: Set5, Set14 та BSD100. Оцінювались показники PSNR (пікове відношення сигнал / шум) та SSIM (індекс структурної подібності), хоча MOS (середній бал) вважається найбільш сприятливою метрикою. По суті, реалізація SRGAN має кращий показник PSNR або SSIM, щоб отримати результат, який є більш привабливим для людського ока. У результаті виходить колекція оригінальних зображень із чіткішими та реалістичнішими деталями.

2.2.4 Deep Image Prior

Загалом, ми виявили, що глибокі архітектури без занадто великої кількості коротких шляхів від входу до результату встановлюють дуже добрі глибокі пріоритети. Занадто багато коротких шляхів порушує вивчення звичайних шаблонів зображень, що призводить до зберігання пам’яті на рівні пікселів та дефіциту попередніх зображень. Хоча можливі й інші варіанти, ми в основному експериментували з повністю згортковими архітектурами, де вкладений z має однакову просторову роздільну здатність, як вихідний сигнал мережі f0. Ми використовуємо архітектор декодера ("пісочний годинник") (можливо, з пропуском посилання) для f0 у всіх наших експериментах, якщо не зазначено інше, невелика кількість гіперпараметрів змінюється. Хоча найкращих результатів можна досягти, ретельно змінивши архітектуру для конкретного завдання (і, можливо, для конкретного зображення), ми виявили, що широкий спектр гіперпараметрів та архітектур дає прийнятні результати. Ми використовуємо LeakyReLU [4] як нелінійність. В якості методу вибіркової вибірки ми просто використовуємо кроки, реалізовані в модулях згортки. Ми також спробували середнє / максимальне агрегування та зменшення вибірки з ядром Lanczos, але не виявили постійної різниці між ними. Як селективна виборча операція, ми обираємо між двокамерною передискретизацією та безпосереднім оточенням. Альтернативним методом збільшення зразка може бути використання перенесених обмоток, але результати, які ми отримали, були гіршими. Ми використовуємо

відбиваючу підкладку замість нульової підкладки в згорнутих шарах скрізь, крім експерименту, щоб змінити функцію.

При адаптації мереж ми іноді використовуємо регуляризацію на основі шуму. Тобто з кожною ітерацією ми порушуємо вхід z додатковим нормальним шумом. Хоча ми виявили, що така регуляризація перешкоджає процесу оптимізації, ми також зазначили, що мережа змогла оптимізувати свою ціль до нуля незалежно від адитивних дисперсій шуму (тобто мережа завжди могла адаптуватися до розумної дисперсії для великої кількості оптимізації кроки).

Ми виявили, що процес оптимізації має тенденцію до дестабілізації, оскільки втрати зменшуються і наближаються до певної величини. Дестабілізація спостерігається, оскільки втрати значно зростають і створюється розмите зображення f0. З цього моменту дестабілізації втрати знову зменшуються, поки не дестабілізуються знову. Для вирішення цієї проблеми ми просто відстежуємо оптимізаційні втрати і повертаємося до параметрів попередньої ітерації, якщо різниця у втратах між двома послідовними ітераціями перевищує певний поріг.

Нарешті, ми використовуємо оптимізатор ADAM [5] у всіх наших експериментах. Всі експерименти реалізовані в PyTorch. Нижче ми наводимо решту деталей архітектури мережі (рис. 2.17). Після цього надається велика кількість додаткових експериментальних результатів. Електронне збільшення рекомендується майже для всіх зображень.

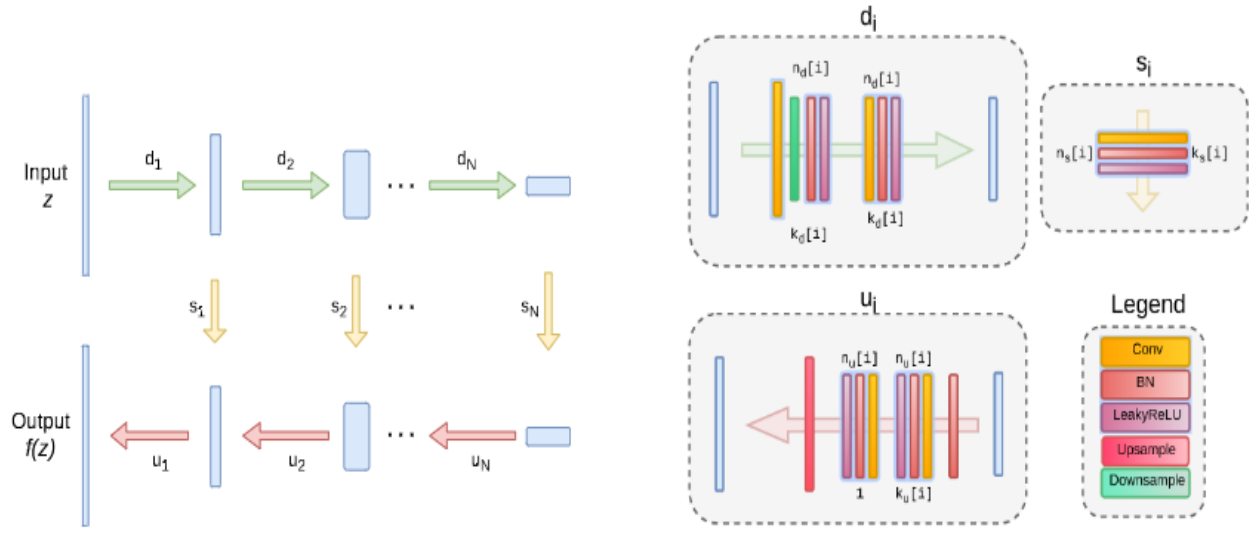


Рис.2.17. Архітектура Deep Image Prior

Результати порівняння показників технології Deep Image Prior з іншими поширеними методами на різних датасетах (Set14, Set5) та з різними коефіцієнтами збільшення (x4 та x8) та наведені нижче.

	Baboon	Barbara	Bridge	Coastguard	Comic	Face	Flowers	Foreman	Lenna	Man	Monarch	Pepper	Ppt3	Zebra
No prior	22.24	24.89	23.94	24.62	21.06	29.99	23.75	29.01	28.23	24.84	25.76	28.74	20.26	21.69
Bicubic	22.44	25.15	24.47	25.53	21.59	31.34	25.33	29.45	29.84	25.7	27.45	30.63	21.78	24.01
TV prior	22.34	24.78	24.46	25.78	21.95	31.34	25.91	30.63	29.76	25.94	28.46	31.32	22.75	24.52
Glasner et al.	22.44	25.38	24.73	25.38	21.98	31.09	25.54	30.4	30.48	26.33	28.22	32.02	22.16	24.34
Ours	22.29	25.53	24.38	25.81	22.18	31.02	26.14	31.66	30.83	26.09	29.98	32.08	24.38	25.71
SRResNet-MSE	23.0	26.08	25.52	26.31	23.44	32.71	28.13	33.8	32.42	27.43	32.85	34.28	26.56	26.95
LapSRN	22.83	25.69	25.36	26.21	22.9	32.62	27.54	33.59	31.98	27.27	31.62	33.88	25.36	26.98

Рис.2.18. Деталізоване збільшення роздільної здатності PSNR на датасеті Set14 з коефіцієнтом збільшення x4.

	Baboon	Barbara	Bridge	Coastguard	Comic	Face	Flowers	Foreman	Lenna	Man	Monarch	Pepper	Ppt3	Zebra
No prior	21.09	23.04	21.78	23.63	18.65	27.84	21.05	25.62	25.42	22.54	22.91	25.34	18.15	18.85
Bicubic	21.28	23.44	22.24	23.65	19.25	28.79	22.06	25.37	26.27	23.06	23.18	26.55	18.62	19.59
TV prior	21.3	23.72	22.3	23.82	19.5	28.84	22.5	26.07	26.74	23.53	23.71	27.56	19.34	19.89
SelfExSR	21.37	23.9	22.28	24.17	19.79	29.48	22.93	27.01	27.72	23.83	24.02	28.63	20.09	20.25
Ours	21.38	23.94	22.2	24.21	19.86	29.52	22.86	27.87	27.93	23.57	24.86	29.18	20.12	20.62
LapSRN	21.51	24.21	22.77	24.1	20.06	29.85	23.31	28.13	28.22	24.2	24.97	29.22	20.13	20.28

Рис.2.19. Деталізоване збільшення роздільної здатності PSNR на датасеті Set14 з коефіцієнтом збільшення x8.

	Baby	Bird	Butterfly	Head	Woman
No prior	30.16	27.67	19.82	29.98	25.18
Bicubic	31.78	30.2	22.13	31.34	26.75
TV prior	31.21	30.43	24.38	31.34	26.93
Glasner et al.	32.24	31.1	22.36	31.69	26.85
Ours	31.49	31.8	26.23	31.04	28.93
LapSRN	33.55	33.76	27.28	32.62	30.72
SRResNet-MSE	33.66	35.1	28.41	32.73	30.6

Рис.2.20. Деталізоване збільшення роздільної здатності PSNR на датасеті Set5 з коефіцієнтом збільшення x4.

	Baby	Bird	Butterfly	Head	Woman
No prior	26.28	24.03	17.64	27.94	21.37
Bicubic	27.28	25.28	17.74	28.82	22.74
TV prior	27.93	25.82	18.4	28.87	23.36
SelfExSR	28.45	26.48	18.8	29.36	24.05
Ours	28.28	27.09	20.02	29.55	24.5
LapSRN	28.88	27.1	19.97	29.76	24.79

Рис.2.21. Деталізоване збільшення роздільної здатності PSNR на датасеті Set5 з коефіцієнтом збільшення x8.

РОЗДІЛ 3
АРХІТЕКТУРА ТА ВИМОГИ ДО СИСТЕМИ ОБРОБКИ ЗОБРАЖЕНЬ

3.1. Функціональні вимоги

Функціональні вимоги - це особливий спосіб опису послуг та дій, які має виконувати програмне забезпечення. Він надає опис програмної системи та її компонентів. У програмній інженерії та системній інженерії функціональні вимоги можуть варіюватися від абстрактного твердження високого рівня про необхідність відправника до детальних математичних специфікацій функціональних вимог. Функціональні вимоги до програмного забезпечення допомагають визначити прогнозовану поведінку системи.

Після детального аналізу галузі та ретельного вивчення пропонованих на ринку рішень було вирішено розробити систему обробки зображень із такими функціональними вимогами:

1. Дозвольте користувачеві завантажувати цифрові фотографії у форматі img або png.

2. Оцініть ступінь пошкодження специфікації кольору.

3. Визначення втрати яскравості, контрастності та експозиції зображення.

4. Покращити фотографію користувача, змінивши параметри, виявлені системою.

5. Демонстрація покращеного зображення "до - після".

6. Виявлення текстури на зображенні.

7. Видалення шуму.

8. Видаліть розмитість або розкол.

Діаграма прецедентів використовується для поліпшення розуміння функціональних можливостей системи. Він розробляється після визначення основних функціональних вимог до системи. Зручно та ефективно він описує типові взаємодії між користувачем та системою та описує процес її роботи.

Прикладами використання є методи збору, моделювання та уточнення системних вимог. Випадок використання відповідає набору поведінки, які система може виконувати у взаємодії зі своїми суб'єктами, і які дають видимий результат, який сприяє досягненню її цілей. Актори представляють роль користувачів та інших систем у взаємодії.

При аналізі вимог до їх ідентифікації варіант використання називається відповідно до конкретної мети користувача, яку він представляє для свого головного героя. Справа додатково деталізується текстовим описом або додатковими графічними моделями, що пояснюють загальну послідовність дій та подій, а також такі варіанти, як особливі умови, винятки або ситуації помилок.

Відповідно до Інженерії знань програмного забезпечення (SWEBOK), випадки використання належать до сценарійних методів виявлення вимог, а також до методів аналізу на основі моделей. Але випадки використання також підтримують збір вимог на основі історії, придбання додаткових вимог, системну документацію та приймальні випробування.

Ілюстрація взаємозв'язку між варіантами використання розробленої програмної системи обробки зображень із використанням штучного інтелекту для користувачів представлена у вигляді прецедентної схеми на рис. 3.1.

Рисунок 3.1. Діаграма Прецедентів

3.2 Нефункціональні вимоги.

Нефункціональні вимоги - це специфікація, яка описує можливості системи та обмеження, що покращують її функціональність. Це можуть бути швидкість, безпека, надійність тощо.

Структура ISO / IEC 25000 визначає нефункціональні вимоги як вимоги до якості системи та якості програмного забезпечення. Одне з найважливіших джерел знань для бізнес-аналітиків, BABOK пропонує термін нефункціональні вимоги (NFR), що є найпоширенішим визначенням сьогодні. Однак ці позначення враховують один і той же тип речовини: вимоги, що описують характеристики, а не поведінку продукту.

3.2.1. Вимоги до надійності, доступності та ремонтопридатності системи

Система обробки зображень повинна бути спроектована відповідно до сучасних програмних протоколів безпеки, щоб уникнути критичних ситуацій під час використання.

Надійність продукту повинна гарантуватися:

1. Захист технічних засобів електропостачання шляхом використання джерел безперебійного живлення;

2. Дублювання сховищ інформації.

Наступні системні повідомлення повинні оброблятися в екстрених випадках HTTP:

1. Сторінку не знайдено (404 не знайдено);

2. Сервер не відповідає (500 внутрішніх помилок сервера).

Підключення системи до Інтернету повинно здійснюватися через захищений канал та захищене налаштованим брандмауером.

Усі передачі персональних даних повинні бути зашифровані. Особисті дані користувачів можуть не зберігатися на сервері.

Якщо під час роботи системи виникає помилка, слід запропонувати спосіб відновлення неповного сеансу.

3.2.2. Вимоги ергономіки і технічної естетики системи

Розроблений програмний продукт слід оптимізувати за допомогою обробки графіки та забезпечувати найшвидшу швидкість завантаження.

1. Система повинна відображати підказки в місцях, які можуть бути складними для клієнта.

2. Під час використання програми переконайтесь, що місцезнаходження користувача чітке.

3. Зменшіть витрати часу споживача на навігацію.

4. Відображається правильно з усіма розширеннями екрана в популярних сучасних операційних системах.

3.3.3. Вимоги до апаратного забезпечення

Мінімальні системні вимоги до комп'ютера користувача:

1. Браузери Google Chrome, Internet Explorer, Opera або Mozilla Firefox;
2. Операційна система Windows XP/Vista/7/8/8.1/10.
3. Процесор: Intel Core i5 4-го покоління або AMD FX-6300;
4. Об'єм оперативної пам'яті: 2 GB DDR3.
5. 20 Gb вільної пам'яті на жорсткому диску;

3.3.4. Обмеження, правила та стандарти

Система обробки зображень буде реалізована мовою програмування Python 3.7. Функціонування програми буде проходити під керівництвом операційних систем MS Windows 8, MS Windows 8.1, MS Windows 10.

3.3.5. Вимоги до інтерфейсу користувача

1. Фон програми повинен бути білого кольору, кнопок – сірі;
2. Текст повинен бути чорного кольору, основний шрифт Times New Roman розміром 12 px.
3. Поля для вводу білого кольору з темно-сірими рамками.

3.3.6.Вимоги до продуктивності

Система повинна:

1. Генерувати виклики менш ніж за 10 секунд;
2. Функціонувати цілодобово;
3. Виводити користувачу повідомлення про стан не більше ніж через 5 секунд після того, як користувач надіслав інформацію.
4. Швидкість оновлення зображення (фреймрейт) має складати не менше 30 кадрів/сек;

3.3. Архітектура мережі

Спираючись на детальний аналіз предметної області та сучасні тенденції підходу до розробки нейронних мереж, було вирішено обрати основними компонентами моделі:

- залишкові щільні блоки (RDB),
- мережа вилучення характеристик (FENet)
- допоміжна мережа уваги (AAN).

Ці компоненти прив'язані до скрізної архітектури для комбінованого навчання системи обробки зображень. Проведем більш детальний аналіз і розглянемо їх уважніше.

Залишкові щільні блоки (RDB) складаються з трьох наборів шарів conv, кожен з яких є нормалізованими пакетами (BN) і запускає нелінійний активатор функції ReLU. Як показано на малюнку 3.2, вхід і вихід кожного шару поєднуються з наступними шарами. Кожен рівень конв перевіряє 64 фільтри заданого розміру ядра; їх результати потім поєднуються з шаром 1 × 1 conv для локального залишкового навчання.

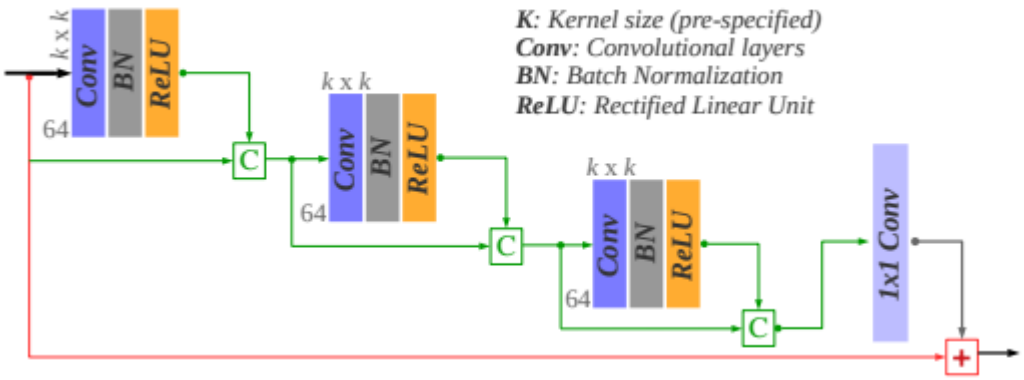


Рис. 3.2. Залишковий щільний блок (RDB)

Мережа вилучення функцій (FENet) використовує RDB як будівельні блоки для забезпечення двоступеневого навчання "залишок у залишку". Як показано на малюнку 3.3, дві паралельні гілки на першому кроці використовують

вісім блоків RDB, кожен для окремого дослідження фільтрів 3 × 3 та 5 × 5 у вхідному просторі зображення. Потім ці фільтри об'єднують і передають загальній гілці для другого етапу навчання.

Наступним кроком є використання чотирьох RDB з фільтрами 3 × 3, які в підсумку генерують 32 карти функцій. Це робиться для вивчення локальних щільних інформаційних функцій при збереженні загальної неглибокої архітектури для забезпечення швидкого вилучення функцій.

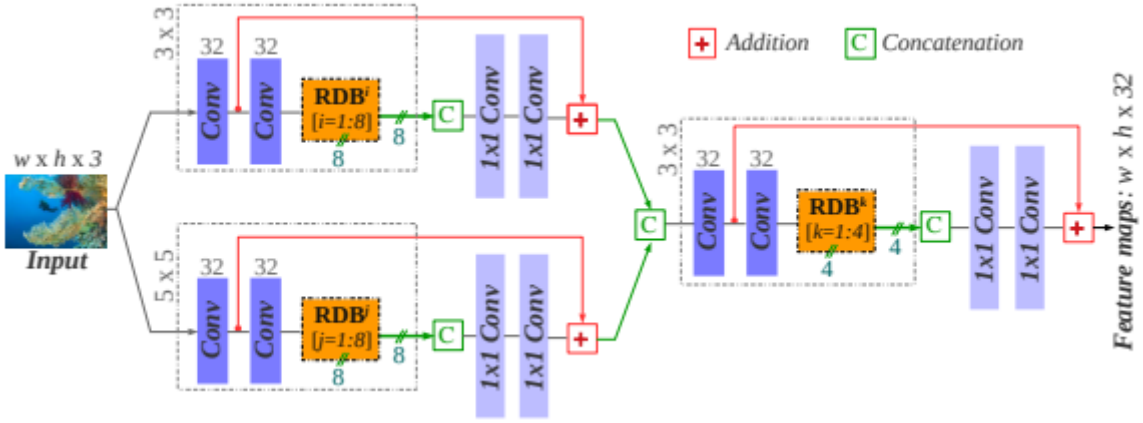


Рис. 3.3. Мережа вилучення функцій (FENet)

Допоміжна мережа уваги (AAN) вчиться моделювати зорову увагу в витягнутому просторі функцій FENet. Як показано на малюнку 3.4, два послідовних шари conv вчаться генерувати вихід одного каналу, що представляє видимість (ймовірність) для кожного пікселя. Ми бачимо прогнозовану карту вибору як значення зеленої інтенсивності, чорні пікселі представляють область фону.

Глибоке вивчення системи обробки зображень відбувається на первинній гілці через ряд шарів conv та deconv (деконволюційний).

Як показано на малюнку 3.4, розширене зображення (LR) та зображення генеруються окремими вихідними рівнями на різних етапах мережі. Покращене зображення генерується із шару conv, який безпосередньо слідує за FENet; це контролюється для вивчення вдосконалень за допомогою спеціальних функцій втрат, що застосовуються до неглибокого шару джерела.

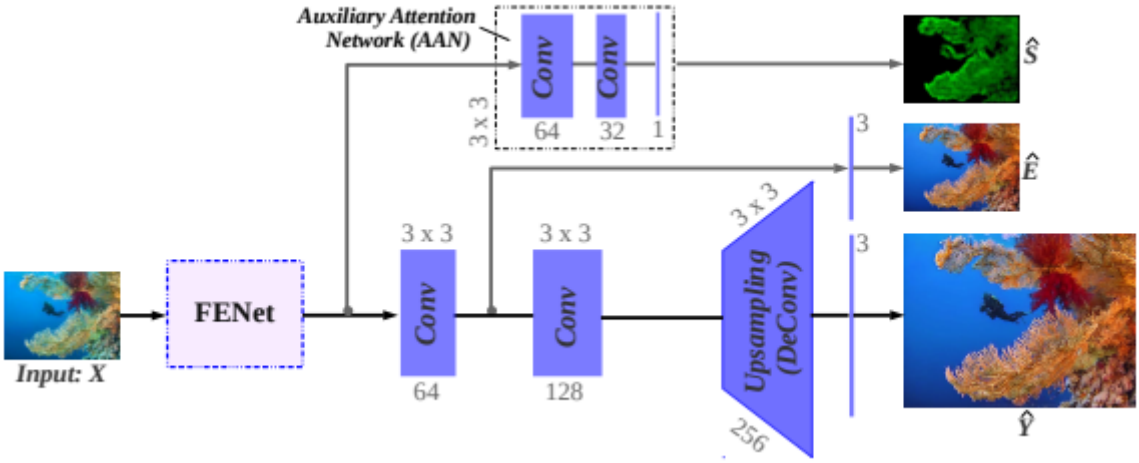


Рис. 3.4. архітектура «End-to-end»

Розширені функції також розповсюджуються на інший шар conv, за яким слідують шари deconv для підвищення вибірки.

Остаточний результат системи генерується на основі вибіркових функцій на основі заданого масштабу: 2 ×, 3 × або 4 ×.

3.4. Вибір ключових ознак зображення

Можна використовувати багато функцій для аналізу пар зображень на візуальну схожість. Але такі цифри вкрай небажані.

По-перше, час роботи результуючого алгоритму безпосередньо залежить від кількості функцій, таких як, в даному випадку, велика кількість значень, які потрібно обчислити.

По-друге, більша кількість функцій ускладнює архітектуру нейронної мережі, що призводить до вищих вимог до ресурсів, що беруть участь у процесі машинного навчання, і до часу самого навчання.

Тому необхідно зменшувати кількість ознак, але не наосліп, за рахунок якості результату, а з найбільшим зменшенням складності їх обчислення з найменшими втратами точності результату навчання. В якості фактора беремо абсолютну різницю між значеннями одного атрибута, розрахована для обох зображень пари.

Зменшуючи фактори, ми зменшуємо характеристики, зменшуючи складність розрахунків.

Факторний аналіз передбачає аналіз кореляційної матриці з побудовою графіка, який показує важливість кожного фактора і на основі якого приймається рішення про кількість та склад обраних (основних) факторів (у нашому випадку - ознак).

Слід зазначити, що з введенням цього етапу в метод, прагнення оптимізувати алгоритм доповнюється оптимізацією тимчасових характеристик.

Нехай, $s_{i.j}$ – значення *j*-ї ознаки, обчислене для *i*-ї пари зображень,

i = 1, .., I; j = 1, .., J;

I - кількість пар зображень;

J - кількість ознак;

t_j- середній час обчислення *j-ї* ознаки на зображенні;

a_i- ступінь близькості зображень в парі *i*.

З урахуванням використання двозначної шкали ознака приймає значення 1, якщо зображення пари є тематично близькими і 0 в іншому випадку. Покладемо *C(k,l)* - коефіцієнт кореляції між послідовностями чисел *k* і *l*.

Тоді можливо побудувати кореляційну матрицю:

$$\mathrm{Cs} = \left(\mathrm{cs}_{j1,j2}\right) = \left(C\left(S_{j1}, S_{j2}\right)\right)_{j1=1,j2=1}^{l,j}, \tag{3.1}$$

де S_{jx}- послідовність значень $S_{t,jx,}S_{2,jx}, \dots, S_{l,jx}$, і вектор:

$$Ca = \left(ca_j\right)\left(C\left(S_j, a\right)\right)_{j=1}^{l}, \tag{3.2}$$

Де a - послідовність значень$a_1, a_2, \dots, a_l$.

Тобто $cs_{k,l}$ - кореляція між *k*-м і *l*-м ознаками, а ca_j - кореляція між *j*-м ознакою і ступенем подібності зображень в парі.

Аналізуючи матрицю *Cs*, можна знайти ознаки, які мають сильну статистичну взаємозв'язок з іншими. Ці ознаки будемо називати надлишковими. Аналізуючи вектор *Ca*, можна знайти ознаки, які статистично ніяк не пов'язані зі ступенем близькості пари і, відповідно, не можуть застосовуватися для її оцінки. Ці ознаки будемо називати марними. Ознаки, що вимагають багато часу на їх обчислення t_j, будемо називати трудомісткими.

Щоб оцінити ступінь надмірності кожної ознаки, ми визначаємо вектор власних значень матриці Cs. У теорії аналізу даних ця процедура використовується у факторному аналізі, який є частиною багатовимірного статистичного аналізу та зменшує розмірність даних. Нехай X = (x_j) - вектор власних значень кореляційної матриці. Значення x_j визначає ступінь статистичної незалежності j-ї ознаки від інших. Пропонується використовувати вираз як таку міру:

$$h_j = \frac{x_j ca_j}{t_j}. \tag{3.3}$$

У цій формулі агрегуються три важливих показника, які необхідно враховувати при відборі ключових ознак: надмірність по відношенню до іншими ознаками, кореляція з результатом і час обчислення.

Якщо ранжирування значення h_j відкласти на графіку, отримуємо аналог графіка "Кам'яниста осип"(рис. 4.2.) з теорії факторного аналізу.

Типовою процедурою за критерієм Кайзера або візуально на графіку відсіваються не ключові ознаки, залишаючи ті, які будуть використовуватися в подальшій роботі.

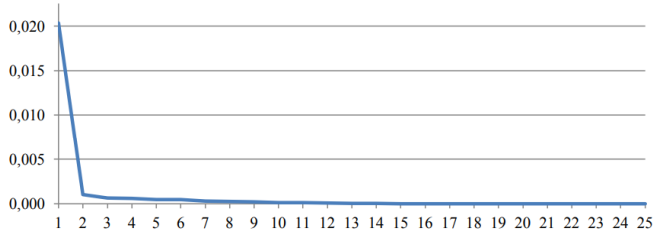


Рисунок 3.5. «Кам'яниста осип»

РОЗДІЛ 4
РЕАЛІЗАЦІЯ СИСТЕМИ

4.1. Інструментарій для розробки системи

При роботі над системою обробки зображень з використанням алгоритмів машинного навчання було вирішено використовувати мову програмування Python. Це пояснюється великою кількістю реалізованих алгоритмів класифікації та багатим набором методів встановлення нейронних мереж.

Завдяки популярності використання цього інструменту у світі глибокого навчання, існує багато готових до використання рішень, які значно прискорюють процес розробки системи.

Загалом, Python - це мова програмування високого рівня, спрямована на підвищення продуктивності розробника та читання коду. Синтаксис ядра Python мінімалістичний. Стандартна бібліотека містить значну кількість корисних функцій.

Python підтримує різноманітні парадигми програмування, включаючи структурну, функціональну, об’єктно-орієнтовану, імперативну та аспектно-орієнтовану. Основні особливості мови включають динамічне введення тексту, автоматичне звільнення пам'яті, корисні типи даних за замовчуванням, обробку винятків.

Код на Python зазвичай компілюється у функції та класи, які потім можна об'єднати в модулі, які, у свою чергу, за бажанням об'єднуються в пакети. Однією з переваг цієї мови програмування є читабельність програмного коду.

Ця функція є найважливішим для більшості професіоналів при виборі інструменту розробки. Легко читається, зрозуміле та вищої якості, що відрізняє Python від інших інструментів у світі мов сценаріїв. Код Python легше читати. Це тому, що синтаксис базується на відступах. У програмуванні вважається гарною нотою писати програмний код із використанням форматування відступів, це дозволяє краще, швидше та простіше оптимізувати його в програмі.

Python корисний, оскільки синтаксис побудований на цьому неписаному правилі програмістів, і він також не містить зайвих синтаксичних символів. Це означає, що повторне використання та обслуговування набагато простіше, ніж використання коду в інших мовах сценаріїв.

Автор Python Гідо ван Россум стверджував, що програмний код читається довше і частіше, ніж пишеться, тому не дивно, що принцип СУХОСТІ (Не повторюйся) є однією з парадигм цієї мови.

З вищесказаного випливає ще одна важлива перевага Python: висока швидкість розробки. Порівняно із компіляцією або строго набраними мовами, такими як C, C ++ та Java, Python багаторазово підвищує продуктивність розробника.

Загалом код Python часто становить лише одну третину або навіть п’яту частину еквівалентного коду C ++ або Java. Це означає менше введення з клавіатури, менше часу на налагодження та менше роботи з обслуговування та оптимізації продукту. Крім того, програми, створені на Python, працюють без компіляції, минаючи довгі етапи компіляції та прив'язки, необхідні для деяких інших мов програмування, додатково збільшуючи продуктивність програміста. Більшість програм Python працюють без змін на всіх основних платформах. Передача програмного коду з операційної системи Linux в Windows, як правило, є простою копією програмних файлів з однієї машини на іншу, просто встановіть один з 23 інтерпретаторів Python. Крім того, Python пропонує безліч можливостей для створення портативних графічних інтерфейсів, програм доступу до баз даних, веб-додатків та багатьох інших типів програм. Це робить його корисним інструментом для розробки міжплатформенних додатків.

Python постачається з великою кількістю зібраних та портативних функцій, відомих як стандартна бібліотека. Ця бібліотека забезпечує безліч функцій, необхідних для створення додатків, це дозволяє програмісту здійснювати пошук від простого тексту в шаблоні до мережевих функцій. Python дозволяє розширити ваші функціональні можливості як за допомогою ваших самостійно написаних бібліотек, так і за допомогою бібліотек, створених сторонніми розробниками.

Сторонні розробки включають засоби створення веб-сайтів, математичне програмування, розробку ігор тощо. Наприклад, розширення NumPy, яке часто використовують розробники, позиціонується як безкоштовний та потужніший еквівалент математичної системи програмування Mathlab.

Наприклад, класичний Python, як і багато інших інтерпретованих мов, які не використовують компілятори JIT, має загальний недолік: виконання програм на відносно повільній швидкості. Зберігаючи байт-код, інтерпретатору не потрібно витрачати зайвий час на перекомпіляцію коду модуля кожного разу, коли програма запускається. Можливість інтеграції компонентів дозволяє виконувати складні частини програми, які вимагають високої швидкості виконання, іншими мовами програмування. Це недолік, який компенсується іншими перевагами Python, зокрема:

• проста, універсальна та інтуїтивно зрозуміла мова програмування;

• витратив мало часу на написання коду;

• простий синтаксис;

• модульність (можна використовувати модуль коду в інших програмах);

• велика кількість різноманітних бібліотек та додатків, що спрощують обробку даних;

• доступне програмне забезпечення;

• незалежний від платформи (програма працює незалежно від операційної системи, в якій вона працювала).

Все вищесказане робить Python ідеальним інструментом для вирішення найрізноманітніших проблем. Швидка швидкість розробки є суттєвою перевагою при виборі мови програмування для прийняття бізнес-рішень, оскільки для цього не потрібно багато ресурсів. Це також стало можливим завдяки різноманітним готовим до використання програмним рішенням з відкритим кодом. Розробники з усього світу щодня оновлюють свої сховища унікальними алгоритмами Python.

Хорошим прикладом доречності використання цієї мови програмування є відомі компанії, які часто використовують Python, такі як Apple, NASA, Google та інші.

4.1.1. Фреймворк для створення нейронних мереж Keras

Keras - це високорівневий API нейронної мережі, написаний на Python, який може працювати на TensorFlow, CNTK або Theano. Він розроблений з акцентом на швидкі експерименти. Здатність переходити від ідеї до результату з якомога меншою затримкою є ключовим фактором хорошого дослідження.

Спочатку Keras був призначений для дослідників, щоб дозволити їм швидко проводити експерименти. Keras має наступні основні характеристики:

• дозволяє запускати той самий код на процесорі або графічному процесорі;

• має зручний API, що спрощує розробку прототипів моделей глибокого навчання;

• включає вбудовану підтримку мереж згортки (для розпізнавання образів), повторюваних мереж (для обробки послідовностей) та різних їх комбінацій;

• включає підтримку довільних архітектур мережі: моделі з декількома входами або виходами, частини шарів, частини моделей тощо.

Це означає, що Keras підходить для створення майже будь-якої моделі глибокого навчання. Фреймворк Keras поширюється за безкоштовною ліцензією і може бути використаний у комерційних проектах безкоштовно. Він сумісний з будь-якою версією Python від 2.7 до 3.8. Користувачів понад 200 000 керас - від академічних дослідників та інженерів до стартапів та великих корпорацій до аспірантів та любителів.

Keras використовується в 26 Google, Netflix, Uber, CERN, Yelp, Square та сотнях стартапів, що вирішують найрізноманітніші завдання. Keras - це також бібліотека модельного рівня, яка забезпечує будівельні блоки високого рівня для проектування моделей глибокого навчання. Він не реалізує операції низького рівня, такі як маніпуляції тензорами та диференціація, використовуючи спеціалізовану та оптимізовану бібліотеку підтримки тензорів. У цьому випадку Keras не покладається на одну бібліотеку підтримки тензорів, а використовує модульний підхід, тобто до фреймворку Keras можна підключити кілька бібліотек низького рівня. На даний момент підтримуються три такі бібліотеки: TensorFlow, Theano та

Microsoft Cognitive Toolkit (CNTK). Будь-який код, який використовує Keras, можна запустити з будь-якої з цих бібліотек, не змінюючи нічого в коді: ви можете швидко перемикатися між ними під час розробки, що часто є корисним, наприклад, якщо одна з бібліотек демонструє кращу ефективність при вирішенні заданого конкретне завдання.

4.1.2. Бібліотека для роботи з масивами NumPy

NumPy - це бібліотека для мови програмування Python, яка підтримує великі багатовимірні масиви та масиви. з великою колекцією математичних функцій високого рівня для роботи над цими масивами. Родоначальник NumPy, Numeric, спочатку був створений Джимом Гугуніним за співпраці кількох інших розробників. У 2005 році Тревіс Оліфант створив NumPy, включивши функції Numarray до Numarray із числовими змінами з основними змінами.

NumPy - це програмне забезпечення з відкритим кодом, у якому багато членів. NumPy має на меті реалізувати довідкову програму Python, яка є оптимізатором інтерфейсу байт-коду.

Математичні алгоритми, написані для цієї версії Python, часто працюють набагато повільніше, ніж складені еквіваленти. NumPy частково вирішує проблему повільності, надаючи багатовимірні масиви та функції та оператори, які ефективно працюють з масивами, вимагаючи від вас переписувати якийсь код, як правило, внутрішні цикли з NumPy.

Основною функціональністю NumPy є "ndarray" - структура даних для n-вимірного масиву. Ці масиви мають суворе представлення пам'яті. На відміну від вбудованої структури даних списків Python, цей масив повинен мати усі елементи одного типу. Такі масиви також можуть переглядати буфери пам'яті, виділені розширеннями C / C ++, Cython та Fortran для інтерпретатора CPython, не копіюючи дані, забезпечуючи сумісність із існуючими бібліотеками номерів. NumPy має вбудовану підтримку ndarrays, виділених для пам'яті.

4.1.3. Програмна бібліотека для налаштування машинного навчання TensorFlow

Бібліотека TensorFlow була вперше представлена у відкритому коді в листопаді 2015 року, Google вдалося залучити потенційних користувачів, і в наш час ця технологія стає все більш популярною. Бібліотека програмного забезпечення для машинного навчання - це нове покоління власних технологій DistBelief, розроблених командою Google Brain для різноманітних завдань, включаючи пошук зображень та вдосконалені алгоритми розпізнавання мови. TensorFlow - це нейронна мережа, яка вчиться вирішувати проблеми за допомогою позитивного підкріплення та обробляє дані на різних рівнях (вузлах), що допомагає знайти правильний результат.

Отримавши доступ до вихідного коду бібліотеки машинного навчання TensorFlow, Google спростив процес побудови та розгортання складних нейронних мереж. TensorFlow не дозволяє кожному розробнику скористатися перевагами машинного навчання, але надає API-інтерфейси Python та C / C ++, що дозволяють підключатися до програми розробника. Цей тип машинного навчання призначений виключно для дослідницьких цілей, але завдяки програмному забезпеченню з відкритим кодом, такому як TensorFlow, компанія отримує потужні інструменти для використання та обробки власних даних у недорогих хмарних умовах. Бібліотеки TensorFlow значно спрощують інтеграцію в програми елементів машинного навчання та функцій штучного інтелекту, призначених для розпізнавання мови, комп'ютерного зору або обробки природної мови. Звичайно, TensorFlow не єдина бібліотека для глибокого навчання, але, як і пошукова система Google, вона вважається найкращою в класі.

Альтернативи включають програмне забезпечення Torch, створене швейцарськими дослідниками, та Каліфорнійський університет у розробці Berkeley Caffe, остання версія якого, Caffe2, була розроблена разом з Facebook. Згідно з інформацією, опублікованою на веб-сайті TensorFlow, бібліотекою користується

низка великих компаній, включаючи Airbnb, Airbus, Dropbox, Snapchat та Uber (хоча, можливо, не найкращим чином).

4.2. Деталі реалізації системи

Глибоке навчання програми контролюється за допомогою парних даних форми ({X}, {S, E, Y}). Бібліотека TensorFlow використовується для реалізації конвеєра оптимізації.

Щоб визначити якість отриманих даних, спочатку ми аналізуємо глибокі зображення, що генеруються системою, якісно щодо кольору, контрасту та різкості.

Як показано на малюнку 4.1, покращені зображення розглядаються як відповідні істини.

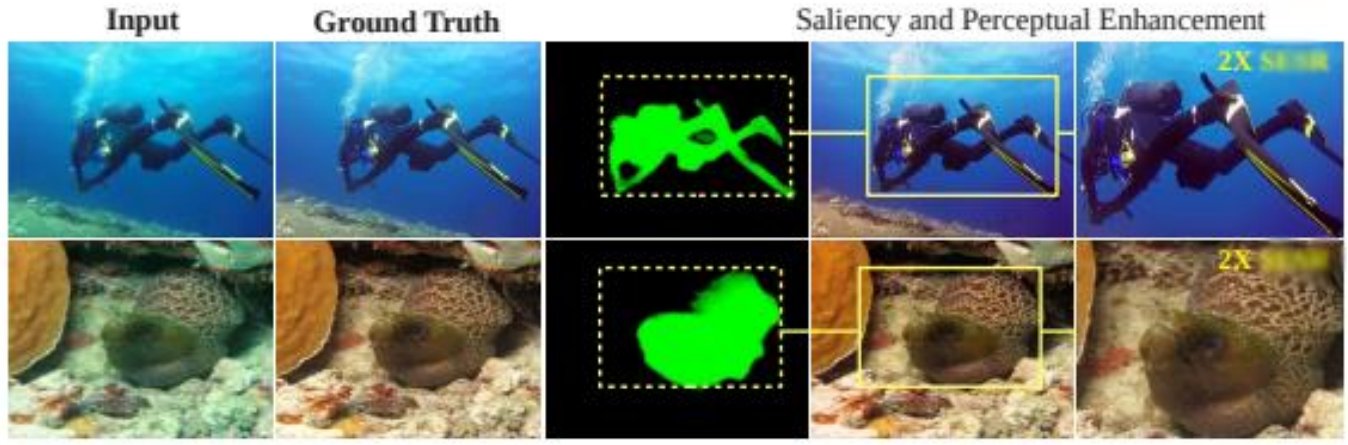


Рис. 4.1. Результат роботи системи

Зокрема, коригується зеленуватий підводний відтінок, часто відновлюються справжні піксельні кольори та відновлюється загальна чіткість зображення. Крім того, сформована карта розділення передбачає, що вона фокусується на правому передньому плані для покращення контрасту.

Далі демонструються внески кожного терміну збитків. Зміст для вивчення вдосконалення. Ми спостерігаємо, що передача кольору погіршується без L_P^{LR} та L_{Color}^{LR}, тоді як $L_{Content}^{LR}$ сприяє вивченню тонших деталей текстури.

Також помічаємо значно низькоконтрастне зображення без використання $L_{Contrast}^{LR}$, що підтверджує корисність оцінки контрастності за допомогою за допомогою CMI.

Далі порівнюємо ефективність перцептивного покращення зображення із наступними моделями:

1) відносне глобальне розтягнення гістограми (RGHS) [27],
2) неконтрольована корекція кольору (UCM) [28],
3) багатомасштабне злиття (MS-Fusion) [29],
4) багатомасштабний Retinex (MSRetinex), [30]
5) Water-Net [31],
6) UGAN [32],
7) Fusion-GAN [33],
8) FUnIE-GAN [34].

Перші чотири - це моделі, засновані на фізиці, а решта - моделі, засновані на навчанні; вони забезпечують продуктивність SOTA для покращення зображення в просторі RGB.

Їх ефективність визначається кількісно на основі загальних наборів тестів кожного набору даних на основі стандартної статистики:

• пікове відношення сигнал / шум (PSNR),

• вимірювання структурної подібності (SSIM)

• міра якості підводного зображення (UIQM).

PSNR та SSIM кількісно визначають якість реконструкції та структурну схожість сформованих зображень (відносно базової істини), тоді як UIQM оцінює якість зображень на основі кольору, яскравості та контрасту. Оцінка підсумована на рисунку 4.2, а також на рисунку 4.3. Наведено кілька якісних порівнянь.

Як показано на рис. 4.3, UCM та MS-Retinex часто страждають від перенасичення, тоді як RGBH, MS-Fusion та Water-Net не витримують декурляційних відтінків.

	Dataset	RGHS	UCM	MS-Fusion	MS-Retinex
PSNR	UFO-120	20.05 ± 3.1	20.99 ± 2.2	21.32 ± 3.3	21.69 ± 3.6
	EUVP	20.12 ± 2.9	20.55 ± 1.8	19.85 ± 2.4	21.27 ± 3.1
	UImNet	19.98 ± 1.8	20.48 ± 2.2	19.59 ± 3.2	22.63 ± 2.5
SSIM	UFO-120	0.75 ± 0.06	0.78 ± 0.07	0.79 ± 0.09	0.75 ± 0.10
	EUVP	0.69 ± 0.11	0.73 ± 0.14	0.70 ± 0.05	0.69 ± 0.15
	UImNet	0.61 ± 0.08	0.67 ± 0.06	0.64 ± 0.11	0.74 ± 0.04
UIQM	UFO-120	2.36 ± 0.33	2.41 ± 0.53	2.76 ± 0.45	2.69 ± 0.59
	EUVP	2.45 ± 0.46	2.48 ± 0.77	2.51 ± 0.36	2.48 ± 0.09
	UImNet	2.32 ± 0.48	2.38 ± 0.42	2.79 ± 0.55	2.84 ± 0.37

Water-Net	UGAN	Fusion-GAN	FUnIE-GAN	MY SYSTEM
22.46 ± 1.9	23.45 ± 3.1	24.07 ± 2.1	25.15 ± 2.3	27.15 ± 3.2
20.14 ± 2.3	23.67 ± 1.5	23.77 ± 2.4	26.78 ± 1.1	25.25 ± 2.1
21.02 ± 1.6	23.88 ± 2.1	23.12 ± 1.9	24.68 ± 2.4	25.52 ± 2.7
0.79 ± 0.05	0.80 ± 0.08	0.82 ± 0.07	0.82 ± 0.08	0.84 ± 0.03
0.68 ± 0.18	0.67 ± 0.11	0.68 ± 0.05	0.86 ± 0.05	0.75 ± 0.07
0.71 ± 0.07	0.79 ± 0.08	0.75 ± 0.07	0.77 ± 0.06	0.81 ± 0.05
2.83 ± 0.48	3.04 ± 0.28	2.98 ± 0.28	3.09 ± 0.51	3.13 ± 0.45
2.55 ± 0.06	2.70 ± 0.31	2.58 ± 0.07	2.95 ± 0.38	2.98 ± 0.28
2.92 ± 0.35	3.32 ± 0.55	3.19 ± 0.27	3.23 ± 0.32	3.26 ± 0.36

Рис.4.2. Підсумовані оцінки

Для порівняння, відновлення кольору та покращення контрасту UGAN, Fusion-GAN та FUnIE-GAN, як правило, кращі.

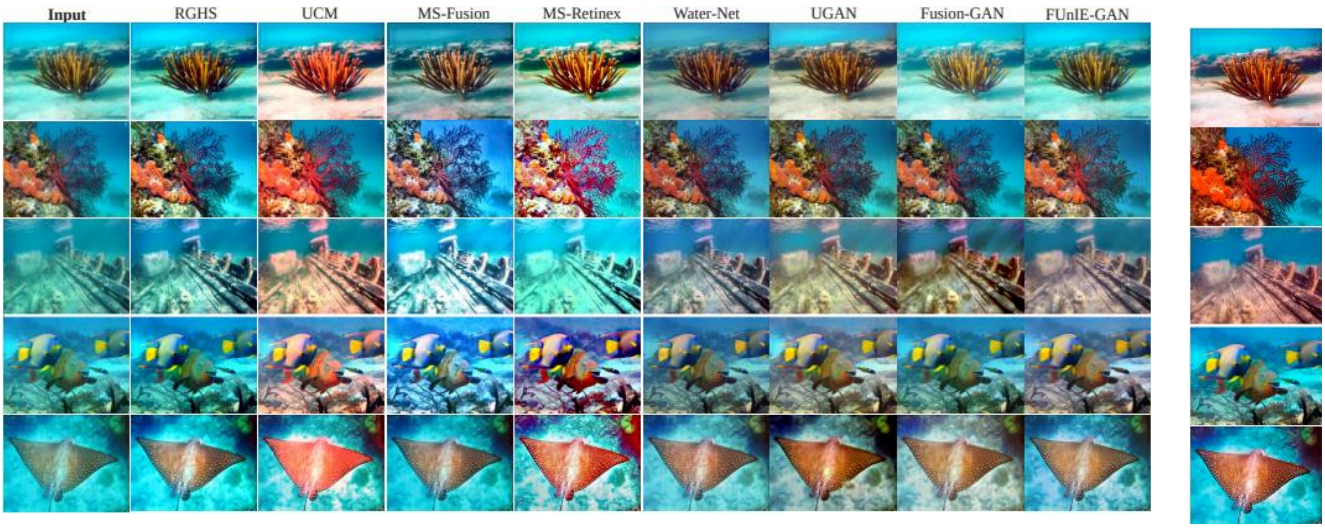


Рис. 4.3. Порівняння результату роботи системи

Система, досягає конкурентоспроможності та часто кращої продуктивності щодо PSNR та SSIM. Зокрема, вона, як правило, досягає кращих показників UIQM.

ВИСНОВКИ

В результаті дипломної роботи була успішно розроблена і побудована система обробки зображень із використанням штучного інтелекту. Проводиться поглиблена оцінка існуючих аналогів та розчинів. Проаналізовано сучасні практичні та теоретичні технології цифрової обробки зображень.

Необхідність та зміст усіх запропонованих функцій та алгоритмів були математично перевірені та доведені. Розроблена архітектура системи працювала бездоганно і зарекомендувала себе серед лідерів.

Після ретельного аналізу генеративна нейронна мережа була успішно створена та навчена. Проаналізувавши результати, ми можемо впевнено зробити висновок, що система досягає конкурентоспроможності і часто перевершує PSNR та SSIM. Зокрема, він прагне досягти кращих показників UIQM. DeepImProc особливо корисний у підводному домені завдяки своїм унікальним оптичним властивостям, таким як загасання, заломлення та зворотне розсіювання. Ці артефакти викликають нелінійні спотворення залежно від діапазону та довжини хвилі, що суттєво впливає на зір, незважаючи на те, що часто використовуються високоякісні камери.

Розроблена система не тільки добре виконала поставлене завдання, але й довела, що нейронна мережа не обов'язково займає багато місця на диску, споживає багато ресурсів і вчиться довго і болісно.

Слід зазначити, що під час планування та розробки системи було вирішено розробити програмний продукт, з можливістю подальшого вдосконалення та розширення функціональних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

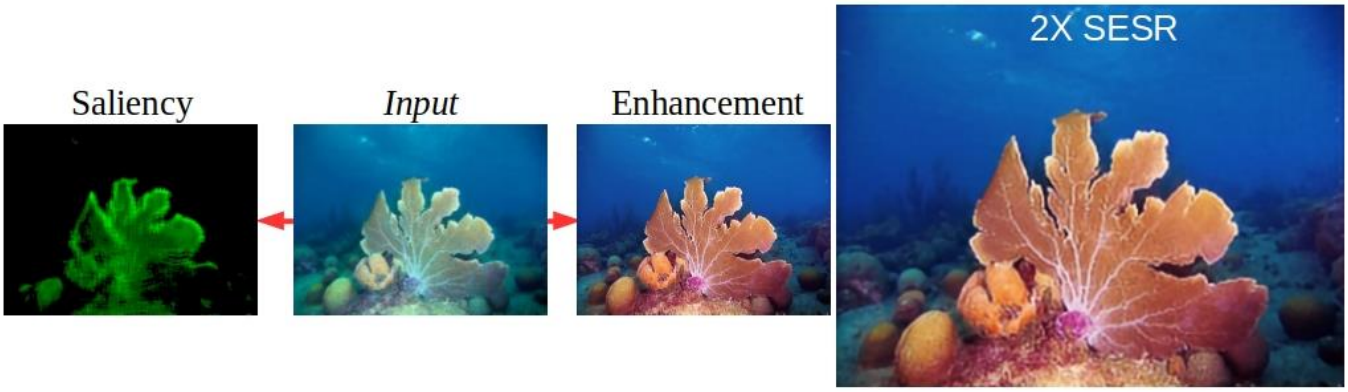
1. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – М.:Вильямс, 2006. – 1104 с.
2. Jaeyong Chung and Taehwan Shin. Simplifying deep neural networks for neuromorphic architectures. *In Design Automation Conference (DAC)*, 2016 53nd ACM/EDAC/IEEE, pages 1–6.IEEE, 2016
3. Ahn, N., Kang, B., Sohn, K.A.: Fast, accurate, and lightweight super-resolution with cascading residual network. In: ECCV (2018)
4. Cai, J., Gu, S., Timofte, R., Zhang, L.: Ntire 2019 challenge on real image superresolution: Methods and results. In: CVPRW (2019)
5. Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: ECCV (2014)
6. Freedman, G., Fattal, R.: Image and video upscaling from local self-examples. TOG (2011)
7. Barni M., Bartolini F., Cappellini V. Image processing for virtual restoration of artworks. *IEEE Multimedia*, vol. 7, no. 2, pp. 34-37, 2000
8. Kaggle - [Електронний ресурс] система організації конкурсів з дослідження даних, а також соціальна мережа фахівців з обробки даних та машинного навчання. Спосіб доступу - https://www.kaggle.com/
9. D. Berman, D. Levy, S. Avidan, and T. Treibitz. Underwater Single Image Color Restoration using Haze-Lines and a New Quantitative Dataset. *arXiv preprint arXiv:1811.01343, 2018*
10. B. Cai, X. Xu, K. Jia, C. Qing, and D. Tao. DehazeNet: An End-to-end System for Single Image Haze Removal. IEEE Transactions on Image Processing, 25(11):5187–5198, 2016
11. C. Dong, C. C. Loy, K. He, and X. Tang. Image Superresolution using Deep Convolutional Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(2):295–307, 2015

12. A. Hore and D. Ziou. Image Quality Metrics: PSNR vs. SSIM. In International Conference on Pattern Recognition, pages 2366–2369. IEEE, 2010.
13. A. Ignatov, N. Kobyshev, R. Timofte, K. Vanhoey, and L. Van Gool. DSLR-quality Photos on Mobile Devices with Deep Convolutional Networks. *In IEEE International Conference on Computer Vision (ICCV)*, pages 3277–3285, 2017.
14. C. Feichtenhofer, H. Fassold, and P. Schallauer. A Perceptual Image Sharpness Metric based on Local Edge Gradient Analysis. IEEE Signal Processing Letters, 20(4):379–382, 2013.
15. J. Kim, J. Kwon Lee, and K. Mu Lee. Accurate Image Super-resolution using Very Deep Convolutional Networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1646–1654, 2016.
16. C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, ´ A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic Single Image Super-resolution using a Generative Adversarial Network. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4681– 4690, 2017.
17. K. Nasrollahi and T. B. Moeslund. Super-resolution: A comprehensive survey. In Machine Vision and Applications, volume 25, pages 1423–1468. 2014
18. A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In International Conference on Learning Representations (ICLR), 2016.
19. W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep laplacian pyramid networks for fast and accurate superresolution. In Proc. IEEE Conf. Comput. Vis. Pattern Recognit., pages 624–632, 2017
20. C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. CoRR, abs/1609.04802, 2016
21. B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, July 2017

22. R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, et al. Ntire 2017 challenge on single image superresolution: Methods and results. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, July 2017.
23. R. Timofte, R. Rothe, and L. Van Gool. Seven ways to improve example-based single image super resolution. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016
24. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014
25. Y. Bei, A. Damian, S. Hu, S. Menon, N. Ravi, and C. Rudin. New techniques for preserving global structure and denoising with low information loss in single-image superresolution. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2018.
26. Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In Proceedings of the 26th annual international conference on machine learning, pages 41–48. ACM, 2009.
27. D. Huang, Y. Wang, W. Song, J. Sequeira, and S. Mavromatis. Shallow-water Image Enhancement using Relative Global Histogram Stretching Based on Adaptive Parameter Acquisition. In International Conference on Multimedia Modeling, pages 453–465. Springer, 2018
28. K. Iqbal, M. Odetayo, A. James, R. A. Salam, and A. Z. H. Talib. Enhancing the Low Quality Images using Unsupervised Colour Correction Method. In IEEE International Conference on Systems, Man and Cybernetics, pages 1703– 1709. IEEE, 2010
29. C. Ancuti, C. O. Ancuti, T. Haber, and P. Bekaert. Enhancing Underwater Images and Videos by Fusion. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 81–88, 2012.
30. S. Zhang, T. Wang, J. Dong, and H. Yu. Underwater Image Enhancement via Extended Multi-scale Retinex. Neurocomputing, 245:1–9, 2017.

31. C. Li, C. Guo, W. Ren, R. Cong, J. Hou, S. Kwong, and D. Tao. An Underwater Image Enhancement Benchmark Dataset and Beyond. In IEEE Transactions on Image Processing (TIP), pages 1–1. IEEE, 2019.
32. C. Fabbri, M. J. Islam, and J. Sattar. Enhancing Underwater Imagery using Generative Adversarial Networks. In IEEE International Conference on Robotics and Automation (ICRA), pages 7159–7165. IEEE, 2018.
33. H. Li, J. Li, and W. Wang. A Fusion Adversarial Underwater Image Enhancement Network with a Public Test Dataset. *arXiv preprint arXiv:1906.06819*, 2019
34. M. J. Islam, Y. Xia, and J. Sattar. Fast Underwater Image Enhancement for Improved Visual Perception. *arXiv preprint arXiv:1903.09766,* 2019.

ДОДАТОК 1. Результати виконання програми.



ДОДАТОК 2. Лістинг коду.

data_utils.py

```
#!/usr/bin/en
v python
                """
                # > Various modules for handling data
                """
                from __future__ import division
                from __future__ import absolute_import
                import os
                import random
                import fnmatch
                import numpy as np
                from scipy import misc
                from glob import glob
                import skimage.transform


                def deprocess(x):
                    # [-1,1] -> [0, 1]
                    return (x+1.0)*0.5


                def deprocess_uint8(x):
                    # [-1,1] -> np.uint8 [0, 255]
                    im = ((x+1.0)*127.5)
                    return np.uint8(im)
```

```
def preprocess(x):
    # [0,255] -> [-1, 1]
    return (x/127.5)-1.0


def getPaths(data_dir):
    exts = ['*.png','*.PNG','*.jpg','*.JPG', '*.JPEG']
    image_paths = []
    for pattern in exts:
        for d, s, fList in os.walk(data_dir):
            for filename in fList:
                if (fnmatch.fnmatch(filename, pattern)):
                    fname_ = os.path.join(d,filename)
                    image_paths.append(fname_)
    return np.asarray(image_paths)


def read_and_resize(paths, res=(480, 640), mode_='RGB'):
    img = imageio.imread(paths, pilmode=mode_).astype(np.float)
    img = skimage.transform.resize(img, res)
    return img


def preprocess_mask(x):
    # attention mask [0,255] -> [0, 1]
    x = x*1.0 / 255.0
    x[x<0.2] = 0
```

```
    return np.expand_dims(x, axis=3)


def deprocess_mask(x):
    # [0,1] -> np.uint8 [0, 255]
    x *= 255.
    x[x<0.1] = 0
    return np.uint8(x)


def normalize_mask(img):
    # img [0, 1] => normaized [0, 255]
    img *= 255.
    M, m = np.max(img), np.min(img)
    return (img-m)/(M-m)



def get_cmi(imgs, masks):
    cmis = []
    for i in range(imgs.shape[0]):
        mean = np.sum(imgs[i,:], axis=-1)
        mask = np.reshape(masks[i,:], mean.shape)
        F = mean*mask
        B = mean*(1-mask)
        cmis.append(F-B)
    return np.array(cmis)
```

```
class dataLoaderUFO():
    def __init__(self, data_path, SCALE=2):
        # SCALE = 2 (320, 240)  => (640,480)
        # SCALE = 3 (214, 160)  => (640,480)
        # SCALE = 4 (160, 120)  => (640,480)
        self.SCALE = SCALE
        if (self.SCALE==2): self.lr_res_ = (240, 320)
        elif (self.SCALE==3): self.lr_res_ = (160, 214)
        else: self.lr_res_ = (120, 160)
        self.hr_folder = "hr/"
        self.mask_folder = "mask/"
        self.lr_dist_folder = "lrd/"
        self.get_train_and_val_paths(data_path)



    def get_train_and_val_paths(self, data_path):
        self.num_train, self.num_val = 0, 0
        self.train_lrd_paths, self.val_lrd_paths= [], []
        self.train_hr_paths, self.val_hr_paths  = [], []
        self.train_mask_paths, self.val_mask_paths =  [], []


        data_dir = os.path.join(data_path, "train_val/")
        lrd_path = sorted(os.listdir(data_dir+self.lr_dist_folder))
        hr_path = sorted(os.listdir(data_dir+self.hr_folder))
```

```
        mask_path = sorted(os.listdir(data_dir+self.mask_folder))
        num_paths = min(len(lrd_path), len(hr_path), len(mask_path))
        all_idx = range(num_paths)
        # 95% train-val splits
        random.shuffle(all_idx)
        self.num_train = int(num_paths*0.95)
        self.num_val = num_paths-self.num_train
        train_idx = set(all_idx[:self.num_train])
        # split data paths to training and validation sets
        for i in range(num_paths):
            if i in train_idx:

self.train_lrd_paths.append(data_dir+self.lr_dist_folder+lrd_path[i])

self.train_hr_paths.append(data_dir+self.hr_folder+hr_path[i])

self.train_mask_paths.append(data_dir+self.mask_folder+mask_path
[i])
            else:

self.val_lrd_paths.append(data_dir+self.lr_dist_folder+lrd_path[i])

self.val_hr_paths.append(data_dir+self.hr_folder+hr_path[i])

self.val_mask_paths.append(data_dir+self.mask_folder+mask_path[i
])
        print ("Loaded {0} samples for
training".format(self.num_train))
```

```
        print ("Loaded {0} samples for
validation".format(self.num_val))



    def load_batch(self, batch_size=1):
        self.n_batches = self.num_train//batch_size
        for i in range(self.n_batches-1):
            batch_lrd =
self.train_lrd_paths[i*batch_size:(i+1)*batch_size]
            batch_hr = self.train_hr_paths[i*batch_size:(i+1)*batch_size]
            batch_m =
self.train_mask_paths[i*batch_size:(i+1)*batch_size]


            imgs_lrd, imgs_lr, imgs_hr, imgs_mask = [], [], [], []
            for idx in range(len(batch_lrd)):
                img_lrd = read_and_resize(batch_lrd[idx], res=self.lr_res_)
                img_lr = read_and_resize(batch_hr[idx], res=self.lr_res_)
                img_hr = read_and_resize(batch_hr[idx], res=(480, 640))
                img_mask = read_and_resize(batch_m[idx],
res=self.lr_res_, mode_='L')
                imgs_lrd.append(img_lrd)
                imgs_lr.append(img_lr)
                imgs_hr.append(img_hr)
                imgs_mask.append(img_mask)
            imgs_lrd = preprocess(np.array(imgs_lrd))
            imgs_lr = preprocess(np.array(imgs_lr))
```

```
            imgs_hr = preprocess(np.array(imgs_hr))
            imgs_mask = preprocess_mask(np.array(imgs_mask))
            cmis = get_cmi(imgs_lr,imgs_mask)
            yield imgs_lrd, imgs_lr, imgs_hr, imgs_mask, cmis




    def load_val_data(self, batch_size=1):
        idx = np.random.choice(np.arange(self.num_val), batch_size,
replace=False)
        paths_lrd = [self.val_lrd_paths[i] for i in idx]
        paths_hr = [self.val_hr_paths[i] for i in idx]
        paths_mask = [self.val_mask_paths[i] for i in idx]
        imgs_lrd, imgs_lr, imgs_hr, imgs_mask = [], [], [], []
        for idx in range(len(paths_hr)):
            img_lrd = read_and_resize(paths_lrd[idx], res=self.lr_res_)
            img_lr = read_and_resize(paths_hr[idx], res=self.lr_res_)
            img_hr = read_and_resize(paths_hr[idx], res=(480, 640))
            img_mask = read_and_resize(paths_mask[idx],
res=self.lr_res_, mode_='L')
            imgs_lrd.append(img_lrd)
            imgs_lr.append(img_lr)
            imgs_hr.append(img_hr)
            imgs_mask.append(img_mask)
        imgs_lrd = preprocess(np.array(imgs_lrd))
        imgs_lr = preprocess(np.array(imgs_lr))
        imgs_hr = preprocess(np.array(imgs_hr))
        imgs_mask = preprocess_mask(np.array(imgs_mask))
```

```
        cmis = get_cmi(imgs_lr,imgs_mask)
        return imgs_lrd, imgs_lr, imgs_hr, imgs_mask, cmis
```

/utils/ssim_psnr_utils.py

```
from
__future__
import
division
        import numpy as np
        import math
        from scipy.ndimage import gaussian_filter


        def getSSIM(X, Y):
            #Computes the mean structural similarity between two images.
            assert (X.shape == Y.shape), "Image-patche provided have different
        dimensions"
            nch = 1 if X.ndim==2 else X.shape[-1]
            mssim = []
            for ch in xrange(nch):
                Xc, Yc = X[...,ch].astype(np.float64), Y[...,ch].astype(np.float64)
                mssim.append(compute_ssim(Xc, Yc))
            return np.mean(mssim)




        def compute_ssim(X, Y):
            """Compute the SSIM per single channel (given two images)
```

```
"""
# variables are initialized as suggested in the paper
K1 = 0.01
K2 = 0.03
sigma = 1.5
win_size = 5
# means
ux = gaussian_filter(X, sigma)
uy = gaussian_filter(Y, sigma)
# variances and covariances
uxx = gaussian_filter(X * X, sigma)
uyy = gaussian_filter(Y * Y, sigma)
uxy = gaussian_filter(X * Y, sigma)
# normalize by unbiased estimate of std dev
N = win_size ** X.ndim
unbiased_norm = N / (N - 1)  # eq. 4 of the paper
vx  = (uxx - ux * ux) * unbiased_norm
vy  = (uyy - uy * uy) * unbiased_norm
vxy = (uxy - ux * uy) * unbiased_norm
R = 255
C1 = (K1 * R) ** 2
C2 = (K2 * R) ** 2
# compute SSIM (eq. 13 of the paper)
sim = (2 * ux * uy + C1) * (2 * vxy + C2)
D = (ux ** 2 + uy ** 2 + C1) * (vx + vy + C2)
SSIM = sim/D
mssim = SSIM.mean()
return mssim
```

```
def getPSNR(X, Y):
    #assume RGB image
    target_data = np.array(X, dtype=np.float64)
    ref_data = np.array(Y, dtype=np.float64)
    diff = ref_data - target_data
    diff = diff.flatten('C')
    rmse = math.sqrt(np.mean(diff ** 2.) )
    if rmse == 0: return 100
    else: return 20*math.log10(255.0/rmse)
```

/utils/uiqm_utils.py

```
Computes
the
Underwater
Image
Quality
Measure
(UIQM)
        """
        from scipy import ndimage
        from PIL import Image
        import numpy as np
        import math
```

```
def mu_a(x, alpha_L=0.1, alpha_R=0.1):
    """
      Calculates the asymetric alpha-trimmed mean
    """
    # sort pixels by intensity - for clipping
    x = sorted(x)
    # get number of pixels
    K = len(x)
    # calculate T alpha L and T alpha R
    T_a_L = math.ceil(alpha_L*K)
    T_a_R = math.floor(alpha_R*K)
    # calculate mu_alpha weight
    weight = (1/(K-T_a_L-T_a_R))
    # loop through flattened image starting at T_a_L+1 and ending at K-
T_a_R
    s   = int(T_a_L+1)
    e   = int(K-T_a_R)
    val = sum(x[s:e])
    val = weight*val
    return val




def s_a(x, mu):
    val = 0
```

```
    for pixel in x:
        val += math.pow((pixel-mu), 2)
    return val/len(x)



def _uicm(x):
    R = x[:,:,0].flatten()
    G = x[:,:,1].flatten()
    B = x[:,:,2].flatten()
    RG = R-G
    YB = ((R+G)/2)-B
    mu_a_RG = mu_a(RG)
    mu_a_YB = mu_a(YB)
    s_a_RG = s_a(RG, mu_a_RG)
    s_a_YB = s_a(YB, mu_a_YB)
    l = math.sqrt( (math.pow(mu_a_RG,2)+math.pow(mu_a_YB,2)) )
    r = math.sqrt(s_a_RG+s_a_YB)
    return (-0.0268*l)+(0.1586*r)



def sobel(x):
    dx = ndimage.sobel(x,0)
    dy = ndimage.sobel(x,1)
    mag = np.hypot(dx, dy)
    mag *= 255.0 / np.max(mag)
```

```
    return mag



def eme(x, window_size):
    """
      Enhancement measure estimation
      x.shape[0] = height
      x.shape[1] = width
    """
    # if 4 blocks, then 2x2...etc.
    k1 = x.shape[1]/window_size
    k2 = x.shape[0]/window_size


    # weight
    w = 2./(k1*k2)


    blocksize_x = window_size
    blocksize_y = window_size


    # make sure image is divisible by window_size - doesn't matter if we cut out some pixels
    x = x[:blocksize_y*k2, :blocksize_x*k1]
```

```
    val = 0
    for l in range(k1):
        for k in range(k2):
            block = x[k*window_size:window_size*(k+1),
l*window_size:window_size*(l+1)]
            max_ = np.max(block)
            min_ = np.min(block)



            # bound checks, can't do log(0)
            if min_ == 0.0: val += 0
            elif max_ == 0.0: val += 0
            else: val += math.log(max_/min_)
    return w*val




def _uism(x):
    """
      Underwater Image Sharpness Measure
    """
    # get image channels
    R = x[:,:,0]
    G = x[:,:,1]
    B = x[:,:,2]



    # first apply Sobel edge detector to each RGB component
```

```
    Rs = sobel(R)
    Gs = sobel(G)
    Bs = sobel(B)


    # multiply the edges detected for each channel by the channel itself
    R_edge_map = np.multiply(Rs, R)
    G_edge_map = np.multiply(Gs, G)
    B_edge_map = np.multiply(Bs, B)


    # get eme for each channel
    r_eme = eme(R_edge_map, 10)
    g_eme = eme(G_edge_map, 10)
    b_eme = eme(B_edge_map, 10)


    # coefficients
    lambda_r = 0.299
    lambda_g = 0.587
    lambda_b = 0.144


    return (lambda_r*r_eme) + (lambda_g*g_eme) +
(lambda_b*b_eme)
```

```
def plip_g(x,mu=1026.0):
    return mu-x



def plip_theta(g1, g2, k):
    g1 = plip_g(g1)
    g2 = plip_g(g2)
    return k*((g1-g2)/(k-g2))



def plip_cross(g1, g2, gamma):
    g1 = plip_g(g1)
    g2 = plip_g(g2)
    return g1+g2-((g1*g2)/(gamma))



def plip_diag(c, g, gamma):
    g = plip_g(g)
    return gamma - (gamma * math.pow((1 - (g/gamma) ), c) )
```

```
def plip_multiplication(g1, g2):
    return plip_phiInverse(plip_phi(g1) * plip_phi(g2))
    #return plip_phiInverse(plip_phi(plip_g(g1)) * plip_phi(plip_g(g2)))



def plip_phiInverse(g):
    plip_lambda = 1026.0
    plip_beta   = 1.0
    return plip_lambda * (1 - math.pow(math.exp(-g / plip_lambda), 1 /
plip_beta));



def plip_phi(g):
    plip_lambda = 1026.0
    plip_beta   = 1.0
    return -plip_lambda * math.pow(math.log(1 - g / plip_lambda),
plip_beta)



def _uiconm(x, window_size):
    """
      Underwater image contrast measure
      https://github.com/tkrahn108/UIQM/blob/master/src/uiconm.cpp
```

```
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5609219
    """


    plip_lambda = 1026.0
    plip_gamma  = 1026.0
    plip_beta   = 1.0
    plip_mu     = 1026.0
    plip_k      = 1026.0


    # if 4 blocks, then 2x2...etc.
    k1 = x.shape[1]/window_size
    k2 = x.shape[0]/window_size


    # weight
    w = -1./(k1*k2)


    blocksize_x = window_size
    blocksize_y = window_size


    # make sure image is divisible by window_size - doesn't matter if
we cut out some pixels
    x = x[:blocksize_y*k2, :blocksize_x*k1]
```

```
    # entropy scale - higher helps with randomness
    alpha = 1


    val = 0
    for l in range(k1):
        for k in range(k2):
            block = x[k*window_size:window_size*(k+1),
l*window_size:window_size*(l+1), :]
            max_ = np.max(block)
            min_ = np.min(block)


            top = max_-min_
            bot = max_+min_


            if math.isnan(top) or math.isnan(bot) or bot == 0.0 or top ==
0.0: val += 0.0
            else: val += alpha*math.pow((top/bot),alpha) *
math.log(top/bot)


            #try: val += plip_multiplication((top/bot),math.log(top/bot))
    return w*val
```

```
def getUIQM(x):
    """
      Function to return UIQM to be called from other programs
      x: image
    """
    x = x.astype(np.float32)
    ### from
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7300447
    #c1 = 0.4680; c2 = 0.2745; c3 = 0.2576
    ### from
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7300447
    c1 = 0.0282; c2 = 0.2953; c3 = 3.5753


    uicm   = _uicm(x)
    uism   = _uism(x)
    uiconm = _uiconm(x, 10)
    uiqm = (c1*uicm) + (c2*uism) + (c3*uiconm)
    return uiqm
```

measure_calc.py

```
## python libs
import os
import numpy as np
from glob import glob
from os.path import join
from ntpath import basename
from PIL import Image
```

```
## local libs
from utils.uiqm_utils import getUIQM
from utils.ssm_psnr_utils import getSSIM, getPSNR




# measurement in a common dimension
im_w, im_h = 320, 240


## data paths
REAL_im_dir = "data/sample_test_ufo/lrd/"  # real/input im-dir with {f.ext}
                GEN_im_dir  = "data/output/keras_out/"  # generated im-dir with
                                                                {f_SESR/EN.ext}
  GTr_im_dir  = "data/sample_test_ufo/hr/"  # ground truth im-dir with {f.ext}


## mesures uqim for all images in a directory
                                    def measure_UIQMs(dir_name, file_ext=None):
    """
      # measured in RGB
      Assumes:
        * dir_name contain generated images
                                            * to evaluate on all images: file_ext = None
        * to evaluate images that ends with "_SESR.png" or "_En.png"
            * use file_ext = "_SESR.png" or "_En.png"
    """
    paths = sorted(glob(join(dir_name, "*.*")))
```

```
    if file_ext:
        paths = [p for p in paths if p.endswith(file_ext)]
    uqims = []
    for img_path in paths:
        im = Image.open(img_path).resize((im_w, im_h))
        uqims.append(getUIQM(np.array(im)))
    return np.array(uqims)




def measure_SSIM(gtr_dir, gen_dir):
    """
     # measured in RGB
     Assumes:
       * gtr_dir contain ground-truths {filename.ext}
                        * gen_dir contain generated images {filename_SESR.png}
                                                                                """
    gtr_paths = sorted(glob(join(gtr_dir, "*.*")))
    gen_paths = sorted(glob(join(gen_dir, "*.*")))
    ssims = []
    for gtr_path in gtr_paths:
        fname = basename(gtr_path).split('.')[0]
        gen_path = join(gen_dir, fname + '_SESR.png') # for SESR
        #gen_path = join(gen_dir, fname + '_En.png') # enhancement
        if gen_path in gen_paths:
            r_im = Image.open(gtr_path).resize((im_w, im_h))
            g_im = Image.open(gen_path).resize((im_w, im_h))
            # get ssim on RGB channels (SOTA norm)
```

```
            ssim = getSSIM(np.array(r_im), np.array(g_im))
            ssims.append(ssim)
    return np.array(ssims)




def measure_PSNR(gtr_dir, gen_dir):
    """
      # measured in lightness channel
      Assumes:
        * gtr_dir contain ground-truths {filename.ext}
                        * gen_dir contain generated images {filename_SESR.png}
                                                                    """
    gtr_paths = sorted(glob(join(gtr_dir, "*.*")))
    gen_paths = sorted(glob(join(gen_dir, "*.*")))
    psnrs = []
    for gtr_path in gtr_paths:
        fname = basename(gtr_path).split('.')[0]
        gen_path = join(gen_dir, fname + '_SESR.png') # for SESR
        #gen_path = join(gen_dir, fname + '_En.png') # enhancement
        if gen_path in gen_paths:
            r_im = Image.open(gtr_path).resize((im_w, im_h))
            g_im = Image.open(gen_path).resize((im_w, im_h))
            # get psnt on L channel (SOTA norm)
            r_im = r_im.convert("L"); g_im = g_im.convert("L")
            psnr = getPSNR(np.array(r_im), np.array(g_im))
            psnrs.append(psnr)
    return np.array(psnrs)
```

```
### compute SSIM and PSNR
SSIM_measures = measure_SSIM(GTr_im_dir, GEN_im_dir)
PSNR_measures = measure_PSNR(GTr_im_dir, GEN_im_dir)
print ("SSIM >> Mean: {0} std: {1}".format(np.mean(SSIM_measures),
np.std(SSIM_measures)))
print ("PSNR >> Mean: {0} std: {1}".format(np.mean(PSNR_measures),
np.std(PSNR_measures)))



                                        ### compute and compare UIQMs
gen_uqims = measure_UIQMs(GEN_im_dir, file_ext="_En.png")  # or
file_ext="_SESR.png"
print ("Generated UQIM >> Mean: {0} std: {1}".format(np.mean(gen_uqims),
np.std(gen_uqims)))
```

Test_ImProc_TF.py

```
Import os
import time
```

```
import numpy as np
from glob import glob
from ntpath import basename
from os.path import join, exists
from PIL import Image
import tensorflow as tf
## local libs
from utils.data_utils import getPaths
from utils.data_utils import preprocess, deprocess
from utils.data_utils import deprocess_uint8, deprocess_mask




class Deep_SESR_GraphOP:
    def __init__(self, frozen_model_path):
        # loads the graph
        self.gen_graph = tf.Graph()
        with self.gen_graph.as_default():
            od_graph_def = tf.GraphDef()
            with tf.gfile.GFile(frozen_model_path, 'rb') as fid:
                serialized_graph = fid.read()
                od_graph_def.ParseFromString(serialized_graph)
                tf.import_graph_def(od_graph_def, name='')
        # graphOP handler for Deep SESR
        ops = self.gen_graph.get_operations()
        all_tensor_names = {output.name for op in ops for output in op.outputs}
```

```
        self.output_dict = {}
        for key in ['conv2d_42/Tanh', 'conv2d_45/Tanh', 'conv2d_48/Sigmoid']:
            tensor_name = key + ':0'
            if tensor_name in all_tensor_names:
                self.output_dict[key] =
self.gen_graph.get_tensor_by_name(tensor_name)
        self.inp_im_tensor = self.gen_graph.get_tensor_by_name('input_3:0')
        self.sess = tf.Session(graph=self.gen_graph)




    def predict(self, frame):
        """
          Given an input frame, returns:
           - en_im: enhanced image (same size)
           - sesr_im: enhanced and super-resolution image (size * scale)
           - mask: saliency mask (same size)
        """
        output_dict = self.sess.run(self.output_dict,
                                    feed_dict={self.inp_im_tensor: np.expand_dims(frame,
0)})
        en_im = output_dict['conv2d_42/Tanh'][0]
        sesr_im = output_dict['conv2d_45/Tanh'][0]
        mask = output_dict['conv2d_48/Sigmoid'][0]
        return en_im, sesr_im, mask
```

```
## load specific model
ckpt_name =  "deep_sesr_2x_1d"
frozen_model_path = os.path.join("models/", ckpt_name+".pb")
assert (os.path.exists(frozen_model_path))
generator = Deep_SESR_GraphOP(frozen_model_path)


# input and output data shape
scale = 2
hr_w, hr_h = 640, 480 # HR
lr_w, lr_h = 320, 240 # LR (1/2x)
lr_res, lr_shape = (lr_w, lr_h), (lr_h, lr_w, 3)
hr_res, hr_shape = (hr_w, hr_h), (hr_h, hr_w, 3)


## for testing arbitrary local data
data_dir = "data/sample_test_ufo/lrd/"
#data_dir = "data/test_mixed/"
test_paths = getPaths(data_dir)
print ("{0} test images are loaded".format(len(test_paths)))


## create dir for output test data
samples_dir = os.path.join("data/output/", "tf_out")
if not os.path.exists(samples_dir): os.makedirs(samples_dir)


# testing loop
```

```
times = [];
for img_path in test_paths:
    # prepare data
    img_name = basename(img_path).split('.')[0]
    img_lrd = np.array(Image.open(img_path).resize(lr_res))
    im = preprocess(img_lrd)
    # get output
    s = time.time()
    gen_lr, gen_hr, gen_mask = generator.predict(im)
    tot = time.time()-s
    times.append(tot)
    # process raw outputs
    gen_lr = deprocess_uint8(gen_lr).reshape(lr_shape)
    gen_hr = deprocess_uint8(gen_hr).reshape(hr_shape)
    gen_mask = deprocess_mask(gen_mask).reshape(lr_h, lr_w)
    # save generated images
    Image.fromarray(img_lrd).save(join(samples_dir, img_name+'.png'))
    Image.fromarray(gen_lr).save(join(samples_dir, img_name+'_En.png'))
    Image.fromarray(gen_mask).save(join(samples_dir, img_name+'_Sal.png'))
    Image.fromarray(gen_hr).save(join(samples_dir, img_name+'_SESR.png'))
    print ("tested: {0}".format(img_path))


# some statistics
num_test = len(test_paths)
if (num_test==0):
    print ("\nFound no images for test")
else:
    print ("\nTotal images: {0}".format(num_test))
```

```
# accumulate frame processing times (without bootstrap)
Ttime, Mtime = np.sum(times[1:]), np.mean(times[1:])
print ("Time taken: {0} sec at {1} fps".format(Ttime, 1./Mtime))
print("\nSaved generated images in in {0}\n".format(samples_dir))
```

/models/Deep_ImProc_2x.json

{"class_name": "Model", "keras_version": "2.2.4", "config": {"layers": [{"class_name": "InputLayer", "config": {"dtype": "float32", "batch_input_shape": [null, 240, 320, 3], "name": "input_3", "sparse": false}, "inbound_nodes": [], "name": "input_3"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_1", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["input_3", 0, 0, {}]]], "name": "conv2d_1"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_2", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["conv2d_1", 0, 0, {}]]], "name": "conv2d_2"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_3", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides":

[1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["conv2d_2", 0, 0, {}]]], "name": "conv2d_3"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_1", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_3", 0, 0, {}]]], "name": "batch_normalization_1"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_1"}, "inbound_nodes": [[["batch_normalization_1", 0, 0, {}]]], "name": "activation_1"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_1", "axis": -1}, "inbound_nodes": [[["conv2d_2", 0, 0, {}], ["activation_1", 0, 0, {}]]], "name": "concatenate_1"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_4", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_1", 0, 0, {}]]], "name": "conv2d_4"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_2", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_4", 0, 0, {}]]], "name":

"batch_normalization_2"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_2"}, "inbound_nodes": [[["batch_normalization_2", 0, 0, {}]]], "name": "activation_2"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_2", "axis": -1}, "inbound_nodes": [[["concatenate_1", 0, 0, {}], ["activation_2", 0, 0, {}]]], "name": "concatenate_2"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_5", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_2", 0, 0, {}]]], "name": "conv2d_5"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_3", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_5", 0, 0, {}]]], "name": "batch_normalization_3"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_3"}, "inbound_nodes": [[["batch_normalization_3", 0, 0, {}]]], "name": "activation_3"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_3", "axis": -1}, "inbound_nodes": [[["concatenate_2", 0, 0, {}], ["activation_3", 0, 0, {}]]], "name": "concatenate_3"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_6", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1,

1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_3", 0, 0, {}]]], "name": "conv2d_6"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_1"}, "inbound_nodes": [[["conv2d_6", 0, 0, {}], ["conv2d_2", 0, 0, {}]]], "name": "add_1"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_7", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_1", 0, 0, {}]]], "name": "conv2d_7"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_4", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_7", 0, 0, {}]]], "name": "batch_normalization_4"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_4"}, "inbound_nodes": [[["batch_normalization_4", 0, 0, {}]]], "name": "activation_4"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_4", "axis": -1}, "inbound_nodes": [[["add_1", 0, 0, {}], ["activation_4", 0, 0, {}]]], "name": "concatenate_4"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_8", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer":

{"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_4", 0, 0, {}]]], "name": "conv2d_8"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_5", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_8", 0, 0, {}]]], "name": "batch_normalization_5"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_5"}, "inbound_nodes": [[["batch_normalization_5", 0, 0, {}]]], "name": "activation_5"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_5", "axis": -1}, "inbound_nodes": [[["concatenate_4", 0, 0, {}], ["activation_5", 0, 0, {}]]], "name": "concatenate_5"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_9", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_5", 0, 0, {}]]], "name": "conv2d_9"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_6", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_9", 0, 0, {}]]], "name": "batch_normalization_6"}, {"class_name": "Activation", "config": {"activation": "relu",

"trainable": true, "name": "activation_6"}, "inbound_nodes": [[["batch_normalization_6", 0, 0, {}]]], "name": "activation_6"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_6", "axis": -1}, "inbound_nodes": [[["concatenate_5", 0, 0, {}], ["activation_6", 0, 0, {}]]], "name": "concatenate_6"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_10", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_6", 0, 0, {}]]], "name": "conv2d_10"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_2"}, "inbound_nodes": [[["conv2d_10", 0, 0, {}], ["add_1", 0, 0, {}]]], "name": "add_2"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_11", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_2", 0, 0, {}]]], "name": "conv2d_11"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_7", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_11", 0, 0, {}]]], "name": "batch_normalization_7"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_7"}, "inbound_nodes": [[["batch_normalization_7", 0, 0, {}]]], "name": "activation_7"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_8", "axis": -1}, "inbound_nodes": [[["add_2", 0, 0, {}], ["activation_7", 0, 0, {}]]], "name": "concatenate_8"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_12", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_8", 0, 0, {}]]], "name": "conv2d_12"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_8", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_12", 0, 0, {}]]], "name": "batch_normalization_8"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_8"}, "inbound_nodes": [[["batch_normalization_8", 0, 0, {}]]], "name": "activation_8"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_9", "axis": -1}, "inbound_nodes": [[["concatenate_8", 0, 0, {}], ["activation_8", 0, 0, {}]]], "name": "concatenate_9"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_13", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_9", 0, 0, {}]]], "name": "conv2d_13"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_9", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_13", 0, 0, {}]]], "name": "batch_normalization_9"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_9"}, "inbound_nodes": [[["batch_normalization_9", 0, 0, {}]]], "name": "activation_9"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_10", "axis": -1}, "inbound_nodes": [[["concatenate_9", 0, 0, {}], ["activation_9", 0, 0, {}]]], "name": "concatenate_10"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_14", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_10", 0, 0, {}]]], "name": "conv2d_14"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_3"}, "inbound_nodes": [[["conv2d_14", 0, 0, {}], ["add_2", 0, 0, {}]]], "name": "add_3"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_15", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_3", 0, 0, {}]]], "name": "conv2d_15"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_10", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_15", 0, 0, {}]]], "name": "batch_normalization_10"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_10"}, "inbound_nodes": [[["batch_normalization_10", 0, 0, {}]]], "name": "activation_10"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_12", "axis": -1}, "inbound_nodes": [[["add_3", 0, 0, {}], ["activation_10", 0, 0, {}]]], "name": "concatenate_12"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_16", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_12", 0, 0, {}]]], "name": "conv2d_16"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_11", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_16", 0, 0, {}]]], "name": "batch_normalization_11"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_11"}, "inbound_nodes": [[["batch_normalization_11", 0, 0, {}]]], "name": "activation_11"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_13", "axis": -1}, "inbound_nodes": [[["concatenate_12", 0, 0, {}], ["activation_11", 0, 0, {}]]], "name": "concatenate_13"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_17", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_13", 0, 0, {}]]], "name": "conv2d_17"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_12", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_17", 0, 0, {}]]], "name": "batch_normalization_12"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_12"}, "inbound_nodes": [[["batch_normalization_12", 0, 0, {}]]], "name": "activation_12"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_14", "axis": -1}, "inbound_nodes": [[["concatenate_13", 0, 0, {}], ["activation_12", 0, 0, {}]]], "name": "concatenate_14"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_18", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_14", 0, 0, {}]]], "name": "conv2d_18"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_4"}, "inbound_nodes": [[["conv2d_18", 0, 0, {}], ["add_3", 0, 0, {}]]], "name": "add_4"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_19", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_4", 0, 0, {}]]], "name": "conv2d_19"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_13", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_19", 0, 0, {}]]], "name": "batch_normalization_13"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_13"}, "inbound_nodes": [[["batch_normalization_13", 0, 0, {}]]], "name": "activation_13"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_16", "axis": -1}, "inbound_nodes": [[["add_4", 0, 0, {}], ["activation_13", 0, 0, {}]]], "name": "concatenate_16"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_20", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_16", 0, 0, {}]]], "name": "conv2d_20"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_14", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_20", 0, 0, {}]]], "name": "batch_normalization_14"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_14"}, "inbound_nodes": [[["batch_normalization_14", 0, 0, {}]]], "name": "activation_14"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_17", "axis": -1}, "inbound_nodes": [[["concatenate_16", 0, 0, {}], ["activation_14", 0, 0, {}]]], "name": "concatenate_17"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_21", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_17", 0, 0, {}]]], "name": "conv2d_21"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_15", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_21", 0, 0, {}]]], "name": "batch_normalization_15"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_15"}, "inbound_nodes": [[["batch_normalization_15", 0, 0, {}]]], "name": "activation_15"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_18", "axis": -1}, "inbound_nodes": [[["concatenate_17", 0, 0, {}], ["activation_15", 0, 0, {}]]], "name": "concatenate_18"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_22", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_18", 0, 0, {}]]], "name": "conv2d_22"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_5"}, "inbound_nodes": [[["conv2d_22", 0, 0, {}], ["add_4", 0, 0, {}]]], "name": "add_5"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_23", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_5", 0, 0, {}]]], "name": "conv2d_23"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_16", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_23", 0, 0, {}]]], "name": "batch_normalization_16"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_16"}, "inbound_nodes": [[["batch_normalization_16", 0, 0, {}]]], "name": "activation_16"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_20", "axis": -1}, "inbound_nodes": [[["add_5", 0, 0, {}], ["activation_16", 0, 0, {}]]], "name": "concatenate_20"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_24", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_20", 0, 0, {}]]], "name": "conv2d_24"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_17", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_24", 0, 0, {}]]], "name": "batch_normalization_17"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_17"}, "inbound_nodes": [[["batch_normalization_17", 0, 0, {}]]], "name": "activation_17"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_21", "axis": -1}, "inbound_nodes": [[["concatenate_20", 0, 0, {}], ["activation_17", 0, 0, {}]]], "name": "concatenate_21"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_25", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_21", 0, 0, {}]]], "name": "conv2d_25"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_18", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_25", 0, 0, {}]]], "name": "batch_normalization_18"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_18"}, "inbound_nodes": [[["batch_normalization_18", 0, 0, {}]]], "name": "activation_18"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_22", "axis": -1}, "inbound_nodes": [[["concatenate_21", 0, 0, {}], ["activation_18", 0, 0, {}]]], "name": "concatenate_22"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_26", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_22", 0, 0, {}]]], "name": "conv2d_26"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_6"}, "inbound_nodes": [[["conv2d_26", 0, 0, {}], ["add_5", 0, 0, {}]]], "name": "add_6"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_27", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_6", 0, 0, {}]]], "name": "conv2d_27"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_19", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_27", 0, 0, {}]]], "name": "batch_normalization_19"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_19"}, "inbound_nodes": [[["batch_normalization_19", 0, 0, {}]]], "name": "activation_19"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_24", "axis": -1}, "inbound_nodes": [[["add_6", 0, 0, {}], ["activation_19", 0, 0, {}]]], "name": "concatenate_24"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_28", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_24", 0, 0, {}]]], "name": "conv2d_28"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_20", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_28", 0, 0, {}]]], "name": "batch_normalization_20"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_20"}, "inbound_nodes": [[["batch_normalization_20", 0, 0, {}]]], "name": "activation_20"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_25", "axis": -1}, "inbound_nodes": [[["concatenate_24", 0, 0, {}], ["activation_20", 0, 0, {}]]], "name": "concatenate_25"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_29", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_25", 0, 0, {}]]], "name": "conv2d_29"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_21", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_29", 0, 0, {}]]], "name": "batch_normalization_21"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_21"}, "inbound_nodes": [[["batch_normalization_21", 0, 0, {}]]], "name": "activation_21"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_26", "axis": -1}, "inbound_nodes": [[["concatenate_25", 0, 0, {}], ["activation_21", 0, 0, {}]]], "name": "concatenate_26"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_30", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_26", 0, 0, {}]]], "name": "conv2d_30"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_7"}, "inbound_nodes": [[["conv2d_30", 0, 0, {}], ["add_6", 0, 0, {}]]], "name": "add_7"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_31", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_7", 0, 0, {}]]], "name": "conv2d_31"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_22", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_31", 0, 0, {}]]], "name": "batch_normalization_22"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_22"}, "inbound_nodes": [[["batch_normalization_22", 0, 0, {}]]], "name": "activation_22"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_28", "axis": -1}, "inbound_nodes": [[["add_7", 0, 0, {}], ["activation_22", 0, 0, {}]]], "name": "concatenate_28"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_32", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros",

"config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_28", 0, 0, {}]]], "name": "conv2d_32"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_23", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_32", 0, 0, {}]]], "name": "batch_normalization_23"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_23"}, "inbound_nodes": [[["batch_normalization_23", 0, 0, {}]]], "name": "activation_23"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_29", "axis": -1}, "inbound_nodes": [[["concatenate_28", 0, 0, {}], ["activation_23", 0, 0, {}]]], "name": "concatenate_29"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_33", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_29", 0, 0, {}]]], "name": "conv2d_33"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_24", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_33", 0, 0, {}]]], "name": "batch_normalization_24"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_24"}, "inbound_nodes": [[["batch_normalization_24", 0, 0, {}]]], "name": "activation_24"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_30", "axis": -1}, "inbound_nodes": [[["concatenate_29", 0, 0, {}], ["activation_24", 0, 0, {}]]], "name": "concatenate_30"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_34", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_30", 0, 0, {}]]], "name": "conv2d_34"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_8"}, "inbound_nodes": [[["conv2d_34", 0, 0, {}], ["add_7", 0, 0, {}]]], "name": "add_8"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_35", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_8", 0, 0, {}]]], "name": "conv2d_35"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_25", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_35", 0, 0, {}]]], "name": "batch_normalization_25"}, {"class_name": "Activation", "config":

{"activation": "relu", "trainable": true, "name": "activation_25"}, "inbound_nodes": [[["batch_normalization_25", 0, 0, {}]]], "name": "activation_25"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_32", "axis": -1}, "inbound_nodes": [[["add_8", 0, 0, {}], ["activation_25", 0, 0, {}]]], "name": "concatenate_32"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_36", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_32", 0, 0, {}]]], "name": "conv2d_36"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_26", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_36", 0, 0, {}]]], "name": "batch_normalization_26"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_26"}, "inbound_nodes": [[["batch_normalization_26", 0, 0, {}]]], "name": "activation_26"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_7", "axis": -1}, "inbound_nodes": [[["add_1", 0, 0, {}], ["add_2", 0, 0, {}]]], "name": "concatenate_7"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_33", "axis": -1}, "inbound_nodes": [[["concatenate_32", 0, 0, {}], ["activation_26", 0, 0, {}]]], "name": "concatenate_33"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_11", "axis": -1}, "inbound_nodes": [[["concatenate_7", 0, 0, {}], ["add_3", 0, 0, {}]]], "name": "concatenate_11"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer":

{"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_37", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["concatenate_33", 0, 0, {}]]], "name": "conv2d_37"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_15", "axis": -1}, "inbound_nodes": [[["concatenate_11", 0, 0, {}], ["add_4", 0, 0, {}]]], "name": "concatenate_15"}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_27", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.5, "center": true}, "inbound_nodes": [[["conv2d_37", 0, 0, {}]]], "name": "batch_normalization_27"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_19", "axis": -1}, "inbound_nodes": [[["concatenate_15", 0, 0, {}], ["add_5", 0, 0, {}]]], "name": "concatenate_19"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_27"}, "inbound_nodes": [[["batch_normalization_27", 0, 0, {}]]], "name": "activation_27"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_23", "axis": -1}, "inbound_nodes": [[["concatenate_19", 0, 0, {}], ["add_6", 0, 0, {}]]], "name": "concatenate_23"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_34", "axis": -1}, "inbound_nodes": [[["concatenate_33", 0, 0, {}], ["activation_27", 0, 0, {}]]], "name": "concatenate_34"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_27", "axis": -1}, "inbound_nodes": [[["concatenate_23", 0, 0, {}], ["add_7", 0, 0, {}]]], "name": "concatenate_27"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null,

"kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_38", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_34", 0, 0, {}]]], "name": "conv2d_38"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_31", "axis": -1}, "inbound_nodes": [[["concatenate_27", 0, 0, {}], ["add_8", 0, 0, {}]]], "name": "concatenate_31"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_9"}, "inbound_nodes": [[["conv2d_38", 0, 0, {}], ["add_8", 0, 0, {}]]], "name": "add_9"}, {"class_name": "Concatenate", "config": {"trainable": true, "name": "concatenate_35", "axis": -1}, "inbound_nodes": [[["concatenate_31", 0, 0, {}], ["add_9", 0, 0, {}]]], "name": "concatenate_35"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_39", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["concatenate_35", 0, 0, {}]]], "name": "conv2d_39"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_40", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["conv2d_39", 0, 0, {}]]], "name": "conv2d_40"}, {"class_name": "Add", "config": {"trainable": true, "name": "add_10"}, "inbound_nodes":

[[["conv2d_40", 0, 0, {}], ["conv2d_1", 0, 0, {}]]], "name": "add_10"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_41", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 128, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["add_10", 0, 0, {}]]], "name": "conv2d_41"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_43", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [1, 1]}, "inbound_nodes": [[["conv2d_41", 0, 0, {}]]], "name": "conv2d_43"}, {"class_name": "UpSampling2D", "config": {"data_format": "channels_last", "interpolation": "nearest", "trainable": true, "name": "up_sampling2d_1", "size": [2, 2]}, "inbound_nodes": [[["conv2d_43", 0, 0, {}]]], "name": "up_sampling2d_1"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_44", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 256, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["up_sampling2d_1", 0, 0, {}]]], "name": "conv2d_44"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_46",

"bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["add_9", 0, 0, {}]]], "name": "conv2d_46"}, {"class_name": "Activation", "config": {"activation": "relu", "trainable": true, "name": "activation_28"}, "inbound_nodes": [[["conv2d_44", 0, 0, {}]]], "name": "activation_28"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_47", "bias_regularizer": null, "bias_constraint": null, "activation": "linear", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 32, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["conv2d_46", 0, 0, {}]]], "name": "conv2d_47"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_42", "bias_regularizer": null, "bias_constraint": null, "activation": "tanh", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 3, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["conv2d_41", 0, 0, {}]]], "name": "conv2d_42"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_45", "bias_regularizer": null, "bias_constraint": null, "activation": "tanh", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 3, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["activation_28", 0, 0, {}]]], "name":

"conv2d_45"}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_48", "bias_regularizer": null, "bias_constraint": null, "activation": "sigmoid", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 1, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}, "inbound_nodes": [[["conv2d_47", 0, 0, {}]]], "name": "conv2d_48"}], "input_layers": [["input_3", 0, 0]], "output_layers": [["conv2d_42", 0, 0], ["conv2d_45", 0, 0], ["conv2d_48", 0, 0]], "name": "model_2"}, "backend": "tensorflow"}

Todo: презентацию сюда