

Кафедра _____ *прикладної математики* _____

ПОЯСНЮВАЛЬНА ЗАПISКА

до бакалаврської кваліфікаційної роботи на тему

Розроблення мобільного додатку для визначення хмарності неба

Студент група ПМ-41, спеціальність 113, Тимощук П. Т.

/група, шифр, прізвище та ініціали/

Керівник роботи



/підпис/

(Топилко П. І.)

/прізвище та ініціали/

Консультанти

(_____)

(_____)

(_____)

(_____)

Завідувач кафедри

_____ Костробій П.П.
« 01 » червня 20 21 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут ІМФН Кафедра прикладної математики

Спеціальність 113 Прикладна математика

«ЗАТВЕРДЖУЮ»

Завідувач кафедри прикладної математики

Костробій П.П.

« 01 » червня 2021 р.

ЗАВДАННЯ

на кваліфікаційну роботу (проект) студента групи ПМ-41 ОР бакалавр

Тимощука Павла Тарасовича

/прізвище, ім'я, по батькові/

1. Тема роботи (проекту) Розроблення мобільного додатку для визначення хмарності неба
/у разі виконання комплексної роботи в дужках вказується "комплексна робота (проект)"/

затверджена наказом по університету від « » 2021 р. №

2. Термін подання студентом закінченої роботи (проекту) 31 травня 2021 року

3. Вихідні дані для роботи (проекту) Deep Convolutional Neural Network for Cloud Coverage Estimation from Snapshot Camera Images [Електронний ресурс], Image Segmentation iOS [Електронний ресурс], Cloud Fraction [Електронний ресурс].

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити) Деталізація пікселів хмари на основі середнього кольору зображення. Оптимізація швидкодії комп'ютерної програми. Розроблення візуального представлення значення відсотка хмарності.

5. Перелік графічного матеріалу

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту) мова програмування Swift, бібліотека Fritz AI, середовище Xcode.

7. Консультування роботи (проекту), із зазначенням розділів роботи

Розділ	Консультант	Завдання видав		Завдання прийняв	
		підпис	Дата	підпис	дата

8. Дата, коли видано завдання 18 лютого 2021 року

Керівник

Топилко П. І.

/підпис/

Завдання прийняв до виконання

Тимошук П. Т.

/підпис/

№ з/п	Назва етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1	Ознайомлення з літературою про сегментацію зображень	22.02 - 26.04	виконано
2	Програмна реалізація сегментації зображень	15.03 - 05.04	виконано
3	Програмна реалізація визначення належності пікселя до хмари	19.04 - 30.04	виконано
4	Оптимізація швидкодії комп'ютерної програми	03.05 - 07.05	виконано
5	Розроблення візуального представлення значення відсотка хмарності	07.05 - 11.05	виконано
6	Написання та оформлення згідно вимог тексту бакалаврської кваліфікаційної роботи	22.05 - 31.05	виконано

Керівник

Топилко П. І.

/підпис/

Завдання прийняв до виконання

Тимошук П. Т.

/підпис/

Розглянуто задачу визначення хмарності неба. Досліджено метод сегментації зображення. Досліджено метод оцінки характеристик пікселя для визначення його належності до хмари. Розглянуто задачу деталізації пікселів хмари на основі середнього кольору зображення, оптимізації швидкодії програми та розроблення візуального представлення значення відсотка хмарності. Розроблено мобільний застосунок, для визначення хмарності неба.

The problem of determining the cloud cover of the sky is considered. The method of image segmentation is investigated. The method of estimating the characteristics of a pixel to determine its belonging to the cloud is investigated. The problem of detailing cloud pixels based on the average color of the image, optimizing the program performance and developing a visual representation of the percentage of clouds is considered. A program has been written that implements this task.

Зміст

Вступ.....	6
1. Сегментація зображення	7
1.1. Теоретичні аспекти сегментації зображення.....	7
1.2. Використання бібліотеки Fritz AI для сегментації зображення, яке може містити фрагменти неба	7
2. Розпізнання пікселів хмари на зображенні неба	8
2.1. Використання характеристик пікселя для визначення його належності до хмари	9
2.2. Деталізація пікселів хмари на основі середнього кольору зображення	10
3. Оптимізація часу виконання розрахунку результату.....	12
3.1. Розбиття зображення на групи фіксованого розміру для проведення обчислень з використанням меншої кількості пікселів	12
3.2. Пошук оптимального розміру груп розбиття зображення.....	13
4. Розроблення візуального представлення значення відсотка хмарності.....	15
Опис програми	16
Висновки	17
Список використаної літератури	18
Додаток 1. Текст програми	19
Додаток 2. Результати роботи програми	28

Вступ

Хмари грають багато важливих ролей у кліматичній системі. Зокрема, будучи яскравими об'єктами у видимій частині сонячного спектру, вони ефективно відбивають світло в космосі і, таким чином, сприяють охолодженню планети. Хмарний покрив відіграє важливу роль в енергетичному балансі атмосфери, а його варіація є наслідком зміни клімату. Також хмарність неба впливає на сонячну енергетику. Чим меншим є відсоток хмарності, тим більшим є обсяг виробництва енергії на сонячних електростанціях.

Для сонячних електростанцій є важливо передбачувати відсоток хмарності для оцінки їх продуктивності. Одним з рішень є розроблення додатків для визначення хмарності в реальному часі, використовуючи смартфон.

Для обчислення хмарності неба можна використовувати розділення пікселя між сегментами хмари та неба. Цей метод добре працює для зображень з певним середнім кольором неба, але для різних зображень середні кольори можуть сильно відрізнятись, що приведе до значної втрати точності. Також швидкодія даного методу є низькою, оскільки виконуються складні обчислення для кожного пікселя зображення, що є дуже затратним у часі.

1. Сегментація зображення

При визначенні хмарності неба не можна бути впевненим що на зображенні не буде сторонніх об'єктів, таких як будівлі, рослини або топологічні особливості. Щоб позбутись зайвих ділянок зображення необхідно застосувати метод розділення зображення, який також відомий як сегментація.

1.1. Теоретичні аспекти сегментації зображення

З комп'ютерної точки зору, сегментація – це процес розділення цифрового зображення на декілька сегментів. Результатом сегментації зображення є фрагменти, які разом покривають все зображення. Для отримання фрагмента неба було використано бібліотеку Fritz AI. В даній бібліотеці використовується семантична сегментація.

Семантична сегментація відноситься до процесу прив'язки кожного пікселя на зображенні до мітки класу. Ці мітки можуть містити людину, машину, квітку, предмет меблів тощо.

1.2. Використання бібліотеки Fritz AI для сегментації зображення, яке може містити фрагменти неба

Підключивши бібліотеку Fritz AI та перетворивши результат сегментації, отримано наступні зображення:



а)



б)

Рисунок 1. Приклад розпізнавання неба на фоні лісу



а)



б)

Рисунок 2. Приклад розпізнавання неба на фоні гір



а)



б)

Рисунок 3. Приклад розпізнавання неба на зображенні де фрагменти неба відсутні

На рисунку. 1 та рисунку. 2 (а) виділено фрагменти неба, які зображено на рисунку. 1 та рисунку. 2 (б) відповідно. Зображення на рисунку. 3 (а) не містить сегментів неба, тому рисунку. 3 (б) є порожнім.

Отримані зображення не містять фрагментів, які можуть внести похибки у подальші обчислення, тому їх можна використовувати для обчислення відсотка хмарності неба.

2. Розпізнання пікселів хмари на зображенні неба

Для того щоб розрахувати відсоток хмарності неба необхідно обчислити кількість пікселів хмари і поділити їх на кількість всіх пікселів неба. Це призводить до вирішення задачі визначення належності пікселя до хмари.

2.1. Використання характеристик пікселя для визначення його належності до хмари

Класифікація окремих пікселів на сегменти неба та хмари відбувається на основі кольору кожного з них. Піксель класифікується на сегменти неба чи хмари. Для отримання даних для оцінки окремих пікселів на сегменти неба та хмари на основі кольору пікселя використано матеріали зі статті «Deep Convolutional Neural Network for Cloud Coverage Estimation from Snapshot Camera Images».

Щоб визначити поріг розділення між сегментами хмари та неба, автори статті зібрали 120 зображень, які містять лише небо та хмари, тобто без включення об'єктів, таких як тварини, рослини, будівлі та топологічні особливості. Потім було проаналізовано їх у відтінку, насиченні та просторі значення (яскравості) (HSV), що є загальним циліндрично-координатним поданням пікселів у кольоровому просторі RGB.

Вибрано наступні канали:

- r (червоний)
- g (зелений)
- b (синій)
- s_{cyl} (насиченість у циліндричній системі координат)
- s_{cone} (насиченість у конусній системі координат)
- v (яскравість)
- br (різниця, яка визначається як $br = \frac{b - r}{b + r}$)

Варто зазначити, що r , g та b нормуються до значень від 0 до 1.

Ці канали були обрані, оскільки вони мали значну дисперсію між зображеннями неба. Зібрані цифрові зображення, які містять лише небо та лише хмари, забезпечили наступний критерій V :

$$V = -6.28r + 0.454g - 4.11b - 1.81s_{cyl} - 4.04s_{cone} + 8.88v + 1.53br + 0.586.$$

Якщо $V < 0$, то піксель вважається хмарним пікселем.





Але ця формула не завжди дає точні результати, оскільки середні кольори неба та хмар можуть сильно відрізнятися. Для вирішення цієї

проблеми потрібно коригувати кольорові канали кожного пікселя в залежності від середнього кольору зображення.

2.2. Деталізація пікселів хмари на основі середнього кольору зображення

Для знаходження величини корекції проведено аналіз п'яти зображень з різними середніми кольорами. Обчисливши середнє арифметичне кожної компоненти середніх кольорів (корегований середній колір), отримано результати у яких змінено один з параметрів. Після цього отримано нові результати.

Таблиця 1. Зміна результату в залежності від корегування середнього кольору

Зображення	Початковий результат	Ітерації	Результат ітерації	Очікуваний результат
	61% Середній колір зображення: $r = 0.822368$ $g = 0.842105$ $b = 0.888158$	1 ітерація	72%	80-90%
		2 ітерація	70%	
		3 ітерація	73%	
		4 ітерація	88%	
		5 ітерація	80%	
	68% Середній колір зображення: $r = 0.521472$ $g = 0.662577$ $b = 0.846626$	1 ітерація	11%	40-50%
		2 ітерація	2%	
		3 ітерація	15%	
		4 ітерація	100%	
		5 ітерація	90%	
	12% Середній колір зображення: $r = 0.247934$ $g = 0.586777$ $b = 0.867769$	1 ітерація	0%	0-5%
		2 ітерація	0%	
		3 ітерація	0%	
		4 ітерація	0%	
		5 ітерація	0%	
	45% Середній колір зображення: $r = 0.598639$ $g = 0.755102$ $b = 0.85034$	1 ітерація	45%	40-50%
		2 ітерація	42%	
		3 ітерація	46%	
		4 ітерація	63%	
		5 ітерація	51%	
	55%	1 ітерація	54%	70-80%
		2 ітерація	48%	

	Середній колір зображення: $r = 0.623762$ $g = 0.69802$ $b = 0.80198$	3 ітерація	55%	
		4 ітерація	85%	
		5 ітерація	75%	

Проаналізувавши покращення і погіршення результатів для різних зображень змінено інший параметр. Ці дії необхідно продовжувати поки не буде отримано результати максимально близькі до очікуваних.

Таблиця 2. Зміна корегованого середнього кольору в залежності від попередньо отриманих результатів

Ітерація	Корегований середній колір
1 ітерація	($r = 0.6$; $g = 0.75$; $b = 0.85$)
2 ітерація	($r = 0.6$; $g = 0.5$; $b = 0.85$)
3 ітерація	($r = 0.6$; $g = 0.85$; $b = 0.85$)
4 ітерація	($r = 0.4$; $g = 0.85$; $b = 0.85$)
5 ітерація	($r = 0.5$; $g = 0.85$; $b = 0.8$)

Проаналізувавши дані з таблиці 1, оптимальним корегованим середнім кольором буде ($r = 0.5$; $g = 0.85$; $b = 0.8$), оскільки він забезпечує результати найближчі до очікуваних.

3. Оптимізація часу виконання розрахунку результату

Оскільки сучасні зображення мають високу роздільну здатність то критерій V необхідно обчислювати дуже багато разів (12000000 разів для зображення розмірністю 4000×3000 пікселів, яке містить тільки небо), що є затратно у часі.

3.1. Розбиття зображення на групи фіксованого розміру для проведення обчислень з використанням меншої кількості пікселів

Для оптимізації часу виконання розрахунку результату прийнято рішення обчислювати критерій V для частини пікселів. Для цього необхідно розділити зображення на групи певної розмірності, так щоб досягти максимальної швидкодії з найменшими втратами точності результатів.

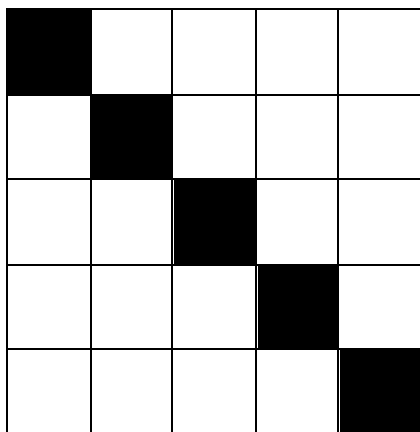







Рисунок 4. Група об'єднання пікселів 5×5

На рисунку. 4 зображено групу об'єднання пікселів розмірністю 5×5 . Обчислюватись критерій V буде тільки для пікселів зображених чорним кольором. Для випадку групи об'єднання пікселів розмірністю $n \times n$ кількість складних обчислень зменшиться у n разів.

3.2. Пошук оптимального розміру груп розбиття зображення

Таблиця 3. Зміна часу та результату обчислення хмарності в залежності від зміни розмірів груп об'єднання пікселів

Зображення і його роздільна здатність	Розмір груп об'єднання пікселів	Час обчислень	Результат обчислень
w: 4752 h: 3168 	1*1	113 с	81.3817%
	5*5	30 с	81.381714%
	30*30	13 с	81.40434%
	100*100	10 с	81.29915%
	500*500	9 с	81.342735%
w: 4000 h: 3000 	1*1	59 с	42.43563%
	5*5	18 с	42.435634%
	30*30	9 с	42.41984%
	100*100	8 с	42.41055%
	500*500	7 с	42.42921%
w: 4000 h: 3000 	1*1	45 с	0.07061231%
	5*5	15 с	0.0702467%
	30*30	9 с	0.0664146%
	100*100	7 с	0.050503287%
	500*500	7 с	0.034644034%
w: 4000 h: 3000 	1*1	89 с	52.706326%
	5*5	24 с	52.70325%
	30*30	10 с	52.682503%
	100*100	8 с	52.8075%
	500*500	7 с	53.95%
w: 4000 h: 3000 	1*1	71 с	74.36131%
	5*5	20 с	74.36032%
	30*30	9 с	74.37062%
	100*100	8 с	74.45019%
	500*500	7 с	73.603035%

Використовуючи дані з таблиці 3 обчислено середній приріст швидкодії відносно попереднього розміру груп об'єднання (δ) та середнє падіння точності відносно результату з розміром групи об'єднання 1*1 (ϵ)

$$\delta = \frac{\sum_{i=1}^n |\tau_i - \tau_{i-1}|}{n},$$

де τ - час обчислень, n - кількість зображень;

$$\epsilon = \frac{\sum_{i=1}^n |x_i - x_1|}{n},$$

де x - результат обчислень, n - кількість зображень.

Таблиця 4. Середній приріст швидкодії та падіння точності при зміні розмірів груп об'єднання пікселів

Розмір груп об'єднання пікселів	Середній приріст швидкодії	Середнє падіння точності
1*1	1x	0%
5*5	3.46x	0,1049929976%
30*30	2.12x	1,213495323%
100*100	1.22x	5,790017853%
500*500	1.11x	10,87600793%

Проаналізувавши дані з таблиці 4, можна дійти висновку, що оптимальним буде розбиття на групи 30*30 пікселів, оскільки в такому випадку час обчислення буде меншим в 7.35 разів у порівнянні з часом обчислення без розбиття на групи, а похибки будуть достатньо малими щоб ними знехтувати. Також слід зазначити, що для зображень з низькою роздільною здатністю великі розміри груп розбиття приведуть до критичної втрати точності обчислень, тому для зображення необхідно корегувати розмір розбиття залежно від роздільної здатності зображення.

4. Розроблення візуального представлення значення відсотка хмарності

Метеорологи оцінюють хмарність за 10-бальною шкалою. Наприклад, 0 балів — небо ясне, 5 балів — 50 % неба закрито хмарами, 10 балів — усе небо закрито хмарами. У даному випадку хмарний покрив було оцінено через відсоток хмарного покриття, від 0% до 100%, оскільки *CloudPercentage* (відсоток хмарності неба) розраховується за формулою

$$CloudPercentage = \frac{N_{cloud}}{N_{all}},$$

де N_{all} - загальна кількість пікселів неба, а N_{cloud} - кількість пікселів хмар.

Пересічному користувачеві не потрібно відображати багато значень після коми, а достатньо цілих чисел. Тому, для кращого сприйняття результатів обчислення, їх необхідно заокруглювати.

Функція яка реалізує візуальне представлення відсотка хмарності:

```
func addCloudPercentageLabel(view: UIView) {
    cloudPercentageLabel.frame.size = CGSize(width: view.frame.width, height: 50)
    cloudPercentageLabel.frame.origin = CGPoint(x: 0, y: view.frame.height/2)
    view.addSubview(cloudPercentageLabel)
    cloudPercentageLabel.textAlignment = .center
    cloudPercentageLabel.font = UIFont(name: "AppleSDGothicNeo-Bold", size: 30)
}
```

Заокруглення значення відсотка хмарності:

```
extension Float {
    func roundedValue(base: Int) -> String {
        let roundedInt = Int(roundf(self))%base < (base/2)+1 ?
        Int(roundf(self))/base*base : (Int(roundf(self))/base+1)*base
        return String(roundedInt)+»%" }
}
```

Опис програми

Програма написана на мові програмування Swift у середовищі Xcode. Вона реалізовує задачу розроблення додатку для визначення хмарності неба в реальному часі.

У програмі реалізовано такі головні функції:

1. UIImagePickerController (отримання зображення з галереї);
2. addPhotoAndCalculateCloudPercentage (перетворення зображення та обчислення відсотку хмарності);
3. calculateV (класифікація окремих пікселів на сегменти неба та хмари на основі кольору пікселя);
4. averageColor (отримання середнього кольору зображення);
5. correctionValue (корегування параметру в залежності від середнього кольору)
6. roundedValue (округлення числа)

Вхідні дані програми: зображення з галереї девайсу

У результаті виконання програми отримуємо такі вихідні дані:

1. Відсоток хмарності неба, у випадку якщо вхідне зображення містить фрагмент неба;
2. Повідомлення про те, що вхідне зображення не містить фрагментів неба.

Результат програми виводиться на екран.

Висновки

Передбачення відсотка хмарності неба для сонячних електростанцій є важливим для оцінки їх продуктивності.

В представлений роботі розроблено додаток для визначення хмарності в реальному часі. Застосовано метод сегментації зображення, щоб позбутись зайвих ділянок зображення. Підключено бібліотеку Fritz AI та перетворено результат сегментації у зображення, які підходять для обчислення відсотка хмарності неба. Класифіковано окремі пікселі на сегменти хмари та неба на основі кольору кожного з них. Вибрано насиченість у циліндричній системі координат, насиченість у конусній системі координат, яскравість, червоний, зелений та синій канали, оскільки вони мали значну дисперсію між пікселями неба та хмари.

Для розрахунку відсотка хмарності неба обчислено кількість пікселів хмари та поділено їх на кількість всіх пікселів неба. Проведено коригування кольорових каналів кожного пікселя в залежності від середнього кольору зображення для покращення результатів класифікації пікселів на сегменти хмари та неба.

Оптимізовано час виконання розрахунку результату, використовуючи розділення зображення на групи певної розмірності. Досягнуто максимальної швидкодії з найменшими втратами точності результатів. Оцінено хмарний покрив через відсоток, від 0% до 100% та заокруглено результат обчислення, для кращого сприйняття користувачем.

Список використаної літератури

1. Cloud Fraction [Електронний ресурс] – Режим доступу до ресурсу: http://earthobservatory.nasa.gov/GlobalMaps/view.php?d1=MODAL2_M_CLD_FR
2. Deep Convolutional Neural Network for Cloud Coverage Estimation from Snapshot Camera Images [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/321984089_Deep_Convolutional_Neural_Network_for_Cloud_Coverage_Estimation_from_Snapshot_Camera_Images
3. Image Segmentation iOS [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.fritz.ai/develop/vision/image-segmentation/#ios>
4. SwiftUI: Read the Average Color of an Image [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/swlh/swiftui-read-the-average-color-of-an-image-c736adb43000>
5. A 2019 Guide to Semantic Segmentation [Електронний ресурс] – Режим доступу до ресурсу: <https://heartbeat.fritz.ai/a-2019-guide-to-semantic-segmentation-ca8242f5a7fc>
6. A review of semantic segmentation using deep neural networks [Електронний ресурс] – Режим доступу до ресурсу: <https://link.springer.com/article/10.1007/s13735-017-0141-z>

Додаток 1. Текст програми

```
// AnalyzingPhoto.swift
// Cloudy
// Created by Павло Тимошук on 25.11.2020.
import Foundation
import AVFoundation
import UIKit
import Vision
import Fritz
import FritzVisionSkySegmentationModelAccurate
class PhotoAnalyzer {
    init(_ photo: UIImage){
        self.photo = photo
    }
    var photo = UIImage()
    var visionModel = FritzVisionSkySegmentationModelAccurate()
    let context = CIContext()
    func createSkyImage() -> UIImage? {
        let fritzImage = FritzVisionImage(image: self.photo)
        guard let result = try? visionModel.predict(fritzImage),
            let mask = result.buildSingleClassMask(
                forClass: FritzVisionSkyClass.sky )
        else {
            return nil
        }
        guard let skyImage = createMask(of: self.photo, fromMask: mask) else { return nil }
        return skyImage
    }
    func getCloudPercentage(image: UIImage) -> Float? {
        let averageColor = image.averageColor
        print("Average image colors: ", averageColor!)
        print("Normal colors: 0.5 0.85 0.8")
        let rCorrection = Float((averageColor?.rgba.red!).correctionValue(by: 0.5))
        let gCorrection = Float((averageColor?.rgba.green!).correctionValue(by: 0.85))
        let bCorrection = Float((averageColor?.rgba.blue!).correctionValue(by: 0.80))
        print("Color correction: ", rCorrection, gCorrection, bCorrection)
        if let imageData = getDataFrom(image: image) {
            let accuracyValue = 30
            var pixelCount = 0
            var cloudPixelCount = 0
            let timeStart = Date()
            for y in 0 ..< imageData.height {
```

```

        for x in 0 ..< imageData.width {
            let index = y * imageData.width + x
            let pixel = imageData.pixels[index]
            if pixel.alpha != 0 && x%accuracyValue == y%accuracyValue {
                if calculateVSC(pixel: pixel, colorCorrection: (rCorrection, gCorrection,
bCorrection)) < 0 {
                    cloudPixelCount += 1
                }
                pixelCount += 1
            }
        }
    }
    let timeSpent = Calendar.current.dateComponents([.second], from: timeStart, to:
Date()).second!
    print("image_dimension:", "w:", imageData.width, " h:", imageData.height
,"accuracy_value: ", accuracyValue, "time_spent: ", timeSpent, "cloud_percentage: ",
Float(cloudPixelCount) / Float(pixelCount) * 100)
    return Float(cloudPixelCount) / Float(pixelCount) * 100
} else {
    return nil
}
}

func getDataFrom(image: UIImage) -> (width: Int, height: Int, pixels:
UnsafeMutableBufferPointer<Pixel>)? {
    guard let cgImage = image.cgImage else { return nil }
    let colorSpace = CGColorSpaceCreateDeviceRGB()
    var bitmapInfo: UInt32 = CGBitmapInfo.byteOrder32Big.rawValue
    bitmapInfo |= CGImageAlphaInfo.premultipliedLast.rawValue &
CGBitmapInfo.alphaInfoMask.rawValue
    let width = Int(image.size.width)
    let height = Int(image.size.height)
    let bytesPerRow = width * 4
    let imageData = UnsafeMutablePointer<Pixel>.allocate(capacity: width * height)
    guard let imageContext = CGContext(
        data: imageData,
        width: width,
        height: height,
        bitsPerComponent: 8,
        bytesPerRow: bytesPerRow,
        space: colorSpace,
        bitmapInfo: bitmapInfo
    ) else { return nil }
    imageContext.draw(cgImage, in: CGRect(x: 0, y: 0, width: width, height: height))

```

```

        return (width, height, UnsafeMutableBufferPointer<Pixel>(start: imageData, count:
width * height))
    }
    func calculateVSC(pixel: Pixel, colorCorrection: (r: Float, g: Float, b: Float)) -> Float {
        let r = Float(pixel.red)/255
        let g = Float(pixel.green)/255
        let b = Float(pixel.blue)/255
        let v = ([r,g,b].max()! + [r,g,b].min()!)/2
        let scyl: Float = ([r,g,b].max()! - [r,g,b].min()!) == 0 ? 0 : ([r,g,b].max()!-
[r,g,b].min()!)/(1-abs(2*v-1))
        let br = (b-r)/(b+r)
        return -6.28*r*colorCorrection.r + 0.454*g*colorCorrection.g -
4.11*b*colorCorrection.b - 1.8*scyl + 8.88*v + 1.53*br + 0.586  }
    func createMask(of image: UIImage, fromMask mask: UIImage, withBackground
background: UIImage? = nil) -> UIImage? {
        guard let imageCG = image.cgImage, let maskCI = mask.ciImage else { return nil }
        let imageCI = CIIImage(cgImage: imageCG)
        let background = background?.cgImage != nil ? CIIImage(cgImage:
background!.cgImage!) : CIIImage.empty()
        guard let filter = CIFilter(name: "CIBlendWithAlphaMask") else { return nil }
        filter.setValue(imageCI, forKey: "inputImage")
        filter.setValue(maskCI, forKey: "inputMaskImage")
        filter.setValue(background, forKey: "inputBackgroundImage")
        guard let maskedImage = context.createCGImage(filter.outputImage!, from:
maskCI.extent) else {
            return nil
        }
        return UIImage(cgImage: maskedImage)
    }
    struct Pixel {
        public var value: UInt32
        public var red: UInt8 {
            get {
                return UInt8(value & 0xFF)
            } set {
                value = UInt32(newValue) | (value & 0xFFFFFFF0)
            }
        }
        public var green: UInt8 {
            get {
                return UInt8((value >> 8) & 0xFF)
            } set {
                value = (UInt32(newValue) << 8) | (value & 0xFFFF00FF)
            }
        }
    }

```

```

    }
}
public var blue: UInt8 {
    get {
        return UInt8((value >> 16) & 0xFF)
    } set {
        value = (UInt32(newValue) << 16) | (value & 0xFF00FFFF)
    }
}
public var alpha: UInt8 {
    get {
        return UInt8((value >> 24) & 0xFF)
    } set {
        value = (UInt32(newValue) << 24) | (value & 0x00FFFFFF)
    } } } }
extension UIImage {
    var averageColor: UIColor? {
        var bitmap = [UInt8](repeating: 0, count: 4)
        if #available(iOS 9.0, *) {
            let context = CIContext()
            let inputImage: CImage = ciImage ?? CoreImage.CImage(cgImage: cgImage!)
            let extent = inputImage.extent
            let inputExtent = CIVector(x: extent.origin.x, y: extent.origin.y, z:
extent.size.width, w: extent.size.height)
            let filter = CIFilter(name: "CIAreaAverage", parameters: [kCIInputImageKey:
inputImage, kCIInputExtentKey: inputExtent])!
            let outputImage = filter.outputImage!
            let outputExtent = outputImage.extent
            assert(outputExtent.size.width == 1 && outputExtent.size.height == 1)
            context.render(outputImage, toBitmap: &bitmap, rowBytes: 4, bounds:
CGRect(x: 0, y: 0, width: 1, height: 1), format: .RGBA8, colorSpace:
CGColorSpaceCreateDeviceRGB())
        } else {
            let context = CGContext(data: &bitmap, width: 1, height: 1, bitsPerComponent: 8,
bytesPerRow: 4, space: CGColorSpaceCreateDeviceRGB(), bitmapInfo:
CGImageAlphaInfo.noneSkipFirst.rawValue)!
            let inputImage = cgImage ?? CIContext().createCGImage(ciImage!, from:
ciImage!.extent)
            context.draw(inputImage!, in: CGRect(x: 0, y: 0, width: 1, height: 1))
        }
        let alpha: CGFloat = CGFloat(bitmap[3]) / 255.0
        let multiplier: CGFloat = alpha*255.0
    }
}

```

```

        return UIColor(red: CGFloat(bitmap[0])/multiplier, green:
CGFloat(bitmap[1])/multiplier, blue: CGFloat(bitmap[2])/multiplier, alpha: alpha)
    }
}
extension UIColor {
    var rgba: (red: CGFloat, green: CGFloat, blue: CGFloat, alpha: CGFloat) {
        var red: CGFloat = 0
        var green: CGFloat = 0
        var blue: CGFloat = 0
        var alpha: CGFloat = 0
        getRed(&red, green: &green, blue: &blue, alpha: &alpha)
        return (red, green, blue, alpha)
    }
}
extension Float {
    func correctionValue(by: Float) -> Float {
        if self > by { return 1+(self-by)/by }
        else if self < by { return 1-(by-self)/by }
        else { return 1 }
    }
    func roundedValue(base: Int) -> String {
        let roundedInt = Int(roundf(self))%base < (base/2)+1 ? Int(roundf(self))/base*base :
(Int(roundf(self))/base+1)*base
        return String(roundedInt)+"%"
    }
}

```

```

// ViewController.swift
// Cloudy
// Created by Павло Тимошук on 25.11.2020.
import UIKit
import Photos
class ViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {
    var imageView: UIImageView!
    var imagePicker = UIImagePickerController()
    var isCurrentImageSky = Bool()
    var image = UIImage()
    var skyImage = UIImage()
    var cloudPercentageLabel = UILabel()
    override func viewDidLoad() {
        super.viewDidLoad()
        addBackground(view: self.view)
    }
}

```

```

        addSelectImageButton(view: self.view)
        addCloudPercentageLabel(view: self.view)
        self.view.insertSubview(imageViewCreating(), at: 0)
    }
    func addBackground(view: UIView) {
        let layer0 = CAGradientLayer()
        layer0.colors = [
            UIColor(red: 0.883, green: 0.968, blue: 1, alpha: 1).CGColor,
            UIColor(red: 0.351, green: 0.816, blue: 0.992, alpha: 1).CGColor,
            UIColor(red: 0.197, green: 0.269, blue: 0.296, alpha: 1).CGColor
        ]
        layer0.locations = [0, 0.5, 1]
        layer0.startPoint = CGPoint(x: 0.25, y: 0.5)
        layer0.endPoint = CGPoint(x: 0.75, y: 0.5)
        layer0.transform = CATransform3DMakeAffineTransform(CGAffineTransform(a:
0, b: 1, c: -1, d: 0, tx: 1, ty: 0))
        layer0.bounds = view.bounds.insetBy(dx: -1*view.bounds.size.width, dy: -
1*view.bounds.size.height)
        layer0.position = view.center
        view.layer.addSublayer(layer0)
        let imageSize: CGFloat = 150
        let padding = (view.frame.height - 3*imageSize)/6
        let sunImage = UIImageView()
        sunImage.layer.opacity = 0.4
        sunImage.frame = CGRect(x: (view.frame.width-imageSize)/2,
                                y: padding,
                                width: imageSize,
                                height: imageSize)
        sunImage.image = UIImage(named: "sun-image")
        view.addSubview(sunImage)
        let cloudedSun = UIImageView()
        cloudedSun.layer.opacity = 0.4
        cloudedSun.frame = CGRect(x: (view.frame.width-imageSize)/2,
                                y: (view.frame.height-imageSize)/2,
                                width: imageSize,
                                height: imageSize)
        cloudedSun.image = UIImage(named: "clouded-sun-image")
        view.addSubview(cloudedSun)
        let cloud = UIImageView()
        cloud.layer.opacity = 0.4
        cloud.frame = CGRect(x: (view.frame.width-imageSize)/2,
                                y: view.frame.height-imageSize-padding,
                                width: imageSize,

```



```

        height: imageSize)
cloud.image = UIImage(named: "cloud-image")
view.addSubview(cloud)
}
func addSelectImageButton(view: UIView) {
    let selectImageButton = UIButton()
    let selectImageButtonSize = CGSize(width: 230, height: 60)
    selectImageButton.frame = CGRect(x: (view.frame.width -
selectImageButtonSize.width)/2,
        y: view.frame.height/2 + 115,
        width: selectImageButtonSize.width,
        height: selectImageButtonSize.height)
    selectImageButton.addTarget(self, action: #selector(selectImageButtonAction), for:
.touchUpInside)

    let shadows = UIView()
    shadows.isUserInteractionEnabled = false
    shadows.frame = CGRect(x: 0,
        y: 0,
        width: selectImageButtonSize.width,
        height: selectImageButtonSize.height)
    shadows.clipsToBounds = false
    selectImageButton.addSubview(shadows)
    let shadowPath0 = UIBezierPath(roundedRect: shadows.bounds, cornerRadius: 20)
    let layer0 = CALayer()
    layer0.shadowPath = shadowPath0.cgPath
    layer0.shadowColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.2).CGColor
    layer0.shadowOpacity = 1
    layer0.shadowRadius = 3
    layer0.shadowOffset = CGSize(width: 4, height: 4)
    layer0.bounds = shadows.bounds
    layer0.position = shadows.center
    shadows.layer.addSublayer(layer0)
    let layer1 = CALayer()
    layer1.bounds = shadows.bounds
    layer1.position = shadows.center
    layer1.cornerRadius = 20
    layer1.borderWidth = 3
    layer1.borderColor = UIColor(red: 0.355, green: 0.717, blue: 0.979, alpha:
1).CGColor
    layer1.backgroundColor = UIColor(red: 0.302, green: 0.635, blue: 0.765, alpha:
1).CGColor
    shadows.layer.addSublayer(layer1)

```

```

let selectImageButtonText = UILabel()
selectImageButtonText.frame = CGRect(x: 0, y: 0, width: 230, height: 60)
selectImageButtonText.textColor = UIColor(red: 1, green: 1, blue: 1, alpha: 1)
selectImageButtonText.font = UIFont(name: "Roboto-Regular", size: 30)
selectImageButtonText.text = "Select image"
selectImageButton.addSubview(selectImageButtonText)
selectImageButtonText.textAlignment = .center
view.addSubview(selectImageButton)
}
func addCloudPercentageLabel(view: UIView) {
    cloudPercentageLabel.frame.size = CGSize(width: view.frame.width, height: 50)
    cloudPercentageLabel.frame.origin = CGPoint(x: 0, y: view.frame.height/2)
    view.addSubview(cloudPercentageLabel)
    cloudPercentageLabel.textAlignment = .center
    cloudPercentageLabel.font = UIFont(name: "AppleSDGothicNeo-Bold", size: 30)
}
@objc func selectImageButtonAction() {
    if UIImagePickerController.isSourceTypeAvailable(.savedPhotosAlbum){
        imagePicker.delegate = self
        imagePicker.sourceType = .savedPhotosAlbum
        imagePicker.allowsEditing = false
        present(imagePicker, animated: true, completion: nil)
    }
}
func imagePickerController(_ picker: UIImagePickerController,
                           didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
    imagePicker.dismiss(animated: true, completion: nil)

    guard let image = info[.originalImage] as? UIImage else {
        fatalError("Expected a dictionary containing an image, but was provided the
following: \(info)")
    }
    self.imageView.image = image
    self.image = image
    self.view.bringSubviewToFront(self.imageView)
    let indicator = Indicator()
    indicator.showIndicator()

    DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) {
        self.cloudPercentageLabel.text = ""
        self.addPhotoAndCalculateCloudPercentage(image: image)
        indicator.hideIndicator()
    }
}

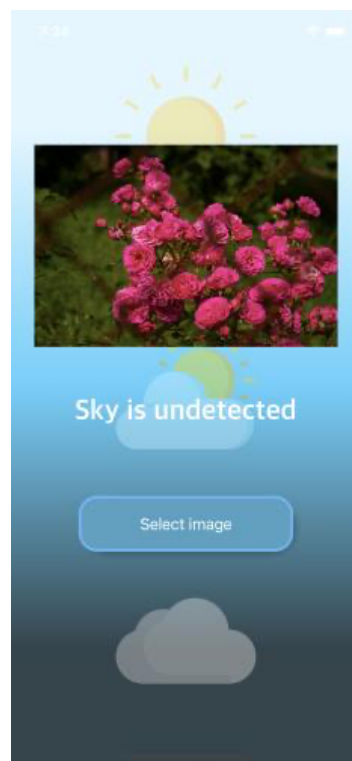
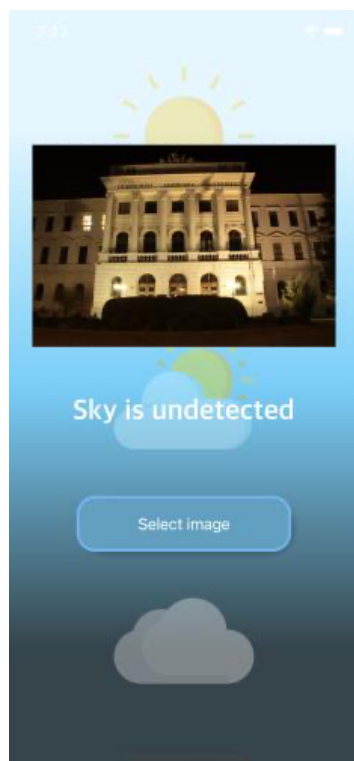
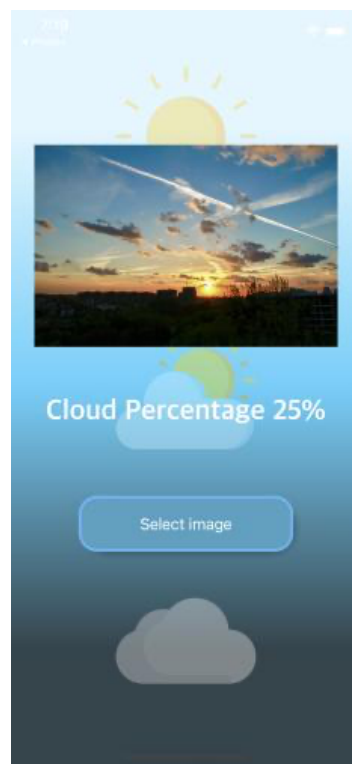
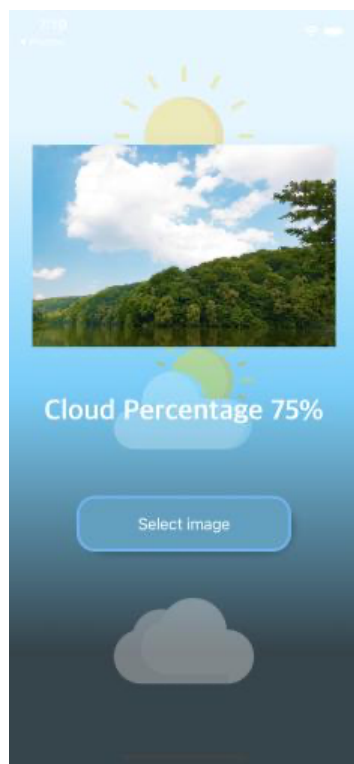
```

```

    }
}
func addPhotoAndCalculateCloudPercentage(image: UIImage) {
    let photoAnalyzer = PhotoAnalyzer(image)
    if let image = photoAnalyzer.createSkyImage() {
        isCurrentImageSky = false
        skyImage = image
        if let cloudPercentage = photoAnalyzer.getCloudPercentage(image: image) {
            if !cloudPercentage.isNaN {
                cloudPercentageLabel.text = "Cloud Percentage " +
cloudPercentage.roundedValue(base: 5)
            } else {
                cloudPercentageLabel.text = "Sky is undetected" }
        } else {
            cloudPercentageLabel.text = "Sky is undetected" }
        } else {
            print("ERROR")
            cloudPercentageLabel.text = "Sky is undetected" } }
func imageViewCreating() -> UIImageView {
    imageView = .init()
    imageView.frame.origin = CGPoint(x: 25, y: 50)
    imageView.frame.size = CGSize(width: self.view.frame.width-50, height:
self.view.frame.height/2)
    imageView.contentMode = .scaleAspectFit
    self.view.bringSubviewToFront(imageView)
    imageView.isUserInteractionEnabled = true
    let gestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(changePhoto))
    imageView.addGestureRecognizer(gestureRecognizer)
    return imageView }
@objc func changePhoto() {
    imageView.image = isCurrentImageSky ? image : skyImage
    isCurrentImageSky.toggle() }
func alert(alertTitle: String, alertMessage: String, alertActionTitle: String) {
    AudioServicesPlaySystemSound(SystemSoundID(4095))
    let alert = UIAlertController(title: alertTitle, message: alertMessage, preferredStyle:
.alert)
    let action = UIAlertAction(title: alertActionTitle, style: .cancel) { (action) in }
    alert.addAction(action)
    self.present(alert, animated: true, completion: nil) }
}

```

Додаток 2. Результати роботи програми



31 травня 2021 року

(дата, місяць, рік)

Тимошук Павло Тарасович

(П.І.П.)

Денна

(Форма навчання)

113 Прикладна математика

(Напрямок)

—

(Спеціальність)

ІМФН

(Інститут)

ЗАЯВА (Декларація)

Усвідомлюючи свою відповідальність за надання неправдивої інформації стверджую, що подана кваліфікаційна робота (дипломний проект, дипломна робота) на тему:

Розроблення мобільного додатку для визначення хмарності неба

Одночасно заявляю, що ця робота:

- не порушує авторських прав відповідно до Закону України «Про авторське право та суміжні права» статей 21–25;
- не використовувалась іншими особами, а також дані та інформація не отримувались в недозволений спосіб;
- не передавалась іншим особам і подається до захисту вперше.

Я усвідомлюю, що у разі порушення цих правил, моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».



(Підпис студента)