Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут прикладної математики та фундаментальних наук

Кафедра прикладної математики

Курсова робота

за дисципліною спеціалізації

на тему:

«Розроблення мобільного додатку для визначення хмарності неба»

Виконав:
студент групи ПМ-41
Тимощук П. Т.

Перевірив:
Топилко П. І.

Львів 2021

Розглянуто задачу розробки мобільного додатку для визначення хмарності неба. Досліджено метод сегментації зображення. Досліджено метод оцінки характеристик пікселя для визначення його належності до хмари. Написано програму, яка реалізовує цю задачу.

Зміст

Вступ

Хмарний покрив (також відомий як хмарність, або кількість хмар) відноситься до частки неба, затемненого хмарами при спостереженні з певного місця. Хмарність відповідає тривалості сонця, оскільки найменш хмарні регіони є найбільш сонячними, а найхмарніші райони - найменш сонячними.

Хмари відіграють кілька 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. Таким чином, хмарний покрив відіграє важливу роль в енергетичному балансі атмосфери, а його зміни є наслідком зміни клімату та очікуваних останніми дослідженнями.

1. Сегментація зображення

З комп'ютерної точки зору, сегментація - це процес розділення цифрового зображення на декількох сегментах (більшість пікселів, які часто називають суперпікселями). Мета сегментації вимагають у спрощеному і / або зміненому зображенні зображення для полегшення його аналізу або передачі каналів зв'язку. Сегментацію зображень можна використовувати для виділення об'єктів та меж на зображеннях. Точніше, сегментація зображення - це процес приєднання таких міток кожному піксельному зображенню, що пікселі з однаковими мітками мають спільні візуальні характеристики.

Результатом сегментації зображення є безліч сегментів, які разом покривають все зображення, або безліч контурів, виділених із зображення. Усі пікселі в сегменті схожих елементів, що характеризують характеристику або визначену владу, такий колір, яскравість або текстуру. Сусідні сегменти істотно відображаються за цими характеристиками.

2. Перетворення моделі сегментованої ділянки у зображення

```
extension UIImage {
  func toJpegData (compressionQuality: CGFloat, hasAlpha: Bool = true, orientation: Int = 6) -> Data? {
      guard cgImage != nil else { return nil }
      let options: NSDictionary =    [
                                  kCGImagePropertyOrientation: orientation,
                                  kCGImagePropertyHasAlpha: hasAlpha,
                                  kCGImageDestinationLossyCompressionQuality: compressionQuality
                                ]
      return toData(options: options, type: .jpeg)
  }

  func toData (options: NSDictionary, type: ImageType) -> Data? {
      guard cgImage != nil else { return nil }
      return toData(options: options, type: type.value)
  }
  // about properties: https://developer.apple.com/documentation/imageio/1464962-cgimagedestinationaddimage
  func toData (options: NSDictionary, type: CFString) -> Data? {
      guard let cgImage = cgImage else { return nil }
      return autoreleasepool { () -> Data? in
          let data = NSMutableData()
          guard let imageDestination = CGImageDestinationCreateWithData(data as CFMutableData, type, 1, nil) else { return nil }
          CGImageDestinationAddImage(imageDestination, cgImage, options)
          CGImageDestinationFinalize(imageDestination)
          return data as Data
      }
  }

  // https://developer.apple.com/documentation/mobilecoreservices/uttype/uti_image_content_types
  enum ImageType {
      case image // abstract image data
      case jpeg                   // JPEG image
      case png                    // PNG image
```

```
        var value: CFString {
            switch self {
            case .image: return kUTTypeImage
            case .jpeg: return kUTTypeJPEG
           case .png: return kUTTypePNG
            }
        }
    }
}
```

3. Використання характеристик пікселя для визначення його належності до хмари

У цій роботі я класифікую окремі пікселі на сегменти неба та хмари на основі кольору пікселя. Діапазон кольорів, присутній у фіксованих зображеннях, обмежений, що полегшує розрахунок оцінки хмарного покриття.

Хмарний покрив оцінив через відсоток хмарного покриву, від 0% до 100%. Піксель класифікується на сегменти неба чи хмари. Для отримання даних для оцінки окремих пікселів на сегменти неба та хмари на основі кольору пікселя я використав 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 небісних 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_{sc}:

$$V_{sc} = -6.28r + 0.454g - 4.11b - 1.81scyl - 4.04scone + 8.88v + 1.53br + 0.586.$$

Якщо $V_{sc} < 0$, то піксель вважається хмарним пікселем.
Обчислюється N_{all} (загальна кількість пікселів неба) та N_{cloud} (кількість пікселів неба).

І нарешті обчислюється $CloudPercentage$ (відсоток хмарності неба)

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

Опис програми

Програма написана на мові програмування Swift у середовищі Xcode. Вона реалізовує задачу розробки мобільного додатку для визначення хмарності неба.

Вхідні дані програми:

Зображення з галереї девайсу

У програмі реалізовано такі основні функції:

1. imagePickerController (отримання зображення з галереї).
2. addPhotoAndCalculateCloudPercentage (перетворення зображення та обчислення відсотку хмарності).
3. calculateVSC (класифікація окремі пікселі на сегменти неба та хмари на основі кольору пікселя).

У результаті виконання програми отримуємо такі вихідні дані: відсоток хмарності неба, у випадку якщо вхідне зображення містить фрагмент неба або повідомлення про те, що вхідне зображення не містить фрагментів неба.

Результат програми виводиться на екран.

Висновки

У даній курсовій розглянуто задачу розробки мобільного додатку для визначення хмарності неба. Досліджено метод сегментації зображення. Досліджено метод оцінки характеристик пікселя для визначення його належності до хмари. Написано програму, яка реалізовує цю задачу.

Список використаної літератури

1. Cloud cover. https://en.wikipedia.org/wiki/Cloud_cover
2. Deep Convolutional Neural Network for Cloud Coverage Estimation from Snapshot Camera Images. https://www.researchgate.net/publication/321984089_Deep_Convolutional_Neural_Network_for_Cloud_Coverage_Estimation_from_Snapshot_Camera_Images
3. Image Segmentation. https://www.fritz.ai/features/image-segmentation.html
4. Image Segmentation iOS. https://docs.fritz.ai/develop/vision/image-segmentation/#ios
5. Сегментація зображення. https://uk.wikipedia.org/wiki/Сегментація_зображення

Додаток

Текст програми

```
//
//  ViewController.swift
//  Cloudy
//
//  Created by Павло Тимощук 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()
    override func viewDidLoad() {
        super.viewDidLoad()
       addBackground(view: self.view)
        addSelectImageButton(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)
        }
        @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.addPhotoAndCalculateCloudPercentage(image: image)
                indicator.hideIndicator()
            }
        }
        func addPhotoAndCalculateCloudPercentage(image: UIImage) {
            let photoAnalyzer = PhotoAnalyzer(image)
            if let image = photoAnalyzer.createSkyImage() {
                isCurrentImageSky = false
                skyImage = image
                let cloudPercentage = photoAnalyzer.getCloudPercentage(image: image)!
                if !cloudPercentage.isNaN {
                    self.alert(alertTitle: "", alertMessage: "Cloud Percentage " + String(cloudPercentage) + "%", alertActionTitle: "OK")
                } else {
                    self.alert(alertTitle: "", alertMessage: "Sky is undetected", alertActionTitle: "OK")
                }
            } else {
                print("ERROR")
            }
        }
        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()
        }
        // MARK: - Make ALERT
        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)
        }
    }

    //
    //  AnalyzingPhoto.swift
    //  Cloudy
    //
    //  Created by Павло Тимощук 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 }
```

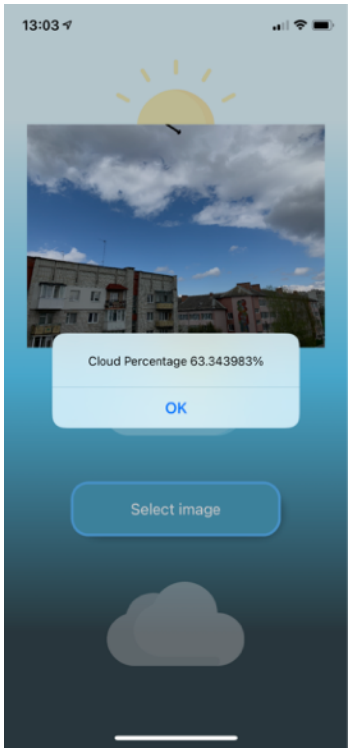
```
            guard let skyImage = createMask(of: self.photo, fromMask: mask) else { return nil }
            return skyImage
        }
        func getCloudPercentage(image: UIImage) -> Float? {
            guard let cgImage = image.cgImage else { return nil }
            let colorSpace = CGColorSpaceCreateDeviceRGB()
            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))
            let pixels = UnsafeMutableBufferPointer<Pixel>(start: imageData, count: width * height)
            var cloudPercentage: Float = 0
            print(width,height)
            var pixelCount = 0
            var cloudPixelCount = 0
            let accuracyValue = 5
            for y in 0..<height {
                for x in 0..<width {
                    let index = y * width + x
                    let pixel = pixels[index]
                    if pixel.alpha != 0 && x%accuracyValue==y%accuracyValue {
                        if calculateVSC(pixel: pixel) < 0 {
                            cloudPixelCount += 1
                        }
                        pixelCount += 1
                    }
                }
            }
            cloudPercentage = Float(cloudPixelCount) / Float(pixelCount) * 100
            print(cloudPercentage,"%")
```

```
            return cloudPercentage
        }
        func calculateVSC(pixel: Pixel) -> Float {
            var vsc: Float = 0
            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)
            vsc = -6.28*r + 0.454*g - 4.11*b - 1.8*scyl + 8.88*v + 1.53*br + 0.586
            return vsc
        }
             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 = CIImage(cgImage: imageCG)
                              let background = background?.cgImage != nil ? CIImage(cgImage:
background!.cgImage!) : CIImage.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 & 0xFFFFFF00)
                }
            }
            public var green: UInt8 {
                get {
                    return UInt8((value >> 8) & 0xFF)
```

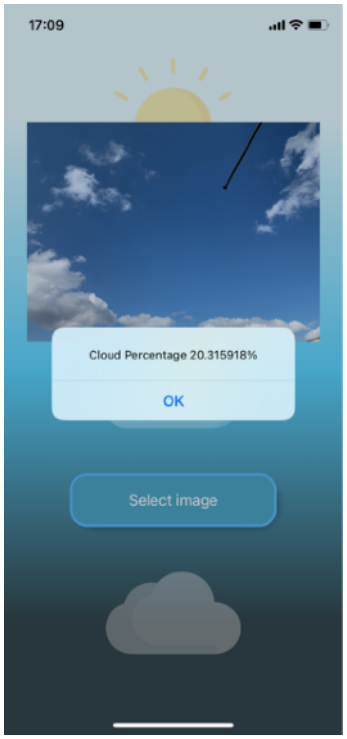
```
            } set {
                value = (UInt32(newValue) << 8) | (value & 0xFFFF00FF)
            }
        }
        public var blue: UInt8 {
            get {
                return UInt8((value >> 16) & 0xFF)
            } set {
                value = (UInt32(newValue) << 16) | (value & 0xFF00FFFF)
            }
        }
        public var alpha: UInt8 {
            get {
                return UInt8((value >> 24) & 0xFF)
            } set {
                value = (UInt32(newValue) << 24) | (value & 0x00FFFFFF)
            }
        }
    }
}
extension UIImage {
    func withAdjustment(bySaturationVal: CGFloat, byContrastVal: CGFloat) -> UIImage {
        guard let cgImage = self.cgImage else { return self }
        guard let filter = CIFilter(name: "CIColorControls") else { return self }
        filter.setValue(CIImage(cgImage: cgImage), forKey: kCIInputImageKey)
        filter.setValue(bySaturationVal, forKey: kCIInputSaturationKey)
        filter.setValue(byContrastVal, forKey: kCIInputContrastKey)
        guard let result = filter.value(forKey: kCIOutputImageKey) as? CIImage else { return self
}
                guard let newCgImage = CIContext(options: nil).createCGImage(result, from:
result.extent) else { return self }
            let image = UIImage(cgImage: newCgImage, scale: UIScreen.main.scale, orientation:
imageOrientation)
        return UIImage(data: image.pngData()!)!
    }
}
```

Результати роботи програми

Приклад 1.



Приклад 2.



Приклад 3.