МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Факультет прикладної математики та інформатики
Кафедра дискретного аналізу та інтелектуальних систем

Дипломна робота

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ РОЗПІЗНАВАННЯ ГРИБІВ ТА СИСТЕМИ ЙОГО АВТОМАТИЧНОГО РОЗГОРТАННЯ

Виконав: студент групи ПМі-45с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)
Смолій Ю. Ю.
(підпис) (прізвище та ініціали)
Керівник Квасниця Г. А.
(підпис) (прізвище та ініціали)
Рецензент
(підпис) (прізвище та ініціали)

ЛЬВІВ – 2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Дискретного аналізу та інтелектуальних систем
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____
«13» вересня 2022 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТА

Смолій Юрія Юрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи

Розробка застосунку для розпізнавання грибів та системи його автоматичного розгортання

керівник роботи Квасниця Галина Андріївна кандидат фіз.-мат. наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від **"13" вересня 2022 року № 15**

2. Строк подання студентом роботи **13.06.2023р.**

3. Вихідні дані для роботи:

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ

- CV – (“Computer Vision”) комп’ютерний зір
- ПЗ – програмне забезпечення
- iOS, iPadOS, macOS, tvOS, watchOS – операційні системи, розроблені компанією Apple та призначені для використання на приладах Apple.
- ЦП – центральний процесор
- ГП – графічний процесор
- MLP – (“Multi-Layered Perceptron”) багатошаровий персептрон
- Автопілот - це програмне забезпечення або інструмент, який може керувати транспортом лише за певних умов за допомогою систем транспортного засобу.
- TrainSet - набір прикладів, що використовуються в залежності від параметрів (наприклад, ваги зв'язків між нейронами в штучних нейронних мережах) моделі.
- TestSet - набір даних, що використовується для забезпечення неупередженої оцінки остаточної відповідності моделі до набору навчальних даних.

SGD – (“Stochastic Gradient Descent, incremental gradient descent”) стохастичний градієнтний спуск

- Парсинг – процес витягування даних з різних ресурсів.
- CNN – (“Convolutional neural network”) згорткова нейронна мережа.
- RELU – (“Rectified Linear Unit”) – зрізаний лінійний вузол.
- CI/CD – (“Continious Integration/Continious Delivery”) – конвеєр автоматичного згортання, тестування та розгортання коду з використанням хмарних технологій.

ВСТУП

Серед любителів активного відпочинку знайдеться чимало тих, хто любить збирати гриби. З якою проблемою зустрічаються недосвідчені грибники, особливо ті, хто вперше вирушив в ліс на « тихе полювання»?

Прийнято вважати, що найчастішою проблемою є невпевненість у виборі їстівного гриба. Змалку нас навчають не брати ті гриби, щодо яких є найменший сумнів. Додаток, розроблений у рамках дипломної роботи, може підказати грибнику, який саме гриб знаходиться перед ним, або ж підтвердити чи спростувати його здогадки. Варто зазначити, що додатком рекомендовано користуватись виключно у цілях розваги, оскільки він не гарантує точного підтвердження класифікації знайдених грибів. Розроблений додаток не вимагає активного підключення до мережі інтернет, що значно підвищує зручність користування, оскільки у лісах України мережа частіше відсутня, а ніж навпаки.

Для визначення назви гриба та його класифікації їстівний чи неїстівний, буде використано натреновану модель штучного інтелекту у форматі, що підтримується та призваний виконувати обчислення на залізі смартфонів під платформою IOS.

Також навколо проєкту буде налаштовано CI/CD процес (автоматиче розгортання) та за допомогою різних інструментів зроблено аналіз відмінностей між ними. Зростання популярності мобільних додатків на платформі iOS та постійна зміна вимог користувачів, створюють потребу у швидкому та ефективному розгортанні додатків на пристроях.Через швидкий розвиток ринку мобільних додатків, традиційний ручний процес розгортання стає обмежуючим і неефективним.

Однак, системи автоматичного розгортання для iOS застосунків з'являються як відповідь на цю проблему. Вони дозволяють розробникам автоматизувати процеси розгортання, тестування, моніторингу та оновлення додатків на пристроях, що працюють під управлінням iOS. Актуальність розробки таких систем полягає в наступному:

- Зменшення часу розгортання: Системи автоматичного розгортання дозволяють значно зменшити час, який необхідний для розгортання додатків на

пристроях. Це особливо важливо при випуску оновлень, або нових функцій, коли швидкість розгортання може мати вирішальне значення для задоволення потреб користувачів та конкурентоспроможності продукту на ринку.

- Покращення якості та стабільності: Автоматичне тестування та моніторинг додатків на пристроях дозволяє виявляти проблеми та помилки швидше і ефективніше. Це допомагає покращити якість та стабільність додатків, запобігаючи появі помилок у роботі.
- Забезпечення безперервної поставки: Системи автоматичного розгортання сприяють впровадженню практики Continuous Deployment/Delivery (CI/CD). Це означає, що нові зміни та оновлення можуть бути швидко та автоматично розгорнуті після успішного проходження тестів та перевірок. Цей підхід дозволяє забезпечити безперервну поставку продукту до користувачів, зменшуючи час і зусилля, необхідні для розгортання.
- Зниження ризику та вартості: Автоматичне розгортання допомагає уникнути помилок, що можуть виникнути при ручному процесі розгортання. Це знижує ризик виникнення збоїв у роботі додатку. Крім того, автоматичні процеси розгортання можуть бути більш ефективними та менш витратними з точки зору ресурсів, оскільки зменшують потребу у ручній праці та можуть бути легко масштабовані.

Враховуючи ці фактори, розробка систем автоматичного розгортання для iOS застосунків стає актуальною задачею.

1 ОПИС ВИКОРИСТАНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ПЗ

1.1 Wolfram Language

Завдяки використанню вбудованого обчислювального інтелекту Wolfram Language надає доступ до обчислювальної потужності на високому рівні. Мова покладається на значну глибину алгоритмів і реальних знань, ретельно інтегрованих протягом трьох десятиліть. Використання Wolfram language дозволило пришвидшити процес збору даних для тренування моделі на прикладі перетворення назв грибів на їх царства.

1.2 Wolfram mathematica

Mathematica забезпечує єдину інтегровану, постійно розширювану систему, яка охоплює технічні обчислення і легко доступна в хмарі через будь-який вебпереглядач, а також на всіх сучасних настільних системах. Mathematica використовується як середовище виконання коду, написаного на мові Wolfram language. Також у середовищі Wolfram mathematica здійснювалось навчання моделі.

1.3 Wolfram neural network

Для імплементації задуманого функціоналу було використано нейронну мережу з репозиторію нейронних мереж “Wolfram Neural Net Repository”, а саме “Wolfram ImageIdentify Net V1”.

Випущена в 2017 році, мережа Wolfram Research була навчена для понад 4000 класів об’єктів. Це частка серверної частини функції ImageIdentify у Wolfram Language 11.1, котра була розроблена для досягнення балансу між точністю класифікації, розміром і швидкістю оцінки.

Внутрішній навчальний набір Wolfram ImageIdentify складається з понад 3 мільйонів навчальних зображень і понад 4000 класів об’єктів (котрі не є загальнодоступні).

1.4 Python та Python пакети

Python - це потужна мова програмування, яка легко вивчається. Вона має ефективні високорівневі структури даних і простий та ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис і динамічне введення Python роблять її ідеальною мовою для написання сценаріїв і швидкої розробки додатків у багатьох областях на більшості платформ.

Інтерпретатор Python і велика стандартна бібліотека є вільно доступною у вихідній, або двійковій формі для всіх основних платформ з веб-сайту Python, і може вільно поширюватися.

Інтерпретатор Python легко розширюється за допомогою нових функцій і типів даних, реалізованих на C або C++ (або інших мовах, які можна викликати з C). Python також підходить як мова розширення для програм, що налаштовуються.

Після навчання нейронної мережі на своїй підбірці даних, для перетворення вихідної моделі було використано мову програмування Python та її пакет “coremltools”. В результаті отримано модель у форматі CoreMl, що використовується у IOS додатках.

1.5 Core ML Framework

Core ML застосовує алгоритм машинного навчання до набору навчальних даних для створення моделі. Після чого розробник використовує модель для здійснення прогнозів на основі нових вхідних даних. Core Ml Framework надає гнучкий функціонал для навчання різних моделей на підготовленому наборі даних. Можна навчити модель класифікувати фотографії, або виявляти певні об’єкти на фотографії, аналізуючи зображення.

Модель після навчання та тестуванна можна інтегрувати у додаток, що буде встановлений на пристрої користувача, після чого додаток без доступу до інтернету зможе аналізувати зображення та виводиди результати. Процес інтеграції моделі зображений на рисунку 1.1.

Рисунок 0.1 – Процес інтеграції моделі

Після того, як модель буде розміщена на пристрої користувача, розробник може використовувати Core ML, щоб перенавчити, або налаштувати модель на пристрої з даними відповідного користувача.

Core ML оптимізує продуктивність на пристрої, використовуючи процесор, графічний процесор і нейронний механізм, мінімізуючи при цьому обсяг пам’яті та енергоспоживання. Виконання моделі, суворо на пристрої користувача усуває будь-яку потребу в мережевому з’єднанні, що допомагає зберегти конфіденційність даних користувача.

Core ML підтримує Vision для аналізу зображень, Natural Language для обробки тексту, Speech для перетворення аудіо в текст і Sound Analysis для визначення звуків у аудіо. Сам Core ML будується на основі низькорівневих примітивів, таких як Accelerate і BNNS, а також шейдерів Metal Performance. Архітектура Core ML фреймворка зображена на рисунку 1.2.

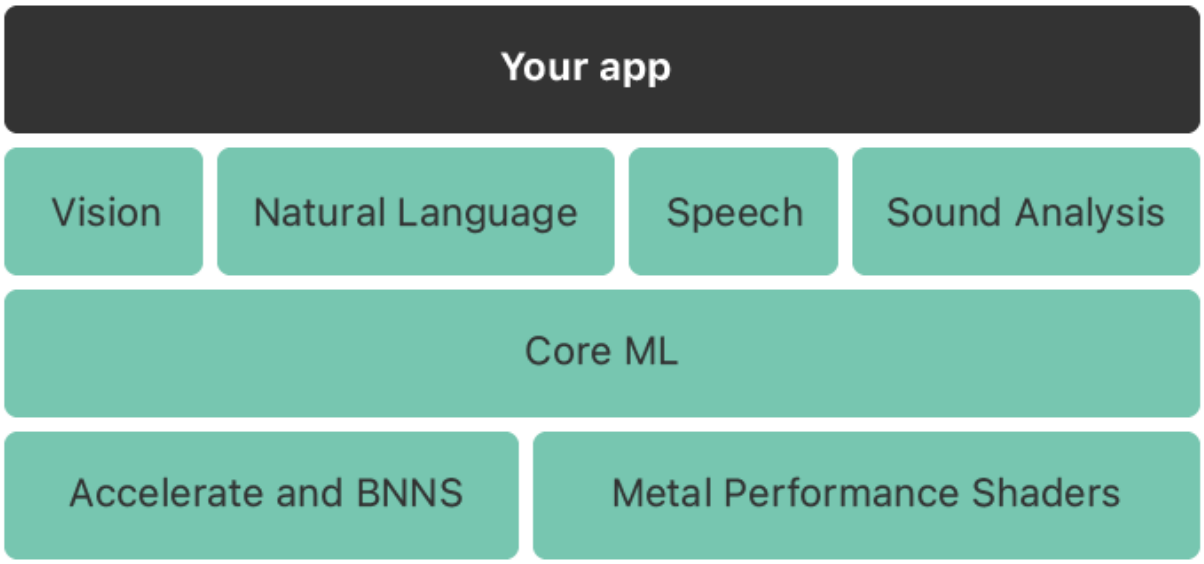


Рисунок 0.2 – Архітектура Core ML фреймворка

1.6 Swift language

Swift - це потужна та інтуїтивно зрозуміла мова програмування для iOS, iPadOS, macOS, tvOS та watchOS платформ. Swift включає сучасні функції та можливості у розробці програмного забезпечення.

З самого початку Swift створювалася, щоб бути швидкою. Використовуючи високопродуктивну технологію компілятора LLVM, код Swift перетворюється в оптимізований рідний код, який отримує максимум користі від сучасного обладнання. Синтаксис і стандартна бібліотека також були налаштовані, щоб зробити найбільш очевидний спосіб написання коду найкращим, незалежно від того, працює він у годиннику на вашому зап’ясті чи на кластері серверів.

Swift є наступницею мов C і Objective-C. Вона включає примітиви низького рівня, такі як типи керування потоками та оператори. Вона також надає об’єктно-орієнтовані функції, такі як класи, протоколи та генерики, надаючи розробникам Cocoa та Cocoa Touch необхідну продуктивність та потужність.

Також варто згадати що Swift розробляється відкрито на Swift.org, з вихідним кодом, системою відстеження помилок, форумами та регулярними збірками, доступними для всіх. Ця широка спільнота розробників, як всередині Apple, так і сотні зовнішніх учасників, працюють разом, щоб зробити Swift ще кращою.

1.7 Microsoft Visual Studio Code та Apple Xcode

Середовища розробки програмного забезпечення, що були використані під час роботи над Python кодом (Visual Studio Code) та Swift проєктом (Xcode).

Visual Studio Code - це легкий, але потужний редактор вихідного коду, який працює на вашому пристрої та доступний для Windows, macOS та Linux. Він поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов (наприклад, C++, C#, Java, Python, PHP, Go) і середовищ виконання (наприклад, .NET і Unity).).

Xcode - це інтегроване середовище розробки (IDE) Apple для macOS, яке використовується для розробки програмного забезпечення для macOS, iOS, iPadOS, watchOS і tvOS. Він був випущений наприкінці 2003 року; останньою стабільною версією є версія 14.3. Xcode включає інструменти командного рядка (CLT), які дозволяють розробку в стилі UNIX за допомогою програми Terminal в macOS. Їх також можна завантажити та встановити без графічного інтерфейсу.

1.8 iNaturalist

iNaturalist - спільна ініціатива Каліфорнійської академії наук і Національного географічного товариства. iNaturalist зберігає та ділиться даними про усі внесені та перевірені науковцями спостереження.

Кожне спостереження може зробити будь-який зареєстрований користувач і тим самим зробити внесок у науку про біорізноманіття, від найрідкіснішого метелика до найпоширенішого бур'яну. У роботі використано ресурс iNaturals як сховище потрібних даних для збору та навчання моделі нейронної мережі.

1.9 GitHub Actions

GitHub Actions - це платформа для імплементації CI/CD процесів, що дозволяє користувачу автоматизовувати збірки, тести та конвеєри розгортання. Застосунок Actions виділяє для користувача віртуальну машину з встановленою операційною системою на вибір, також є можливість під'єднати персональний комп'ютер і використовувати його у ролі віртуальної машини. Функціонал GitHub Actions дуже гнучкий: можна налаштувати різні тригери для запуску кожного конвеєра, налаштувати виділену обчислювальну машину згідно власних потреб, зберігати вибрані файли у хмарному сховищі і користуватись віртуальною машиною, як повноцінним комп'ютером, використовуючи інтерфейс командного рядка.

1.10 Xcode Cloud

Xcode Cloud – це інтегрований у середовище розробки "Xcode" інструмент автоматичної інтеграції та розгортання застосунків і програм, котрий дозволяє налаштувати автоматичне тестування, надсилання збірки для розробників та збір відгуків від користувачів бета версій. Xcode Cloud також має механізм зручного керування версійністю над проєктом.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Нейронні мережі

Нейронна мережа — це серія алгоритмів, які намагаються розпізнати основні зв’язки в наборі даних за допомогою процесу, який імітує роботу мозку людини. У цьому сенсі нейронні мережі належать до систем нейронів, як органічних, так і штучних за своєю природою.

Нейронні мережі – це потужна техніка машинного навчання, яка дозволяє створювати модульну композицію операцій (шарів), які можуть моделювати різноманітні функції з високою продуктивністю виконання та навчання. Нейронні мережі, як правило, стійкі до шумового введення і пропонують хороші можливості узагальнення. Вони є центральним компонентом у багатьох областях, як-от обробка зображень та аудіо, обробка природної мови, робототехніка, керування автомобілем, медичні системи тощо. Архітектура найпростішої нейронної мережі зображена на рисунку 2.1.

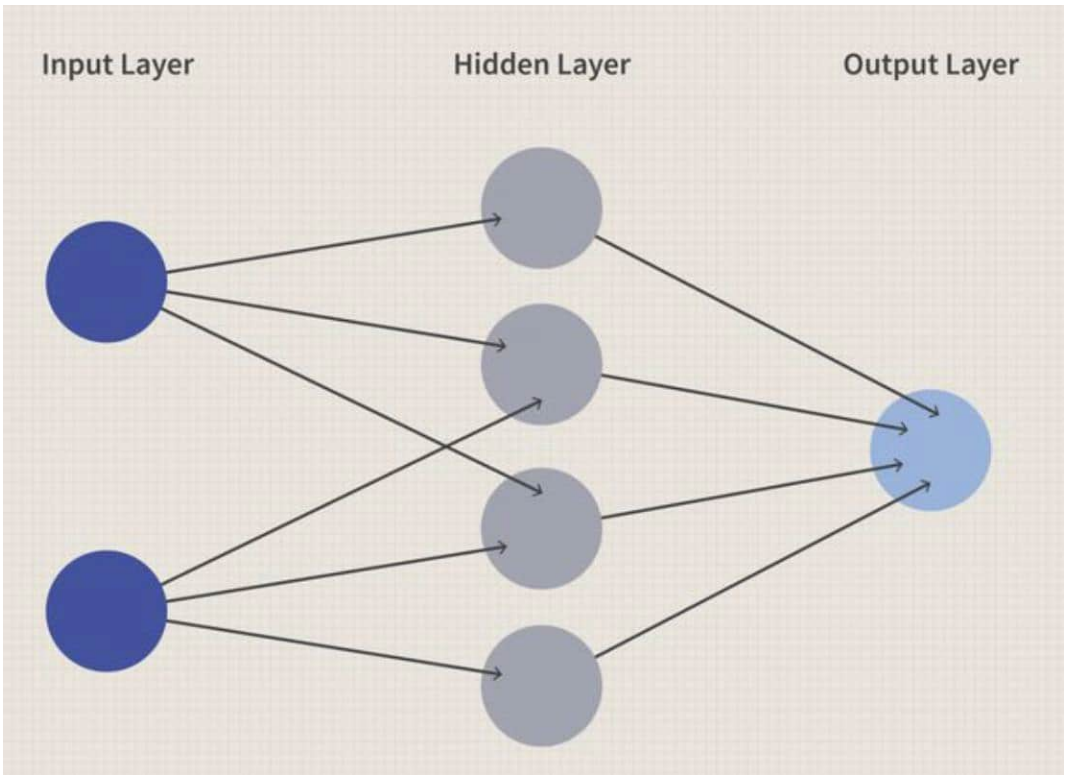


Рисунок 0.1 – Архітектура найпростішої нейронної мережі

Wolfram Language пропонує розширені можливості для представлення, побудови, навчання та розгортання нейронних мереж. Для символічної композиції та маніпуляцій доступна велика різноманітність типів шарів. Завдяки спеціальним кодерам і декодерам, різноманітні типи даних, такі як зображення, текст та аудіо, можуть використовуватися як вхідні та вихідні дані, поглиблюючи інтеграцію з мовою Wolfram Language.

2.2 Згорткова нейронна мережа

Згорткова нейронна мережа містить тривимірне розташування нейронів замість стандартного двовимірного масиву. Перший шар називається згортковим шаром. Кожен нейрон у згортковому шарі обробляє лише інформацію з невеликої частини поля зору. Вхідні функції беруться пакетно, як фільтр. Мережа розуміє зображення по частинах і може обчислювати ці операції кілька разів, щоб завершити повну обробку зображення. Обробка включає перетворення зображення з шкали RGB, або HSI в шкалу сірого. Подальші зміни значення пікселів допоможуть виявити краї і зображення можна класифікувати за різними категоріями.

Поширення є однонаправленим, де CNN містить один або кілька згорткових шарів з подальшим об’єднанням, і двонаправленим, де вихід шару згортки надходить до повністю підключеної нейронної мережі для класифікації зображень, як показано на діаграмі вище. Фільтри використовуються для вилучення певних частин зображення. У MLP вхідні дані помножуються на вагові показники і надходять до функції активації. Згорткова мережа використовує RELU, а MLP використовує нелінійну функцію активації, а потім softmax. Нейронні згортки показують дуже ефективні результати в розпізнаванні зображень і відео, семантичному розборі.

2.3 Нейрон з погляду штучної нейронної мережі

Нейронна мережа працює подібно до нейронної мережі людського мозку. «Нейрон» у нейронній мережі — це математична функція, яка збирає та класифікує інформацію відповідно до певної архітектури.

2.4 Вузли нейронної мережі

Нейронна мережа містить шари взаємопов'язаних вузлів. Кожен вузол відомий як персептрон і подібний до множинної лінійної регресії. Персептрон подає сигнал, вироблений множинною лінійною регресією, у функцію активації, яка може бути нелінійною.

2.5 Персептрон

У сфері машинного навчання персептрон є алгоритмом контрольованого навчання для бінарної класифікації. Він також часто відомий як перцептрон. Бінарний класифікатор є функцією, яка вирішує, чи належить вхідний вектор чисел до певного класу. Це лінійний класифікатор, що базується на функції лінійного предиктора, що поєднує вагові коефіцієнти з вектором ознак.

Персептрон є математичною, або комп'ютерною моделлю сприйняття інформації мозком, яка була запропонована Френком Розенблатом у 1957 році і реалізована у вигляді електронної машини "Марк-1" в 1960 році. Він став однією з перших моделей нейромереж, а "Марк-1" був першим нейрокомп'ютером у світі.

2.6 Багатошаровий персептрон

У багатошаровому персептроні (MLP) персептрони розташовані у взаємопов’язаних шарах. Вхідний шар збирає вхідні шаблони. Вихідний рівень має

класифікації, або вихідні сигнали на які можуть зображатися вхідні шаблони. Наприклад, шаблони можуть містити перелік величин для технічних індикаторів цінного паперу одного з біржових ринків. Потенційні результати можуть бути «купити», «утримати» або «продати».

Приховані шари точно налаштовують вхідні зважування, поки похибка нейронної мережі не стане мінімальною. Існує гіпотеза, що приховані шари екстраполюють помітні ознаки у вхідних даних, які мають передбачувану силу щодо вихідних даних. Це описує вилучення ознак, яке реалізує корисність, подібну до статистичних методів, таких як аналіз головних компонентів.

2.7 Навчання нейронної мережі

Усі нейрони даного шару генерують вихід, але вони не мають однакової ваги для наступного шару нейронів. Це означає, що якщо нейрон на шарі спостерігає заданий шаблон, це може означати менше для загальної картини й буде частково, або повністю приглушено. Це те, що ми називаємо зважуванням: велика вага означає, що вхід важливий, мала вага означає, що ми повинні його ігнорувати. Кожен нейронний зв’язок між нейронами матиме відповідну вагу.

В цьому полягає магія адаптивності нейронної мережі: ваги будуть коригуватися протягом тренування, щоб відповідати цілям, які ми поставили (визнати, що собака — це собака, а кіт — це кіт). Простіше кажучи: навчання нейронної мережі означає знаходження відповідної ваги нейронних з’єднань завдяки циклу зворотного зв’язку, який називається градієнтним розповсюдженням назад.

2.8 Градієнтний спуск

Градієнтний спуск - це метод оптимізації першого порядку, який використовує ітерації для знаходження локального мінімуму функції. У цьому методі кроки, які ми робимо, пропорційні протилежному значенню градієнту (або наближеного градієнту)

функції у поточній точці. Якщо замість цього ми робимо кроки, пропорційні самому значенню градієнту, то ми будемо наближатися до локального максимуму цієї функції. Цей процес відомий як градієнтний підйом.

Градієнтний спуск ґрунтується на спостереженні. Якщо функція кількох змінних F(x) є визначеною та диференційовною в околі точки a, то F(x) зменшується найшвидше, якщо йти від a в напрямку, протилежному градієнтові F в a, - ÑF(a). З цього випливає, що якщо

$$b = a - y\tilde{N}F(a) \tag{2.1}$$

для достатньо малого y, то F(a) >= F (b). Іншими словами, член yÑF(a) віднімається від a, оскільки ми хочемо рухатися проти градієнту, тобто вниз до мінімуму. Враховуючи це спостереження, починають з припущення x0 про локальний мінімум F, і розглядають таку послідовність x0, x1, x2,…, що

$$x_{n+1} = x_n - y_n\tilde{N}F(x_n),\ n >= 0 \tag{2.2}$$

Ми маємо

$$F(x0) >= F(x1) >= F(x2) >= \ldots, \tag{2.3}$$

і тому сподіваємося, що послідовність (x_n) збігається до бажаного локального мінімуму. Варто відзначити, що значення кроку, позначеного як y, може змінюватися на кожній ітерації. За певних припущень про функцію F та варіантів вибору кроку y, збіжність до локального мінімуму може бути гарантовано. Якщо функція F є опуклою, то всі локальні мінімуми є також глобальними мінімумами. Тому в такому випадку градієнтний спуск може збігатися до глобального розв'язку.

Зокрема, якщо F є опуклою функцією, то всі локальні мінімуми також є глобальними мінімумами, що робить можливим збіжність градієнтного спуску до глобального розв'язку.

Цей алгоритм корисний у випадках, коли оптимальні точки неможливо знайти, прирівнявши нахил функції до 0.

Як алгоритм рухається вниз по сходах?

Це суть алгоритму. Загальна ідея полягає в тому, щоб почати з випадкової точки і знайти спосіб оновлювати цю точку з кожною ітерацією, щоб ми спускалися вниз по схилу.

Етапи алгоритму такі

- Знайдіть нахил цільової функції щодо кожного параметра/області. Іншими словами, обчислити градієнт функції.
- Виберіть випадкове початкове значення для параметрів. (Щоб пояснити, у прикладі з параболою розрізняйте «y» відносно «x». Якщо б у нас було більше функцій, таких як x1, x2 тощо, ми беремо часткову похідну від «y» щодо кожної з ознак.)
- Оновіть функцію градієнта, додавши значення параметрів.
- Розрахуйте розміри кроків для кожної функції як: розмір кроку = градієнт * швидкість навчання.
- Обчисліть нові параметри як: нові параметри = старі параметри -розмір кроку
- Повторюйте кроки з 3 по 5, поки градієнт не стане майже 0.

«Швидкість навчання» є гнучким параметром, який сильно впливає на збіжність алгоритму. Більша швидкість навчання змушує алгоритм робити величезні кроки вниз по схилу і він може перестрибнути через мінімальну точку, тим самим пропустивши її. Тому завжди добре дотримуватися низької швидкості навчання, наприклад 0,01. Також можна математично показати, що алгоритм градієнтного спуску робить більші кроки вниз по схилу, якщо початкова точка знаходиться високо і робить маленькі кроки, коли наближається до місця призначення.

2.9 Метод стохастичного градієнта

Стохастичний градієнтний спуск — ітеративний метод оптимізації градієнтного спуску за допомогою стохастичного наближення. Цей метод також використовується для прискорення пошуку цільової функції. Він досягає цього шляхом використання обмеженого розміром тренувального набору, який випадковим чином вибирається при кожній ітерації.

Коли на кожній ітерації алгоритму вибирається лише один об'єкт з навчальної вибірки випадковим чином, то цей метод називається стохастичним (stochastic). Таким чином вектор *w* налаштовується кожен раз на новобраний об'єкт.

2.10 Застосування нейронної мережі

Нейронні мережі широко використовуються для фінансових операцій, корпоративного планування, ідентифікації об’єктів на зображеннях, торгівлі, бізнес-аналітики. Також їх використовують у побудові приладів автоматичної фіксації дорожніх штрафів, різного класу та типу автопілотів та обслуговування продуктів. Нейронні мережі отримали широке поширення в бізнес-додатках для прогнозування та маркетингових досліджень, виявлення шахрайства та оцінка ризиків.

Нейронна мережа оцінює дані про ціни та виявляє можливості для прийняття торгових рішень на основі аналізу даних. Мережі можуть розрізняти тонкі нелінійні взаємозалежності та закономірності, які інші методи технічного аналізу не можуть.

Завжди будуть набори даних і класи завдань, які краще аналізувати за допомогою попередньо розроблених алгоритмів. Важливий не стільки алгоритм, як добре підготовлені вхідні дані щодо цільового показника, котрі в кінцевому підсумку визначають рівень успішності нейронної мережі.

2.11 Нейронні мережі на мові Wolfram

Wolfram Language має найсучасніші можливості для побудови, навчання та розгортання систем машинного навчання нейронних мереж. Доступно багато стандартних типів шарів, які символічно збираються в мережу, яку потім можна негайно навчити та розгорнути на доступних ЦП і графічних процесорах.

2.12 Класифікація зображень за допомогою нейронних мереж

Розпізнавання зображень (класифікація зображень) — це завдання ідентифікації зображень і їх класифікації в один із кількох, попередньо визначених окремих класів. Тому програмне забезпечення та програми для розпізнавання зображень можуть ідентифікувати те, що саме зображено і відрізнити один об’єкт від іншого.

Область досліджень, спрямованих на створення машин з цією здатністю, називається комп’ютерним зором. Одне із завдань CV (комп’ютерного зору)- класифікація зображень, що є основою для вирішення різноманітних завдань.

2.13 Standalone архітектура

Окрема програма – це програма, яка працює локально на пристрої та не вимагає нічого іншого для роботи. Вся логіка вбудована в додаток, тому йому не потрібно підключення до Інтернету чи будь-які інші служби.

На відміну від веб-програм, які працюють у браузері і не потребують встановлення, автономні програми потребують абсолютно протилежного. Для запуску їм не потрібен браузер, але часто вимагають встановлення пристрою. Однак існують також портативні автономні програми, створені для деяких платформ (наприклад, Windows), які можна запускати лише подвійним клацанням окремого файлу без встановлення. Дану програму розроблено на базі standalone архітектури,

що значно підвищує зручність користуванням додатком там, де його використання буде найчастішим, а саме лісах, парках та полях.

2.14 Поняття автоматичного розгортання

Автоматичне розгортання є процесом, що передбачає автоматизацію розгортання програмного забезпечення на пристроях, що працюють під управлінням операційної системи iOS. Це означає, що замість ручного розгортання та установки додатків та сервісів, використовуються спеціальні інструменти та процеси, які автоматизують ці дії. У даному випадку автоматизовано надсилання зібраного IOS застосунку у TestFlight, використовуючи GitHub Actions у комбінації з Fastline та інший інструмент часткової оптимізації - Xcode Cloud.

2.15 Огляд існуючих систем автоматичного розгортання

На сьогоднішній день існує кілька систем, які надають можливість автоматичного розгортання iOS застосунків та сервісів. Кожна з цих систем має свої особливості, інструменти та функціонал, що дозволяють забезпечити автоматизацію процесів розгортання та управління додатками на пристроях, що працюють під управлінням iOS. Кожна з цих систем має свої переваги та обмеження. Вибір конкретної системи залежить від потреб та вимог проекту, розміру команди розробників, складності додатка та багатьох інших факторів. Важливо враховувати особливості кожної системи та їх сумісність з іншими інструментами, що використовуються в процесі розробки. Розглянемо кілька з них:

2.15.1 Xcode та Testflight

Xcode є офіційним інтегрованим середовищем розробки (IDE) для платформи iOS. Він має вбудований інструмент TestFlight, що дозволяє розробникам створювати бета-версії своїх додатків та надсилати їх для тестування спільноті тестерів перед остаточним розгортанням. Xcode забезпечує зручність та простоту у розгортанні

додатків на пристроях, а TestFlight дозволяє контролювати та керувати тестуванням додатків.

2.15.2 Fastline

Fastlane є інструментом автоматизації процесів розробки та розгортання додатків для платформи iOS. Він надає широкий спектр функцій, включаючи автоматичне збирання та завантаження додатків на пристрої, автоматизоване тестування, генерацію скріншотів та управління сертифікатами та профілями розробника. Fastline інтегрується з різними інструментами розробки, такими як Xcode, і надає зручний інтерфейс командного рядка для автоматизації рутинних завдань.

2.15.3 Fabric

Fabric, раніше відомий як Crashlytics, є платформою для розгортання, моніторингу та аналізу додатків на платформі iOS. Вона надає інструменти для автоматичного розгортання оновлень додатків, моніторингу додатків під час роботи та збору аналітичних даних про використання додатків. Fabric також має можливості відстеження та аналізу збоїв та помилок додатків, що допомагає розробникам виявляти та виправляти проблеми швидко та ефективно.

2.15.4 Bitrise

Bitrise є хмарною платформою для автоматизації розробки та розгортання додатків на платформі iOS. Вона надає інтегроване середовище для автоматичного збирання, тестування та розгортання додатків, а також для неперервної поставки. Bitrise має широкий спектр інтеграцій з іншими інструментами розробки, такими як GitHub, Slack, JIRA, що сприяє зручності та ефективності розробки продукту.

2.16 Переваги та недоліки існуючих систем

Існуючі системи автоматичного розгортання iOS застосунків та сервісів мають свої переваги та недоліки. З одного боку, вони забезпечують автоматизацію розгортання та спрощують процес розробки. Вони також дозволяють швидше впроваджувати зміни та оновлення, що поліпшує ефективність роботи над продуктом.

З іншого боку, деякі системи можуть мати обмеження щодо функціональності, несумісності з деякими інструментами, або вимагати складнішого налаштування.

Загальною перевагою систем автоматичного розгортання є покращення CI/CD процесів. CI (Continuous Integration) означає безперервну інтеграцію, що передбачає автоматичне об'єднання та перевірку коду з декількох джерел у спільну версію. CD (Continuous Deployment/Delivery) включає безперервну автоматичну поставку/доставку після успішного проходження тестів та перевірок. Ці процеси допомагають зменшити час між оновленнями, забезпечують стабільність та якість розроблюваного продукту.

Системи автоматичного розгортання iOS застосунків та сервісів є важливим інструментом для ефективної розробки та управління продуктом. Вони спрощують процес розгортання та дозволяють швидко впроваджувати зміни та оновлення. Крім того, вони покращують CI/CD процеси, що допомагає забезпечити стабільність та якість розроблюваного продукту.

3 СТВОРЕННЯ ЗАСТОСУНКУ

3.1 Парсинг даних

Для збору даних використано ресурс iNaturalist та мову програмування Wolfram language,що запускалась у середовищі Wolfram mathematica. За допомогою стандартного методу “ResourceFunction” було під’єднано функцію "INaturalistSearch" з офіційного репозиторію функцій мови Wolfram “Wolfram Function Repository”. Це необхідно для парсингу даних з ресурсу iNaturalist. Підключення функції парсингу зображено на рисунку 3.1.

Рисунок 0.1 – Підключення функції парсингу

Після цього створено два списки з назвами токсичних грибів та їстівних\напівїстівних. За допомогою вбудованого у мову розробки інтелекту інтерпретовано назви грибів у назви сімейств, до яких вони відносяться. Опісля видалено дублікати назв сімейств, використавши стандартний метод “DeleteDublicates”. Таким чином отримано два списки назв сімейств їстівних на токсичних грибів. На рисунку 3.2 продемонстровано списки назв грибів.

```
toxicMushrooms = Interpreter["Species"][
  {"Amanita phalloides", "Amanita muscaria", "Amanita pantherina",
   "Gyromitra gigas", "Galerina marginata", "Paxillus involutus",
   "Boletus satanas", "Entoloma rhodopolium", "Inocybe", "Boletus calopus" ,
   "Porphyrellus pseudoscaber" , "Hebeloma" , "Hebeloma crustuliniforme" ,
   "Helvella" , "Helvella monachella" , "Hygrocybe conica" ,
   "Tylopilus felleus" , "Lycoperdon umbrinum" , "Geastrum fornicatum" ,
   "Caloscypha fulgens", "Hygrophoropsis aurantiaca" , "Lyophyllum favrei" ,
   "Amanita citrina" , "Hypholoma lateritium" , "Chalciporus piperatus" ,
   "Psilocybe semilanceata" , "Ramaria stricta" , "Ramaria fennica" ,
   "Rhizina undulata" , "Tricholoma imbricatum" , "Sarcoscypha austriaca" ,
   "Russula emetica" , "Russula atrorubens" , "Stereum hirsutum" ,
   "Stropharia aeruginosa" , "Coltricia perennis" , "Coltricia cinnamomea" ,
   "Trametes versicolor" , "Ganoderma lucidum" , "Lactarius blennius" }]

edibleMushrooms =
 Interpreter["Species"][{"Boletus edulis", "Boletus aereus",
   "Suillus granulatus", "Lycoperdon perlatum", "Lactarius deliciosus",
   "Amanita caesarea", "Macrolepiota procera", "Russula delica",
   "Cantharellus aurora", "Cantharellus cibarius", "Chroogomphus rutilus",
   "Coprinus comatus", "Morchella esculenta", "Hydnum repandum",
   "Suillus luteus", "Marasmius oreades", "Agaricus", "Boletus scaber",
   "Boletus aurantiacus", "Pleurotus ostreatus", "Calvatia gigantea",
   "Lactarius deliciosus", "Tricholoma flavovirens", "Russula virescens",
   "Russula cyanoxantha", "Laetiporus sulphureus", "Boletus edulis",
   "Russula foetens", "Lactarius torminosus", "Coprinopsis atramentaria",
   "Suillellus luridus", "Morchella esculenta", "Amanita rubescens",
   "Armillaria mellea", "Lepista nuda", "Lactarius rufus",
   "Lactarius turpis", "Lactarius piperatus", "Lactarius vellereus",
   "Flammulina velutipes", "Agaricus bisporus", "Pleurotus ostreatus",
   "Auricularia auricula-judae", "Volvariella volvacea",
   "Hypsizygus tessulatus", "Stropharia rugosoannulata"}]

edibleMushrooms = DeleteDuplicates[edibleMushrooms]

toxicMushrooms = DeleteDuplicates[toxicMushrooms]
```

Рисунок 0.2 – Списки назв грибів

Написано функцію для отримання даних про гриби конкретного сімейства. Для цього використано імпортовану функцію, котрій передано потрібні параметри: отримувати дані лише про записи, що містять фотографію гриба, зроблену у регіоні «Європа». Максимальна та бажана кількість фото для парсингу була встановлена 200. Функція парсингу повертає текст формату json, що у своїй структурі містить посилання на квадратні фотографії розміром 224*224 пікселя, що будуть використовуватись у подальших кроках для навчання моделі. Функція парсингу грибів зображена на рисунку 3.3.

```
imageURLs[species_] :=
 Normal[
   Dataset[ToAssociations@
     Dataset[ToAssociations@ INaturalistSearch + [ species , "RawData",
          "HasImage" → True, "QualityGrade" → "Research", MaxItems → 200,
          "ObservationGeoRange" → GeoBounds[ Europe GEOGRAPHIC REGION ]]][[
       "results"][All, "photos"]]][All, 1][All, 4]
```

Рисунок 0.3 – Функція парсингу зображень грибів

Для зручності подальшого використання зображень були написані допоміжні функції, що служать для коректного визначення назви файлу з зображенням грибів у форматі “Повна-назва-їстівність-номер”, та створення словника з ключем – назвою гриба та значенням – його зображенням. На рисунках 3.4 та 3.5 зображена функція експорту зображень та ілюстрація назв експортованих зображень відповідно

```
imagesExport[species_, tag_String] :=
 MapIndexed[
  Export[imagesDirectory <> StringReplace[species["ScientificName"], " " → "-"] <>
     "-" <> tag <> "-" <> ToString[First[#2]] <> ".png", #1] &,
  Map[squareImage, imageURLs[species]]]
```

Рисунок 0.4 – Функція експорту зображень

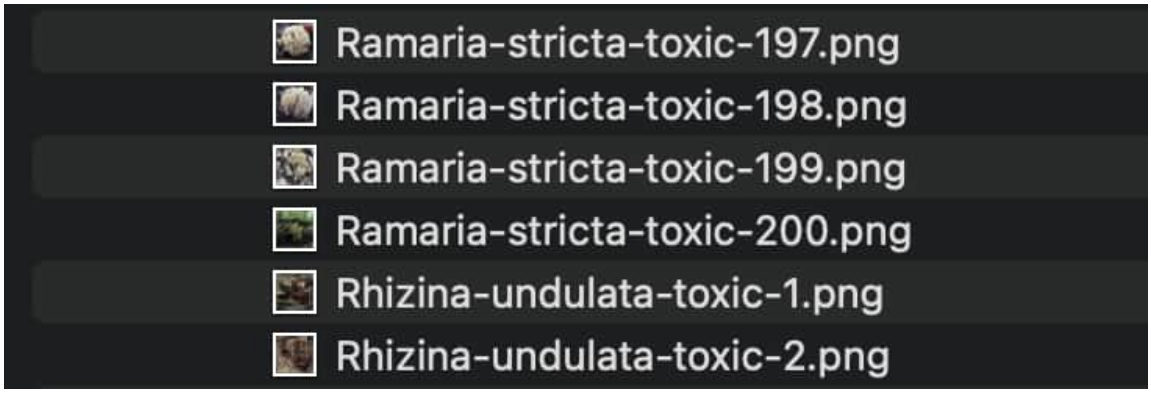


Рисунок 0.5 – Ілюстрація назв експортованих зображень

Після створення словника та збереження зображень, їх потрібно імпортувати в систему Wolfram mathematica для подальшого тренування моделі на зібраних даних.

Для цього було створено додатковий список назв грибів у форматі “Повна назва-їстівність”. При імпортуванні фотографій вдалося досягнути мінімальної втрати завантажених з ресурсу iNaturlist зображень грибів. З цією метою було створено кілька циклів з різними визначеними межами для ітератора та імпортування зображень у середовище. Функції імпортування зображень продемонстровані на рисунку 3.6.

```
all200 = Flatten@Map[Thread[Table[Import[imagesDirectory <> # <> "-" <> ToString[i] <> ".png"], {i, 200}] → #] &,
    classLabels200];

all1 = Flatten@Map[Thread[Table[Import[imagesDirectory <> # <> "-" <> ToString[i] <> ".png"], {i, 156}] → #] &,
    {"Inocybe-toxic"}];

all2 = Flatten@Map[Thread[Table[Import[imagesDirectory <> # <> "-" <> ToString[i] <> ".png"], {i, 114}] → #] &,
    {"Boletus-calopus-toxic"}];

all3 = Flatten@Map[Thread[Table[Import[imagesDirectory <> # <> "-" <> ToString[i] <> ".png"], {i, 99}] → #] &,
    {"Hebeloma-toxic"}];
```

Рисунок 0.6 – Функції імпортування зображень

Так як для функції навчання моделі першим і єдиним параметром передається набір усіх даних, то усі попередньо створені словники були конкатеновані в один, котрий містить усі зібрані дані. Зконкатенований словник даних налічує 9003 пар зображень та їх класифікацій.

3.2 Навчання претренованої моделі

Перш за все для навчання моделі потрібно було створити TrainSet та TestSet. Для створення цих двох окремих словників було використано стандартну функцію Wolfram language “TakeDrop” котра повертає два набори даних. Для першого (TrainSet) взято перші 7000 наборів даних у псевдовипадковому порядку, а для другого (“TestSet”) решта 2008 зображень та їх класифікацій.

Для навчання використано претреновану модель нейронної мережі для класифікацій об’єктів на зображеннях з офіційного репозиторію “Wolfram Neural Net Repository” “Wolfram ImageIdentify Net V1”, що містить у собі 24 шари та є згортковою нейронною мережею. За допомогою функції “NetModel” відбувалось

імпортування моделі в середовище розробки. Після цього, використовуючи стандартну функцію “NetTake”, отримано лише конкретні рівні цієї моделі, починаючи від згорткового шару закінчуючи єднальним (pooling layer).

Для створення моделі на базі імпортованої, було приєднано мережу, що складається з визначеної розмірності ланцюга лінійних шарів з конкретними назвами та визначеним результатом. Операції над моделями продемонстровані на рисунку 3.7.

Рисунок 0.7 – Операції над моделями

Після усіх підготовчих кроків розпочато тренування з’єднаної моделі. Для цього використано функцію “NetTrain”, передаючи їй “TrainSet” словник та задаючи максимальну кількість раундів тренування 50. Метод тренування моделі “SGD” стохастичний градієнтний спуск.

«Стохастичний» означає «випадковий». Під час вибору точок на кожному кроці для обчислення похідних, SGD випадково вибирає одну точку з усього набору даних на кожній ітерації, щоб значно скоротити обчислення.

Зазвичай на кожному кроці відбирають невелику кількість точок даних замість однієї точки. Це називається градієнтним спуском «міні-пакет». Міні-пакет намагається знайти баланс між гідністю градієнтного спуску та швидкістю SGD. На рисунку 3.8 зображено тренування моделі.

```
In[•]:= trainedNet = NetTrain[joinedNet, trainSet, All, LearningRateMultipliers → {"linear10" → 1, _ → 0},
        ValidationSet → Scaled[0.1], MaxTrainingRounds → 50, "SGD"]
```

Рисунок 0.8 – Тренування моделі

Отримую наступний звіт тренування (рисунок 3.9).

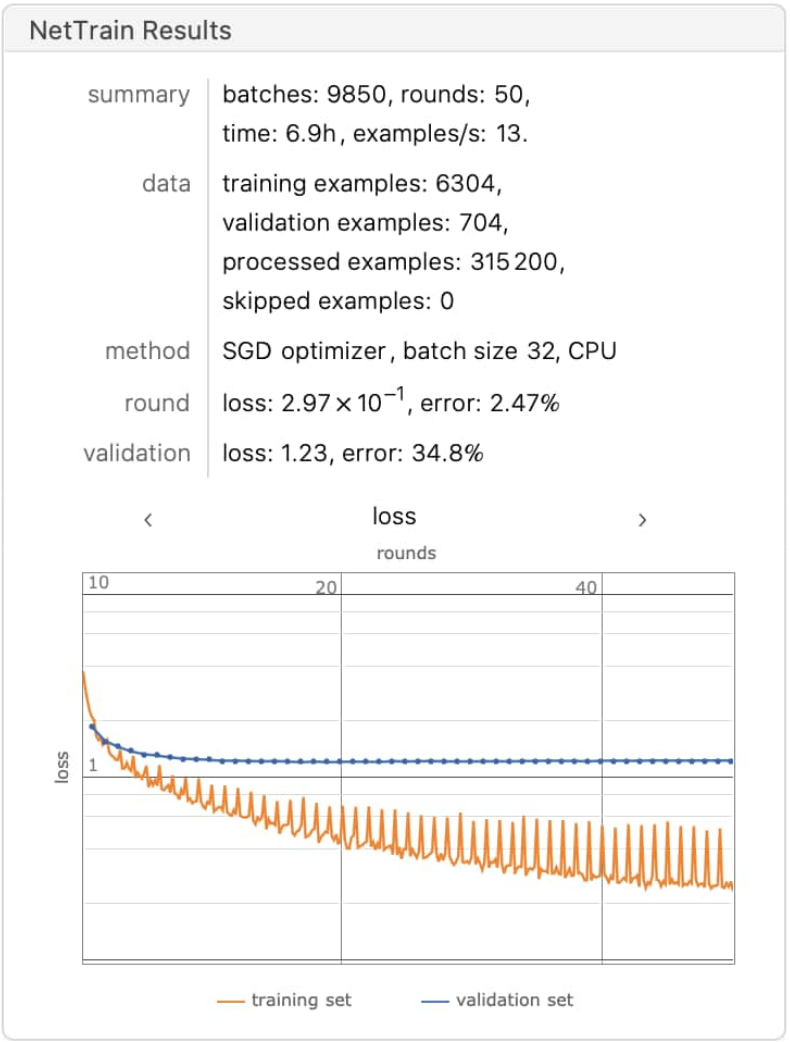


Рисунок 0.9 – Результати тренування моделі

З рафіка видно, як з кожним раундом покращувались результати. Точність передбачень складає близько 63.2%.

Для додаткового аналізу проведено тестування моделі за допомогою попередньо створеного набору даних “TestSet”. Для виведення результатів використано готову функцію, вказавши бажане відображення результуючих даних. Результат тестування зображений на рисунку 3.10.

```
In[•]:= NetMeasurements[trainedNet["TrainedNet"], testSet, "ConfusionMatrixPlot"]
```

Out[•]=

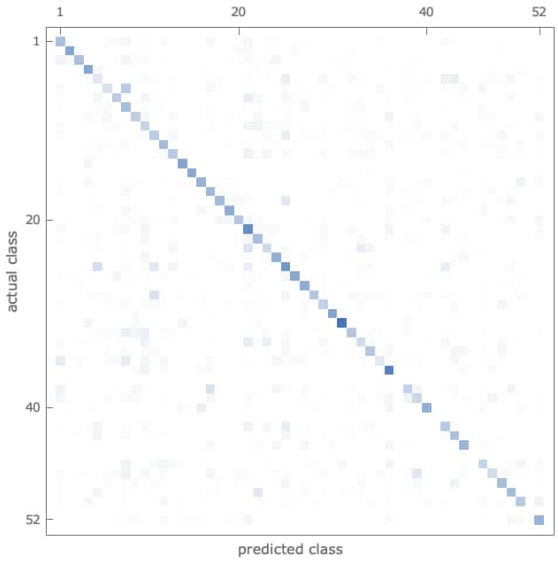


Рисунок 0.10 – Результати тестування моделі

Після досягнення бажаного результату, модель експортувалась у форматі onxx.

3.3 Форматування моделі

Отримано готову модель у форматі onnx. Аби переформатувати модель, використано офіційний пакет для мови програмування Python під назвою “coremltools”. Для цього використовувався Python код, у якому вказано усі потрібні для перетворення параметри, включно з очікуваним форматом моделі після переформатування наявної. Таким чином отримано модель у потрібному форматі.

3.4 Написання Swift проєкта

Розроблено Swift проєкт з базовим графічним інтерфейсом та наступною логікою роботи та функціональністю.

Користувач має змогу змінити мову інтерфейсу на англійську, а мову передбачень на латинську, або інтерфейс та мову передбачень на українську, застосувавши клавішу “Укр”(“Lat”). Зображення інтерфейсу двома мовами показано нижче. Слід зауважити, що для створення фонового зображення було використано

штучний інтелект, котрий здатен генерувати зображення “Text-to-image”. Інтерфейс англійською та українською мовою зображений на рисунках 3.11 та 3.12 відповідно.

Рисунок 0.11 – Інтерфейс англійською мовою

Рисунок 0.12 – Інтерфейс українською мовою

При натисканні на кнопку “Інфо” (“Info”) на екрані з’являться короткий опис назви та автора програми, інструкція та застереження щодо використання. Це вікно

можна закрити натиснувши клавішу “Окей” (“Okay”), або посунувши пальцем вниз екрану. Ця сторінка зображена на рисунку 3.13.

Рисунок 0.13 – Короткі відомості програми українською мовою

Клавіша “Галерея” (“Gallery”) знаходиться у лівому нижньому кутку та відповідає за вибір однієї фотографії з фототеки користувача для здійснення передбачення на основі цієї фотографії. Для цього використовується стандартний системний графічний інтерфейс користувача. Для передбачення одночасно можна

обрати лише одну фотографію. Для пошуку фотографії за змістом, можна використати вбудований функціонал операційної системи IOS.

Приклад наведено на рисунку 3.14.

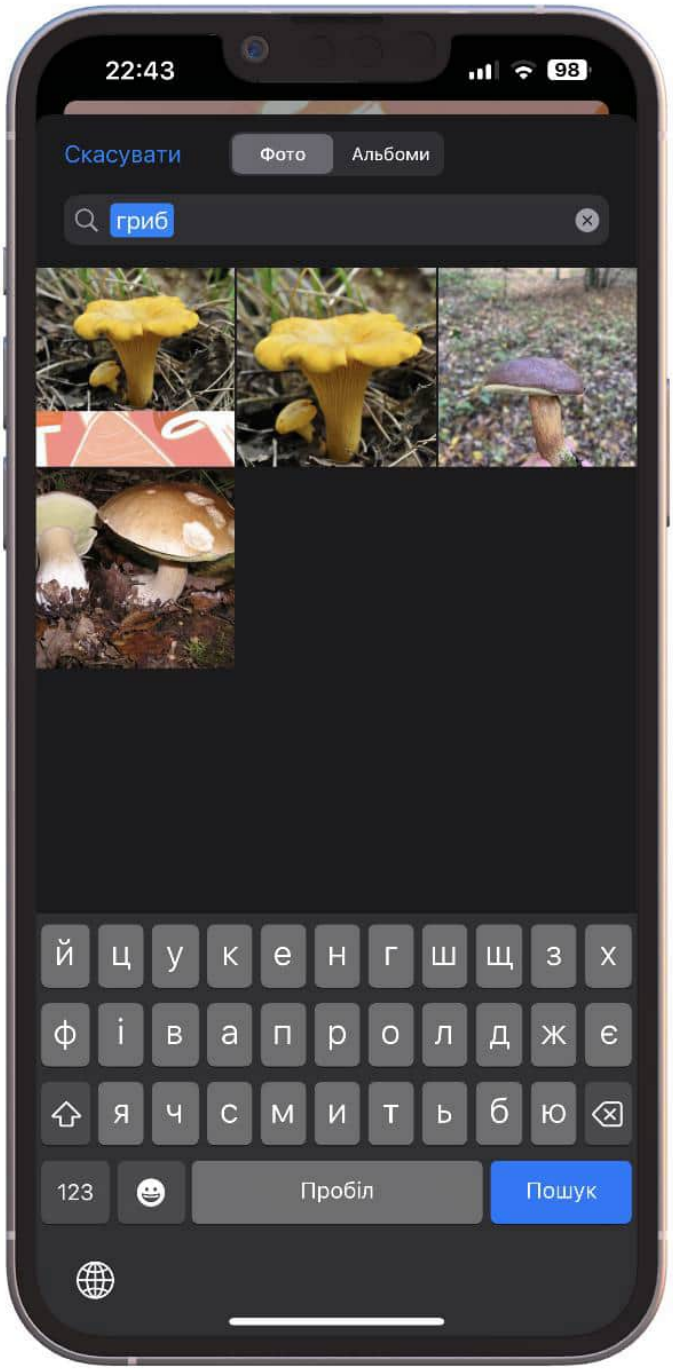


Рисунок 0.14 – Інтерфейс додавання фотографій

Клавіша “Камера” (“Camera”), яку можна побачити у правому нижньому кутку, відповідає за відкриття системної програми для зйомки фотографії у режимі

реального часу. Користувач має можливість використати щойно зроблену фотографію, або ж спробувати зробити кращу, можливо чіткішу фотографію. Нижче наведені знімки екрану та описаного вище функціоналу клавіші “Камера”. Процес фотографування та функціонал верифікації сфотографованого зображення відображено на рисунках 3.15 та 3.16 відповідно.

Рисунок 0.15 – Процес фотографування

Рисунок 0.16 – Функціонал верифікації сфотографованого зображення

Після вибору фотографії з фототеки, або здійснення зйомки перебуваючи безпосередньо біля гриба, програма опрацьовує зображення та виводить на екран свої передбачення. На екрані з’являються два найбільш “впевнені” передбачення. Натискаючи клавіші “Укр” (“Lat”) користувач може змінювати переклад передбачення. Передбачення латинською та українською мовою зображені на рисунках 3.17 та 3.18 відповідно.

Рисунок 0.17 – Виведені передбачення латинською мовою

Рисунок 0.18 – Виведені передбачення українською мовою

4 РОЗРОБКА СИСТЕМИ АВТОМАТИЧНОГО РОЗГОРТАННЯ

4.1 Конфігурування Xcode Cloud

Розробку системи автоматичного розгортання розпочато з конфігурування Xcode Cloud. Для цього потрібно було придбати “Apple Developer Account”. Після отримання доступу до порталу розробника Apple згенеровано ключі та сертифікати для підписання зібраних застосунків. Для цього використано функціонал порталу. Створені сертифікати зображені на рисунку 4.1.

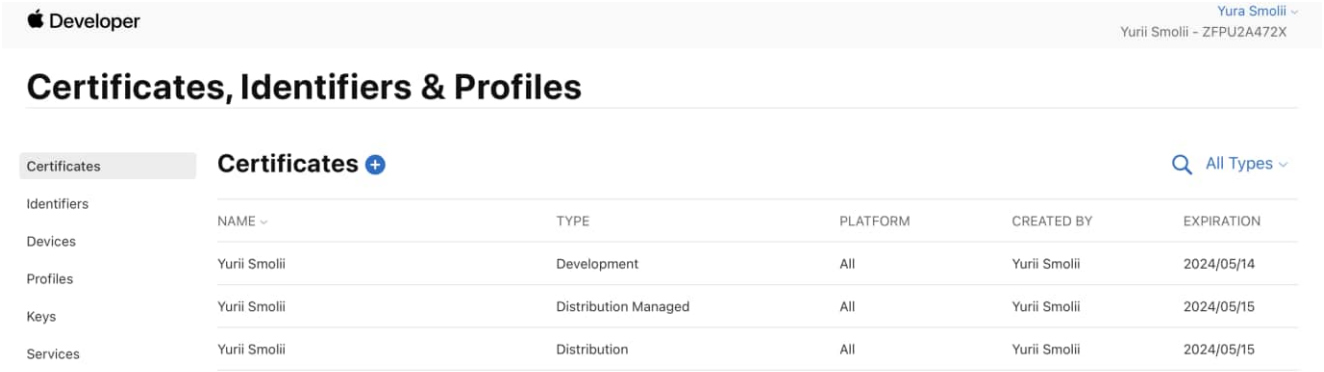


Рисунок 0.1 – Створені сертифікати

Також для застосунка був створений унікальний ідентифікатор з назвою застосунка “MushroomsAI”, який зображений на рисунку 4.2.

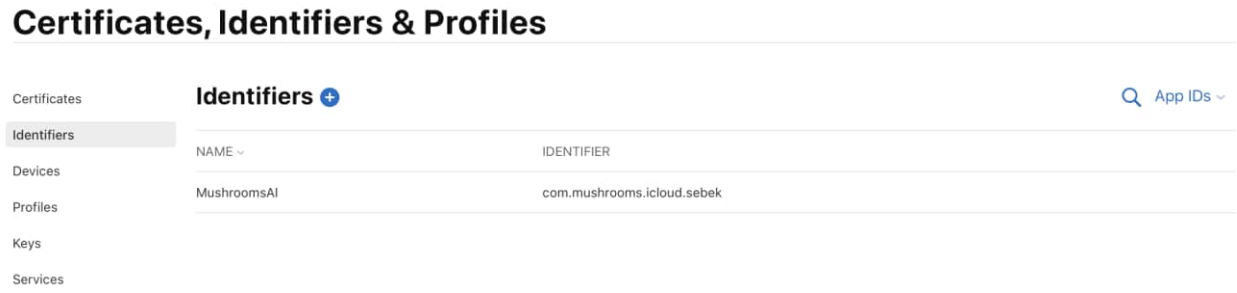


Рисунок 0.2 – Створений ідентифікатор

Опісля створення усіх потрібних сертифікатів, налаштовано їх використання на стороні Xcode Cloud. Конфігурація зображена на рисунку 4.3.

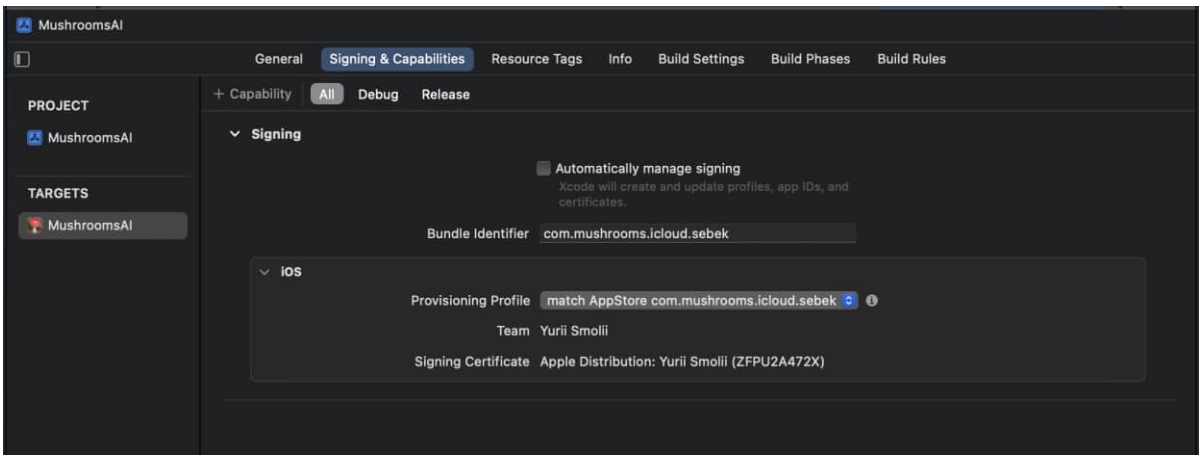


Рисунок 0.3 – Конфігурація сертифікатів

Після налаштувань перевірено роботу Xcode Cloud, виконавши надсилання зібраного застосунка у TestFlight. Процес надсилання зображений на знімках екрану нижче.

Обирання останньої збірки зображене на рисунку 4.4.

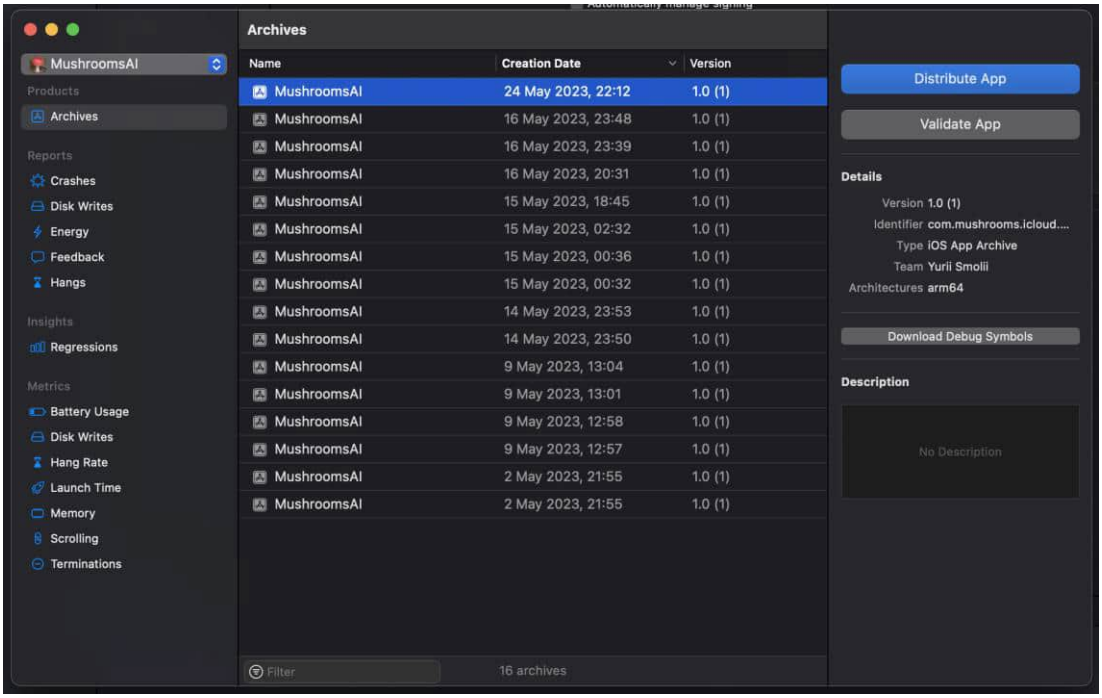


Рисунок 0.4 – 1 крок - вибір збірки

Обирано метод дистрибюції, а саме App Store Connect, оскільки цікавить саме надсилання у TestFlight (рисунок 4.5).

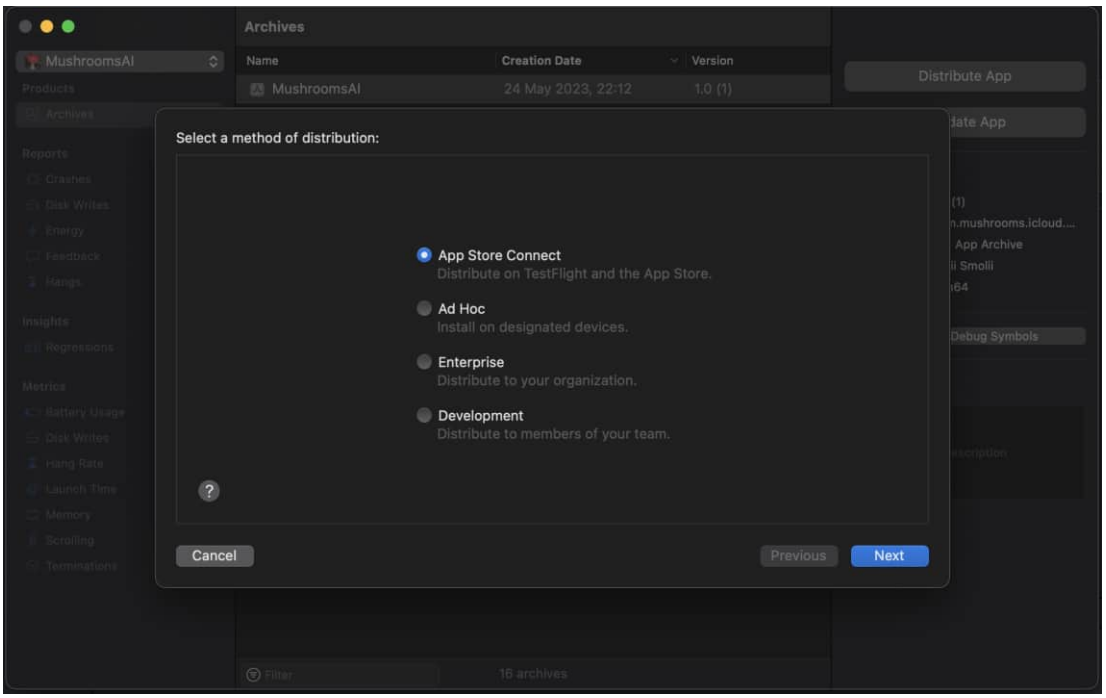


Рисунок 0.5 – 2 крок - Вибір методу дистрибюції

Крок вибору отримувача зображений на рисунку 4.6.

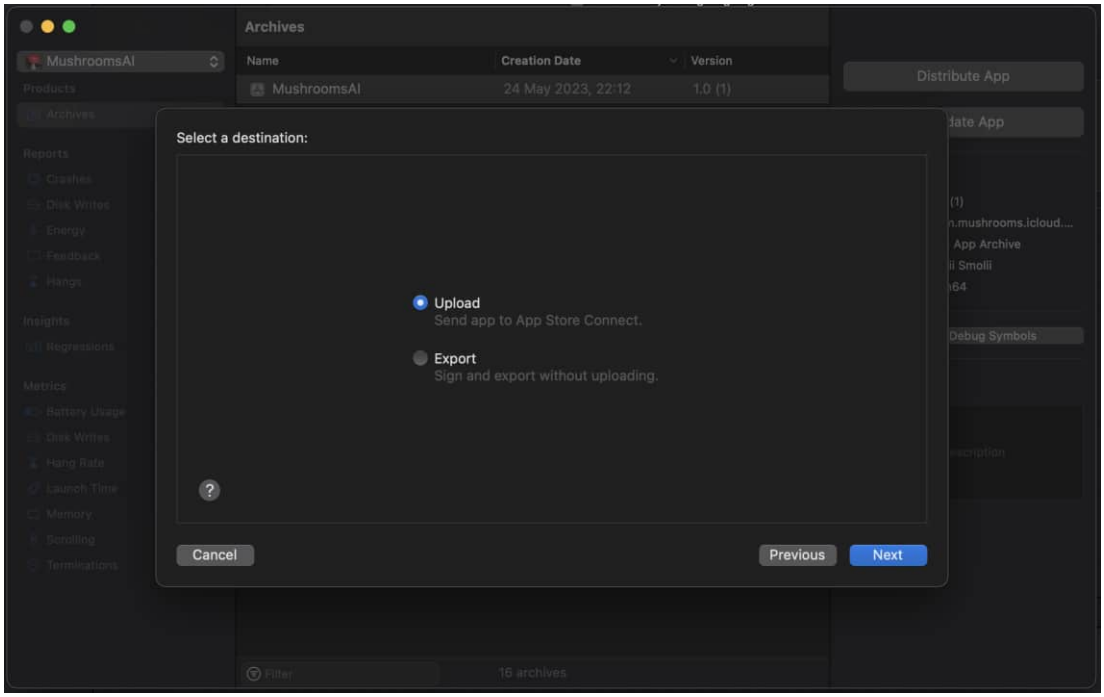


Рисунок 0.6 – 3 крок - Вибір отримувача

Наступним кроком Xcode Cloud автоматично перевіряє правильність зібраного застосунка. Після цього обрано два параметри (рисунок 4.7).

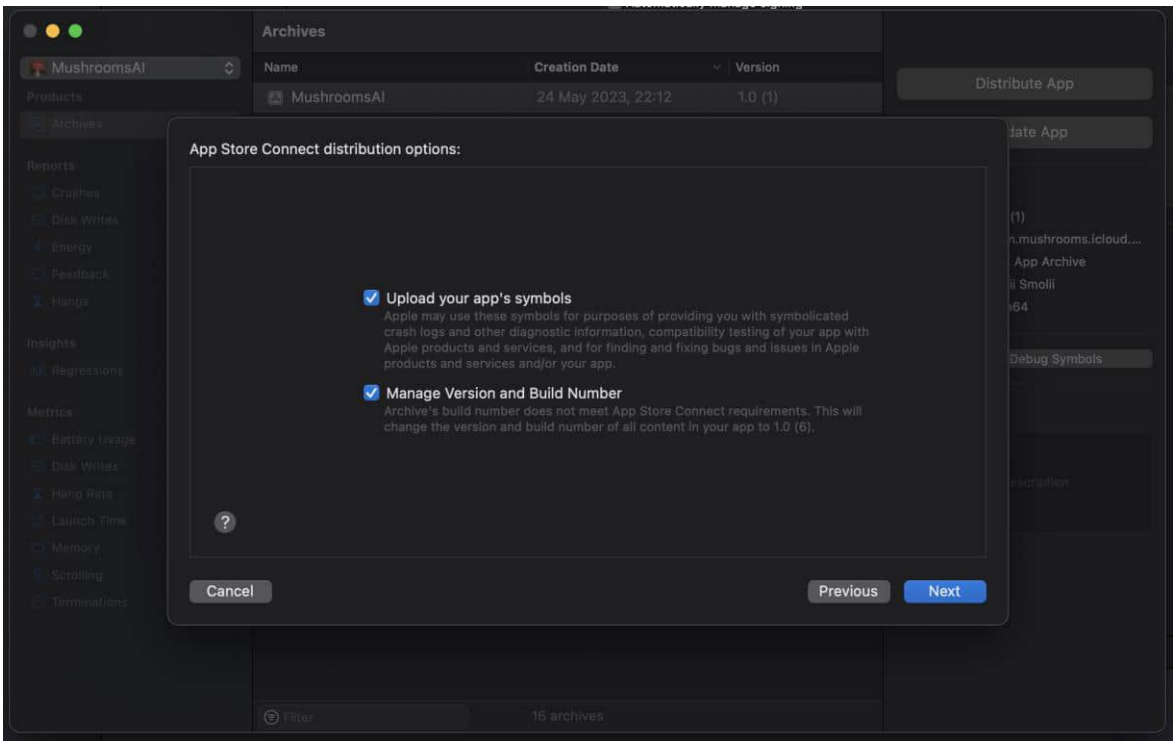


Рисунок 0.7 – 4 крок - Вибір налаштувань

Наступним етапом вибрано, створені на попередніх кроках, сертифікат та профіль (рисунок 4.8).

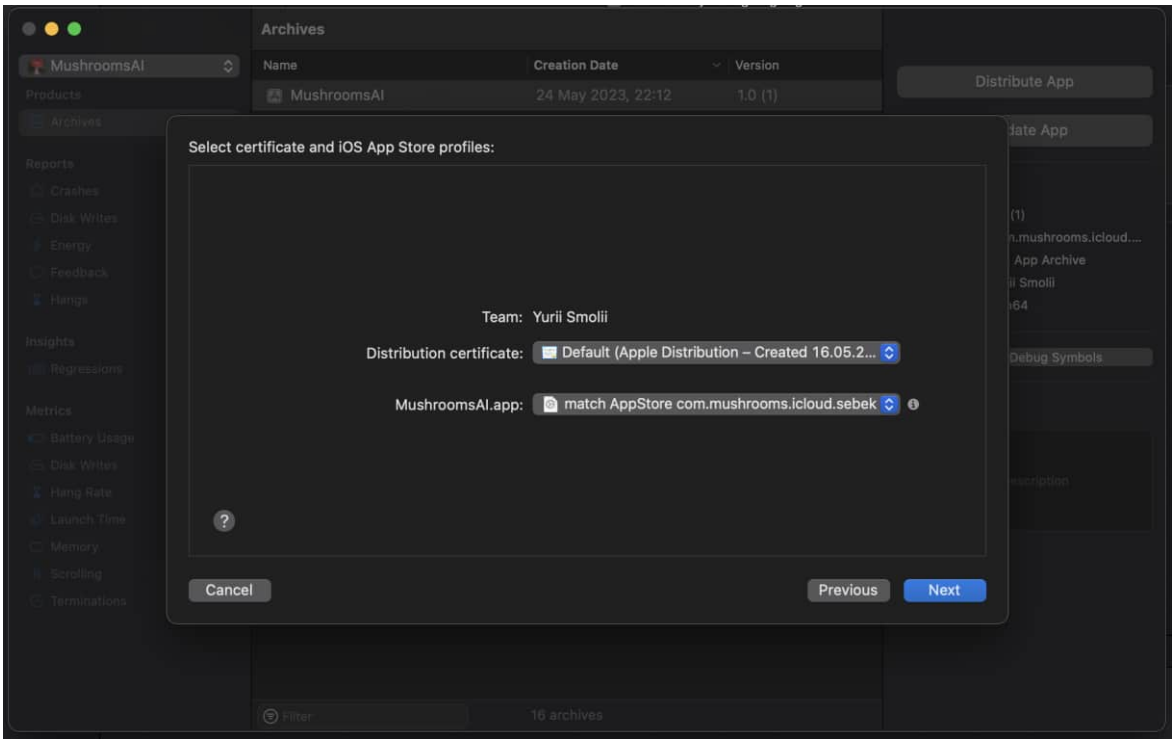


Рисунок 0.8 – 5 крок - Вибір сертифікату та профіля

Далі відбувається фінальна валідація усіх конфігурацій. Натискаючи кнопку “Upload” надіслано команду почати процес дистрибюції застосунка на TestFligh. Початок дистрибюції зображений на рисунку 4.9.

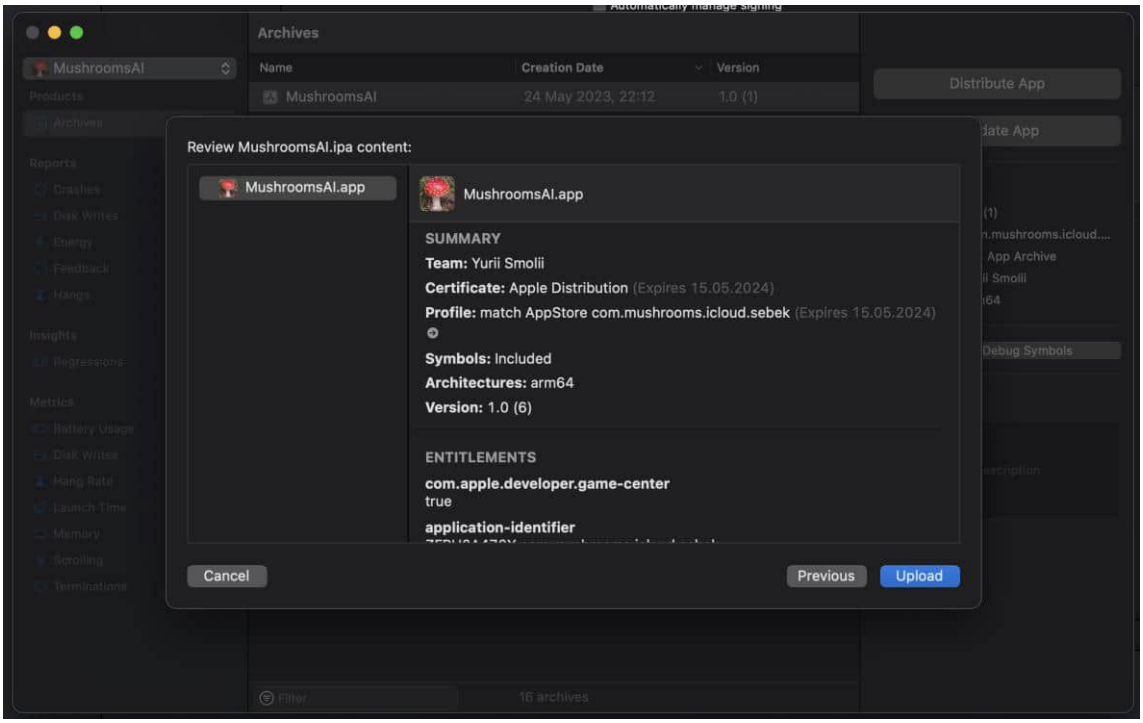


Рисунок 0.9 – 6 крок - Початок дистрибюції

Через певний відрізок часу застосунок потрапляє у TestFlight (рисунок 4.10). Кожному з бета–тестувальників та усім розробникам на електронну скриньку приходять листи про вихід нової версії та успішну дистрибюцію (рисунок 4.11). У застосунку TestFlight з’являється можливість оновити застосунок до останньої версії. Це сповіщення зображене на рисунку 4.12.

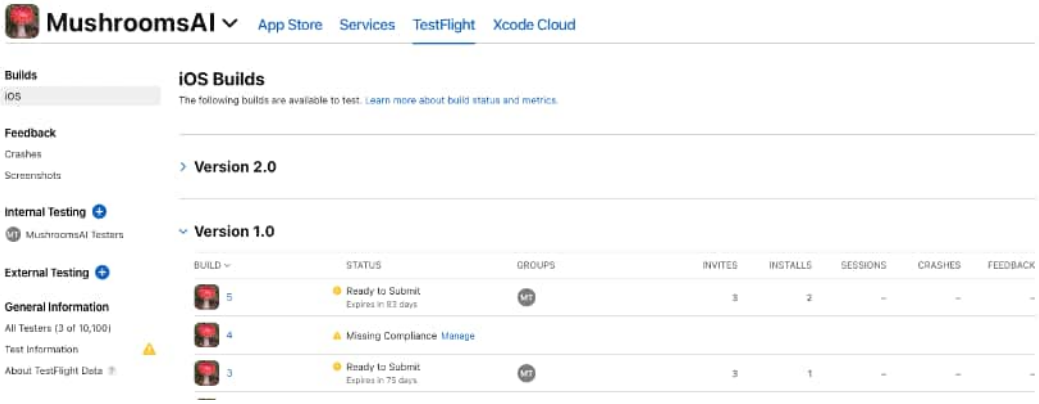


Рисунок 0.10 – TestFlight

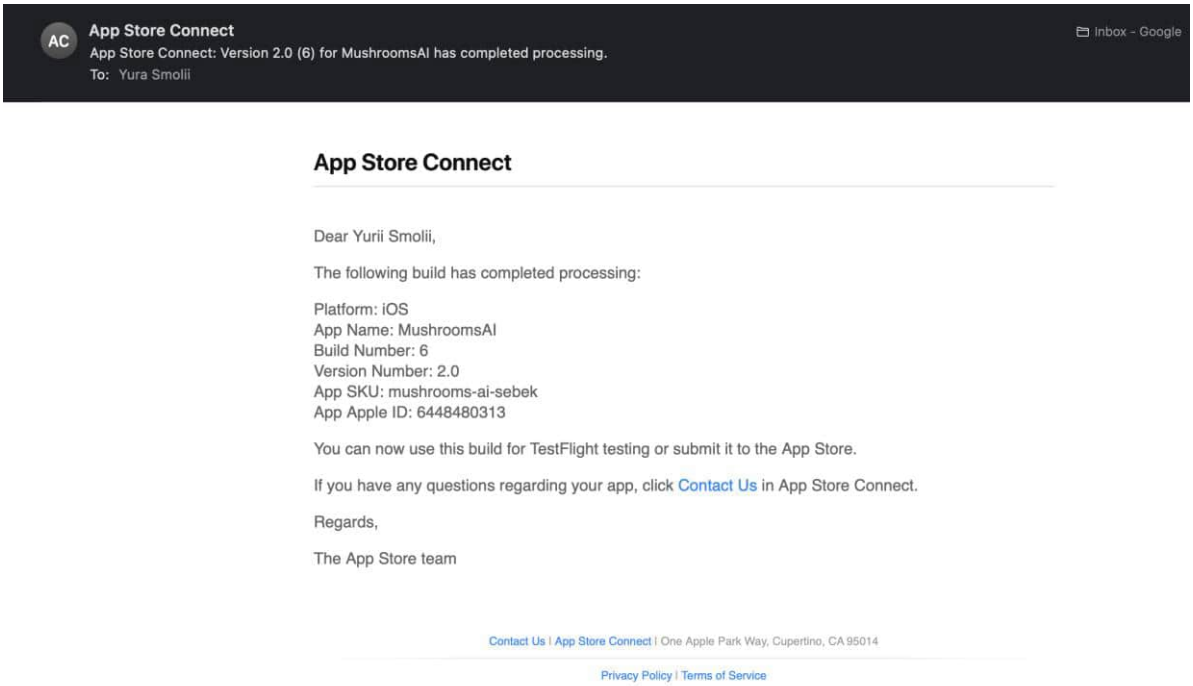


Рисунок 0.11 – Сповіщення про успішну дистрибюцію

Рисунок 0.12 – Сповіщення про вихід нової версії

На даному етапі успішно завершнено конфігурування та перевірено роботу автоматичного розгортання за допомогою Xcode Cloud.

4.2 GitHub Actions та Fastline

Наступним кроком розпочато розробку системи автоматичного розгортання за допомогою GitHub Actions та інструменту Fastline.

Для цього створено репозиторій та під’єднано Xcode проєкт до нього, надіслано вихідний код проєкту у віддалений репозиторій.

Наступний крок – налаштування безпосередньо конвеєра, який називається Action. У ньому можна налаштувати різноманітні збудники розгортання застосунка. Вибрано наступних два варіанти:

- Ручна команда з можливістю встановлення версії застосунка та повідомлення про внесені зміни у нову версію застосунка.
- Автоматичний збудник, котрий реагуватиме на потрапляння нових змін у віддалений репозиторій на git гілці “main”.

Після налаштування збудника конвеєра потрібно налаштувати віддалену обчислювальну машину, на котрій буде збиратись застосунок. Згідно вимог до збирання IOS застосунків, налаштовано автоматичне встановлення інтерпретатора “Ruby”, встановлення інструменту “Fastline” за допомогою вбудованого пакетного менеджера “gem”, Xcode у версії без графічного інтерфейсу.

Для підписання на віддаленій машині зібраного застосунка потрібно передати відповідні сертифікати на віртуальну машину, Для цього потрібно згенерувати API ключ, за допомогою якого інструмент Fastline зробить запит на портал розробника Apple. API ключ складається з секретних даних, саме тому для їх збереження використано GitHub Actions Secrets, та застосовано Fastline для генерації API ключа. Збереження секретних ключів зображене на рисунку 4.13.

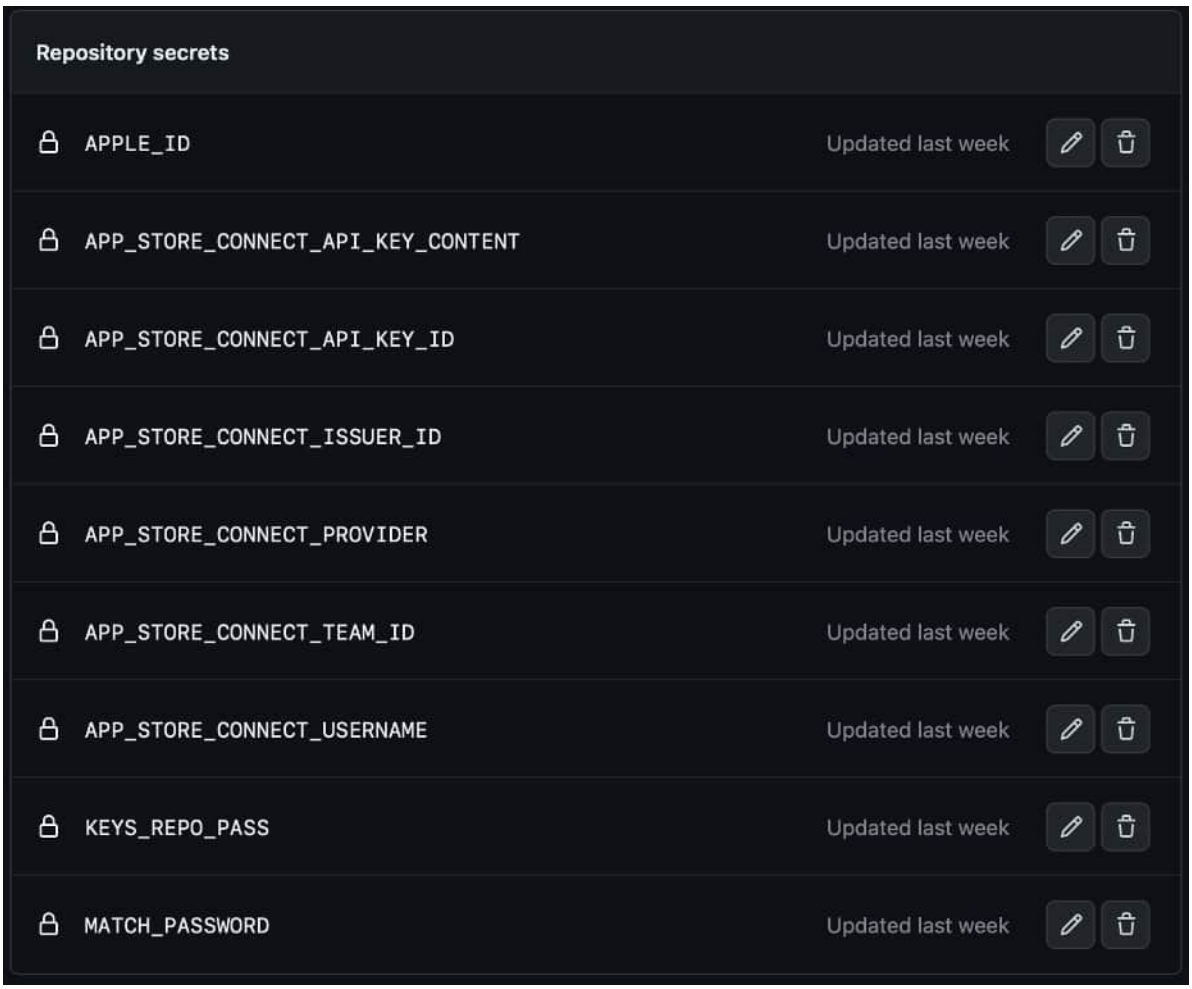


Рисунок 0.13 – Збереження секретних ключів

Після додавання усіх ключів розроблено внутрішній конвеєр, що використовуватиметься інструментом Fastline (рисунок 4.14). Він складається з кількох дій:

- Api_key – крок генерації ключа для отримання сертифікатів та профіля підписання.
- Create_keychain – крок на якому створюється запис у стандартному менеджері ключів та сертифікатів на віртуальній машині для подальшого зберігання сертифікатів.
- Match – внутрішній інструмент, котрий слідкує за актуальністю усіх сертифікатів та профілів та зберігає їх останню версію на окремому віддаленому репозиторії.

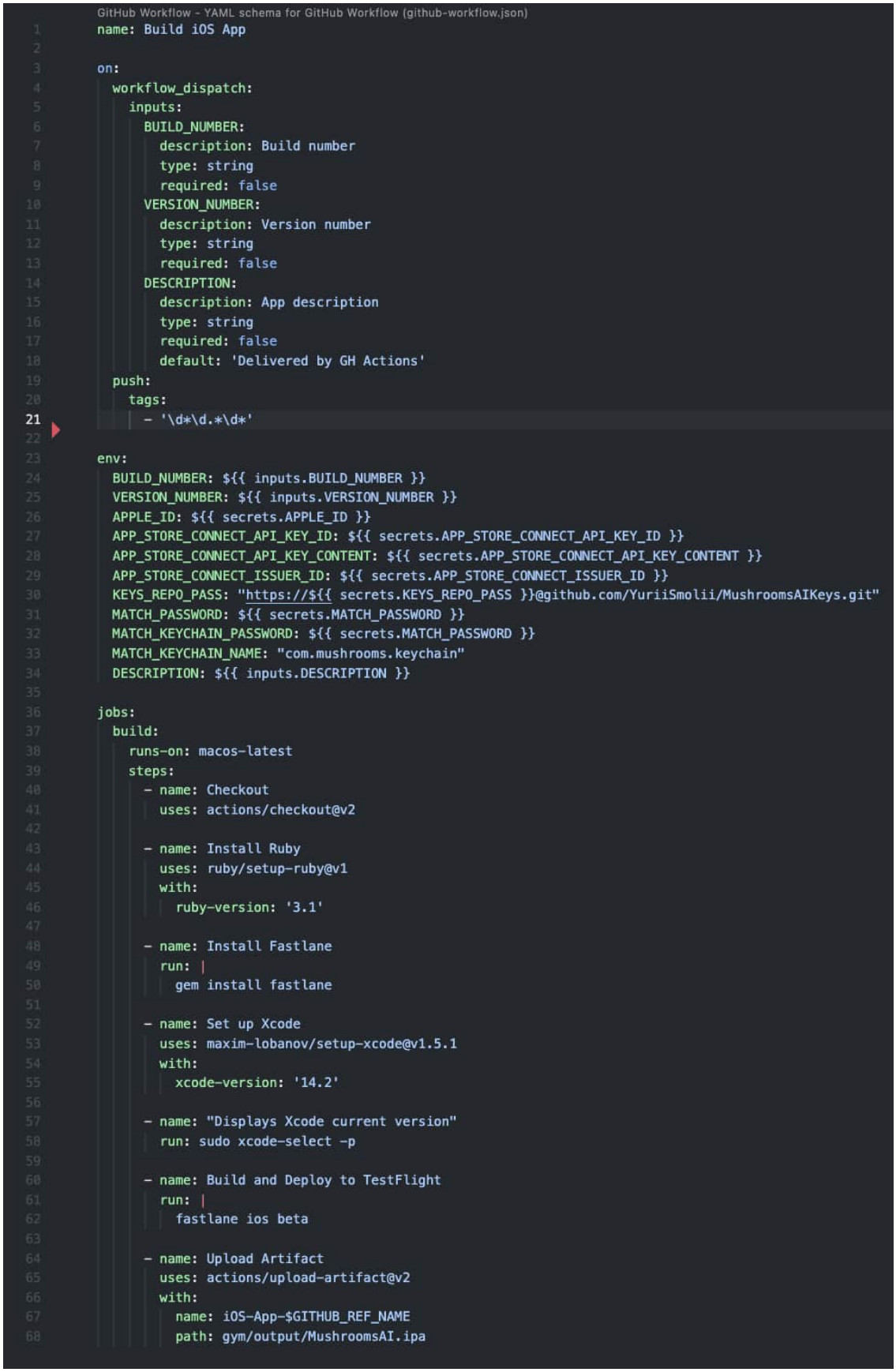
- Increment_build_number – для примусового вказування версії застосунка.
- Increment_version_number – для примусового вказування номера збірки застосунка.
- Gym – крок, на якому застосунок збирається з вихідного коду проєкта у .ipa файл, котрий буде використовуватись далі для дистрибюції у TestFlight.
- Pilot – крок на якому Fastline буде виконувати дистрибюцію зібраного файла на TestFlight.

Fastlane > fastfile

```
default_platform(:ios)

platform :ios do
  desc "Push a new beta build to TestFlight"
  lane :beta do
    # Select Xcode version
    xcversion(version: "14.2")

    # Configure access to Apple Dev Account via API_KEY
    api_key = app_store_connect_api_key(
      key_id: ENV['APP_STORE_CONNECT_API_KEY_ID'],
      issuer_id: ENV['APP_STORE_CONNECT_ISSUER_ID'],
      key_content: ENV['APP_STORE_CONNECT_API_KEY_CONTENT'],
      duration: 1200, # optional (maximum 1200)
      in_house: false # optional but may be required if using match/sigh
    )

    create_keychain(
      name: ENV["MATCH_KEYCHAIN_NAME"],
      password: ENV["MATCH_KEYCHAIN_PASSWORD"],
      unlock: true,
      default_keychain: true
    )

    # match(git_basic_authorization: ENV['KEYS_REPO_PASS'])
    match(
      username: ENV['APPLE_ID'],
      git_url: ENV['KEYS_REPO_PASS'],
      app_identifier: "com.mushrooms.icloud.sebek",
      keychain_password: ENV['MATCH_KEYCHAIN_PASSWORD'],
      keychain_name: ENV["MATCH_KEYCHAIN_NAME"],
      api_key: api_key,
      verbose: true,
      force: true,
      # readonly: is_ci,
      generate_apple_certs: true,
      team_id: "ZFPU2A472X",
      type: "appstore",
      include_mac_in_profiles: true,
      skip_confirmation: true
    )

    increment_build_number(
      build_number: ENV['BUILD_NUMBER'], # specify specific build number (
      xcodeproj: "./MushroomsAI.xcodeproj" # (optional, you must specify t
    )

    increment_version_number(
      version_number: ENV['VERSION_NUMBER'], # Set a specific version numb
      xcodeproj: "./MushroomsAI.xcodeproj"
    )

    # Build your app
    gym(
      scheme: "MushroomsAI",
      configuration: "Release",
      project: "./MushroomsAI.xcodeproj",
      export_method: "app-store",
      xcargs: "-allowProvisioningUpdates"
    )

    # Distribute your app to TestFlight
    pilot(
      api_key: api_key,
      app_identifier: 'com.mushrooms.icloud.sebek',
      beta_app_description: ENV['DESCRIPTION'],
    )
  end
end
```

Рисунок 0.14 – Fastline конвеєр

Конвеєр GitHub Acions зображений на рисунку 4.15.



```
GitHub Workflow - YAML schema for GitHub Workflow (github-workflow.json)
name: Build iOS App

on:
  workflow_dispatch:
    inputs:
      BUILD_NUMBER:
        description: Build number
        type: string
        required: false
      VERSION_NUMBER:
        description: Version number
        type: string
        required: false
      DESCRIPTION:
        description: App description
        type: string
        required: false
        default: 'Delivered by GH Actions'
  push:
    tags:
      - '\d*\d.*\d*'

env:
  BUILD_NUMBER: ${{ inputs.BUILD_NUMBER }}
  VERSION_NUMBER: ${{ inputs.VERSION_NUMBER }}
  APPLE_ID: ${{ secrets.APPLE_ID }}
  APP_STORE_CONNECT_API_KEY_ID: ${{ secrets.APP_STORE_CONNECT_API_KEY_ID }}
  APP_STORE_CONNECT_API_KEY_CONTENT: ${{ secrets.APP_STORE_CONNECT_API_KEY_CONTENT }}
  APP_STORE_CONNECT_ISSUER_ID: ${{ secrets.APP_STORE_CONNECT_ISSUER_ID }}
  KEYS_REPO_PASS: "https://${{ secrets.KEYS_REPO_PASS }}@github.com/YuriiSmolii/MushroomsAIKeys.git"
  MATCH_PASSWORD: ${{ secrets.MATCH_PASSWORD }}
  MATCH_KEYCHAIN_PASSWORD: ${{ secrets.MATCH_PASSWORD }}
  MATCH_KEYCHAIN_NAME: "com.mushrooms.keychain"
  DESCRIPTION: ${{ inputs.DESCRIPTION }}

jobs:
  build:
    runs-on: macos-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2

      - name: Install Ruby
        uses: ruby/setup-ruby@v1
        with:
          ruby-version: '3.1'

      - name: Install Fastlane
        run: |
          gem install fastlane

      - name: Set up Xcode
        uses: maxim-lobanov/setup-xcode@v1.5.1
        with:
          xcode-version: '14.2'

      - name: "Displays Xcode current version"
        run: sudo xcode-select -p

      - name: Build and Deploy to TestFlight
        run: |
          fastlane ios beta

      - name: Upload Artifact
        uses: actions/upload-artifact@v2
        with:
          name: iOS-App-$GITHUB_REF_NAME
          path: gym/output/MushroomsAI.ipa
```

Рисунок 0.15 – Конвеєр GitHub Acions

Після закінчення імплементації внутрішнього Fastline конвеєра та зовнішнього GitHub Actions конвеєра, проведено тестування імплементованого механізму автоматичного розгортання застосунку. Для цього двічі запущено зовнішній конвеєр.

Спершу у ручному режимі (рисунок.4.16).

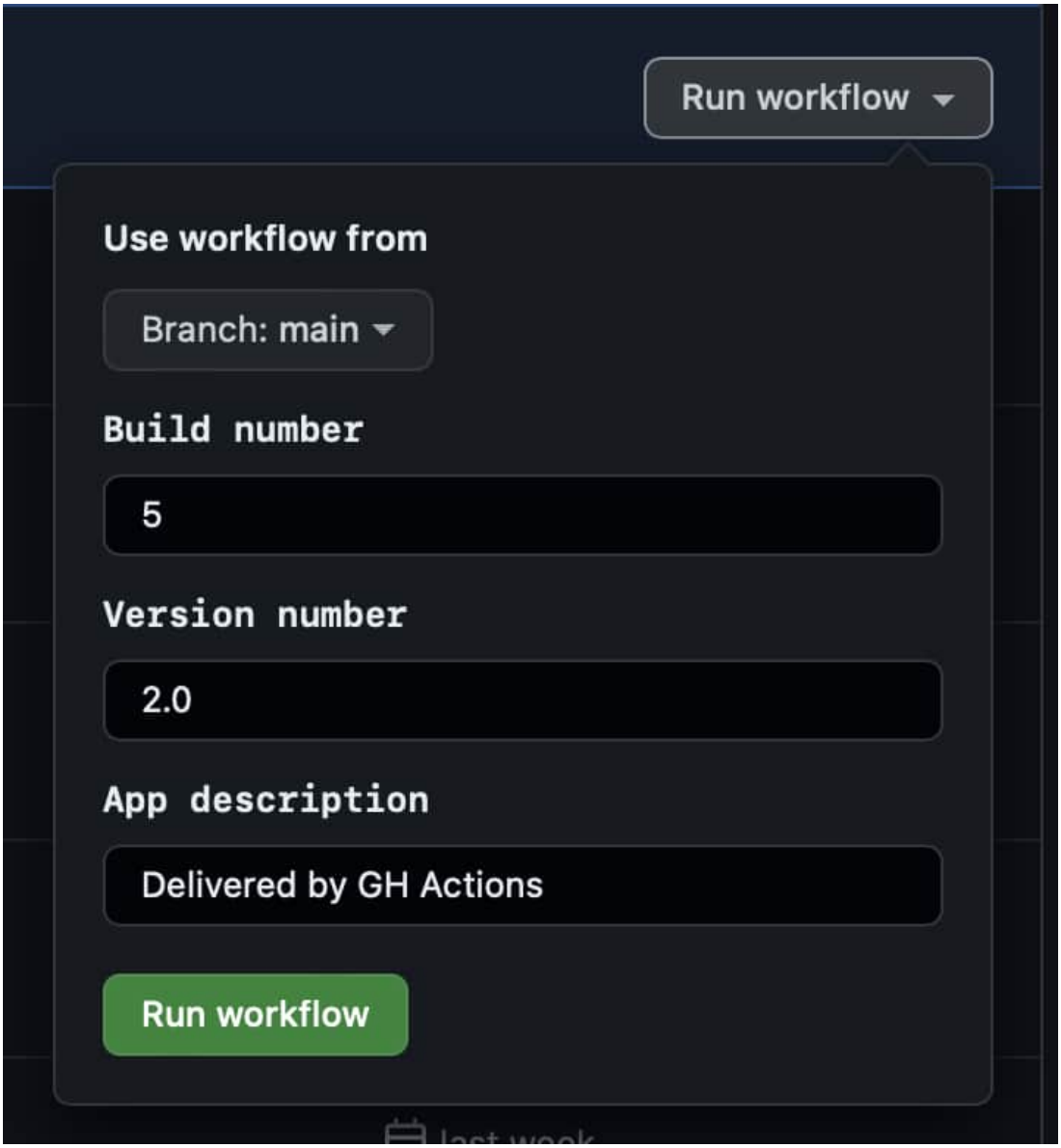


Рисунок 0.16 – Ручний запуск конвеєра

Та у автоматичному режимі, надіславши код у віддалений репозиторій на “main” гілці.

Обидва конвеєри успішно запустились на пройшли усі внутрішні етапи. Результат зображений на рисунку 4.17.

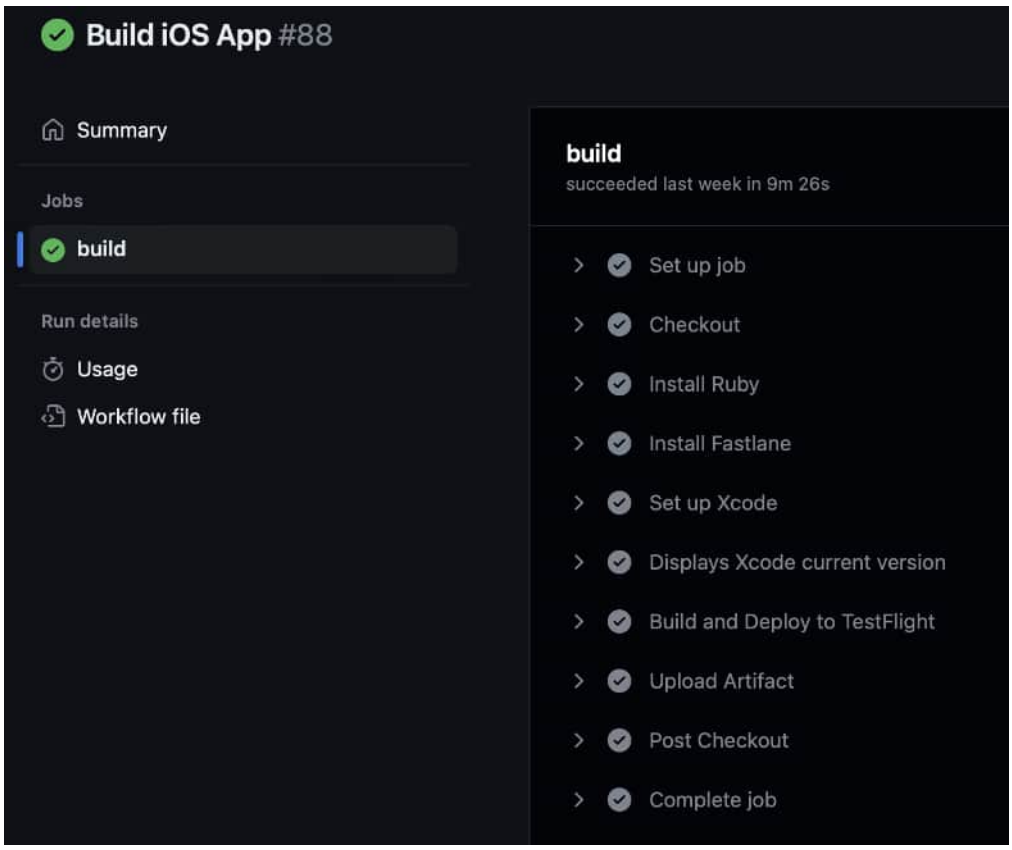


Рисунок 0.17 – Результат виконання конвеєрів

Після успішного виконання усіх конвеєрів, перевірено присутність нової версії у TestFlight та відправлення відповідного листа усім тестувальникам застосунку (рисунок 4.18).

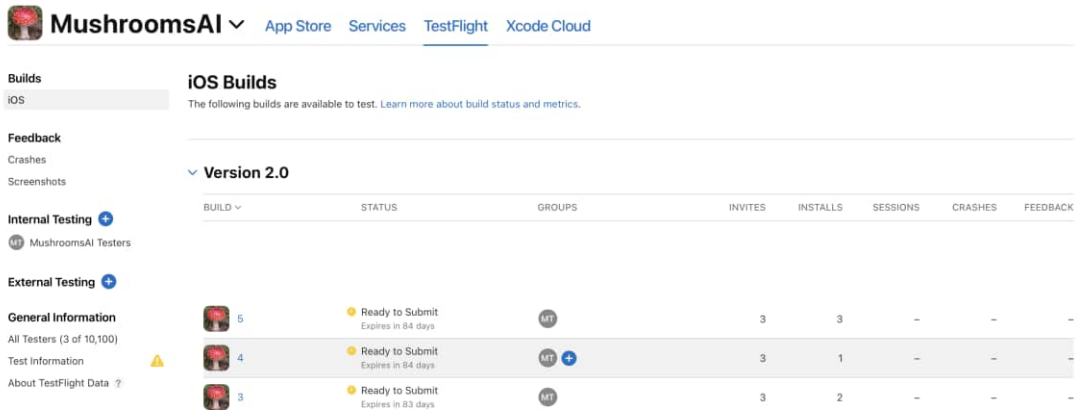


Рисунок 0.18 – Портал TestFlight

Отже, є підтвердження правильності усіх конфігурацій автоматичного розгортання.

4.3 Порівняння різних імплементацій

4.3.1 Xcode Cloud

Xcode Cloud є послугою, наданою компанією Apple, яка пропонує інтегровану платформу для розробки, тестування та розгортання додатків iOS. Основні переваги Xcode Cloud включають:

- Xcode Cloud надає інтегровану робочу область, де можна здійснювати всі кроки розробки, включаючи збірку, тестування та розгортання.
- Xcode Cloud автоматично здійснює збірку та тестування вашого коду після коміту до репозиторію.
- Xcode Cloud ідеально поєднується з Xcode IDE і дозволяє безпосередньо працювати з проектом у зручному інтерфейсі.

Недоліки такого підходу.

Після завершення роботи над певною задачею, Xcode Cloud вимагає доволі багато ручних кроків, таких як створення сертифікатів та їх вибір. Після чого процес розгортання складно називати власне автоматичним.

4.3.2 GitHub Actions та Fastline

GitHub Actions – це сервіс, який надає можливість автоматизувати різні процеси у вашому репозиторії на GitHub, включаючи збірку, тестування та розгортання. Fastlane – це інструмент, який допомагає автоматизувати розгортання та інші завдання, пов'язані з розробкою iOS–додатків. Основні переваги GitHub Actions + Fastlane включають:

- Гнучкість: GitHub Actions дає широкі можливості налаштування автоматичних робочих процесів. Є можливість налаштувати специфічні кроки для збірки, тестування та розгортання застосунку.
- Можливість повної автоматизації: Після закінчення роботи над завданням, від користувача буде потрібно лише надіслати код у віддалений репозиторій.

- Спільна робота: GitHub Actions дозволяє команді спільно працювати над автоматичним розгортанням. Кожен учасник команди може вносити зміни до робочого процесу та додавати власний код.
- Широкий вибір інтеграцій: GitHub Actions має багато інтеграцій з іншими інструментами, такими як Fastlane, які полегшують роботу з розгортанням та іншими задачами.

Недоліки:

Процес налаштування автоматичного розгортання за допомогою GitHub Actions та Fastline вимагає поглибленого розуміння роботи таких сервісів та інструментів як GitHub Actions, GitHub Actions Secrets, Fastline, Match, що робить процес розробки конвеєра складним. В окремих випадках кожен запуск конвеєра може бути платним.

Вибір між Xcode Cloud та GitHub Actions + Fastlane залежить від особистих уподобань та потреб. Якщо у проєкті використати Xcode та інші сервіси Apple, то Xcode Cloud може бути зручним варіантом. З іншого боку, якщо працювати з репозиторієм на GitHub та потребувати більше гнучкості у налаштуванні робочих процесів, GitHub Actions та Fastlane можуть краще задовільнити ці потреби.

ВИСНОВКИ

В цій роботі успішно розроблено iOS-додаток з використанням моделі CoreML для розпізнавання грибів. Додаток дозволяє користувачам швидко та точно визначати види грибів шляхом аналізу фотографій.

Для розробки додатку використано мову програмування Swift та фреймворк CoreML для інтеграції моделі машинного навчання. Модель CoreML навчена на великому наборі даних зображень грибів, що дозволяє досягти високої точності розпізнавання. Користувач завантажує фотографію гриба у програму, а додаток автоматично визначатиме його.

Для забезпечення швидкого та ефективного розгортання додатку налаштовано системи автоматичного розгортання в Testflight. Використовуючи GitHub Actions + Fastlane та Xcode Cloud створено конвеєри, які забезпечують збірку та розгортання додатку на платформі Testflight. Це дозволить команді розробників ефективно співпрацювати та швидко надавати можливість оновлення та виправлення помилок користувачам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wolfram Language & System [Веб ресурс] – Режим доступу до ресурсу: https://reference.wolfram.com/language/

2. Wolfram Network Programming [Веб ресурс] – Режим доступу до ресурсу: https://reference.wolfram.com/language/guide/NetworkProgramming.html

3. Python 3.10.4 documentation [Веб ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/

4. Coremltools API [Веб ресурс] – Режим доступу до ресурсу: https://apple.github.io/coremltools/

5. Swift Documentation [Веб ресурс] – Режим доступу до ресурсу: https://www.swift.org/documentation/

6. iNaturalist API [Веб ресурс] – Режим доступу до ресурсу: https://api.inaturalist.org/v1/docs/

7. Core ML framework [Веб ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/coreml

8. Stephan Wolfram An Elementary Introduction To The Wolfram Language 2nd edition / Stephan Wolfram - 2022 - 350с

9. Bruce F. Torrence The Student's Introduction to Mathematica and the Wolfram Language 3rd edition / Bruce F. Torrence – Eve A. Torrence – 2019 – 554с

10. Kroly Nyisztor Machine Learning with Core ML 2 and Swift: A beginner-friendly guide to integrating machine learning into your apps (Swift Clinic) / Kroly Nyisztor – 2018 – 154с