МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

15.03 — КМР. 2002–“С” 2023.01.11. 037 ПЗ

ЧЕЧАЙЛЮК ВОЛОДИМИР ЮРІЙОВИЧ

2024 р.

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.9:003.76:655.24

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету інформаційних технологій	Завідувач кафедри комп'ютерних наук
Болбот І. М., д.т.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ 2024 р.	**____ _____ 2024 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Інтелектуальна система створення шрифту на основі почерку

Спеціальність _____121 «Інженерія програмного забезпечення»_____
(код і назва)

Освітня програма _____Програмне забезпечення інформаційних систем___
(назва)

Орієнтація освітньої програми ___освітня-професійна_____
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____професор, д.т.н. _____ _Семко В. В.____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

_____професор д.т.н._____ _____ ___Хиленко В.В. _____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ **_____Чечайлюк В.Ю._____**
(підпис) (ПІБ студента)

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
доцент к.т.н. _____ **Голуб Б. Л.**
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"_1_" ___листопада_____2023 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ
Чечайлюк В.Ю.
(прізвище, ім'я, по батькові)
Спеціальність 121 «інженерія програмного забезпечення»
(код і назва)
Освітня програма Програмне забезпечення інформаційних систем
(назва)
Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Тема магістерської кваліфікаційної роботи Інтелектуальна система створення шрифту на основі почерку
затверджена наказом ректора НУБіП України від "_1_" ___листопада_2023р. №_2002 С__
Термін подання завершеної роботи на кафедру 29.11.2024
(рік, місяць, число)
Вихідні дані до магістерської кваліфікаційної роботи: Наукові статті та матеріали з дослідження методів комп'ютерного бачення, обробки зображень та машинного навчання.
Перелік питань, що підлягають дослідженню:
1. Провести аналіз актуальності цифрового почерку.
2. Дослідити суть технічної проблеми.
3. Дослідити методи обробки зображення для визначення символів із фото.
4. Дослідити можливості застосування машинного навчання для визначення символів із фото.
4. Дослідити методи розробки шрифтів.
Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_1_" _листопада_2023 р.

Керівник магістерської кваліфікаційної роботи _____ Хиленко В.В.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Чечайлюк В.Ю.
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення

ІТ – інформаційні технології

ІС – інформаційна система

RGB - Red Green Blue

UML - unified modeling language

ВСТУП

Вміння писати в свій час кардинально змінило історію людства та дозволило зробити величезний стрибок в розвитку. Можливість зафіксувати певні знання та інформацію на сторонніх предметах (дерево, камінь а згодом і папір) дозволило зменшити залежність від усної комунікації та спростило накопичення знань від покоління до покоління.

Сьогодні текст та письмо вже зовсім звичні для нас і є невід'ємною частиною людського життя. Більше того, з розвитком цифрових технологій та друку все меншою стає потреба в написанні тексту від руки. Книги більше не пишуть від руки, а більшість інформації зберігається та розповсюджується через інтернет. Нотатки зручніше створювати на смартфоні, а для спілкування все частіше використовують голосові повідомлення та відео дзвінки. Письмо поступово втрачає своє практичне значення, перетворюючись на елемент культурної спадщини та раритету.

В той же час почерк - це унікальне поєднання ліній, яке не повторюється у двох окремих людей. Крім того, навіть написана кілька разів та сама буква тією самою рукою буде, десь більше десь менше, але все ж відрізнятись від попередньої. Це робить почерк - певною ідентичністю, чимось що виділяє особистість з поміж інших.

В той же час, почерк несе в собі певні культурні риси. Він часто залежить від того, як саме пишуть в регіоні та може відрізнятись від міста до міста. Більше того, часто по почерку можна визначити певні риси характеру людини, наприклад якщо почерк рівний та майже одноманітний - це може говорити про охайність та правильність особистості. Також почерк може вказувати на професію людини, наприклад - лікарі, через потребу у швидкому нотуванні тексту їх почерк часто стає погано розбірним та розмазаним.

Про те, що почерк людини відображає її темперамент, помітив ще Аристотель. Трохи пізніше цим питанням займалися Лафатер, Лейбніц, Гете та ін. У наші дні графологія - вчення про зв'язок почерку з характером - є

розвиненою наукою з твердою теоретичною основою і цілим рядом різних напрямів. Розпізнавання особистісного типу людини за почерком сьогодні широко використовується в психологічній діагностиці, в космічній медицині, в криміналістиці. Відомі випадки, коли по одному лише почерку фахівцям вдавалося розкривати злочини. У ряді країн західної Європи аналіз особистості людини по його почерку застосовується при прийомі на роботу. [1]

З іншого боку, з розвитком інтернету, соціальних мереж та блогінгу, питання цифрової унікальності стає все більш гострим. Люди придумують собі нові імена, аватари, образи і так далі, намагаючись таким чином передати свою ідентичність. При цьому кожна людина має в своєму розпорядженні інструмент, який дозволяє виражати унікальність ще зі школи.

Також, хоч досліджувана система в основному націлена саме на роботу із почерком, шрифт створений в результаті може бути результатом певних дизайнерських рішень та потреб. Система створення шрифтів відкриває нові можливості не лише для збереження особистої ідентичності, але й для дизайнерських та комерційних цілей. Створення унікального шрифту може бути корисним для брендів, які прагнуть підкреслити свою індивідуальність. Власний шрифт може зекономити витрати на графічний дизайн, дозволяючи компаніям створювати впізнавані матеріали, які будуть відображати їхню фірмову естетику.

Важливо також зазначити, що розробка такої системи потребує глибокого розуміння різних технологій та детального дослідження на різних етапах. Робота вимагає детального розгляду різних технологічних сфер та їх поєднання між собою.

Таким чином, розробка інтелектуальної системи, що дозволяє перетворити рукописний текст у цифровий шрифт, є не лише технологічним викликом, але й способом зберегти культурне та особистісне значення почерку у світі цифрових технологій та навіть принести комерційну вигоду для компаній.

Об’єктом дослідження є індивідуальний почерк людини.

Предметом дослідження є автоматичний процес створення шрифту із фото почерку.

Практична цінність системи полягає у автоматизації рішення для цифровізації почерку.

Апробація розробленого додатку була представлена на конференції «ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ РОЗРОБКИ КОМП'ЮТЕРНИХ СИСТЕМ 2024». Подання можна знайти за посиланням - http://econference.nubip.edu.ua/index.php/itete/XV/author/submission/3400

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИЗНАЧЕННЯ ВИМОГ

1.1 Визначення суті технічної проблеми

Письмо - це набір певних ліній на площині, зазвичай папері. Шрифт - це комп’ютерний файл, який містить всю необхідну інформацію для відображення певних символів на екрані. Загальна суть проблеми - перетворити лінії нарисовані на папері у комп’ютерний файл. При цьому від користувача необхідно вимагати мінімальну кількість дій та обладнання.
Технічну проблему можна розділити на два етапи:

1. Визначення окремих символів з почерку.
2. Формування файлу шрифту на основі окремих зображень символів.

1.1.1 Визначення окремих символів з фото почерку. Тобто необхідно програмно виділити кожен окремий контур написаний на папері та визначити що це за символ. Для вирішення даної проблеми, можна скористатись наступними опціями:

1. Малювання символів на сенсорному дисплеї за допомогою стилуса. Цей метод потребує від користувача додатково сенсорний дисплей, стилус, та додаткове програмне забезпечення для реалізації. Хоча, натомість, він може дати простіший та ефективніший спосіб обробки символів.
2. Розпізнавання символів із фото тексту користувача за допомогою алгоритмів машинного навчання. Даний метод потребує великої кількості фотографій різних почерків, щоб натренувати модель, та не дозволить використовувати дизайнерські малюнки, які не дуже схожі на справжні літери написані рукою.
3. Розпізнавання символів із фото користувача за допомого алгоритмів комп’ютерного зору та математичних операцій. В цьому випадку, користувачу потрібно мати лише папір та ручку. Цей метод є більш універсальним, хоча з іншого боку має більшу залежність від якості фото.

1.1.2 Формування файлу шрифту на основі окремих зображень. Комп'ютерний шрифт - це файл, який містить в собі векторне зображення кожного окремого символу. Існують також шрифти з растровими зображеннями, але їх майже неможливо точно масштабувати та змінювати, тому майже завжди використовуються саме векторні шрифти формату .ttf (TrueType Format), або .otf (OpenType Format).

Отже, для того, аби шрифт був максимально схожим на реальний почерк людини, необхідно для початку правильно векторизувати всі зображення, а потім завантажити у відповідну таблицю файлу шрифта кожен символ на свою позицію. Також після або до завантаження літер, необхідно переконатись, що вони мають правильні розміри, та вірно розміщенні в комірках гліфів. Після всіх операцій шрифт потрібно зберегти до файлової системи користувача.

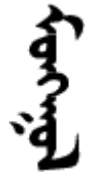
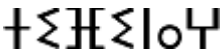
1.2 Опис предметної області

Дана робота поєднує в собі дві різні області, які варто виділити окремо.

1.2.1 Письмо та почерк. Люди в різних куточках світу розвивали своє письмо окремо, але не кожна мова має свій повністю окремий алфавіт, через запозичення тих чи інших символів між територіальними угруповуваннями. Саме тому наразі виділяють лише 9 основних абеток (див. табл. 1.1).

Основні абетки світу Таблиця 1.1

Հայերէն Armenian	Кирилиця Cyrillic
მხედრული Georgian (Mkhedruli)	Ελληνικά Greek
한글 Korean (Hangeul)	AB

	Latin/Roman
 Mongolian	 N'Ko
 Tifinagh	

Цікавий також факт, що деякі мови не мають своєї абетки, а використовують ієрогліфи для позначення не конкретних символів, а одразу певних звуків, слів, або навіть понять.

Почерк кожної людини унікальний, що робить його особливим серед інших форм вираження. Він формується під впливом багатьох факторів, включаючи освіту, культуру, звички, навіть характер і емоційний стан людини. У процесі письма мозок координує рухи руки, і цей процес настільки індивідуальний, що почерк може вважатися своєрідним відбитком людини. Це знайшло своє відображення в таких дисциплінах, як графологія, яка аналізує зв’язок між почерком і психологічними характеристиками особистості.

Особливості почерку значно залежать від мови та алфавіту, на яких пише людина. Наприклад, у мовах, які використовують латинський алфавіт, літери переважно базуються на простих геометричних формах — прямі лінії, дуги, петлі. У кирилиці, зокрема в слов’янських мовах, літери мають більш складну структуру з численними перехрестями та закругленнями.

У мовах, що використовують ієрогліфи, наприклад китайській, почерк вимагає зовсім іншого підходу. Ієрогліфи складаються з базових елементів — рисок і крапок, які утворюють складні форми. Кожен символ несе не тільки

звукове, а й смислове навантаження. Японська мова, яка використовує три системи письма (кандзі, хірагану і катакану), поєднує логографічні і фонетичні елементи, що робить почерк складним та багатогранним.

У арабській писемності почерк формується завдяки плавним зв'язкам між літерами, які змінюють форму залежно від їхнього положення в слові (на початку, середині чи кінці). Це додає унікальної краси та динаміки арабському письму, роблячи його схожим на мистецтво каліграфії. Аналогічно, писемності Південної Азії, такі як деванагарі, вирізняються складною системою утворення символів, що вимагає злагодженого чергування вертикальних, горизонтальних і діагональних ліній.

Почерк є не тільки інструментом спілкування, але й базовою формою для створення друкарських шрифтів. Багато традиційних друкарських стилів, таких як гарнітури Times New Roman або Arial, спиралися на аналіз ручного письма. Цей зв'язок між почерком і шрифтами зберігається досі, особливо у створенні дизайнерських чи персоналізованих текстових стилів, які імітують ручне письмо.

1.2.2 Комп'ютерний шрифт. Шрифт — це набір символів (літер, цифр, знаків пунктуації та інших графічних елементів), створений у певному стилі та розмірі. Він використовується для візуального представлення тексту як у друкованому, так і в цифровому вигляді. Шрифти відіграють ключову роль у типографіці, дизайні та спілкуванні, адже вони впливають на читабельність тексту, його естетичність і емоційне сприйняття.

Шрифт складається з кількох основних компонентів:

- Гліфи - це візуальні представлення символів, кожен з яких відповідає певній літері, цифрі чи знаку. Наприклад, символи "A", "a" і "@" є окремими гліфами.
- Таблиця кодування (Unicode). Вона визначає відповідність між символами шрифту і їхніми кодами в стандарті Unicode, що забезпечує правильне відображення тексту на різних платформах.

- Метрики. Метрики включають розміри і просторові характеристики кожного символу, такі як ширина символу, відступи між літерами (кернінг) та висота рядка. Вони визначають, як текст відображається на екрані або у друкованому вигляді.
- Інструкції з рендерингу. У деяких форматах шрифтів (наприклад, TrueType) є інструкції, які допомагають правильно рендерити символи на різних роздільностях.

Існує кілька різних форматів шрифтів, які відрізняються будовою, сферою застосуванням, та відповідно мають певні переваги та недоліки. Ось основні з них:

1. TrueType (TTF):
    - один із найпоширеніших форматів, розроблений компаніями Apple і Microsoft.
    - містить інструкції з рендерингу, що забезпечує високу якість відображення символів на будь-якій роздільності.
2. OpenType (OTF):
    - удосконалена версія TTF, яка підтримує розширені таблиці гліфів, наприклад, для лігатур, альтернативних символів чи багатомовних наборів.
    - є більш гнучким і сумісним із сучасними стандартами.
3. WOFF/WOFF2:
    - веб-шрифти, створені спеціально для використання в інтернеті.
    - оптимізовані для швидкого завантаження та роботи в браузерах.
4. Bitmap Fonts:
    - растрові шрифти, які зберігаються як набір пікселів для кожного символу.
    - використовуються у випадках, коли потрібна висока швидкість або простота, але мають обмеження у масштабуванні.
5. SVG Fonts:

- векторні шрифти, що зберігають кожен символ у форматі SVG (Scalable Vector Graphics).
- часто використовуються в графічних додатках або для дизайнерських цілей.

6. Variable Fonts:
    - новий формат шрифтів, який дозволяє змінювати властивості символів (вагу, нахил, ширину) без необхідності завантаження різних стилів шрифту.

Шрифти є складними технічними об’єктами, що поєднують графіку, математику та програмування. Вони забезпечують основу для передачі тексту у будь-якому візуальному середовищі, від паперу до цифрових екранів. Кожен формат має свої переваги та сфери застосування, що дозволяє вибирати найкраще рішення залежно від конкретних вимог і контексту.

1.3 Аналіз існуючих програм-аналогів

Програмою аналогом буде вважатись така програма, яка дає можливість автоматично створити шрифт із вашого почерку. Будемо також розглядати програми, які використовують не лише метод розбору фотографії, а й інші доступні способи.

Отже першим прикладом альтернативних продуктів, є додаток під назвою Caligraph (рис. 1.1).

Рис. 1.1. Домашня сторінка сервісу Caligraph

Даний сервіс є веб додатком (https://www.calligraphr.com/) та його ідея дуже схожа на ідею цієї роботи. Додаток пропонує створити шрифт із почерку просто та зручно, та має безкоштовну пробну версію із обмеженням в 75 гліфів. Також додаток підтримує велику кількість різних мов, в тому числі й мови з використанням ієрогліфів. Єдиним ускладненням на шляху до створення шрифту, стає потреба роздрукувати шаблон таблиці, який потім потрібно заповнити необхідними символами і зісканувати (рис. 1.2).

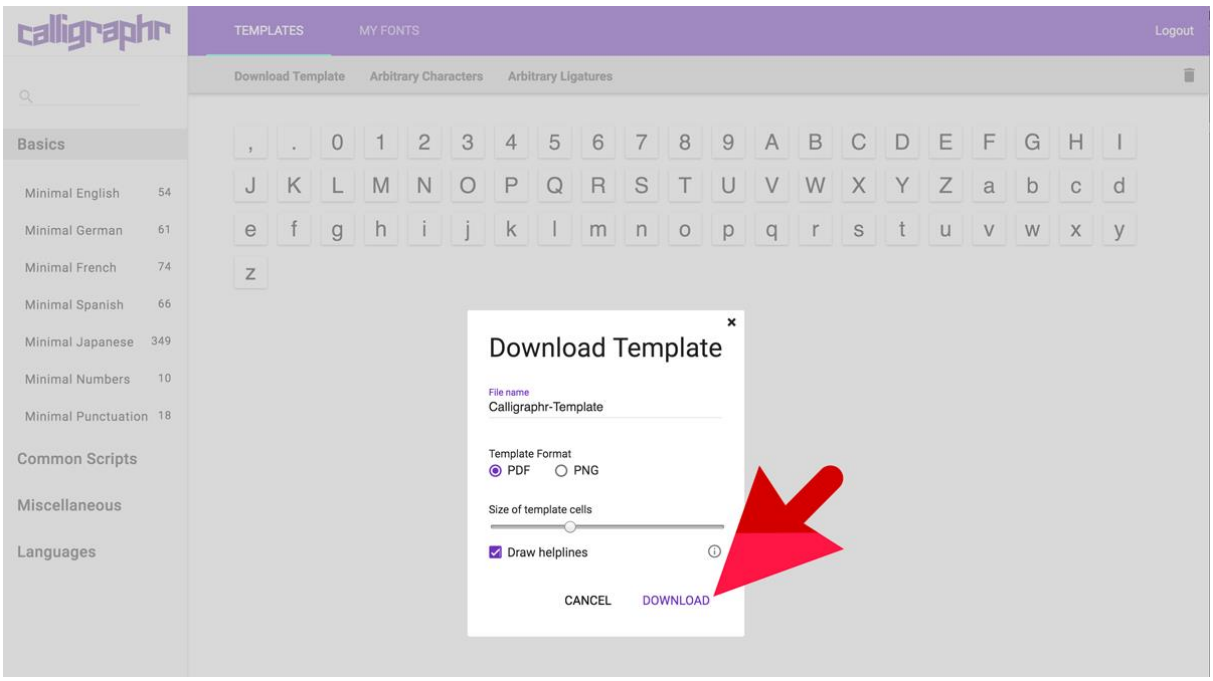


Рис. 1.2. Вимога до завантаження шаблону на сайті Caligraph

Бачимо віконце із пропозицією завантажити шаблон. Також на фоні помітно загальний інтерфейс додатку та його структуру.

Наступним прикладом програми-аналога є сервіс YourFonts. Зовнішній вигляд домашньої сторінки цього ресурсу, можна побачити на рис. 1.3.

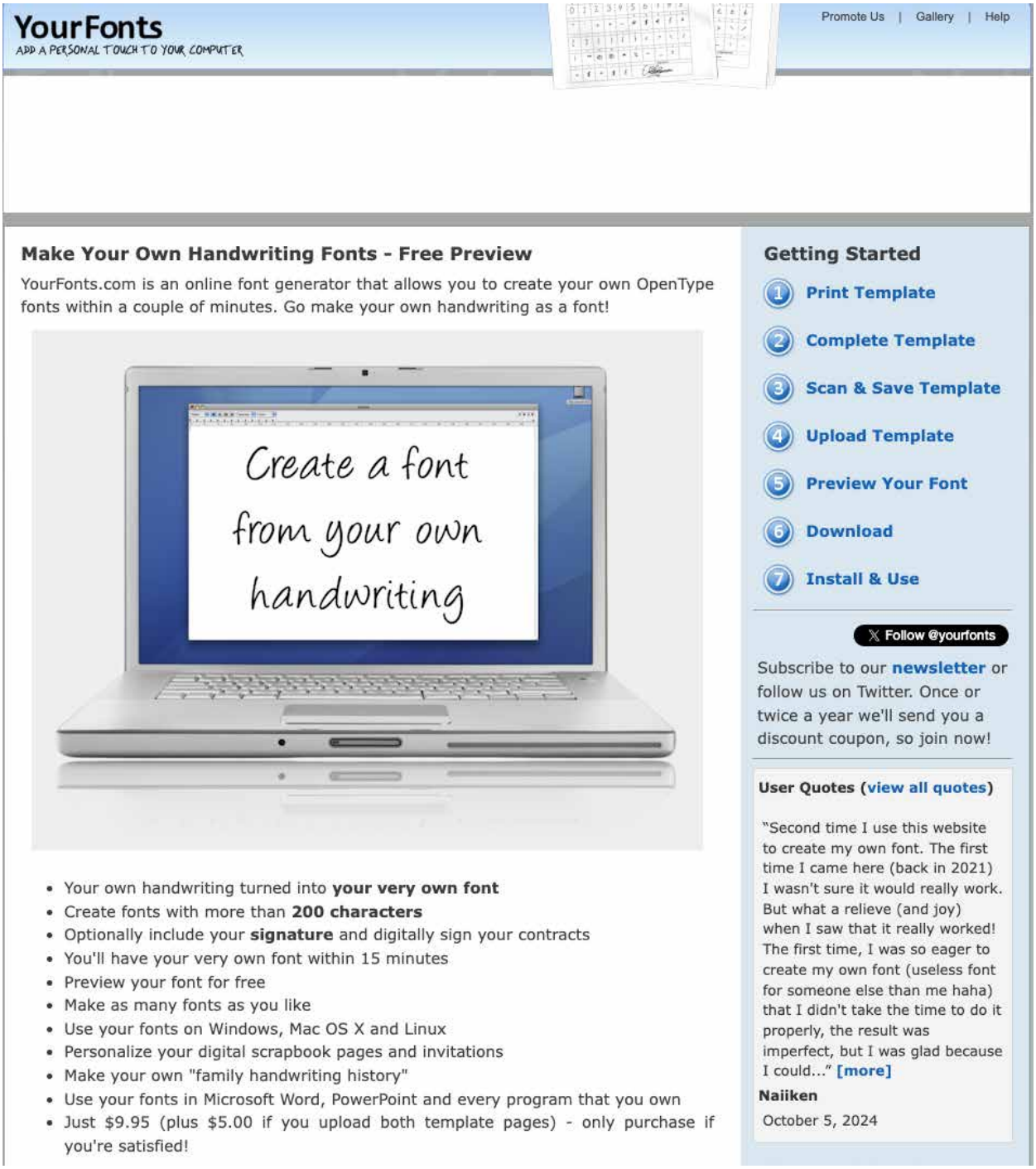


Рис 1.3. Домашня сторінка ресурсу YourFonts

Одразу помітно, що дизайн даної платформи значно старіший, хоч і достатньо інформативний. Кроки для створення шрифта майже ідентичні із попереднім аналогом, але даний сервіс не дає можливості безкоштного користування. Натомість розробники пропонують попередньо переглянути відображення шрифта, перед його оплатою та завантаженням.

Наступним прикладом альтернативного рішення є мобільний застосунок HandWriting FontMaker (рис. 1.4)

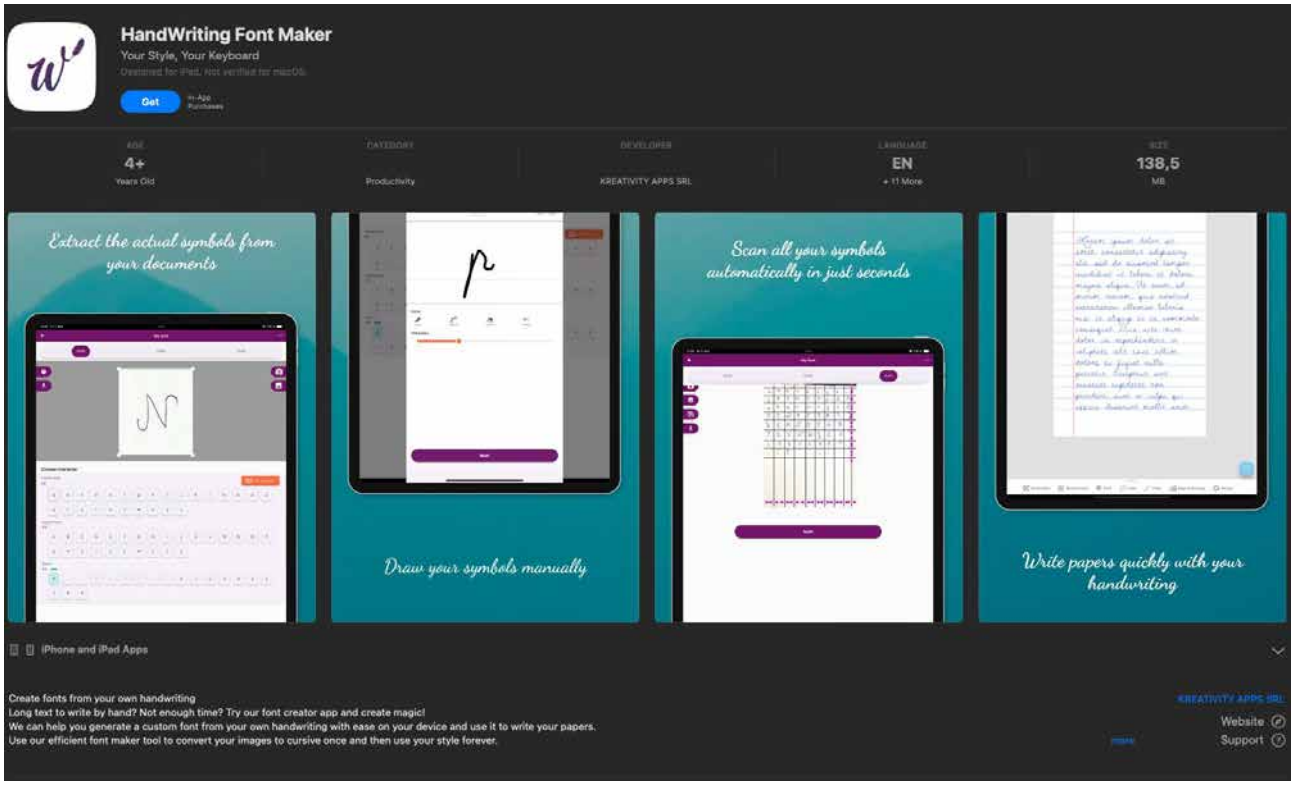


Рис. 1.4. Сторінка додатку в магазині App Store

Цей додаток пропонує трохи інший підхід - він працює на девайсах із сенсорними екранами, тому дозволяє малювати потрібні контури прямо на екрані. Звісно, використовуючи такий додаток, можна намалювати символи й пальцем, але для правильного відтворення саме почерку, необхідно додатково мати спеціальний стилус, та впевнитись, що пристрій підтримує його. Більшість сучасних смартфонів не підтримують тонкі стилуси, отже найкращим варіантом буде саме використання планшету. Також варто зауважити, що навіть маючи все необхідне обладнання, процес написання стилусом на сенсорному екрані, серйозно відрізняється від писання ручкою на папері, отже результат також навряд буде правильним.

Отже підсумуємо всі переваги та недоліки проаналізованих систем за топомогою табл. 1.2

Порівняння програм аналогів Таблиця 1.2

	Даний проєкт	Caligraph	YourFonts	HandWriting FontMaker
Автоматичне створення шрифту	+	+	+	+
Можливість використання на різних пристроях	+	+	+	-
Мінімальні вимоги до обладнання користувача	+	+	+	-
Мінімальна кількість необхідних кроків	+	-	-	-
Підтримка великої кількості різних мов	-	+	+	+
Візуальний інтерфейс	-	+	+	+
Вартість	+	-	-	+

1.4 Вимоги до програмної системи і до її компонентів

Вимоги до програмного забезпечення - це сукупність вимог і очікувань, які ставляться до програмного забезпечення в рамках проекту розробки. Ці вимоги описують те, що повинно бути зроблено програмним забезпеченням та як воно повинно поводитися у різних умовах. [4]

Вимоги до програмного забезпечення можуть бути функціональними і нефункціональними. Функціональні вимоги описують функціональність програмного забезпечення, тобто те, що воно повинно здійснювати. Наприклад, функціональна вимога до програмного забезпечення може звучати як "додавання нового користувача у систему". Нефункціональні вимоги описують

характеристики програмного забезпечення, такі як його продуктивність, безпека та надійність.

Функціональні вимоги визначають завдання, які програмне забезпечення має виконувати, описуючи його основні функції. Вони включають сценарії взаємодії користувача із системою, вимоги до обробки та збереження даних, а також вимоги до користувацького інтерфейсу та його функціональних можливостей.

Нефункціональні вимоги встановлюють вимоги до якості та характеристик програмного забезпечення, таких як продуктивність, безпека, надійність і масштабованість. Ці вимоги не стосуються основних функцій системи, але є критичними для забезпечення її ефективної та стабільної роботи.

1.4.1 Функціональні вимоги до системи. Функціональні вимоги системи зазвичай розглядаються з точки зору взаємодії користувача із системою. Функціональні вимоги зручно відобразити на діаграмі прецедентів, «use case» в мові UML (рис. 1.5).

Рис. 1.5 Діаграма прецедентів

В нашому випадку основна задача системи саме обробка зображення, а для користувача все повинно бути максимально просто. Отже користувач має мати змогу переглянути інструкцію яким чином зробити фото, та як завантажити його в систему. Після цього система сама створить файл та надасть його користувачу до завантаження. При цьому система повинна максимально добре працювати з файлами різної якості, та різним освітленням. Отже список функціональних вимог системи виглядає наступним чином:

1. Система повинна приймати файли зображень на вхід у форматі PNG.
2. Система повинна опрацювати вхідний файл із наступними вимогами:
    a. Необхідна підтримка двох алфавітів, найбільш поширених в Україні: кирилиця та латиниця.

b. Необхідно мати підтримку режиму налагодження. Тобто зберігати всі проміжні стани зображень для їх аналізу.

3. Після успішної роботи програми, файл формату .ttf повинен бути завантаженим до файлової системи користувача.

1.4.2 Нефункціональні вимоги до системи. Нефункціональні вимоги визначають якісні характеристики системи. Зазвичай, чим вищі вимоги до якості, тим кращим є продукт, але варто враховувати, що підвищення таких вимог збільшує вартість розробки. Оскільки ресурси обмежені, важливо визначити основні нефункціональні вимоги, які забезпечать необхідний рівень якості системи.

- Простий інтерфейс. Інтерфейс системи повинен максимально простим та містити усі необхідні інструкції.
- Адаптивність. Система повинна максимально адаптувати всі необхідні параметри для обробки зображень різних розмірів та якості.
- Точність. Система повинна максимально точно знаходити всі наявні символи та впізнавати їх незалежно від освітлення.
- Стійкість до помилок. У випадку будь яких помилок, системо повинна повідомляти користувача про суть проблеми та самостійно закінчувати процес виконання.

1.5 Розрахунок вартості розробки

Існує багато алгоритмічних моделей, які використовуються для прогнозування витрат, собівартості та створення графіка розробки програмного забезпечення. Однією з найцікавіших є модель COCOMO. Ця модель має детально розроблену технічну документацію та доступна для широкого використання. З моменту її появи у 1981 році модель зазнала значного розвитку, а її остання версія була опублікована у 1995 році.

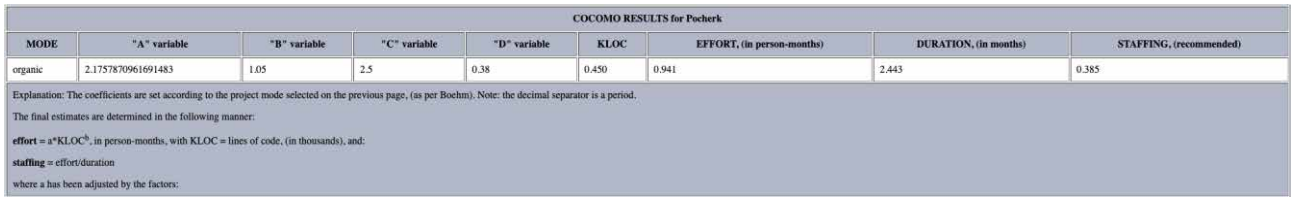
Модель COCOMO (Constructive Cost Model) призначена для оцінки часових витрат на виконання проєкту на основі визначених параметрів. Для розрахунків доступні численні інструменти, і для даного проєкту було обрано веб-додаток від NASA. Цей інструмент є простим у використанні та надійним. Посилання на використаний калькулятор: COCOMO Calculator.

Для оцінки було обрано стандартний режим розрахунку, оскільки проєкт має невеликий масштаб. Кількість рядків коду приблизно становить 450, що охоплює основну логіку системи. Параметри було підібрано, враховуючи, що проєкт є невеликим і не потребує значних ресурсів для реалізації. На рис. 1.6 зображено список обраних параметрів.

Product Attributes	
Required Reliability	0.75 (VL)
Database Size	0.94 (VL)
Product Complexity	1.15 (H)
Computer Attributes	
Execution Time Constraint	1.00 (VL)
Main Storage Constraint	1.00 (VL)
Platform Volatility	0.87 (L)
Computer Turnaround Time	0.87 (VL)
Personnel Attributes	
Analyst Capability	1.19 (L)
Applications Experience	1.00 (N)
Programmer Capability	1.00 (N)
Platform Experience	1.10 (L)
Programming Language and Tool Experience	0.95 (H)
Project Attributes	
Modern Programming Practices	1.00 (N)
Use of Software Tools	1.00 (N)
Required Development Schedule	1.08 (L)
New (Values are probably wrong)	
Required reusability	1.00 (L)
Documentation match to life-cycle needs	1.00 (VL)
Personnel continuity	1.10 (L)
Multisite development	1.00 (VL)

Рис. 1.6. Список параметрів моделі COCOMO

Результати калькуляції можна побачити на рис. 1.7.



COCOMO RESULTS for Pocherk

MODE	"A" variable	"B" variable	"C" variable	"D" variable	KLOC	EFFORT, (in person-months)	DURATION, (in months)	STAFFING, (recommended)
organic	2.1757870961691483	1.05	2.5	0.38	0.450	0.941	2.443	0.385

Explanation: The coefficients are set according to the project mode selected on the previous page, (as per Boehm). Note: the decimal separator is a period.

The final estimates are determined in the following manner:

effort = a*KLOC^b, in person-months, with KLOC = lines of code, (in thousands), and:

staffing = effort/duration

where a has been adjusted by the factors:

Рис. 1.7. Результати роботи онлайн калькулятора COCOMO

Основним показником, який нас цікавить, є тривалість розробки. Згідно з результатами оцінки, реалізація проєкту займе 2,5 місяці роботи.

Для оцінки вартості цього часу враховуємо, що над розробкою працюватиме один студент, оскільки робота є індивідуальною. Для розрахунку було використано відкриту статистику від ресурсу DOU (https://jobs.dou.ua/salaries/). Задавши параметри: досвід роботи — 6 років, мову програмування — Python, та позицію — Senior Software Developer, отримуємо медіанну зарплату у 4500 доларів на місяць. Помноживши цю суму на 2,5 місяці, отримуємо 11 250 доларів. На момент написання роботи курс долара становить 41,65 грн. Перевівши суму у гривні, отримуємо 468 562 грн — це базова оцінка вартості розробки проєкту.

Додатково підрахуємо витрати на електроенергію, яка буде використана для реалізації проєкту. Ноутбук, що використовується для роботи, оснащений літійполімерним акумулятором на 70 Вт·год (https://www.apple.com/ua/macbook-pro-14-and-16/specs/), якого, в середньому, вистачає на повний робочий день. З урахуванням тривалості проєкту у 2,5 місяці та середньої кількості робочих днів у місяці (21 день), загальна кількість робочих днів складає 53. Отже, загальні витрати електроенергії становитимуть:

70 Вт·год * 53 = 3710 Вт·год = 3,71 кВт·год .

На момент написання звіту тариф на електроенергію становить 4,32 грн за 1 кВт·год. Таким чином, витрати на електроенергію будуть:

3,71 кВт·год * 4,32 грн = 16 грн.

Інших витрат проєкт не передбачає, оскільки він є неприбутковим (благодійним) і не передбачає доходу. Податки також не сплачуються. У результаті загальна вартість проєкту становить приблизно 468 578 грн.

1.6 Постановка завдання

Цей проєкт має на меті розробити інтелектуальну систему автоматичного створення шрифтів на основі фотографій рукописного тексту користувача. Система повинна забезпечувати зручний і зрозумілий процес перетворення рукописного тексту у цифровий шрифт, мінімізуючи ручну участь користувача. Основні завдання:

1. Обробка зображення:
    - реалізувати алгоритми попередньої обробки фотографій (згладжування, бінаризація, видалення шумів).
    - розпізнавати контури символів на зображенні та розділяти їх на окремі елементи.
2. Векторизація символів:
    - перетворити растрові зображення символів у векторний формат для створення шрифту.
    - забезпечити точність передачі форм символів, зберігаючи унікальні риси рукопису.
3. Формування шрифтів:
    - генерувати файли шрифтів у стандартному форматі (TTF), сумісному із більшістю програм і платформ.
    - реалізувати інструменти для автоматичного налаштування метрик шрифту, таких як відступи між символами, висота рядків тощо.
4. Простота використання:
    - розробити інтуїтивно зрозумілий інтерфейс, що дозволить користувачам завантажувати зображення, налаштовувати параметри й експортувати шрифти.
    - зробити процес створення шрифту доступним навіть для людей без спеціальних знань у дизайні чи програмуванні.
5. Тестування та оптимізація:
    - перевірити систему на різних типах почерків, щоб забезпечити її універсальність.

- оптимізувати швидкість роботи системи та точність обробки зображень.

У результаті розробки має бути створена система, яка дозволяє користувачам швидко й ефективно створювати унікальні цифрові шрифти на основі рукописного тексту. Ця система повинна бути універсальною, легкою у використанні та здатною генерувати шрифти, які зберігають унікальні особливості почерку користувача. Вона матиме практичне застосування у сферах дизайну, маркетингу, освіти та інших галузях, де потрібна персоналізація тексту.

1.7 Висновки до розділу 1

У цьому розділі було розглянуто проблему, яку проєкт спрямований вирішити, а також обґрунтовано актуальність його розробки. Детально описано предметну область, пов’язану зі створенням шрифтів на основі почерку, та проведено аналіз існуючих програм-аналогів. На основі цього аналізу визначено їхні ключові переваги та недоліки, враховуючи специфіку предметної області та потреби користувачів.

Результатом аналізу стало формування вимог до системи, які поділяються на функціональні та нефункціональні. Функціональні вимоги охоплюють усі необхідні функції для розв’язання проблем користувачів, тоді як нефункціональні визначають мінімальні якісні показники, що дозволять створити конкурентоспроможний продукт з використанням обмежених ресурсів.

На основі сформованих вимог було чітко визначено постановку завдання для розробки програмного продукту. Це завдання передбачає реалізацію визначених функцій і характеристик системи відповідно до встановлених критеріїв.

Окрім цього, у розділі виконано оцінку вартості реалізації проєкту. Загальні витрати на розробку складають приблизно 468 тис. грн.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальна архітектура програмної системи

Виходячи із завдання на розробку та визначених вимог, можна спроєктувати загальну архітектуру додатка. Опис архітектури є важливим етапом, який допомагає зрозуміти структуру інформаційної системи та закладає основу для подальшої розробки.

Оскільки основна функціональна вимога користувацького інтерфейсу - простота у користуванні, та фактом, що візуальна складова наразі не є пріоритетом, було прийнято рішення обмежитись консольним інтерфейсом із достатньою кількістю інструкцій та описом.

Далі необхідно визначити як саме програма буде визначати написані користувачем символи на папері. Для цього можна розглянути два різні підходи:

1. Розпізнавання тексту за допомогою нейронної мережі
2. Пошук букв алфавіту за допомогою комп’ютерного зору

Першочерговою ідеєю був саме варіант із нейронною мережею, але дослідивши деталі та зваживши всі переваги та недоліки, було обрано другий варіант. Отже розглянемо детальніше кожен із підходів.

2.1.1 Розпізнавання тексту за допомогою нейронної мережі. Отже для цього користувачу достатньо було б лише сфотографувати написаний текст. Враховуючи, що текст може бути написаний раніше, наприклад сторінка із конспекту лекції, цей варіант може максимально простим. Але, потрібно також враховувати, що для повноцінного шрифту, необхідно мати в сканованому тексті всі можливі літери алфавіту. Зазвичай на одній сторінці малоймовірно зустріти всі можливі букви алфавіту, але також, як відомо, великі літери в письмі відрізняються від малих, отже також потрібно мати всі літери алфавіту великими літерами, що зустріти у випадково написаному тексті неможливо. Отже все-рівно

доведеться вимагати від користувача написати наперед продуманий текст великими та малими літерами.

Допустимо, користувач написав необхідний текст і на фото присутні всі літери алфавіту в реченнях. Завдання натренувати математичну модель для розпізнавання тексту не є новою, та існують безліч готових рішень для такої реалізації, але також є кілька нюансів:

1. Математичні моделі для розпізнавання рукописного тексту розповсюдження в основному для латиниці, для інших алфавітів відкритих даних для тренування моделей майже немає, або дуже мало.
2. Моделі для розпізнавання рукопису зазвичай працюють над вгадуванням слів, або інколи навіть всього речення. В даному випадку системі навіть недостатньо буде розпізнати кожну літеру окремо, система повинна також вирізати кожну окрему букву, тобто визначити межі цієї букви в тексті. Для того щоб натренувати модель на вирізання окремих літер з речення та визначення що це за літера, потрібна велика кількість прикладів, яких наразі у відкритому доступі просто не існує.

Отже даний метод потребує великої кількості прикладів розпізнавання та вирізання літер із рукописного тексту різними мовами, яких наразі недостатньо у відкритому доступі.

Однією із переваг даного методу є те, що літери в тексті мають більш природні з’єднання та закінчення між літерами ніж в алфавіті. Хоча з іншого боку ці з’єднання та закінчення можуть погано поєднувати з іншими літерами та виглядати навіть гірше ніж просто окремі букви без з’єднань.

2.1.2 Пошук букв алфавіту за допомогою комп’ютерного зору. Даний підхід вимагає від користувача написати на папері всі бажані символи в алфавітному порядку (див. рис. 2.1).

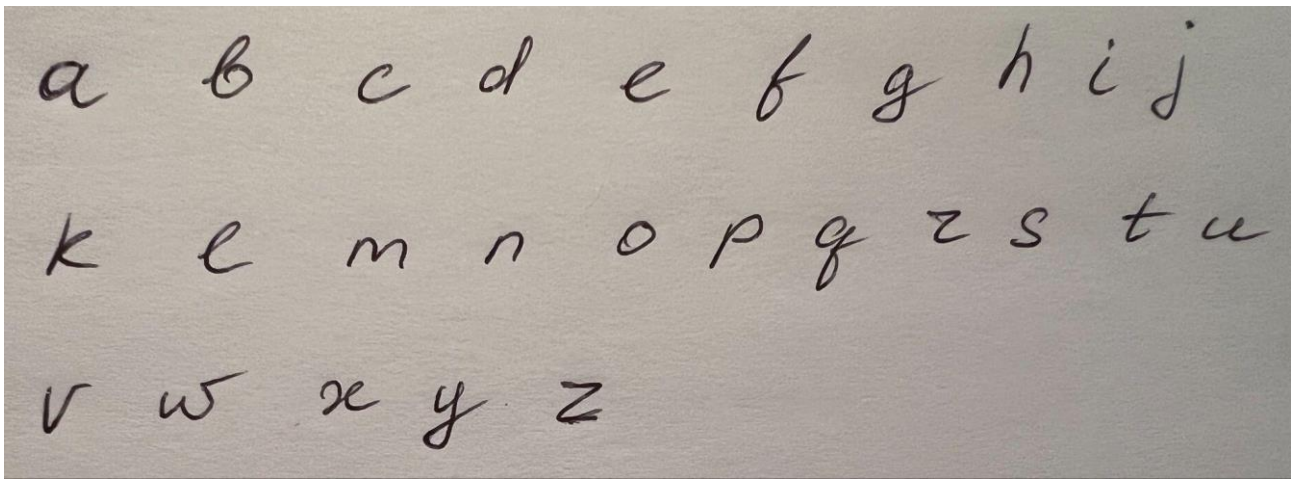


Рис. 2.1 Приклад фото алфавіту користувача

Важливо також зазначити, що папір повинен бути повністю білий без додаткових ліній. Завдяки наперед визначеному порядку символів, програмно є можливість визначити де знаходиться яка літера без необхідності впізнавання літер, а отже і без необхідності тренувати нейронну мережу.

Натомість необхідно розробити алгоритм, який буде знаходити контури та за координатами визначати порядок літери. Маючи визначений контур із співвіднесеним індексом літери в алфавіті вирізати його із зображення не є проблемним.

З іншого боку цей підхід передбачає певні складнощі та обмеження порівняння із розпізнаванням за допомогою нейронної мережі. Наприклад, фото може мати різний розмір, літери можуть знаходитись на різних відстанях, літери в рядку не завжди будуть трохи відрізнятись за віссю у та загалом вимоги до зображення відчутно збільшуються.

Також такий підхід дозволяє не обмежуватись лише почерком людини, в алфавітному порядку можна малювати будь які символи. Ці символи можуть навіть зовсім не бути схожими на реальні літери, а повністю відображати фантазію користувача.

Загальну схему роботи системи можна розглянути на рис. 2.2.

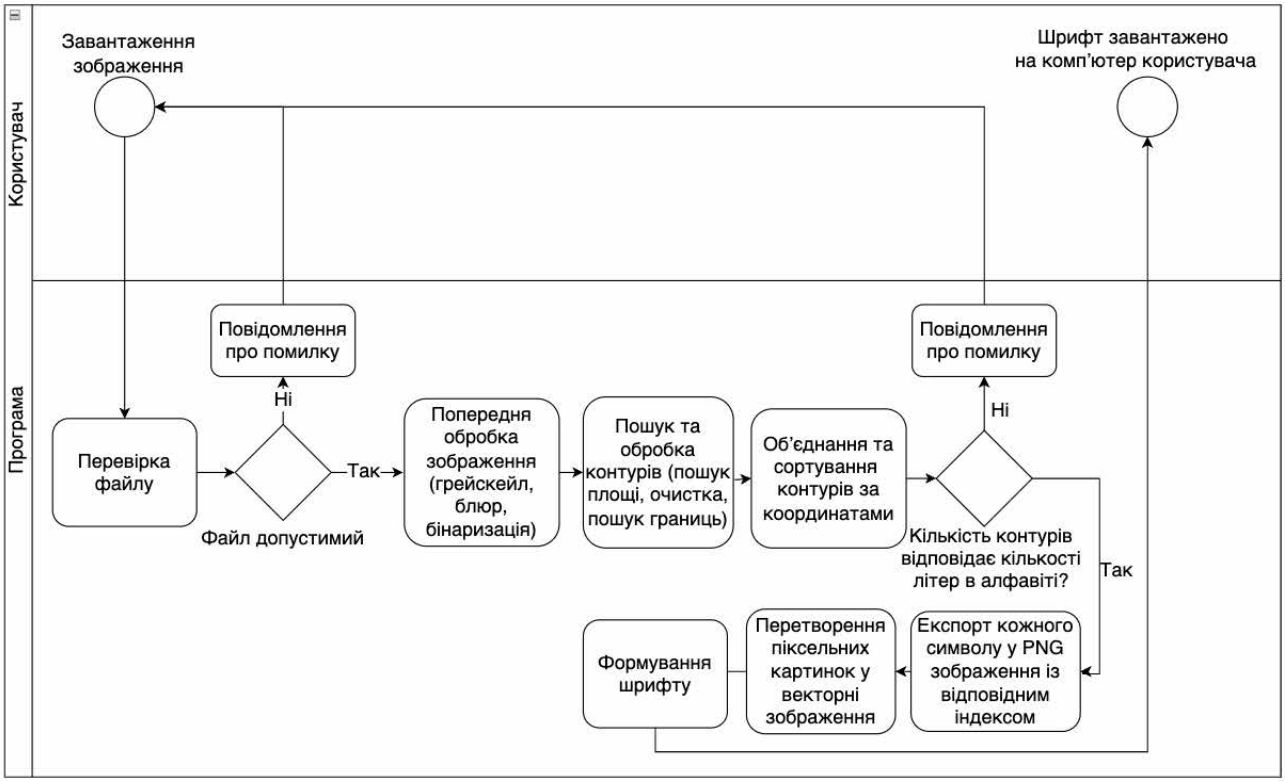


Рис. 2.2 Загальна схема роботи системи

Дана діаграма є прикладом BPMN (Business Process Model and Notation) діаграми, та зазвичай покликана моделювати саме бізнес процес тієї чи іншої системи. У випадку розроблюваної системи, така діаграма дозволяє нам показати одночасно взаємодію системи із користувачем, та загальний алгоритм того, що відбувається в процесі роботи програми.

2.2 Вибір технологій для реалізації програмного забезпечення

Отже для обраного підходу, необхідно використовувати алгоритми та методи комп'ютерного бачення. Найбільш розповсюдженою бібліотекою для комп'ютерного бачення - є OpenCV [4]. Дана бібліотека написана мовою C++, але також має інтерфейси на Python та JavaScript. Через персональний досвід розробки застосунків мовою Python, та великою кількістю доступних бібліотек та інтеграцій, Python було обрано як основну мову для розробки системи [5].

Також Python має бібліотеку Numpy, яка дає можливість зручно працювати з растровими зображеннями, та виконувати різні математичні операції з ними.

Для перетворення растрового зображення у векторне, найкраще підходить мультиплатформна програма Potrace [6]. Дана програма не має окремих бібліотек, отже її доведеться використовувати саме через консольний інтерфейс.

Для допомоги в створенні файлу шрифта існує додаток FontForge, який також має бібліотеку для Python та дозволяє автоматизувати всі дії із шрифтом.

2.3 Діаграма класів та кооперацій

Діаграми класів відображають різні класи, які складають систему, та їхні взаємозв'язки. Їх відносять до "статичних діаграм", оскільки вони демонструють класи разом із методами та атрибутами, а також статичні зв'язки між ними. На таких діаграмах показано, які класи "знають" про інші класи або є їхніми складовими, проте не деталізується, які саме методи викликаються у процесі взаємодії.

Переваги діаграм класів:

- Вони дозволяють визначити, які класи та об'єкти необхідні для реалізації функціоналу системи або додатку.
- Допомагають зрозуміти залежності між класами та об'єктами, а також визначити необхідні методи й атрибути для їхньої взаємодії.
- Забезпечують гнучкість у модифікації та розширенні структури програми, адже зміни в діаграмі легко перенести в код.
- Служать основою для створення програмного коду та документації, спрощуючи процес розробки.

В досліджуваній системі можна чітко виділити два окремі процеси, які можна відобразити у вигляді кооперацій: визначення літер із фото та створення шрифту.

2.3.1 Кооперація "визначення літер із фото". Ця кооперація класів відповідає за обробку фото користувача та визначення літер в алфавітному порядку з нього. Після завантаження зображення даний процес виділить необхідні контури та збереже їх до окремих файлів. На рис. 2.3 зображено діаграму класів для даної кооперації:

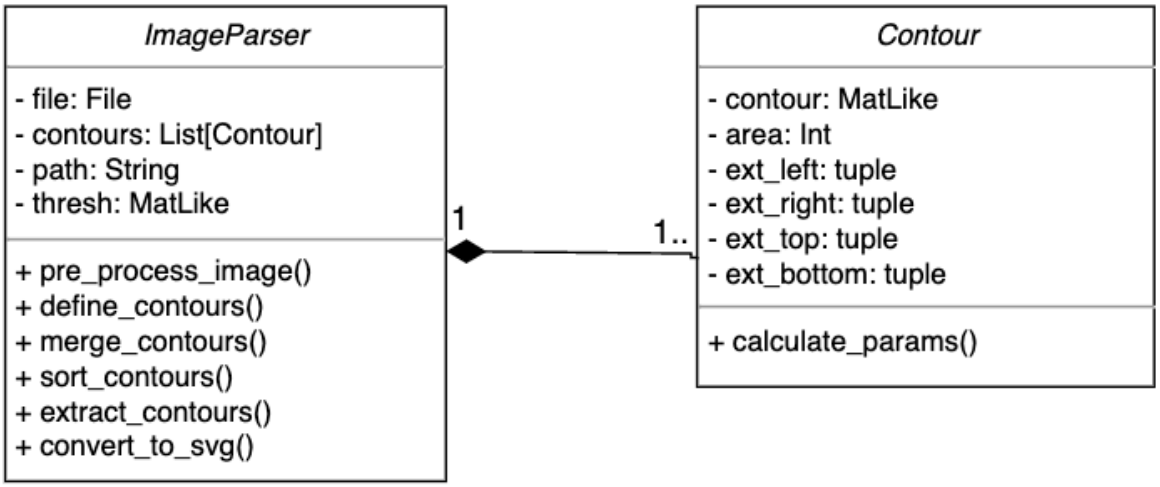


Рис. 2.3 Діаграма класів для кооперації “визначення літер із фото”

На даній діаграмі зображено 2 класи:

- ImageParser - основний клас процес, який займається обробкою зображенням, визначенням контурів, їх сортуванням та збереженням.
- Contour - клас, який містить всю необхідну для контуру інформацію, та метод для обчислення.

2.3.2 Кооперація “створення шрифту”. Даний процес призначений для того, аби створити кінцевий результат роботи системи, а саме шрифт на основі зображень із попередього процесу. Переглянути діаграму класів для цього процесу можна на рис. 2.4.

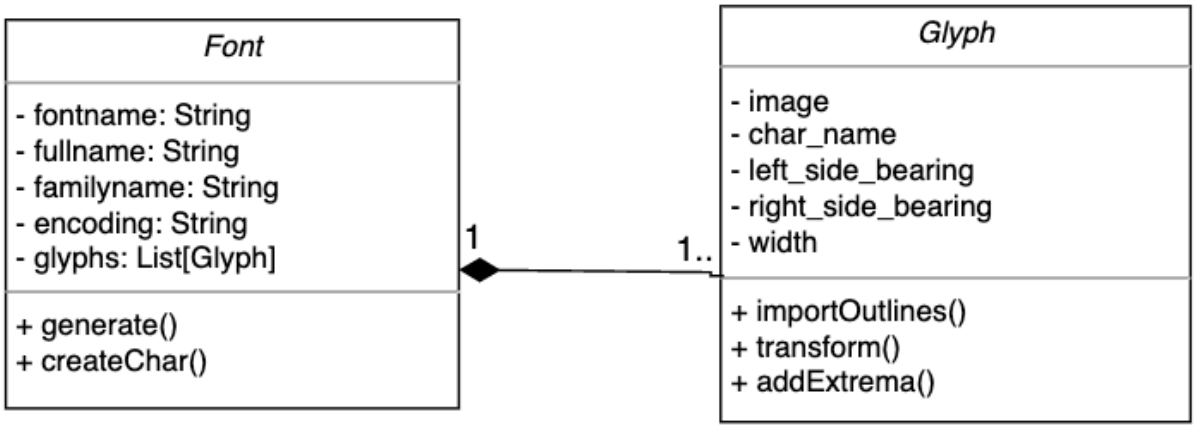


Рис. 2.4 діаграма класів для кооперації “створення шрифту”

Ця діаграма демонструє наступні класи:

- Font - клас, який зберігає в собі всю необхідну інформацію та функціонал для створення шрифта.

- Glyph - клас, який відображає один гліф в шрифті. Містить всю інформацію про символ, враховуючи його векторне представлення. Також має методи для роботи з цим гліфом.

2.4 Діаграма компонентів

Діаграми компонентів UML відображають концептуальну взаємодію між різними системами, охоплюючи як логічне, так і фізичне моделювання. Компоненти в UML є автономними та модульними елементами системи, які можна замінити на альтернативні, зберігаючи при цьому загальну структуру. Вони здатні містити конструкції будь-якої складності, залишаючись самодостатніми. Взаємодія між компонентами здійснюється виключно через інтерфейси, що забезпечує чітке розмежування їхніх функцій.

Кожен компонент має власний інтерфейс і, водночас, може використовувати операції та сервіси інших компонентів через їхні інтерфейси. На діаграмі компонентів ці інтерфейси відображають зв’язки та залежності, що ілюструють архітектуру програмного забезпечення.

Діаграма компонентів застосовується для представлення архітектури програмної системи з точки зору її фізичних елементів. Компоненти виступають як логічні одиниці з чітко визначеними функціями, які можуть існувати незалежно одна від одної. На рис. 2.5 зображено приклад діаграми компонентів, створеної для цього проєкту.

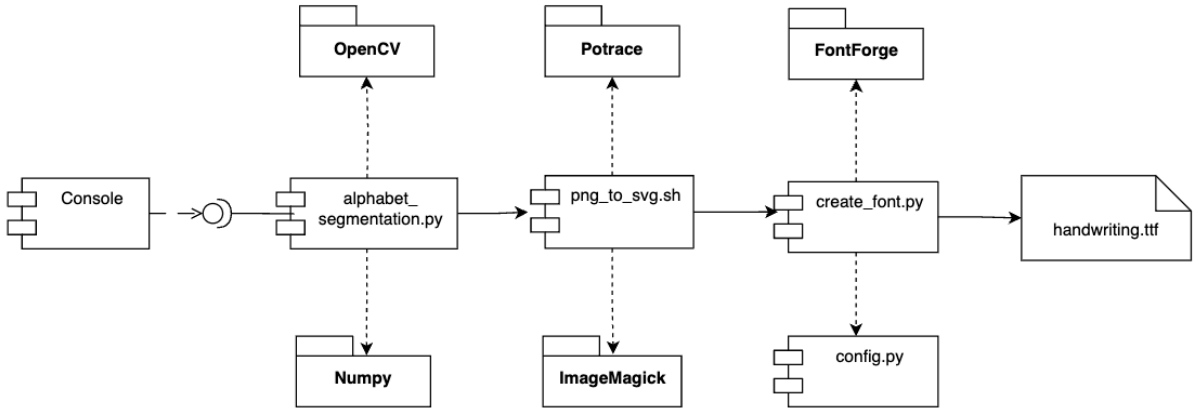


Рис. 2.5. Діаграма компонентів

Ця діаграма відображає всі ключові компоненти проектованої системи. На ній відсутнє розділення на компоненти виконання та розгортання, оскільки програмне забезпечення створюється на мові програмування Python. Як інтерпретована мова, Python не вимагає компіляції, що спрощує процес розробки та використання.

Розглянемо кожен компонент окремо:

- Console – як було згадано раніше, система наразі не передбачає графічного інтерфейсу взаємодії, натомість запуск команд здійснюється через консоль користувача.
- alphabet_segmentation.py – компонент для виділення символів із фото користувача в алфавітному порядку. Використовує бібліотеки OpenCV та Numpy
- OpenCV - бібліотека із відкритим кодом, яка містить необхідні класи на функції для реалізації необхідних алгоритмів обробки зображення.
- Numpy – бібліотека для математичних операції із багатовимірними масивами. Забезпечує швидку обробку даних, зокрема для маніпуляцій із пікселями зображень.
- png_to_svg.sh - скрипт на мові bash, який відповідає за перетворення всіх зображень символів у векторні файли. Для цього спочатку йде зміна

формату файлу за допомогою програми ImageMagick, а потім сама векторизація за допомогою програми Potrace.

- ImageMagick - окрема програма, яка містить безліч функціоналу для роботи із зображеннями, але в даному випадку ми використовуємо її лише для перетворення зображення із формату PNG у формат PBM.
- Potrace - програма для векторизації зображень. Автоматична та самостійна програма, яка приймає на вхід растрове зображення та повертає векторне у форматі SVG.
- create_font.py - модуль для створення та формування кінцевого шрифту. Працює із файлами виділеними на попередніх етапах, та використовує бібліотеку FontForge.
- FontForge - бібліотека, яка містить велику кількість функцій для створення та редагування шрифтів.
- config.py - файл конфігурації. Містить статичні дані для конфігурації алгоритму створення шрифту.

2.5 Висновки до розділу 2

У другому розділі докладно розглянуто проектування інформаційної системи. На початку було представлено загальний опис архітектури системи, де обґрунтовано ключові проєктні рішення, зокрема відмову від створення візуального інтерфейсу на користь консольному. Це рішення було ухвалено для оптимізації витрат і спрощення процесу розробки.

У розділі також наведено огляд програмних технологій, використаних у розробці, з детальною аргументацією їх вибору. Основними технологіями стали мова програмування Python та бібліотеки OpenCV, Potrace, FontForge, які забезпечують ефективність і гнучкість у реалізації програмного забезпечення.

Подальше наповнення розділу присвячене моделюванню інформаційної системи за допомогою UML-діаграм. Серед представлених діаграм — BPMN

діаграма, діаграма класів із кооперaціями і діаграма компонентів. Ці моделі дозволяють візуалізувати систему з різних рівнів абстракції та точок зору, що сприяє глибшому розумінню її структури та логіки роботи.

3 ДОСЛІДЖЕННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка алгоритму вирізання символів

Для початку розглянемо процес обробки зображень та пошуку літер. Даний процес є ключовим на етапі до створення шрифту та містить найбільшу кількість потенційних проблем та нюансів.

3.1.1 Попередня обробка зображення. Більшість процесів, які виконують обробку зображень, первинного його, якимось чином обробляють. Це може бути корекція кольорів, застосування певних фільтрів чи покращення контрасту, шумозаглушення, зміна розмірів зображення або налаштування яскравості, щоб забезпечити кращу якість для подальшої обробки. Основним завданням всіх цих процесів є підготувати, тобто зробити зображення більш придатним до основної обробки, щоб забезпечити максимальну точність аналізу та виділення необхідних деталей.

У випадку розроблюваної системи, варто відмітити, що основними об’єктами з якими доведеться працювати будуть контури. Контур — це сукупність точок, що визначають межу об’єкта на зображенні, які окреслюють його форму та основні риси. Контури дають змогу виділити та ідентифікувати окремі елементи зображення, що є ключовим етапом для подальшого розпізнавання та векторизації символів.

Контури вирізняються через пошук різких змін кольорів і найкращим зображенням для пошуку контуру буде саме чорно-біле зображення. Чорно-біле зображення в звичному сприйнятті містить лише відтінки сірого, хоча за назвою, можна сплутати з двокольоровим зображенням, яке містить лише чорний та білий колір. Отже щоб зробити зображення повністю сірим, достатньо лише помножити RGB канали кожного пікселя на наступні коефіцієнти: 0.2989, 0.5870, 0.1140. На рис. 3.1 зображено фотографію після згаданої операції.

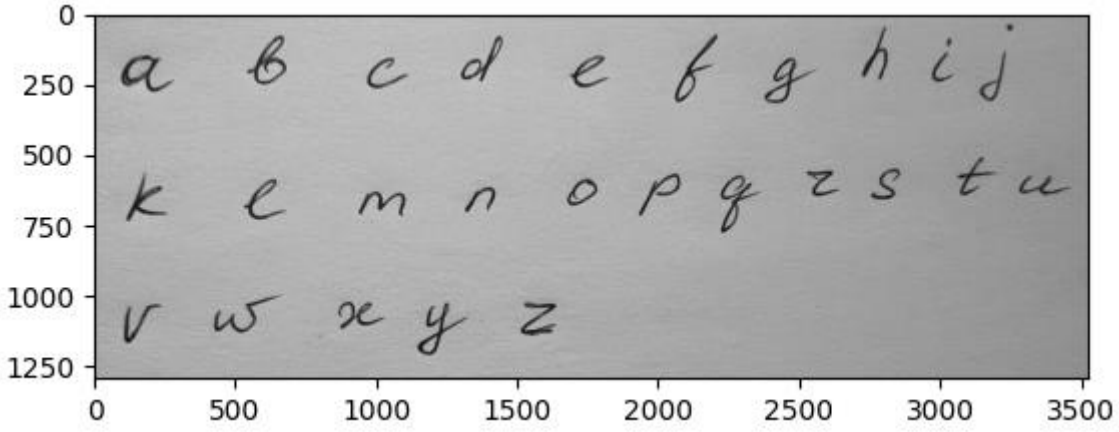


Рис. 3.1 Чорно-біле відображення вхідного фото

Після перетворення зображення у чорно-біле представлення, застосуємо Гауссове згладжування. Загалом згладжування - це досить поширений метод попередньої обробки зображення. Даний метод створює ефект розмивання, та робить зображення менш чітким. На перший погляд може виникнути питання: навіщо робити фото почерку менш чітким? Однак відповідь досить проста: що менше чіткості має зображення, то менше контурів на ньому буде знайдено. Отже, згладжування може допомогти нам позбутися перепадів кольорів у місцях, де вони не потрібні, наприклад, у тінях або дрібних крапках на фото.

Наступним етапом після згладжування, відбуватиметься один із найголовніших етапів попередньої обробки, а саме - бінаризація. Бінаризація - це перетворення зображення на бінарне (двійкове), тобто таке, що має лише два кольори. В нашому випадку нам потрібно відокремити чорний від білого, що якраз і допомагає зробити бінаризація. Двійкове зображення містить лише кольори 255 (білий) та 0 (чорний), отже для даної функції необхідно визначити конкретний поріг вище якого, колір буде відноситись до білого, а нижче - до чорного. Спробуємо наприклад поріг кольору 50 - результат зображено на рис. 3.2

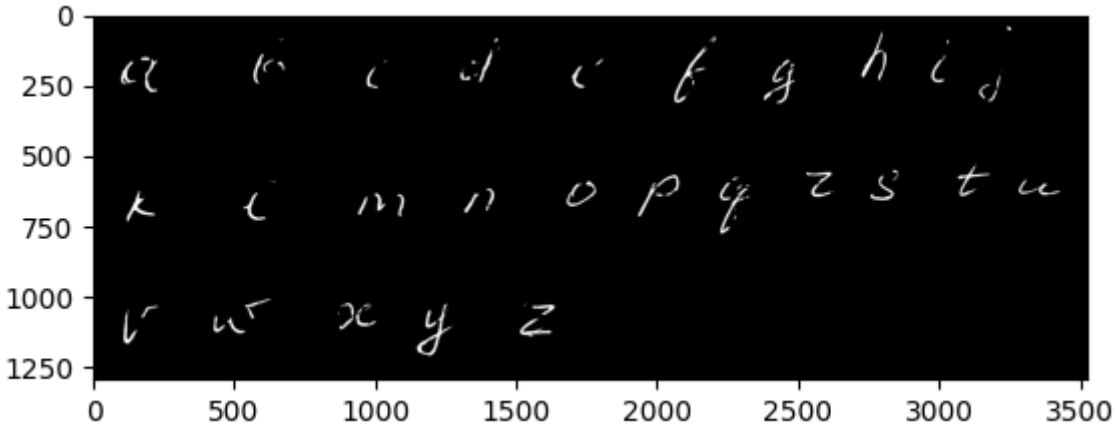


Рис. 3.2 Двійкове зображення із пороговим значенням 50

На цьому зображенні ми бачимо, що контури літер погано проглядаються і відповідно розрізнити їх програмно, буде неможливо. Передбачаючи фото різної якості та різного освітлення, визначити одне єдине порогове значення для всіх користувачів навряд буде хорошою ідеєю. Нащастя для цього існують відповідні алгоритми, які дозволяють динамічно визначати поріг кольору. Одним із методів бінаризації зображення є метод Оцу, заснований на алгоритмах кластеризації. На рис. 3.3 зображено результат роботи даного методу.

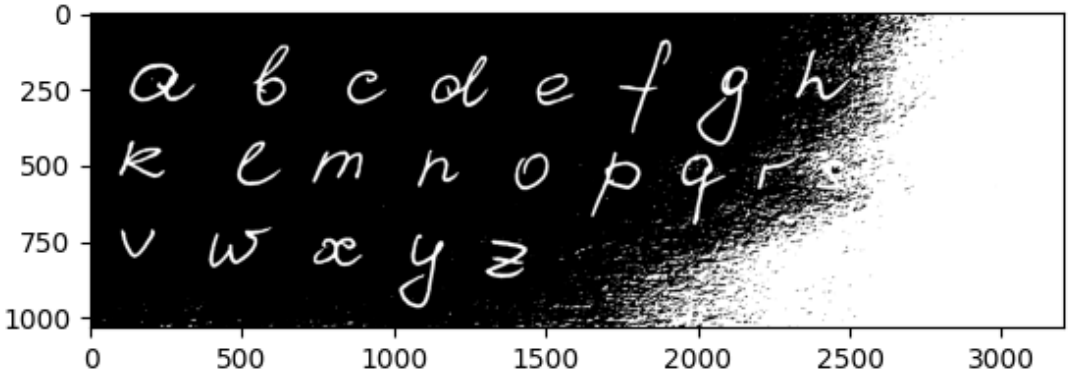


Рис 3.3 Двійкове зображенням після застосування методу Оцу

Варто зауважити, що дане зображення є результатом обробки іншого фото. Це зроблено для наглядності, оскільки попереднє фото абсолютно нормально виглядає після використання методу Оцу, але тут саме через різницю в освітленні, помітно, що даний метод не буде завжди працювати. Іншим можливим методом є метод трикутника, який визначає середнє порогове значення на основі геометричної формули. На рис. 3.4 зображено результат роботи даного методу.

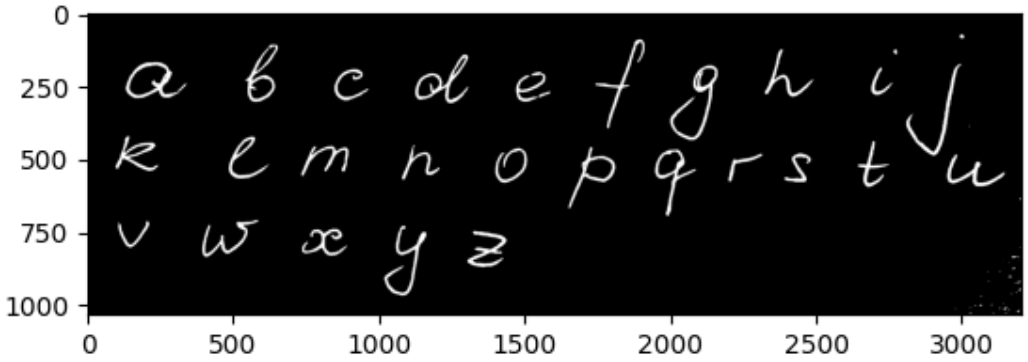


Рис 3.4 Двійкове зображення після застосування методу трикутника

Цей метод і був обраний як остаточний, оскільки в ході експерименту він показав кращий результат.

3.1.2 Пошук та обробка контурів. Після того, як ми отримали двійкове зображення хорошої якості, завдання пошуку контурів полягає лише у виділенні білих пікселів на чорному фоні. Але також для спрощення та оптимізації роботи системи, часто застосовують алгоритми апроксимації, які дозволяють не зберігати всі пікселі контуру, а лише деякі, які будуть самі лінії.

Після знаходження всіх контурів, була знайдена суттєва проблема - крапки над літерами є окремими контурами, які необхідно об’єднати із основами літер. Для цього необхідно визначити загальну площу контуру та координати екстремальних точок для подальшої обробки. Екстремальні точки - це найбільш крайні точки контуру зліва, справа, згори та знизу. Також при визначенні площі контурів є можливість відкидати над малі контури, які можуть випадково потрапити до загального списку контурів.

Маючи площу та екстремальні координати, можемо реалізувати алгоритм, який буде виявляти крапки над літерами та об’єднувати їх із основами. Для цього поступово шукаємо контури з малою площею - це і будуть крапки. Знаходимо відстань між нижньою екстремальною точкою кожної крапки та верхньою екстремальною точкою великого контуру (літери). Якщо ця відстань менша ніж певна визначена величина - крапка відноситься до літери, отже контури можна об’єднати. В даному випадку кожен контур зберігається як двовимірний масив,

або матриця, отже об’єднанням контурів по суті є об’єднання масивів. На рис. 3.5 зображено приклад об’єднання двох контурів - крапки та основи літери

Рис. 3.5 приклад об’єднання крапки та основи літери

Бачимо, що об’єднання працює правильно.

3.1.3 Сортування контурів та експорт в окремі зображення. Оскільки для даного дослідження було вирішено не використовувати нейронні мережі, одним із найважливіших завдань залишається правильно визначити літери на зображенні. Як було описано раніше, ключовим в даному випадку є саме те, що літери написані в алфавітному порядку, що в свою чергу дає змогу зрозуміти яка саме це літера лише за координатами її місцерозташування на фото. Для цього, нам потрібно посортувати отримані в попередніх пунктах контури в тому ж порядку, що вони розташовані на зображені. Нажаль процес визначення контурів не залишає їх порядок, отже сортування доведеться реалізувати додатково.

Оскільки на зображенні всі літери знаходяться в алфавітному порядку, можна візуально розділити картинку на стовпчики та лінії, але варто також пам’ятати, що алфавіт написано на білому папері без додаткової розмітки чи чогось подібного. Отже в більшості випадків контури не будуть формувати чіткі лінії, а їх координати можуть бути відчутно віддалені від цих уявних ліній.

Отже для сортування нам необхідно для початку посортувати всі контури за віссю Y верхньої екстремальної точки. Таким чином ми можемо проходячи по кожному контуру, бути впевненими, що попередній контур знаходиться в тому самому рядку, якщо лише їх верхні екстремальні точки не мають суттєвої відмінності за Y координатою. Якщо ж різниця координат суттєва - це означає, що почався наступний рядок. Після того, як суцільний рядок контурів виділено

та знайдено перехід на наступний рядок, контури в рядку можна сортувати за віссю X. Таким чином, порівнюючи лише координати сусідніх контурів, ми даємо можливість літерам відхилятись від чіткої горизонтальної лінії, та при цьому досить впевнено можемо визначити порядок. Навіть якщо літери будуть написані навкоси, або зображення матиме нахил, сортування буде працювати.

Після сортування кожен контур вирізається із загального зображення, відбувається інверсія кольору для того аби мати чорні літери на білому тлі, та зберігається в окремий .png файл, в назві якого записується індекс літери в алфавіті.

Таким чином, незалежно від самих символів, їх розбірливості та правильності - літери будуть посортовані у алфавітному порядку, що в подальшому дасть нами можливість з легкістю визначати що саме це за літера. Тобто даний алгоритм дає можливість не обмежуватись лише правильними символами, можна створювати абсолютно різні дизайнерські шрифти, які нейронна мережа ніколи б не розпізнала

3.2 Створення шрифту

Після того, як кожен символ із зображення вирізано в окремий файл, система має можливість створити на їх основі шрифт. Як вже раніше було зазначено - файл шрифту містить в собі таблицю із необхідними символами у вигляді векторних зображень. Отже для створення шрифту нам спочатку необхідно отримати векторні зображення кожного символу.

3.2.1 Перетворення растрових картинок у векторні. Для початку розглянемо різницю між растровими зображеннями та векторними.

Растрові зображення - складаються із певної кількості пікселів, кожен з яких має певний колір. Цей тип картинок є найбільш розповсюдженим в більшості сфер, оскільки дає можливість досить точно відображати найменші деталі. Також растрові зображення досить просто виводити на цифрові екрани, оскільки останні також складаються із пікселів. Але даний тип зображень майже

не використовується для побудови шрифтів. Основною причиною цього є те, що растрове зображення дуже складно збільшити чи зменшити без втрати якості. Зміна розміру у растрових зображень відбувається за допомогою алгоритмів інтерполяцій. Найбільш точним алгоритмом інтерполяції наразі є Super-Resolution, побудований на передбаченні нових пікселів за допомогою машинного навчання. У випадку шрифтів, ми можемо одномоментно змінити розмір шрифту в усьому великому файлі, відповідно можна уявити які потрібно ресурси для того, аби кожен символ було збільшено за допомогою алгоритму машинного навчання. Також великим недоліком растрових зображень є їх розмір на диску.

Векторні зображення – це графічні об’єкти, що створюються за допомогою математичних формул. На відміну від растрових зображень, які складаються з пікселів, векторні зображення описуються через геометричні примітиви, такі як точки, лінії, криві та багатокутники. Ці примітиви базуються на математичних координатах, що дозволяє легко масштабувати векторні зображення без втрати якості. Також такі зображення займають значно менше місця на диску, та відповідно файл шрифту має незначну вагу. Одним із найбільш розповсюдженим форматом векторних зображень є .svg.

Scalable Vector Graphics (скорочено SVG) (з англ. масштабована векторна графіка) — специфікація мови розмітки, що базується на XML, та формат файлів для двомірної векторної графіки, як статичної, так і анімованої та інтерактивної. SVG може бути виключно декларативним, або містити описи сценаріїв. Зображення можуть містити зовнішні лінки шляхом застосування простих XLink-ів [7].

Як вже раніше згадувалось, для векторизації зображень було використано додаток із відкритим вихідним кодом - Potrace. Насправді завдання векторизації є досить складним, та по своїй суті включає більшість кроків, які було описано в попередніх пунктах. Складність перетворення растрового зображення у векторне полягає у різниці між цими типами графіки. Растрові зображення зберігають дані про кожен піксель, але не мають інформації про структуру або форми об’єктів.

Під час векторизації алгоритм повинен “зрозуміти”, де закінчуються одні елементи зображення і починаються інші, виділити їх контури та визначити геометричну структуру. Приклад, яким чином Potrace перетворює растрове зображення у векторне можна переглянути на рис. 3.6

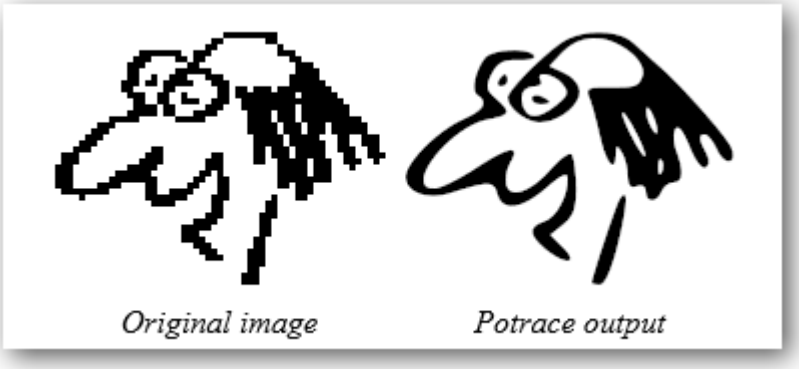


Рис. 3.6 Приклад роботи Potrace

Але, нажаль Potrace немає можливості приймати на вхід зображення формату .png, натомість нам доведеться виконати ще одне перетворення зображення у формат .pbm. Даний формат є підтримує лише двійкові зображення, тобто зображення з двома кольорами, що абсолютно підходить для зображень символів. Для перетворення зображення у потрібний формат було використано бібліотеку ImageMagick.

Варто також зазначити, що Potrace та ImageMagick є консольними додатками, отже досліджувана система може взаємодіяти з ними лише викликаючи під процеси. Отже для кожного файлу символу необхідно спочатку викликати процес перетворення його у вірний формат, а потім використати програму векторизації.

3.2.2 Формування файлу шрифту. Як вже раніше згадувалось, шрифт — це досить складний файл, який в певній мірі можна навіть назвати програмою. Це пов’язано з тим, що сучасні формати шрифтів, такі як TrueType (TTF) або OpenType (OTF), не лише зберігають інформацію про форму кожного символу, але й включають додаткові метадані та інструкції для їх правильного відображення.

Файл шрифту містить набір гліфів, кожен з яких відповідає окремому символу алфавіту, цифрам чи спеціальним знакам. Крім геометричних даних про форму символів, шрифт зберігає інформацію про їх розташування, відступи між ними (кернінг), висоту рядків та базову лінію, що дозволяє забезпечити узгодженість у вигляді тексту під час його відображення.

Більше того, шрифтові файли можуть містити спеціальні інструкції для різних операційних систем і програм, які описують, як обробляти й відображати символи. Наприклад, для TrueType шрифти включають байткод інструкцій, що визначають рендеринг кожного символу на різних роздільностях. У OpenType файли можуть бути інтегровані додаткові таблиці, такі як GSUB (для лігатур і альтернативних форм символів) та GPOS (для тонкого налаштування позиціонування символів).

Також шрифти підтримують багатомовність, зберігаючи символи для кількох мов і алфавітів. Важливою частиною є також таблиця відповідності символів (кодування Unicode), яка дозволяє кожному гліфу відповідати конкретному коду, що забезпечує коректне відображення тексту у різних системах.

Як було зазначено раніше, для автоматизації створення шрифту, система використовує бібліотеку FontForge, яка містить безліч різних інструментів для роботи із шрифтами. При створенні шрифту за допомогою цієї бібліотеки, спочатку необхідно присвоїти назву кінцевого файлу, визначити сімейство шрифта та його кодування:

```
font = fontforge.font()
font.fontname = "handwriting"
font.fullname = "Custom Handwriting Font"
font.familyname = "CustomHandwriting"
font.encoding = "UnicodeFull"
```

Далі в об’єкт font можна додавати раніше виділені файли svg. Але для коректного відображення шрифту недостатньо лише завантажити файл кожного символу. На рис. 3.7 зображено приклад друку шрифта без додаткової обробки:

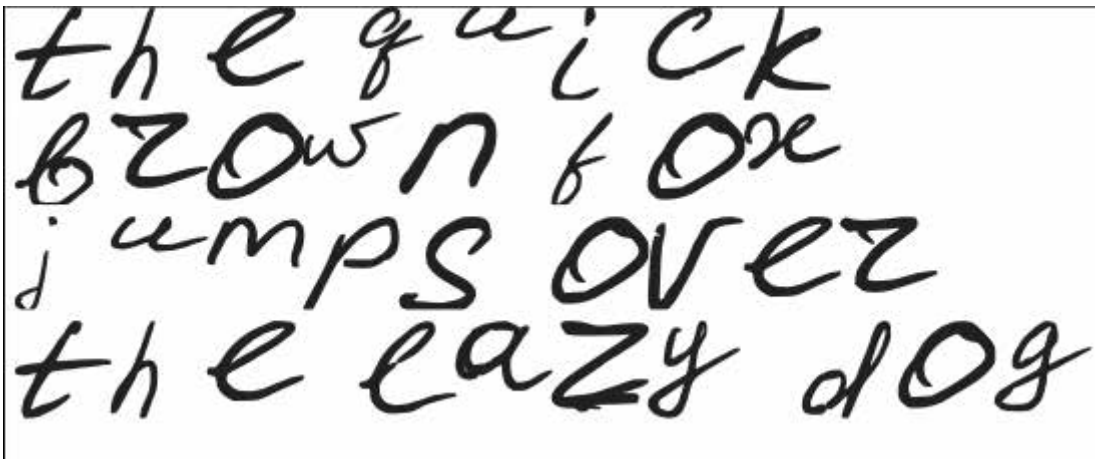


Рис. 3.7 Приклад друку шрифта без додаткової обробки

З прикладу можна побачити, що всі літери мають абсолютно різний розмір. Це зумовлено тим, що при конвертації растрового зображення у векторне, було примінено однакове співвідношення сторін для всіх окремих літер. Тобто, якщо літера “u” - широка і низька, її буде зменшено до тієї ж ширини, що і літера “i”. Отже в процесі побудови шрифта, необхідно також нормалізувати розмір кожного символу.

Також із попереднього рисунку, можна помітити, що деякі літери знаходяться на досить великій відстані одна від одної, а деякі, навпаки, притуляються до сусідів. Це зумовлено тим, що рамки кожного гліфу зазвичай знаходяться там де й сама крайня точка цього гліфу. Відповідно, якщо крайня точка першої літери знаходиться десь зверху, а початок наступної літери, знизу - між літерами здаватись що є великий простір. Для того, щоб зменшити цей ефект, існує процес під назвою “кернінг”.

Кернінг — процес зміни розмірів між буквених пропусків (інтервалів) між сусідніми буквами для поліпшення зовнішнього вигляду і легкості читання тексту. Цей параметр відповідає за індивідуальну роботу з кожною буквою і підбір її місцеположення залежно від вибраного шрифту, малюнка самої букви та її сусідніх букв, смислового навантаження слова і т. д. Значення кернінгу встановлюється у відсотках ширини пробілу. Більшість шрифтів мають регламентовані між літерні інтервали, так звані апроші. Але тим не менше, цей параметр потребує редагування. Зазвичай використовуються таблиці кернінгу, які є утилітами програм верстки [8]. Приклад зображено на рис. 3.8

Рис. 3.8 Приклад редагування кернінгу шрифта

Отже для кращого поєднання між літерами, також варто попрацювати із кернінгом. Беручи до уваги, що у кожної людини літери різні, та відповідно кернінг потрібен різний, а також враховуючи велику кількість можливих комбінацій дотику різних літер, передбачити якийсь універсальний алгоритм налаштування кернінгу, майже неможливо. Тож в даному проєкті було прийнято рішення обмежитись стандартними налаштуваннями кернінгу в шрифті.

Наступною знайденою проблемою, є те, як саме розташовані символи в своїх комірках. З прикладу вище бачимо, що літери “скакають” і не тримаються якоїсь більш менш чіткої лінії. Також, всім відомо, що літери часто можуть виходити за вертикальні межі своїх комірок. Такі виходи за рамки також називають верхнім або нижнім виносами. Наприклад буква “у” має “хвостик” знизу, яка виходить за базову лінії тексту. Отже система також повинна передбачати правильне положення літер відносно базової лінії. Для цього необхідно визначити, які літери мають нижній винос та яким цей винос зазвичай буває. Після цього можна передбачити зміщення таких літер вниз на визначену величину.

3.3 Демонстрація роботи програми.

Як було зазначено раніше, система наразі не передбачає візульного інтерфейсу, а вся взаємодія із програмою відбувається через консольні команди. Для того аби запустити роботу програми достатньо викликати перий компонент системи через команду “python3 alphabet_segmentation.py”. Приклад роботи програми можна побачити на рис. 3.9

```
volodymyr@AppleOsX-1300-T452960J6H HandwrittenFont % python3 alphabet_segmentation.py
Extracted 26 letters.
Conversion complete.
Copyright (c) 2000-2023. See AUTHORS for Contributors.
 License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
 with many parts BSD <http://fontforge.org/license.html>. Please read LICENSE.
 Version: 20230101
 Based on sources from 2023-01-01 05:27 UTC-D.
Core python package 'pkg_resources' not found: Cannot discover plugins
Font created successfully and saved to custom_handwriting.ttf
```

Рис. 3.9. Приклад запуску програми

Як можна побачити зі скріншоту, програма в даній конфігурації знайшла та вирізала 26 літер, після чого завершилась конвертація (з PNG в SVG формат), далі програма виводить інформацію про ліцензію додатку FontForge, та попередження про відсутність певних неважливих пакетів. В результаті бачимо повідомлення, що шрифт успішно створено та завантажено до вказаного файлу. Після цього файл можна знайти у відповідній директорії.

Результуючий файл має формат .ttf та його можна відкрити за допомогою відповідного редактора. Ось як це виглядає у візуальному інтерфейсі редактора (рис. 3.10.)

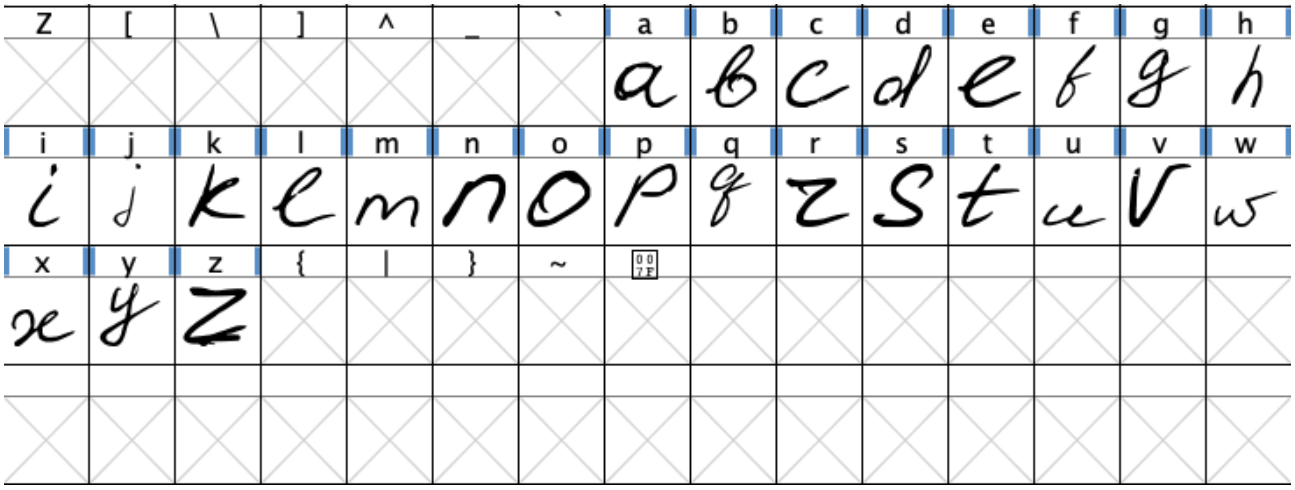


Рис. 3.10. Приклад відображення шрифта у візуальному редакторі

На даному скріншоті помітно, що символи все-рівно мають різні розміри та шрифт ще потребує доопрацювань. На рис. 3.11. зображено приклад тексту написаний результуючим шрифтом.

Рис. 3.11. Приклад тексту написаний створеним шрифтом

На рисунку видно, що всі літери було коректно розпізнано системою, та завантажено у шрифт. Помітно також, що розміри гліфів все ж не є вірними і загалом вигляд шрифту далекий від ідеального відтворення почерку, а отже алгоритм потребує додаткових конфігурацій.

Після певних ручних виправлень та маніпуляцій, шрифт можна привести до потрібного стану, який буде виглядати наступним чином (рис. 3.12.)

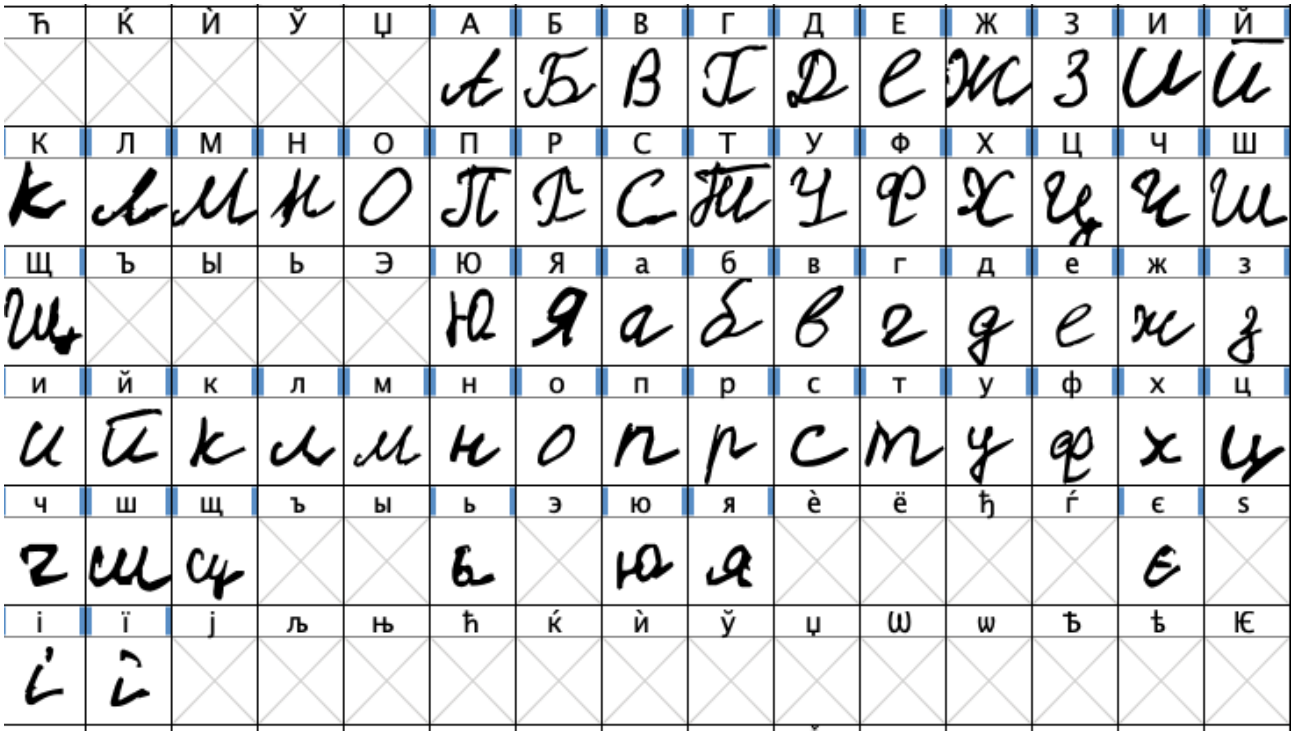


Рис. 3.12. Приклад шрифта після ручних форматувань

Даний шрифт містить лише символи кирилиці, отже і речення (рис. 3.13) написане українською

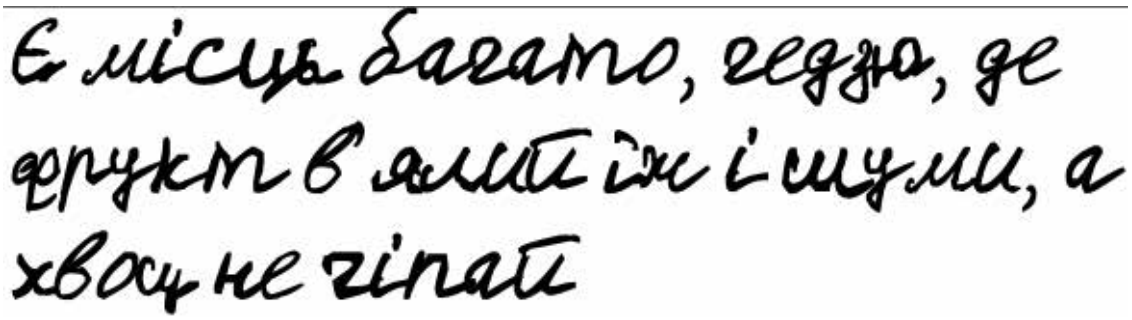


Рис. 3.13. Приклад тексту написаний шрифтом відредагованим вручну

Бачимо, що літери виглядають значно краще, хоча кернінг не завжди вдало комбінує літери між собою і вони інколи або напливають один на одне, або між ними залишається певна відстань.

3.4 Рекомендації щодо впровадження та експлуатації системи.

Для розуміння процесу впровадження необхідно ознайомитися з апаратною архітектурою додатку. Для цього зазвичай використовується діаграма розгортання. Deployment-діаграма, як один із типів UML-діаграм, моделює фізичне розташування компонентів програмної системи та їхні зв’язки з точки зору апаратного забезпечення і мережевих ресурсів. Її основна мета — показати, як компоненти програмного забезпечення розподілені між серверами, мережами та іншими пристроями. На рис. 3.14 наведено діаграму розгортання для даного проєкту.

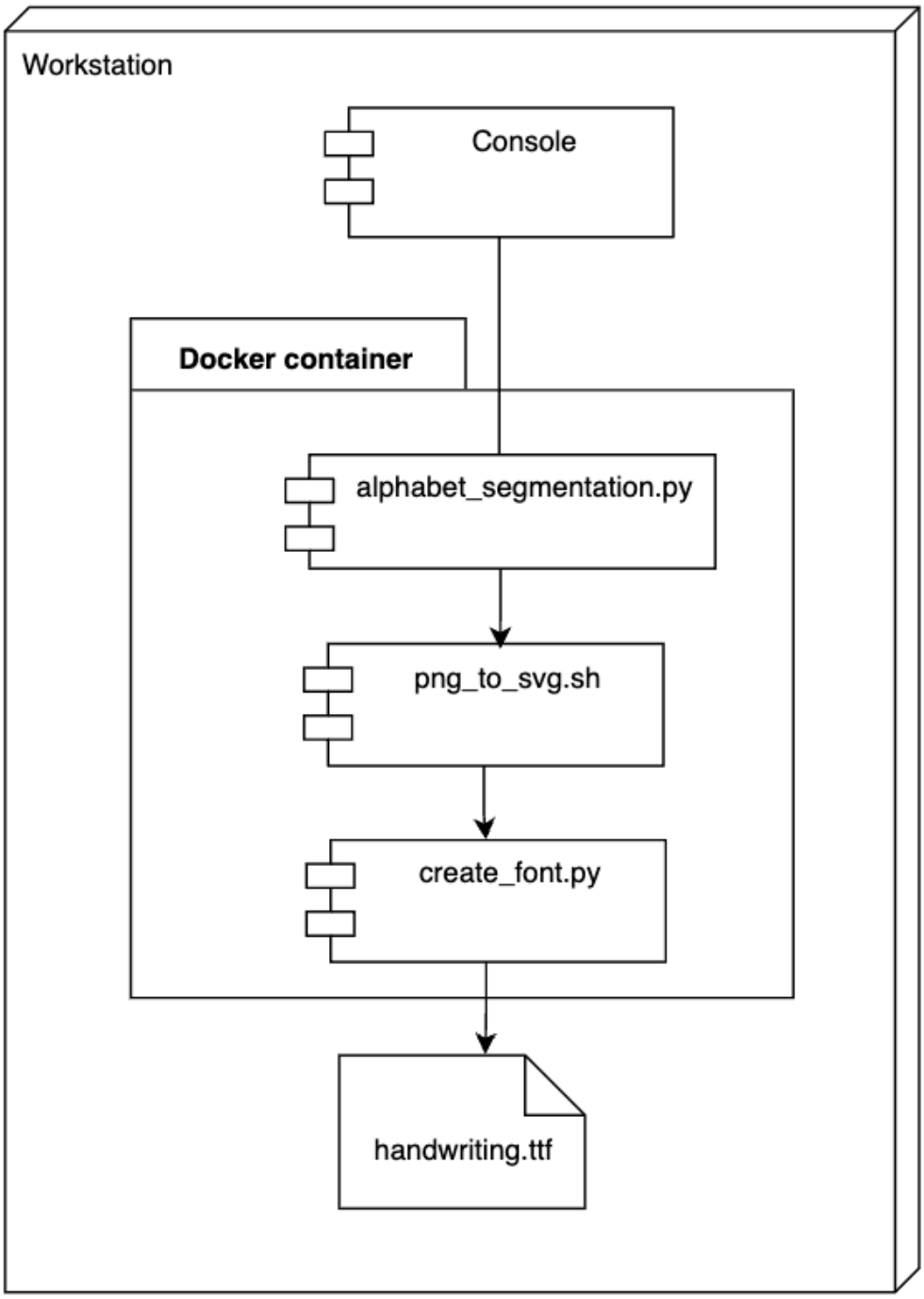


Рис. 3.14 Діаграма розгортання

Для полегшення розповсюдження даного додатку, було використано Докер, як середовище віртулізації та контейнеризації. Таким чином при завантаження проєкту із репозиторію, достатньо запустити одну докер команду для того, аби встановити всі необхідні бібліотеки та залежності.

3.5 Висновки до розділу 3

У третьому розділі було детально висвітлено процес дослідження та розробки програмного забезпечення, який охоплює всі ключові етапи створення системи. Розділ структуровано відповідно до логіки розробки, починаючи з алгоритмів обробки зображень і закінчуючи демонстрацією роботи програми та рекомендаціями щодо її впровадження. У розділі описано всі проблеми з якими довелось зіштовхнутись під час дослідження.

На початку розділу було описано розробку алгоритму вирізання символів, який включає кілька етапів. Спочатку реалізовано методи попередньої обробки зображень, зокрема згладжування, бінаризацію та видалення шумів. Далі описано алгоритми пошуку контурів, їх обробки, сортування та експорту у вигляді окремих зображень, що дозволяє якісно підготувати символи для подальшого використання.

У наступному підрозділі описано процес створення шрифту. Основну увагу приділено перетворенню растрових зображень символів у векторний формат, що є важливим етапом для забезпечення масштабованості та збереження деталей почерку. Також розглянуто формування файлу шрифта, включаючи генерацію гліфів, налаштування метрик і підготовку шрифтового файлу у стандартних форматах, придатних для використання.

У розділі також проведено демонстрацію роботи програми, яка показує функціональність системи на практиці. Цей підрозділ наочно ілюструє виконання завдань системою, підтверджуючи її працездатність і ефективність.

Завершується розділ рекомендаціями щодо впровадження та експлуатації розробленої системи.

У підсумку, третій розділ детально розкриває технічні аспекти реалізації програмного забезпечення, демонструє виконану роботу та надає практичні рекомендації для успішного впровадження системи в експлуатацію.

ВИСНОВКИ

У ході виконання роботи було розроблено програмне забезпечення, яке дозволяє автоматизувати процес створення шрифту на основі фотографій рукописного тексту. У дослідженні проаналізовано проблему, яку вирішує система, обґрунтовано актуальність її розробки, описано предметну область і проведено порівняння з існуючими програмами-аналогами. На основі цього аналізу було сформульовано функціональні та нефункціональні вимоги до системи, які стали основою для проєктування та реалізації програмного продукту.

У процесі роботи було спроєктовано інформаційну систему, використовуючи методи UML моделювання. Було створено діаграми, які відображають архітектуру системи з різних точок зору, що значно спростило розробку. Також проведено детальний опис вибору програмних технологій.

На етапі розробки програмного забезпечення було вирішено низку важливих технічних завдань, таких як обробка зображень, об’єднання та сортування контурів, перетворення растрових символів у векторний формат, та налаштування шрифта. Алгоритми, що були створені, забезпечують достатню точність обробки зображення і дозволяють генерувати шрифти у стандартному форматі (TTF).

Однак під час дослідження та реалізації проєкту довелося зіткнутися з численними технологічними перешкодами. Наприклад, труднощі виникали у забезпеченні точності розпізнавання контурів при низькій якості зображення, а також у подальшій векторизації складних символів. Деякі з цих проблем вдалося вирішити шляхом оптимізації алгоритмів та підбору налаштувань, але частина залишилася невирішеною через обмеженість ресурсів та часу. Хоча дослідження показало, що навіть вручну створені шрифти, які відтворюють почерк, майже неможливо зробити ідеальними, система має потенціал для розвитку та удосконалення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Академія мистецтв імені Павла Чубинського: Що може розповісти про людину його почерк [Електронний ресурс] – Режим доступу до ресурсу - https://koukim.com/ua/cikavo_znati.html
2. Опорний конспект з лекцій курсу Аналіз вимог до програмного забезпечення.
3. ДИПЛОМНА РОБОТА: Автоматизована система розрахунку вартості програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу - https://ela.kpi.ua/bitstream/123456789/29278/1/Kyseliov_bakalavr.pdf
4. Документація OpenCV [Електронний ресурс] - Режим доступу - https://docs.opencv.org/4.x/index.html
5. Python documentation [Електронний ресурс] – Режим доступу до ресурсу - https://docs.python.org/3/
6. Документація Potrace [Електронний ресурс] - Режим доступу до ресурсу - https://potrace.sourceforge.net/
7. Опис файлового формату SVG [Електронний ресурс] - Режим доступу до ресурсу - https://uk.wikipedia.org/wiki/Scalable_Vector_Graphics
8. Що таке Кернінг [Електронний ресурс] - Режим доступу до ресурсу - https://uk.wikipedia.org/wiki/Кернінг
9. Handwritten Character Recognition Models Based on Convolutional Neural Networks [Електронний ресурс] - Режим доступу до ресурсу - https://openarchive.nure.ua/server/api/core/bitstreams/4ea61272-3e48-4e64-a99b-d385ad5070e9/content
10. Convolutional neural network [Електронний ресурс] - Режим доступу до ресурсу - https://en.wikipedia.org/wiki/Convolutional_neural_network
11. IAM Handwriting Database [Електронний ресурс] - Режим доступу до ресурсу - https://fki.tic.heia-fr.ch/databases/iam-handwriting-database
12. Image Processing in Python [Електронний ресурс] - Режим доступу до ресурсу - https://www.geeksforgeeks.org/image-processing-in-python/

13. Комп'ютерний зір [Електронний ресурс] - Режим доступу до ресурсу - https://uk.wikipedia.org/wiki/Комп’ютерний зір
14. Handwriting words recognition with PyTorch [Електронний ресурс] - Режим доступу до ресурсу - https://pylessons.com/handwriting-recognition-pytorch
15. Finding extreme points in contours with OpenCV [Електронний ресурс] - Режим доступу до ресурсу - https://pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/
16. Опис будови svg файлів [Електронний ресурс] - Режим доступу до ресурсу - https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths
17. Про формат TTF [Електронний ресурс] - Режим доступу до ресурсу - https://docs.fileformat.com/uk/font/ttf/
18. Font Engine [Електронний ресурс] - Режим доступу до ресурсу - https://developer.apple.com/fonts/TrueType-Reference-Manual/RM02/Chap2.html#intro
19. Документація Numpy [Електронний ресурс] - Режим доступу до ресурсу - https://numpy.org/

```
import math
import subprocess
import cv2
from matplotlib import pyplot as plt
import numpy as np
import os

def sort_contours(contours_info, row_threshold=100):
    """
    Organizes contours into a matrix based on their positions.

    Parameters:
        contours_info (list): List of contour dictionaries.
        row_threshold (int): Vertical distance threshold to group contours into the same row.

    Returns:
        List[List[dict]]: Matrix of contours organized as rows and columns.
    """
    # Sort contours by their top coordinate (extTop[1])
    contours_info_sorted = sorted(contours_info, key=lambda c: c['extTop'][1])

    rows = []
    current_row = []
    last_top = None

    for info in contours_info_sorted:
        current_top = info['extTop'][1]
        if last_top is None:
```

```
            last_top = current_top
            current_row.append(info)
        elif abs(current_top - last_top) < row_threshold:
            current_row.append(info)
        else:
            # Sort current row left to right based on extLeft[0]
            current_row.sort(key=lambda c: c['extLeft'][0])
            rows.append(current_row)
            current_row = [info]
            last_top = current_top

    # Add the last row
    if current_row:
        current_row.sort(key=lambda c: c['extLeft'][0])
        rows.append(current_row)

    result = []
    for row in rows:
        result.extend(row)

    return result

def segment_characters(image_path, output_folder):
    # Ensure output directory exists
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Load the image
    image = cv2.imread(image_path)
```

```
if image is None:
    print("Error: Unable to load image.")
    return

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian Blur to reduce noise
blurred = cv2.GaussianBlur(gray, (5, 5), 0)



# Apply Otsu's Thresholding after Gaussian filtering
_, thresh = cv2.threshold(
    blurred, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_TRIANGLE
)
# cv2.imshow('Dilated Image', thresh)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# Find contours on the dilated image
contours, hierarchy = cv2.findContours(
    thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
)

# Calculate bounding boxes and extreme points
contours_info = []
for idx, cnt in enumerate(contours):
    area = cv2.contourArea(cnt)
    if area < 40:  # Adjust area threshold as needed
```

```
            continue
        info = {
            'index': idx,
            'contour': cnt,
            'area': area,
            'extLeft': tuple(cnt[cnt[:, :, 0].argmin()][0]),
            'extRight': tuple(cnt[cnt[:, :, 0].argmax()][0]),
            'extTop': tuple(cnt[cnt[:, :, 1].argmin()][0]),
            'extBot': tuple(cnt[cnt[:, :, 1].argmax()][0]),
        }
        contours_info.append(info)
        roi = thresh[
            info['extTop'][1]:info['extBot'][1],
            info['extLeft'][0]:info['extRight'][0]
        ]
        x = info['extLeft'][0]
        y = info['extLeft'][1]
        cv2.imwrite(os.path.join("mediate_res", f'letter_{x}_{y}_{area}.png'), roi)

dot_threshold = 500
dot_dist_threshold = 100

# Merge small contours (dots) with nearby larger contours (letters)
merged_contours = []
skip_indices = set()
for i, info in enumerate(contours_info):
    if i in skip_indices:
        continue
    cnt = info['contour']
```

```
 if info['area'] < dot_threshold:  # Adjust area threshold as needed
        # Possible dot, find nearby larger contour
        for j in range(len(contours_info)):
            if i == j:
                continue
            other = contours_info[j]
            if other['area'] > dot_threshold:
                # Calculate distance between dot bottom and letter top
                distance = math.dist(info['extBot'], other['extTop'])
                if distance < dot_dist_threshold:
                    # Merge contours
                    merged = np.concatenate((cnt, other['contour']))

                    # if the countour was already merged, we need to remove the previous
one
                    if j in skip_indices:
                        for cnt_i, cnt in enumerate(merged_contours):
                            if cnt['index'] == j:
                                merged_contours.pop(cnt_i)
                    info['index'] = j
                    info['contour'] = merged
                    info['area'] += other['area']
                    info['extBot'] = other['extBot']
                    info['extLeft'] = other['extLeft'] if other['extLeft'][0] < info['extLeft'][0]
else info['extLeft']
                    info['extRight'] = other['extRight'] if other['extRight'][0] >
info['extRight'][0] else info['extRight']
                    merged_contours.append(info)
                    skip_indices.add(j)
```

```
                skip_indices.add(i)
                break
        skip_indices.add(i)

    for i, info in enumerate(contours_info):
        if i not in skip_indices:
            merged_contours.append(info)

    # Sort contours from left to right and from top to bottom
    merged_contours = sort_contours(merged_contours)
    # Extract and save each letter
    for idx, info in enumerate(merged_contours):
        roi = thresh[
            info['extTop'][1]:info['extBot'][1],
            info['extLeft'][0]:info['extRight'][0]
        ]
        roi = cv2.bitwise_not(roi)
        if roi.shape[0] > 0 and roi.shape[1] > 0:
            cv2.imwrite(os.path.join(output_folder, f'letter_{idx}.png'), roi)

    print(f"Extracted {len(merged_contours)} letters.")

if __name__ == "__main__":
    image_path = "input_ukr_2.jpg"
    output_folder = "result"
    segment_characters(image_path, output_folder)
    subprocess.run(
        ["./png_to_svg.sh", "result"],
    )
```

```
#!/bin/bash

# Check if ImageMagick and Potrace are installed
if ! command -v magick &> /dev/null; then
  echo "ImageMagick (magick) could not be found. Please install it first."
  exit 1
fi

if ! command -v potrace &> /dev/null; then
  echo "Potrace could not be found. Please install it first."
  exit 1
fi

# Set the directory to work on
if [ -n "$1" ]; then
  WORK_DIR="$1"
else
  WORK_DIR="."
fi

# Check if the directory exists
if [ ! -d "$WORK_DIR" ]; then
  echo "The specified directory does not exist: $WORK_DIR"
  exit 1
fi

# Loop through all PNG files in the specified directory
for pngfile in "$WORK_DIR"/*.png; do
  if [ -e "$pngfile" ]; then
    FILE=`basename "$pngfile" .png`
    # echo "Converting $pngfile to $WORK_DIR/$FILE.svg"
    magick "$pngfile" "$WORK_DIR/$FILE.pbm"
    potrace -s -o "$WORK_DIR/$FILE.svg" "$WORK_DIR/$FILE.pbm" --tight
    rm "$WORK_DIR/$FILE.pbm"
  else
    echo "No PNG files found in the directory: $WORK_DIR"
    exit 1
  fi
done

echo "Conversion complete."
```

```
import fontforge
import os

# alphabet = "abcdefghijklmnopqrstuvwxyz0123456789"
alphabet = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯабвгґдеєжзиіїйклмнопрсту
фхцчшщьюя123456789"
bearing_table = {
    "Default": [60, 60],
    "A": [60, -50],
    "a": [30, 40],
    "B": [60, 0],
    "C": [60, -30],
    "c": [60, 40],
    "b": [60, 40],
    "D": [60, 10],
    "d": [30, -20],
    "e": [30, 40],
    "E": [70, 10],
    "F": [70, 0],
    "f": [0, -20],
    "G": [60, 30],
    "g": [20, 60],
    "h": [40, 40],
    "I": [80, 50],
    "i": [60, 60],
    "J": [40, 30],
    "j": [-70, 40],
    "k": [40, 20],
```

"K": [80, 0],
"H": [60, 10],
"L": [80, 10],
"l": [60, 0],
"M": [60, 30],
"m": [40, 60],
"N": [70, 10],
"n": [30, 40],
"O": [70, 10],
"o": [40, 40],
"P": [70, 0],
"p": [60, 40],
"Q": [70, 10],
"q": [20, 30],
"R": [70, -10],
"r": [60, 40],
"S": [60, 60],
"s": [20, 40],
"T": [60, -10],
"t": [-10, 20],
"U": [70, 20],
"u": [40, 40],
"V": [60, -10],
"v": [20, 20],
"W": [70, 20],
"w": [40, 40],
"X": [60, -10],
"x": [10, 20],
"y": [20, 30],

```
    "Y": [40, 0],
    "Z": [60, -10],
    "z": [10, 20],
    "1": [-10, 30],
    "2": [-10, 30],
    "3": [10, 40],
    "4": [30, 30],
    "5": [30, 40],
    "6": [20, 20],
    "7": [30, 20],
    "8": [30, 20],
    "9": [30, 30],
    "0": [50, 40],
}
descenders_shift = {
    "g": 500,
    "j": 600,
    "p": 500,
    "q": 600,
    "y": 500,
    "f": 400,
    "з": 300,
    "р": 200,
    "у": 300,
    "ф": 300,
    "д": 300,
    "щ": 100,
    "ц": 100,
}
```

```
def create_font_from_images(image_folder, output_font_path):
    # Create a new font
    font = fontforge.font()
    font.fontname = "handwriting"
    font.fullname = "Custom Handwriting Font"
    font.familyname = "CustomHandwriting"
    font.encoding = "UnicodeFull"


    # Loop over image files in the image folder
    for image_file in os.listdir(image_folder):
        if image_file.endswith(".svg"):
            file_name = os.path.splitext(image_file)[0]
            letter_index = int(file_name.split("_")[-1])
            if letter_index >= len(alphabet):
                continue

            char_name = alphabet[letter_index]

            # Skip files not matching a valid character
            if len(char_name) != 1:
                continue

            # Create a new glyph for the character
            glyph = font.createChar(ord(char_name), char_name)

            # Import the PNG image as a glyph outline
            glyph.importOutlines(os.path.join(image_folder, image_file), scale=False)
```

```
            glyph.left_side_bearing = bearing_table.get(char_name, (2, 2))[0]
            glyph.right_side_bearing = bearing_table.get(char_name, (2, 2))[1]

            glyph.transform([6, 0, 0, 6, 0, 0])

            # Move glyph to align with the baseline
            x_min, y_min, x_max, y_max = glyph.boundingBox()
            y_min = y_min + descenders_shift.get(char_name, 0)
            glyph.transform([1, 0, 0, 1, 0, -y_min])



    # Generate the font file (TTF)
    font.generate(output_font_path)
    print(f"Font created successfully and saved to {output_font_path}")

if __name__ == "__main__":
    create_font_from_images("result", "custom_handwriting.ttf")
```