**­<PROJECT 중간보고서>**

**Pitalium Project**

**With NS Solutions**

**1조**

**2009-13329 오성민**

**2010-13397 오영진**

**2011-13364 전민영**

Table of Contents

1. Abstract
2. Introduction
3. Background Study
4. Goal & Requirements
5. Approach
6. Project Architecture
7. Implementation Spec

7.1 Different Area Recognition

**7.2 Shift Check**

**7.3 Similarity calculation**

**7.4 Categorization**

1. Current Status
2. Future Work
3. Division & Assignment of Work
4. Schedule
5. Demo plan
6. **Abstract**

Chrome, Safari, Firefox 등 새로운 웹 브라우저가 생겨나고, Android와 같은 모바일 브라우저까지 등장하면서 인터넷을 사용하는 환경이 다양해지고 있다. 다양한 디바이스에서 인터넷을 사용할 수 있게 되면서 웹 어플리케이션은 수 많은 브라우저를 통해 보여지게 되는데, 브라우저 마다 상이한 rendering 방식을 사용하기 때문에 웹 페이지가 예상치 못한 레이아웃으로 보여지는 경우가 자주 나타난다. 이 때문에 웹 개발자는 여러 브라우저들에서 결과가 의도한 대로 잘 나오는지 확인해야 하는 불편함을 겪는다. 이러한 불편함을 해소하기 위해 테스트 자동화 툴이 필요하다.

1. **Introduction**

Pitalium은 다양한 플랫폼 위의 서로 다른 웹 브라우저에 의해서 보여지는 웹 페이지의 스크린샷을 비교하여 차이점을 보여줌으로써, 개발자가 원하는 대로 웹 페이지가 rendering되었는지 확인하는 것을 도와준다. 기존의 Pitalium은 rendering의 사소한 차이까지 모두 보여주기 때문에 user가 필요로 하는 정보를 실용적으로 보여주지 못하고 있다. (Figure 1.) 본 프로젝트는 이러한 Pitalium을 user-friendly 하고 실용적으로 개선시키는 것을 목표로 한다.



Figure 1. Result of previous pitalium

* 1. **Image comparison improvement**

기존의 Pitalium의 근본적인 문제는 단순한 Image comparison method에 기인한다. Pixel by pixel 비교를 통해 모든 ‘Difference’를 잡아내기 때문에, user가 정말로 필요로 하는 ‘Identifiable Difference’를 구분하기가 어렵다. 본 프로젝트에서는 이를 해결하기 위하여 user의 입장에서의 ‘Difference’를 Categorization하고, 각각의 category를 구분하기 위해 Image comparison method를 개선시킨다.

* 1. **User Interface improvement**

UI개선은 세 단계로 이루어진다. 먼저 Image comparison을 할 때, 한번에 1\*n pair로 돌려볼 수 있게 하고, 한 브라우저를 기준(이하 Expected Image)으로 다른 브라우저(이하 Actual Image)들과의 유사도를 한눈에 볼 수 있도록 만들고, 다른 부분의 구체적인 정보를 interactive하게 화면에 띄워주는 것이다. 이는 항목별로 자세히 서술하겠다.

기존의 시스템에서는 두 스크린샷을 보는 페이지로 들어가야 Image comparison 알고리즘을 돌리는데, 알고리즘 자체는 오래 걸리지 않지만 File System Writing과 프론트에 띄우는 일에 시간이 더 소요되기 때문에 확인할 때마다 알고리즘을 돌리는 것이 오히려 비효율적이다. 따라서 새로운 시스템에서는 사용자가 비교하고 싶은 스크린샷들을 선택하고 그것들 전체에 대한 비교 알고리즘 돌려서 목록은 프론트로 가져오고 분석 데이터는 json파일로 저장해 놓는다.

기존의 시스템에서는 유사도(이하 Similarity)가 수치로 나오지 않았기 때문에 두 스크린샷의 비교 결과를 확인하기 위해서, 결과 이미지를 user가 일일이 비교하여야 하고 따라서 시간과 노력이 많이 소요된다. 하지만 새로운 시스템에서는 한번에 분석하여 결과를 Similarity로 보여주기 때문에 다른 브라우저들과의 유사도를 한눈에 확인하는 표&그래프를 만들 수 있고, 사용자는 많은 정보를 종합적으로 빠르게 인지할 수 있다.

마지막으로 Image Comparison Algorithm의 개선으로 인해 그 결과 수치들을 프론트에서도 받아 볼 수 있게 되었고, 이를 보여주는 방식 또한 HTML5의 기능을 활용하여 유저에게 interactive하게 전달할 수 있다. 비교 결과 다른 부분을 화면에 표시하는 것을 기존에는 이미지 덧씌우기로 처리하였는데, 새로운 시스템에서는 이를 html의 div요소로 만들어 화면에 띄우고, 각각의 div에 마우스를 올렸을 때 동적으로 반응하여 그 부분에 대한 분석 결과물을 보여준다. (Figure 2.)

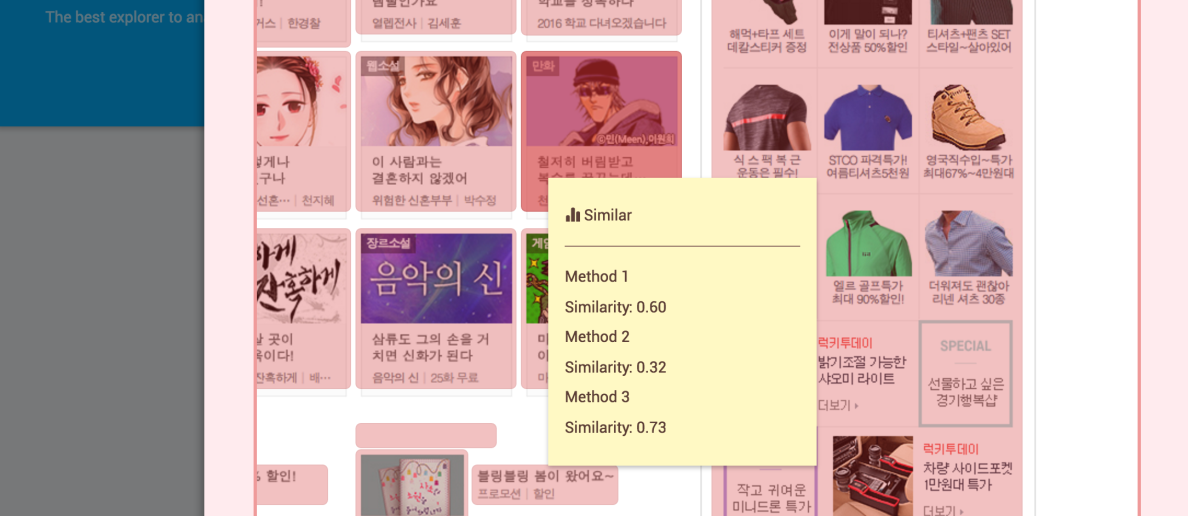


Figure 2. Interactive result

1. **Background study**
   1. **Similarity calculation method**

Similarity를 계산하기 위하여 다음과 같은 방법들을 사용하였다.

* + 1. Correlation match

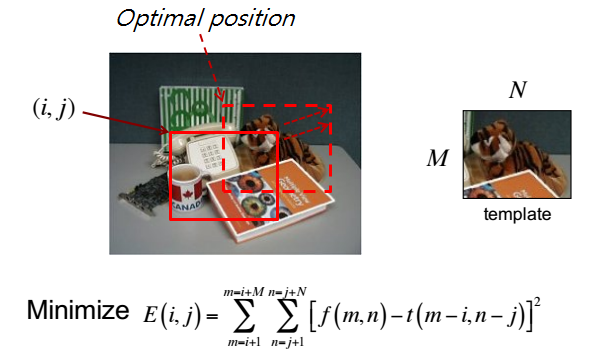


Figure 3. Cross-correlation method find minimum error.

주어진 template 이미지가 전체 이미지에서 어디에 위치해있는지 찾기 위하여, 오차를 계산해 나가는 방법이다. (Figure 3.) 비교 영역을 움직여가며 오차를 최소화하는 위치를 찾는다. 이 오차 값을 normalize하여 Similarity의 metric으로 사용할 수 있다. Similarity를 계산하는 방법은 다음과 같다.

DRW000013cc6f1d

* + 1. Feature matrix

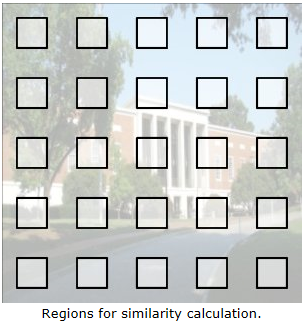
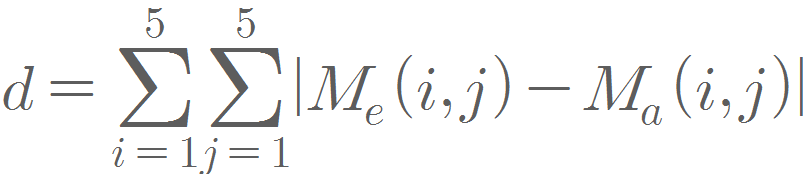
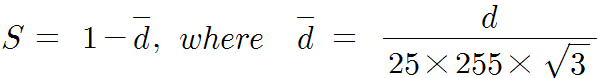


Figure 4. How to extract feature matrix from image

각각의 이미지로부터 Feature matrix를 구하여 이를 비교하는 방법이다. 이미지를 5x5 gird로 분할하고, 각 gird로부터 feature를 뽑아내어 feature value를 원소로 갖는 5x5 matrix를 만든다. 본 프로젝트에서는 average color를 feature로 이용하였다. 각각의 Feature matrix *,* 로부터 Similarity를 구하는 방법은 다음과 같다.





1. **Goal & Requirements & Specification**

본 프로젝트의 목표와 요구사항은 다음과 같다.

* 1. **Goal & Requirement**

4.1.1 Image comparison

- Image comparison에서 사소한 차이를 무시하고 ‘Identifiable Difference’를 구별해 낸다.

- 이미지 사이의 Similarity를 계산하여 user에게 유용한 정보를 제공한다.

- ‘Differences’를 Categorization하여 user가 필요한 정보를 선택적으로 볼 수 있도록 한다.

- 모든 브라우저를 비교하는 total running time을 최적화한다.

* + 1. User Interface

- 전체 테스트 결과를 한 눈에 알아볼 수 있도록 한다.

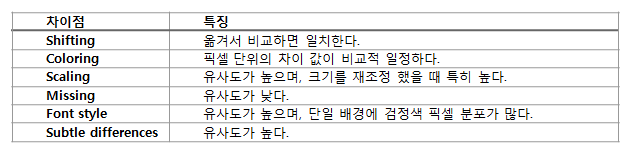
- 각 이미지 쌍에 대한 결과를 similarity를 통해 요약해 볼 수 있도록 한다.

- 각 Rectangle area에 대한 정보를 interactive하게 보여준다.

* 1. **Specification**
* 두 이미지 간의 Similarity를 계산할 수 있어야 한다.
* 다양한 웹 브라우저들 사이에서의 비교를 효율적으로 하여야 한다. (모든 Image comparison이 10초 이내에 이루어 져야 한다.)
* 데이터가 누적되더라도 시스템이 느려지지 않고 잘 작동하여야 한다.
* 테스트 결과를 한 눈에 확인할 수 있어야 한다.
* Java-native한 library들로만 구성하여야 한다.

1. **Approach**

우선 브라우저 간 rendering 방식의 차이로 나타날 수 있는 결과를 분류하고, 각 차이점들의 특징을 잡아낸다. 그리고 그 특징을 걸러낼 수 있는 비교 알고리즘을 고안한다. 프로젝트를 진행하기에 앞서 고려한 예상되는 브라우저 간 차이의 종류는 다음과 같다.



상기 차이점들 중, 다른 shifting에 대하여 다른 차이점들과 구분 지을 수 있는 특징이 분명한 것으로 보고 우선 shifiting을 구별해 내고자 한다. 다른 차이점들에 대하여는 서로 겹치는 특징이 많아 이산적인 분류가 어려울 것으로 보이며, 다양한 유사도 측정 알고리즘을 통해 각 원인으로 추정되는 확률을 제공하기로 한다. 본 프로젝트에서는 일차적으로 다음과 같은 네 가지로 분류하고자 한다.

*Shifting, Scaling, Missing, Similar*

**5.1 Comparison approach**

기준이 되는 Expected Image와 비교 대상인 Actual Image들을 유저로부터 입력 받아, 각 Image pair(Figure 5.(a))에 대해 Comparison을 수행한다.

Image comparison은 크게 4단계(Figure 5.(b))로 나눌 수 있다.

Figure 5. Image comparison process

먼저, Different Area Recognition step에서 주어진 Image pair를 비교하고, 비교 영역을 세분화한다. 웹 페이지 상에서 오브젝트들은 일반적으로 사각형단위로 영역을 할당받으므로, 세분화된 비교 영역을 Rectangle area로 할당하여 영역 내의 dominant object에 대한 비교를 효과적으로 할 수 있도록 하였다. 세분화된 비교 영역은 이러한 Rectangle area의 list로 반환된다.

각각의 Rectangle area에 대해서 Shift Check – Similarity Calculation – Categorization의 순으로 Image comparison을 진행한다. Shift Check는 rectangle area의 shift 여부를 검사하며 Similarity calculation은 다양한 method들을 이용하여 Similarity 값들을 구한다. 각각 다른 method로부터 구한 Similarity들의 조합을 이용하면, 주어진 rectangle area에서의 차이점의 특징을 알 수 있고, 이를 이용하여 Categorization을 할 수 있다.

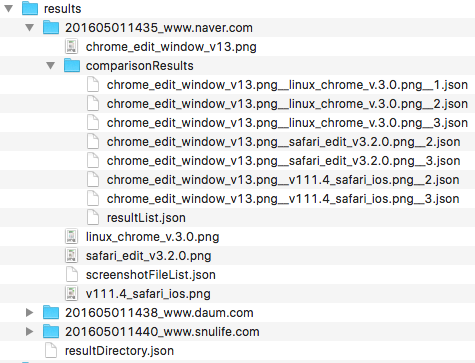
Image comparison이 끝난 rectangle은 결과 정보를 가진 Compared rectangle로 반환된다. 각 Image pair는 comparison의 결과로 이러한 Compared rectangle의 list를 갖는다. 그리고 이를 이용하여 Pitalium을 이용하는 user가 필요로 하는 정보를 제공할 수 있다.

1. **Project architecture**

**6.1 Database**

본 프로젝트는 Git을 통해 재배포 가능한 testing tool을 목표로 하기 때문에 별도로 설치해야 하는 외부 DBMS를 사용하지 않는다. 이미지들은 프로젝트 폴더 내의 별도의 폴더에 존재하며, 검사 결과 정보는 json format을 따른 파일로 저장한다. 검사에 대한 API요청은 필요에 따라 json file을 읽거나 쓰게 되며, file system을 database 처럼 사용한다. Pitalium은 각 개발자가 local machine에서 사용하는 것을 염두에 두기 때문에, 이와 같은 구조를 채택하였다. Local file system을 사용할 경우, 다른 사용자와 검사 기록을 공유하기 위해 생성된 json 파일만 주고 받으면 되기 때문이다.

검사 대상인 이미지들과 검사 결과는 아래와 같은 구조로 저장된다.



* <resultDirectory.json>은 <results> 폴더 내의 subdirectory들의 목록 정보를 하위 이미지 개수, 검사 결과 수, 브라우저 수와 함께 담고 있다. 각 subdirectory는 검사를 수행할 이미지 군을 의미한다. 검사는 같은 폴더 내의 이미지들과 이루어진다.
* <screenshotFileList.json> 은 각 이미지들의 파일명과, 플랫폼, 브라우저, 브라우저 버전, 파일 크기, 이미지 크기정보를 담고 있다.
* <resultList.json>은 기 수행된 검사 수행 정보를 담고 있다. Expected로 설정된 파일 이름과, 이와 함께 비교된 target image들의 리스트, 그리고 각 image pair의 검사결과 요약 정보가 있다. 요약정보로는 entire similarity와 총 different rectangle의 수가 제공된다. 유저는 이 json파일에 담긴 리스틀통해 과거 검사기록을 관리하고, 조회할 수 있다.
* <[expected\_file\_name]\_\_[target\_file\_name]\_\_index.json>는 각 이미지 쌍의 검사 상세 정보를 담고 있다. 이미지 간에 차이가 있다고 판별 된 different rectangle들의 list가 저장되는데, 각 rectangle들은 difference type과 shift된 정도, 세가지 유사도 측정 알고리즘에 유사도 정보를 함께 가지고 있다. 이 정보를 토대로 front-end에서는 interactive pin을 구현한다.

필요에 따라 database를 사용할 수 있는데, 이때는 java native DBMS 이며, Spring frame work자체에서 사용할 수 있는 HyperSQL을 사용한다. 이 경우의 database schema는 다음과 같다.

pitalium-explorer.png

**6.2 Server side**

Pitalium explorer의 서버로는 Spring framework을 사용한다. 본 프로젝트는 기존의 Pitalium explorer를 개선하는 것이기 때문에, 서버 framework를 변경하지 않는다.

서버에서 제공되는 API명세는 다음과 같다

|  |  |  |
| --- | --- | --- |
| Method/URL | Parameter | Description |
| GET  \_results/list | Int page  Int pageSize  Boolean refresh | 초기 선택 페이지를 위해 results폴더 내의 subdirectory list를 반환한다.  - return: Page<ResultDirectory> |
| GET  \_screenshots/list | String name  Boolean refresh | Subdirectory를 선택했을 때, screenshot들의 리스트와 과거 검사 기록을 얻을 수 있다.  - return: Map<String, List> |
| POST  \_screenshots/compare | String directoryName  String expectedFilename  String[] targetFilenames | 기준 이미지와, 비교 이미지들을 선택했을 때, 1:N 비교를 수행하고 상세 내역을 JSON format으로 저장하며, 검사 결과의 요약을 반환한다.  - return: ResultListOfExpected |
| GET  \_screenshots/images | String directoryName  String expectedFilename  String targetFilename | 검사기록을 상세하게 조회하고자 할 때, 검사 수행된 expected 이미지와 target 이미지를 받아올 수 있다  - return: Map<String, byte[]> |
| GET  \_screenshots/result | String directoryName  Int resultListId  Int targetResultId | 검사기록을 상세하게 조회하고자 할 때, JSON format으로 기록된 상세 내역을 읽어온다.  - return: List<ComparedRectangle> |
| DELETE  \_screenshots/result | String directoryName  Int resultListId  Int[] targetResultIds | 검사기록을 상세를 삭제한다. |

Spring framework의 servelet context에서 front-end를 위한 static file들의 위치를 mvc resource로 설정한다. 모든 html파일은 API를 통하지 않고 접근 가능하다. 따라서 controller는 static html문서 파일에 data를 바인딩 할 필요 없이, JSON 형식의 데이터만을 전송한다. Client side에서는 html문서를 받자마자 hifive framework를 통해 ajax 요청을 하게 되고, ajax를 통해 호출된 API는 html문서에 필요한 작업을 수행한 뒤 결과를 반환한다.

6.2.1 Use case & API request scenario

1. Request: User는 초기화면에 접속하면서 html 문서를 얻고 ajax로 \_result/list에 GET방식의 http request를 보낸다.
2. Response: 서버는 results 폴더 하위의 subdirectory들의 목록을 응답한다. 응답 데이터는 ResultDirectory 오브젝트의 list이며, ResultDirectory 오브젝트는 폴더의 기본정보와 내부에 담고 있는 이미지들의 수, 검사결과의 수 정보 등을 담고 있다..
3. Request: User는 검사를 수행하거나, 과거 검사 기록을 보기 위해subdirectory 중 하나를 선택한다.
4. Response: 서버는 선택한 subdirectory에 있는 screenshot들의 목록과, 기 수행된 결과 내역을 보낸다. 응답 데이터는 List<ScreenshotFile>과 List<ResultListOfExpected>를 값으로 갖는 Map이다.
5. Request: User는 expected image와 target image를 선택해 새로운 검사를 수행하거나, 기존 수행된 결과를 볼 수 있다.
6. Response: 만약 새로운 검사가 수행되었다면 서버는 비교 연산을 수행하고 결과 목록의 갱신을 위해 검사 결과의 요약을 전송한다. 이때 전송되는 데이터는 ResultOfExpected가 된다. 만약 기존 검사 결과에 대한 상세 정보를 요청 받았다면 서버는 상세 요청 받은 검사 기록에 대한 expected image와 target image 그리고 결과 내역을 보낸다. 이때 결과 내역은 List<ComparedRectangle>이며, ComparedRectangle은 x, y, width, height값과 shift 여부와 similarity 정보를 담고 있다.

Server-side에서 사용하는 object들의 상세와 관계는 다음 UML과 같다.

pitalium-explorer-filestructure.png

pitalium-explorer-filestructure%20(1).png

1. **Implementation Spec**

**7.1 Different Area Recognition**

a. DiffPoints compare (BufferedImage image1, Rectangle imageArea1, BufferedImage image2, Rectangle imageArea2, CompareOption[] options)

주어진 영역 내에서 image1과 image2의 Color 값을 픽셀마다 비교한다. 동일한 Color 값을 갖지 않는 pixel을 different point로 간주하고 different point의 list인 DiffPoints를 반환한다.

b. List<Rectangle> convertDiffPointsToAreas (DiffPoints diffPoints)

주어진 Different Point들을 Union하여 Different Area를 구한다. GROUP\_DISTANCE의 default 값을 10px로 설정하였고, 이 거리 내에 있는 점들을 하나의 group으로 union하였다. (Figure 6.)

Figure 6. How to construct Different Areas using Different Points.

**7.2 Shift Check**

주어진 Rectangle Area에서 actual sub image가 shift되었는지 검사한다. Actual sub image를 template으로 두고, expected Image와 Correlation match를 시행한다. 주어진 Rectangle Area에서 모든 pixel의 value가 같으면 shift된 것으로 간주한다.

Figure 7. How to calculate IntegralImage array.

Time Complexity를 줄이기 위하여 integralImage array를 이용하여 Dynamic Programming을 적용하였다. integralImage array는 왼쪽 귀퉁이로부터 누적된 Color value의 합을 저장한다. (Figure 7.) 주어진 Rectangle Area의 integral value(Sum of color values)는 integralImage array의 네 귀퉁이 값을 이용하여 계산한다. (Figure 8.)

Figure 8. How to calculate the sum of the color values in given area.

먼저 주어진 영역의 integral value를 먼저 비교한다. 이 값이 같으면 Normal Image array의 주어진 영역에서의 모든 점들이 template image에서 해당하는 점과 같은 값을 갖는지 확인한다.

- 모든 pixel이 같은 Color value를 가지는 위치를 찾으면, Shift position 정보를 가지는 Compared Rectangle을 생성하고 Categorization step으로 넘어간다.

- 주어진 Rectangle area가 Shift되지 않았다면, Similarity calculation step으로 넘어가 Similarity를 계산한다.

**7.3 Similarity calculation**

Figure 9. SimilarityUtils class

Similarity를 계산하기 위해서 3가지 metric을 사용하였다. 각 pixel의 값을 이용하는 ‘distance norm’과 ‘the number of different pixels ‘을 구하기 위하여 *calcSimilarityPixelByPixel* method를 구현하였으며 area 전체의 feature를 이용하는 ‘Feature distance’를 구하기 위하여 *calcSimilarityByFeatureMatrix* method를 구현하였다. (Figure 9.)

* + 1. Distance Norm

주어진 area에서 Expected Image와 Actual Image 각각의 같은 위치에 해당하는 점들을 Pixel by pixel 비교한다. 먼저, 각각의 대응되는 점들의 normalized RGB-distance를 구한다. (Figure 10.) 주어진 area에서 모든 점들의 distance value를 normalize한 Distance Norm을 구하고 이를 이용하여 Similarity를 계산한다. (Figure 11.)

Figure 10. How to calculate distance value using corresponding points.

Figure 11. How to calculate Similarity using Distance Norms.

* + 1. The number of different pixels

전체 점들 중에서 ‘다른’ 점들의 비율을 이용하여 Similarity를 계산한다. 여기서 Difference는 다음과 같이 정의 된다. 주어진 점에서의 distance value *d*가 user가 설정하는 Threshold값 보다 크면 (Figure 12.), 다시 말해 일정 이상 Color value가 차이 나면 ‘다른’ 점으로 간주한다.

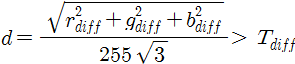


Figure 12. Condition to consider point different

* + 1. Feature distance

Expected Image와 Actual Image의 주어진 area를 5 by 5 grid로 나누어서 각 grid의 average color value를 feature로 하는 5 by 5 Feature matrix를 만든다. 이 Feature matrix간의 distance를 계산하여 이를 Similarity의 metric으로 사용한다. (Figure 13.)

Figure 13. How to calculate Feature distance.

**7.4 Categorization**

앞서 계산한 shift, similarity 정보를 이용하여 각 Rectangle area의 Category를 다음과 같이 나눌 수 있다. (Figure 14.)

Figure 14. Work flow for Categorization

* + 1. Shift

주어진 area의 image가 shift 된 경우를 의미하며 Shift position 정보를 user에게 제공한다. Shift position은 Actual image를 어느 정도 옮기면 Expected Image와 정확히 일치하는지를 의미한다.

* + 1. Scaling

주어진 area에서 Expected Image에 있는 object가 scaling(확대 혹은 축소)되어 Actual Image에 Rendering되었음을 의미한다. 즉, 두 image에서 object의 size가 다른 경우에 해당한다.

* + 1. Missing

주어진 area에서 Expected Image에 있는 object가 Actual Image에 나타나지 않는 경우를 의미한다.

* + 1. Similar

위의 세 경우에 해당하는 않는 경우 Similarity 정보만을 제공한다.

1. **Current status**

현재 Image Comparison에서 Categorization step전까지 프로토타입으로 구현하였다. Shift의 경우 Shift position을, 그렇지 않은 경우 다양한metric을 이용하여 계산한 Similarity들을 모두 보여주도록 하였다. (Figure 15.)

(

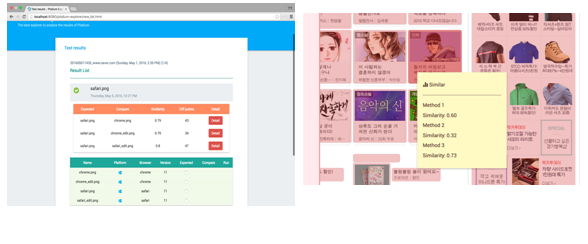


Figure 15. Current status

1. **Future work**

앞으로 구현해야 할 사항은 다음과 같다.

* Image의 특정한 상황을 잘 대표하는 새로운 metric을 이용한 Similarity calculation을 추가한다. 관점에 따라 다양한 유사도 measure를 적용할 수 있으므로, 필요에 따라 다양한 방법을 추가한다. 각 픽셀 color값의 차이를 Manhattan distance, L2 norm, Gaussian Distance 등 어떤 방법으로 계산하는지에 따라 유사도 값은 달라질 수 있다. 구분하고자 하는 차이의 종류에 따라 각 metric에 따른 값이 다를 것이며, 이를 통해 차이점들을 구분 할 수 있을 것으로 기대된다.
* 앞서 말했듯이 다양한 방법을 이용한 Similarity 정보들을 이용해 Scaling, Missing, Similar로의 Categorization을 수행할 수 있다. 이를테면 coloring의 경우에는 total sum of different point를 이용한 유사도는 낮을지 몰라도, norm을 이용한 유사도는 높을 것이다.
* Time complexity를 줄이기 위해 Image comparison 알고리즘을 최적화 할 수 있는 여지가 남아있다. 또는 더 나은 UX를 위하여, 검사 수행 결과를 lazy loading할 수도 있다.
* 차이가 발생한 DOM을 잡아내는 방법을 개선할 수 있다. 현재는 값이 다른 pixel point들을 aggregate하기 위해서 Rectangle들을 이용하고 있지만, k-nn과 같은 clustering 기법을 통한다면 different object를 좀 더 정확하게 잡아낼 수 있을 것으로 예상된다. 다만 time complexity를 충분히 고려해야 할 것이다.

1. **Division & Assignment of Work**

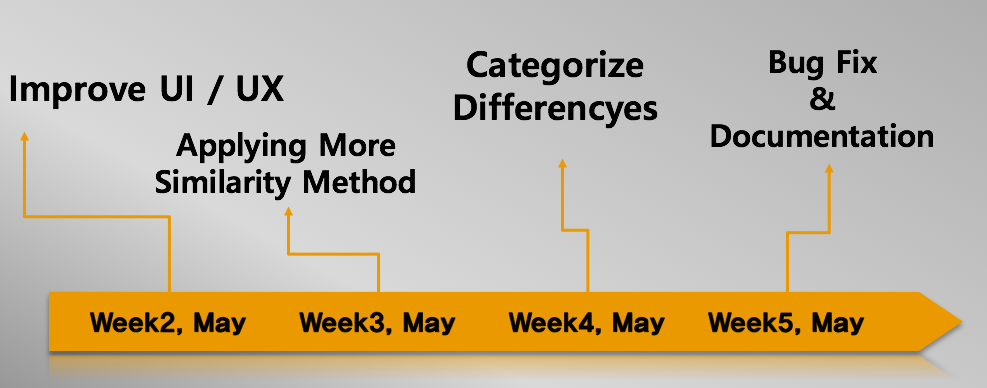
Comparison Logics: 오영진

Back-end (database layer & server-side): 오성민

Front-end (UI & interactivity): 전민영

1. **Schedule**

앞으로의 계획은 다음과 같다.



1. **Demo plan**

**12.1 Environment**

* 서버에 Tomcat 7을 설치한 후 Spring을 구동시켜 테스트 환경을 세팅한다. 이로써 인터넷에 연결되어 있으면 어디서든 테스트가 가능하게 된다.
* Scaling, Missing, Similar, Shift 등 Category 별로 총 4세트, 1세트에 5개의 스크린샷 파일을 포함하고 있다.
* 4세트를 순차적으로 Run 시키고 세부 결과 확인을 위해 스크린샷 중에 similarity가 가장 낮은 결과물과 가장 높은 결과물의 세부 정보를 확인한다.
* UI/UX의 개선된 부분을 확인하기 위해 View 페이지의 항목들을 조정하면서 확인한다. 세부 항목은 다음과 같다. Dual view, Swipe, Edge detection, Category 별 on/off, Similarity구간 별 on/off, Similarity에 따른 Opacity 조정.

**12.2 Test flow**

1. Pitalium explorer 사이트에 접속한다.
2. Entry를 선택해, image들과 기 수행된 result들의 목록을 불러온다.
3. 기 수행된 result를 클릭하여, 각 이미지 pair 별 상세 수행 내역을 확인한다.
4. 한 Image를 선택하여 1:N 비교를 수행한다.
5. 새롭게 수행된 결과가 result list에 표시되는지 확인한다.
6. 새롭게 수행된 결과의 image pair별 상세 수행 내역을 확인한다.
7. 유사도가 가장 낮다고 계산된 검사 결과를 확인한다.
8. Different box에 유사도 또는 차이점 별 분류 결과 내역을 확인한다.
9. 검사결과 확인 시, Dual view, Swipe, Edge detection, Single등의 View의 작동을 확인한다.
10. Shifting, Missing, Sizing 등의 원인 별 on/off switch를 동작한다.
11. Similarity 수치에 따른 threshold나 opacity를 조절해본다.