# HASIL DAN ANALISIS

Bab ini membahas mengenai hasil dari pengenalan citra digital menggunakan segmentasi *felzenzswalb*, ekstraksi *region adjacency graph*, dan *graph matching*. Pada bab ini akan dikaji mengenai penentuan parameter segmentasi dan tingkat akurasi dari metode pencocokan yang digunakan untuk pengambilan citra dalam penelitian ini.

## Uji Parameter dan Hasil Segmentasi Citra

Pada segmentasi *felzenzswalb* dari *scikit-*image terdapat 3 buah parameter bebas utama yaitu *scale*, *sigma*, dan *min\_size*. *Scale* merupakan parameter bebas yang dapat diatur untuk mengubah besaran setiap *cluster* atau *region*, semakin besar nilai *scale* semakin besar ukuran setiap *cluster*-nya. *Sigma* merupakan parameter yang berfungsi untuk mengatur ukuran *kernel* pada *gaussian smoothing* dalam segmentasi ini. Dan *min\_size* merupakan nilai ukuran minimum komponen untuk setiap *cluster*.

Berdasar hasil uji pada referensi pada citra 320 x 240 dengan nilai *scale* =1, , dan *min\_size* = 300 (Felzenszwalb, 2004), nilai terbaik untuk parameter *scale* dan *sigma* telah ditetapkan sebagai nilai *default* pada modul di *scikit-image* (*scale* =1, ). Maka dari itu, dalam penelitian ini parameter yang disetel hanyalah *min\_size* yang nilainya tentu akan berbeda karena bergantung pada ukuran piksel setiap citra. Gambar 4.1 menunjukkan kode dalam bahasa pemrograman python untuk operasi penyetelan parameter segmentasi.

|  |
| --- |
| # pengulangan uji parameter min\_size  **for** k **in** range**(**200**,**400**,**10**):**  count **=** np**.**array**([**0**,** 0**,** 0**,** 0**])**  **print** **(**'min\_size ->> {}'**.**format**(**k**))**  # uji segmentasi untuk 200 data artificial  **for** i **in** range**(**0**,**200**):**  **if(**i **%** 50 **==** 0**):**  **print(**count**)**  **print(**'----------------------------------------------'**)**  # segmentasi citra  img **=** io**.**imread**(**aFile**[**i**])**  labels **=** segmentation**.**felzenszwalb**(**img**,** min\_size**=**k **)**  labels **=** labels**.**astype**(**int**)**  # cek keunikan label untuk 4 kelas citra  u **=** len**(**np**.**unique**(**labels**))**  **if(**i**<**50 **and** u **!=** 2**):**  count**[**0**]** **=** count**[**0**]** **+** 1  **elif(**i**>=**50 **and** i**<**100 **and** u **!=** 3**):**  count**[**1**]** **=** count**[**1**]** **+** 1  **elif(**i **>=** 100 **and** i**<**150 **and** u **!=** 4**):**  count**[**2**]** **=** count**[**2**]** **+** 1  **elif(**i **>=** 150 **and** i**<**200 **and** u **!=**3**):**  count**[**3**]** **=** count**[**3**]** **+** 1  **else:**  **pass**  # print hasil uji untuk setiap parameter  **print(**count**)**  **print(**'jumlah segmen unik : {}'**.**format**(**np**.**sum**(**count**)))** |

Gambar 4.1 Operasi Penyetelan Parameter Segmentasi

Penyetelan parameter segmentasi dilakukan terhadap citra *artificial* sebanyak 200 buah yang telah disiapkan. Pengujian ini tidak menggunakan *ground truth* sebagai pembandingnya. Karena citra *artificial* yang digunakan sudah sederhana, uji parameter akan dilakukan untuk mencari keunikan jumlah setiap kelompok citra dengan melakukan *tracing* di sekitar nilai terbaik pada referensi (*min\_size =* 300). Semakin kecil nilai keunikan *cluster* pada setiap kelas citra, maka semakin baik segmentasi yang dilakukan. Berikut merupakan hasil observasi penentuan parameter untuk tahap segmentasi ditunjukkan oleh Tabel 4.1.

Tabel 4.1 Hasil Pengujian Parameter Segmentasi berdasar Keunikan setiap Kelompok Citra

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *min\_size* | Kelompok Citra | | | | Total *Cluster* Unik |
| I | II | III | IV |
| 200 | 2 | 0 | 27 | 3 | 32 |
| 210 | 2 | 0 | 25 | 3 | 30 |
| 220 | 2 | 0 | 23 | 3 | 28 |
| 230 | 2 | 0 | 23 | 2 | 27 |
| 240 | 2 | 0 | 22 | 2 | 26 |
| 250 | 2 | 0 | 22 | 2 | 26 |
| 260 | 2 | 0 | 23 | 0 | 25 |
| 270 | 0 | 0 | 21 | 0 | 21 |
| 280 | 0 | 0 | 21 | 0 | 21 |
| 290 | **0** | **0** | **20** | **0** | **20** |
| 300 | 0 | 0 | 22 | 0 | 22 |
| 310 | 0 | 0 | 22 | 0 | 22 |
| 320 | 0 | 2 | 24 | 0 | 26 |
| 330 | 0 | 2 | 24 | 0 | 26 |
| 340 | 0 | 2 | 25 | 0 | 27 |

Berdasar hasil observasi di atas, nilai keunikan *cluster* terendah didapat pada *min\_size* = 290. Adapun untuk parameter setelahnya nilai total keunikan akan semakin tinggi. Sehingga nilai di atas yang akan digunakan untuk penelitian ini.

Setelah menentukan parameter untuk proses segmentasi. Selanjutnya melalui parameter tersebut dilakukan segmentasi terhadap keseluruhan citra *artificial* dan motif batik. Kemudian label segmentasi disimpan dalam format \*.csv di dalam sistem operasi.Tabel 4.2 menunjukkan sampel hasil segmentasi citra yang mewakili setiap kelas citra.

Tabel 4.2 Sampel Hasil Segmentasi Citra untuk Setiap Kelas

|  |  |  |
| --- | --- | --- |
| Nama File | Citra Asli | Citra Segmentasi |
| img030.jpg |  |  |
| img094.jpg |  |  |
| img101.jpg |  |  |
| img156.jpg |  |  |
| batik020.jpg |  |  |
| batik114.jpg |  |  |
| batik173.jpg |  |  |

## Ekstraksi Region Adjacency Graph

Pada tahap ini tidak ada parameter yang menentukan keakuratan pembangunan RAG. Sehingga seluruh citra akan langsung diproses dengan fungsi yang sama dan *class* hasil pembangunan RAG akan langsung disimpan dalam format \*.gpickle dalam sistem operasi. Gambar 4.2 menunjukkan operasi pembangunan RAG, serta penyimpanannya dalam sistem operasi.

|  |
| --- |
| # Pengulangan untuk setiap citra  **for** i **in** range**(**len**(**imgFile**)):**  img **=** io**.**imread**(**'blank.jpg'**)**  # membaca label dari \*.csv  label **=** genfromtxt**(**labelFile**[**i**],** delimiter**=**','**)**  label **=** label**.**astype**(**int**)**  # fungsi utama pembentukan RAG  g**=**graph**.**rag\_mean\_color**(**img**,**label**,**connectivity**=**2**,**mode**=**'distance'**)**  gshow **=** graph**.**show\_rag**(**label**,**g**,**img**,**edge\_width**=**4**)**  # operasi penyimpanan graph dalam \*.gpickle  **if(**i**<**9**):**  name **=** 'img00{}.gpickle'**.**format**(**i**+**1**)**  **elif** **(**i**<**99**):**  name **=** 'img0{}.gpickle'**.**format**(**i**+**1**)**  **else:**  name **=** 'img{}.gpickle'**.**format**(**i**+**1**)**  path3 **=** 'graph/'**+**name  nx**.**write\_gpickle**(**g**,** path3**)** |

Gambar 4.2 Operasi Pembangunan RAG

*Class* RAG yang telah disimpan, tidak dapat dibaca secara manual dalam *text editor*. Namun dapat dibaca kembali dengan mudah melalui fungsi dari modul *networkx*.*read\_gpickle*(). Tabel 4.3 berisi sampel hasil pembangunan RAG yang mewakili setiap kelas citra.

Tabel 4.3 Tabel Hasil Pembangunan RAG

|  |  |  |
| --- | --- | --- |
| Nama File | Citra Segmentasi | RAG |
| img030.jpg |  |  |
| img094.jpg |  |  |
| img101.jpg |  |  |
| img143.jpg |  |  |
| batik020.jpg |  |  |
| batik114.jpg |  |  |
| batik173.jpg |  |  |

## *Graph Matching*

Pada tahap ini akan diuji tingkat akurasi dari kedua metode *graph matching* yang telah disebutkan sebelumnya yaitu *graph isomorphism* dengan algoritma VF2 dan *error-tolerant matching* menggunakan *graph edit distance*. Kedua algoritma ini akan diujikan terhadap seluruh kelompok citra baik *artificial* maupun motif batik.

Sebelum dilakukan proses *graph matching*, data pemanggilan disimpan dalam bentuk \*.json untuk mempermudah dalam pemanggilan berkas-berkas yang dibutuhkan. Gambar 4.3 menunjukkan potongan program penyimpanan berkas dalam format \*.json pada sistem operasi.

|  |
| --- |
| # inisiasi list  path1 **=** 'img/'  path2 **=** 'labels'  path3 **=** 'graph/'  imgFile **=** **[]**  labelFile **=** **[]**  graphFile **=** **[]**  dataKu **=** **[]**  # ambil file dari sistem operasi  **for** filename1 **in** glob**.**glob**(**os**.**path**.**join**(**path1**,** '\*.jpg'**)):**  imgFile**.**append**(**filename1**)**  **for** filename2 **in** glob**.**glob**(**os**.**path**.**join**(**path2**,** '\*.csv'**)):**  labelFile**.**append**(**filename2**)**  **for** filename3 **in** glob**.**glob**(**os**.**path**.**join**(**path3**,** '\*.gpickle'**)):**  graphFile**.**append**(**filename3**)**  # penyimpanan dalam tipe data dictionary untuk setiap citra  **for** i **in** range**(**len**(**imgFile**)):**  data **=** **{**'key'**:** 0**,** 'img'**:** '0'**,** 'label'**:** '0'**,** 'graph'**:** '0'**}**  data**[**'img'**]** **=** imgFile**[**i**]**  data**[**'label'**]** **=** labelFile**[**i**]**  data**[**'graph'**]** **=** graphFile**[**i**]**  **if** **(**i **<** 50**):**  data**[**'key'**]** **=** 1  **elif** **(** i**<**100 **):**  data**[**'key'**]** **=** 2  **elif** **(** i**<**150 **):**  data**[**'key'**]** **=** 3  **else:**  data**[**'key'**]** **=** 4  # tambah ke list dataKu  dataKu**.**append**(**data**)**  # simpan dalam \*.json dalam sistem operasi  **with** open**(**'dataKu1.json'**,** 'w'**)** **as** outfile**:**  **for** hostDict **in** dataKu**:**  json**.**dump**(**hostDict**,** outfile**)**  outfile**.**write**(**'\n'**)** |

Gambar 4.3 Potongan Program Penyimpanan \*.json

### Algoritma VF2

Pada algoritma ini, proses *matching* langsung dapat dilakukan tanpa menentukan parameter bebas terlebih dulu. Algoritma ini akan langsung membandingkan kedua *graph* memiliki sifat *isomorphism* bernilai benar atau salah.

Dalam program ini operasi *isomorphism* telah disimpan dalam modul *networkx*. Pemanggilan fungsi dapat dilakukan dengan baris kode *gm.is\_isomorphic*(), yang akan menghasilkan output *boolean*. Gambar 4.4 menunjukkan potongan program dari operasi *graph matching* menggunakan algoritma ini.

|  |
| --- |
| # Pengulangan untuk setiap citra  **for** i **in** range**(**len**(**aList**)):**  # inisiasi ulang list yang cocok  matchedList **=** **[]**  img **=** io**.**imread**(**aList**[**i**][**'img'**])**  # segmentasi dan ekstrak RAG citra input  label **=** segmentation**.**felzenszwalb**(**img**,** min\_size**=**290 **)**  g1**=**graph**.**rag\_mean\_color**(**img**,**label**,**connectivity**=**2**,**mode**=**'distance'**)**  # perbandingan graph ke seluruh graph tersimpan  **for** j **in** range**(**0**,**len**(**aList**)):**  g2 **=** nx**.**read\_gpickle**(**aList**[**j**][**'graph'**])**  gm **=** isomorphism**.**GraphMatcher**(**g1**,**g2**)**  **if(**gm**.**is\_isomorphic**()):**  matchedList**.**append**(**aList**[**j**][**'img'**])** |

Gambar 4.4 Potongan Program Operasi Graph Matching dengan Algoritma VF2

### *Graph Edit Distance*

Pada *Graph Edit Distance* (GED), untuk proses *matching* dilakukan dengan menggunakan sebuah parameter yang berfungsi sebagai pembatas nilai *cost* dari transformasi menjadi .

Pada program ini algoritma disimpan dalam berkas “ged4py” yang berisi kelas, abstrak, dan algortima untuk pencarian GED. Hasil dari setiap operasi *matching* melalui GED berupa nilai *cost* dalam tipe data *integer*. Kemudian pembatasan diberikan melalui teknik *tracing* untuk menemukan hasil yang dianggap optimal. Gambar 4.5 menunjukkan potongan operasi utama *graph matching* menggunakan GED.

|  |
| --- |
| # Pengulangan untuk setiap Citra  **for** i **in** range**(**len**(**aList**)):**  # inisiasi list cocok  matchedList **=** **[]**  img **=** io**.**imread**(**aList**[**i**][**'img'**])**  label **=** segmentation**.**felzenszwalb**(**img**,** min\_size**=**290 **)**  g1 **=** graph**.**rag\_mean\_color**(**img**,**label**,**connectivity**=**2**,**mode**=**'distance'**)**  # matching terhadap seluruh citra  **for** j **in** range**(**0**,**len**(**aList**)):**  g2 **=** nx**.**read\_gpickle**(**aList**[**j**][**'graph'**])**  # hitung cost GED dan berikan nilai pembatas pada fungsi if  **if(**ged**.**compare**(**g1**,**g2**,False))** **<** 1**:**  matchedList**.**append**(**aList**[**j**][**'img'**])** |

Gambar 4.5 Potongan Program Operasi Utama Graph matching menggunakan Graph Edit Distance

### Hasil Evaluasi *Graph Matching*

Pada penelitian ini evaluasi dilakukan kedua jenis data terhadap data sesamanya, *artificial* terhadap *artificial*, dan motif batik terhadap motif batik. Setiap data dibandingkan dengan seluruh data sehingga pada citra *artificial* didapat masing-masing 200 buah nilai *precision*, *recall*, dan, *f-score*, sedangkan pada citra motif batik didapat 180 buah nilai *precision*, *recall*, dan, *f-score.* Kemudian untuk setiap jenis data, diambil sebuah nilai rata-rata untuk setiap algoritma/parameter yang diberikan. Gambar 4.6 merupakan bagian dari program evaluasi hasil *graph matching*.

|  |
| --- |
| aList **=** **[]**  **with** open**(**'dataKu1.json'**)** **as** f**:**  **for** line **in** f**:**  j\_content **=** json**.**loads**(**line**)**  aList**.**append**(**j\_content**)**  precisionList **=** **[]**  recallList **=** **[]**  nList **=** **[]**  **for** i **in** range**(**len**(**aList**)):**  matchedList **=** **[]**  relevant **=** 0  img **=** io**.**imread**(**aList**[**i**][**'img'**])**  label **=** segmentation**.**felzenszwalb**(**img**,** min\_size**=**290 **)**  g1**=**graph**.**rag\_mean\_color**(**img**,**label**,**connectivity**=**2**,**mode**=**'distance'**)**  **for** j **in** range**(**0**,**len**(**aList**)):**  g2 **=** nx**.**read\_gpickle**(**aList**[**j**][**'graph'**])**  gm **=** isomorphism**.**GraphMatcher**(**g1**,**g2**)**  **if(**gm**.**is\_isomorphic**()):**  matchedList**.**append**(**aList**[**j**][**'img'**])**  **if** **(**aList**[**i**][**'key'**]** **==** aList**[**j**][**'key'**]):**  relevant **+=** 1  n **=** len**(**matchedList**)**  precision **=** **(**relevant**/**n**)**  recall **=** **(**relevant**/**50**)**  precisionList**.**append**(**precision**)**  recallList**.**append**(**recall**)**  nList**.**append**(**n**)**  p **=** np**.**mean**(**np**.**array**(**precisionList**))**  r **=** np**.**mean**(**np**.**array**(**recallList**))**  n **=** np**.**mean**(**np**.**array**(**nList**))**  fm **=** **(**2**\***p**\***r**)/(**p**+**r**)**  **print** **(**'nilai precision : %.2f %%' **%** **(**p**\***100**))**  **print** **(**'nilai recall : %.2f %%' **%** **(**r**\***100**))**  **print** **(**'nilai f-measure : %.2f %%' **%** **(**fm**\***100**))** |

Gambar 4.6 Potongan Program Evaluasi Graph Matching

Program di atas dijalankan berulang secara *trial & error* untuk kedua jenis citra. Hal ini dilakukan agar mendapatkan hasil evaluasi yang baik dari kedua algoritma dan parameternya. Tabel 4.4 menunjukkan hasil evaluasi dari proses *graph matching* yang telah diujikan dengan berbagai parameter.

Tabel 4.4 Hasil Evaluasi Graph Matching

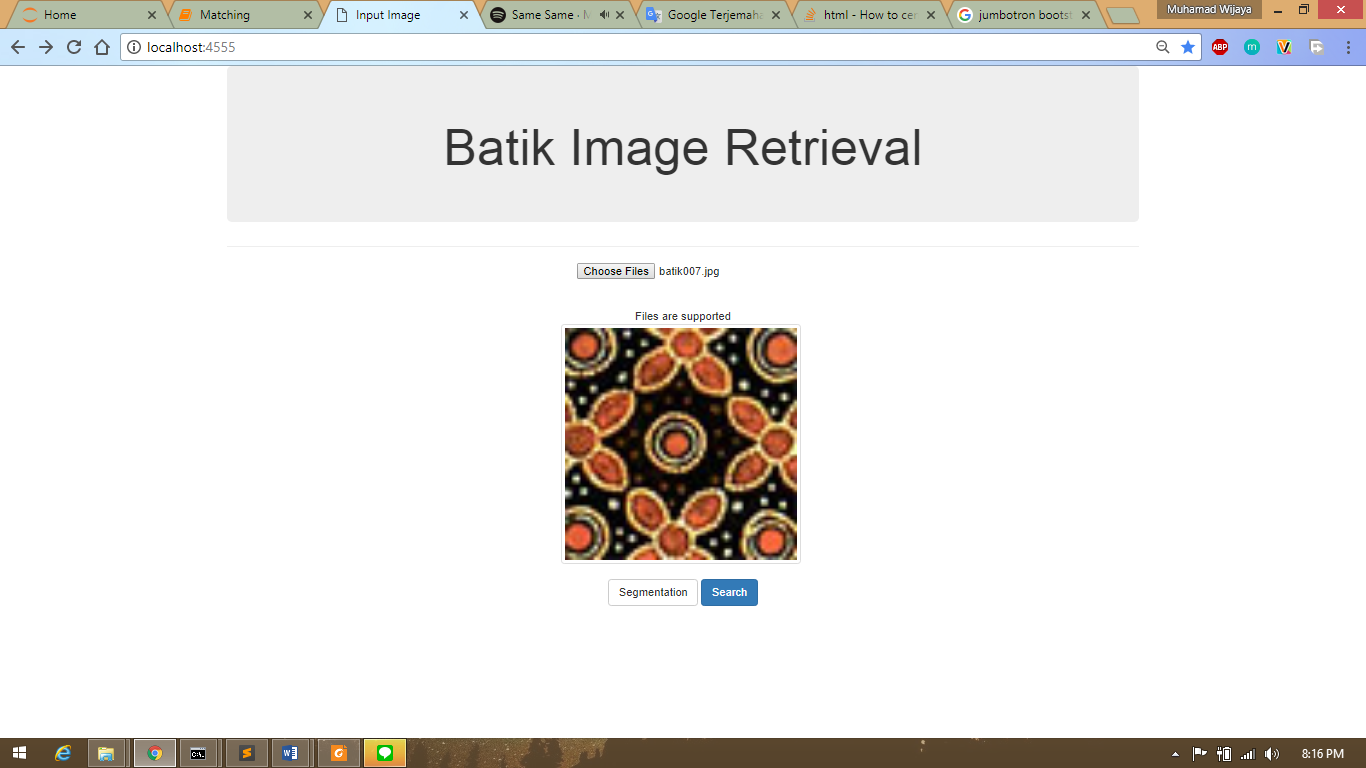
|  |  |  |  |
| --- | --- | --- | --- |
| Data |  |  |  |
| *Artificial* (VF2) | 96.24 | 84.52 | **90.00** |
| *Artificial* () | 97.08 | 78.94 | 87.07 |
| *Artificial* () | 97.08 | 78.94 | 87.07 |
| *Artificial* () | 46.72 | 81.12 | 59.29 |
| *Artificial* () | 38.65 | 82.40 | 52.61 |
| *Artificial* () | 31.49 | 87.56 | 45.65 |
| Batik (VF2) | 87.78 | 19.41 | 31.79 |
| Batik () | 96.67 | 12.30 | 21.82 |
| Batik () | 75.83 | 36.30 | 49.09 |
| Batik () | 63.79 | 53.48 | 58.18 |
| Batik () | 59.52 | 60.00 | **59.76** |
| Batik () | 47.46 | 65.63 | 55.08 |

Berdasarkan tabel di atas, hasil evaluasi terbaik pada data citra *artificial* diperoleh melalui algoritma VF2 dengan nilai 90.00 %. Baik nilai *precision* maupun *recall* untuk algoritma ini memiliki nilai yang baik yaitu dan . Kemudian pada data citra motif inti batik, hasil tertinggi dicapat melalui *Graph Edit Distance* dengan parameter . Nilai yang diperoleh sebesar , serta *precision* dan *recall* yaitu dan .

## Uji Coba Halaman Antarmuka

Pada tahap ini akan diujikan program dengan parameter yang telah dianalisa sebelumnya ke dalam bentuk program antarmuka berbasis *web*. Halaman yang akan diujikan meliputi halaman masukan, halaman segmentasi, dan halaman hasil pengambilan citra.

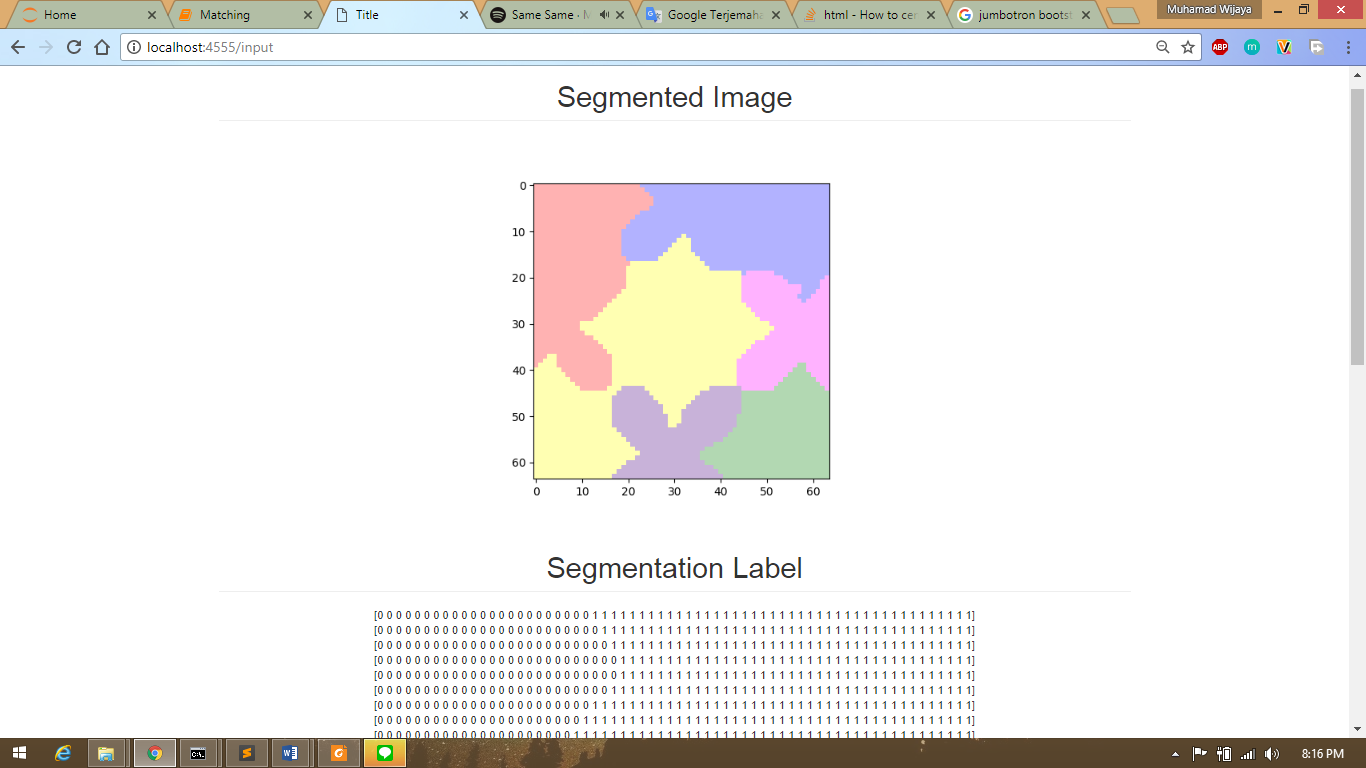
### Halaman Masukan



Gambar 4.7 Halaman Masukan Citra

Halaman ini dapat ditampilkan dengan mengetikan *localhost*:4555 di *address bar* pada *browser*. Dapat dilihat pada Gambar 4.7 bahwa keseluruhan fungsi pada halaman ini sudah dapat beoperasi. Mulai dari fungsi pilih berkas, hingga menampilkannya dalam kotak citra yang tersedia dan menyimpan alamat berkasnya dalam program. Begitupun fungsi dari kedua *button*, yang akan dijelaskan di sub bab berikutnya.

### Halaman Segmentasi



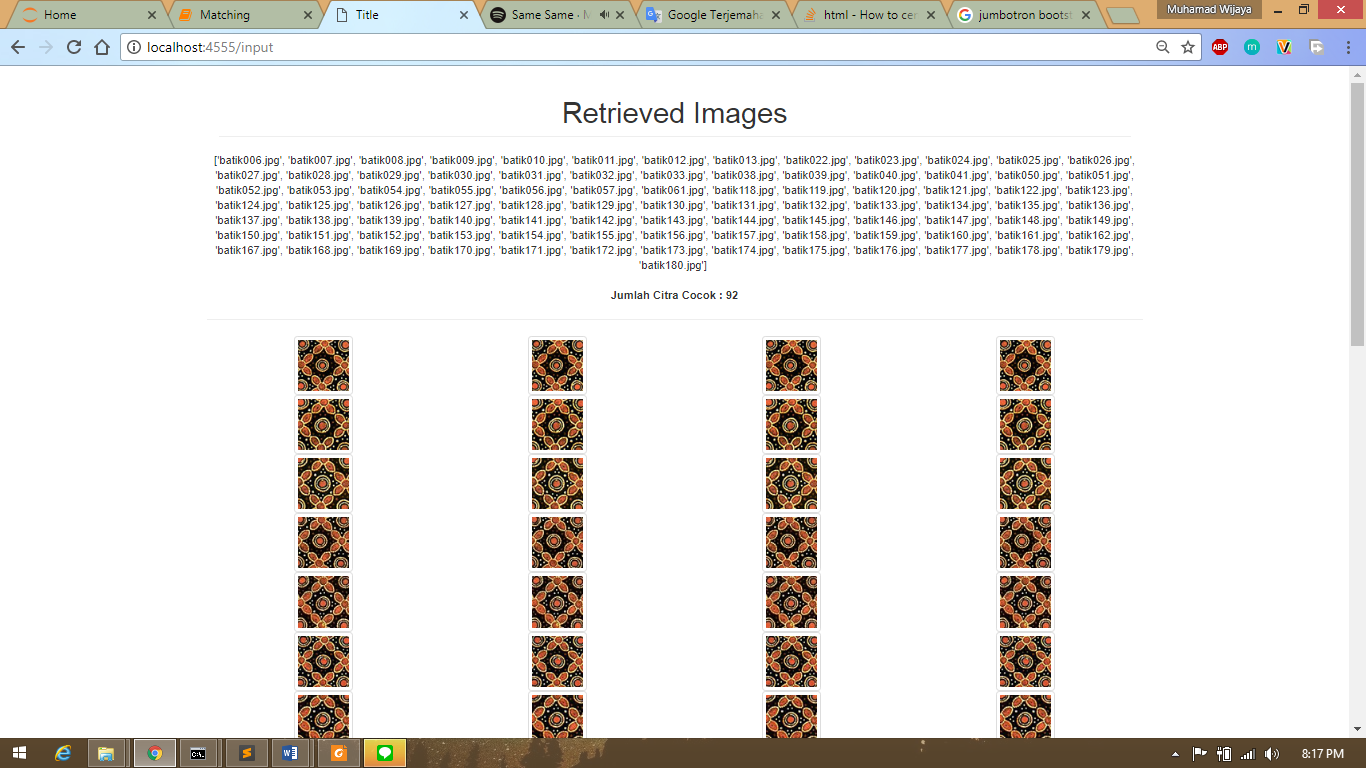
Gambar 4.8 Halaman Hasil Segmentasi

Pada Gambar 4.8 ditunjukkan halaman hasil segmentasi dari citra masukan. Melalui fungsi yang dipanggil dari *button “Segment”*, *python* dapat menangkap alamat berkas citra yang diambil dari “*halaman input.html”*. Hasil proses dari operasi segmentasi pada *python* juga dapat di-*render* dan dicetak kembali pada “*segmentation.html*”, sehingga halaman ini dapat bekerja sesuai fungsinya.

### Halaman Hasil *Matching*

Berkas citra yang telah disimpan dari halaman masukan, kemudian diproses dalam *python* melalui algoritma *graph matching* seperti pada program analisis. Adapun algoritma yang digunakan merupakan hasil terbaik yang diambil dari hasil evaluasi penelitian.

Untuk data citra *artificial* algortima *graph matching* yang digunakan adalah algoritma VF2. Sedangkan untuk data citra motif inti batik menggunakan *Graph Edit Distance* dengan parameter . Gambar 4.9 menunjukkan halaman hasil *Graph Matching* citra masukan dengan data citra motif inti batik.



Gambar 4.9 Halaman Hasil Graph Matching untuk Citra Motif Inti Batik