

TUGAS BESAR DETEKSI DAN KLASIFIKASI CACAT PADA CIRCUIT BOARD

Fikki Maulana Aulia Siswanto (13217033)
Anindyo Putra Hadindra Soelistyono (13217037)
Rizky Ardi Maulana (13217054)

Tanggal Percobaan: 01/12/2020
EL4125-Pengolahan Citra Digital

Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Pada proyek tugas besar mata kuliah EL4125 Pengolahan Citra Digital ini kami memilih judul yaitu deteksi dan klasifikasi cacat pada circuit board. Kami melakukan proses pendeteksian kecacatan yang ada pada PCB yang diproduksi massal, kemudian kami melakukan klasifikasi jenis cacat yang ada pada desain PCB tersebut. Pertama kami terlebih dahulu melakukan *Defect Detection and Extraction* untuk mendapatkan data yang sekiranya terdapat *defect*, kemudian kami melakukan *labelling* untuk setiap data yang sekiranya terdapat *defect*, diantaranya adalah *spour*, *spourius cooper*, dan *short*. Setelah itu dilakukan *training model* di tahap *Defect Classification* untuk mendapatkan hasil *training* untuk prediksi di tahap selanjutnya, yaitu tahap PCB *Defect Classifier Testing*, dimana di tahap ini dilakukan testing menggunakan *Training Set* yang kemudian akan digunakan *Valiation Set* untuk memastikan apakah model yang dibuat sudah memenuhi spesifikasi atau belum.

Kata kunci: *Defect Detection and Extraction, Defect Labelling, Defect Classification, PCB Defect Classifier Testing.*

1. PENDAHULUAN

Papan sirkuit cetak (PCB) secara mekanis mendukung dan secara elektrik menyambungkan komponen listrik atau elektronik menggunakan trek konduktif, bantalan dan fitur lain yang diukir dari satu atau lebih lapisan lembaran tembaga yang dilaminasi di antara lapisan lembaran substrat non-konduktif. PCB adalah komponen penting dalam perangkat elektronik, yang menjadi fondasi sekaligus sambungan rangkaian. Awalnya papan sirkuit dirancang secara manual dan desain tangan bebas, dan berkembang mengikuti kebutuhan produksi massal dan efisiensi ukuran sirkuit menghasilkan teknologi cetak papan sirkuit seperti yang sudah ada saat ini.

Dalam proses produksinya, PCB yang selesai dicetak perlu diperiksa ulang kesesuaian dengan desain dan fungsi yang diinginkan. Pemeriksaan dapat dilakukan secara manual menggunakan tenaga manusia, namun proses tersebut membutuhkan waktu yang lama dan sumber daya manusia yang besar. Oleh karena itu, untuk

meningkatkan produktivitas cetak PCB dikembangkan teknologi deteksi cacat PCB menggunakan teknologi pengolahan citra.

Deteksi cacat PCB menggunakan PCB template yaitu papan sirkuit yang telah diperiksa kesesuaian dengan desain dan fungsionalitasnya, dan PCB yang akan dideteksi adanya cacat. Algoritma secara umum adalah membandingkan kedua PCB tersebut untuk menentukan apakah ada kesalahan dalam PCB yang diperiksa. Metode deteksi memanfaatkan modul pengolahan citra *binaryzation, morphological operation*, dan operasi matematika citra. Pada tahap berikutnya, cacat yang dideteksi dapat diklasifikasi jenis cacatnya menggunakan model *machine learning* yang dilatih menggunakan dataset jenis cacat PCB.

Dalam dokumen ini, studi deteksi dan klasifikasi cacat pada *circuit board* memiliki tujuan sebagai berikut:

- Melakukan deteksi cacat pada circuit board berdasarkan circuit board template.
- Melakukan klasifikasi cacat pada circuit board.
- Membuat model machine learning yang melakukan deteksi dan klasifikasi cacat pada *circuit board* secara otomatis.

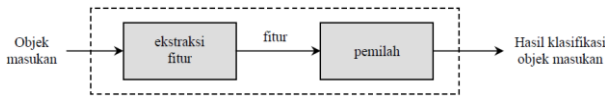
Studi dilakukan menggunakan MATLAB dan dataset yang tersedia pada halaman berikut <http://>

2. STUDI PUSTAKA

2.1 PENGENALAN

Berbeda dengan disiplin ilmu pengolahan citra yang dibatasi oleh penggunaan citra sebagai masukan maupun keluarannya, suatu aplikasi pengenalan pola bertujuan untuk melakukan proses pengenalan terhadap suatu objek (misalnya citra) ke dalam salah satu kelas tertentu, berdasarkan pola yang dimilikinya.

Secara umum, proses pengenalan pola ini dapat digambarkan dalam diagram blok sederhana berikut:



Gambar 2.1 Diagram Blok sistem pengenalan pola

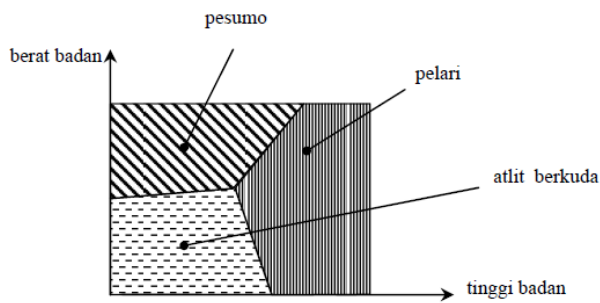
Pengertian pola (pattern) dapat dibedakan dengan istilah ciri/fitur (feature) sebagai berikut:[1]

- Fitur: segala jenis aspek pembeda, kualitas, maupun karakteristik.
- Pola: komposit/gabungan dari fitur yang merupakan sifat dari sebuah objek.

Kualitas dari suatu vektor fitur dapat dilihat dari kemampuannya dalam melakukan pemilahan objek berdasarkan keanggotaan kelasnya:

- Objek dari kelas yang sama harus memiliki vektor fitur yang sama
- Objek dari kelas yang berbeda harus memiliki vektor fitur yang berlainan

Tugas dari pemilah (classifier) adalah untuk menyekat ruang fitur ke dalam daerah-daerah yang dilabeli sebagai kelas yang berbeda.[1]



Gambar 2.2 Contoh klasifikasi atlet berdasarkan vektor fitur tinggi dan berat badan

Metoda pengenalan pola terbagi dalam tiga kelompok, yaitu: [1]

a. Statistik (statistical)

Proses pemilahan dilakukan berdasarkan model statistik dari fitur, yang didefinisikan sebagai suatu keluarga dari fungsi kerapatan peluang kelas bersyarat $P(x | c_i)$ - peluang vektor fitur x apabila diberikan kelas c_i .

b. Sintaktik (syntactic)

Pemilahan berdasarkan keserupaan ukuran struktural. Dengan cara ini, deskripsi hirarkis suatu pola kompleks dapat diformulasikan sebagai gabungan dari beberapa pola yang lebih sederhana.

Pada metoda ini, pengetahuan direpresentasikan secara formal grammar atau deskripsi relasional (graf).

c. Jaringan saraf (neural network)

Pemilahan dilakukan berdasarkan tanggapan suatu neuron jaringan pengolah sinyal terhadap suatu stimulus masukan (pola). Metoda jaringan saraf tiruan menyimpan pengetahuan dalam bentuk arsitektur jaringan dan kekuatan pembobot sinaptik.

2.2 PENGENALAN POLA SECARA STATISTIK

Perbincangan mengenai metoda statistik dalam persoalan pengenalan pola tidak dapat dilepaskan dari teorema Bayes. Secara matematis, teorema ini didefinisikan menurut relasi berikut: [1]

$$P[\omega_j | x] = \frac{P[x | \omega_j] \cdot P[\omega_j]}{P[x]} = \frac{P[x | \omega_j] \cdot P[\omega_j]}{\sum_{k=1}^n P[x | \omega_k] \cdot P[\omega_k]}$$

(Persamaan 1)

Secara prinsip, proses pemilahan dalam suatu aplikasi pengenalan pola adalah dengan memilih kelas ω_i yang paling mungkin apabila diketahui suatu vektor fitur x . Atau dengan kata lain, dalam persoalan pemilahan vektor ciri satu dimensi x ke dalam dua kelas ω_1 dan ω_2 berlaku:

“apabila $P(\omega_1 | x) > P(\omega_2 | x)$, pilih kelas ω_1 , selain itu pilih kelas ω_2 ”

yang dapat dinyatakan secara matematis sebagai:

$$P[\omega_1 | x] > P[\omega_2 | x]$$

(Persamaan 2)

atau

$$\Lambda(x) : \frac{P(\omega_1 | x)}{P(\omega_2 | x)} > 1$$

(Persamaan 3)

Suku $\Lambda(x)$ disebut nisbah likelihood, dan aturan keputusan (3) disebut uji nisbah likelihood (UNL). Persamaan (3) tersebut dapat diperluas dengan memasukkan parameter fungsi biaya C_{ij} , yaitu penalti atas kesalahan memilih kelas ω padahal

kelas sesungguhnya adalah ω_1 . Modifikasi ini menyebabkan UNL yang dihasilkan menjadi:[1]

$$\Lambda(x) : \frac{P(\omega_1 | x)}{P(\omega_2 | x)} > \frac{\omega_1}{\omega_2} \frac{C_{12} - C_{22}}{C_{21} - C_{11}}$$

(Persamaan 4)

Substitusi persamaan 1 dan 4 akan menghasilkan :

$$\Lambda(x) : \frac{P(x | \omega_1)}{P(x | \omega_2)} > \frac{(C_{12} - C_{22})P(\omega_2)}{(C_{21} - C_{11})P(\omega_1)}$$

(Persamaan 5)

Persamaan (5) ini dikenal sebagai Kriteria Bayes, yaitu aturan keputusan UNL yang meminimumkan Resiko Bayes secara umum. Dua macam bentuk khusus dari persamaan ini adalah Kriteria A-Posteriori Maksimum (MAP) dan Kriteria Likelihood Maksimum (ML).

1. Kriteria MAP

Penerapan kriteria MAP dilakukan pada proses klasifikasi di mana kedua kelas yang diamati (diasumsikan) memiliki fungsi biaya yang satu-nol.

$$C_{11} = C_{22} = 0$$

$$C_{12} = C_{21} = 1$$

2. Kriteria ML

Kriteria ini merupakan penyederhanaan lebih jauh dari kriteria MAP, yaitu dengan menarik asumsi bahwa peluang prior dari kedua kelas sama besar:

$$P(\omega_1) = P(\omega_2) = \frac{1}{2}$$

Sehingga :

$$\Lambda(x) : \frac{P(x | \omega_1)}{P(x | \omega_2)} > 1$$

2.3 JARINGAN SYARAF TIRUAN

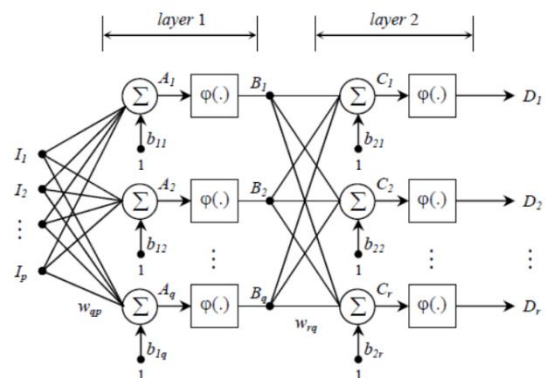
Jaringan syaraf tiruan (JST) merupakan mesin yang didesain untuk memodelkan cara yang digunakan otak untuk mengerjakan suatu fungsi tertentu. Suatu JST berwujud sebagai jaringan yang terdiri oleh satu atau beberapa neuron yang terhubung satu sama lain dalam bobot yang berbeda-beda. Jaringan ini memiliki kemampuan untuk memperoleh pengetahuan dari suatu proses belajar, dan kemudian menyimpannya untuk kebutuhan penggunaan berikutnya.

Pengembangan JST dipicu oleh adanya pemikiran bahwa otak manusia melakukan komputasi dalam cara yang sama sekali berbeda dengan pada komputer digital. Oleh karena itu, JST didesain untuk memiliki beberapa persamaan sifat dengan otak, yaitu dalam hal:

- Pengetahuan yang dimiliki diperoleh dari suatu proses belajar.
- Pengetahuan disimpan dalam bentuk bobot koneksi antar neuron.

Suatu JST dikatakan memiliki arsitektur feedforward apabila setiap output dari setiap neuron diteruskan dari sisi input ke sisi outputnya. Dalam sebuah arsitektur feedforward tersebut, neuron dapat dikelompokkan dalam satu atau lebih layer, di mana setiap layer tidak memiliki hubungan feedback dengan layer lainnya. Ilustrasi mengenai suatu arsitektur feedforward-network yang tersusun oleh 2 buah layer diberikan pada Gambar 4.

Algoritma belajar yang amat populer bagi arsitektur feedforward-network adalah algoritma backpropagation. Pada algoritma ini, proses pelatihan terhadap jaringan berjalan melalui dua proses propagasi, yaitu propagasi maju dan propagasi balik.



Gambar 2.3 Ilustrasi feedforward network dengan 2 layer

Propagasi maju dikerjakan pada vektor input setahap demi setahap, hingga dihasilkan nilai D(k) pada layer output. Pada iterasi ke n, nilai dari masing-masing variabel jaringan adalah:

$$A_j(n) = \sum_{i=1}^p w_{ji} \cdot I_i(n) + b_{j1}$$

$$B_j(n) = \varphi(A_j(n))$$

$$C_k(n) = \sum_{j=1}^q w_{kj} \cdot B_j(n) + b_{k2}$$

$$D_k(n) = \varphi(C_k(n))$$

Respon aktual dari jaringan tersebut diberikan oleh Dk. Apabila diinginkan vektor target (label

numerik dari masing-masing kelas) adalah T_k , maka besar error yang terjadi pada iterasi ke- n tersebut adalah:

$$e_k(n) = T_k - D_k(n)$$

Sum of squared error jaringan untuk iterasi ke- n adalah jumlah dari squared error untuk seluruh neuron pada layer 2:

$$E(n) = \sum_{k=1}^r e_k^2(n)$$

Dengan N vektor target, diperoleh nilai MSE sebagai squared error rata-rata jaringan, yaitu:

$$\text{MSE}(n) = \frac{1}{N} \sum_{k=1}^r e_k^2(n)$$

Propagasi balik bekerja dengan menggunakan faktor koreksi $\Delta w(n)$ untuk bobot $w_{kjk}(n)$, yang nilainya sebanding dengan gradien $\delta E(n)/\delta w(n)$. Nilai $\delta E(n)/\delta w_{kjk}(n)$ ini selanjutnya akan diamati untuk menentukan arah pencarian dalam ruang bobot bagi $w_{kjk}(n)$.

Apabila nilai MSE yang diperoleh hingga iterasi tertentu belum memenuhi kriteria error yang diharapkan, maka proses training dapat diulangi kembali dengan menggunakan data yang sama.

2.4 RESNET34 ARCHITECTURE

ResNet (Residual Neural Network) adalah kerangka *deep residual learning* untuk tugas klasifikasi gambar. Yang mendukung beberapa konfigurasi arsitektural, memungkinkan untuk mencapai rasio yang sesuai antara kecepatan kerja dan kualitas.

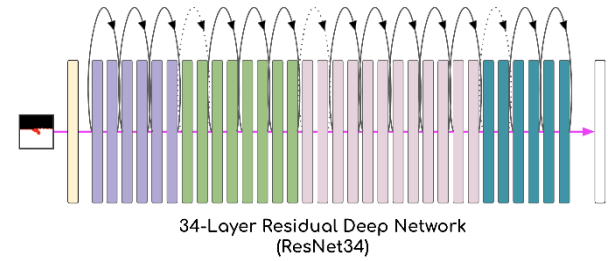
Residual Network menjadi populer untuk tugas pengenalan dan klasifikasi gambar karena kemampuannya untuk menyelesaikan gradien yang hilang dan meledak saat menambahkan lebih banyak lapisan ke jaringan saraf yang sudah dalam. Hal ini karena dimasukkan koneksi pintas yang memutar jaringan ke dalam versi residu pasangannya, yaitu memasukkan informasi dari 2 layer sebelumnya untuk proses konvolusi dalam menentukan layer berikutnya.

Dalam pengujian, arsitektur ResNet memiliki nilai error validasi yang lebih baik daripada model plain network:

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	25.03

Gambar 2.4 Tabel top-1 error (%) terhadap ImageNet validation [2]

Dalam arsitektur ini akan digunakan Residual Network 34-layer untuk pembelajaran. *Pre-trained* model ResNet model dapat diunduh melalui fastai.



Gambar 2.5 Ilustrasi building block dari ResNet34

2.5 APPROACH YANG DILAKUKAN

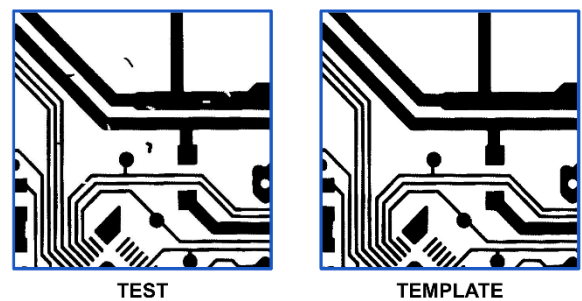
Klasifikasi defect PCB dapat menggunakan pendekatan hybrid untuk membangun aplikasi pengklasifikasi, yang menggabungkan metode referensi dan metode non-referensi.

Metode Referensi: papan PCB yang diinspeksi (papan uji atau *test*) dibandingkan dengan gambar *template* referensi (papan *template*) yang tidak memiliki cacat, untuk memungkinkan identifikasi perbedaan.

Metode Non-Referensi: papan PCB yang diinspeksi diperiksa kesesuaiannya dengan prinsip-prinsip desain, tanpa bergantung pada gambar *template*.

Metode referensi relatif mudah untuk diterapkan tetapi memiliki persyaratan penyimpanan yang besar (menampung semua gambar). Metode non-referensi tidak mengalami masalah penyimpanan, tetapi dapat dibatasi dalam jumlah cacat yang ditemukan.

Walaupun model *deep learning* dapat dilatih untuk mengklasifikasikan cacat yang ada pada seluruh papan, akan timbul tantangan untuk pelatihan model yang dilakukan mengingat banyaknya fitur yang harus diamati. Perhatikan contoh test dan template papan PCB berikut ini:

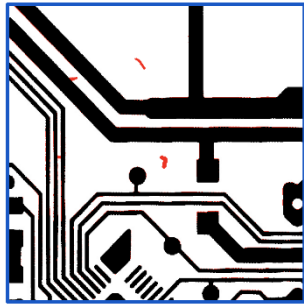


Gambar 2.6 Contoh gambar PCB *test* dan *template* setelah melewati tahap preprocessing

Untuk menurunkan beban yang di-cover oleh model, pertama-tama akan diekstrak cacat individu dari gambar *test*. Hal ini dapat dilakukan dengan membuat gambar perbedaan (atau

differenced image) dan mengekstraksi bagian gambar yang berbeda.

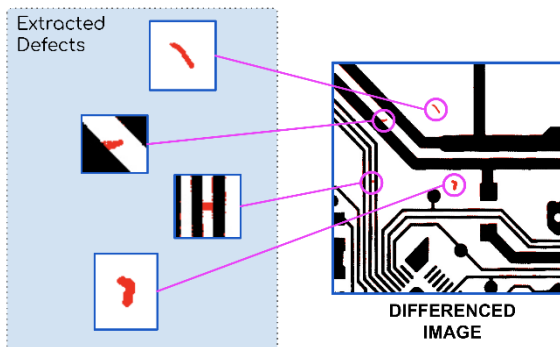
Berikut adalah gambar perbedaan yang didapatkan menggunakan pemrosesan gambar *image subtraction* dengan Python:



Gambar 2.6 Contoh gambar perbedaan antara *test* dan *template* (*differenced*)

Perhatikan cacat yang disorot dengan warna merah. Gambar di atas tidak menunjukkan semua cacat bergantung dari metode *differencing* yang dilakukan.

Dengan gambar perbedaan yang menyoroti cacat pada gambar uji, berikutnya dapat dilakukan pemrosesan gambar untuk mengekstrak bagian berwarna dari gambar:



Gambar 2.6 Contoh gambar hasil ekstraksi defect dari *differenced image*

Dengan menggunakan gambar hasil ekstraksi ini sebagai dataset, beban yang ditempatkan pada pengklasifikasi *deep learning* akan berkurang secara signifikan.

3. METODOLOGI

3.1 ALAT DAN BAHAN YANG DIGUNAKAN

Berikut adalah alat dan bahan yang digunakan pada praktikum ini.

- 1) Komputer dengan koneksi internet.
- 2) Akses ke *coding environment* Google Colab untuk menggunakan Python.
- 3) Dataset PCB yang telah melalui tahap *preprocessing* menjadi citra *circuit board* biner. Dataset yang digunakan diperoleh dari halaman GitHub oleh tangsanli5201 <https://github.com/tangsanli5201/Deep-PCB>
- 4) Library *tetryon_ai* dari halaman GitHub oleh Sean-McClure https://github.com/sean-mcclure/tetryon_ai.git
- 5) Produk Defect Turker untuk labelling defect diambil dari halaman GitHub oleh Sean-McClure https://github.com/sean-mcclure/defect_turker.git

3.2 LANGKAH-LANGKAH TIAP PERCOBAAN

Berikut adalah langkah-langkah untuk percobaan dalam bentuk diagram alir.

3.2.1 DEFECT DETECTION AND EXTRACTION

- a. Menggunakan dataset dari halaman GitHub oleh tangsanli5201.
- b. Mempersiapkan Python 3 notebook pada Google Collab. Langkah berikutnya dilakukan pada Collab.
- c. Install dan import library yang diperlukan.
- d. Membuat folder pada *workspace* dan *clone* dataset PCB.
- e. Membandingkan gambar PCB dari test dengan template menghasilkan "*differenced image*".
- f. Mengekstraksi defect dari *differenced image*.
- g. Menyimpan gambar hasil ekstraksi defect dalam *.zip* dan file labelnya dalam *.json*

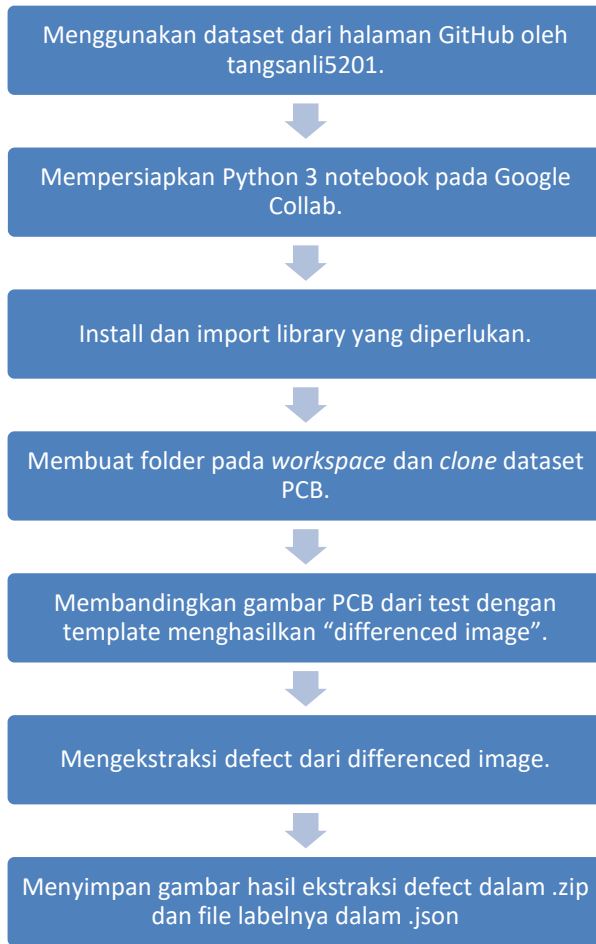


Diagram 3.2.1 Diagram alir langkah percobaan 3.2.1

3.2.2 DEFECT LABELLING

- Menggunakan file gambar hasil ekstraksi defect pada tahap sebelumnya.
- Menggunakan produk "Defect Turker" oleh Sean-Mclure. Program Defect Turker dijalankan secara lokal di PC.
- Menjalankan web service sederhana dengan perintah `python3 -m http.server` untuk meng-execute program.
- Menambahkan .zip gambar hasil ekstraksi defect dan label .json.
- Memulai aplikasi dengan akses `http://localhost:8000/` melalui browser.
- Melakukan labelling terhadap seluruh gambar defect latih yang muncul pada aplikasi berdasarkan klasifikasi defect yang sesuai.
- Mengunduh hasil labelling dari aplikasi dalam format .csv

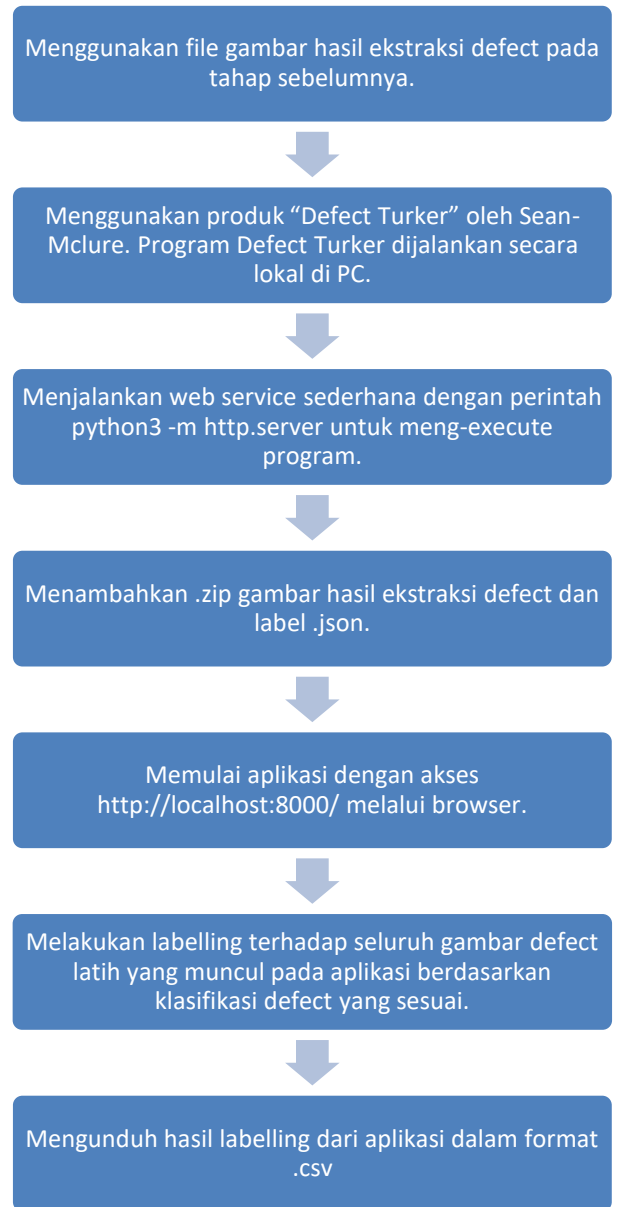


Diagram 3.2.2 Diagram alir langkah percobaan 3.2.2

3.2.3 DEFECT CLASSIFICATION

- Menggunakan gambar defect hasil ekstraksi dan file labelling dari tahap sebelumnya.
- Kembali ke Google Collab. Langkah berikutnya dilakukan pada Collab.
- Instal dan import library yang diperlukan.
- Mengunggah file .csv yang berisi hasil labelling gambar ekstraksi defect.
- Mengkonversi data pada .csv sehingga menjadi data frame yang sesuai.
- [OPTIONAL] membuat plot dengan Matplotlib

- g. Membuat list dan folder terpisah untuk masing-masing kelas jenis defect.
- h. Menambahkan gambar ekstraksi defect ke folder masing-masing. Gambar defect yang tidak dilabeli dapat dihapus.
- i. Membaca data citra yang akan digunakan untuk training classifier. Langkah ini sebaiknya diperiksa apakah citra dibaca dengan benar.
- j. Mengunduh pre-trained classifier dari fastai. Penulis menggunakan arsitektur ResNet34
- k. Melatih ResNet34 pada data defect. Model dilatih untuk 10 epoch.
- l. Meng-*unfreeze* model, kemudian mencari range dari learning rate yang baik
- m. Melatih model dengan differential learning rate
- n. Periksa confusion matrix (CM) untuk melihat adanya misclassification. Apabila diperoleh CM yang baik, dapat lanjut ke langkah berikutnya.
- o. Simpan dan export model hasil latih untuk prediksi.
- p. Mengunduh file .json dengan holdout name, dan gambar holdout untuk digunakan dalam PCB Defect Classifier.

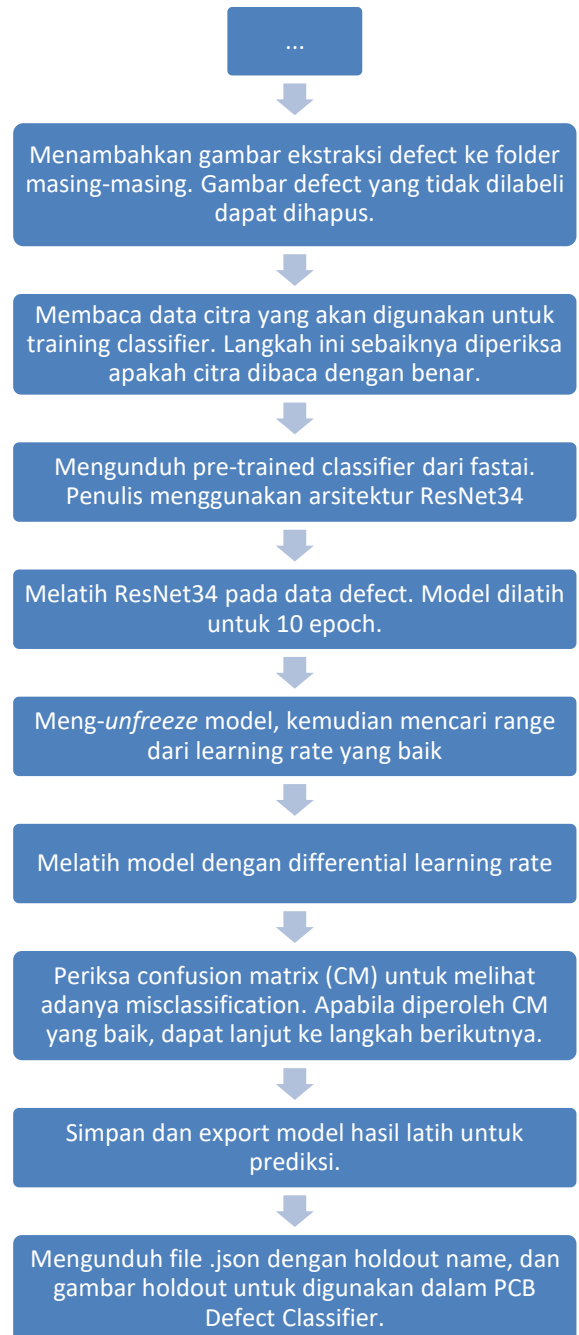
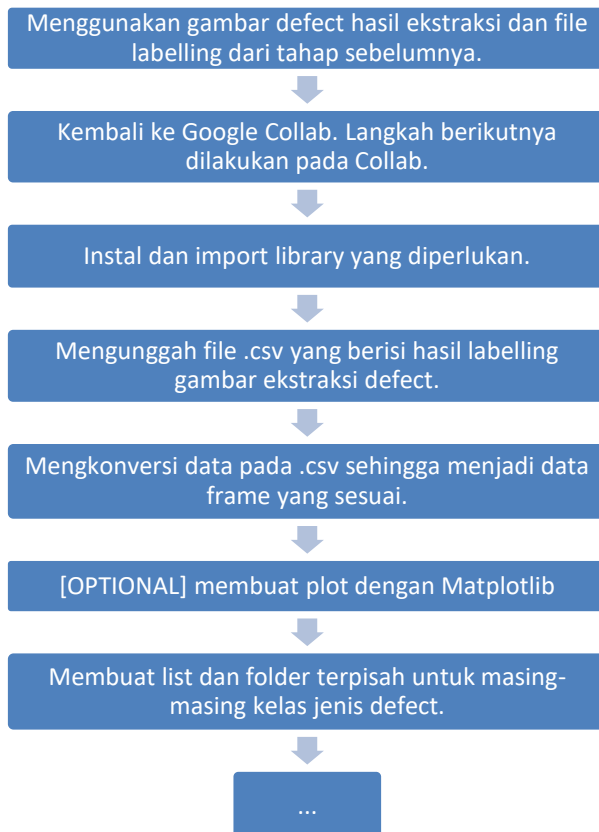


Diagram 3.2.3 Diagram alir langkah percobaan 3.2.3

3.2.4 PCB DEFECT CLASSIFIER TESTING

- a. Menggunakan model classifier dan file dari tahap sebelumnya.
- b. File gambar akan dilewatkan serangkaian proses deteksi.
- c. Gambar melewati proses subtraksi (test dikurangi template) dan disimpan pada folder *difference_image_test*.
- d. Hasil subtraksi melewati proses feature extraction untuk diambil kontur defect dan disimpan pada folder *defect_detected_test*.

- e. Hasil extracted defect melewati proses prediksi yang mengandung model *classifier* dari percobaan sebelumnya.
- f. Hasil prediksi klasifikasi defect dapat diamati.

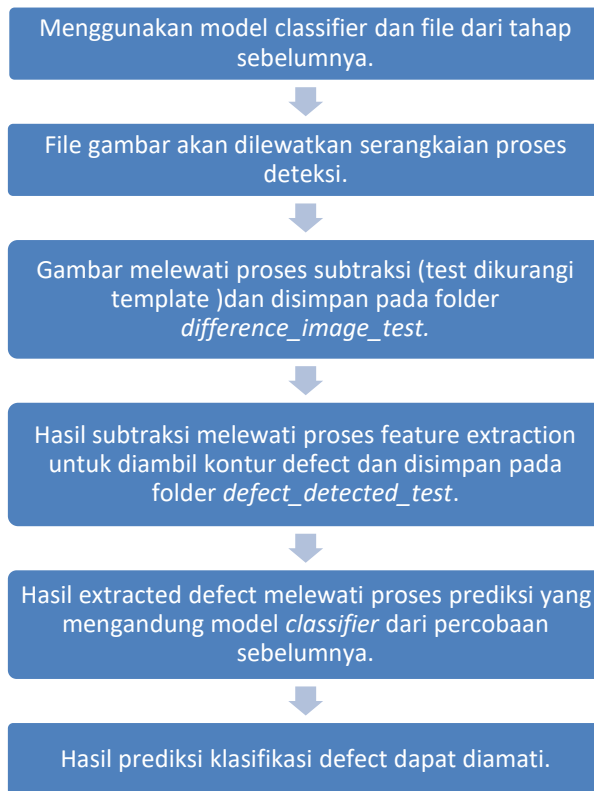


Diagram 3.2.4 Diagram alir langkah percobaan 3.2.4

4. HASIL DAN ANALISIS

Proses melatih model dilakukan menggunakan platform Google Collab.

Dataset yang digunakan untuk melatih model defect classifier dan pengujiannya diambil dari halaman GitHub oleh tangsanli5201 <https://github.com/tangsanli5201/DeepPCB>.

Dataset tersebut berupa gambar PCB *template* dan *test*, yang telah melalui tahap *preprocessing* yaitu pengolahan citra untuk menyiapkan pelatihan model *deep learning*. Tahap *preprocessing* yang dilakukan pada gambar antara lain penyesuaian orientasi, menghilangkan noise, dan *binaryzation*.

Menggunakan data tersebut, dilakukan serangkaian percobaan diperoleh hasil sebagai berikut.

4.1 DEFECT DETECTION AND EXTRACTION

Pada tahap ini akan dilakukan deteksi dan ekstraksi *defect* PCB dari gambar PCB *test* dengan referensi gambar PCB *template*.

Pertama akan dilakukan instalasi dan import library yang diperlukan:

```
!pip install git+https://github.com/sean-mcclure/tetryon_ai.git
from tetryonai.main import *
```

Berikutnya dapat dibuat folder directory untuk menyimpan *differenced image*, *extracted defect*, dan *hold out* gambar untuk menguji model.

- ▶ difference_img
- ▶ extracted_defects
- ▶ hold_out

Akses dataset didapatkan dengan *clone* project **DeepPCB** oleh tangsanli5201 ke work environment dengan:

```
!git clone
https://github.com/tangsanli5201/DeepPCB.git
```

Dalam percobaan ini, akan digunakan dataset dalam folder group00041 untuk melatih model classifier.

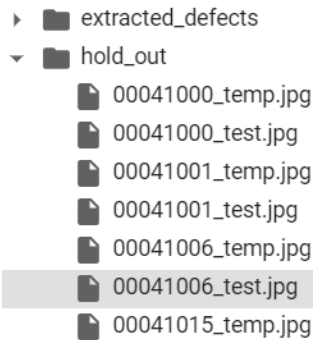
Gambar pada folder group00041 akan dipisahkan menjadi 2 bagian, yaitu gambar yang akan diekstraksi defectnya dan gambar yang akan dijadikan gambar pengujian model. Gambar untuk menguji model disediakan 20% dari gambar total dengan:

```
split_percentage =
round(len(temps_and_tests['group00041']['
temps']) * 0.2)
```

Dengan iterasi `index` dan `conditional index < split_percentage` gambar dalam group00041 dapat dipisahkan ke folder yang diinginkan.

Gambar untuk pengujian disimpan dalam folder *hold_out*:

```
if(index < split_percentage):
    copy_files(**{
        "file_paths" : [temp_file_path],
        "target_directory" : "hold_out/"
    })
copy_files(**{
    "file_paths" : [test_file_path],
    "target_directory" : "hold_out/"
})
```

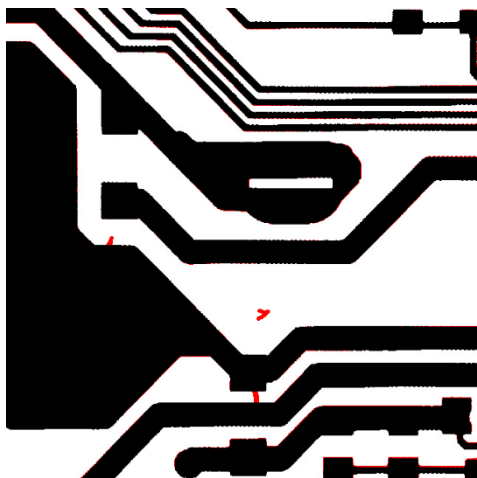
Gambar sisanya digunakan untuk deteksi defect. Untuk mendeteksi defect, hal yang dilakukan adalah membandingkan gambar PCB *test* dan *template* untuk menghasilkan *differenced image* dengan:

```

else:
    subtract_images(**{
        "image_path_1" : temp_file_path,
        "image_path_2" : test_file_path,
        "write_path" :
            "difference_img/diff_image_" + str(index)
            + ".png"
    })

```

Gambar hasil subtraction akan disimpan pada folder *differenced_img*.



Gambar 4.1 Salah satu gambar perbedaan antara *test* dan *template* (*differenced*)

Selanjutnya seluruh gambar dalam *differenced_img* akan diekstrak defectnya (merah) dengan:

```

all_differenced_images =
os.listdir('difference_img')

for index, filename in
enumerate(all_differenced_images):

os.mkdir("extracted_defects/extracts_" +
str(index))

```

```

this_write_path =
"extracted_defects/extracts_" +
str(index) + "/"

extract_contours_from_image(**{
    "image_path" : "difference_img/"
+ filename,
    "write_path" : this_write_path,
    "hsv_lower" : [0,150,50],
    "hsv_upper" : [10,255,255]
})

```

yang melakukan ekstraksi kontur untuk bagian gambar dengan piksel antara HSV[0,150,50] hingga HSV[10,255,255] sehingga akan diperoleh gambar hasil ekstraksi defect disimpan pada folder *extracted_defect*.



Gambar 4.2 Beberapa contoh hasil ekstraksi defect

Kemudian gambar hasil ekstraksi defect ini dipindahkan dalam satu directory *all_extract* untuk kemudian disimpan dalam .zip untuk tahap labelling. File lain yang diperlukan untuk labelling adalah .json yang menyimpan label atau filename gambar. JSON file dibuat dengan:

```

import json

file_names_and_sizes = {}

file_names = os.listdir('all_extracts/')

for index, file in enumerate(file_names):

    file_names_and_sizes[file_names[index]]
= os.path.getsize('all_extracts/' + file)

with open('extracted_defects.json', 'w')
as outfile:

    json.dump(file_names_and_sizes,
outfile)

```

File .zip yang berisi gambar ekstraksi defect dan file .json yang berisi filename gambar tersebut diunduh dan digunakan untuk tahap berikutnya yaitu *labelling*.

4.2 DEFECT LABELLING

Tahap labelling menggunakan file gambar hasil ekstraksi defect dan filename-nya yang diperoleh dari tahap sebelumnya. Labelling ini akan dilakukan secara lokal di PC (bukan di platform Google Colab), menggunakan aplikasi Defect Turker buatan Sean McClure. Berikut langkah menggunakan Defect Turker.

Dilakukan *clone* directory Defect Turker dengan (run di cmd):

```
git clone https://github.com/sean-mcclure/defect_turker.git
```

Pada folder *img* dalam directory *defect_turker*, disimpan seluruh gambar dalam file ZIP ekstraksi defect dan file JSON. Berikutnya akan dijalankan aplikasi Defect Turker secara lokal dengan host web (run di cmd) di direktori *defect_turker*:

```
python3 -m http.server
```

Web server sudah berjalan, berikutnya buka alamat berikut dengan browser <http://localhost:8000/>

Pada halaman tersebut akan tampak tampilan berikut:



Gambar 4.3 Interface untuk labelling dengan Defect Turker

Menggunakan aplikasi berikut, penulis melakukan labelling dengan mengklik button jenis defect gambar (dari file *extracted_defects*) yang ditampilkan oleh aplikasi. Setelah satu gambar diberi label, gambar berikutnya akan muncul. Proses labelling dilakukan untuk seluruh gambar defect hasil ekstraksi. Untuk dataset folder *group00041* diperlukan labelling pada sebanyak sekitar 1300 gambar (1376 gambar).

Pada pojok kanan aplikasi terdapat tombol download untuk mengunduh file *image_labels.csv* yang berisi filename gambar defect hasil ekstraksi dan labelnya. File CSV ini digunakan untuk melakukan training pada model *classifier* pada tahap berikutnya.

4.3 DEFECT CLASSIFICATION

Pada tahap ini akan dilakukan training model defect classification menggunakan platform Google Colab. Tahap ini menggunakan file *image_labels.csv* dari tahap sebelumnya.

Pertama-tama install dan import library yang diperlukan:

```
!pip install torch==1.7.1+cu101
torchvision==0.8.2+cu101
torchaudio==0.7.2 -f
https://download.pytorch.org/whl/torch_stable.html
!pip install fastai==1.0.61
```

```
from fastai.vision import *
```

Image_labels.csv diunggah ke workspace Colab, kemudian data file CSV diubah menjadi dataframe menggunakan library pandas:

```
Import pandas as pd
image_labels_frame =
pd.read_csv('nama_file.csv', sep=',')
```

Berikutnya akan dilakukan perhitungan terhadap jumlah count untuk setiap label menggunakan fungsi dalam library *tetryonai* *get_feature_counts*:

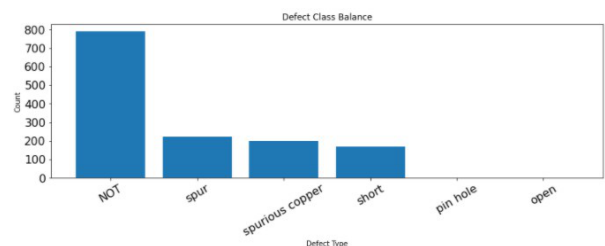
```
labelled_counts = get_feature_counts(**{
    "data_frame" : image_labels_frame,
    "feature" : "label"
})
```

Setelah diperoleh nilai count untuk masing-masing label, untuk mengamati sebaran jenis defect dapat dibuat plotnya menggunakan *Matplotlib*:

```
import matplotlib.pyplot as plt
from pylab import rcParams
rcParams['figure.figsize'] = 14, 4
rcParams['text.color'] = "black"
rcParams['axes.labelcolor'] = "black"
rcParams['xtick.color'] = "black"
rcParams['ytick.color'] = "black"

index = np.arange(len(labelled_counts.keys()))
plt.bar(index, list(labelled_counts.values()))
plt.xlabel('Defect Type', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.xticks(index, labelled_counts.keys(),
    fontsize=5, rotation=30)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.title('Defect Class Balance')
plt.show()
```

diperoleh hasil sebagai berikut:



Gambar 4.4 Histogram kategori untuk hasil labelling

Sebelum melakukan training, perlu dilakukan persiapan terhadap data dengan label supaya

sesuai dengan susunan yang dibutuhkan *training classifier*. Persiapan yang dilakukan antara lain membuat folder, memindahkan setiap jenis defect yang sejenis pada folder yang sama dan .csv baru untuk masing-masing folder, serta menghapus sisa gambar yang tidak memiliki label.

Gambar yang sudah disiapkan kemudian dibaca menggunakan:

```
np.random.seed(42)
data = ImageDataBunch.from_folder(path, train='.', valid_pct=0.2, ds_tfms=get_transforms(), size=224, num_workers=4, bs=10).normalize(imagenet_stats)
```

Kumpulan gambar dengan normalisasi disimpan dalam variabel data.

Berikutnya akan dilakukan training model menggunakan kode berikut:

```
from fastai.metrics import error_rate # 1
- accuracy
learn = create_cnn(data, models.resnet34, metrics=error_rate)
```

Training akan dilakukan pada variabel data yang berisi gambar ternormalisasi menggunakan CNN dengan arsitektur ResNet34. Berikutnya training dilakukan untuk 10 kali *fit_one_cycle* atau 10 epoch. Epoch adalah berapa kali perulangan algoritma learning akan menelusuri seluruh training dataset.

```
defaults.device = torch.device('cuda')
learn.fit_one_cycle(10)
```

Berikut hasil run dari program di atas:

epoch	train_loss	valid_loss	error_rate	time
0	1.168086	0.451977	0.152174	05:49
1	0.840938	0.449679	0.148551	06:19
2	0.705897	0.365581	0.094203	06:49
3	0.550424	0.271514	0.097826	06:46
4	0.426063	0.226180	0.061594	06:46
5	0.377865	0.254306	0.083333	06:50
6	0.313929	0.219193	0.061594	06:51
7	0.260761	0.196972	0.061594	06:48
8	0.263895	0.175666	0.054348	06:48
9	0.233745	0.191307	0.047101	06:49

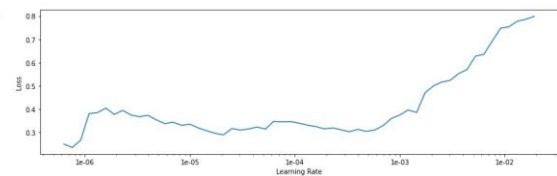
Gambar 4.5 Output training dengan CNN Resnet34 (*fit_one_cycle* 10 epoch)

Sekarang network akan di-*unfreeze*, yang berarti semua bobot dalam model sekarang dapat diperbarui. Untuk ini dapat digunakan metode *fastai unfreeze()* di objek learning.

Model yang di-*unfreeze* baiknya tidak menggunakan kecepatan pembelajaran yang sama di semua lapisan. Lapisan yang berbeda dalam deep model mendapatkan keuntungan dari *differential learning rate* di mana berbagai kecepatan digunakan tergantung di mana di jaringan itu diterapkan. Untuk menggunakan *differential learning rate*, pertama-tama kita perlu menemukan kecepatan yang sesuai.

```
learn.unfreeze()
learn.lr_find()
learn.recorder.plot()
```

Plot dari *learning rate* dan *loss* diperoleh sebagai berikut:



Gambar 4.6 Grafik *learning rate* vs *loss*

Pendekatan penentuan nilai rate dilakukan dengan mengambil nilai di bawah titik terendah supaya *loss* tidak menjadi lebih buruk. Untuk itu diambil rate dengan rentang 3×10^{-5} dan 3×10^{-4} .

Berikutnya akan dijalankan *fit_one_cycle* dengan menggunakan argumen *max_lr* untuk menentukan range dari *learning rate*. Model yang di-*unfreeze* ditrain ulang dengan epoch lebih sedikit, 4 epoch, menggunakan *differential learning rate*.

```
learn.fit_one_cycle(4, max_lr=slice(3e-5, 3e-4))
```

Berikut hasil run dari program di atas:

epoch	train_loss	valid_loss	error_rate	time
0	0.316559	0.291128	0.090580	10:12
1	0.280868	0.196575	0.050725	10:07
2	0.254090	0.133350	0.036232	10:06
3	0.179867	0.137466	0.036232	10:04

Gambar 4.5 Output training dengan *differential learning rate* (*fit_one_cycle* 4 epoch)

Dapat diamati pada baris terakhir bahwa model yang didapat memiliki loss yang lebih kecil dari hasil latih sebelumnya.

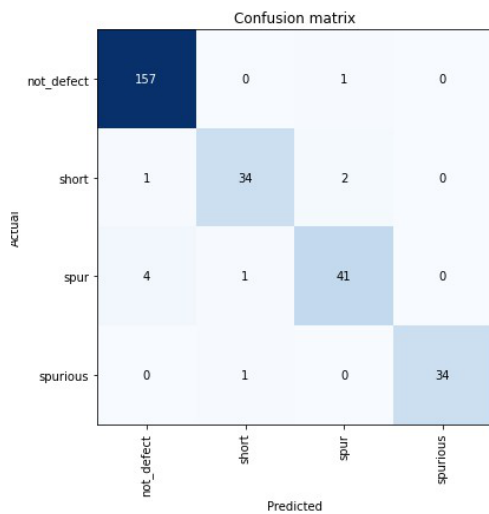
Berikutnya model hasil training akan disimpan dan diekspor dalam sebuah file `defect_classifier.pkl`

```
learn.save('defect_classifier')
learn.load('defect_classifier')
learn.export('defect_classifier.pkl')
```

Untuk mengamati error classification, dilakukan pengamatan pada *confusion matrix* terhadap dataset untuk *validation* seperti berikut:

```
interp = ClassificationInterpretation.from_m_learner(learn)
interp.plot_confusion_matrix(figsize=(6,6))
```

Berikut adalah plot *confusion matrix*:



Gambar 4.7 *confusion matrix* untuk model hasil training

Diperoleh model *classifier* dengan true prediction (diagonal identitas) sebanyak 266 dan false prediction sebanyak 10. Error validation sekitar 3,76%.

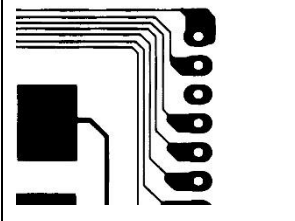
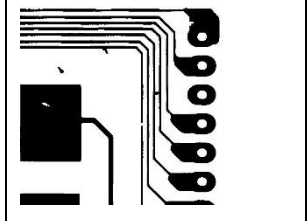
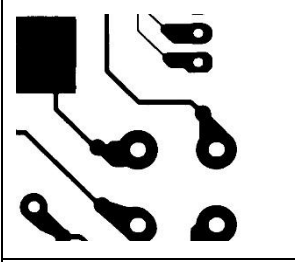
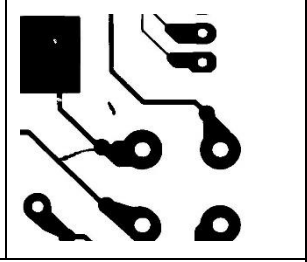
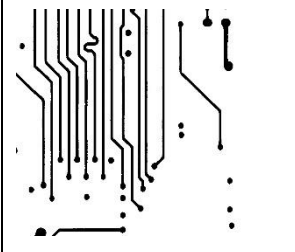
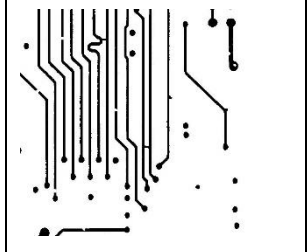
Model yang diperoleh akan digunakan untuk testing PCB defect classifier pada tahap berikutnya.

4.4 PCB DEFECT CLASSIFIER TESTING

Proses testing classifier akan menggunakan model classifier `defect_classifier.pkl` dan file gambar dari tahap sebelumnya.

File gambar akan digunakan untuk dilakukan serangkaian deteksi. Untuk testing ini digunakan gambar 12100039, 12100040 dan 20085042.

Tabel 1 Gambar Template dan Test Untuk Pengujian

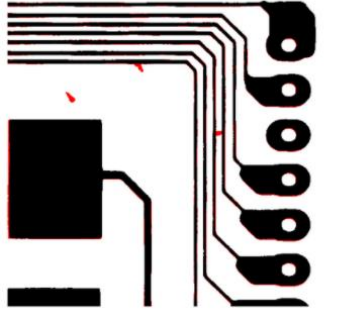
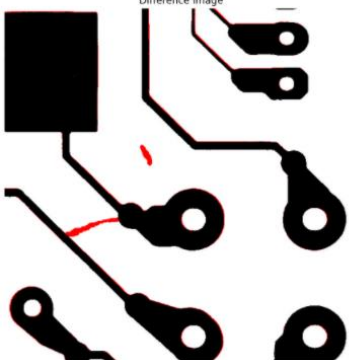
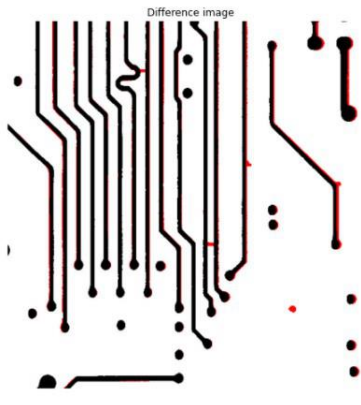
Template	Test
	
Gambar 12100039	
	
Gambar 12100040	
	
Gambar 20085042	

Gambar melewati proses subtraksi (test dikurangi template) dan disimpan pada folder `difference_image_test`. Proses ini dapat dilakukan dengan menggunakan fungsi dari library `tetryonai`:

```
subtract_images(**{
    "image_path_1": image_path_1,
    "image_path_2": image_path_2,
    "write_path":
        "difference_image_test/diff_test" + id +
        ".png"
})
```

Hasil deteksi dengan subtraksi diperoleh sebagai berikut:

Tabel 2 Hasil Proses Subtraksi Gambar Template dan Gambar Test

Differenced image

Gambar 12100039

Gambar 12100040

Gambar 20085042

Hasil subtraksi melewati proses feature extraction untuk diambil kontur defect dan disimpan pada folder *defect_detected_test*. Proses extraction juga dapat dilakukan menggunakan fungsi dari library tetryonai:

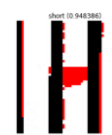


```
extract_contours_from_image(**{
    "image_path": path_to_diff,
    "write_path": "defect_detected_test/"
+ id + "/",
    "hsv_lower": [0, 150, 50],
    "hsv_upper": [10, 255, 255]
})
```

Hasil extracted defect melewati proses prediksi yang mengandung model *classifier* dari percobaan sebelumnya. Proses ini dilakukan menggunakan kode berikut:

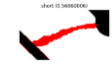

```
#Predicting defect
img = open_image(filename)
prediction = learn.predict(img)
```

Hasil prediksi klasifikasi defect pada tiga rangkaian 12100039, 1210040, dan 20085042 dapat diamati sebagai berikut.



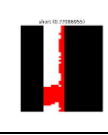





Tabel 3 Hasil Deteksi dan Prediksi Cacat pada Rangkaian 12100039

Defect	Actual	Prediction	Confidence
	Short	Short	94,84%
	Spurious copper	Spurious copper	99,99%
	Spur	Spur	96,13%

Tabel 4 Hasil Deteksi dan Prediksi Cacat pada Rangkaian 12100040

Defect	Actual	Prediction	Confidence
	Short	Short	56,86%
	Spurious copper	Spurious copper	99,95%

Tabel 5 Hasil Deteksi dan Prediksi Cacat pada Rangkaian 20085042

Defect	Actual	Prediction	Confidence
	Spur	Spur	82,84%
	Spur	Short (miss)	53,39%
	Short	Short	77,08%
	Short	Spur (miss)	60,07%
	Not defect	Short (miss)	58,16%
	Short	Short	99,83%
	Spurious copper	Spurious copper	99,98%
	Short	Short	90,87%



Prediksi defect cukup representatif terhadap klasifikasi defect sebenarnya. Hasil testing *classifier* untuk PCB defect dapat dikatakan cukup aplikatif meskipun masih terdapat beberapa salah prediksi. Kesalahan tersebut disebabkan jumlah dataset yang digunakan untuk proses pembelajaran masih cukup sedikit, yakni pada orde ratusan. Penambahan jumlah dataset dapat meningkatkan performa model dalam melakukan klasifikasi.

5. KESIMPULAN

Dari hasil percobaan yang telah dilakukan dapat ditarik kesimpulan sebagai berikut.

1. Dataset diambil dari folder group00041 dari GitHub tangsanli5201.

Metode deteksi defect dilakukan menggunakan metode *subtracting*, yaitu mencari selisih antara gambar PCB test dengan PCB template. Hasil selisih diberi warna merah. Gambar hasil deteksi disimpan pada folder *difference_img*.

Metode ekstraksi dilakukan dengan menduplikat bagian gambar dengan warna piksel antara HSV[0,150,50]  hingga HSV[10,255,255]  sehingga akan diperoleh gambar hasil ekstraksi defect disimpan pada folder *extracted_defect*.

2. Labelling dilakukan pada gambar hasil ekstraksi defect dari gambar PCB dalam folder group00041, sebanyak 1376 gambar dengan label *not defect*, *short*, *spur*, dan *spurious copper*.
3. Model training menggunakan CNN dengan arsitektur ResNet34, diperoleh hasil model *classifier* dengan hasil validasi *true prediction* sebanyak 266 dan *false prediction* sebanyak 10. Error validation sebesar 3,76%.
4. Model hasil training diuji menggunakan rangkaian 12100039, 12100040, dan 20085042 dalam master dataset diperoleh 12 defect dengan hasil prediksi 9 tepat dan 3 meleset.

Prediksi defect cukup representatif terhadap klasifikasi defect sebenarnya, dan masih terdapat ruang untuk pengembangan di masa depan.

Berikut saran dan usulan mengenai masalah teknis yang belum dilakukan, antara lain:

1. Melakukan proses pembelajaran dengan jumlah dataset yang lebih banyak.
2. Membuat algoritma *defect detection* yang lebih andal yang dapat mendeteksi jenis cacat lain dan mengabaikan bagian rangkaian yang tidak cacat.
3. Membuat aplikasi *classifier* yang dapat memberikan anotasi defect pada rangkaian testing untuk memudahkan pengamatan.
4. Membuat aplikasi untuk prediksi *cost* berdasarkan deteksi dan klasifikasi cacat yang muncul.

DAFTAR PUSTAKA

- [1] McClure, Sean. *Building End-to-End Defect Classifier Application for Printed Circuit Boards*. towards data science, 2020. <https://towardsdatascience.com/building-an-end-to-end-deep-learning-defect-classifier-application-for-printed-circuit-board-pcb-6361b3a76232>
- [2] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in CVPR, 2016.
- [3] Rafael C. Gonzalez dan Richard E. Woods, *Digital Image Processing*, Pearson, New York, 2018.