LAMPIRAN

Surat Keterangan Bebas Plagiat

UNIVERSITAS DARMA PERSADA
UPT PERPUSTAKAAN
Gedung Rektorat Lantai 3,
Jl.Taman Malaka Selatan, Pondok Kelapa – Jakarta Timur 13450

SURAT KETERANGAN
HASIL PENGECEKAN TURNITIN

UPT Perpustakaan Universitas Darma Persada menerangkan telah selesai melakukan pemeriksaan duplikasi/*similarity* menggunakan perangkat lunak Turnitin terhadap hasil karya sebagai berikut:

Judul	: PENGEMBANGAN MODEL DETEKSI PENYAKIT GIGI BERBASIS VISI KOMPUTER DENGAN ARSITEKTUR MOBILENETV2 (CNN) PADA POLI GIGI KLINIK MITRA SEHAT KARAWANG
Penulis	: Dhio Andreas Gemilang
NIM	: **2021230024**
Tgl pemeriksaan	: 25 Juli 2025

Dengan hasil Tingkat Kesamaan (*similarity index*) **22%**

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 25 Juli 2025

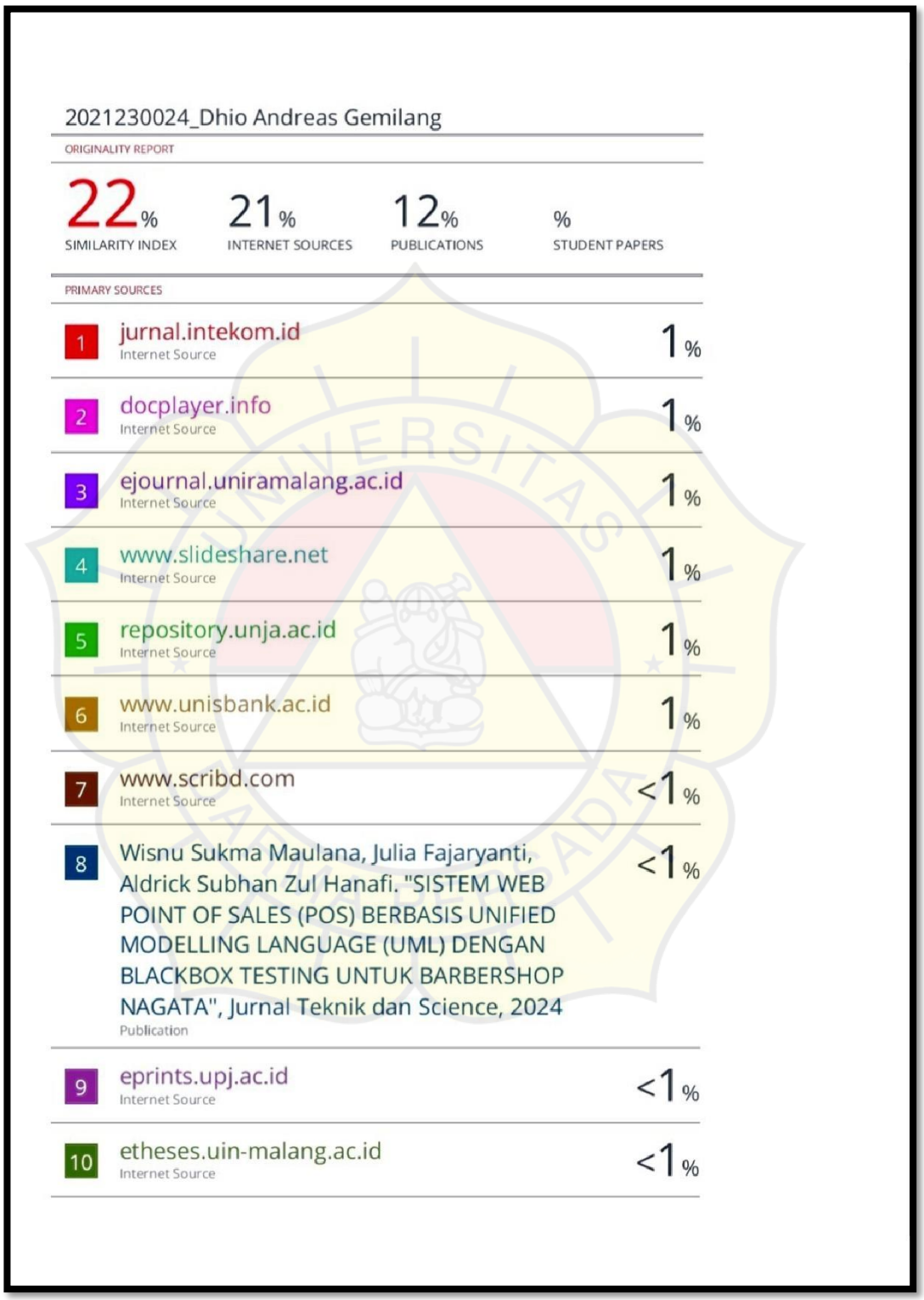
Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

Batas maksimal similarity 25% untuk Fakultas Teknik, Kelautan dan Pasca Sarjana

Hasil Turnitin



2021230024_Dhio Andreas Gemilang

ORIGINALITY REPORT

22%	21%	12%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	jurnal.intekom.id Internet Source	1%
2	docplayer.info Internet Source	1%
3	ejournal.uniramalang.ac.id Internet Source	1%
4	www.slideshare.net Internet Source	1%
5	repository.unja.ac.id Internet Source	1%
6	www.unisbank.ac.id Internet Source	1%
7	www.scribd.com Internet Source	<1%
8	Wisnu Sukma Maulana, Julia Fajaryanti, Aldrick Subhan Zul Hanafi. "SISTEM WEB POINT OF SALES (POS) BERBASIS UNIFIED MODELLING LANGUAGE (UML) DENGAN BLACKBOX TESTING UNTUK BARBERSHOP NAGATA", Jurnal Teknik dan Science, 2024 Publication	<1%
9	eprints.upj.ac.id Internet Source	<1%
10	etheses.uin-malang.ac.id Internet Source	<1%

Source Code App.py

```
1. import os
2. from flask import Flask, request, jsonify, render_template,
   redirect, url_for, session, flash
3. from flask_sqlalchemy import SQLAlchemy
4. from werkzeug.security import generate_password_hash,
   check_password_hash
5. from functools import wraps
6. import tensorflow as tf
7. from tensorflow.keras.preprocessing import image
8. import numpy as np
9. from PIL import Image
10.      from flask_cors import CORS
11.      import random
12.      import string
13.      from itsdangerous import URLSafeTimedSerializer
14.      from email.mime.text import MIMEText
15.      import smtplib
16.      from flask import request, redirect, url_for,
   render_template, flash
17.      from werkzeug.utils import secure_filename
18.      import os
19.      # --- Inisialisasi Flask & CORS
20.      app = Flask(__name__)
21.      CORS(app)
22.      # --- Konfigurasi APP
23.      app.config['SECRET_KEY'] = 'myverysecretkey12345'
24.      app.config['SQLALCHEMY_DATABASE_URI'] =
   'mysql+pymysql://root:@localhost:3306/deteksi_penyakit_gigi'
25.      app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
26.      db = SQLAlchemy(app)
27.      # --- Serializer untuk Token
28.      s = URLSafeTimedSerializer(app.config['SECRET_KEY'])
29.      # --- Konfigurasi Email
30.      app.config['MAIL_SERVER'] = 'smtp.gmail.com'
31.      app.config['MAIL_PORT'] = 587
32.      app.config['MAIL_USERNAME'] = 'your_email@gmail.com'
33.      app.config['MAIL_PASSWORD'] = 'your_app_password'
34.      app.config['MAIL_USE_TLS'] = True
35.      # --- Tambahan CORS header
36.      @app.after_request
37.      def add_cors_headers(response):
38.      response.headers['Access-Control-Allow-Origin'] = '*'
39.      response.headers['Access-Control-Allow-Headers'] =
   'Content-Type,Authorization'
40.      response.headers['Access-Control-Allow-Methods'] =
   'GET,PUT,POST,DELETE,OPTIONS'
41.      return response
42.      # --- Model Database
43.      class Pengguna(db.Model):
44.      __tablename__ = 'pengguna'
45.      id_pengguna = db.Column(db.Integer, primary_key=True)
46.      nama = db.Column(db.String(100), nullable=False)
47.      email = db.Column(db.String(100), unique=True,
   nullable=False)
```

```
48.       password = db.Column(db.String(255), nullable=False)
49.       role = db.Column(db.Enum('admin', 'user'),
   default='user')
50.       def set_password(self, raw_password):
51.       self.password = generate_password_hash(raw_password)
52.       def check_password(self, raw_password):
53.       return check_password_hash(self.password,
   raw_password)
54.       class HasilKlasifikasi(db.Model):
55.       __tablename__ = 'hasil_klasifikasi'
56.       id = db.Column(db.Integer, primary_key=True)
57.       nama = db.Column(db.String(100))
58.       filename = db.Column(db.String(255))
59.       prediksi = db.Column(db.String(50))
60.       confidence = db.Column(db.Float)
61.       waktu = db.Column(db.DateTime,
   server_default=db.func.now())
62.       class Dataset(db.Model):
63.       __tablename__ = 'dataset'
64.       id = db.Column(db.Integer, primary_key=True)
65.       label = db.Column(db.String(100), nullable=False)
66.       filename = db.Column(db.String(255), nullable=False)
67.       created_at = db.Column(db.DateTime,
   server_default=db.func.now())
68.       # --- Dekorator Login
69.       def login_required(role=None):
70.       def decorator(f):
71.       @wraps(f)
72.       def decorated_function(*args, **kwargs):
73.       if 'username' not in session:
74.       flash('Anda perlu login untuk mengakses halaman ini.',
   'warning')
75.       return redirect(url_for('login'))
76.       if role and session.get('role') != role:
77.       flash(f'Akses ditolak untuk role: {role}', 'danger')
78.       return redirect(url_for('dashboard_admin' if
   session.get('role') == 'admin' else 'dashboard_user'))
79.       return f(*args, **kwargs)
80.       return decorated_function
81.       return decorator
82.       # --- Load Model AI
83.       MODEL_PATH = 'best_model.h5'
84.       if not os.path.exists(MODEL_PATH):
85.       print(f"Error: File model '{MODEL_PATH}' tidak
   ditemukan.")
86.       exit(1)
87.       try:
88.       model = tf.keras.models.load_model(MODEL_PATH)
89.       print("Model berhasil dimuat.")
90.       except Exception as e:
91.       print(f"ERROR saat memuat model: {e}")
92.       exit(1)
93.       CLASS_NAMES = ["Gigi Berkarang", "Gigi Berlubang",
   "Bukan Gigi"]
94.       IMG_HEIGHT = 224
95.       IMG_WIDTH = 224
```

```
96.      # --- Kirim Email Reset
97.      def send_reset_email(to_email, token):
98.      reset_url = url_for('reset_password', token=token,
   _external=True)
99.      subject = "Reset Password Akun Anda"
100.     body = f"Klik link berikut untuk reset
   password:\n\n{reset_url}\n\nJika Anda tidak meminta, abaikan
   saja."
101.     message = MIMEText(body)
102.     message['Subject'] = subject
103.     message['From'] = app.config['MAIL_USERNAME']
104.     message['To'] = to_email
105.     try:
106.     with smtplib.SMTP(app.config['MAIL_SERVER'],
   app.config['MAIL_PORT']) as server:
107.     server.starttls()
108.     server.login(app.config['MAIL_USERNAME'],
   app.config['MAIL_PASSWORD'])
109.     server.send_message(message)
110.     print(f"Email reset terkirim ke {to_email}")
111.     except Exception as e:
112.     print(f"[ERROR Kirim Email]: {e}")
113.     # --- ROUTES
114.     @app.route('/')
115.     def home():
116.     return redirect(url_for('login'))
117.     @app.route('/login', methods=['GET', 'POST'])
118.     def login():
119.     if 'username' in session:
120.     return redirect(url_for('dashboard_admin' if
   session.get('role') == 'admin' else 'dashboard_user'))
121.     if request.method == 'POST':
122.     email = request.form['username']
123.     password = request.form['password']
124.     user = Pengguna.query.filter_by(email=email).first()
125.     if user and user.check_password(password):
126.     session['username'] = user.nama
127.     session['role'] = user.role
128.     flash('Login berhasil.', 'success')
129.     return redirect(url_for('dashboard_admin' if user.role
   == 'admin' else 'dashboard_user'))
130.     else:
131.     flash('Email atau password salah.', 'danger')
132.     return render_template('login.html')
133.     @app.route('/register', methods=['GET', 'POST'])
134.     def register():
135.     if request.method == 'POST':
136.     try:
137.     nama = request.form['username']
138.     email = request.form['email']
139.     password = request.form['password']
140.     confirm_password = request.form['confirm_password']
```

```
141.     if not nama or not email or not password or not
   confirm_password:
142.     flash('Semua kolom wajib diisi.', 'danger')
143.     return render_template('register.html')
144.     if password != confirm_password:
145.     flash('Konfirmasi password tidak cocok.', 'danger')
146.     return render_template('register.html')
147.     existing =
   Pengguna.query.filter_by(email=email).first()
148.     if existing:
149.     flash('Email sudah terdaftar.', 'warning')
150.     return render_template('register.html')
151.     user = Pengguna(nama=nama, email=email, role='user')
152.     user.set_password(password)
153.     db.session.add(user)
154.     db.session.commit()
155.     flash('Registrasi berhasil. Silakan login.',
   'success')
156.     return redirect(url_for('login'))
157.     except Exception as e:
158.     print(f"ERROR REGISTER: {e}")
159.     flash('Terjadi kesalahan saat registrasi.', 'danger')
160.     return render_template('register.html')
161.     return render_template('register.html')
162.     @app.route('/profil')
163.     @login_required()
164.     def profil():
165.     user =
   Pengguna.query.filter_by(nama=session.get('username')).first
   ()
166.     return render_template('profil.html', user=user)
167.     @app.route('/profil/edit', methods=['GET', 'POST'])
168.     @login_required()
169.     def edit_profil():
170.     nama_lama = session.get('username')
171.     user =
   Pengguna.query.filter_by(nama=nama_lama).first()
172.     if not user:
173.     flash('User tidak ditemukan.', 'danger')
174.     return redirect(url_for('profil'))
175.     if request.method == 'POST':
176.     new_email = request.form['email']
177.     new_password = request.form['password']
178.     if not new_email:
179.     flash('Email tidak boleh kosong.', 'danger')
180.     return render_template('edit_profil.html', user=user)
181.     user.email = new_email
182.     if new_password:
183.     user.set_password(new_password)
184.     db.session.commit()
185.     flash('Profil berhasil diperbarui.', 'success')
186.     return redirect(url_for('profil'))
187.     return render_template('edit_profil.html', user=user)
188.     @app.route('/logout')
189.     @login_required()
190.     def logout():
```

```
191.     session.clear()
192.     flash('Anda telah logout.', 'info')
193.     return redirect(url_for('login'))
194.     @app.route('/lupa_akun', methods=['GET', 'POST'])
195.     def lupa_akun():
196.     if request.method == 'POST':
197.     email = request.form['email']
198.     user = Pengguna.query.filter_by(email=email).first()
199.     if user:
200.     # Buat password baru secara acak
201.     new_password =
   ''.join(random.choices(string.ascii_letters + string.digits,
   k=8))
202.     user.set_password(new_password)
203.     db.session.commit()
204.     flash(f'Password baru Anda: {new_password}',
   'success')
205.     else:
206.     flash('Email tidak ditemukan.', 'danger')
207.     return render_template('lupa_akun.html')
208.     @app.route('/reset_password/<token>', methods=['GET',
   'POST'])
209.     def reset_password(token):
210.     return render_template('reset_password.html')
211.     @app.route('/predict', methods=['POST'])
212.     @login_required()
213.     def predict():
214.     if 'file' not in request.files:
215.     return jsonify({"error": "Tidak ada file yang
   diupload."}), 400
216.     file = request.files['file']
217.     if file.filename == '':
218.     return jsonify({"error": "Tidak ada file yang
   dipilih."}), 400
219.     try:
220.     # --- Siapkan gambar
221.     img = Image.open(file.stream).convert('RGB')
222.     img = img.resize((IMG_HEIGHT, IMG_WIDTH))
223.     img_array = image.img_to_array(img)
224.     img_array = np.expand_dims(img_array, axis=0) / 255.0
225.     # --- Prediksi menggunakan model
226.     predictions = model.predict(img_array)
227.     scores = tf.nn.softmax(predictions[0])
228.     predicted_class_index = np.argmax(scores)
229.     predicted_class_name =
   CLASS_NAMES[predicted_class_index]
230.     confidence = float(np.max(scores))
231.     print(f"[DEBUG Predict] Class: {predicted_class_name},
   Confidence: {confidence}")
232.     # --- Threshold minimum
233.     MIN_CONFIDENCE = 0.5
234.     # --- Tolak jika bukan gigi atau confidence rendah
235.     if predicted_class_name == "Bukan Gigi" or confidence
   < MIN_CONFIDENCE:
236.     return jsonify({
237.     "prediction": "Tidak Dikenali",
```

```
238.     "confidence": confidence,
239.     "message": "Gambar yang Anda upload bukan gambar gigi
   yang dapat dikenali. Silakan coba gambar lain."
240.     })
241.     # --- Simpan hasil prediksi valid (HANYA Gigi
   Berlubang / Berkarang)
242.     hasil = HasilKlasifikasi(
243.     nama=session.get('username'),
244.     filename=file.filename,
245.     prediksi=predicted_class_name,
246.     confidence=confidence
247.     )
248.     db.session.add(hasil)
249.     db.session.commit()
250.     # --- Kirim respon JSON ke frontend
251.     return jsonify({
252.     "prediction": predicted_class_name,
253.     "confidence": confidence,
254.     "all_scores": {label: float(scores[i]) for i, label in
   enumerate(CLASS_NAMES)}
255.     })
256.     except Exception as e:
257.     print(f"[ERROR Predict]: {e}")
258.     return jsonify({"error": str(e)}), 500
259.     @app.route('/dashboard_user')
260.     @login_required(role='user')
261.     def dashboard_user():
262.     return render_template('dashboard_user.html')
263.     @app.route('/dashboard_admin')
264.     @login_required(role='admin')
265.     def dashboard_admin():
266.     from sqlalchemy import func
267.     data = (
268.     db.session.query(HasilKlasifikasi.prediksi,
   func.count(HasilKlasifikasi.id))
269.     .group_by(HasilKlasifikasi.prediksi)
270.     .all()
271.     )
272.     labels = [row[0] for row in data] or ['Kalkulus',
   'Karies']
273.     counts = [row[1] for row in data] or [0, 0]
274.     return render_template('dashboard_admin.html',
   chart_labels=labels, chart_data=counts)
275.     @app.route('/deteksi')
276.     @login_required(role='user')
277.     def deteksi():
278.     return render_template('deteksi.html')
279.     @app.route('/riwayat')
280.     @login_required(role='user')
281.     def riwayat():
282.     nama_user = session.get('username')
283.     data_riwayat =
   HasilKlasifikasi.query.filter_by(nama=nama_user).order_by(Ha
   silKlasifikasi.waktu.desc()).all()
284.     return render_template('riwayat.html',
   data=data_riwayat)
```

```
285.     @app.route('/riwayat_admin')
286.     @login_required(role='admin')
287.     def riwayat_admin():
288.     data =
   HasilKlasifikasi.query.order_by(HasilKlasifikasi.waktu.desc(
   )).all()
289.     return render_template('riwayat_admin.html',
   data=data)
290.     @app.route('/hapus_riwayat/<int:id>')
291.     @login_required(role='admin')
292.     def hapus_riwayat(id):
293.     data = HasilKlasifikasi.query.get_or_404(id)
294.     db.session.delete(data)
295.     db.session.commit()
296.     flash('Data berhasil dihapus.', 'success')
297.     return redirect(url_for('riwayat_admin'))
298.     @app.route('/kelola_pengguna')
299.     @login_required(role='admin')
300.     def kelola_pengguna():
301.     users = Pengguna.query.order_by(Pengguna.nama).all()
302.     return render_template('kelola_pengguna.html',
   users=users)
303.     @app.route('/admin_dataset', methods=['GET', 'POST'],
   endpoint='dataset_admin')
304.     @login_required()
305.     def admin_dataset():
306.     if request.method == 'POST':
307.     file = request.files.get('file')
308.     label = request.form.get('label')
309.     if file and label:
310.     filename = secure_filename(file.filename)
311.     save_path = os.path.join('static/uploads', filename)
312.     file.save(save_path)
313.     data = Dataset(label=label, filename=filename)
314.     db.session.add(data)
315.     db.session.commit()
316.     flash('Data berhasil diupload.')
317.     dataset =
   Dataset.query.order_by(Dataset.id.desc()).all()
318.     return render_template('admin_dataset.html',
   dataset=dataset)
319.     @app.route('/admin/upload_dataset', methods=['POST'])
320.     @login_required()
321.     def upload_dataset():
322.     if 'image' not in request.files or
   request.form.get('label') is None:
323.     flash('Lengkapi data upload!', 'danger')
324.     return redirect(url_for('dataset_admin'))
325.     file = request.files['image']
326.     label = request.form.get('label')
327.     if file.filename == '':
328.     flash('File tidak dipilih!', 'danger')
329.     return redirect(url_for('dataset_admin'))
```

```
    # Simpan file ke folder static/uploads/
    filename = secure_filename(file.filename)
    save_path = os.path.join('static/uploads', filename)
    file.save(save_path)
    # Simpan ke database
    new_data = Dataset(label=label, filename=filename)
    db.session.add(new_data)
    db.session.commit()
    flash('Gambar berhasil diupload!', 'success')
    return redirect(url_for('dataset_admin'))
    # --- MAIN
    if __name__ == '__main__':
    print("DEBUG: Memulai server Flask...")
    app.run(host='0.0.0.0', port=5000, debug=True)
```