P-ISSN: 1979-5661
E-ISSN: 2622-321X

Klasifikasi Sampah Berbasis *Convolutional Neural Networks* (CNN) untuk Peningkatan Efisiensi Pengelolaan Sampah

I Gusti Agung Gede Arya Kadyanan[a1], I Nengah Artawan [a2], Putu Herdy Juniawan[a3]

[a]Program Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Udayana, Bali
Jln. Raya Kampus UNUD, Bukit Jimbaran, Kuta Selatan, Badung, 08261, Bali, Indonesia
[1]gungde@unud.ac.id
[2]nengahartawan@gmail.com
[3]herdy478@gmail.com

Abstract

Plastik adalah masalah lingkungan mendesak, dengan lebih dari 300 juta ton diproduksi setiap tahun, dan sekitar 8 juta ton masuk ke lautan. Botol plastik adalah jenis limbah yang umum, tetapi hanya sebagian kecil yang didaur ulang, sementara sisanya mencemari lingkungan. Untuk mengatasi ini, diperlukan pendekatan baru seperti kecerdasan buatan (AI) yang dapat mengenali dan mengklasifikasikan botol plastik di antara limbah lainnya. Dalam penelitian ini, kami mengembangkan model Convolutional Neural Network (CNN) yang berhasil mengidentifikasi botol plastik dengan akurasi 88%. Ini menunjukkan potensi besar dalam mendukung program daur ulang. Dengan integrasi lebih lanjut ke dalam sistem pengelolaan limbah, model ini dapat meningkatkan efisiensi pemilahan sampah dan mengurangi plastik yang berakhir di lautan.

Keywords: *Image Processing, Waste Management, Convolutional Neural Network, Image Classification, Computer Vision, Environmental sustainability*

1. Introduction

Sampah plastik kini menjadi salah satu masalah lingkungan yang kritis di dunia. Sampah ini mengancam lingkungan dan banyak habitat laut. Ketika sampah plastik masuk ke lautan, sampah ini menyebabkan kerusakan pada ekologi, estetika, dan ekonomi [2]. Studi terbaru melaporkan bahwa lebih dari 300 juta metrik ton plastik dihasilkan setiap tahun [3], di mana 8 juta metrik ton sampah plastik telah dibuang ke laut [4]. Botol plastik banyak digunakan dalam berbagai aspek kehidupan masyarakat modern. Hampir semua air minum dalam kemasan adalah botol plastik [5]. Botol plastik bekas dihasilkan setiap hari. Sekitar 600 miliar botol plastik dibuang setiap tahun di dunia, dan hanya sekitar 47% yang dikumpulkan [2]. Botol plastik yang tidak dikumpulkan masuk ke tanah, perairan pedalaman, dan jika berakhir di lautan, sampah tersebut akhirnya dapat membentuk pulau-pulau sampah seperti di Pasifik yang lebarnya masing-masing dapat mencapai beberapa kilometer. Oleh karena itu, pemantauan dan pengelolaan botol plastik sangat penting.

Kecerdasan Buatan (AI) adalah bidang yang berkembang pesat, baik dalam studi maupun aplikasinya. Saat ini penerapan AI, atau setidaknya machine learning, sudah ada di mana-mana, baik di bidang medis, ekonomi, akademis, produktivitas, sosial, atau bahkan lingkungan. AI paling jago mengenali pola pada benda-benda yang ditemuinya dan melakukannya lebih cepat dan lebih efisien daripada manusia. Itulah sebabnya kami akan menggunakan algoritma deep learning untuk tugas mengenali botol plastik di antara sampah lainnya.

Berdasarkan masalah ini, diperlukan pendekatan baru untuk mencapai akurasi tinggi dalam menyelesaikannya. Salah satu implementasinya adalah di Tempat Pembuangan Akhir (TPA), di mana teknologi AI akan diintegrasikan ke dalam mesin yang berfungsi seperti sinar-X. Mesin ini akan sepenuhnya otomatis, mampu memilah sampah menggunakan pendekatan deep learning. Deep learning adalah metode machine learning yang meniru cara kerja sistem saraf otak manusia. Dalam hal ini, digunakan Convolutional Neural Networks (CNN), yang terdiri dari beberapa lapisan dan sangat efektif dalam pemrosesan gambar dan deteksi objek. Proses deep learning melibatkan dua sesi, yaitu sesi pelatihan dan sesi pengujian. Selama sesi pelatihan, model mempelajari karakteristik setiap data sehingga dapat membedakan satu label dari label lainnya. Dalam sesi pengujian, data yang diuji

dianalisis berdasarkan hasil dari sesi pelatihan, yang membantu dalam deteksi dan klasifikasi yang lebih akurat [2].

2. Metode Penelitian

Penelitian ini dilakukan untuk mengembangkan model Convolutional Neural Network (CNN) yang mampu melakukan klasifikasi sampah secara otomatis. Proses penelitian melibatkan beberapa langkah utama yang dijelaskan sebagai berikut:

2.1. Studi Pustaka

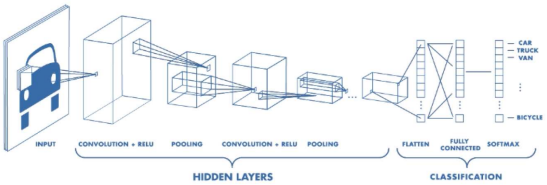
Tahap awal penelitian ini dimulai dengan melakukan studi pustaka. Studi pustaka dilakukan untuk mengkaji berbagai literatur terkait teknologi deep learning, khususnya CNN, dan penerapannya dalam klasifikasi gambar. Penelitian ini mengacu pada berbagai sumber literatur ilmiah yang membahas prinsip-prinsip dasar CNN, serta studi kasus dan aplikasi nyata dari teknologi ini dalam bidang klasifikasi objek, termasuk sampah.

2.2. Deep Learning

Deep learning adalah metode pembelajaran yang memungkinkan model komputer dengan berbagai tingkat pemrosesan untuk mempelajari representasi data melalui berbagai tingkat abstraksi. Pada dasarnya, model deep learning dibangun di atas jaringan saraf tiruan. Penelitian di bidang ini dimulai pada tahun 1980-an tetapi baru-baru ini mengalami kemajuan pesat dengan munculnya komputer yang lebih cepat. Deep learning juga merupakan cabang baru pembelajaran mesin yang telah mendapatkan popularitas dalam beberapa tahun terakhir dengan pengembangan teknologi akselerasi GPU [1]. Deep learning sangat cocok untuk aplikasi visi komputer, seperti klasifikasi objek dalam gambar. Salah satu metode pembelajaran mesin yang dapat diterapkan untuk mengklasifikasikan gambar objek adalah Convolutional Neural Network (CNN).

2.3. Convolutional Neural Network

Convolutional Neural Network (CNN) adalah salah satu jenis Jaringan Saraf Tiruan (ANN) yang dirancang khusus untuk pemrosesan data visual seperti gambar, video, dan suara [3].CNN terdiri dari berbagai lapisan, di mana setiap lapisan memiliki peran yang unik. Lapisan pertama dalam CNN adalah lapisan konvolusi, yang bertugas mengekstrak fitur-fitur



Gambar 1. Lapisan Konvolusi

Algoritma Convolutional Neural Network (CNN) dimulai dengan tahap preprocessing yang melibatkan modifikasi data gambar, seperti penyesuaian ukuran dan pengurangan noise. Selanjutnya, proses ini meliputi deteksi region of interest (ROI), penekanan latar belakang, ekstraksi fitur gambar, dan pengenalan objek, yang memungkinkan pencocokan fitur dengan objek dalam gambar. Konvolusi gambar adalah langkah akhir dalam pengambilan keputusan. CNN mencakup berbagai jenis lapisan, termasuk lapisan komposit dan lapisan yang terhubung penuh. Lapisan transformasi dan lapisan agregasi membedakan CNN dari Artificial Neural Networks (ANN) yang tidak memiliki kedua lapisan ini. Kedua lapisan ini bertugas melakukan filter pada gambar untuk menemukan fitur. Meskipun fitur dan perilaku lapisan CNN yang terhubung penuh serupa dengan perhitungan ANN, tugasnya adalah menentukan hasil fitur menggunakan label yang tersedia. Penelitian dan pengembangan terus dilakukan pada algoritma CNN untuk meningkatkan akurasi, menghemat sumber daya, dan mengurangi kesalahan. Ini termasuk modifikasi pada lapisan CNN untuk mencapai hasil yang optimal, seperti dalam model-model AlexNet, VGG, ResNet, Inception, dan MobileNet [4]. Setiap model CNN memiliki jumlah lapisan, kompleksitas, dan struktur yang berbeda-beda. Dalam penelitian ini, kami menganalisis model algoritma CNN yang sesuai untuk memastikan klasifikasi sampah yang optimal. Dengan demikian, algoritma ini dapat digunakan secara langsung untuk klasifikasi sampah otomatis oleh sistem. Tujuan dari penelitian ini adalah untuk memperpendek waktu proses daur ulang sampah serta mempermudah pengumpulan dan pemilahan sampah.

P-ISSN: 1979-5661
E-ISSN: 2622-321X

2.4. Dataset

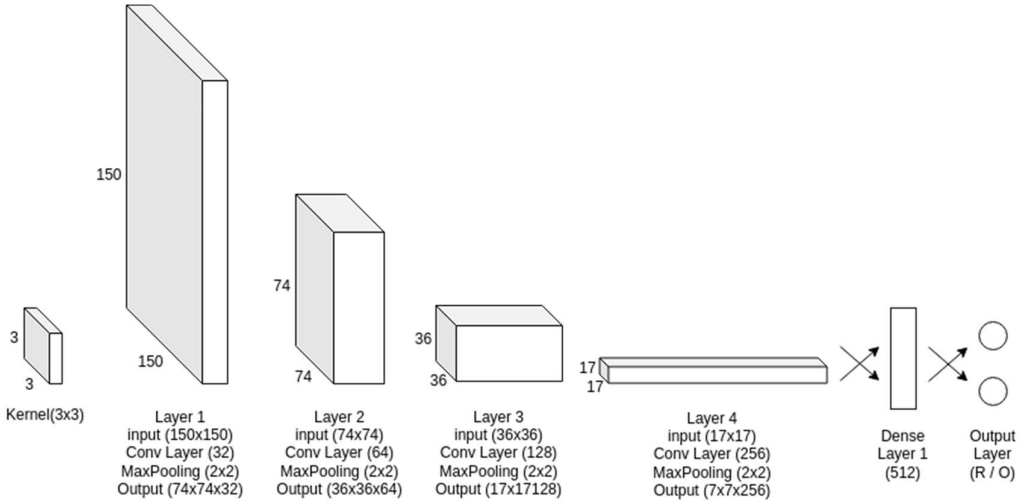
Data yang digunakan dalam penelitian ini merupakan data sekunder yang didapatkan dari dataset Water Classification Data oleh Sashaank Sekar dari Kaggle. Dataset tersebut terdiri dari lebih dari sekitar 25 ribu gambar sampah berwarna berukuran berbeda-beda yang nanti akan di preproses. Dataset dibagi hampir 50/50 untuk sampah organik dan sampah daur ulang.

Untuk Preprocessing, dataset yang aslinya dibagi menjadi training (22K data) dan testing (2,4K data), akan disatukan menjadi data organik (13.8K) dan data daur ulang(11K). Setelah itu data akan kembali dipisah dengan fungsi train_test_split dari library sklearn/scikit_learn.

2.5. Structure

Model CNN yang dibangung memiliki Total parameter 6,812,482, Trainable parameter 6,812,482 dan 0 Non-trainable parameter.

Berikut merupakan desain dari model CNN yang akan digunakan.



Gambar 2. Struktur CNN

3. Hasil dan Pembahasan

3.1. Penginstalan Library

```
[ ] import os
    import cv2
    import random
    import numpy as np
    import matplotlib.pyplot as plt
    from typing import *
    from tensorflow import keras
    import warnings

    from sklearn.metrics import accuracy_score
    from sklearn.metrics import classification_report
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import train_test_split
```

Gambar 3. *Code Install Library*

Langkah awal yang harus dilakukan dalam sebuah proyek pengembangan aplikasi berbasis Python adalah melakukan instalasi library yang diperlukan. Library adalah sekumpulan modul yang berisi kode-kode fungsi yang dapat digunakan kembali untuk berbagai tugas tertentu. Dalam konteks ini, beberapa library penting yang perlu diinstal mencakup `os` untuk berinteraksi dengan sistem operasi, `cv2` dari OpenCV untuk pengolahan citra, serta `numpy` untuk komputasi ilmiah yang melibatkan array dan operasi matematika. Selain itu, kita juga menggunakan `matplotlib.pyplot` untuk visualisasi data, `tensorflow.keras` untuk membangun dan melatih model deep learning, dan `scikit-learn` untuk berbagai fungsi yang mendukung evaluasi dan pemilihan model. Sebelum menjalankan kode, penting untuk memastikan bahwa semua library ini telah terinstal di lingkungan Python yang digunakan. Jika belum, instalasi dapat dilakukan melalui `pip` atau `conda`. Dengan demikian, instalasi library merupakan tahap fundamental yang memastikan bahwa kode dapat berjalan dengan baik dan berfungsi sesuai dengan yang diharapkan.

3.2. Setup Path Dataset

```
[ ] path = os.path.abspath(r'C:\Users\Herdy\Downloads\MODEL_SI\DATASET-2')
    print(path)

C:\Users\Herdy\Downloads\MODEL_SI\DATASET-2
```

Create a list of all folders

```
[ ] sub_path = os.path.join(path, '')
    folders = os.listdir(sub_path)
    folders

['Daur-Ulang', 'Organik']
```

Gambar 4. *Code Path* Dataset

Pada tahap awal dalam proyek pengolahan data, salah satu langkah penting yang perlu dilakukan adalah menentukan lokasi atau path dari dataset yang akan digunakan. Path ini menunjuk pada direktori di mana dataset tersebut disimpan dalam sistem file komputer kita. Pada kode yang diberikan, kita memanfaatkan fungsi `os.path.abspath()` untuk mendapatkan path absolut dari direktori dataset. Dalam contoh ini, path absolut yang digunakan adalah `C:\Users\Herdy\Downloads\MODEL_SI\DATASET-2`. Penggunaan path absolut ini penting untuk memastikan bahwa kita selalu mengakses lokasi yang tepat dari dataset, terlepas dari direktori kerja saat ini.

Setelah kita memiliki path absolut, langkah berikutnya adalah menggabungkannya dengan sub-direktori atau file lain menggunakan fungsi `os.path.join()`. Dalam contoh ini, `sub_path` adalah hasil penggabungan `path` dengan string kosong (`""`), sehingga nilainya tetap sama dengan `path`. Kemudian, kita menggunakan fungsi `os.listdir()` untuk mendapatkan daftar folder atau file yang ada di dalam direktori `sub_path`. Fungsi ini akan mengembalikan daftar nama folder yang terdapat di dalam path yang telah ditentukan, yang nantinya bisa kita gunakan untuk memproses setiap folder atau file dalam dataset tersebut.

Dengan demikian, penentuan dan pengaturan path dataset ini merupakan langkah awal yang esensial dalam proses pengolahan data, karena memastikan bahwa kita dapat mengakses dan memanipulasi dataset dengan tepat dalam proyek yang sedang kita kerjakan.

3.3. Load Images

```
[ ] def load_imgs(path: str) -> List[str]:

        """

        Params:
        - path : str : The Path of the dataset

        This Function takes the default Path of the dataset and returns the list of images and their labels
        Note the Images are the Pathe of the images

        return:
        - imgs : List[str] : List of the images

        """

        dirs = os.listdir(path)
        imgs = []
        labels = []
        for folder in dirs:
            for img in os.listdir(os.path.join(path, folder)):
                img_path = os.path.join(path, folder, img)
                imgs.append(img_path)
                labels.append(folder)
        return imgs, labels
```

Gambar 5. *Code* Load *Images*

Setelah path dataset berhasil diatur, langkah selanjutnya adalah memuat gambar-gambar dari direktori tersebut agar dapat diproses lebih lanjut, seperti untuk pelatihan model atau analisis data. Fungsi `load_imgs()` yang disediakan bertujuan untuk melakukan hal ini dengan mengumpulkan semua gambar beserta labelnya dari direktori yang telah ditentukan. Fungsi ini menerima satu parameter, yaitu `path`, yang merupakan lokasi direktori dataset. Pertama, fungsi ini mengambil daftar semua folder yang ada di dalam direktori tersebut menggunakan `os.listdir(path)`, di mana setiap folder dianggap sebagai kategori atau label dari dataset.

Selanjutnya, fungsi ini menginisialisasi dua list kosong, yaitu `imgs` dan `labels`, yang akan digunakan untuk menyimpan path dari setiap gambar serta labelnya. Fungsi ini kemudian melakukan iterasi melalui setiap folder, dan di dalam setiap folder, iterasi dilakukan untuk setiap gambar yang ada. Path

lengkap untuk setiap gambar dibentuk dengan menggabungkan `path` utama, nama folder (yang berfungsi sebagai label), dan nama file gambar menggunakan `os.path.join()`. Path gambar yang dihasilkan kemudian disimpan dalam list `imgs`, sementara nama folder sebagai labelnya disimpan dalam list `labels`. Pada akhir fungsi, kedua list ini dikembalikan untuk digunakan dalam proses selanjutnya. Dengan demikian, fungsi `load_imgs()` memastikan bahwa semua gambar dan label dalam dataset telah berhasil dimuat dan siap digunakan untuk tahap pemrosesan berikutnya.

```
def show_images(imgs: List[str] , labels: List[str]) -> None:

    """

    Params:
    - imgs : List[str] : List of the images
    - labels : List[str] : List of the labels of the images

    This Function takes the list of images and their labels and shows
    25 random images with their labels

    return:
    - None

    """

    plt.figure(figsize=(16,12))
    for i in range(25):
        j = random.randint(0,len(imgs))
        plt.subplot(5,5,i+1)
        plt.imshow(plt.imread(imgs[j]))
        plt.title(labels[j])
        plt.axis('off')
    plt.show()
```

Gambar 6. Display Random Images

Setelah gambar dan label berhasil dimuat, langkah berikutnya adalah menampilkan beberapa contoh gambar dari dataset untuk mendapatkan gambaran visual mengenai data yang akan kita proses. Fungsi `show_images()` bertujuan untuk melakukan ini dengan menampilkan 25 gambar secara acak beserta labelnya dalam format grid 5x5. Fungsi ini menerima dua parameter utama, yaitu `imgs`, yang merupakan daftar path gambar, dan `labels`, yang merupakan daftar label yang sesuai dengan gambar-gambar tersebut. Pertama-tama, fungsi ini membuat sebuah figure dengan ukuran 16x12 inci untuk menampung grid gambar. Kemudian, fungsi ini melakukan iterasi sebanyak 25 kali, di mana pada setiap iterasi, satu gambar dipilih secara acak dari daftar `imgs` menggunakan indeks acak. Gambar tersebut kemudian ditampilkan dalam subplot yang disusun dalam grid 5x5, dengan setiap gambar diberi judul sesuai labelnya dan sumbu subplot dihapus untuk memberikan tampilan yang lebih fokus pada gambar. Setelah semua gambar ditambahkan ke grid, hasilnya ditampilkan menggunakan `plt.show()`. Dengan demikian, fungsi ini sangat membantu dalam memvisualisasikan dataset secara langsung, yang penting untuk memahami karakteristik data sebelum melanjutkan ke tahap pemrosesan atau pelatihan model.

3.4. Label Encoding

```
def label_encoding(labels: List[str]) -> Tuple[List[int], Dict[str, int]]:

    """

    Params:
    - labels : List[str] : List of the labels

    This function takes the list of labels and returns the encoded labels and the label dictionary

    return:
    - encoded_labels : List[int] : List of the encoded labels
    - label_dict : Dict[str, int] : Dictionary of the labels and their encoded values

    """

    label_dict = {k:v for v,k in enumerate(np.unique(labels))}
    encoded_labels = [label_dict[label] for label in labels]
    return encoded_labels, label_dict
```

Gambar 7. *Code Label Encoding*

Setelah menampilkan gambar dan label untuk mendapatkan gambaran visual dari dataset, langkah berikutnya adalah mengubah label-label kategorikal menjadi format numerik yang dapat diproses oleh algoritma machine learning. Fungsi `label_encoding()` berfungsi untuk melakukan encoding pada label tersebut, sehingga setiap label kategorikal dapat diubah menjadi representasi numerik yang unik.

Fungsi `label_encoding()` menerima satu parameter, yaitu `labels`, yang merupakan daftar label kategorikal yang sudah dimuat dari dataset. Langkah pertama dalam fungsi ini adalah membuat sebuah dictionary `label_dict` yang memetakan setiap label unik ke nilai integer yang sesuai. Dictionary ini dibuat dengan menggunakan dictionary comprehension, di mana `np.unique(labels)` menghasilkan

semua label unik dalam daftar, dan `enumerate()` memberikan indeks numerik untuk setiap label unik tersebut.
Selanjutnya, fungsi ini membuat list `encoded_labels` yang berisi label yang telah diencoding. Proses encoding dilakukan dengan mencocokkan setiap label dalam daftar `labels` dengan nilai integer yang sesuai dari `label_dict`.
Fungsi ini kemudian mengembalikan dua output: `encoded_labels`, yaitu daftar label yang sudah diencoding dalam format numerik, dan `label_dict`, yaitu dictionary yang menunjukkan pemetaan antara label kategorikal asli dengan nilai integer yang sesuai. Dengan adanya encoding ini, label-label kategorikal dapat dengan mudah digunakan dalam model machine learning, yang umumnya memerlukan data dalam format numerik untuk pelatihan dan evaluasi.

3.5. Load Data

```python
def load_data(path: str) -> Tuple[np.array, np.array, Dict[str, int]]:

    """

    Params:
    - path : str : The Path of the dataset

    This function takes the path of the dataset and returns the images, encoded labels and the label dictionary
    Note that Images are the numpy array of the images

    return:
    - img_arr : np.array : Numpy array of the images
    - encoded_labels : np.array : Numpy array of the encoded labels
    - label_dict : Dict[str, int] : Dictionary of the labels and their encoded values

    """

    imgs, labels = load_imgs(path)
    encoded_labels, label_dict = label_encoding(labels)
    img_arr = []
    for img in imgs:
        img = cv2.imread(img)
        img = cv2.resize(img, (150,150))
        img = img/255
        img_arr.append(img)
    img_arr = np.array(img_arr)
    encoded_labels = np.array(encoded_labels)
    return img_arr, encoded_labels, label_dict
```

Gambar 8. *Code Load* Data

Setelah berhasil memuat dan mengenkode label, langkah selanjutnya adalah memproses gambar-gambar dari dataset agar siap digunakan dalam pelatihan model machine learning. Fungsi `load_data()` dirancang untuk melakukan langkah-langkah ini secara bersamaan. Pertama, fungsi ini mengambil path direktori dataset sebagai input dan memanggil fungsi `load_imgs(path)` untuk mendapatkan daftar path gambar dan label dari dataset. Kemudian, label-label tersebut diencoding menjadi format numerik menggunakan `label_encoding(labels)`.
Setelah itu, fungsi ini memproses setiap gambar dengan cara membaca gambar dari path yang diberikan menggunakan `cv2.imread(img)`, kemudian mengubah ukuran gambar menjadi 150x150 piksel untuk memastikan konsistensi ukuran menggunakan `cv2.resize(img, (150,150))`. Gambar yang sudah diubah ukurannya dinormalisasi dengan membaginya dengan 255, sehingga nilai piksel berada dalam rentang [0, 1], yang mempermudah pelatihan model.
Semua gambar yang telah diproses kemudian disimpan dalam list `img_arr`, yang akhirnya diubah menjadi array NumPy untuk memudahkan manipulasi data. Demikian pula, label yang sudah diencoding juga diubah menjadi array NumPy. Fungsi ini mengembalikan tiga output: array gambar yang telah diproses, array label yang telah diencoding, dan dictionary yang memetakan label asli ke

nilai numeriknya. Dengan cara ini, fungsi `load_data()` mempersiapkan data dalam format yang tepat dan siap digunakan untuk pelatihan model machine learning.

3.6. Count Labels

```python
def count_labels(labels: List[str]) -> Dict[str, int]:

    """

    Params:
    - labels : List[str] : List of the labels

    This function takes the list of labels and returns the dictionary of the labels and their counts

    return:
    - label_dict : Dict[str, int] : Dictionary of the labels and their counts

    """

    label_dict = {}
    for label in labels:
        if label in label_dict:
            label_dict[label] += 1
        else:
            label_dict[label] = 1
    return label_dict
```

Gambar 9. *Code Count Labels*

Setelah memproses gambar dan label, langkah berikutnya adalah menghitung jumlah kemunculan setiap label dalam dataset untuk memahami distribusi data. Fungsi `count_labels()` dirancang untuk melakukan tugas ini dengan mengidentifikasi berapa kali setiap label muncul dalam daftar. Fungsi ini menerima satu parameter, yaitu `labels`, yang merupakan daftar dari label-label yang sudah dimuat dan diencoding sebelumnya.

Fungsi ini dimulai dengan menginisialisasi sebuah dictionary kosong, `label_dict`, yang akan digunakan untuk menyimpan jumlah kemunculan setiap label. Fungsi kemudian melakukan iterasi melalui setiap label dalam daftar `labels`. Untuk setiap label, fungsi memeriksa apakah label tersebut sudah ada dalam dictionary `label_dict`. Jika sudah ada, jumlah kemunculannya akan ditambah satu. Jika belum ada, label tersebut akan ditambahkan ke dalam dictionary dengan nilai awal satu.

Setelah proses iterasi selesai, dictionary `label_dict` akan berisi pasangan kunci-nilai di mana kuncinya adalah label, dan nilainya adalah jumlah kemunculan label tersebut dalam dataset. Fungsi ini kemudian mengembalikan dictionary tersebut. Dengan cara ini, fungsi `count_labels()` membantu dalam menganalisis distribusi label dalam dataset, yang penting untuk memahami keseimbangan data dan potensi kebutuhan untuk teknik penyeimbangan data jika diperlukan.

3.7. Plotting Training History

```python
def plot_history(history: keras.callbacks.History) -> None:

    """

    Params:
    - history : keras.callbacks.History : The History object of the model

    This function takes the history object of the model and plots the accuracy and loss of the model

    return:
    - None

    """
    fig, ax = plt.subplots(1,2, figsize=(16,6))
    ax[0].plot(history.history['accuracy'], label='accuracy')
    ax[0].plot(history.history['val_accuracy'], label='val_accuracy')
    ax[0].legend()
    ax[0].set_title('Accuracy')
    ax[1].plot(history.history['loss'], label='loss')
    ax[1].plot(history.history['val_loss'], label='val_loss')
    ax[1].legend()
    ax[1].set_title('Loss')
    plt.show()
```

Gambar 10. *Code Plotting Training History*

Setelah memproses gambar dan label, serta menghitung distribusi label, langkah berikutnya adalah mengevaluasi performa model machine learning yang telah dilatih. Fungsi `plot_history()` dirancang untuk memvisualisasikan metrik kinerja model selama pelatihan. Fungsi ini menerima satu parameter, yaitu `history`, yang merupakan objek dari `keras.callbacks.History`. Objek ini berisi data sejarah pelatihan model, termasuk akurasi dan loss selama setiap epoch.

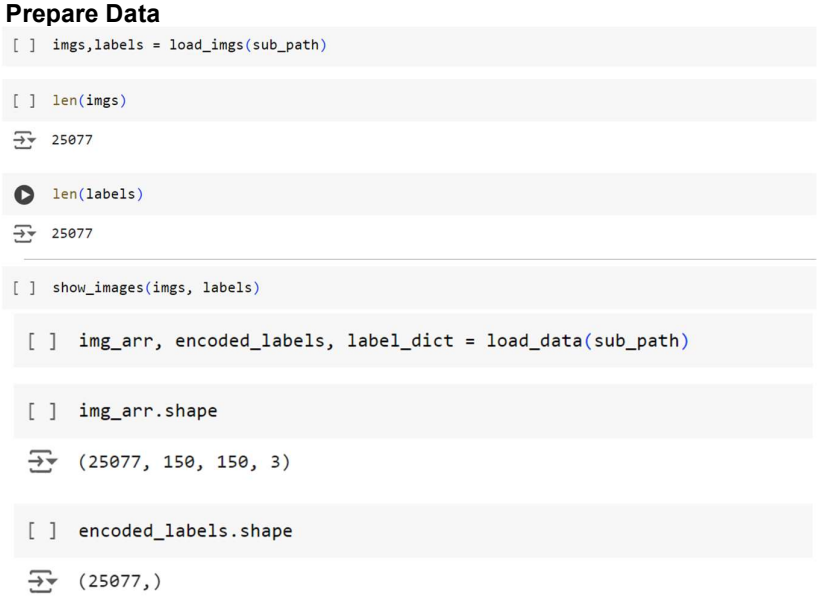
Fungsi ini memulai dengan membuat dua subplot dalam satu figure menggunakan `plt.subplots(1,2, figsize=(16,6))`. Ukuran figure diatur untuk menampung dua plot berdampingan, satu untuk akurasi dan

satu untuk loss. Subplot pertama (`ax[0]`) digunakan untuk menampilkan grafik akurasi. Fungsi ini memplot dua kurva: satu untuk akurasi pelatihan (`history.history['accuracy']`) dan satu lagi untuk akurasi validasi (`history.history['val_accuracy']`). Kurva-kurva ini kemudian diberi label dan legenda agar mudah dibedakan, dan subplot ini diberi judul "Accuracy".

Subplot kedua (`ax[1]`) menampilkan grafik loss. Fungsi ini memplot kurva loss pelatihan (`history.history['loss']`) dan loss validasi (`history.history['val_loss']`). Sama seperti subplot pertama, kurva-kurva ini diberi label dan legenda, serta subplot ini diberi judul "Loss".

Setelah kedua grafik selesai dipasang, fungsi `plt.show()` dipanggil untuk menampilkan figure. Dengan menggunakan `plot_history()`, kita dapat memvisualisasikan bagaimana akurasi dan loss model berubah selama pelatihan, yang membantu dalam menilai performa model dan mengidentifikasi potensi masalah seperti overfitting atau underfitting.

3.8. Load and Prepare Data



Gambar 11. *Code Load and Prepare Data*

Langkah berikutnya adalah memeriksa dan memvisualisasikan data yang telah dimuat untuk memastikan bahwa semuanya telah diatur dengan benar. Pertama, kita menggunakan fungsi `load_imgs(sub_path)` untuk memuat gambar dan label dari direktori yang telah ditentukan. Dengan memanggil fungsi ini, kita mendapatkan dua output: `imgs`, yang berisi daftar path gambar, dan `labels`, yang berisi daftar label yang sesuai.

Untuk memastikan bahwa data telah dimuat dengan benar, kita dapat memeriksa jumlah gambar dan label yang ada dengan menggunakan `len(imgs)` dan `len(labels)`. Ini memberikan informasi mengenai total gambar dan label yang ada dalam dataset.

Selanjutnya, kita menggunakan fungsi `show_images(imgs, labels)` untuk menampilkan beberapa contoh gambar dari dataset beserta labelnya. Fungsi ini membantu kita memeriksa dan memastikan bahwa gambar dan label yang dimuat benar-benar sesuai dengan yang diharapkan, serta memberikan gambaran visual mengenai data yang akan diproses.

Setelah itu, kita memanggil fungsi `load_data(sub_path)` untuk memproses gambar dan label lebih lanjut. Fungsi ini mengembalikan tiga output: `img_arr`, yaitu array yang berisi gambar-gambar yang telah diproses dan dinormalisasi; `encoded_labels`, yaitu array yang berisi label yang telah diencoding menjadi format numerik; dan `label_dict`, yaitu dictionary yang memetakan label asli ke nilai numeriknya.

Untuk memastikan bahwa data telah diproses dengan benar, kita memeriksa bentuk dari `img_arr` dan `encoded_labels` menggunakan `img_arr.shape` dan `encoded_labels.shape`. Ini memberikan

informasi mengenai dimensi array gambar dan label, memastikan bahwa semua data telah diproses dengan benar dan siap digunakan untuk tahap selanjutnya, seperti pelatihan model machine learning.

3.9. Split Dataset

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(img_arr, encoded_labels, test_size=0.2, random_state=42)
```

Gambar 12. *Code Split Dataset*

Langkah berikutnya adalah membagi data menjadi set pelatihan dan set pengujian untuk melatih dan mengevaluasi model machine learning. Fungsi `train_test_split()` dari scikit-learn digunakan untuk melakukan pembagian ini. Fungsi ini menerima beberapa parameter, di antaranya `img_arr`, yang berisi array gambar yang telah diproses; `encoded_labels`, yang berisi label yang telah diencoding; serta parameter `test_size`, yang menentukan proporsi data yang akan digunakan sebagai set pengujian.

Dalam hal ini, `test_size=0.2` berarti 20% dari data akan digunakan untuk set pengujian, sementara sisa 80% akan digunakan untuk set pelatihan. Parameter `random_state=42` digunakan untuk memastikan bahwa pembagian data dilakukan secara konsisten setiap kali kode dijalankan. Dengan menetapkan nilai untuk `random_state`, kita dapat memastikan hasil yang sama pada setiap eksekusi, yang berguna untuk replikasi dan evaluasi model.

Hasil dari fungsi `train_test_split()` adalah empat variabel: `X_train` dan `X_test` yang masing-masing berisi data gambar untuk pelatihan dan pengujian; serta `y_train` dan `y_test` yang masing-masing berisi label yang sesuai untuk pelatihan dan pengujian. Pembagian ini memungkinkan model untuk dilatih pada satu subset data (`X_train` dan `y_train`) dan dievaluasi pada subset data yang terpisah (`X_test` dan `y_test`). Dengan cara ini, kita dapat menilai performa model pada data yang tidak terlihat sebelumnya, memberikan gambaran yang lebih akurat tentang bagaimana model akan bekerja pada data baru di dunia nyata.

3.10. Build CNN Model

```
CNN_model = keras.Sequential([keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
                              keras.layers.MaxPooling2D(2,2),
                              keras.layers.Conv2D(64, (3,3), activation='relu'),
                              keras.layers.MaxPooling2D(2,2),
                              keras.layers.Conv2D(128, (3,3), activation='relu'),
                              keras.layers.MaxPooling2D(2,2),
                              keras.layers.Conv2D(256, (3,3), activation='relu'),
                              keras.layers.MaxPooling2D(2,2),
                              keras.layers.Flatten(),
                              keras.layers.Dropout(0.3),
                              keras.layers.Dense(512, activation='relu'),
                              keras.layers.Dense(2, activation='sigmoid')])

[ ]  CNN_model.summary()
```

Gambar 13. *Code Build CNN Model*

Setelah membagi data menjadi set pelatihan dan pengujian, langkah berikutnya adalah membangun model Convolutional Neural Network (CNN) untuk melakukan klasifikasi gambar. Model ini didefinisikan menggunakan `keras.Sequential()`, yang memungkinkan kita untuk menyusun lapisan-lapisan model secara berurutan.

Model dimulai dengan lapisan `Conv2D` pertama yang memiliki 32 filter dengan ukuran kernel 3x3 dan menggunakan fungsi aktivasi ReLU. Lapisan ini berfungsi untuk mengekstrak fitur dasar dari gambar dengan ukuran input (150,150,3), yang berarti gambar berukuran 150x150 piksel dengan tiga saluran warna (RGB). Setelah itu, lapisan `MaxPooling2D` dengan ukuran pool 2x2 diterapkan untuk mengurangi dimensi spasial gambar dan mengurangi jumlah parameter yang perlu dipelajari.

Model kemudian melanjutkan dengan beberapa lapisan `Conv2D` tambahan (64, 128, dan 256 filter) yang semakin mendalam, serta lapisan `MaxPooling2D` yang menyertainya untuk terus mengurangi dimensi spasial dari fitur. Proses ini memungkinkan model untuk mengekstrak fitur yang semakin kompleks dari gambar.

Setelah semua lapisan konvolusi dan pooling, output dari lapisan terakhir diubah menjadi vektor 1D menggunakan lapisan `Flatten`. Lapisan ini mengubah data 2D menjadi format yang bisa diterima oleh lapisan dense berikutnya. Selanjutnya, lapisan `Dropout` dengan rate 0.3 diterapkan untuk mengurangi

risiko overfitting dengan secara acak menonaktifkan 30% neuron selama pelatihan, sehingga model lebih generalis dan tidak terlalu bergantung pada fitur tertentu.

Kemudian, lapisan `Dense` dengan 512 neuron dan fungsi aktivasi ReLU ditambahkan untuk memproses fitur yang telah diekstrak dan menghasilkan representasi yang lebih terstruktur. Lapisan terakhir adalah lapisan `Dense` dengan 2 neuron dan fungsi aktivasi sigmoid, yang menghasilkan output probabilitas untuk dua kelas yang berbeda (misalnya, berbisa dan tidak berbisa). Terakhir, kita menggunakan `CNN_model.summary()` untuk menampilkan ringkasan arsitektur model, yang memberikan gambaran menyeluruh tentang struktur dan kompleksitas model sebelum memulai proses pelatihan.

3.11. Compile Model

```
CNN_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Gambar 14. *Code Compile Model*

Langkah selanjutnya adalah mengompilasi model agar siap untuk dilatih. Langkah ini dilakukan dengan menggunakan metode `compile()` pada objek model. Pertama, model dikompilasi dengan memilih optimizer, loss function, dan metrics yang akan digunakan selama pelatihan. Dalam hal ini, optimizer yang dipilih adalah `adam`, yang merupakan salah satu algoritma optimisasi populer untuk pelatihan model machine learning. Optimizer ini membantu menyesuaikan bobot model berdasarkan gradient descent untuk meminimalkan loss function. Loss function yang digunakan adalah `sparse_categorical_crossentropy`, yang cocok untuk masalah klasifikasi dengan label numerik. Loss function ini mengukur seberapa baik model memprediksi label yang benar, dan tujuan pelatihan adalah meminimalkan nilai loss ini. Selain itu, kita juga menentukan metrik yang akan digunakan untuk mengevaluasi performa model selama pelatihan, dalam hal ini `accuracy`. Metrik ini menghitung persentase prediksi yang benar dibandingkan dengan total prediksi, memberikan gambaran mengenai akurasi model pada data pelatihan.Dengan langkah ini, model telah siap untuk proses pelatihan, di mana optimizer akan menyesuaikan bobot berdasarkan loss function, dan metrik akurasi akan digunakan untuk memantau performa model.

3.12. Training Model

```
[ ] history = CNN_model.fit(X_train, y_train, epochs=25,batch_size=128, validation_split=0.2, callbacks=callbacks)

Epoch 1/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 159s 1s/step - accuracy: 0.7196 - loss: 0.6549 - val_accuracy: 0.8166 - val_loss: 0.4481
Epoch 2/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 164s 1s/step - accuracy: 0.8375 - loss: 0.3900 - val_accuracy: 0.8256 - val_loss: 0.3981
Epoch 3/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 151s 1s/step - accuracy: 0.8484 - loss: 0.3553 - val_accuracy: 0.8615 - val_loss: 0.3405
Epoch 4/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.8651 - loss: 0.3224 - val_accuracy: 0.8732 - val_loss: 0.3110
Epoch 5/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.8646 - loss: 0.3270 - val_accuracy: 0.8791 - val_loss: 0.2998
Epoch 6/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.8819 - loss: 0.2945 - val_accuracy: 0.8824 - val_loss: 0.3012
Epoch 7/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.8842 - loss: 0.2821 - val_accuracy: 0.8747 - val_loss: 0.3101
Epoch 8/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.8991 - loss: 0.2492 - val_accuracy: 0.8826 - val_loss: 0.2929
Epoch 9/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 145s 1s/step - accuracy: 0.9050 - loss: 0.2409 - val_accuracy: 0.8764 - val_loss: 0.3069
Epoch 10/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.9160 - loss: 0.2120 - val_accuracy: 0.8871 - val_loss: 0.3002
Epoch 11/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.9254 - loss: 0.1899 - val_accuracy: 0.8682 - val_loss: 0.3845
Epoch 12/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.9311 - loss: 0.1747 - val_accuracy: 0.8627 - val_loss: 0.3765
Epoch 13/25
126/126 ━━━━━━━━━━━━━━━━━━━━ 149s 1s/step - accuracy: 0.9390 - loss: 0.1534 - val_accuracy: 0.8804 - val_loss: 0.3573
```

Gambar 7. *Code Training Model*

Setelah mengompilasi model, langkah selanjutnya adalah melatih model dengan menggunakan data pelatihan. Ini dilakukan dengan metode `fit()` pada objek model, yang mengatur proses pelatihan model CNN.

Fungsi `fit()` memerlukan beberapa parameter untuk mengatur pelatihan. Pertama, kita menentukan `X_train` dan `y_train` sebagai data gambar dan label pelatihan yang akan digunakan untuk melatih model. Model akan belajar dari data ini untuk mengoptimalkan bobotnya.

Parameter `epochs=25` menunjukkan bahwa model akan dilatih selama 25 iterasi penuh melalui seluruh data pelatihan. Setiap epoch mencakup proses feed-forward dan backpropagation untuk memperbarui bobot model.

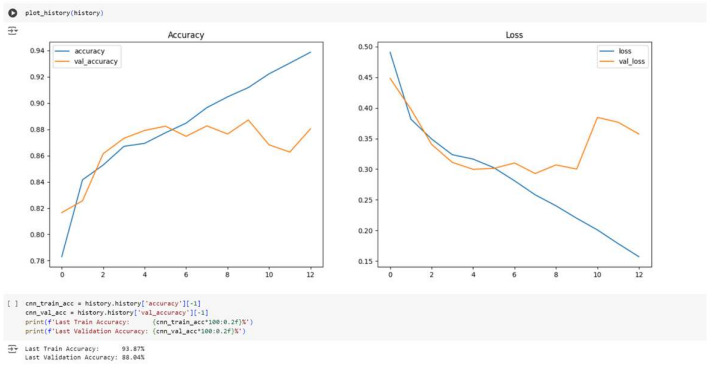
`batch_size=128` mengatur ukuran batch, yaitu jumlah gambar yang diproses sekaligus dalam satu iterasi pelatihan. Menggunakan ukuran batch yang lebih besar dapat mempercepat pelatihan, tetapi memerlukan lebih banyak memori.

`validation_split=0.2` berarti 20% dari data pelatihan akan dipisahkan dan digunakan sebagai data validasi selama pelatihan. Ini memungkinkan kita untuk mengevaluasi performa model pada data yang tidak digunakan untuk pelatihan, membantu mendeteksi overfitting.

Terakhir, parameter `callbacks=callbacks` memungkinkan penggunaan callback selama pelatihan. Callback dapat digunakan untuk berbagai tujuan, seperti menyimpan model terbaik atau menghentikan pelatihan lebih awal jika model tidak menunjukkan peningkatan performa.

Selama pelatihan, hasil dari setiap epoch, termasuk akurasi dan loss pada data pelatihan serta data validasi, disimpan dalam objek `history`. Ini memungkinkan kita untuk memantau perkembangan model dan mengevaluasi kinerjanya selama pelatihan.

3.13. Evaluate Model



Gambar 15. Evaluasi Model

```
[ ]  cnn_y_pred = CNN_model.predict(X_test)
     cnn_y_pred = np.argmax(cnn_y_pred, axis=1)
     cnn_report = classification_report(y_test, cnn_y_pred)
     print(cnn_report)

157/157 ━━━━━━━━━━━━━━━━━━━━ 11s 70ms/step
              precision    recall  f1-score   support

           0       0.89      0.83      0.86      2268
           1       0.87      0.92      0.89      2748

    accuracy                           0.88      5016
   macro avg       0.88      0.88      0.88      5016
weighted avg       0.88      0.88      0.88      5016
```

Gambar 16. Evaluasi Model

Setelah melatih model dengan menggunakan metode `fit()`, langkah berikutnya adalah mengevaluasi performa model dan memvisualisasikan hasilnya.
Pertama, fungsi `plot_history(history)` digunakan untuk menampilkan grafik akurasi dan loss selama pelatihan. Grafik ini membantu kita memahami bagaimana akurasi dan loss model berubah seiring berjalannya waktu, baik pada data pelatihan maupun data validasi. Ini penting untuk mengevaluasi apakah model mengalami overfitting atau underfitting.
Kemudian, kita mengambil akurasi pelatihan dan validasi dari objek `history`. `cnn_train_acc` diatur dengan mengambil nilai akurasi pelatihan pada epoch terakhir (`history.history['accuracy'][-1]`), sedangkan `cnn_val_acc` diatur dengan akurasi validasi pada epoch terakhir (`history.history['val_accuracy'][-1]`). Akurasi ini dikalikan dengan 100 untuk mendapatkan persentase, dan kemudian dicetak menggunakan format `0.2f` untuk menampilkan dua angka di belakang koma. Ini memberikan gambaran tentang performa model pada data pelatihan dan validasi.
Selanjutnya, kita melakukan prediksi pada data pengujian (`X_test`) dengan menggunakan metode `predict()` dari model CNN. Hasil prediksi ini adalah probabilitas untuk masing-masing kelas, sehingga kita menggunakan `np.argmax(cnn_y_pred, axis=1)` untuk mengubahnya menjadi label kelas yang diprediksi dengan memilih kelas dengan probabilitas tertinggi.
Terakhir, kita menggunakan fungsi `classification_report()` dari scikit-learn untuk menghasilkan laporan klasifikasi yang mencakup metrik seperti precision, recall, dan F1-score untuk setiap kelas. Laporan ini

memberikan informasi yang lebih mendalam mengenai kinerja model dalam mengklasifikasikan gambar-gambar di data pengujian, membantu kita menilai seberapa baik model dalam mengenali setiap kelas. Hasil dari laporan ini kemudian dicetak untuk evaluasi lebih lanjut.

3.14. Save Model

```
[ ]  CNN_model.save(r'C:\Users\Herdy\Downloads\MODEL_SI\Model_CNN.h5')
```

Gambar 17. *Saving Model*

Setelah melatih dan mengevaluasi model, langkah terakhir adalah menyimpan model yang telah dilatih untuk digunakan di masa depan tanpa perlu melatihnya kembali. Ini dilakukan dengan menggunakan metode `save()` pada objek model.
Dalam kode ini, `CNN_model.save(r'C:\Users\Herdy\Downloads\MODEL_SI\Model_CNN.h5')` menyimpan model ke dalam file dengan nama `Model_CNN.h5` di path yang telah ditentukan. Format file `.h5` adalah format HDF5 yang digunakan untuk menyimpan struktur model, bobot, dan konfigurasi lainnya.
Dengan menyimpan model, kita dapat dengan mudah memuatnya kembali di masa depan untuk melakukan prediksi pada data baru atau melanjutkan pelatihan jika diperlukan. Ini juga memungkinkan kita untuk berbagi model dengan orang lain atau menyebarkannya ke lingkungan produksi tanpa harus melatih model dari awal.

4. Conclusion

Pengelolaan sampah yang efisien semakin dibutuhkan seiring bertambahnya jumlah/volume sampah setiap harinya. Untuk meningkatkan akurasi pemilihan di TPA diusulkan penggunaan teknologi kecerdasan buatan (AI) berbasis deep learning yang diintegrasikan dalam mesin otomatis seperti sinar-X. Metode ini memanfaatkan Convolutional Neural Network (CNN) yangg mampu memproses dan mengklasifikasikan gambar sampah, sehingga dapat mempercepat proses daur ulang dan mempermudah pemilahan sampah. Model CNN dirancang dan dilatih dengan menggunakan dataset gambar sampah, yang bertujuan untuk meningkatkan efisiensi pengelolaan sampah melalui klasifikasi yang lebih akurat. teknologi ini diharapkan menjadi solusi cerdas dalam menghadapi tantangan pengelolaan sampah moderen.

References

[1] Septian, M. R. D., Paliwang, A. A. A., Cahyanti, M., & Swedia, E. R. (2020). Penyakit Tanaman Apel Dari Citra Daun Dengan Convolutional Neural Network. *Sebatik*, *24*(2), 207-212.

[2] Rasidi, A. I., Pasaribu, Y. A. H., Ziqri, A., & Adhinata, F. D. (2022). Klasifikasi sampah organik dan non-organik menggunakan convolutional neural network. *Jurnal Teknik Informatika dan Sistem Informasi*, *8*(1), 142-149.

[3] Sunanto, O. D. S., & Utomo, P. H. (2022). Implementasi Deep Learning dengan Convolutional Neural Network untuk klasifikasi gambar sampah organik dan anorganik. *UNEJ e-Proceeding*, 373-382.

[4] Feriawan, J., & Swanjaya, D. (2020). Perbandingan Arsitektur Visual Geometry Group dan MobileNet Pada Pengenalan Jenis Kayu. In *Prosiding SEMNAS INOTEK (Seminar Nasional Inovasi Teknologi)* (Vol. 4, No. 3, pp. 185-190).

[5] t, Q. Lin, J. Allebach, and E. J. Delp, “Training object detection and recognition CNN models using data augmentation,” *IS T Int. Symp. Electron. Imaging Sci. Technol.*, pp. 27–36, 2017, doi: 10.2352/ISSN.2470-1173.2017.10.IMAWM-163.

[6] D. Konstantinidis, V. Argyriou, T. Stathaki, and N. Grammalidis, “A modular CNN-based building detector for remote sensing images,” *Comput. Networks*, vol. 168, p. 107034, 2020, doi: 10.1016/j.comnet.2019.107034.