

LAPORAN PENGENALAN POLA AKSARA JAWA
KELOMPOK 4



Disusun Oleh :

Anggota Kelompok 4

1. Anggara Putra Meldyantono (32602100029)
2. Anita Soffiyun Nada (32602100030)
3. Arabela Muri Agista (32602100032)
4. Dimas Surya Wirastama (32602100039)

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM SULTAN AGUNG
SEMARANG

2024

BAB I

PROGRESS DATA PROCESSING

1.1 Rencana Pengambilan Data

Data aksara Jawa diambil dari Kaggle. Kaggle adalah platform kompetisi sains data dan komunitas daring bagi para ilmuwan data dan praktisi pembelajaran mesin di bawah Google LLC . Kaggle memungkinkan pengguna untuk menemukan dan menerbitkan kumpulan data, menjelajahi dan membangun model dalam lingkungan sains data berbasis web, bekerja sama dengan ilmuwan data dan teknisi pembelajaran mesin lainnya, serta mengikuti kompetisi untuk memecahkan tantangan sains data.

1.1.1 Preprocessing Data

Preprocessing data adalah langkah penting untuk memastikan data yang digunakan dalam analisis atau model pembelajaran mesin berkualitas tinggi dan sesuai dengan tujuan penelitian.

Pembersihan data:

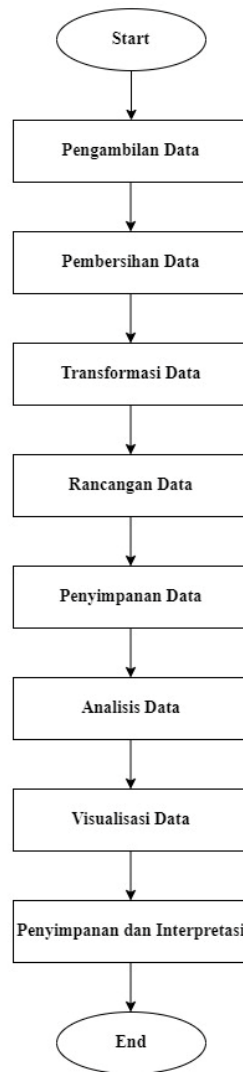
1. Hapus data yang tidak relevan: Buang data yang tidak sesuai dengan tujuan penelitian atau analisis Anda, seperti data yang kosong, duplikat, atau tidak terstruktur.
2. Perbaiki kesalahan data: Koreksi kesalahan ejaan, tanda baca, atau format data yang tidak konsisten.
3. Standarisasi data: Pastikan data aksara Jawa memiliki format yang seragam dan konsisten, seperti konversi ke bentuk normal atau pengkodean karakter yang sesuai.

Transformasi data:

1. Konversi data: Ubah format data ke format yang lebih mudah diolah, seperti konversi teks ke vektor numerik atau representasi matriks.
2. Normalisasi data: Sesuaikan nilai data ke skala yang sama untuk memudahkan perbandingan dan analisis.

3. Fitur engineering: Buat fitur baru dari data yang ada untuk memperkaya informasi dan meningkatkan performa model pembelajaran mesin.

1.2 Flowchart Pemrosesan Data



Gambar di atas merupakan flowschart pemrosesan data yang kami gunakan.

BAB II

MEMBACA DATA DAN PREPROCESSING

2.1 Koleksi Data

2.1.1 Link Kode

https://colab.research.google.com/drive/1dOHaoCQVif_XdshTSfuhSB5lyEJ7g0xR?usp=sharing

2.1.2 Link Dataset Aksara Jawa dari Kaggle

<https://www.kaggle.com/datasets/vzrengamani/hanacaraka>

2.2 Langkah-langkah dan Kode Program Preprocessing

1. Mengimport library

```
import numpy as np import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import
ImageDataGenerator from tensorflow.keras.layers import
Conv2D, MaxPooling2D, MaxPool2D,
Flatten, Dense, Dropout, Input, UpSampling2D, concatenate
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import Callback,
ReduceLROnPlateau from tensorflow.keras.applications
import ResNet50
from tensorflow.keras.regularizers import l2 import
matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore",
category=DeprecationWarning)
```

2. Upload dataset yang sudah didownload

```
from google.colab import files uploaded = files.upload()
<IPython.core.display.HTML object> Saving dataset.zip to
dataset.zip
!unzip dataset.zip
```

3. Membaca koleksi data dan preprocessing

```
# Folder tempat gambar diekstrak train_dir =
'/content/train' valid_dir = '/content/validation'
```

```

# ImageDataGenerator untuk preprocessing dan augmentasi
gambar
train_datagen = ImageDataGenerator( rescale=1./255,
rotation_range=40, width_shift_range=0.2,
height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
horizontal_flip=True, fill_mode='nearest'
)
# Preprocessing
valid_datagen = ImageDataGenerator(rescale=1./255)
# Generator data training
train_generator = train_datagen.flow_from_directory(
train_dir, # Change the directory path if
necessarytarget_size=(150, 150), color_mode='grayscale',
batch_size=32, class_mode='categorical',
shuffle=True
)

# Generator data validasi
valid_generator = valid_datagen.flow_from_directory(
valid_dir, # Change the directory path if
necessarytarget_size=(150, 150), color_mode='grayscale',
batch_size=32, class_mode='categorical',
shuffle=False
)

Found 1259 images belonging to 21 classes.Found 320 images
belonging to 21 classes.

labels = train_generator.class_indices print(labels)
{'ba': 0, 'ca': 1, 'da': 2, 'data': 3, 'dha': 4, 'ga': 5,
'ha': 6,
'ja': 7, 'ka': 8, 'la': 9, 'ma': 10, 'na': 11, 'nga': 12,
'nya':
13,
'pa': 14, 'ra': 15, 'sa': 16, 'ta': 17, 'tha': 18, 'wa':
19, 'ya':
20}

```

4. Visualization

```
import os
import matplotlib.pyplot as plt
def plot_train_images(data_dir, labels): num_classes =
len(labels)
fig, axes = plt.subplots(1, num_classes, figsize=(20, 5))
#
Adjust figsize for better visualization
axes = axes.flatten()
for idx, class_name in enumerate(labels): image_dir =
os.path.join(data_dir, class_name) # Handle the case where
the directory might be empty
if os.listdir(image_dir):
image_path = os.path.join(image_dir,
os.listdir(image_dir)[0]) # Access the first image if
image = plt.imread(image_path)
axes[idx].imshow(image) axes[idx].set_title(class_name)
axes[idx].axis('off')
else:
axes[idx].axis('off')
axes[idx].set_title(f"{class_name}\n(Empty)")
plt.tight_layout() plt.show()
# Convert the dictionary to a list of class names
labels = {'ha': 0, 'na': 1, 'ca': 2, 'ra': 3, 'ka': 4,
'da': 5, 'ta':
6, 'sa': 7, 'wa': 8, 'la': 9, 'pa': 10, 'dha': 11, 'ja':
12, 'ya': 13,
'nya': 14, 'ma': 15, 'ga': 16, 'ba': 17, 'tha': 18, 'nga':
19}
class_names = list(labels.keys())
train_dir = '/content/train' # Update this path as per
your directory structure
plot_train_images(train_dir, class_names)
```



```
import os
import matplotlib.pyplot as plt
```


```

def plot_train_images(data_dir, labels): num_classes =
len(labels)
fig, axes = plt.subplots(1, num_classes, figsize=(20,
5)) #
Adjust figsize for better visualization
axes = axes.flatten()

for idx, class_name in enumerate(labels): image_dir =
os.path.join(data_dir, class_name)
# Handle the case where the directory might be empty
if os.listdir(image_dir):
image_path = os.path.join(image_dir,
os.listdir(image_dir)[0]) # Access the first image if
available
image = plt.imread(image_path)

axes[idx].imshow(image) axes[idx].set_title(class_name)
axes[idx].axis('off')
else:
axes[idx].axis('off')
axes[idx].set_title(f"{class_name}\n(Empty)")
plt.tight_layout()
plt.show()
# Convert the dictionary to a list of class names
labels = {'ha': 0, 'na': 1, 'ca': 2, 'ra': 3, 'ka': 4,
'da': 5, 'ta':
6, 'sa': 7, 'wa': 8, 'la': 9, 'pa': 10, 'dha': 11, 'ja':
12, 'ya': 13,
'nya': 14, 'ma': 15, 'ga': 16, 'ba': 17, 'tha': 18,
'nga': 19}
class_names = list(labels.keys())
train_dir = '/content/train' # Update this path as per
your directory structure

```



```

plot_train_images(train_dir, class_names)
import os

```

```

import matplotlib.pyplot as plt

def plot_valid_images(data_dir, labels, title):
    num_classes = len(labels)
    fig, axes = plt.subplots(1, num_classes, figsize=(20,
5)) #
Adjust figsize for better visualization
axes = axes.flatten()

for idx, class_name in enumerate(labels): image_dir =
os.path.join(data_dir, class_name)
# Handle the case where the directory might be empty
if os.listdir(image_dir):
    image_path = os.path.join(image_dir,
os.listdir(image_dir)[0]) # Access the first image if
available
    image = plt.imread(image_path)

    axes[idx].imshow(image) axes[idx].set_title(class_name)
    axes[idx].axis('off')
else:
    axes[idx].axis('off')
    axes[idx].set_title(f"{class_name}\n(Empty)")
plt.suptitle(title, fontsize=16) plt.tight_layout()
plt.show()

# Convert the dictionary to a list of class names
labels = {'ha': 0, 'na': 1, 'ca': 2, 'ra': 3, 'ka': 4,
'da': 5,
'ta':
6, 'sa': 7, 'wa': 8, 'la': 9, 'pa': 10, 'dha': 11, 'ja':
12, 'ya':
13,
'nya': 14, 'ma': 15, 'ga': 16, 'ba': 17, 'tha': 18,
'nga': 19}
class_names = list(labels.keys())
train_dir = '/content/train' # Update this path as per
your directorystructure

```



```
validation_dir = '/content/validation' # Update this
path as per your directory structure
# Plot train images
plot_valid_images(train_dir, class_names, 'Training
Images')
# Plot validation images
plot_valid_images(validation_dir, class_names,
'Validation Images')
```



```
def plot_augmentasi(generator): x_batch, y_batch =
next(generator)
fig, axes = plt.subplots(1, 20, figsize=(20, 20)) axes =
axes.flatten()
for img, ax in zip(x_batch, axes): ax.imshow(img)
ax.axis('off')
plt.tight_layout() plt.show()
plot_augmentasi(train_generator)
```



Dari Langkah-langkah dan kode program di atas dapat disimpulkan bahwa

1. Membaca koleksi data dengan menggunakan ImageDataGenerator dan flow_from_directory, program ini membaca dan mengelola koleksi data gambar dari direktori yang ditentukan.
2. Processing program ini melakukan rescaling pada gambar dan menerapkan berbagai teknik augmentasi pada gambar training.

BAB III

PENGUNAAN METODE PENGENALAN POLA

3.1 Penggunaan Metode

Metode atau model pada project pengenalan pola aksara Jawa ini, menggunakan CNN. Convolutional Neural Network (CNN) adalah salah satu jenis arsitektur jaringan saraf tiruan yang digunakan untuk memproses data gambar dan video. CNN memiliki kemampuan untuk mengenali pola dan fitur pada gambar dengan cara melakukan operasi konvolusi pada setiap piksel gambar. Fungsi utama dari CNN adalah untuk melakukan klasifikasi, deteksi objek, segmentasi, dan pengenalan wajah pada gambar.

3.2 Kode Program dan Output

3.2.1 Kode Program

```
✓ [7] # Folder tempat gambar diekstrak  
      train_dir = '/content/v3/v3/train'  
      valid_dir = '/content/v3/v3/val'  
      test_dir = '/content/prediction'
```

Gambar diatas menjelaskan tempat folder gambar diekstrak, folder ini digunakan untuk mengorganisir gambar-gambar yang akan dipakai dalam berbagai tahap selanjutnya.

✓
0 d



```
# Generator data training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Generator data validasi
test_datagen = ImageDataGenerator(
    rescale=1./255
)
```

Kode di atas mendefinisikan dua objek `'ImageDataGenerator'` untuk memproses data gambar: `'train_datagen'` digunakan untuk data pelatihan dengan rescaling (normalisasi nilai piksel dari 0-255 menjadi 0-1) serta berbagai augmentasi seperti rotasi, pergeseran, shear, zoom, dan flip horizontal untuk meningkatkan variasi data; sedangkan `'test_datagen'` digunakan untuk data validasi yang hanya melakukan rescaling tanpa augmentasi tambahan.

```

✓ 1d # Generator data training dan validasi
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150), # Sesuaikan dengan input_shape dari model
    batch_size=32,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    valid_dir,
    target_size=(150, 150), # Sesuaikan dengan input_shape dari model
    batch_size=32,
    class_mode='categorical'
)

# Generator data uji
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150), # Sesuaikan dengan input_shape dari model
    batch_size=32,
    class_mode='categorical',
    shuffle=False # Jangan acak untuk evaluasi
)

```

Kode di atas membuat tiga generator data menggunakan metode `flow_from_directory` dari `ImageDataGenerator` untuk menyiapkan data pelatihan, validasi, dan pengujian. `train_generator` menggunakan `train_datagen` untuk mengambil gambar dari `train_dir`, mengubah ukurannya menjadi 150x150 piksel, mengatur batch size sebesar 32, dan menetapkan mode kelas sebagai `'categorical'`. `validation_generator` menggunakan `test_datagen` untuk mengambil gambar dari `valid_dir` dengan pengaturan yang sama seperti `train_generator`, namun tanpa augmentasi. `test_generator` juga menggunakan `test_datagen` untuk mengambil gambar dari `test_dir` dengan pengaturan serupa tetapi dengan `shuffle=False` agar data tidak diacak selama evaluasi.

```

✓ 1d [12] class myCallback(tf.keras.callbacks.Callback):
        def on_epoch_end(self, epoch, logs={}):
            if(logs.get('accuracy') > 0.97):
                print("\nAkurasi di atas 97%")
                self.model.stop_training = True

        callbacks = myCallback()

```

Kode di atas mendefinisikan kelas kustom `myCallback` yang diturunkan dari `tf.keras.callbacks.Callback` untuk menghentikan

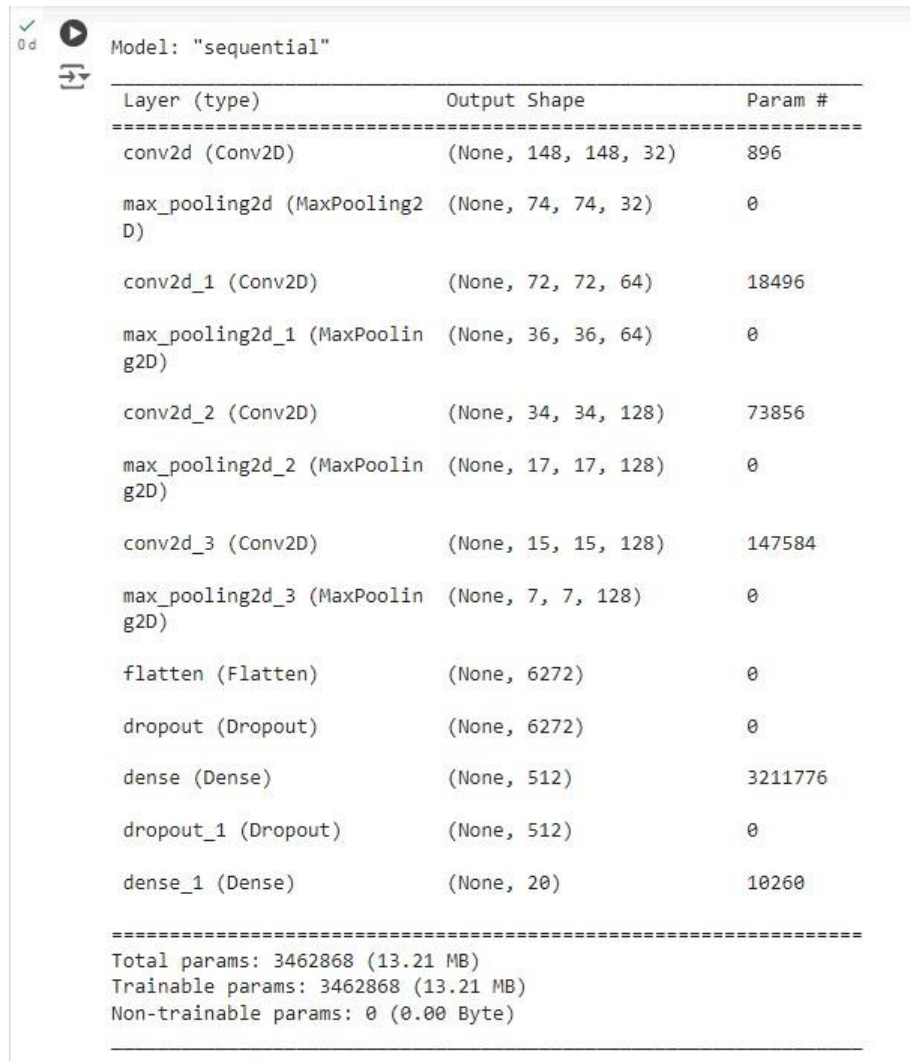
pelatihan model secara otomatis jika akurasi mencapai lebih dari 97%. Metode ``on_epoch_end`` memeriksa akurasi setelah setiap epoch, dan jika akurasi yang diperoleh (``logs.get('accuracy')``) lebih besar dari 0.97, maka mencetak pesan "Akurasi di atas 97%" dan mengatur atribut ``self.model.stop_training`` menjadi ``True`` untuk menghentikan pelatihan. Sebuah instance dari kelas ini, ``callbacks``, kemudian dibuat untuk digunakan dalam proses pelatihan model.

```
✓ 0d model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)), # 3 channels for RGB images
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(20, activation='softmax') # Number of classes
])

model.summary()
```

Kode di atas mendefinisikan arsitektur model Convolutional Neural Network (CNN) menggunakan ``tf.keras.models.Sequential``. Model ini terdiri dari beberapa lapisan: empat lapisan konvolusi (Conv2D) dengan jumlah filter berturut-turut 32, 64, 128, dan 128 serta ukuran kernel 3x3, masing-masing diikuti oleh lapisan max pooling (MaxPooling2D) dengan ukuran 2x2 untuk down-sampling. Setelah itu, lapisan Flatten digunakan untuk meratakan hasil konvolusi menjadi satu dimensi, diikuti oleh dua lapisan dropout dengan probabilitas 0.5 untuk mencegah overfitting, dan satu lapisan Dense dengan 521 unit dan aktivasi ReLU. Lapisan terakhir adalah lapisan Dense dengan 20 unit dan aktivasi softmax untuk klasifikasi menjadi 20 kelas. Model ini kemudian dirangkum menggunakan ``model.summary()`` untuk melihat struktur dan jumlah parameter yang dilatih.

3.2.2 Output



Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dropout (Dropout)	(None, 6272)	0
dense (Dense)	(None, 512)	3211776
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 20)	10260

=====
Total params: 3462868 (13.21 MB)
Trainable params: 3462868 (13.21 MB)
Non-trainable params: 0 (0.00 Byte)

Hasil dari kode di atas menunjukkan struktur detail model CNN yang telah dibuat, termasuk jenis, output shape, dan jumlah parameter untuk setiap layer. Model terdiri dari empat lapisan konvolusi, masing-masing diikuti oleh lapisan max pooling, di mana ukuran keluaran dari lapisan konvolusi berkurang bertahap melalui proses konvolusi dan pooling. Setelah itu, lapisan Flatten meratakan keluaran menjadi satu dimensi, yang kemudian diproses oleh lapisan dense dengan 512 unit dan lapisan dropout untuk mencegah overfitting. Lapisan output menggunakan 20 unit dengan aktivasi softmax untuk klasifikasi multi-kelas. Total parameter yang dapat dilatih

dalam model ini adalah 3,462,868, yang menunjukkan kompleksitas model dan jumlah bobot yang akan disesuaikan selama pelatihan.

```
✓ [14] from keras.optimizers import Adam
0 d

    model.compile(
        loss='categorical_crossentropy',
        optimizer=tf.optimizers.Adam(),
        metrics=['accuracy']
    )
```

Kode di atas mengkompilasi model CNN yang telah didefinisikan dengan menggunakan fungsi loss `categorical_crossentropy`, optimizer `Adam`, dan metrik evaluasi `accuracy`. Fungsi loss `categorical_crossentropy` digunakan karena model ini adalah model klasifikasi multi-kelas, yang cocok untuk menghitung perbedaan antara distribusi output model dan distribusi label sebenarnya. Optimizer `Adam` dipilih karena kemampuannya yang adaptif dalam mengatur learning rate selama pelatihan, yang membantu dalam konvergensi yang lebih cepat dan stabil. Metrik `accuracy` digunakan untuk memantau persentase prediksi yang benar selama proses pelatihan dan evaluasi model.

```
✓ 1) # from keras.callbacks import EarlyStopping

# # Mendefinisikan objek callback EarlyStopping
# callbacks = EarlyStopping(monitor='val_loss', patience=5)

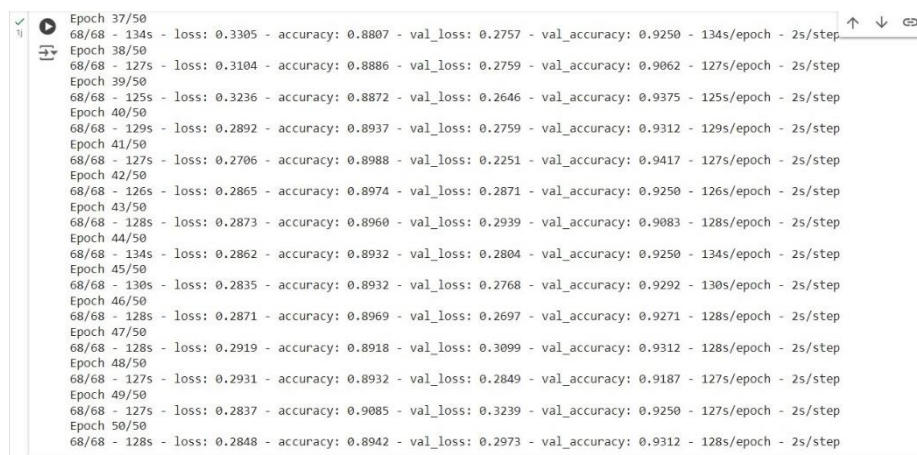
# Melatih model dengan menggunakan generator
history = model.fit(
    train_generator,          # Generator data pelatihan
    epochs=50,               # Jumlah epoch
    validation_data=validation_generator, # Generator data validasi
    verbose=2,               # Tingkat kecerewetan output (0, 1, atau 2)
    # callbacks=[callbacks] # Daftar callbacks yang ingin digunakan
)
```

Kode di atas melatih model CNN menggunakan data dari `train_generator` selama 50 epoch, dengan data validasi dari `validation_generator`. Opsi `verbose=2` digunakan untuk menampilkan output pelatihan yang ringkas namun informatif. Meskipun baris kode untuk `EarlyStopping` diimpor dan objek callback `EarlyStopping`

didefinisikan untuk memantau `val_loss` dan menghentikan pelatihan jika tidak ada perbaikan selama 5 epoch berturut-turut, penggunaan callback ini dikomentari dan tidak diaktifkan dalam pelatihan ini. Hasil pelatihan disimpan dalam variabel `history` yang mencakup informasi tentang loss dan akurasi untuk setiap epoch baik pada data pelatihan maupun validasi.

3.3 Training dan Testing

3.3.1 Training

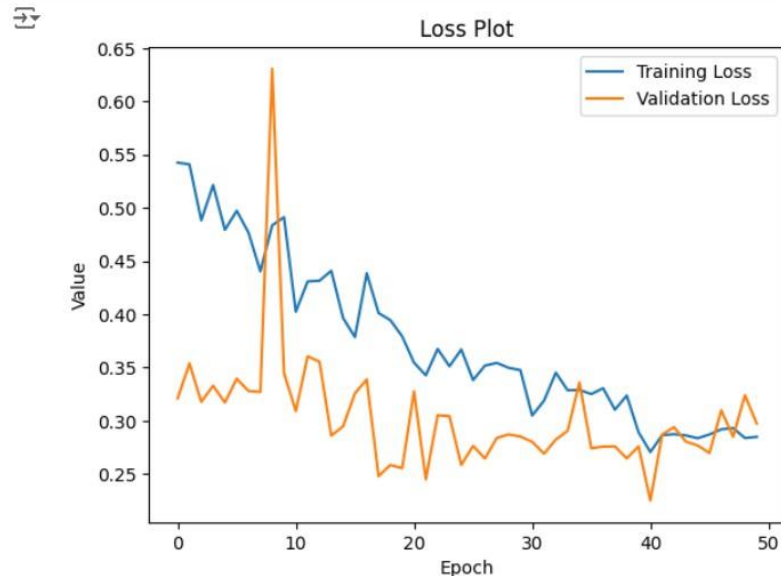


Epoch 37/50	68/68 - 134s - loss: 0.3305 - accuracy: 0.8807 - val_loss: 0.2757 - val_accuracy: 0.9250 - 134s/epoch - 2s/step
Epoch 38/50	68/68 - 127s - loss: 0.3104 - accuracy: 0.8886 - val_loss: 0.2759 - val_accuracy: 0.9062 - 127s/epoch - 2s/step
Epoch 39/50	68/68 - 125s - loss: 0.3236 - accuracy: 0.8872 - val_loss: 0.2646 - val_accuracy: 0.9375 - 125s/epoch - 2s/step
Epoch 40/50	68/68 - 129s - loss: 0.2892 - accuracy: 0.8937 - val_loss: 0.2759 - val_accuracy: 0.9312 - 129s/epoch - 2s/step
Epoch 41/50	68/68 - 127s - loss: 0.2706 - accuracy: 0.8988 - val_loss: 0.2251 - val_accuracy: 0.9417 - 127s/epoch - 2s/step
Epoch 42/50	68/68 - 126s - loss: 0.2865 - accuracy: 0.8974 - val_loss: 0.2871 - val_accuracy: 0.9250 - 126s/epoch - 2s/step
Epoch 43/50	68/68 - 128s - loss: 0.2873 - accuracy: 0.8960 - val_loss: 0.2939 - val_accuracy: 0.9083 - 128s/epoch - 2s/step
Epoch 44/50	68/68 - 134s - loss: 0.2862 - accuracy: 0.8932 - val_loss: 0.2804 - val_accuracy: 0.9250 - 134s/epoch - 2s/step
Epoch 45/50	68/68 - 130s - loss: 0.2835 - accuracy: 0.8932 - val_loss: 0.2768 - val_accuracy: 0.9292 - 130s/epoch - 2s/step
Epoch 46/50	68/68 - 128s - loss: 0.2871 - accuracy: 0.8969 - val_loss: 0.2697 - val_accuracy: 0.9271 - 128s/epoch - 2s/step
Epoch 47/50	68/68 - 128s - loss: 0.2919 - accuracy: 0.8918 - val_loss: 0.3099 - val_accuracy: 0.9312 - 128s/epoch - 2s/step
Epoch 48/50	68/68 - 127s - loss: 0.2931 - accuracy: 0.8932 - val_loss: 0.2849 - val_accuracy: 0.9187 - 127s/epoch - 2s/step
Epoch 49/50	68/68 - 127s - loss: 0.2837 - accuracy: 0.9085 - val_loss: 0.3239 - val_accuracy: 0.9250 - 127s/epoch - 2s/step
Epoch 50/50	68/68 - 128s - loss: 0.2848 - accuracy: 0.8942 - val_loss: 0.2973 - val_accuracy: 0.9312 - 128s/epoch - 2s/step

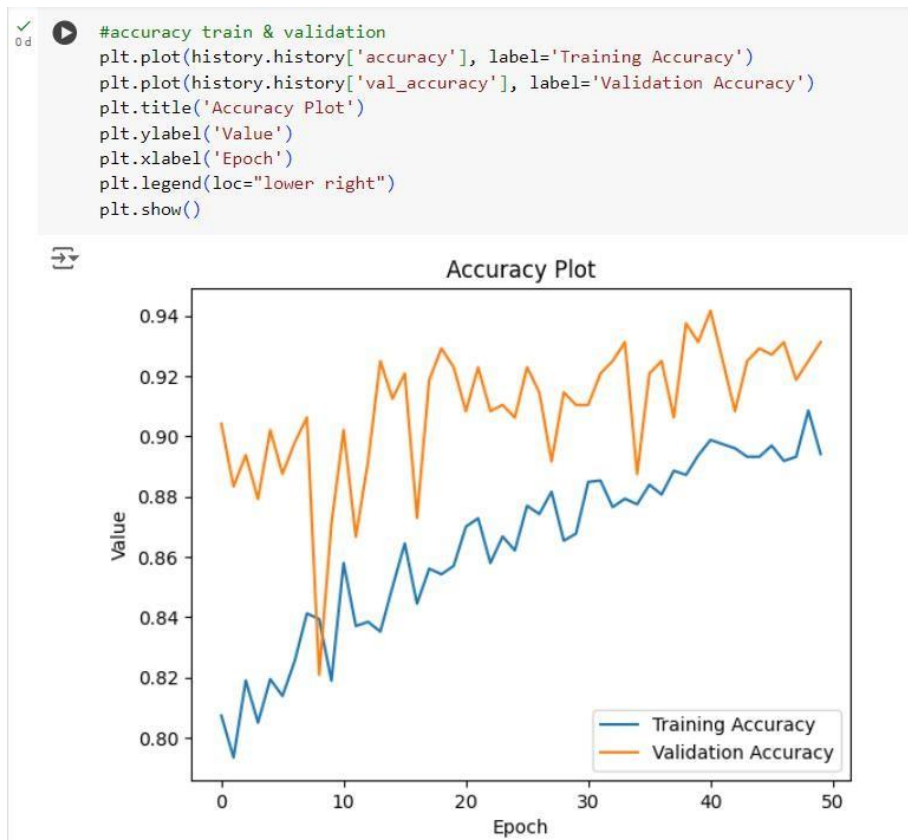
Hasil di atas menunjukkan proses pelatihan model CNN selama 50 epoch. Setiap epoch menampilkan waktu pelatihan, loss, dan akurasi untuk data pelatihan serta loss dan akurasi untuk data validasi. Pada awalnya, loss dan akurasi menunjukkan peningkatan signifikan, tetapi seiring berjalannya epoch, peningkatan ini mulai melambat, menunjukkan model mendekati titik konvergensi. Akurasi pelatihan mencapai sekitar 89%, sedangkan akurasi validasi mencapai sekitar 93%, menunjukkan model yang baik dengan performa stabil pada data yang tidak dilihat selama pelatihan.

3.3.2 Testing

```
from matplotlib import pyplot as plt #loss train & validation
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Plot')
plt.ylabel('Value')
plt.xlabel('Epoch')
plt.legend(loc="upper right")
plt.show()
```



Kode di atas menggunakan `matplotlib` untuk membuat plot yang menggambarkan perubahan loss selama pelatihan dan validasi seiring bertambahnya epoch. Dengan memplot `history.history['loss']` dan `history.history['val_loss']`, grafik ini menunjukkan bagaimana nilai loss untuk data pelatihan dan data validasi berubah dari epoch ke epoch. Plot ini diberi judul "Loss Plot" dan diberi label pada sumbu y sebagai "Value" dan sumbu x sebagai "Epoch". Legenda ditempatkan di kanan atas grafik untuk membedakan antara kurva loss pelatihan dan validasi. Grafik ini membantu dalam menganalisis apakah model mengalami overfitting atau underfitting selama pelatihan.



Kode di atas menggunakan `matplotlib` untuk membuat plot yang menggambarkan perubahan akurasi selama pelatihan dan validasi seiring bertambahnya epoch. Dengan memplot `history.history['accuracy']` dan `history.history['val_accuracy']`, grafik ini menunjukkan bagaimana nilai akurasi untuk data pelatihan dan data validasi berubah dari epoch ke epoch. Plot ini diberi judul "Accuracy Plot" dan diberi label pada sumbu y sebagai "Value" dan sumbu x sebagai "Epoch". Legenda ditempatkan di kanan bawah grafik untuk membedakan antara kurva akurasi pelatihan dan validasi. Grafik ini membantu dalam menganalisis performa model dan memahami seberapa baik model belajar dan generalisasi terhadap data baru.