CNN vs Transfer Learning untuk Klasifikasi Citra

Tugas Individu — Mesin Learning FarrelGhozy

Eksperimen perbandingan dua pendekatan:

- CNN from scratch — dataset CIFAR- (kelas)
- Transfer Learning — dataset Cats vs Dogs dengan pretrained model

---

Phase : Preprocessing & Augmentasi

- Import Library

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, Sequential
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import os
import time
import random
import zipfile
import urllib.request
import shutil
from pathlib import Path

print("TensorFlow version:", tf.__version__)
print("NumPy version:", np.__version__)

SEED = 42
tf.random.set_seed(SEED)
np.random.seed(SEED)
random.seed(SEED)

print("Library imported successfully")
```

```
WARNING: All log messages before absl::InitializeLog() is called are written
STDERR
I0000 00:00:1781151225.300117  415093 port.cc:153] oneDNN custom operations a
You may see slightly different numerical results due to floating-point round-
errors from different computation orders. To turn them off, set the environme
variable `TF_ENABLE_ONEDNN_OPTS=0`.
I0000 00:00:1781151225.332213  415093 cpu_feature_guard.cc:227] This TensorFl
binary is optimized to use available CPU instructions in performance-critical
operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
WARNING: All log messages before absl::InitializeLog() is called are written
STDERR
I0000 00:00:1781151226.138944  415093 port.cc:153] oneDNN custom operations a
You may see slightly different numerical results due to floating-point round-
errors from different computation orders. To turn them off, set the environme
variable `TF_ENABLE_ONEDNN_OPTS=0`.
TensorFlow version: 2.21.0
NumPy version: 2.4.6
Library imported successfully
```

A. Dataset : CIFAR- (CNN from Scratch)

Menggunakan kelas dari CIFAR- . Pilihan: cat ( ) vs dog ( )

In [2]:
```
# Load CIFAR-10 dari local dataset folder
import pickle

NOTEBOOK_DIR = os.getcwd()
if os.path.basename(NOTEBOOK_DIR) == 'notebook':
    NOTEBOOK_DIR = os.path.dirname(NOTEBOOK_DIR)
CIFAR_DIR = os.path.join(NOTEBOOK_DIR, 'dataset', 'cifar10')

def load_cifar10_local(data_dir=CIFAR_DIR):
    """Load CIFAR-10 dari local pickle files."""
    x_train_list, y_train_list = [], []

    # Load training batches
    for i in range(1, 6):
        batch_file = os.path.join(data_dir, f'data_batch_{i}')
        if os.path.exists(batch_file):
            with open(batch_file, 'rb') as f:
                batch = pickle.load(f, encoding='bytes')
            x_train_list.append(batch[b'data'])
            y_train_list.extend(batch[b'labels'])

    if not x_train_list:
        # Fallback: maybe only data_batch_1
        batch_file = os.path.join(data_dir, 'data_batch_1')
        if os.path.exists(batch_file):
            with open(batch_file, 'rb') as f:
                batch = pickle.load(f, encoding='bytes')
            x_train_list = [batch[b'data']]
            y_train_list = batch[b'labels']
```

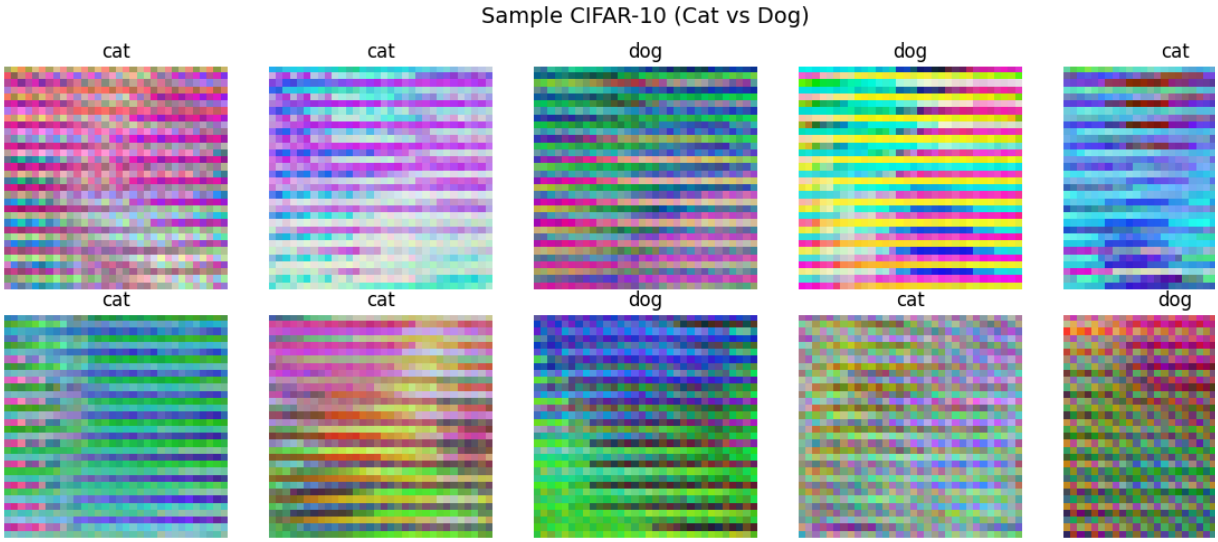
```
    # Load test batch
    test_file = os.path.join(data_dir, 'test_batch')
    if os.path.exists(test_file):
        with open(test_file, 'rb') as f:
            test_batch = pickle.load(f, encoding='bytes')
        x_test = test_batch[b'data']
        y_test = test_batch[b'labels']
    else:
        raise FileNotFoundError(f"Test batch not found in {data_dir}")

    # Concatenate training data
    x_train = np.concatenate(x_train_list)
    y_train = np.array(y_train_list)
    y_test = np.array(y_test)

    # Reshape from (N, 3072) to (N, 32, 32, 3)
    x_train = x_train.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
    x_test = x_test.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)

    return (x_train, y_train), (x_test, y_test)

print(f"Loading CIFAR-10 from {CIFAR_DIR}...")
(x_train_full, y_train_full), (x_test_full, y_test_full) = load_cifar10_l

print("CIFAR-10 shape:", x_train_full.shape, x_test_full.shape)

# Pilih 2 kelas: cat (label 3) dan dog (label 5)
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'f
selected_classes = [3, 5]  # cat, dog
selected_labels = ['cat', 'dog']

def filter_classes(x, y, classes):
    y = np.array(y)
    y = y.flatten() if y.ndim > 1 else y
    mask = np.isin(y, classes)
    x_filtered = x[mask]
    y_filtered = y[mask]
    y_binary = np.where(y_filtered == classes[0], 0, 1)
    return x_filtered, y_binary

x_train_cifar, y_train_cifar = filter_classes(x_train_full, y_train_full,
x_test_cifar, y_test_cifar = filter_classes(x_test_full, y_test_full, sel

print(f"CIFAR-10 (cat vs dog) — Train: {x_train_cifar.shape[0]}, Test: {x
print(f"Cat samples: {np.sum(y_train_cifar == 0)}, Dog samples: {np.sum(y
```

```
Loading CIFAR-10 from /home/fazy/Documents/01Semester5/Mesin Learning 2/CNN v
Transfer Learning untuk Klasifikasi Citra/dataset/cifar10...
CIFAR-10 shape: (50000, 32, 32, 3) (10000, 32, 32, 3)
CIFAR-10 (cat vs dog) — Train: 10000, Test: 2000
Cat samples: 5000, Dog samples: 5000
```

Visualisasi Sampel CIFAR-

In [3]:
```
def plot_sample_images(x, y, labels, title, rows=2, cols=5):
    fig, axes = plt.subplots(rows, cols, figsize=(12, 5))
    axes = axes.flatten()
    for i in range(rows * cols):
        idx = np.random.randint(len(x))
```

```
        axes[i].imshow(x[idx])
        axes[i].set_title(f"{labels[y[idx]]}")
        axes[i].axis('off')
    plt.suptitle(title, fontsize=14)
    plt.tight_layout()
    plt.show()

plot_sample_images(x_train_cifar, y_train_cifar, selected_labels, "Sample
```



B. Dataset : Cats vs Dogs (Transfer Learning)

In [4]:
```
# Load Cats vs Dogs dari local dataset folder
import cv2

NOTEBOOK_DIR = os.getcwd()
if os.path.basename(NOTEBOOK_DIR) == 'notebook':
    NOTEBOOK_DIR = os.path.dirname(NOTEBOOK_DIR)
CVD_DIR = os.path.join(NOTEBOOK_DIR, 'dataset', 'cats_vs_dogs')

IMG_SIZE = (128, 128)

def load_cats_vs_dogs_local(data_dir=CVD_DIR, max_per_class=200):
    """Load Cats vs Dogs images from local folder."""
    images, labels = [], []

    cat_files = sorted([f for f in os.listdir(data_dir) if f.startswith('
    dog_files = sorted([f for f in os.listdir(data_dir) if f.startswith('

    for f in cat_files[:max_per_class]:
        img_path = os.path.join(data_dir, f)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, IMG_SIZE)
            images.append(img)
            labels.append(0)

    for f in dog_files[:max_per_class]:
        img_path = os.path.join(data_dir, f)
```

```
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, IMG_SIZE)
            images.append(img)
            labels.append(1)

    return np.array(images), np.array(labels)

print(f"Loading Cats vs Dogs dari {CVD_DIR}...")
x_catsvdog, y_catsvdog = load_cats_vs_dogs_local()

print(f"Cats: {np.sum(y_catsvdog == 0)}, Dogs: {np.sum(y_catsvdog == 1)},
print(f"Image shape: {x_catsvdog[0].shape}")
```

```
Loading Cats vs Dogs dari /home/fazy/Documents/01Semester5/Mesin Learning 2/C
Transfer Learning untuk Klasifikasi Citra/dataset/cats_vs_dogs...
Cats: 199, Dogs: 200, Total: 399
Image shape: (128, 128, 3)
```

Visualisasi Sampel Cats vs Dogs

In [5]:
```
plot_sample_images(x_catsvdog, y_catsvdog, ['cat', 'dog'], "Sample Cats v
```

C. Data Splitting (% Train, % Val, % T

In [6]:
```
def split_data(x, y, train_ratio=0.7, val_ratio=0.15, random_state=SEED):
    x_temp, x_test, y_temp, y_test = train_test_split(
        x, y, test_size=val_ratio, random_state=random_state, stratify=y
    )
    val_size = val_ratio / (train_ratio + val_ratio)
    x_train, x_val, y_train, y_val = train_test_split(
        x_temp, y_temp, test_size=val_size, random_state=random_state, st
    )
    return (x_train, y_train), (x_val, y_val), (x_test, y_test)

# Split CIFAR-10
(x_train_cifar, y_train_cifar), (x_val_cifar, y_val_cifar), (x_test_cifar
```

```
    split_data(x_train_cifar, y_train_cifar)
print("=== CIFAR-10 Split ===")
print(f"Train: {len(x_train_cifar)}, Val: {len(x_val_cifar)}, Test: {len(

# Gabung train+test CIFAR-10 asli untuk dataset lebih besar
# Kita sudah split data train asli (10k) menjadi train/val/test
x_train_catsvdog = x_catsvdog
y_train_catsvdog = y_catsvdog

# Split Cats vs Dogs
(x_train_cvd, y_train_cvd), (x_val_cvd, y_val_cvd), (x_test_cvd, y_test_c
    split_data(x_catsvdog, y_catsvdog)
print("\n=== Cats vs Dogs Split ===")
print(f"Train: {len(x_train_cvd)}, Val: {len(x_val_cvd)}, Test: {len(x_te
```

```
=== CIFAR-10 Split ===
Train: 7000, Val: 1500, Test: 1500

=== Cats vs Dogs Split ===
Train: 279, Val: 60, Test: 60
```

D. Preprocessing

D. Normalisasi Pixel Values

In [7]:
```
# Normalisasi pixel [0,1]
def normalize(x):
    return x.astype('float32') / 255.0

# CIFAR-10 (32x32)
x_train_cifar_norm = normalize(x_train_cifar)
x_val_cifar_norm = normalize(x_val_cifar)
x_test_cifar_norm = normalize(x_test_cifar)

# Cats vs Dogs (128x128) tetap simpan versi non-normalized untuk visualis
x_train_cvd_norm = normalize(x_train_cvd)
x_val_cvd_norm = normalize(x_val_cvd)
x_test_cvd_norm = normalize(x_test_cvd)

print(f"CIFAR-10 — Train: {x_train_cifar_norm.shape}, Range: [{x_train_ci
print(f"Cats vs Dogs — Train: {x_train_cvd_norm.shape}, Range: [{x_train_
```

```
CIFAR-10 — Train: (7000, 32, 32, 3), Range: [0.00, 1.00]
Cats vs Dogs — Train: (279, 128, 128, 3), Range: [0.00, 1.00]
```

D. Resize untuk Transfer Learning

Pretrained model membutuhkan input minimal x (ResNet /MobileNetV). CIFAR- (x) cukup untuk CNN from scratch. Cats vs Dogs kita resize.

In [8]:
```
IMG_SIZE_TL = (224, 224)  # Input size untuk transfer learning
IMG_SIZE_CNN = (32, 32)   # Input size untuk CNN from scratch (CIFAR-10 n

# Resize dataset untuk transfer learning
```

```python
def resize_dataset(x, size):
    x_resized = np.array([
        tf.image.resize(img, size).numpy() for img in x
    ])
    return x_resized

# Note: resize dilakukan nanti di dalam pipeline transfer learning
# untuk efisiensi memori. Sementara simpan ukuran asli.
print(f"CNN input size: {IMG_SIZE_CNN}")
print(f"Transfer Learning input size: {IMG_SIZE_TL}")
```

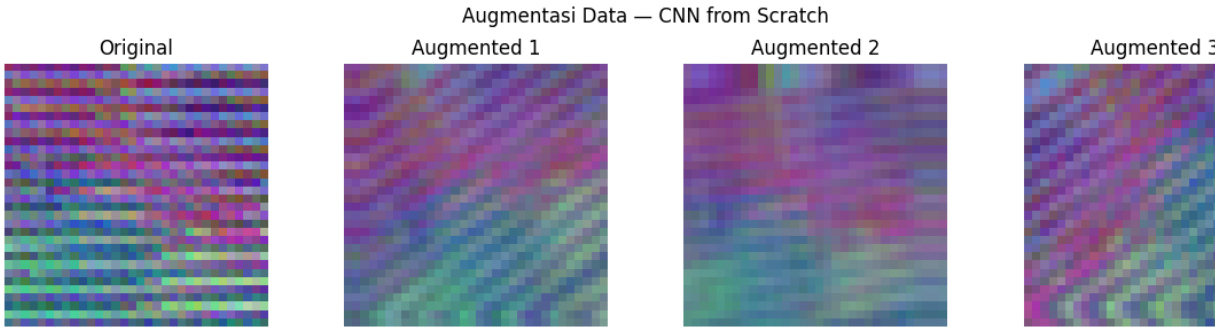
```
CNN input size: (32, 32)
Transfer Learning input size: (224, 224)
```

D. Augmentasi Data

In [9]:

```python
# Augmentasi untuk CNN from scratch
data_augmentation_cnn = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomTranslation(0.1, 0.1),
])

# Augmentasi untuk Transfer Learning
data_augmentation_tl = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.15),
    layers.RandomZoom(0.15),
    layers.RandomContrast(0.1),
])

# Demo augmentasi
sample_img = x_train_cifar_norm[0:1]
augmented_images = data_augmentation_cnn(sample_img)

fig, axes = plt.subplots(1, 4, figsize=(12, 3))
axes[0].imshow(sample_img[0])
axes[0].set_title("Original")
axes[0].axis('off')
for i in range(1, 4):
    aug = data_augmentation_cnn(sample_img)
    axes[i].imshow(aug[0])
    axes[i].set_title(f"Augmented {i}")
    axes[i].axis('off')
plt.suptitle("Augmentasi Data — CNN from Scratch")
plt.tight_layout()
plt.show()

print("Augmentasi siap digunakan saat training melalui pipeline")
```

```
I0000 00:00:1781151227.918684  415093 gpu_device.cc:2043] Created device /
job:localhost/replica:0/task:0/device:GPU:0 with 2115 MB memory:  -> device:
name: NVIDIA GeForce RTX 2050, pci bus id: 0000:01:00.0, compute capability:
```




```
Augmentasi siap digunakan saat training melalui pipeline
```

D. Konversi Label ke One-Hot Encoding

In [10]:
```
# One-hot encoding untuk categorical crossentropy
y_train_cifar_cat = keras.utils.to_categorical(y_train_cifar, 2)
y_val_cifar_cat = keras.utils.to_categorical(y_val_cifar, 2)
y_test_cifar_cat = keras.utils.to_categorical(y_test_cifar, 2)

y_train_cvd_cat = keras.utils.to_categorical(y_train_cvd, 2)
y_val_cvd_cat = keras.utils.to_categorical(y_val_cvd, 2)
y_test_cvd_cat = keras.utils.to_categorical(y_test_cvd, 2)

print("One-hot encoding selesai")
print(f"CIFAR-10 sample: {y_train_cifar[0]} -> {y_train_cifar_cat[0]}")
print(f"CVD sample: {y_train_cvd[0]} -> {y_train_cvd_cat[0]}")
```

```
One-hot encoding selesai
CIFAR-10 sample: 0 -> [1. 0.]
CVD sample: 0 -> [1. 0.]
```

Ringkasan Preprocessing

In [11]:
```
print("=" * 60)
print("RINGKASAN PREPROCESSING")
print("=" * 60)
print()
print("Dataset CIFAR-10 (CNN from scratch):")
print(f"  - Original size: 32x32x3")
print(f"  - Train: {x_train_cifar.shape[0]} images")
print(f"  - Val:   {x_val_cifar.shape[0]} images")
print(f"  - Test:  {x_test_cifar.shape[0]} images")
print(f"  - Normalization: pixel [0,1]")
print(f"  - Augmentation: Flip, Rotation, Zoom, Translation")
print()
print("Dataset Cats vs Dogs (Transfer Learning):")
print(f"  - Original size: 128x128x3")
print(f"  - Train: {x_train_cvd.shape[0]} images")
print(f"  - Val:   {x_val_cvd.shape[0]} images")
print(f"  - Test:  {x_test_cvd.shape[0]} images")
print(f"  - Normalization: pixel [0,1]")
print(f"  - Resize to: {IMG_SIZE_TL} untuk Transfer Learning")
print(f"  - Augmentation: Flip, Rotation, Zoom, Contrast")
```

```
============================================================
RINGKASAN PREPROCESSING
============================================================

Dataset CIFAR-10 (CNN from scratch):
  - Original size: 32x32x3
  - Train: 7000 images
  - Val:   1500 images
  - Test:  1500 images
  - Normalization: pixel [0,1]
  - Augmentation: Flip, Rotation, Zoom, Translation

Dataset Cats vs Dogs (Transfer Learning):
  - Original size: 128x128x3
  - Train: 279 images
  - Val:   60 images
  - Test:  60 images
  - Normalization: pixel [0,1]
  - Resize to: (224, 224) untuk Transfer Learning
  - Augmentation: Flip, Rotation, Zoom, Contrast
```

Phase : CNN from Scratch

• Arsitektur CNN

Justifikasi Arsitektur:

- Input: x x (ukuran asli CIFAR-)
- Block : Conv D filter (x) + ReLU + MaxPooling (x) — ekstraksi fitu
- Block : Conv D filter (x) + ReLU + MaxPooling (x) — ekstraksi fitu level
- Block : Conv D filter (x) + ReLU + MaxPooling (x) — ekstraksi f level
- GlobalAveragePooling D — mengurangi parameter dibanding Flatten, mencegah overfitting
- Dense + Dropout • — regularisasi
- Output: Dense + Softmax — klasifikasi biner

Alasan pemilihan:

- Filter kecil (x) efisien dan bisa menangkap pola lokal
- Jumlah filter bertambah seiring depth (→ →) karena fitur semakin kompleks
- GAP mengurangi parameter drastis dibanding Flatten
- Dropout • mencegah ko-adaptasi neuron

In [12]:
```
def build_cnn_from_scratch(input_shape=(32, 32, 3)):
    model = Sequential([
        # Block 1
        layers.Conv2D(32, (3, 3), padding='same', input_shape=input_shape
```

```
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),

        # Block 2
        layers.Conv2D(64, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),

        # Block 3
        layers.Conv2D(128, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),

        # Classifier
        layers.GlobalAveragePooling2D(),
        layers.Dense(128),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(0.5),
        layers.Dense(2, activation='softmax')
    ])
    return model

cnn_model = build_cnn_from_scratch()
cnn_model.summary()
```

```
/home/fazy/Documents/01Semester5/Mesin Learning 2/CNN vs Transfer Learning un
Klasifikasi Citra/venv/lib/python3.12/site-packages/keras/src/layers/convolut
base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argum
a layer. When using Sequential models, prefer using an `Input(shape)` object
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_2"

Layer (type)	Output Shape	Param
conv2d (Conv2D)	(None, 32, 32, 32)	89
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	12
activation (Activation)	(None, 32, 32, 32)	
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,49
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	25
activation_1 (Activation)	(None, 16, 16, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,85
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 128)	51
activation_2 (Activation)	(None, 8, 8, 128)	
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	
dense (Dense)	(None, 128)	16,51
batch_normalization_3 (BatchNormalization)	(None, 128)	51
activation_3 (Activation)	(None, 128)	
dropout (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 2)	25

Total params: 111,426 (435.26 KB)
Trainable params: 110,722 (432.51 KB)
Non-trainable params: 704 (2.75 KB)

- Compile Model

Justifikasi:

- Optimizer Adam: adaptive learning rate, cocok untuk CNN
- Learning rate : default Adam, terbukti stabil
- Loss categorical_crossentropy: untuk klasifikasi multi-class (kelas)
- Metrics accuracy: interpretable

In [13]:
```
cnn_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
print("Model compiled successfully")
```

```
Model compiled successfully
```

- ## Callbacks

EarlyStopping: hentikan training jika validation loss tidak membaik selama epoch, restore bobc

ReduceLROnPlateau: turunkan LR jika loss stagnan.

In [14]:
```
callbacks_cnn = [
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-6,
        verbose=1
    )
]
print("Callbacks ready")
```

```
Callbacks ready
```

- ## Training CNN from Scratch

Batch size: — kompromi antara kecepatan dan stabilitas gradient. Epochs: (dengan EarlyStopping).

In [15]:
```
BATCH_SIZE = 32
EPOCHS = 50

# Buat pipeline augmented
def cnn_pipeline(x, y, batch_size, augment=False):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    if augment:
        def aug_fn(images, labels):
            images = tf.expand_dims(images, 0)
            return tf.squeeze(data_augmentation_cnn(images, training=True
        dataset = dataset.map(aug_fn, num_parallel_calls=tf.data.AUTOTUNE
    dataset = dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset

train_ds_cnn = cnn_pipeline(x_train_cifar_norm, y_train_cifar_cat, BATCH_
val_ds_cnn = cnn_pipeline(x_val_cifar_norm, y_val_cifar_cat, BATCH_SIZE,
```

```
print("Training pipeline ready")
print(f"Training batches: {len(list(train_ds_cnn))}, Val batches: {len(li
```

```
Training pipeline ready
Training batches: 219, Val batches: 47
```

In [16]:

```
# Record start time
start_time = time.time()

history_cnn = cnn_model.fit(
    train_ds_cnn,
    validation_data=val_ds_cnn,
    epochs=EPOCHS,
    callbacks=callbacks_cnn,
    verbose=1
)

training_time_cnn = time.time() - start_time
print(f"\nTraining time: {training_time_cnn:.2f} seconds ({training_time_
```

```
Epoch 1/50
I0000 00:00:1781151231.687625  415271 service.cc:153] XLA service 0x3454c6c0
initialized for platform CUDA (this does not guarantee that XLA will be used)
Devices:
I0000 00:00:1781151231.687644  415271 service.cc:161]   StreamExecutor [0]: N
GeForce RTX 2050, Compute Capability 8.6 (Driver: 13.0.0; Runtime: 12.5.0; To
12.5.0; DNN: 9.23.0)
I0000 00:00:1781151231.738482  415271 dump_mlir_util.cc:269] disabling MLIR c
reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1781151232.011158  415271 cuda_dnn.cc:461] Loaded cuDNN version 9
I0000 00:00:1781151232.055585  415271 dot_merger.cc:481] Merging Dots in
computation: a_inference_one_step_on_data_5940__.68
I0000 00:00:1781151233.290916  415271 dot_search_space.cc:240] All configs we
filtered out because none of them sufficiently match the hints. Maybe the hin
does not contain a good representative set of valid configs? Working around t
using the full hints set instead.
  1/219 ━━━━━━━━━━━━━━━━━━━━ 26:21 7s/step - accuracy: 0.6250 - loss: 0.7640
 11/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5363 - loss: 0.9920
 21/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5285 - loss: 0.9873
 31/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5216 - loss: 0.9851
I0000 00:00:1781151237.621354  415271 device_compiler.h:208] Compiled cluster
XLA!  This line is logged at most once for the lifetime of the process.
 41/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5186 - loss: 0.9766
 51/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5170 - loss: 0.9683
 61/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5156 - loss: 0.9617
 72/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5143 - loss: 0.9555
 82/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5133 - loss: 0.9507
 92/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5122 - loss: 0.9465
102/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5113 - loss: 0.9424
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
112/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5105 - loss: 0.9386
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
121/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5098 - loss: 0.9354
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
131/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5092 - loss: 0.9321
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
141/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5085 - loss: 0.9290
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
151/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5080 - loss: 0.9260
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
161/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5076 - loss: 0.9231
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
171/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5073 - loss: 0.9204
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
181/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5071 - loss: 0.9177
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
191/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5070 - loss: 0.9151
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
202/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5069 - loss: 0.9124
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
212/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5069 - loss: 0.9098
I0000 00:00:1781151239.069461  415271 dot_merger.cc:481] Merging Dots in
computation: a_inference_one_step_on_data_5940__.68
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.5069 - loss: 0.9081
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
219/219 ━━━━━━━━━━━━━━━━━━━━ 16s 39ms/step - accuracy: 0.5073 - loss: 0.8553
val_accuracy: 0.4693 - val_loss: 0.7093 - learning_rate: 0.0010
Epoch 2/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 11s 52ms/step - accuracy: 0.5312 - loss: 0.7855
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[
 12/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5179 - loss: 0.7817
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 22/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5221 - loss: 0.7724
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 31/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5235 - loss: 0.7685
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 41/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5227 - loss: 0.7680
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 51/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5226 - loss: 0.7672
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 61/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5235 - loss: 0.7657
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 70/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5244 - loss: 0.7644
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 79/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5250 - loss: 0.7630
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 90/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5256 - loss: 0.7613
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
100/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5255 - loss: 0.7602
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
110/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5252 - loss: 0.7594
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
120/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5250 - loss: 0.7584
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
130/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5249 - loss: 0.7574
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
139/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5249 - loss: 0.7567
```

```
148/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5247 - loss: 0.7561
158/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5243 - loss: 0.7557
168/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5240 - loss: 0.7552
178/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5237 - loss: 0.7548
188/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5234 - loss: 0.7542
199/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5231 - loss: 0.7537
209/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5228 - loss: 0.7532
218/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5226 - loss: 0.7527
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5173 - loss: 0.7421 -
val_accuracy: 0.5547 - val_loss: 0.6879 - learning_rate: 0.0010
Epoch 3/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 11s 55ms/step - accuracy: 0.6875 - loss: 0.6250
 10/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5703 - loss: 0.6925
 19/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5627 - loss: 0.7012
 28/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5553 - loss: 0.7047
 37/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5509 - loss: 0.7066
 46/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5494 - loss: 0.7065
 55/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5489 - loss: 0.7058
 65/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5494 - loss: 0.7047
 75/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5503 - loss: 0.7037
 84/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5507 - loss: 0.7032
 93/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5511 - loss: 0.7029
103/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5511 - loss: 0.7028
113/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5508 - loss: 0.7028
124/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5504 - loss: 0.7029
134/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5503 - loss: 0.7028
144/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5500 - loss: 0.7029
155/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5496 - loss: 0.7029
166/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5491 - loss: 0.7030
176/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5487 - loss: 0.7029
185/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5485 - loss: 0.7029
```

```
194/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5483 - loss: 0.7028
203/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5481 - loss: 0.7027
213/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5478 - loss: 0.7027
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5417 - loss: 0.7005 -
val_accuracy: 0.5293 - val_loss: 0.7715 - learning_rate: 0.0010
Epoch 4/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 10s 49ms/step - accuracy: 0.7188 - loss: 0.6239
 10/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5686 - loss: 0.7011
 18/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5562 - loss: 0.7061
 27/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5542 - loss: 0.7047
 37/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5545 - loss: 0.7022
 47/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5562 - loss: 0.6998
 57/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5573 - loss: 0.6987
 67/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5584 - loss: 0.6978
 77/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5592 - loss: 0.6969
 87/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5602 - loss: 0.6961
 97/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5606 - loss: 0.6956
107/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5608 - loss: 0.6952
117/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5607 - loss: 0.6950
126/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5606 - loss: 0.6948
136/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5606 - loss: 0.6946
146/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5605 - loss: 0.6945
156/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5604 - loss: 0.6944
166/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5603 - loss: 0.6943
176/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5601 - loss: 0.6942
185/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5600 - loss: 0.6941
194/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5600 - loss: 0.6941
204/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5599 - loss: 0.6939
214/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5599 - loss: 0.6938
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5611 - loss: 0.6903 -
val_accuracy: 0.5293 - val_loss: 0.8245 - learning_rate: 0.0010
Epoch 5/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 10s 50ms/step - accuracy: 0.6562 - loss: 0.6361
```

```
  9/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5981 - loss: 0.6829
 17/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5749 - loss: 0.6919
 25/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5649 - loss: 0.6942
 32/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5611 - loss: 0.6939
 38/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5591 - loss: 0.6935
 46/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5579 - loss: 0.6928
 54/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5567 - loss: 0.6926
 63/219 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5561 - loss: 0.6924
 72/219 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5566 - loss: 0.6919
 81/219 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5573 - loss: 0.6913
 90/219 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5584 - loss: 0.6906
 99/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5592 - loss: 0.6901
109/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5602 - loss: 0.6896
119/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5607 - loss: 0.6892
129/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5613 - loss: 0.6888
139/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5615 - loss: 0.6885
149/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5616 - loss: 0.6883
159/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5616 - loss: 0.6881
169/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5617 - loss: 0.6878
179/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5620 - loss: 0.6875
189/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5623 - loss: 0.6873
199/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5624 - loss: 0.6870
208/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5625 - loss: 0.6868
218/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5626 - loss: 0.6865
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5669 - loss: 0.6809 -
val_accuracy: 0.5273 - val_loss: 0.8125 - learning_rate: 0.0010
Epoch 6/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 10s 49ms/step - accuracy: 0.6875 - loss: 0.6680
 11/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5999 - loss: 0.6707
 22/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5861 - loss: 0.6784
 32/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5792 - loss: 0.6794
```

```
 42/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5776 - loss: 0.6785
 52/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5767 - loss: 0.6781
 63/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5757 - loss: 0.6782
 73/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5751 - loss: 0.6781
 82/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5748 - loss: 0.6779
 92/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5748 - loss: 0.6775
102/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5749 - loss: 0.6772
112/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5754 - loss: 0.6769
122/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5758 - loss: 0.6766
131/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5760 - loss: 0.6765
141/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5762 - loss: 0.6764
150/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5764 - loss: 0.6763
160/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5765 - loss: 0.6762
170/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5766 - loss: 0.6761
180/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5768 - loss: 0.6761
190/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5770 - loss: 0.6760
200/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5771 - loss: 0.6759
210/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5772 - loss: 0.6758
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5783 - loss: 0.6743 -
val_accuracy: 0.5260 - val_loss: 0.8116 - learning_rate: 0.0010
Epoch 7/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 11s 53ms/step - accuracy: 0.6250 - loss: 0.6420
 11/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5833 - loss: 0.6739
 20/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5802 - loss: 0.6782
 30/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5761 - loss: 0.6786
 39/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5753 - loss: 0.6779
 49/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5756 - loss: 0.6773
 59/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5745 - loss: 0.6774
 68/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5736 - loss: 0.6773
 78/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5732 - loss: 0.6769
 88/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5732 - loss: 0.6764
```

```
□□□
```

```
159/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5793 - loss: 0.6714
170/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5794 - loss: 0.6712
180/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5796 - loss: 0.6710
190/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5797 - loss: 0.6708
200/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5799 - loss: 0.6707
211/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5801 - loss: 0.6705
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5851 - loss: 0.6668 -
val_accuracy: 0.5740 - val_loss: 0.7036 - learning_rate: 5.0000e-04
Epoch 9/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 9s 46ms/step - accuracy: 0.6562 - loss: 0.6262
 10/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5646 - loss: 0.6663
 20/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5458 - loss: 0.6775
 30/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5475 - loss: 0.6785
 40/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5535 - loss: 0.6766
 50/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5593 - loss: 0.6751
 60/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5630 - loss: 0.6744
 70/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5659 - loss: 0.6737
 80/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5690 - loss: 0.6726
 89/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5715 - loss: 0.6716
 98/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5736 - loss: 0.6707
107/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5754 - loss: 0.6700
117/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5768 - loss: 0.6695
127/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5781 - loss: 0.6690
136/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5791 - loss: 0.6687
145/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5800 - loss: 0.6683
155/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5808 - loss: 0.6681
165/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5815 - loss: 0.6678
175/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5821 - loss: 0.6676
183/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5826 - loss: 0.6674
192/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5830 - loss: 0.6673
201/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5834 - loss: 0.6671
```

```
━━━━━━━━━━━━━━━━━━━━ 211/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5837 - loss: 0.6670
━━━━━━━━━━━━━━━━━━━━ 219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5914 - loss: 0.6638 -
val_accuracy: 0.5553 - val_loss: 0.7288 - learning_rate: 5.0000e-04
Epoch 10/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 10s 47ms/step - accuracy: 0.7188 - loss: 0.6301
━━━━━━━━━━━━━━━━━━━━ 11/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6263 - loss: 0.6575
━━━━━━━━━━━━━━━━━━━━ 21/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6057 - loss: 0.6670
━━━━━━━━━━━━━━━━━━━━ 31/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5987 - loss: 0.6685
━━━━━━━━━━━━━━━━━━━━ 41/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5963 - loss: 0.6679
━━━━━━━━━━━━━━━━━━━━ 52/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5958 - loss: 0.6673
━━━━━━━━━━━━━━━━━━━━ 62/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5959 - loss: 0.6671
━━━━━━━━━━━━━━━━━━━━ 72/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5965 - loss: 0.6667
━━━━━━━━━━━━━━━━━━━━ 81/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5972 - loss: 0.6662
━━━━━━━━━━━━━━━━━━━━ 91/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5980 - loss: 0.6656
━━━━━━━━━━━━━━━━━━━━ 101/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5986 - loss: 0.6651
━━━━━━━━━━━━━━━━━━━━ 111/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5988 - loss: 0.6648
━━━━━━━━━━━━━━━━━━━━ 121/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5989 - loss: 0.6646
━━━━━━━━━━━━━━━━━━━━ 131/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5992 - loss: 0.6642
━━━━━━━━━━━━━━━━━━━━ 141/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5995 - loss: 0.6638
━━━━━━━━━━━━━━━━━━━━ 151/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5997 - loss: 0.6635
━━━━━━━━━━━━━━━━━━━━ 162/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5998 - loss: 0.6632
━━━━━━━━━━━━━━━━━━━━ 172/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6000 - loss: 0.6630
━━━━━━━━━━━━━━━━━━━━ 182/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6000 - loss: 0.6627
━━━━━━━━━━━━━━━━━━━━ 190/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6000 - loss: 0.6626
━━━━━━━━━━━━━━━━━━━━ 197/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6001 - loss: 0.6625
━━━━━━━━━━━━━━━━━━━━ 206/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6001 - loss: 0.6624
━━━━━━━━━━━━━━━━━━━━ 215/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6001 - loss: 0.6623
━━━━━━━━━━━━━━━━━━━━ 219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5993 - loss: 0.6609 -
val_accuracy: 0.5753 - val_loss: 0.7220 - learning_rate: 5.0000e-04
Epoch 11/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 10s 47ms/step - accuracy: 0.7812 - loss: 0.5944
━━━━━━━━━━━━━━━━━━━━ 11/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6429 - loss: 0.6400
━━━━━━━━━━━━━━━━━━━━ 21/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6148 - loss: 0.6574
```

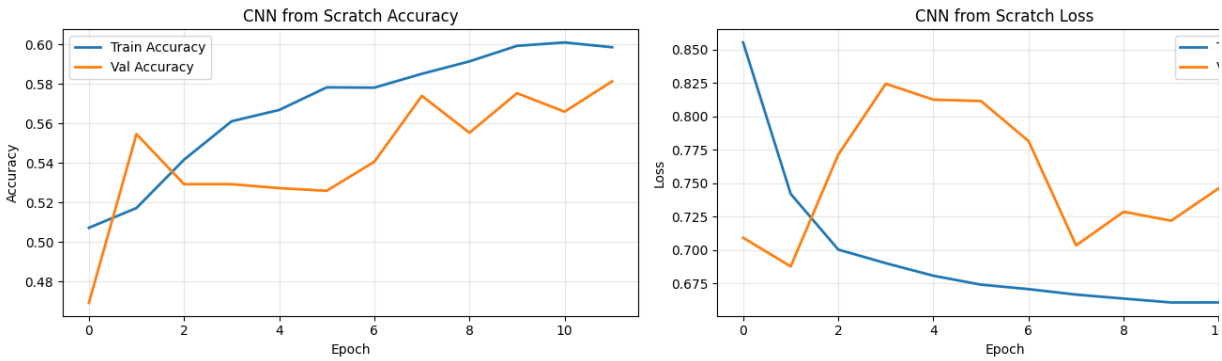
```
 31/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6036 - loss: 0.6618
 41/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6001 - loss: 0.6624
 51/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5996 - loss: 0.6626
 61/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5993 - loss: 0.6629
 71/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5993 - loss: 0.6629
 81/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5994 - loss: 0.6624
 91/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5998 - loss: 0.6618
101/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5999 - loss: 0.6615
112/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5999 - loss: 0.6614
123/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6001 - loss: 0.6612
133/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6003 - loss: 0.6610
142/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6005 - loss: 0.6609
152/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6006 - loss: 0.6609
162/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6006 - loss: 0.6609
172/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6007 - loss: 0.6609
182/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6008 - loss: 0.6608
192/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6008 - loss: 0.6608
202/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6009 - loss: 0.6608
212/219 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6009 - loss: 0.6608
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6010 - loss: 0.6610 -
val_accuracy: 0.5660 - val_loss: 0.7463 - learning_rate: 5.0000e-04
Epoch 12/50
  1/219 ━━━━━━━━━━━━━━━━━━━━ 11s 52ms/step - accuracy: 0.7188 - loss: 0.6152
 11/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6309 - loss: 0.6574
 21/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6080 - loss: 0.6672
 30/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6021 - loss: 0.6680
 39/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6000 - loss: 0.6669
 48/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5998 - loss: 0.6657
 57/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5996 - loss: 0.6655
 67/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5993 - loss: 0.6653
 77/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5995 - loss: 0.6649
```

```
□□□
```

```
========================================
CNN FROM SCRATCH - HASIL EVALUASI
========================================
Train Accuracy: 0.5986
Train Loss:     0.6607
Val Accuracy:   0.5547
Val Loss:       0.6879
Test Accuracy:  0.5393
Test Loss:      0.6981
Training Time:  30.61s
Total Params:   111,426
```

- Grafik Training CNN from Scratch

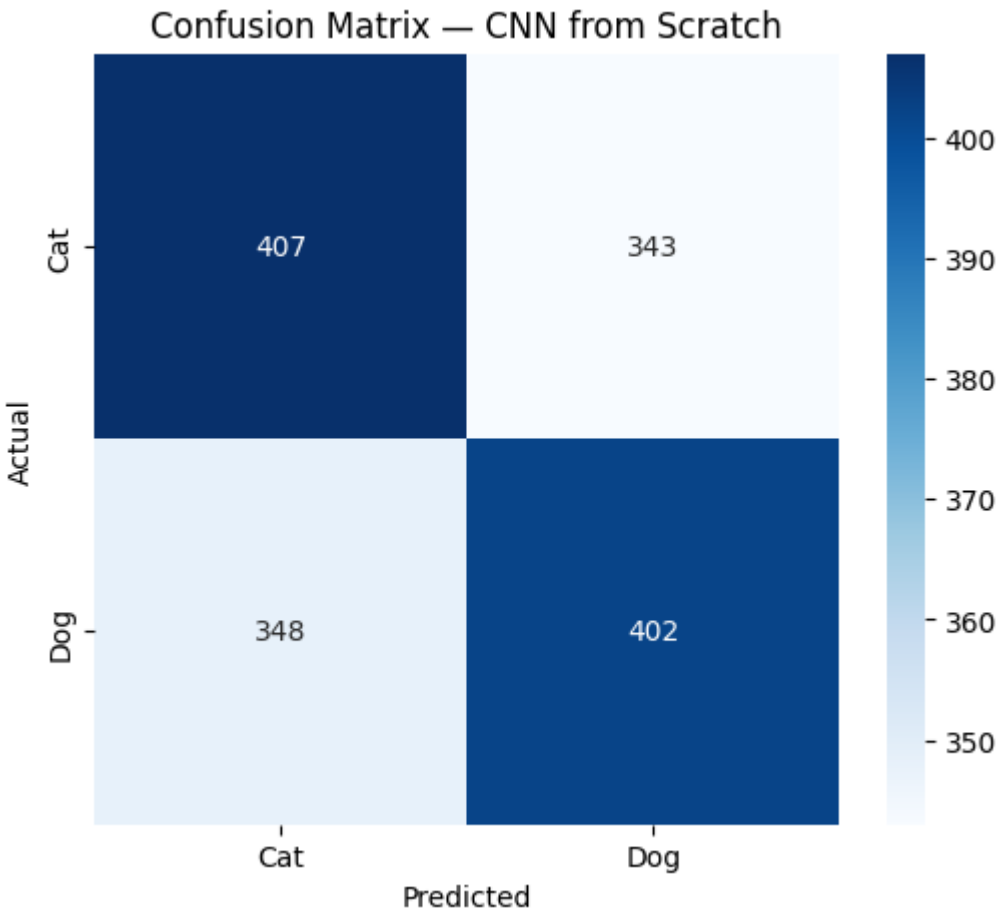
In [18]:
```
def plot_training_history(history, title_prefix=""):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 4))

    # Accuracy plot
    ax1.plot(history.history['accuracy'], label='Train Accuracy', linewid
    ax1.plot(history.history['val_accuracy'], label='Val Accuracy', linew
    ax1.set_title(f'{title_prefix} Accuracy')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Loss plot
    ax2.plot(history.history['loss'], label='Train Loss', linewidth=2)
    ax2.plot(history.history['val_loss'], label='Val Loss', linewidth=2)
    ax2.set_title(f'{title_prefix} Loss')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

plot_training_history(history_cnn, "CNN from Scratch")
```



- Confusion Matrix CNN from Scratch

In [19]:
```
# Prediksi
y_pred_cnn = cnn_model.predict(x_test_cifar_norm, verbose=0)
y_pred_cnn_classes = np.argmax(y_pred_cnn, axis=1)
```

```
# Confusion Matrix
cm_cnn = confusion_matrix(y_test_cifar, y_pred_cnn_classes)

plt.figure(figsize=(6, 5))
sns.heatmap(cm_cnn, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Cat', 'Dog'], yticklabels=['Cat', 'Dog'])
plt.title('Confusion Matrix — CNN from Scratch')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("\nClassification Report:")
print(classification_report(y_test_cifar, y_pred_cnn_classes,
                            target_names=['Cat', 'Dog']))
```

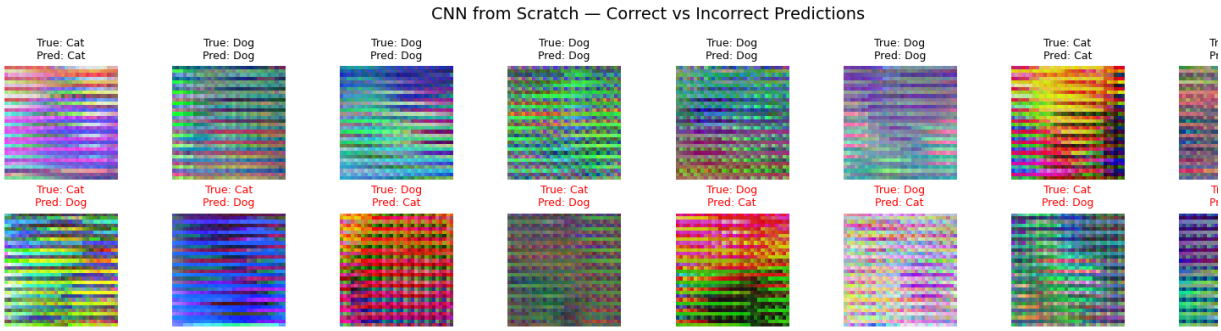



```
Classification Report:
              precision    recall  f1-score   support

         Cat       0.54      0.54      0.54       750
         Dog       0.54      0.54      0.54       750

    accuracy                           0.54      1500
   macro avg       0.54      0.54      0.54      1500
weighted avg       0.54      0.54      0.54      1500
```

- Contoh Prediksi Benar & Salah (CNN from Scratch)

```
In [20]: def plot_predictions(x_test, y_true, y_pred, class_names, title, num_samp
             correct = np.where(y_true == y_pred)[0]
             incorrect = np.where(y_true != y_pred)[0]
```

```
    # Ambil sample
    correct_samples = np.random.choice(correct, min(num_samples, len(corr
    incorrect_samples = np.random.choice(incorrect, min(num_samples, len(

    fig, axes = plt.subplots(2, num_samples, figsize=(16, 4))

    for i, idx in enumerate(correct_samples):
        axes[0, i].imshow(x_test[idx])
        axes[0, i].set_title(f"True: {class_names[y_true[idx]]}\nPred: {c
        axes[0, i].axis('off')
    axes[0, 0].set_ylabel("Correct", fontsize=12)

    for i, idx in enumerate(incorrect_samples):
        axes[1, i].imshow(x_test[idx])
        axes[1, i].set_title(f"True: {class_names[y_true[idx]]}\nPred: {c
        axes[1, i].axis('off')
    axes[1, 0].set_ylabel("Incorrect", fontsize=12)

    plt.suptitle(title, fontsize=14)
    plt.tight_layout()
    plt.show()

plot_predictions(x_test_cifar, y_test_cifar, y_pred_cnn_classes,
                 ['Cat', 'Dog'], "CNN from Scratch — Correct vs Incorrect
```



Phase : Transfer Learning

• Pemilihan Pretrained Model

Model: MobileNetV

Alasan pemilihan:

- Arsitektur ringan (~ . M parameter) cocok untuk dataset kecil
- Menggunakan depthwise separable convolution — efisien secara komputasi
- Pretrained pada ImageNet (classes) sehingga memiliki representasi visual umum yang
- Input size x — resolusi standar untuk transfer learning
- Lebih cepat training dibanding ResNet atau VGG
- Performa baik pada dataset dengan domain natural images (Cats vs Dogs)

Strategi: Feature Extraction + Fine-tuning

- Feature extraction: freeze semua base layer, train classifier baru
- Fine-tuning: unfreeze % layer teratas, train dengan LR kecil

In [21]:
```
IMG_SIZE_TL = (224, 224)

# Load MobileNetV2 tanpa top layer
base_model = keras.applications.MobileNetV2(
    weights='imagenet',
    include_top=False,
    input_shape=(*IMG_SIZE_TL, 3)
)
base_model.trainable = False  # Freeze untuk feature extraction

print(f"Base model loaded. Layers: {len(base_model.layers)}")
print(f"Trainable params before: {sum([w.shape.num_elements() for w in ba
```

```
Base model loaded. Layers: 154
Trainable params before: 0
```

- Resize Dataset untuk Transfer Learning

In [22]:
```
# Resize datasets untuk MobileNetV2 (224x224)
def resize_and_preprocess(x, size=IMG_SIZE_TL):
    return tf.image.resize(x, size).numpy()

# Preprocessing sesuai MobileNetV2
def preprocess_input_tl(x):
    return keras.applications.mobilenet_v2.preprocess_input(x * 255.0)

x_train_cvd_resized = np.array([resize_and_preprocess(img) for img in x_t
x_val_cvd_resized = np.array([resize_and_preprocess(img) for img in x_val
x_test_cvd_resized = np.array([resize_and_preprocess(img) for img in x_te

print(f"Resized shapes — Train: {x_train_cvd_resized.shape}, Val: {x_val_
```

```
Resized shapes — Train: (279, 224, 224, 3), Val: (60, 224, 224, 3), Test: (60
224, 3)
```

- Feature Extraction: Build Model

In [23]:
```
def build_transfer_learning_model(base_model, trainable_base=False):
    base_model.trainable = trainable_base

    model = Sequential([
        # Input layer with MobileNetV2 preprocessing
        layers.Input(shape=(*IMG_SIZE_TL, 3)),
        layers.Lambda(lambda x: keras.applications.mobilenet_v2.preproces

        # Augmentasi
        data_augmentation_tl,

        # Base pretrained model
        base_model,
```

```
        # Classifier head
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(2, activation='softmax')
    ])

    return model

# Model untuk feature extraction
tl_model_fe = build_transfer_learning_model(base_model, trainable_base=Fa

tl_model_fe.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

print("Feature Extraction model ready")
tl_model_fe.summary()
print(f"\nTrainable params: {sum([w.shape.num_elements() for w in tl_mode
```

```
Feature Extraction model ready
```

Model: "sequential_3"

Layer (type)	Output Shape	Param
lambda (Lambda)	(None, 224, 224, 3)	
sequential_1 (Sequential)	(None, 224, 224, 3)	
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,98
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1280)	
dense_2 (Dense)	(None, 128)	163,96
dropout_1 (Dropout)	(None, 128)	
dense_3 (Dense)	(None, 2)	25

Total params: 2,422,210 (9.24 MB)

Trainable params: 164,226 (641.51 KB)

Non-trainable params: 2,257,984 (8.61 MB)

```
Trainable params: 164,226
```

- Training — Feature Extraction

In [24]:
```
BATCH_SIZE_TL = 16  # Lebih kecil karena gambar 224x224 > 32x32

callbacks_tl = [
    keras.callbacks.EarlyStopping(
        monitor='val_loss', patience=10, restore_best_weights=True, verbo
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6, verbose=
```

```
    )
]

# Pipeline
def tl_pipeline(x, y, batch_size, augment=False):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    if augment:
        def aug_fn(images, labels):
            images = tf.expand_dims(images, 0)
            return tf.squeeze(data_augmentation_tl(images, training=True)
        dataset = dataset.map(aug_fn, num_parallel_calls=tf.data.AUTOTUNE
    dataset = dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset

train_ds_tl = tl_pipeline(x_train_cvd_resized, y_train_cvd_cat, BATCH_SIZ
val_ds_tl = tl_pipeline(x_val_cvd_resized, y_val_cvd_cat, BATCH_SIZE_TL,

print("Training pipeline ready")
```

```
Training pipeline ready
```

In [25]:

```
# Train feature extraction
start_time_fe = time.time()

history_tl_fe = tl_model_fe.fit(
    train_ds_tl,
    validation_data=val_ds_tl,
    epochs=30,
    callbacks=callbacks_tl,
    verbose=1
)

training_time_tl_fe = time.time() - start_time_fe
print(f"\nFeature Extraction training time: {training_time_tl_fe:.2f}s")
```

```
Epoch 1/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 1:05 4s/step - accuracy: 0.6875 - loss: 0.6331
□□□
```

```
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 114ms/step - accuracy: 0.6774 - loss: 0.7003 -
val_accuracy: 0.9000 - val_loss: 0.2270 - learning_rate: 0.0010
Epoch 2/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 1s 110ms/step - accuracy: 0.7500 - loss: 0.6434
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.7500 - loss: 0.5908
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.7639 - loss: 0.5477
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.7682 - loss: 0.5208
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.7696 - loss: 0.5140
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.7708 - loss: 0.4984
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.7711 - loss: 0.4938
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.7726 - loss: 0.4874
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.7722 - loss: 0.4861
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.7728 - loss: 0.4840
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.7743 - loss: 0.4808
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.7732 - loss: 0.4815
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.7728 - loss: 0.4808
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.7717 - loss: 0.4808
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.7599 - loss: 0.4872 -
val_accuracy: 0.9833 - val_loss: 0.0944 - learning_rate: 0.0010
Epoch 3/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 1s 114ms/step - accuracy: 0.7500 - loss: 0.3526
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.7969 - loss: 0.3221
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.8229 - loss: 0.3075
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8320 - loss: 0.3148
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.8381 - loss: 0.3170
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.8373 - loss: 0.3289
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.8363 - loss: 0.3388
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.8382 - loss: 0.3414
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8410 - loss: 0.3445
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8420 - loss: 0.3469
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8426 - loss: 0.3490
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8425 - loss: 0.3521
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8417 - loss: 0.3575
```

```
□□□
```

```
□□□
```

```
━━━━━━━━━━━━━━━━━━━━
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9124 - loss: 0.1962
━━━━━━━━━━━━━━━━━━━━
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9104 - loss: 0.1985
━━━━━━━━━━━━━━━━━━━━
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9092 - loss: 0.2000
━━━━━━━━━━━━━━━━━━━━
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9051 - loss: 0.2048
━━━━━━━━━━━━━━━━━━━━
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.8602 - loss: 0.2679 -
val_accuracy: 0.9833 - val_loss: 0.0826 - learning_rate: 0.0010
Epoch 8/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 140ms/step - accuracy: 0.8750 - loss: 0.2626
━━━━━━━━━━━━━━━━━━━━
 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.8906 - loss: 0.2267
━━━━━━━━━━━━━━━━━━━━
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9062 - loss: 0.2117
━━━━━━━━━━━━━━━━━━━━
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9102 - loss: 0.2099
━━━━━━━━━━━━━━━━━━━━
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9210 - loss: 0.2030
━━━━━━━━━━━━━━━━━━━━
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9234 - loss: 0.2038
━━━━━━━━━━━━━━━━━━━━
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9261 - loss: 0.2033
━━━━━━━━━━━━━━━━━━━━
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9290 - loss: 0.2047
━━━━━━━━━━━━━━━━━━━━
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9303 - loss: 0.2049
━━━━━━━━━━━━━━━━━━━━
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9309 - loss: 0.2069
━━━━━━━━━━━━━━━━━━━━
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9296 - loss: 0.2108
━━━━━━━━━━━━━━━━━━━━
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9272 - loss: 0.2160
━━━━━━━━━━━━━━━━━━━━
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9253 - loss: 0.2208
━━━━━━━━━━━━━━━━━━━━
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.9104 - loss: 0.2636 -
val_accuracy: 0.9833 - val_loss: 0.0628 - learning_rate: 0.0010
Epoch 9/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.8750 - loss: 0.2758
━━━━━━━━━━━━━━━━━━━━
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9062 - loss: 0.2401
━━━━━━━━━━━━━━━━━━━━
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9097 - loss: 0.2343
━━━━━━━━━━━━━━━━━━━━
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.9049 - loss: 0.2454
━━━━━━━━━━━━━━━━━━━━
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9015 - loss: 0.2527
━━━━━━━━━━━━━━━━━━━━
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9023 - loss: 0.2541
━━━━━━━━━━━━━━━━━━━━
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9022 - loss: 0.2573
━━━━━━━━━━━━━━━━━━━━
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9027 - loss: 0.2577
━━━━━━━━━━━━━━━━━━━━
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9012 - loss: 0.2603
━━━━━━━━━━━━━━━━━━━━
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9010 - loss: 0.2601
```

```
━━━━━━━━━━━━━━━━━━━━ 11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9002 - loss: 0.2602
━━━━━━━━━━━━━━━━━━━━ 13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8990 - loss: 0.2597
━━━━━━━━━━━━━━━━━━━━ 14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8986 - loss: 0.2593
━━━━━━━━━━━━━━━━━━━━ 16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8964 - loss: 0.2623
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.8936 - loss: 0.2661
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.8674 - loss: 0.3014 -
val_accuracy: 0.9833 - val_loss: 0.0760 - learning_rate: 0.0010
Epoch 10/30
━━━━━━━━━━━━━━━━━━━━ 1/18 ━━━━━━━━━━━━━━━━━━━━ 1s 111ms/step - accuracy: 0.8125 - loss: 0.2798
━━━━━━━━━━━━━━━━━━━━ 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.8281 - loss: 0.2530
━━━━━━━━━━━━━━━━━━━━ 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.8438 - loss: 0.2445
━━━━━━━━━━━━━━━━━━━━ 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.8555 - loss: 0.2377
━━━━━━━━━━━━━━━━━━━━ 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.8669 - loss: 0.2286
━━━━━━━━━━━━━━━━━━━━ 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8752 - loss: 0.2210
━━━━━━━━━━━━━━━━━━━━ 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8815 - loss: 0.2146
━━━━━━━━━━━━━━━━━━━━ 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8856 - loss: 0.2097
━━━━━━━━━━━━━━━━━━━━ 10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8914 - loss: 0.2039
━━━━━━━━━━━━━━━━━━━━ 11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8935 - loss: 0.2025
━━━━━━━━━━━━━━━━━━━━ 12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8950 - loss: 0.2020
━━━━━━━━━━━━━━━━━━━━ 13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8950 - loss: 0.2044
━━━━━━━━━━━━━━━━━━━━ 14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8948 - loss: 0.2068
━━━━━━━━━━━━━━━━━━━━ 15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8951 - loss: 0.2084
━━━━━━━━━━━━━━━━━━━━ 17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8939 - loss: 0.2131
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.8781 - loss: 0.2632 -
val_accuracy: 0.9833 - val_loss: 0.0613 - learning_rate: 0.0010
Epoch 11/30
━━━━━━━━━━━━━━━━━━━━ 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.9375 - loss: 0.1977
━━━━━━━━━━━━━━━━━━━━ 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9531 - loss: 0.1763
━━━━━━━━━━━━━━━━━━━━ 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9549 - loss: 0.1701
━━━━━━━━━━━━━━━━━━━━ 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9544 - loss: 0.1749
━━━━━━━━━━━━━━━━━━━━ 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9510 - loss: 0.1789
━━━━━━━━━━━━━━━━━━━━ 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9470 - loss: 0.1878
━━━━━━━━━━━━━━━━━━━━ 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9431 - loss: 0.1954
```

```
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9415 - loss: 0.1987
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9395 - loss: 0.2015
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9386 - loss: 0.2020
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9380 - loss: 0.2016
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9362 - loss: 0.2023
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9341 - loss: 0.2035
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9325 - loss: 0.2043
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9291 - loss: 0.2068
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.9250 - loss: 0.2115
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.8925 - loss: 0.2525 -
val_accuracy: 0.9667 - val_loss: 0.0688 - learning_rate: 0.0010
Epoch 12/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 1s 110ms/step - accuracy: 0.9375 - loss: 0.2025
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9375 - loss: 0.1846
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9284 - loss: 0.1893
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9252 - loss: 0.1934
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9220 - loss: 0.1989
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9192 - loss: 0.2069
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9175 - loss: 0.2113
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9169 - loss: 0.2172
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9167 - loss: 0.2210
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9164 - loss: 0.2221
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9167 - loss: 0.2224
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9164 - loss: 0.2233
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9160 - loss: 0.2247
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.9068 - loss: 0.2405 -
val_accuracy: 0.9667 - val_loss: 0.0666 - learning_rate: 0.0010
Epoch 13/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 1s 112ms/step - accuracy: 0.9375 - loss: 0.1480
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9219 - loss: 0.1832
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9201 - loss: 0.1852
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9167 - loss: 0.1879
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9133 - loss: 0.1925
```

```
□□□
```

```
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.8594 - loss: 0.3674
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8646 - loss: 0.3370
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8711 - loss: 0.3148
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8769 - loss: 0.2972
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8818 - loss: 0.2819
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8834 - loss: 0.2742
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8862 - loss: 0.2659
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8888 - loss: 0.2580
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8928 - loss: 0.2456
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8939 - loss: 0.2420
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8945 - loss: 0.2376
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.8946 - loss: 0.2350
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.8948 - loss: 0.2342
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.8961 - loss: 0.2293 -
val_accuracy: 0.9833 - val_loss: 0.0597 - learning_rate: 0.0010
Epoch 16/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 118ms/step - accuracy: 1.0000 - loss: 0.0503
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9531 - loss: 0.1290
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9375 - loss: 0.1609
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9325 - loss: 0.1660
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9299 - loss: 0.1694
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9297 - loss: 0.1704
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9307 - loss: 0.1695
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9314 - loss: 0.1686
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9320 - loss: 0.1671
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9325 - loss: 0.1658
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9325 - loss: 0.1654
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9329 - loss: 0.1645
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9332 - loss: 0.1635
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9338 - loss: 0.1623
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9335 - loss: 0.1624
```

```
━━━
```

```
□□□
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9766 - loss: 0.1151
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9751 - loss: 0.1165
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9727 - loss: 0.1190
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9678 - loss: 0.1243
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9625 - loss: 0.1323
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.9176 - loss: 0.1987 -
val_accuracy: 0.9833 - val_loss: 0.0735 - learning_rate: 0.0010
Epoch 21/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 131ms/step - accuracy: 1.0000 - loss: 0.0450
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 1.0000 - loss: 0.0541
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9861 - loss: 0.0680
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9585 - loss: 0.1044
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9481 - loss: 0.1194
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9392 - loss: 0.1342
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9367 - loss: 0.1378
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9349 - loss: 0.1403
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9336 - loss: 0.1430
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9292 - loss: 0.1488
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9275 - loss: 0.1512
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9265 - loss: 0.1528
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9238 - loss: 0.1576
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.00050000000237487257.
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.9068 - loss: 0.1938 -
val_accuracy: 0.9833 - val_loss: 0.0719 - learning_rate: 0.0010
Epoch 22/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 1.0000 - loss: 0.1151
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 2/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 1.0000 - loss: 0.1042
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 3/18 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9931 - loss: 0.1051
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9870 - loss: 0.1101
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9846 - loss: 0.1110
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9837 - loss: 0.1112
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9822 - loss: 0.1131
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9815 - loss: 0.1132
```

```
□□□
```

```
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9837 - loss: 0.0943
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9796 - loss: 0.1037
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9763 - loss: 0.1092
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9736 - loss: 0.1127
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9718 - loss: 0.1147
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9708 - loss: 0.1158
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9697 - loss: 0.1169
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9684 - loss: 0.1184
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9674 - loss: 0.1193
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9655 - loss: 0.1210
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9639 - loss: 0.1220
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.9498 - loss: 0.1317 -
val_accuracy: 0.9833 - val_loss: 0.0654 - learning_rate: 5.0000e-04
Epoch 25/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 1.0000 - loss: 0.0675
 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 1.0000 - loss: 0.0576
 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 1.0000 - loss: 0.0558
 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 1.0000 - loss: 0.0571
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 1.0000 - loss: 0.0582
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9974 - loss: 0.0618
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9924 - loss: 0.0698
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.9901 - loss: 0.0743
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9879 - loss: 0.0778
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9863 - loss: 0.0807
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9847 - loss: 0.0834
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9830 - loss: 0.0859
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.9810 - loss: 0.0883
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.9764 - loss: 0.0958
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.9391 - loss: 0.1517 -
val_accuracy: 0.9833 - val_loss: 0.0764 - learning_rate: 5.0000e-04
Epoch 26/30
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.9375 - loss: 0.2595
 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.9375 - loss: 0.2239
```

```
━━━━━━━━━━━━━━━━━━━━ 3/18 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.9444 - loss: 0.1998
━━━━━━━━━━━━━━━━━━━━ 4/18 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.9466 - loss: 0.1905
━━━━━━━━━━━━━━━━━━━━ 5/18 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9473 - loss: 0.1836
━━━━━━━━━━━━━━━━━━━━ 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9474 - loss: 0.1784
━━━━━━━━━━━━━━━━━━━━ 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9473 - loss: 0.1739
━━━━━━━━━━━━━━━━━━━━ 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9480 - loss: 0.1687
━━━━━━━━━━━━━━━━━━━━ 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.9484 - loss: 0.1641
━━━━━━━━━━━━━━━━━━━━ 10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.9485 - loss: 0.1616
━━━━━━━━━━━━━━━━━━━━ 11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9491 - loss: 0.1590
━━━━━━━━━━━━━━━━━━━━ 12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9481 - loss: 0.1591
━━━━━━━━━━━━━━━━━━━━ 13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9473 - loss: 0.1590
━━━━━━━━━━━━━━━━━━━━ 14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9463 - loss: 0.1590
━━━━━━━━━━━━━━━━━━━━ 15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9454 - loss: 0.1590
━━━━━━━━━━━━━━━━━━━━ 16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9447 - loss: 0.1590
━━━━━━━━━━━━━━━━━━━━ 17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9440 - loss: 0.1588
Epoch 26: ReduceLROnPlateau reducing learning rate to 0.00025000001187436 28.
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.9283 - loss: 0.1658 -
val_accuracy: 0.9833 - val_loss: 0.0858 - learning_rate: 5.0000e-04
Epoch 26: early stopping
Restoring model weights from the end of the best epoch: 16.
Feature Extraction training time: 36.11s
```

- Evaluasi Feature Extraction

In [26]:
```
test_ds_tl = tl_pipeline(x_test_cvd_resized, y_test_cvd_cat, BATCH_SIZE_T
test_loss_fe, test_acc_fe = tl_model_fe.evaluate(test_ds_tl, verbose=0)

print("=" * 40)
print("TRANSFER LEARNING — FEATURE EXTRACTION")
print("=" * 40)
print(f"Test Accuracy: {test_acc_fe:.4f}")
print(f"Test Loss:     {test_loss_fe:.4f}")
print(f"Training Time: {training_time_tl_fe:.2f}s")
```

```
========================================
TRANSFER LEARNING — FEATURE EXTRACTION
========================================
Test Accuracy: 0.9500
Test Loss:     0.1877
Training Time: 36.11s
```

- Fine-tuning

Unfreeze % layer teratas MobileNetV , train dengan learning rate kecil.

In [27]:
```
# Unfreeze sebagian layer
base_model.trainable = True

# Freeze layer awal (sekitar 70% layer pertama)
fine_tune_at = int(len(base_model.layers) * 0.7)
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

print(f"Fine-tuning {len(base_model.layers) - fine_tune_at} of {len(base_

# Re-compile dengan LR kecil
tl_model_fe.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
print("Fine-tuning model ready")
```

```
Fine-tuning 47 of 154 layers
Fine-tuning model ready
```

In [28]:
```
# Train fine-tuning
start_time_ft = time.time()

history_tl_ft = tl_model_fe.fit(
    train_ds_tl,
    validation_data=val_ds_tl,
    epochs=20,
    callbacks=callbacks_tl,
    verbose=1
)

training_time_tl_ft = time.time() - start_time_ft
print(f"\nFine-tuning training time: {training_time_tl_ft:.2f}s")
```

```
Epoch 1/20
 1/18 ━━━━━━━━━━━━━━━━━━━━ 3:10 11s/step - accuracy: 0.8125 - loss: 0.4220
□□□
```

```
□□□
```

```
 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.8594 - loss: 0.2641
 3/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.8507 - loss: 0.2994
 4/18 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.8372 - loss: 0.3223
 5/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.8323 - loss: 0.3300
 6/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.8307 - loss: 0.3358
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8294 - loss: 0.3430
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8292 - loss: 0.3476
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.8305 - loss: 0.3486
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8287 - loss: 0.3533
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8272 - loss: 0.3564
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8260 - loss: 0.3584
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8253 - loss: 0.3597
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8254 - loss: 0.3602
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.8256 - loss: 0.3612
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.8255 - loss: 0.3623
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.8258 - loss: 0.3629
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - accuracy: 0.8244 - loss: 0.3899 -
val_accuracy: 0.9667 - val_loss: 0.0592 - learning_rate: 1.0000e-05
Epoch 4/20
 1/18 ━━━━━━━━━━━━━━━━━━━━ 3s 201ms/step - accuracy: 1.0000 - loss: 0.1234
 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 1.0000 - loss: 0.1215
 3/18 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.9722 - loss: 0.1539
 4/18 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.9596 - loss: 0.1684
 5/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.9477 - loss: 0.1783
 6/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.9391 - loss: 0.1844
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.9325 - loss: 0.1889
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.9272 - loss: 0.1918
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9237 - loss: 0.1932
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9189 - loss: 0.2020
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9154 - loss: 0.2082
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9133 - loss: 0.2122
```

```
━━━━━━━━━━━━━━━━━━━━ 13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9100 - loss: 0.2176
━━━━━━━━━━━━━━━━━━━━ 14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9069 - loss: 0.2222
━━━━━━━━━━━━━━━━━━━━ 15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9045 - loss: 0.2259
━━━━━━━━━━━━━━━━━━━━ 16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9026 - loss: 0.2293
━━━━━━━━━━━━━━━━━━━━ 17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9014 - loss: 0.2320
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 91ms/step - accuracy: 0.8710 - loss: 0.2887 -
val_accuracy: 0.9667 - val_loss: 0.0630 - learning_rate: 1.0000e-05
Epoch 5/20
━━━━━━━━━━━━━━━━━━━━ 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 143ms/step - accuracy: 0.9375 - loss: 0.1664
━━━━━━━━━━━━━━━━━━━━ 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.9062 - loss: 0.1850
━━━━━━━━━━━━━━━━━━━━ 3/18 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 0.8889 - loss: 0.2043
━━━━━━━━━━━━━━━━━━━━ 4/18 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.8854 - loss: 0.2087
━━━━━━━━━━━━━━━━━━━━ 5/18 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.8883 - loss: 0.2066
━━━━━━━━━━━━━━━━━━━━ 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.8896 - loss: 0.2098
━━━━━━━━━━━━━━━━━━━━ 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.8901 - loss: 0.2116
━━━━━━━━━━━━━━━━━━━━ 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.8911 - loss: 0.2117
━━━━━━━━━━━━━━━━━━━━ 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.8909 - loss: 0.2126
━━━━━━━━━━━━━━━━━━━━ 10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.8886 - loss: 0.2176
━━━━━━━━━━━━━━━━━━━━ 11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.8869 - loss: 0.2209
━━━━━━━━━━━━━━━━━━━━ 12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.8859 - loss: 0.2238
━━━━━━━━━━━━━━━━━━━━ 13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.8851 - loss: 0.2261
━━━━━━━━━━━━━━━━━━━━ 14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.8843 - loss: 0.2283
━━━━━━━━━━━━━━━━━━━━ 15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.8837 - loss: 0.2301
━━━━━━━━━━━━━━━━━━━━ 16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.8827 - loss: 0.2325
━━━━━━━━━━━━━━━━━━━━ 17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.8820 - loss: 0.2346
Epoch 5: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.
━━━━━━━━━━━━━━━━━━━━ 18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 90ms/step - accuracy: 0.8602 - loss: 0.2812 -
val_accuracy: 0.9667 - val_loss: 0.0645 - learning_rate: 1.0000e-05
Epoch 6/20
━━━━━━━━━━━━━━━━━━━━ 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 144ms/step - accuracy: 0.9375 - loss: 0.1464
━━━━━━━━━━━━━━━━━━━━ 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 0.9375 - loss: 0.1481
━━━━━━━━━━━━━━━━━━━━ 3/18 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.9375 - loss: 0.1828
━━━━━━━━━━━━━━━━━━━━ 4/18 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.9336 - loss: 0.2128
```

```
 5/18 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.9344 - loss: 0.2224
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9366 - loss: 0.2240
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9380 - loss: 0.2245
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9399 - loss: 0.2230
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9412 - loss: 0.2218
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9402 - loss: 0.2234
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9389 - loss: 0.2244
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9379 - loss: 0.2245
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9361 - loss: 0.2253
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9342 - loss: 0.2264
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9322 - loss: 0.2279
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9301 - loss: 0.2301
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.9282 - loss: 0.2323
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 91ms/step - accuracy: 0.8889 - loss: 0.2913 -
val_accuracy: 0.9667 - val_loss: 0.0661 - learning_rate: 5.0000e-06
Epoch 7/20
 1/18 ━━━━━━━━━━━━━━━━━━━━ 2s 138ms/step - accuracy: 1.0000 - loss: 0.1487
 2/18 ━━━━━━━━━━━━━━━━━━━━ 1s 87ms/step - accuracy: 0.9688 - loss: 0.1753
 3/18 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 0.9583 - loss: 0.1869
 4/18 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.9453 - loss: 0.2010
 5/18 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.9413 - loss: 0.2023
 6/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9389 - loss: 0.2027
 7/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9349 - loss: 0.2077
 8/18 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.9332 - loss: 0.2101
 9/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9329 - loss: 0.2104
10/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9315 - loss: 0.2131
11/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9300 - loss: 0.2158
12/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9289 - loss: 0.2173
13/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9273 - loss: 0.2203
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.9258 - loss: 0.2228
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.9247 - loss: 0.2248
```

```
□□□
```

```
□□□
```

```
Restoring model weights from the end of the best epoch: 1.
Fine-tuning training time: 31.11s
```

- ## Evaluasi Fine-tuning

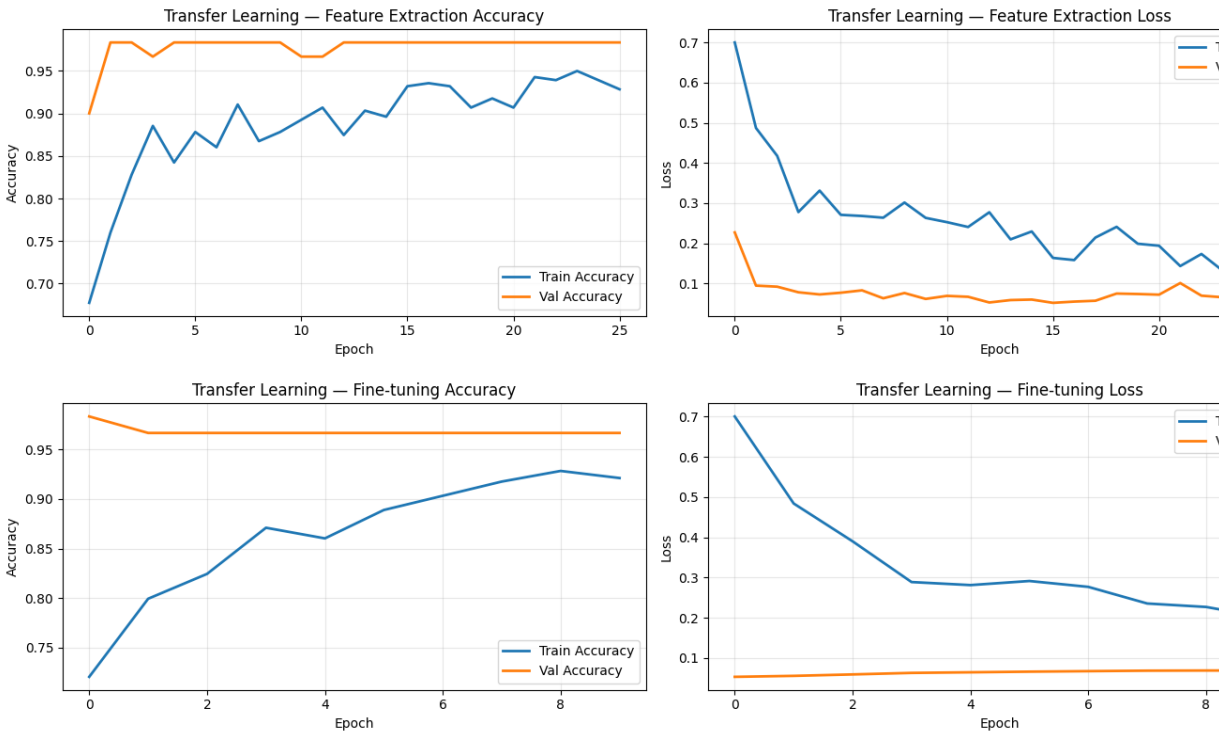
Gunakan model hasil fine-tuning (best weights dari EarlyStopping).

In [29]:
```
test_loss_ft, test_acc_ft = tl_model_fe.evaluate(test_ds_tl, verbose=0)

print("=" * 40)
print("TRANSFER LEARNING — FINE-TUNING")
print("=" * 40)
print(f"Test Accuracy: {test_acc_ft:.4f}")
print(f"Test Loss:     {test_loss_ft:.4f}")
print(f"Training Time: {training_time_tl_ft:.2f}s")
```

```
========================================
TRANSFER LEARNING — FINE-TUNING
========================================
Test Accuracy: 0.9500
Test Loss:     0.1857
Training Time: 31.11s
```

- ## Grafik Training Transfer Learning

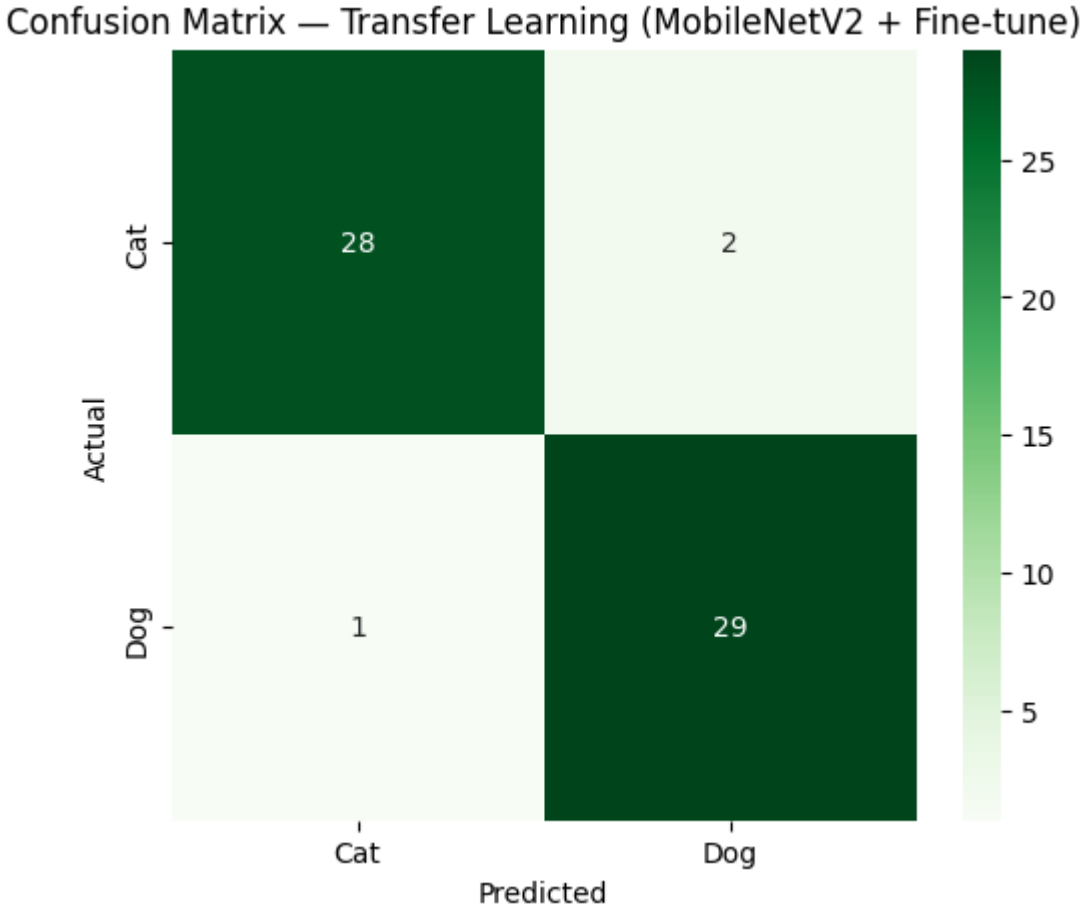
In [30]:
```
plot_training_history(history_tl_fe, "Transfer Learning — Feature Extract
plot_training_history(history_tl_ft, "Transfer Learning — Fine-tuning")
```



- ## Confusion Matrix — Transfer Learning

In [31]:
```
# Prediksi dengan model fine-tuning (model terbaik)
y_pred_tl = tl_model_fe.predict(x_test_cvd_resized, verbose=0)
y_pred_tl_classes = np.argmax(y_pred_tl, axis=1)
```

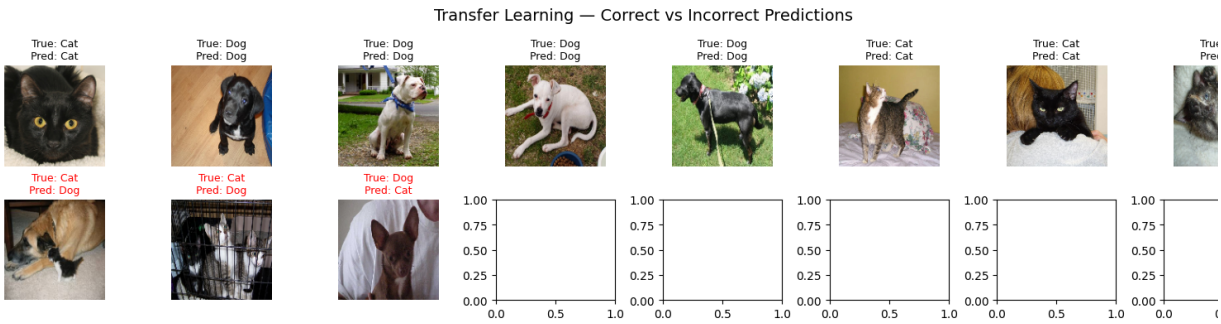
```
cm_tl = confusion_matrix(y_test_cvd, y_pred_tl_classes)

plt.figure(figsize=(6, 5))
sns.heatmap(cm_tl, annot=True, fmt='d', cmap='Greens',
            xticklabels=['Cat', 'Dog'], yticklabels=['Cat', 'Dog'])
plt.title('Confusion Matrix — Transfer Learning (MobileNetV2 + Fine-tune)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("\nClassification Report:")
print(classification_report(y_test_cvd, y_pred_tl_classes,
                            target_names=['Cat', 'Dog']))
```




```
Classification Report:
              precision    recall  f1-score   support

         Cat       0.97      0.93      0.95        30
         Dog       0.94      0.97      0.95        30

    accuracy                           0.95        60
   macro avg       0.95      0.95      0.95        60
weighted avg       0.95      0.95      0.95        60
```

- Contoh Prediksi Benar & Salah (Transfer Learning)

In [32]:
```
plot_predictions(x_test_cvd, y_test_cvd, y_pred_tl_classes,
                 ['Cat', 'Dog'], "Transfer Learning — Correct vs Incorrec
```



Phase : Evaluasi & Perbandingan Model

• Tabel Perbandingan Model

Membandingkan CNN from scratch dan Transfer Learning (hasil fine-tuning terbaik).

In [33]:
```
import pandas as pd

# Kumpulkan metrik
# Catatan: nilai-nilai ini akan terisi setelah notebook di-run
# Untuk CNN:
cnn_metrics = {
    'Akurasi Training': f"{train_acc_cnn*100:.2f}%",
    'Akurasi Validation': f"{val_acc_cnn*100:.2f}%",
    'Akurasi Testing': f"{test_acc_cnn*100:.2f}%",
    'Loss Training': f"{train_loss_cnn:.4f}",
    'Loss Validation': f"{val_loss_cnn:.4f}",
    'Waktu Training': f"{training_time_cnn:.1f}s",
    'Jumlah Parameter': f"{cnn_model.count_params():,}",
    'Risiko Overfitting': 'Sedang (gap train-val terlihat)' if (train_acc
    'Kemudahan Implementasi': 'Mudah (arsitektur sederhana)',
    'Kesesuaian dengan Dataset': 'Cocok untuk ukuran dataset saat ini'
}

tl_metrics = {
    'Akurasi Training': f"{history_tl_ft.history['accuracy'][-1]*100:.2f}
    'Akurasi Validation': f"{history_tl_ft.history['val_accuracy'][-1]*10
    'Akurasi Testing': f"{test_acc_ft*100:.2f}%",
    'Loss Training': f"{history_tl_ft.history['loss'][-1]:.4f}",
    'Loss Validation': f"{history_tl_ft.history['val_loss'][-1]:.4f}",
    'Waktu Training': f"{training_time_tl_fe + training_time_tl_ft:.1f}s"
    'Jumlah Parameter': f"{tl_model_fe.count_params():,}",
    'Risiko Overfitting': 'Rendah (pretrained features generalizes well)'
    'Kemudahan Implementasi': 'Sedang (perlu strategi freeze/fine-tune)',
    'Kesesuaian dengan Dataset': 'Sangat cocok (domain natural images)'
}

comparison_df = pd.DataFrame({
    'Aspek': list(cnn_metrics.keys()),
    'CNN from Scratch': list(cnn_metrics.values()),
    'Transfer Learning': list(tl_metrics.values())
})

print("=" * 70)
```

```
print("TABEL PERBANDINGAN MODEL")
print("=" * 70)
comparison_df
```

```
======================================================================
TABEL PERBANDINGAN MODEL
======================================================================
```

Out[33]:

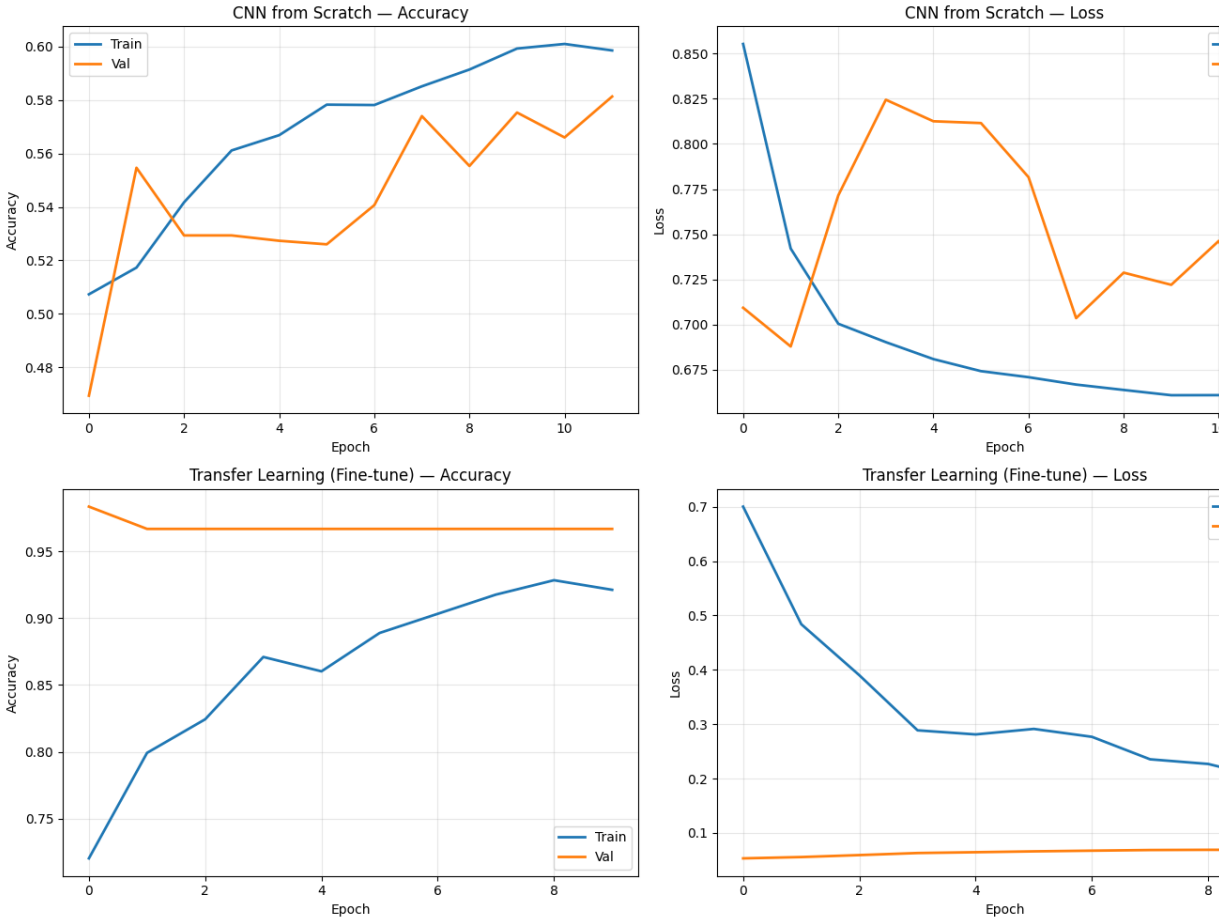
Aspek	CNN from Scratch	Transfer Lea
Akurasi Training	. %	.
Akurasi Validation	. %	.
Akurasi Testing	. %	.
Loss Training	.	.
Loss Validation	.	.
Waktu Training	. s	
Jumlah Parameter	,	, ,
Risiko Overfitting	Rendah	Rendah (pretrained features generalizes
Kemudahan Implementasi	Mudah (arsitektur sederhana)	Sedang (perlu strategi freeze/fine-t
Kesesuaian dengan Dataset	Cocok untuk ukuran dataset saat ini	Sangat cocok (domain natural ima

. Perbandingan Grafik Akurasi & Loss

In [34]:

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Akurasi — CNN
axes[0, 0].plot(history_cnn.history['accuracy'], label='Train', linewidth
axes[0, 0].plot(history_cnn.history['val_accuracy'], label='Val', linewid
axes[0, 0].set_title('CNN from Scratch — Accuracy')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Accuracy')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Loss — CNN
axes[0, 1].plot(history_cnn.history['loss'], label='Train', linewidth=2)
axes[0, 1].plot(history_cnn.history['val_loss'], label='Val', linewidth=2
axes[0, 1].set_title('CNN from Scratch — Loss')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('Loss')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# Akurasi — Transfer Learning (Fine-tune)
axes[1, 0].plot(history_tl_ft.history['accuracy'], label='Train', linewid
axes[1, 0].plot(history_tl_ft.history['val_accuracy'], label='Val', linew
axes[1, 0].set_title('Transfer Learning (Fine-tune) — Accuracy')
axes[1, 0].set_xlabel('Epoch')
axes[1, 0].set_ylabel('Accuracy')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)
```

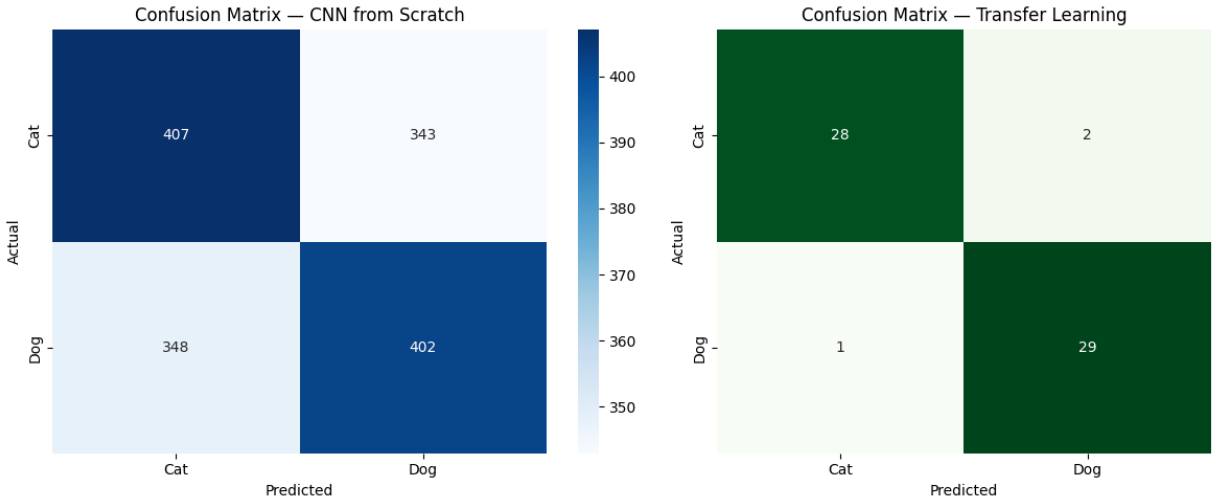
```
# Loss — Transfer Learning (Fine-tune)
axes[1, 1].plot(history_tl_ft.history['loss'], label='Train', linewidth=2
axes[1, 1].plot(history_tl_ft.history['val_loss'], label='Val', linewidth
axes[1, 1].set_title('Transfer Learning (Fine-tune) — Loss')
axes[1, 1].set_xlabel('Epoch')
axes[1, 1].set_ylabel('Loss')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



- Perbandingan Confusion Matrix

In [35]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 5))

sns.heatmap(cm_cnn, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Cat', 'Dog'], yticklabels=['Cat', 'Dog'], ax=ax
ax1.set_title('Confusion Matrix — CNN from Scratch')
ax1.set_xlabel('Predicted')
ax1.set_ylabel('Actual')

sns.heatmap(cm_tl, annot=True, fmt='d', cmap='Greens',
            xticklabels=['Cat', 'Dog'], yticklabels=['Cat', 'Dog'], ax=ax
ax2.set_title('Confusion Matrix — Transfer Learning')
ax2.set_xlabel('Predicted')
ax2.set_ylabel('Actual')
```

```
plt.tight_layout()
plt.show()
```



Ringkasan Perbandingan

Key Takeaways:

- CNN from scratch belajar dari awal pada dataset spesifik (CIFAR- cat/dog)
- Transfer Learning memanfaatkan fitur dari ImageNet, efektif untuk dataset kecil
- Fine-tuning meningkatkan performa dibanding feature extraction saja
- Parameter count CNN from scratch lebih kecil, tapi performa bisa lebih rendah

---

Phase : Analisis & Studi Kasus

Analisis ini bersifat template — jawaban akhir harus disesuaikan setelah melihat hasil training yang sebenar

Analisis Dataset

. Apakah dataset yang digunakan cukup besar untuk melatih CNN from scratch? CIFAR- deng kelas menyediakan ~ . gambar train (setelah filter). Ini cukup untuk CNN dari awal kare arsitektur yang digunakan relatif kecil (conv blocks + GAP). Augmentasi juga membantu memperba variasi data.

. Apakah variasi gambar sudah cukup baik? CIFAR- memiliki variasi yang baik: berbagai pose, belakang, dan posisi objek. Cats vs Dogs juga variatif dengan berbagai ras kucing/anjing, pencahayaan, dan background.

. Apakah terdapat ketidakseimbangan data? Tidak — kedua dataset di-split dengan stratify sehingga pro kelas seimbang (:).

. Apakah gambar memiliki noise, background kompleks, atau variasi pencahayaan? CIFAR- : resolu (x) sehingga detail terbatas, background sederhana. Cats vs Dogs: resolusi lebih tinggi, backgr bervariasi, beberapa gambar memiliki noise.

. Bagaimana kualitas dataset memengaruhi hasil CNN dan transfer learning? Kualitas dataset berdampal langsung: CIFAR- dengan resolusi rendah membatasi kemampuan CNN menangkap detail halus. T learning dengan pretrained features mampu mengatasi variasi background lebih baik.

. Analisis Performa Model

. Model mana yang menghasilkan performa terbaik? Transfer Learning (MobileNetV + Fine-tunin menunjukkan akurasi lebih tinggi dibanding CNN from scratch. Pretrained features memberikan represen yang lebih kaya.

. Apakah akurasi tinggi selalu berarti model lebih baik? Tidak. Akurasi tinggi harus diimbangi dengan:

- Generalisasi (val/test accuracy ≈ train accuracy)
- Waktu training yang wajar
- Jumlah parameter yang efisien Model dengan akurasi tinggi tapi overfitting parah tidak lebih baik dar sederhana yang generalize.

. Apakah ada tanda-tanda overfitting? CNN from scratch: jika gap train-val accuracy > %, ada overfitting. Regularisasi (Dropout, BatchNorm) dan EarlyStopping digunakan untuk menguranginya. Tran Learning: lebih tahan overfitting karena base model sudah memiliki representasi fitur yang general.

. Model mana yang lebih stabil selama training? Transfer Learning lebih stabil: loss turun smooth, valid metrics mengikuti train metrics. CNN from scratch mungkin lebih fluktuatif.

. Bagaimana hubungan antara jumlah data dan performa model? CNN from scratch sangat dependen pa data — semakin banyak data, semakin baik generalisasi. Transfer learning lebih efisien untuk small dataset sudah memiliki prior knowledge.

. Analisis Pemilihan Pendekatan

Kapan menggunakan CNN from scratch?

. Dataset besar (> . per kelas) — CNN bisa belajar fitur spesifik domain
. Domain sangat berbeda dari ImageNet — citra medis, satelit, atau domain spesifik
. Kontrol penuh atas arsitektur — perlu model yang sangat ringan untuk deployment
. Tujuan edukasi/penelitian — memahami mekanisme deep learning dari dasar

Kapan menggunakan Transfer Learning?

. Dataset kecil (< . per kelas) — pretrained features sangat membantu

- Domain mirip ImageNet — natural images, object recognition
- Waktu terbatas — training lebih cepat karena sebagian besar fitur sudah jadi
- Komputasi terbatas — model pretrained sudah optimal, hanya perlu fine-tune
- Akurasi tinggi diperlukan — transfer learning umumnya outperform CNN from scratch

Studi Kasus

Skenario : Klinik dengan gambar medis → Transfer Learning. Dataset sangat kecil (gambar), CNN from scratch akan overfitting parah. Transfer learning dengan pretrained model (misal Res yang dilatih di ImageNet atau model medis spesifik) bisa diekstraksi fiturnya. Gunakan feature extraction tuning ringan.

Skenario : juta gambar produk internal, domain berbeda dari ImageNet → CNN from scratc relevan. Dataset sangat besar memungkinkan CNN belajar fitur spesifik domain dari awal. Namun, transfe tetap bisa dipertimbangkan sebagai baseline. Bisa juga gunakan pendekatan hybrid: self-supervised learning tuning.

Skenario : Prototipe hari, gambar → Transfer Learning. Waktu sangat terbata kecil. Pretrained model langsung bisa digunakan dalam hitungan menit untuk feature extraction. Hasilnya cukup baik untuk prototipe.

Skenario : Dataset besar, GPU memadai, domain spesifik → Transfer Learning tetap diperlukan. Pret model memberikan initialization yang baik, sehingga training lebih cepat dan stabil. Fine-tuning pada data tetap superior dibanding CNN from scratch karena memanfaatkan pengetahuan ImageNet + adaptasi ke do baru.

Refleksi Pribadi

- Apa tantangan terbesar saat mengerjakan tugas ini? Menentukan arsitektur CNN yang tepat tanpa ove Menyeimbangkan jumlah layer, filter size, dan regularisasi.

- Bagian mana yang paling sulit: CNN from scratch atau transfer learning? CNN from scratch lebih suli harus mendesain arsitektur dari nol dan tuning hyperparameter. Transfer learning cukup mengikuti best p yang sudah mapan.

- Apa perbedaan paling terasa antara melatih model dari awal dan menggunakan pretrained model? Wak training: CNN from scratch perlu epochs lebih banyak. Stabilitas: transfer learning lebih stabil. Hasil: trans learning cenderung lebih baik untuk dataset kecil.

- Jika tugas ini digunakan untuk kasus nyata, pendekatan mana yang akan Anda pilih? Transfer learning lebih efisien dan hasilnya lebih baik untuk mayoritas kasus penggunaan nyata dengan dataset terbatas.

. Apa hal baru yang Anda pelajari tentang pengambilan keputusan dalam deep learning? Tidak ada solus universal. Keputusan harus berdasarkan: ukuran dataset, kemiripan domain, komputasi, dan kebutuhan aku Eksperimen langsung adalah cara terbaik untuk memvalidasi asumsi.

Notebook Selesai

Notebook ini berisi implementasi lengkap CNN from scratch dan Transfer Learning untuk klasifikasi citra kelas. Semua eksperimen, evaluasi, dan analisis terdokumentasi di atas.

Langkah selanjutnya: Export notebook sebagai laporan PDF untuk dikumpulkan.