

CNN DENGAN STUDI KASUS SIMPLICITY DATASET

Adrian Mendienta Tumanggor 3123600012

DATASET YANG DIGUNAKAN

- Dataset ini diambil dari paper SIMPLlcity: Semantics-Sensitive Integrated Image Retrieval
- Sistem SIMPLlcity diaplikasikan pada database gambar general-purpose yang berisi sekitar 200.000 gambar.
- Database ini diidentifikasi sebagai database COREL.
- Gambar-gambar dalam database ini disimpan dalam format JPEG , dengan ukuran 384×256 atau 256×384 .
- Untuk evaluasi sistematis, pengujian juga dilakukan pada subset database COREL yang terdiri dari 10 kategori gambar, di mana setiap kategori berisi 100 gambar.

METODE YANG DIGUNAKAN DI PENELITIAN SIMPLICITY

SIMPLIcity (2000)

- Tidak pakai neural network.
- Semua “fitur” diekstraksi manual (warna, tekstur, bentuk).
- Analisis berbasis statistik dan probabilitas.
- Segmentasi pakai k-means.
- Fokus: content-based image retrieval (CBIR).
- Bisa dibilang, arsitekturnya adalah “CNN versi manual sebelum CNN ada.”
- SIMPLIcity tidak punya fully connected layer dan tidak butuh tensor tetap, ukuran bisa bervariasi.
- Ia memproses gambar berdasarkan blok (4×4), bukan dimensi global.

SIZE IMAGE

Resize Langsung

- Mengubah ukuran gambar ke target (mis. 224×224) tanpa mempertahankan rasio aspek.
- Bentuk objek dapat terdistorsi (melebar atau memanjang).
- Cepat dan hemat ruang komputasi.
- Cocok untuk dataset besar yang kurang sensitif terhadap bentuk (mis. tekstur, pola sederhana).
- Kurang akurat untuk klasifikasi berbasis bentuk atau struktur geometris.

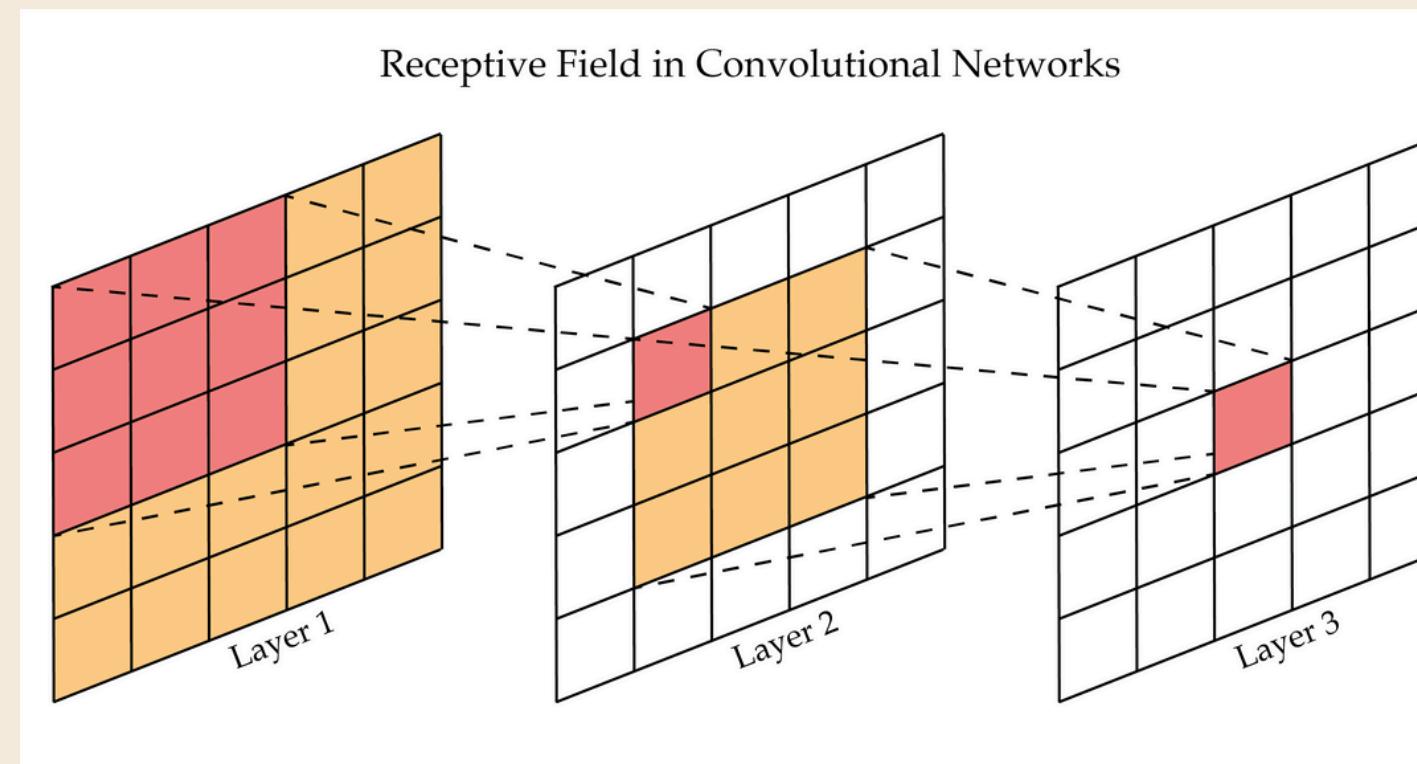
Resize + Padding

- Menjaga rasio aspek asli, lalu menambahkan padding (bingkai kosong) agar ukuran sesuai target.
- Bentuk dan proporsi objek tetap realistik dan tidak terdistorsi.
- Area padding tidak berisi informasi, tapi CNN dapat belajar mengabaikannya.
- Sedikit lebih berat komputasi, tapi hasil lebih stabil.
- Direkomendasikan untuk dataset foto alami seperti SIMPLICITY, ImageNet, COCO, dll.

disini saya memilih untuk menggunakan metode resize + padding 224×224 (resize ke 224×150 / 150×224 , ditambah padding $2 \times 37\text{px}$)

KERNEL SIZE

- Gunakan banyak layer dengan kernel 3×3
 - 3×3 adalah ukuran yang populer karena: cukup kecil untuk efisiensi komputasi, cukup untuk menangkap pola lokal (edges, corners) dan ketika di-stack beberapa kali bisa membangun **receptive field** yang besar tanpa banyak parameter tambahan.
 - **receptive field**
 - menjelaskan bahwa tinggi layernya setiap blok piksel akan merepresentasikan lebih luas dari field yang sebenarnya dari input



saya akan menggunakan 3 layer 3×3 convolutional

FILTER COUNT

- Filter count naik tiap blok (misalnya $32 \rightarrow 64 \rightarrow 128$)
 - layer awal belajar fitur sederhana \rightarrow kurang channel cukup; layer lebih dalam harus merepresentasikan fitur kompleks \rightarrow perlu lebih banyak filter. Dari literatur umum CNN: "Since feature map size decreases with depth, layers near the input tend to have fewer filters while higher layers can have more."
 - Saat spatial size mengecil (karena pooling/stride) dan layer semakin dalam, model harus menangani kompleksitas yang meningkat (bagian objek, struktur). Maka perlu lebih banyak filter agar ruang representasi cukup.

saya memilih menggunakan 3 layer dengan incremental filter count $16 \rightarrow 32 \rightarrow 64$

DETAIL LAINNYA

- Dropout ≈ 0.5 (di dense layer)
- menggunakan Global Average Pooling
 - GAP tidak punya bobot yang harus di-train (no learnable parameters) untuk layer itu sendiri. Jadi bagian akhir model tidak dibuat “besar dan padat” seperti fully connected besar. Ini membantu model generalisasi lebih baik
 - Dalam literatur (misalnya paper “Network in Network”), GAP dipakai agar tiap feature map terakhir bisa secara langsung diasosiasikan (mapped) ke satu kelas (atau set fitur kelas) sehingga interpretasi dan pemetaan fitur \rightarrow kelas jadi lebih langsung.
- Dataset yang digunakan:
 - 100 gambar tiap class(10 class)
 - pembagian 50, 30, 20 untuk train, val , dan test
 - dataset sudah dipisahkan dan siap dipakai /train /validation /test

METODE NEURAL NETWORK YANG AKAN SAYA COBA

Layer	Kernel / Operasi	Input Size	Output Size	Receptive Field (terhadap gambar)	Channels / Units
Input	—	224×224×3	224×224×3	—	3
Conv1	3×3, stride 1, padding=same	224×224×3	224×224×16	3×3	16 filter
Conv2	3×3, stride 1, padding=same	224×224×16	224×224×32	5×5	32 filter
Conv3	3×3, stride 1, padding=same	224×224×32	224×224×64	7×7	64 filter
Global Average Pooling	rata-rata per channel	224×224×64	1×1×64 (jadi vektor 64)	tetap 7×7 (dilihat per neuron)	64
Dense (Softmax)	10 unit	64	10	—	10 kelas

EXPLORE DATASET

```
print("Isi direktori data/SIMPLICITY:")
for split_dir in [config.TRAIN_DIR, config.VAL_DIR, config.TEST_DIR]:
    print(f"- {split_dir.name}:")
    if split_dir.exists():
        for p in sorted(split_dir.iterdir())[:5]:
            print("  ", p.name)
        print("  ...")
    else:
        print("  (TIDAK ADA)")

[6] ✓ 0.0s
...
Isi direktori data/SIMPLICITY:
- train:
  0.jpg
  1.jpg
  10.jpg
  100.jpg
  101.jpg
...
- validation:
  150.jpg
  151.jpg
  152.jpg
  153.jpg
  154.jpg
...
- test:
  180.jpg
  181.jpg
  182.jpg
  183.jpg
  184.jpg
...

def count_images_in_dir(dir_path: Path):
    exts = ("jpg", "jpeg", ".png", ".JPG", ".JPEG", ".PNG")
    return sum(1 for p in dir_path.iterdir() if p.suffix in exts)

data = {
    "split": [],
    "num_images": [],
}

for split_name, split_dir in [
    ("train", config.TRAIN_DIR),
    ("validation", config.VAL_DIR),
    ("test", config.TEST_DIR),
]:
    data["split"].append(split_name)
    data["num_images"].append(count_images_in_dir(split_dir))

df_counts = pd.DataFrame(data)
df_counts

[7] ✓ 0.0s
...
   split  num_images
0   train       500
1 validation     300
2      test      200
```

Dataset

Category	Image Code
People	0-99
Beach	100-199
Building	200-299
Bus	300-399
Dinosaur	400-499

Category	Image Code
Elephant	500-599
Flower	600-699
Horse	700-799
Mountain	800-899
Food	900-999

- pembagian dataset sesuai index yang ditentukan
- dataset sudah dipisahkan menjadi direktori dan isi gambar yang tepat

EXPLORE DATASET

```
import matplotlib.image as mpimg

try:
    from src import config
except ImportError:
    print("Gagal import 'src.config'. Pastikan notebook ada di root project.")
    # Fallback path jika import gagal (sesuaikan jika perlu)
    class Config:
        TRAIN_DIR = Path("data/SIMPLICITY/train")
config = Config()

[17] ✓ 0.0s
```

```
# -----
# Fungsi Helper (dari kode Anda + tambahan)
# -----

def filename_to_index(filename: str) -> int:
    """ '123.jpg' -> 123 """
    stem = Path(filename).stem # '123'
    return int(stem)

def index_to_class_id(index: int) -> int:
    """ 123 -> 1 (karena 100-199) """
    return index // 100

def filename_to_class_id(filename: str) -> int:
    """ '123.jpg' -> 1 """
    return index_to_class_id(filename_to_index(filename))

def class_id_to_name(class_id: int) -> str:
    """ 1 -> 'Beach' """
    if 0 <= class_id < len(CLASS_NAMES):
        return CLASS_NAMES[class_id]
    return "Unknown"

[18] ✓ 0.0s
```

- import dataset train
- definisikan helper functions untuk indexing sesuai kategori yang ada

EXPLORE DATASET

```
# -----
# Visualisasi
# -----  
  
examples = ["0.jpg", "5.jpg", "100.jpg", "200.jpg", "301.jpg", "900.jpg"]
image_base_dir = Path(config.TRAIN_DIR) # Asumsi gambar ada di folder train  
  
# Buat grid plot 2x3
fig, axes = plt.subplots(2, 3, figsize=(15, 9))
axes = axes.flatten() # Ubah jadi array 1D agar mudah di-loop  
  
print(f"Mengambil gambar dari base path: {image_base_dir}")  
  
for i, fname in enumerate(examples):
    ax = axes[i]
    # --- 1. Dapatkan semua info ---
    idx = filename_to_index(fname)
    cid = filename_to_class_id(fname)
    cname = class_id_to_name(cid) # Ini bagian categorical  
  
    # --- 2. Tampilkan gambar ---
    img_path = image_base_dir / fname
    if img_path.exists():
        img = mpimg.imread(img_path)
        ax.imshow(img)
    else:
        ax.text(0.5, 0.5, f"Gambar tidak ditemukan:\n{fname}",
                ha='center', va='center', color='red', fontsize=10)
    # --- 3. Set title dengan semua info ---
    title = f"File: {fname}\n"
    title += f"Index: {idx}, Class ID: {cid}\n"
    title += f"Kategori: {cname}"
    ax.set_title(title, fontsize=10)
    ax.axis("off") # Sembunyikan sumbu x/y  
  
plt.tight_layout()
plt.show()
```

```
# -----
# Daftar Nama Kategori (sesuai gambar dataset)
# -----  
  
CLASS_NAMES = [
    "People",      # 0
    "Beach",       # 1
    "Building",    # 2
    "Bus",          # 3
    "Dinosaur",    # 4
    "Elephant",    # 5
    "Flower",      # 6
    "Horse",        # 7
    "Mountain",    # 8
    "Food",         # 9
]  
[✓] 0.0s
```

- definisikan class categorical untuk tiap index
- tampilkan dalam plot gambar2 untuk eksplorasi

EXPLORE DATASET

Mengambil gambar dari base path: [/home/tumanggors/github/neuralcomp/simplicity_cnn/data/SIMPLICITY/train](https://home/tumanggors/github/neuralcomp/simplicity_cnn/data/SIMPLICITY/train)



CONFIG.PY

```
src > config.py > ...  
47 # ---  
48 # PENGATURAN GAMBAR  
49 # ---  
50  
51 IMG_HEIGHT = 224  
52 IMG_WIDTH = 224  
53 IMG_CHANNELS = 3  
54 INPUT_SHAPE = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)  
55  
56 # Spesifikasi resize + padding sesuai instruksi:  
57 # - inner side ~150  
58 # - padding 37 di dua sisi (atas-bawah atau kiri-kanan)  
59 RESIZED_MIN_SIDE = 150  
60 PADDING_SIZE = 37  
61  
62 # ---  
63 # HYPERPARAMETER TRAINING  
64 # ---  
65  
66 NUM_CLASSES = 10      # SIMPLICITY: 10 kategori  
67 BATCH_SIZE = 16  
68 EPOCHS = 50  
69  
70 LEARNING_RATE = 1e-3  
71  
72 # Early stopping  
73 EARLY_STOPPING_MONITOR = "val_loss"  
74 EARLY_STOPPING_PATIENCE = 6 # di antara 5-7 sesuai instruksi  
75  
76 # Learning rate decay (exponential decay)  
77 LR_DECAY_RATE = 0.9  
78 LR_DECAY_STEPS = 5      # bisa diartikan "setiap 5 epoch"  
79  
80 # ---  
81 # RANDOM SEED (untuk reproducibility)  
82 # ---  
83  
84 RANDOM_SEED = 42  
85
```

- mendefinisikan variable2 penting dalam proses training
- Input gambar untuk model diatur ke 224x224 pixel dengan 3 channel warna (RGB).
- Proses preprocessing menggunakan padding 37 pixel dan resize sisi minimum ke 150 pixel (sesuai penjelasan sbolumnya).
- Model dilatih untuk mengklasifikasikan 10 kelas (kategori).
- Ukuran batch size diatur ke 16 (memproses 16 gambar sekaligus).
- Training akan berjalan selama maksimal 50 epoch.
- Learning rate awal ditetapkan sebesar 1e-3 (0.001).
- Early Stopping diaktifkan untuk memonitor val_loss. Training akan berhenti jika tidak ada perbaikan (penurunan loss) selama 6 epoch berturut-turut.
- Learning Rate Decay (exponential) diterapkan, di mana learning rate akan dikalikan 0.9 setiap 5 step/epoch.

TRAIN MODEL: SETUP DATASET

```
train_ds = dataset.get_train_dataset(batch_size=config.BATCH_SIZE)
val_ds = dataset.get_validation_dataset(batch_size=config.BATCH_SIZE)

for images, labels in train_ds.take(1):
    print("Train batch - images shape:", images.shape)
    print("Train batch - labels shape:", labels.shape)
    break
```

[10] ✓ 0.5s

```
... Train batch - images shape: (16, 224, 224, 3)
    Train batch - labels shape: (16, 10)
```

- set batch size untuk data training dan validation
- disini kita menggunakan batch size 16 tidak 1 seperti SGD umumnya

TRAIN MODEL: COMPILE MODEL

```
cnn_model = model_module.build_model(name=config.MODEL_NAME)
cnn_model.summary()

[11] ✓ 0.1s
...
Model: "simplicity_cnn_tf"
...



| Layer (type)                                    | Output Shape         | Param # |
|-------------------------------------------------|----------------------|---------|
| input_image (InputLayer)                        | (None, 224, 224, 3)  | 0       |
| conv_block1_conv (Conv2D)                       | (None, 224, 224, 16) | 432     |
| conv_block1_bn (BatchNormalization)             | (None, 224, 224, 16) | 64      |
| conv_block1_relu (ReLU)                         | (None, 224, 224, 16) | 0       |
| conv_block2_conv (Conv2D)                       | (None, 224, 224, 32) | 4,608   |
| conv_block2_bn (BatchNormalization)             | (None, 224, 224, 32) | 128     |
| conv_block2_relu (ReLU)                         | (None, 224, 224, 32) | 0       |
| conv_block3_conv (Conv2D)                       | (None, 224, 224, 64) | 18,432  |
| conv_block3_bn (BatchNormalization)             | (None, 224, 224, 64) | 256     |
| conv_block3_relu (ReLU)                         | (None, 224, 224, 64) | 0       |
| global_average_pooling (GlobalAveragePooling2D) | (None, 64)           | 0       |
| dropout (Dropout)                               | (None, 64)           | 0       |
| predictions (Dense)                             | (None, 10)           | 650     |



...
Total params: 24,570 (95.98 KB)
...
Trainable params: 24,346 (95.10 KB)
...
Non-trainable params: 224 (896.00 B)
```

- Output Shape: (None, 224, 224, 16). Ukuran gambar tetap (karena padding='same'), tapi "kedalaman" gambar bertambah dari 3 (RGB) menjadi 16 (peta fitur).
- BatchNormalization (bn) dan ReLU (relu) adalah layer standar untuk menstabilkan dan mempercepat training.
- (Tinggi Kernel × Lebar Kernel × Channel Input) × Jumlah Filter (Channel Output): kernel = 3x3
- Data masuk ke block konvolusi kedua(kernel 3x3), mencari 32 pola.
- Output Shape: (None, 224, 224, 32). Ukuran masih sama, tapi kedalaman fitur bertambah jadi 32.
- Data masuk ke block konvolusi ketiga, mencari 64 pola.
- Output Shape: (None, 224, 224, 64). Ukuran masih sama, kedalaman fitur bertambah jadi 64
- Global average pooling layer:mengambil 64 channel tadi, dan untuk setiap channel, ia menghitung nilai rata-rata dari seluruh 224x224 pixel. Hasilnya adalah 1 angka per channel, sehingga total menjadi 64 angka (vektor).
- Dropout Layer untuk mencegah overfitting dengan "mematikan" beberapa neuron secara acak saat training.

TRAIN MODEL

- set callback fuction untuk early stoppiing
- compile dengan optimizer yang suda diset di config sebelumnya
- lakukan training
- hasilnya adalah berhenti di epoch ke 36 dengan model terbaik pada epoch 30 tidak terjadi perbaikan pada val_loss sbg metric montior setelah 6 epoch

```
# Reuse fungsi dari src/train.py
optimizer = train_module.build_optimizer_with_schedule()
callbacks = train_module.build_callbacks()

cnn_model.compile(
    optimizer=optimizer,
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

[12] ✓ 0.0s

> v
EPOCHS = config.EPOCHS

history = cnn_model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1,
)
[13] ✓ 34m 13.8s

...
Epoch 1/50
32/32 ━━━━━━━━ 0s 1s/step - accuracy: 0.2050 - loss: 2.1825
Epoch 1: val_accuracy improved from None to 0.14000, saving model to /home/tu
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
32/32 ━━━━━━━━ 55s 2s/step - accuracy: 0.2740 - loss: 2.0094 - va
Epoch 2/50
32/32 ━━━━━━━━ 0s 1s/step - accuracy: 0.2050 - loss: 2.1825
...
```

```
...
Epoch 36: val_accuracy did not improve from 0.79667
32/32 ━━━━━━━━ 42s 1s/step - accuracy: 0.6860 - loss: 0.8979 - val_accuracy: 0.6633 - val_loss: 0.9048
Epoch 36: early stopping
Restoring model weights from the end of the best epoch: 30.
```

TRAIN MODEL: VISUALISASI

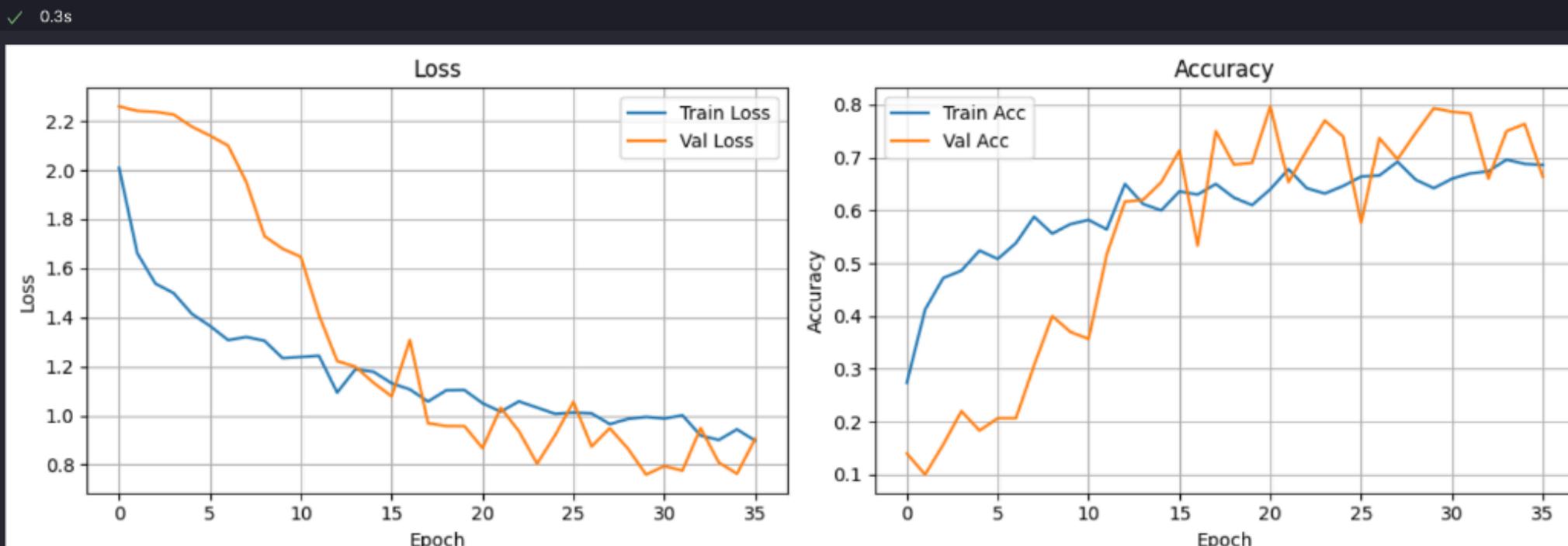
```
train_loss = hist_dict.get("loss", [])
val_loss = hist_dict.get("val_loss", [])
train_acc = hist_dict.get("accuracy", hist_dict.get("acc", []))
val_acc = hist_dict.get("val_accuracy", hist_dict.get("val_acc", []))

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Loss
axes[0].plot(train_loss, label="Train Loss")
if val_loss:
    axes[0].plot(val_loss, label="Val Loss")
axes[0].set_title("Loss")
axes[0].set_xlabel("Epoch")
axes[0].set_ylabel("Loss")
axes[0].legend()
axes[0].grid(True)

# Accuracy
axes[1].plot(train_acc, label="Train Acc")
if val_acc:
    axes[1].plot(val_acc, label="Val Acc")
axes[1].set_title("Accuracy")
axes[1].set_xlabel("Epoch")
axes[1].set_ylabel("Accuracy")
axes[1].legend()
axes[1].grid(True)

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



EVALUASI MODEL

```
utils.set_seed()

best_model = evaluate.load_best_model()
best_model.summary()

[8]    ✓  0.1s

... WARNING:absl:Compiled the loaded model, but the compi
      ==> Loading best model from: /home/tumanggors/github/
```

Layer (type)	Output Shape	Param #
input_image (InputLayer)	(None, 224, 224, 3)	0
conv_block1_conv (Conv2D)	(None, 224, 224, 16)	432
conv_block1_bn (BatchNormalization)	(None, 224, 224, 16)	64
conv_block1_relu (ReLU)	(None, 224, 224, 16)	0
conv_block2_conv (Conv2D)	(None, 224, 224, 32)	4,608
conv_block2_bn (BatchNormalization)	(None, 224, 224, 32)	128
conv_block2_relu (ReLU)	(None, 224, 224, 32)	0
conv_block3_conv (Conv2D)	(None, 224, 224, 64)	18,432
conv_block3_bn (BatchNormalization)	(None, 224, 224, 64)	256
conv_block3_relu (ReLU)	(None, 224, 224, 64)	0
global_average_pooling (GlobalAveragePooling2D)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
predictions (Dense)	(None, 10)	650

... Total params: 24,571 (95.98 KB)

... Trainable params: 24,346 (95.10 KB)

... Non-trainable params: 224 (896.00 B)

... Optimizer params: 1 (8.00 B)

- load best model dari hasil training sebelumnya
- tampilkan summarynya

EVALUASI MODEL

```
    test_loss, test_acc, error_ratio = evaluate.evaluate_on_test(best_model)

    print("\nRingkasan evaluasi test set:")
    print(f"- Test loss : {test_loss:.4f}")
    print(f"- Test acc : {test_acc:.4f}")
    print(f"- Error ratio : {error_ratio:.4f}")

[9] ✓ 3.4s

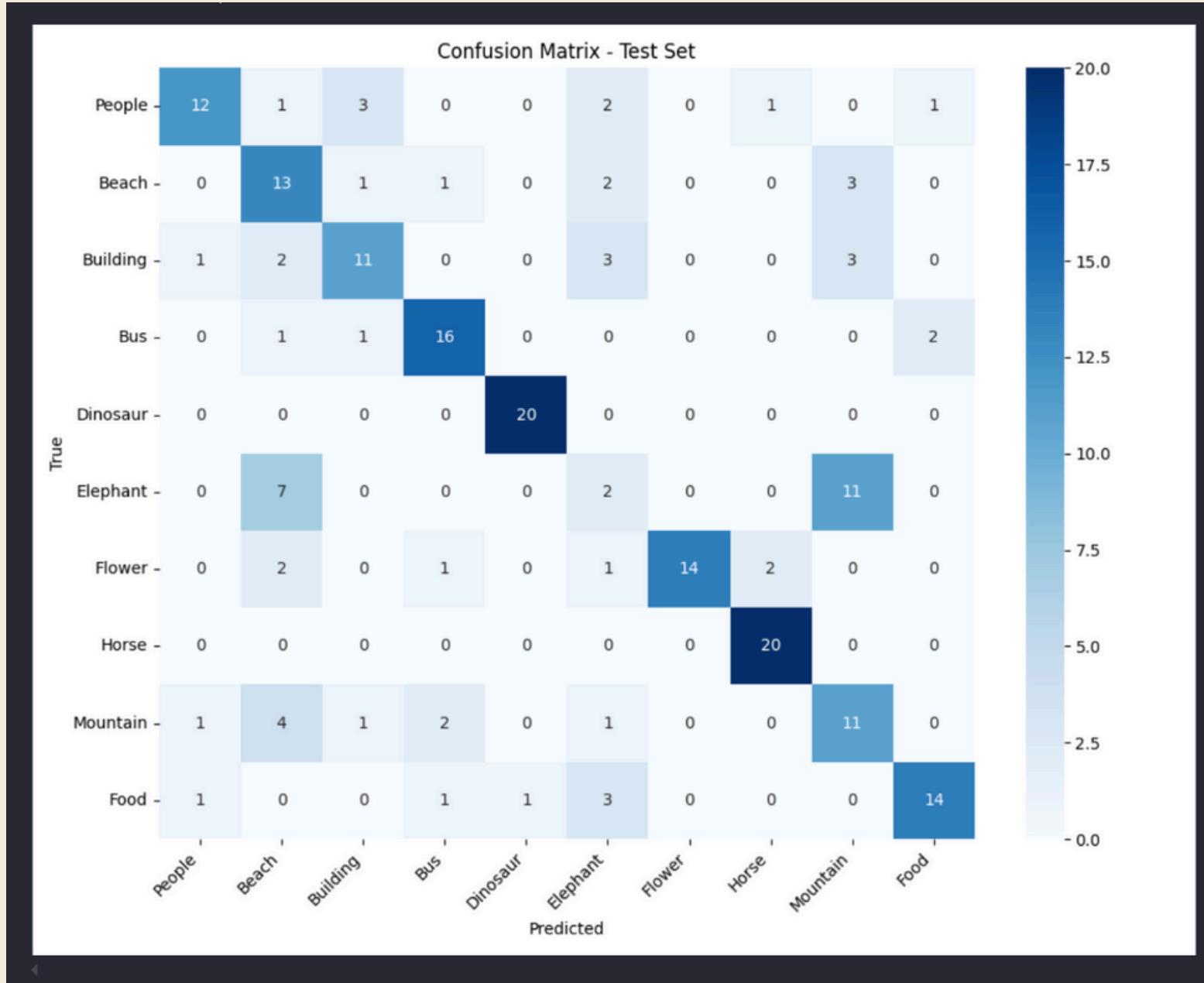
... ==> Loading test dataset...
==> Evaluating on test set...
13/13 ━━━━━━━━ 3s 217ms/step - accuracy: 0.6650 - loss: 1.1029
Test loss      : 1.1029
Test accuracy : 0.6650
Error ratio   : 0.3350

Ringkasan evaluasi test set:
- Test loss : 1.1029
- Test acc  : 0.6650
- Error ratio : 0.3350
```

```
72 def evaluate_on_test(model: tf.keras.Model) -> Tuple[float, float, float]:
73     """
74     Evaluasi model pada test set.
75
76     Return
77     -----
78     (test_loss, test_accuracy, error_ratio)
79     """
80     print("==> Loading test dataset...")
81     test_ds = dataset.get_test_dataset(batch_size=config.BATCH_SIZE)
82
83     print("==> Evaluating on test set...")
84     test_loss, test_acc = model.evaluate(test_ds, verbose=1)
85     error_ratio = 1.0 - float(test_acc)
86
87     print(f"Test loss      : {test_loss:.4f}")
88     print(f"Test accuracy : {test_acc:.4f}")
89     print(f"Error ratio   : {error_ratio:.4f}")
90
91     return float(test_loss), float(test_acc), error_ratio
92
```

- input best model kedalam method evaluate on test
- method evaluate berisi load data test dan prediction test dan mengembalikan nilai loss, accuracy, dan error ratio
- accuracy: 0.665 (66%~ akurasi)
- error ratio: 0.335

TRAIN MODEL: COMPILE MODEL



- Performa Terbaik: Model bekerja sempurna (akurasi 100%) untuk kelas Dinosaur (20/20) dan Horse (20/20), yang ditandai dengan warna biru paling gelap di diagonal.
- Performa Terburuk: Model gagal total di kelas Elephant. Dari semua gambar gajah, tidak ada satupun (0) yang berhasil diprediksi dengan benar.
- Kebingungan Terbesar: Model paling bingung dengan kelas Elephant. Sebagian besar (11) gambar gajah salah diprediksi sebagai Mountain, dan 7 gambar gajah salah diprediksi sebagai Beach.
- Kebingungan Lain: Model juga terlihat sering tertukar antara:
 - Beach (Asli) vs Mountain (Prediksi)
 - Mountain (Asli) vs Beach (Prediksi)
 - Building (Asli) vs Elephant dan Mountain (Prediksi)

```

sample_path = config.FIGURE_DIR / "sample_predictions_test.png"
from IPython.display import Image as IPyImage, display

display(IPyImage(filename=str(sample_path)))

[11] ✓ 0.7s
...
==> Visualizing 9 sample predictions (mencari kelas berbeda)...
Contoh prediksi disimpan di: /home/tumanggors/github/neuralcomp/simplicity_cnn/outputs/figures/sample_predictions_test

...
True: People
Pred: Food
---
- Food (42.2%)
- People (23.4%)
- Bus (9.8%)

True: Beach
Pred: Beach
---
- Beach (52.7%)
- Mountain (33.7%)
- Building (7.6%)

True: Building
Pred: Building
---
- Building (54.9%)
- Mountain (16.2%)
- Bus (13.1%)

...
True: Bus
Pred: Bus
---
- Bus (29.2%)
- Beach (27.1%)
- Building (10.4%)

True: Dinosaur
Pred: Dinosaur
---
- Dinosaur (83.2%)
- Beach (5.0%)
- Mountain (4.9%)

True: Elephant
Pred: Elephant
---
- Elephant (69.5%)
- Food (9.1%)
- People (8.3%)

...
True: Flower
Pred: Flower
---
- Flower (63.2%)
- Food (13.0%)
- People (8.5%)

True: Horse
Pred: Horse
---
- Horse (98.8%)
- Elephant (0.9%)
- People (0.1%)

True: Mountain
Pred: Mountain
---
- Mountain (44.7%)
- Beach (23.6%)
- Building (17.0%)


```

VISUALISASI SAMPLE

- prediksi dari masing2 kategorikal gambar
- model salah menebak orang dengan baju adat menjadi food dengan 42% percentage sedangkan manusia prediksi keduanya dengan 24%

DOKUMENTASI GITHUB & NOTEBOOK

[HTTPS://GITHUB.COM/ADRIANTUMANGGOR/
CONVOLUTION-NN-WITH-SIMPLICITY-DATASET](https://github.com/adriantumanggor/convolution-nn-with-simplicity-dataset)

**TERIMA
KASIH**