CONVOLUTIONAL NEURAL NETWORK FOR CLASSIFICATION CIFAR-10

DEEP LEARNING 2025

Dani Hidayat 11220940000014
M. Abdul Ghofur 11220940000020

TUJUAN PROYEK

01

Membangun model CNN untuk mengklasifikasi gambar ke dalam 10 kelas dari dataset CIFAR-10.

02

Mengevaluasi performa model dan membandingkan hasil akurasi Model yang dibangun dengan model AlexNet.

PENGENALAN DATASET

- Dataset: CIFAR-10
- Jumlah Data: 60.000 Gambar
- Ukuran Gambar: 32 x 32 pixel
- Channel Gambar: 3 Channel (RGB)

- Pembagiagn Kelas:

Terdapat 10 kelas yakni 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'

TRANSFORMASI DATA DAN AUGMENTASI

Transformasi dilakukan untuk mengonversi data ke bentuk tensor dan untuk menstabilkan training
Augmentasi dilakukan untuk membuat model lebih tahan terhadap variasi bentuk input

01
RANDOM HORIZONTAL FLIP

Melakukan pembalikan gambar secara horizontal secara acak. Tujuannya Augmentasi untuk menangani orientasi objek.

02
RANDOM CROP

Melakukan pemotongan acak (crop) gambar ukuran 32x32 dengan menambahkan padding 4 pixel di setiap sisi. Augmentasi untuk variasi posisi

03
TO TENSOR

Mengubah gambar dari format PIL atau numpy array ke tensor PyTorch dan menskalakan pixel dari [0, 255] ke [0.0, 1.0].

04
NORMALIZE

Melakukan normalisasi setiap channel (R, G, B) dipusatkan ke mean 0 dan rentang [-1, 1], mempercepat dan menstabilkan proses pelatihan CNN.

SPLIT DATASET

Pembagian Train - Validation - Test data

TRAIN

80% dari CIFAR-10 train
(40.000 data)

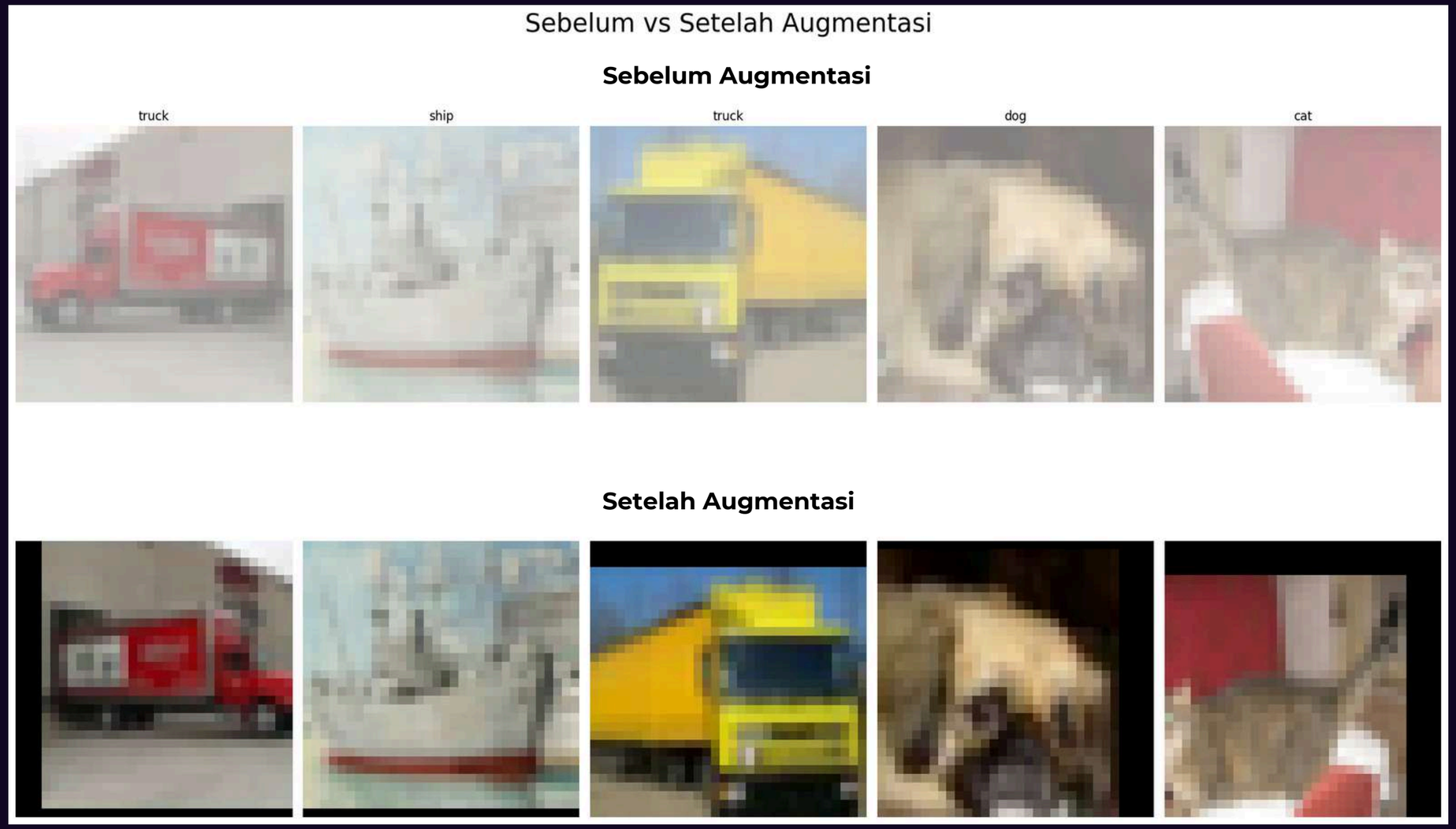
VALIDATION

20% dari CIFAR-10 train
(10.000 data)

TEST

CIFAR-10 test
(10.000 data)

CONTOH DATASET



STRUKTUR DAN ARSITEKTUR ALEXNET

AlexNet CIFAR-10					
Layer	**Dimensi Input**	**Stride**	**Padding**	**Ukuran Kernel**	**Output Dimensi**
Input	3 × 32 × 32	-	-	-	3 × 32 × 32
Conv1	3 × 32 × 32	1	1	3 x 3	64 × 32 × 32
MaxPool1	64 × 32 × 32	2	0	2 x 2	64 × 16 × 16
Conv2	64 × 16 × 16	1	1	3 x 3	192 × 16 × 16
MaxPool2	192 × 16 × 16	2	0	2 x 2	192 × 8 × 8
Conv3	192 × 8 × 8	1	1	3 x 3	384 × 8 × 8
Conv4	384 × 8 × 8	1	1	3 x 3	256 × 8 × 8
Conv5	256 × 8 × 8	1	1	3 x 3	256 × 8 × 8
MaxPool3	256 × 8 × 8	2	0	2 x 2	256 × 4 × 4
Flatten	256 × 4 × 4	-	-	-	4096
Fc6	4096	-	-	1 x 1	1024
Fc7	1024	-	-	1 x 1	512
Fc8	512	-	-	1 x 1	10

```python
class alexNet(nn.Module):
    def __init__(self):
        super(alexNet, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),          # Conv1
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),                         # MaxPool1

            nn.Conv2d(64, 192, kernel_size=3, stride=1, padding=1),        # Conv2
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),                         # MaxPool2

            nn.Conv2d(192, 384, kernel_size=3, stride=1, padding=1),       # Conv3
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1),       # Conv4
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),       # Conv5
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)                          # MaxPool5
        )

        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 4 * 4, 1024),  # FC6
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(1024, 512),          # FC7
            nn.ReLU(inplace=True),
            nn.Linear(512, 10),            # FC8
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)          # flatten
        x = self.classifier(x)
        return x
```

STRUKTUR DAN ARSITEKTUR CNN-1

CNN-1					
Layer	**Dimensi Input**	**Stride**	**Padding**	**Ukuran Kernel**	**Output Dimensi**
Input	3 × 32 × 32	-	-	-	3 × 32 × 32
Conv1	3 × 32 × 32	3	1	3 x 3	32 × 32 × 32
Conv2	32 × 32 × 32	3	1	3 x 3	64 × 32 × 32
MaxPool1	64 × 32 × 32	2	0	2 x 2	64 x 16 x 16
Conv3	64 x 16 x 16	3	1	3 x 3	128 x 16 x 16
MaxPool2	128 x 16 x 16	2	0	2 x 2	128 x 8 x 8
Flatten	128 x 8 x 8	-	-	-	8192
Fc1	8192	-	-	1 x 1	256
Fc2	256	-	-	1 x 1	10

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),     # (3, 32, 32) -> (32, 32, 32)
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),    # (32, 32, 32) -> (64, 32, 32)
            nn.BatchNorm2d(num_features=64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),          # -> (64, 16, 16)
            nn.Dropout(0.5),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),   # -> (128, 16, 16)
            nn.BatchNorm2d(num_features=128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),          # -> (128, 8, 8)
            nn.Dropout(0.5),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),                                   # -> (128 * 8 * 8)
            nn.Linear(128 * 8 * 8, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

STRUKTUR DAN ARSITEKTUR CNN-2

CNN-2					
Layer	**Dimensi Input**	**Stride**	**Padding**	**Ukuran Kernel**	**Output Dimensi**
Input	3 × 32 × 32	-	-	-	3 × 32 × 32
Conv1	3 × 32 × 32	3	1	3 x 3	64 × 32 × 32
MaxPool1	64 × 32 × 32	2	0	2 x 2	64 × 16 × 16
Conv2	64 × 16 × 16	3	1	3 x 3	128 × 16 × 16
MaxPool2	128 × 16 × 16	2	0	2 x 2	128 x 8 x 8
Conv3	128 x 8 x 8	3	1	3 x 3	256 x 8 x 8
MaxPool3	256 x 8 x 8	2	0	2 x 2	256 x 4 x 4
Flatten	256 x 4 x 4	-	-	-	4096
Fc1	4096	-	-	1 x 1	512
Fc2	512	-	-	1 x 1	10

```python
class CNN2(nn.Module):
    def __init__(self):
        super(CNN2, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 4 * 4, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

TRAINING FUNCTION

TUJUAN FUNGSI TRAIN_MODEL

- Fungsi ini digunakan untuk melatih model CNN menggunakan data training dan validasi.
- Dilengkapi dengan:
  - Penghitungan akurasi dan loss tiap epoch.
  - Early stopping jika validasi loss tidak membaik.
  - Penyimpanan model terbaik berdasarkan validasi loss.

PROSES UTAMA YANG DILAKUKAN

a. Inisialisasi

- Variabel untuk menyimpan loss & akurasi.
- Menyimpan best validation loss awal.
- Timer untuk menghitung waktu proses training.

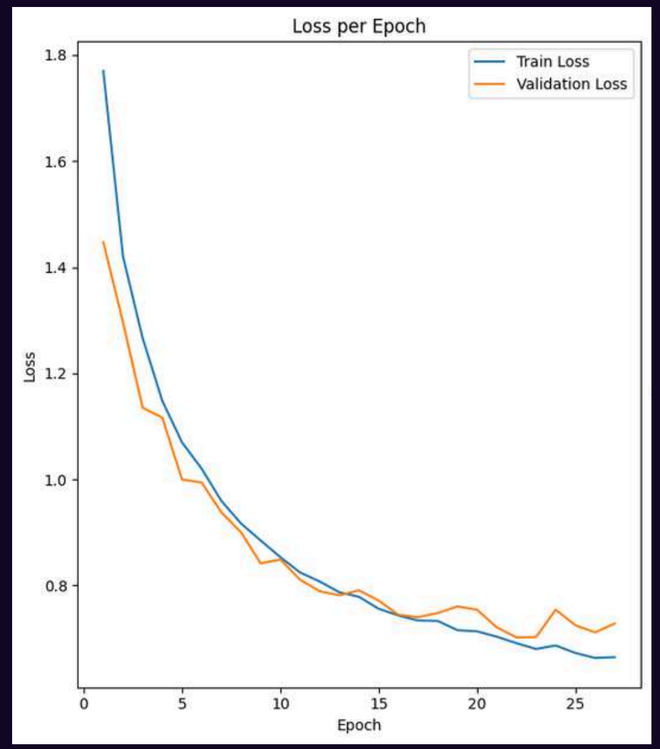
b. Per-Epoch Training Loop

1. Set model ke mode training (model.train()).
2. Loop batch:
   - Kirim gambar & label ke device (CPU/GPU).
   - Forward pass: prediksi output dari model.
   - Hitung loss.
   - Backward pass: update bobot model (optimizer.step()).
3. Hitung total loss dan akurasi untuk data training.

c. Per-Epoch Validation

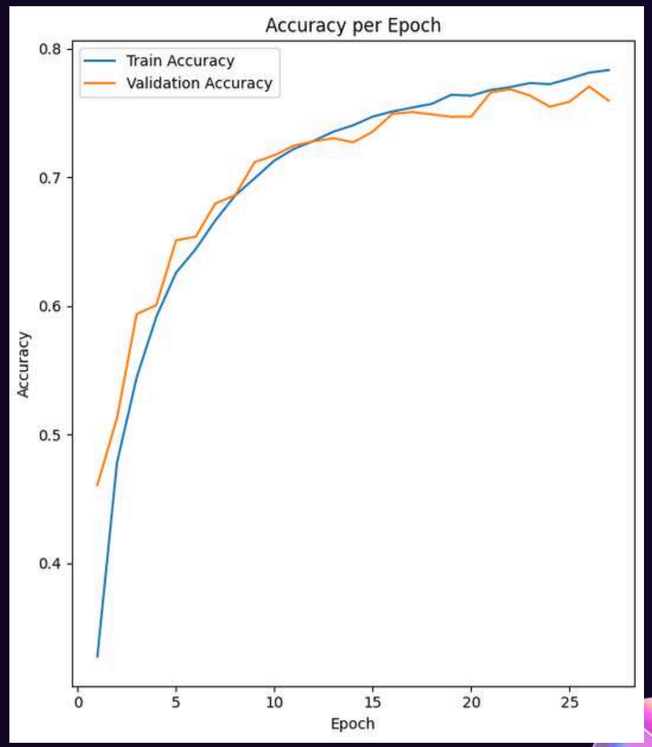
1. Set model ke mode evaluasi (model.eval()).
2. Nonaktifkan gradient computation (with torch.no_grad()).
3. Loop batch:
   - Hitung prediksi dan loss validasi.
4. Simpan loss dan akurasi validasi.

PLOT TRAIN VS VALIDATION LOSS AND ACCURACY ALEXNET



- Train Loss menurun secara konsisten hingga stabil di sekitar 0.7.
- Validation juga menurun hingga sekitar epoch ke-13, lalu mulai konstan.

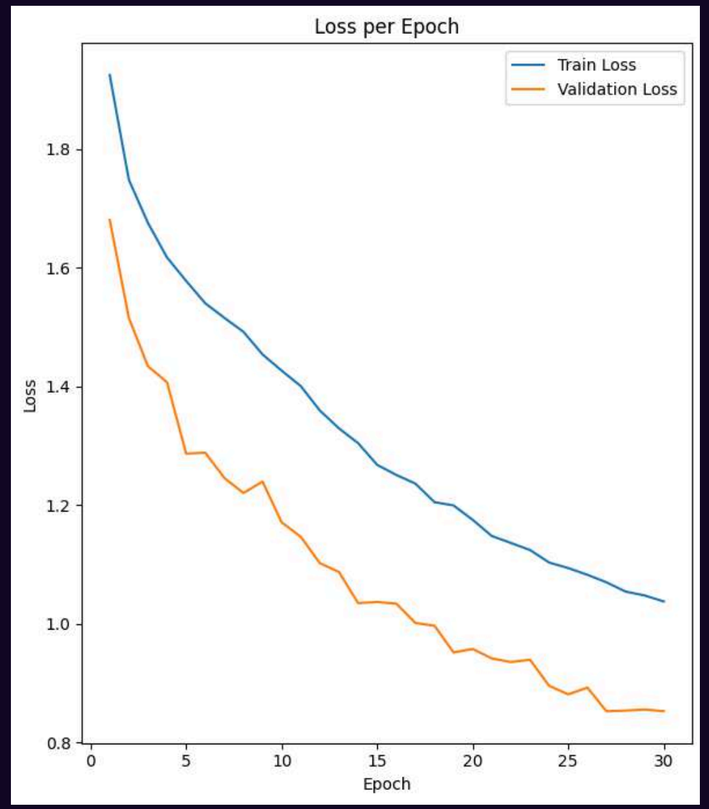
- Model berhasil belajar dengan baik di awal.
- Namun mulai sekitar epoch ke-15, validation loss tidak konsisten membaik, mengindikasikan potensi overfitting ringan (model terlalu fokus pada data train).



- Train Accuracy meningkat tajam dari 0.3 → 0.78.
- Validation Accuracy meningkat cepat sampai sekitar epoch ke-10, lalu stabil di kisaran 0.75–0.77.

- Model cukup baik dalam generalisasi: selisih akurasi train dan validasi tidak besar.
- Tidak terlihat underfitting: model mampu mengenali pola dari data.

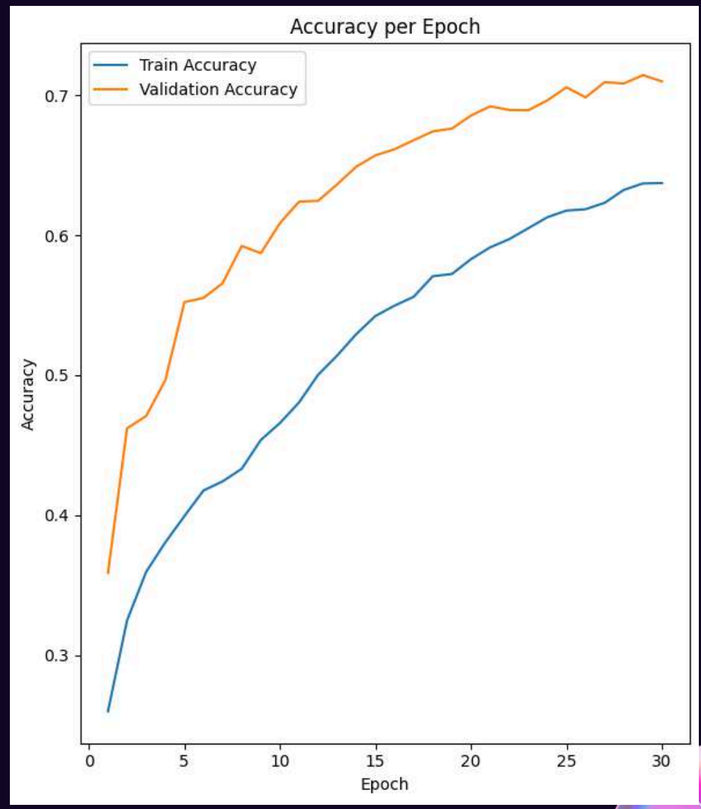
PLOT TRAIN VS VALIDATION LOSS AND ACCURACY CNN-1



- Train Loss Menurun stabil, tapi masih tinggi di akhir epoch (sekitar 1.0).
- Validation loss Lebih rendah dari train loss, turun cepat di awal, kemudian terus menurun hingga sekitar 0.85.

Masalah:

- Model belum cukup kompleks untuk menangani data train dengan baik.



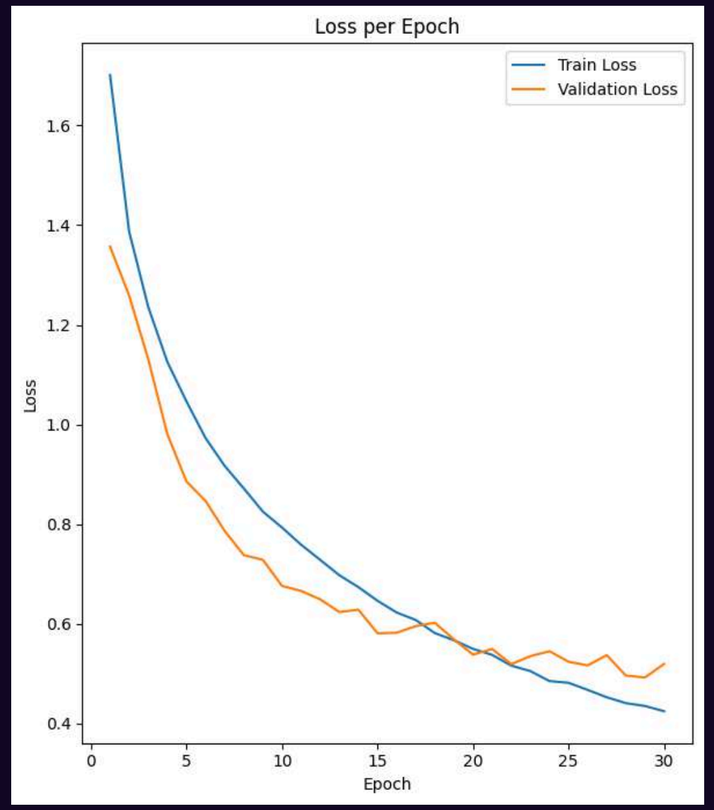
- Train Accuracy Naik lambat dan belum menyentuh 0.65 meski sudah 30 epoch.
- Validation Accuracy Naik cepat hingga menyentuh ~0.71 dan stabil.

- Validasi akurasi lebih tinggi dari train akurasi.

Artinya model underfitting ringan;

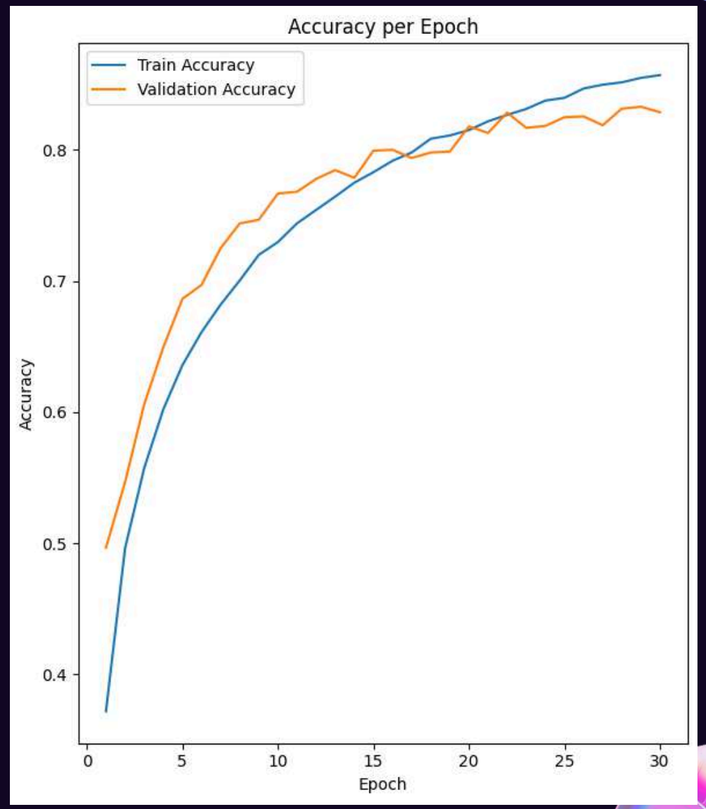
- Model terlalu sederhana.
- Perlu arsitektur yang lebih dalam/kompleks.
- Atau terlalu banyak regularisasi.

PLOT TRAIN VS VALIDATION LOSS AND ACCURACY CNN-2



- Train Loss Menurun stabil dari 1.7 ke ~0.42.
- Validation loss Turun lebih cepat, sempat lebih rendah dari train, lalu stabil di kisaran 0.5–0.55.

- Train Accuracy Naik tajam dari 0.35 ke hampir 0.87.
- Validation Accuracy Naik cepat hingga ~0.84, kemudian stagnan.

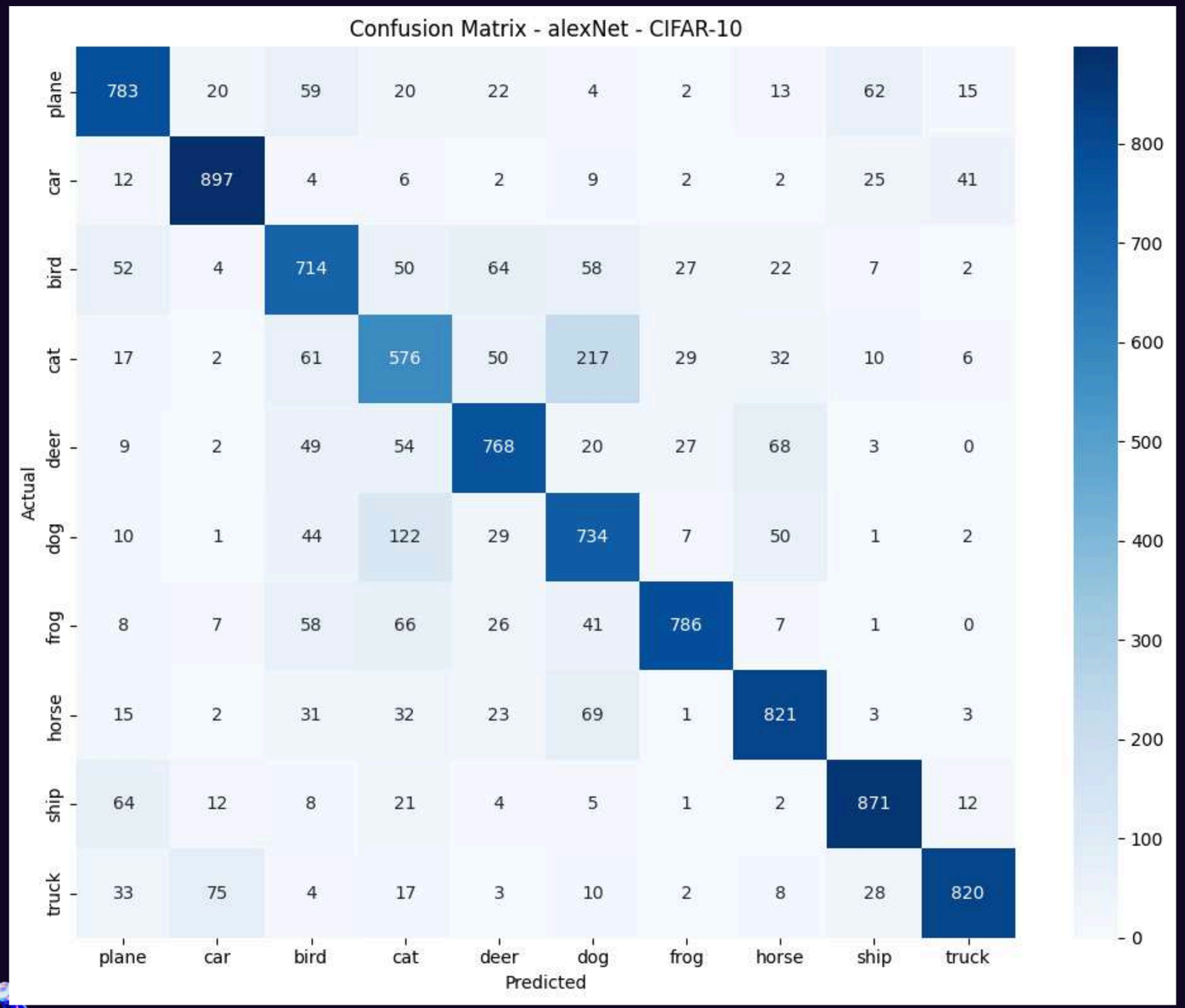


- Model belajar dengan sangat baik.
- Train dan validation sangat dekat, akurasi validasi stabil tinggi (sekitar 84%).
- Tidak ada overfitting atau underfitting mencolok.
- Model paling seimbang dari ketiganya sejauh ini.

PERBANDINGAN LOG TRAIN 3 MODEL

Perbandingan Log Training Model			
Aspek	**AlexNet**	**CNN1**	**CNN2**
Train Accuracy	~0.78	~0.64	~0.87
Val Accuracy	~0.76–0.77	~0.71	~0.84
Train Loss	Rendah, stabil	Tinggi, tetap >1	Turun stabil ke 0.42
Val Loss	Sedikit lebih tinggi dari train	Lebih rendah dari train	Sangat dekat dengan train (~0.5)
Overfitting	Tidak signifikan	Tidak, malah underfit	Tidak
Stabilitas	Stabil	Stabil, tapi kurang optimal	Sangat stabil
Kesimpulan	Model kuat, bisa ditingkatkan	Model terlalu sederhana atau terlalu ketat	Paling seimbang & performa terbaik

EVALUASI MODEL ALEXNET

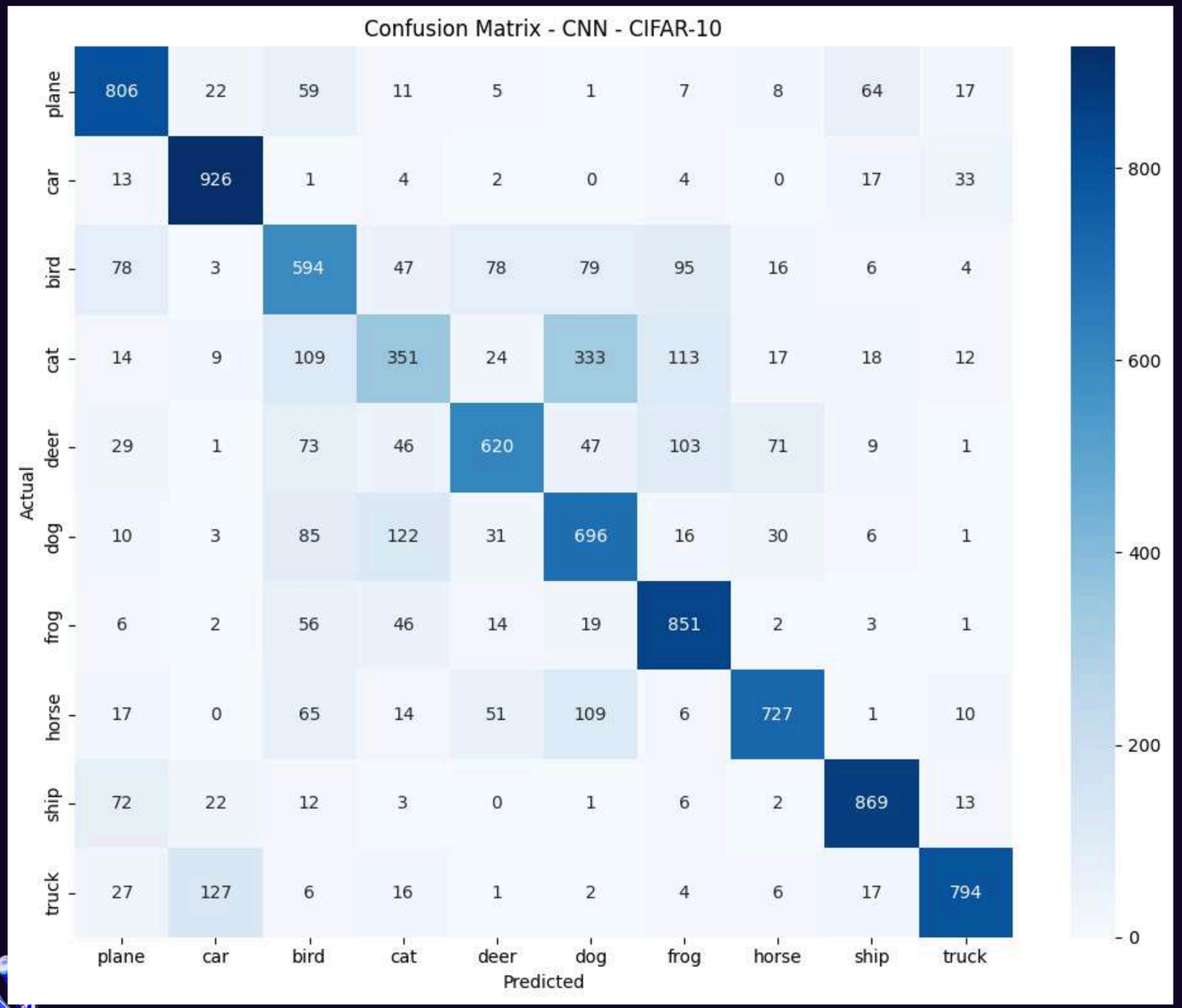



```
Accuracy                0.777000
Precision (macro)       0.781340
Recall (macro)          0.777000
F1 Score (macro)        0.778149
dtype: float64
```

```
Accuracy model alexNet on the 10000 test images: 77.70%
Accuracy of plane: 78.30%
Accuracy of car: 89.70%
Accuracy of bird: 71.40%
Accuracy of cat: 57.60%
Accuracy of deer: 76.80%
Accuracy of dog: 73.40%
Accuracy of frog: 78.60%
Accuracy of horse: 82.10%
Accuracy of ship: 87.10%
Accuracy of truck: 82.00%
```

Model ini cukup baik, dengan performa seimbang antar metrik (tidak ada gap besar antara precision dan recall). Namun, masih bisa ditingkatkan terutama di kelas-kelas sulit.

EVALUASI MODEL CNN-1

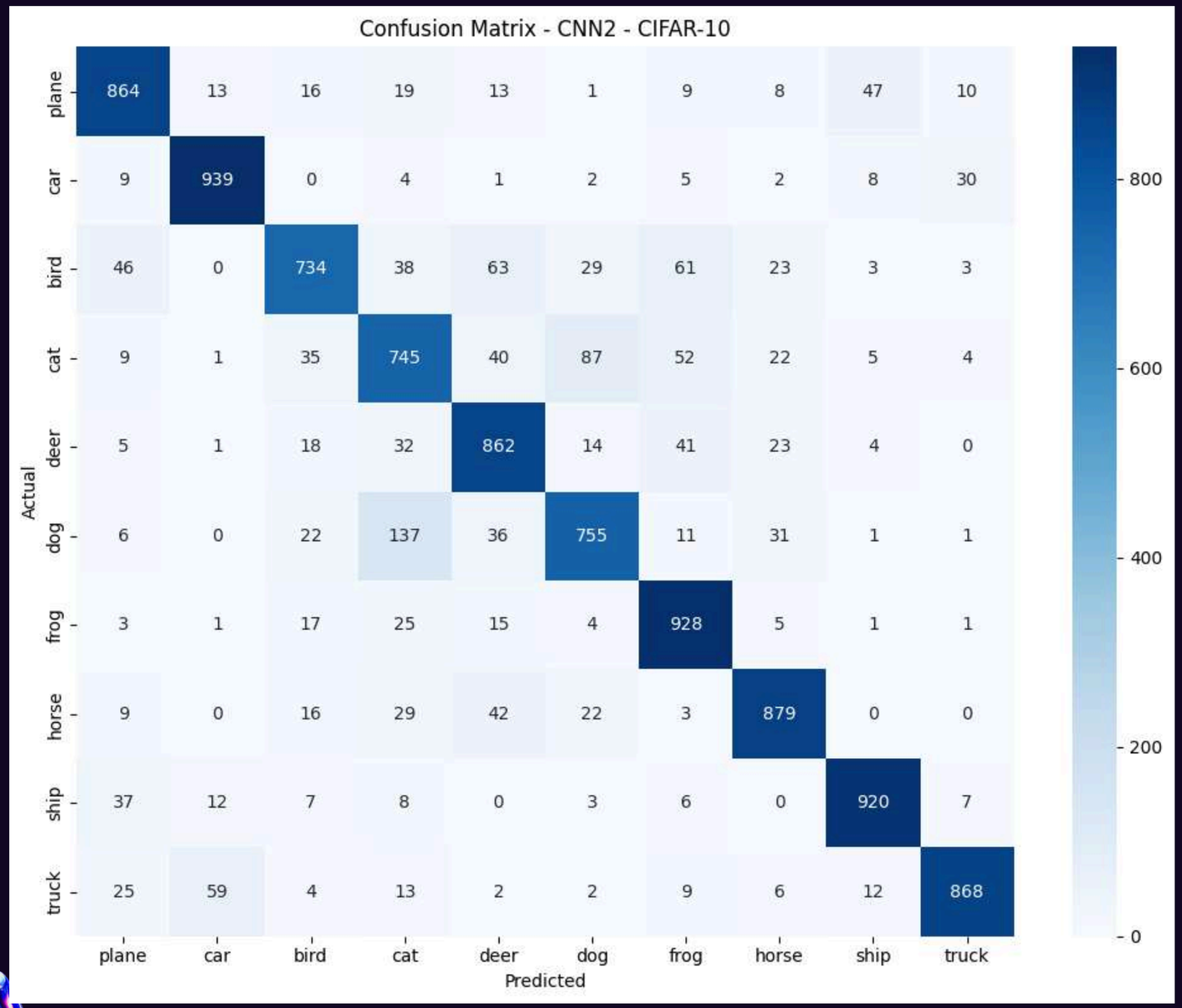



```
Accuracy                  0.723400
Precision (macro)         0.725581
Recall (macro)            0.723400
F1 Score (macro)          0.719334
dtype: float64
```

```
Accuracy model CNN on the 10000 test images: 72.34%
Accuracy of plane: 80.60%
Accuracy of car: 92.60%
Accuracy of bird: 59.40%
Accuracy of cat: 35.10%
Accuracy of deer: 62.00%
Accuracy of dog: 69.60%
Accuracy of frog: 85.10%
Accuracy of horse: 72.70%
Accuracy of ship: 86.90%
Accuracy of truck: 79.40%
```

Performa model CNN ini cukup baik tapi sedikit lebih rendah dibanding AlexNet sebelumnya (yang mencapai 77.7%).

EVALUASI MODEL CNN-2



Accuracy	0.849400
Precision (macro)	0.850969
Recall (macro)	0.849400
F1 Score (macro)	0.849044
dtype: float64	

```
Accuracy model CNN2 on the 10000 test images: 84.94%
Accuracy of plane: 86.40%
Accuracy of car: 93.90%
Accuracy of bird: 73.40%
Accuracy of cat: 74.50%
Accuracy of deer: 86.20%
Accuracy of dog: 75.50%
Accuracy of frog: 92.80%
Accuracy of horse: 87.90%
Accuracy of ship: 92.00%
Accuracy of truck: 86.80%
```

model ini adalah model yang paling baik di antara semua model yang diterapkan ke data CIFAR-10

PERBANDINGAN HASIL EVALUASI KE-TIGA MODEL

Akurasi Per Kelas			
Label	**AlexNet**	**CNN-1**	**CNN-2**
Plane	**86.40%**	80.60%	**86.40%**
Car	93.10%	92.60%	**93.90%**
Bird	69.40%	59.40%	**73.40%**
Cat	57.60%	35.10%	**74.50%**
Deer	80.10%	62.00%	**86.20%**
Dog	**77.30%**	69.60%	75.50%
Frog	89.00%	85.10%	**92.80%**
Horse	85.50%	72.70%	**87.90%**
Ship	91.20%	86.90%	**92.00%**
Truck	83.80%	79.40%	**86.80%**

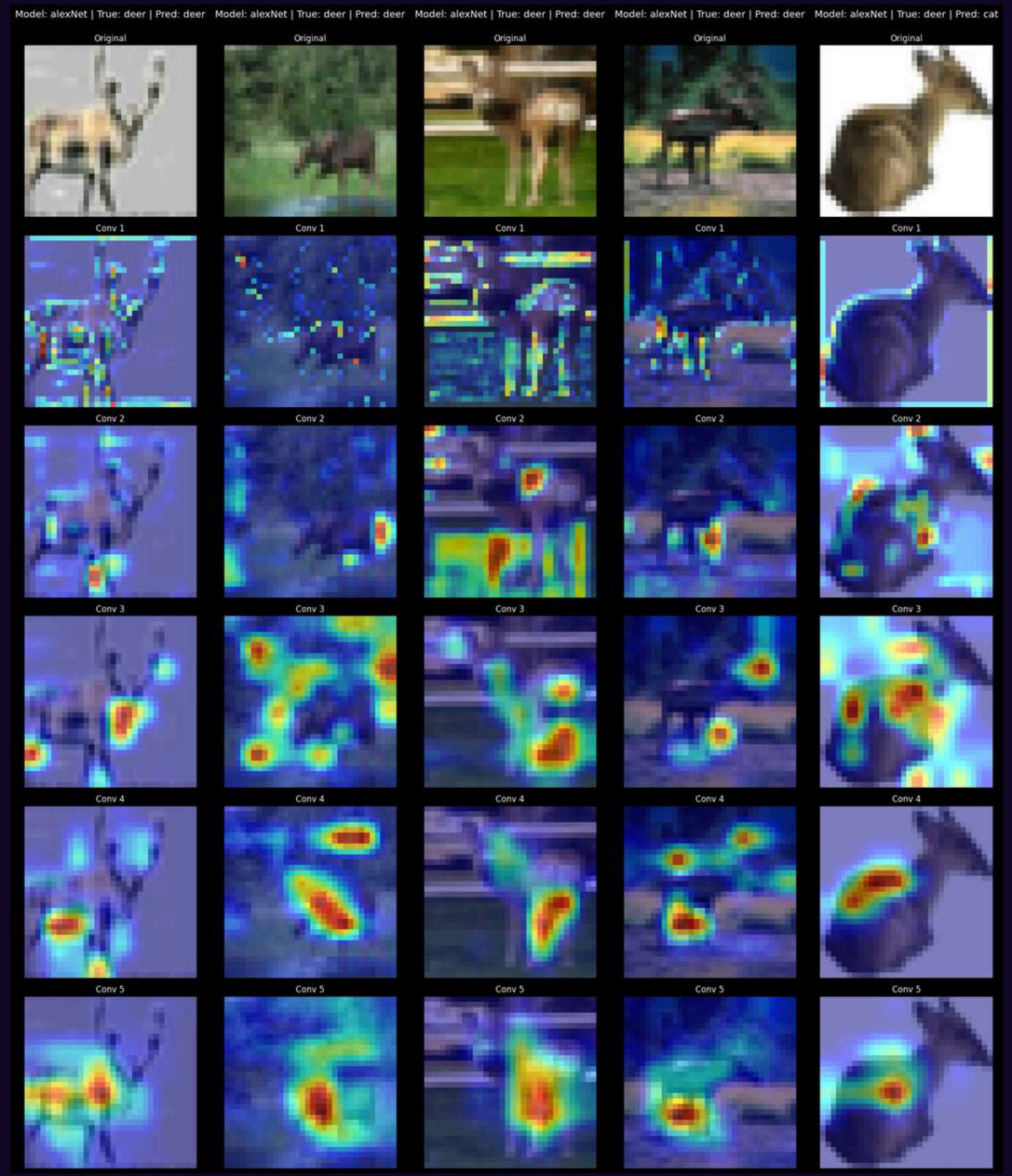
Perbandingan Kinerja Global Model			
Metrik	**AlexNet**	**CNN-1**	**CNN-2**
Accuracy	77.70%	72.34%	**84.94%**
Precision (macro)	**77.95%**	72.56%	72.56%
Recall (macro)	**77.70%**	72.34%	72.34%
F1 Score (macro)	**77.81%**	71.93%	71.93%

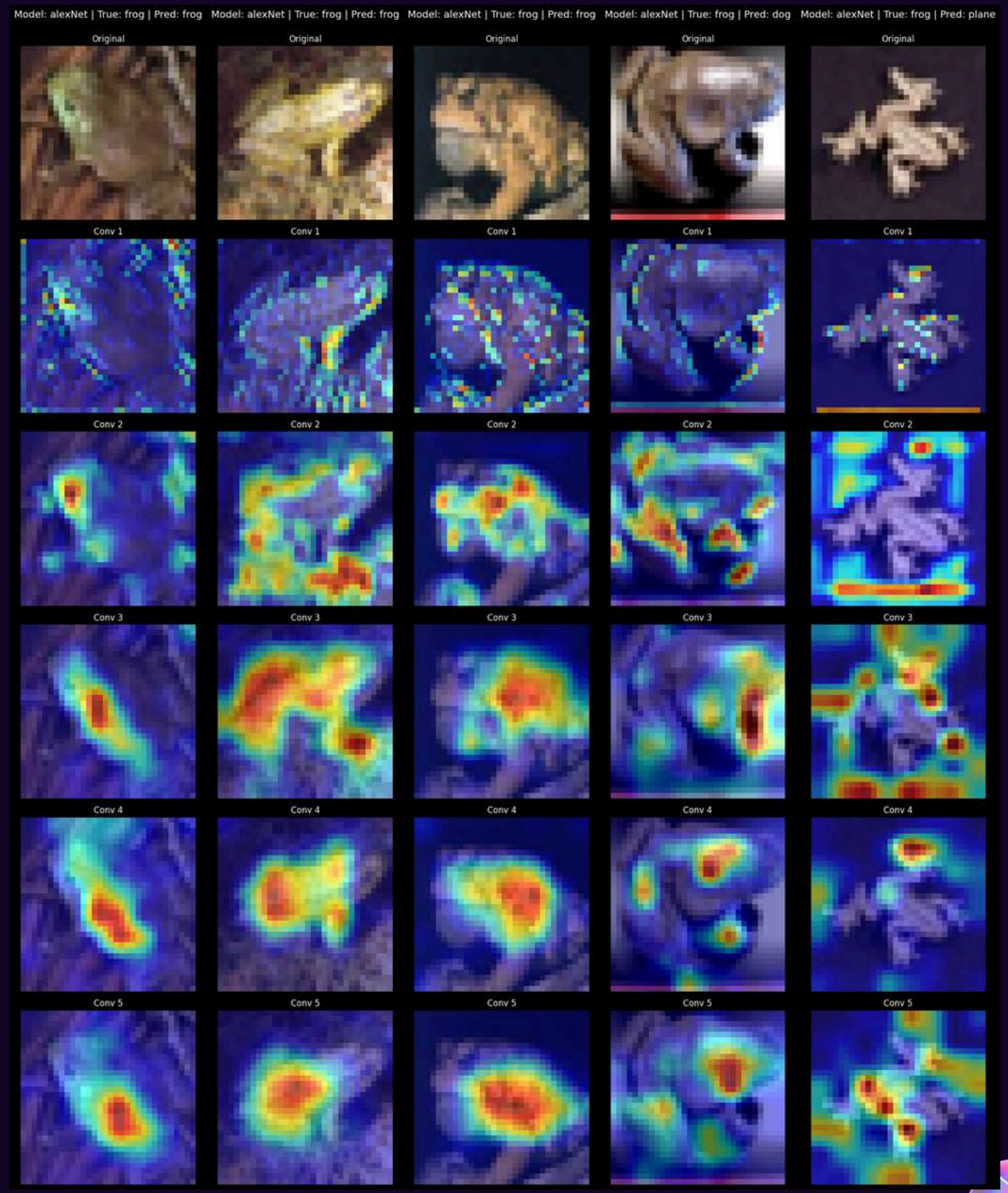
Kesimpulan		
Model	**Kekuatan**	**Kelemahan**
CNN1	Struktur ringan	Akurasi dan f1 rendah, buruk di kelas sulit
AlexNet	Seimbang, macro F1 tertinggi	Belum optimal untuk kelas cat, bird
CNN2	Akurasi tertinggi (84.94%), sangat baik di cat, bird, frog	Macro F1 stagnan → kemungkinan bias kelas

CNN2 unggul dalam hampir semua kelas, termasuk yang sebelumnya lemah seperti cat dan bird. Peningkatan akurasi signifikan ini menunjukkan CNN2 berhasil mengatasi sebagian besar error prediksi dari model lain.

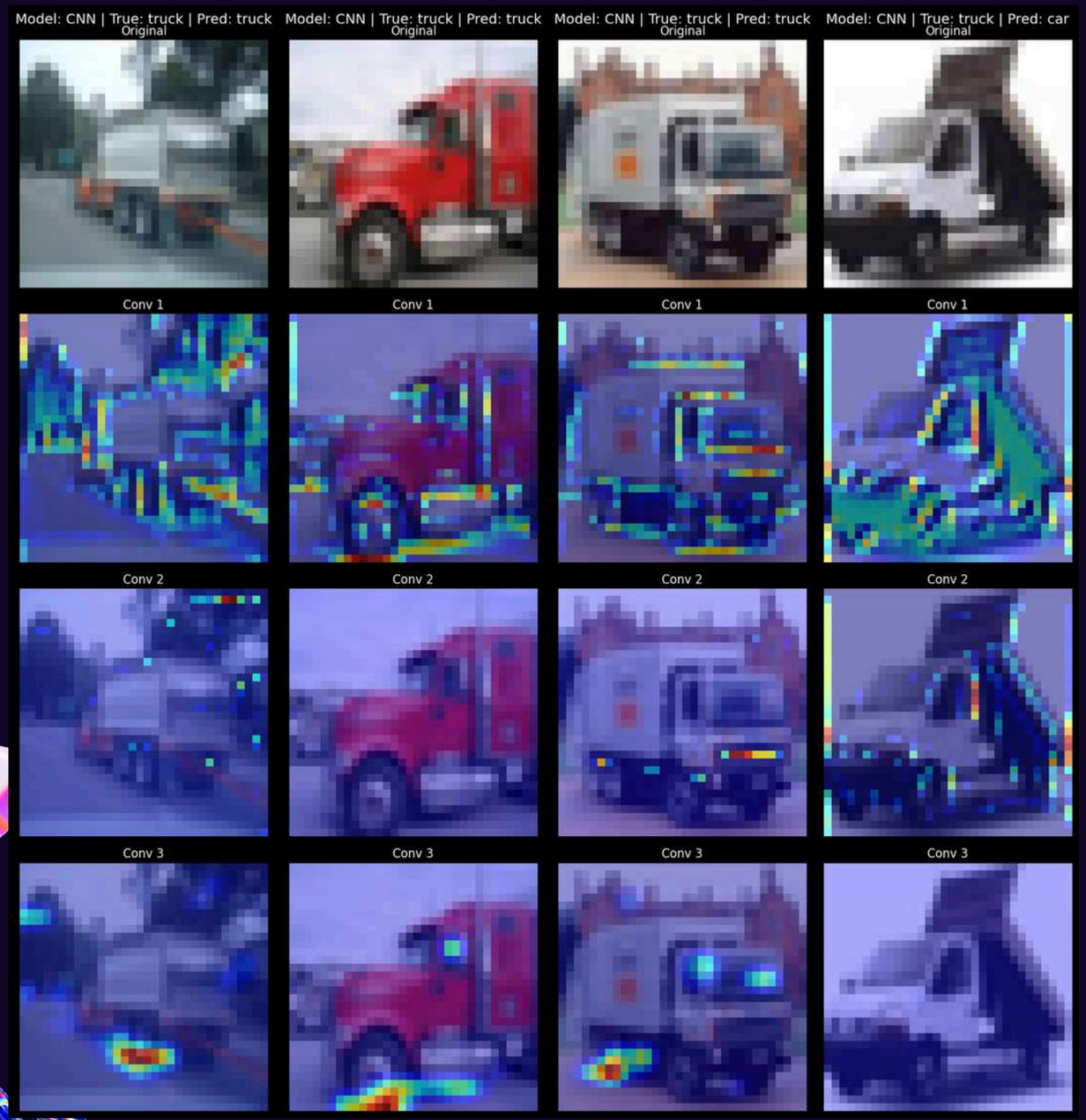
ANALISIS FEATURE MAPS DENGAN GRAD-CAM

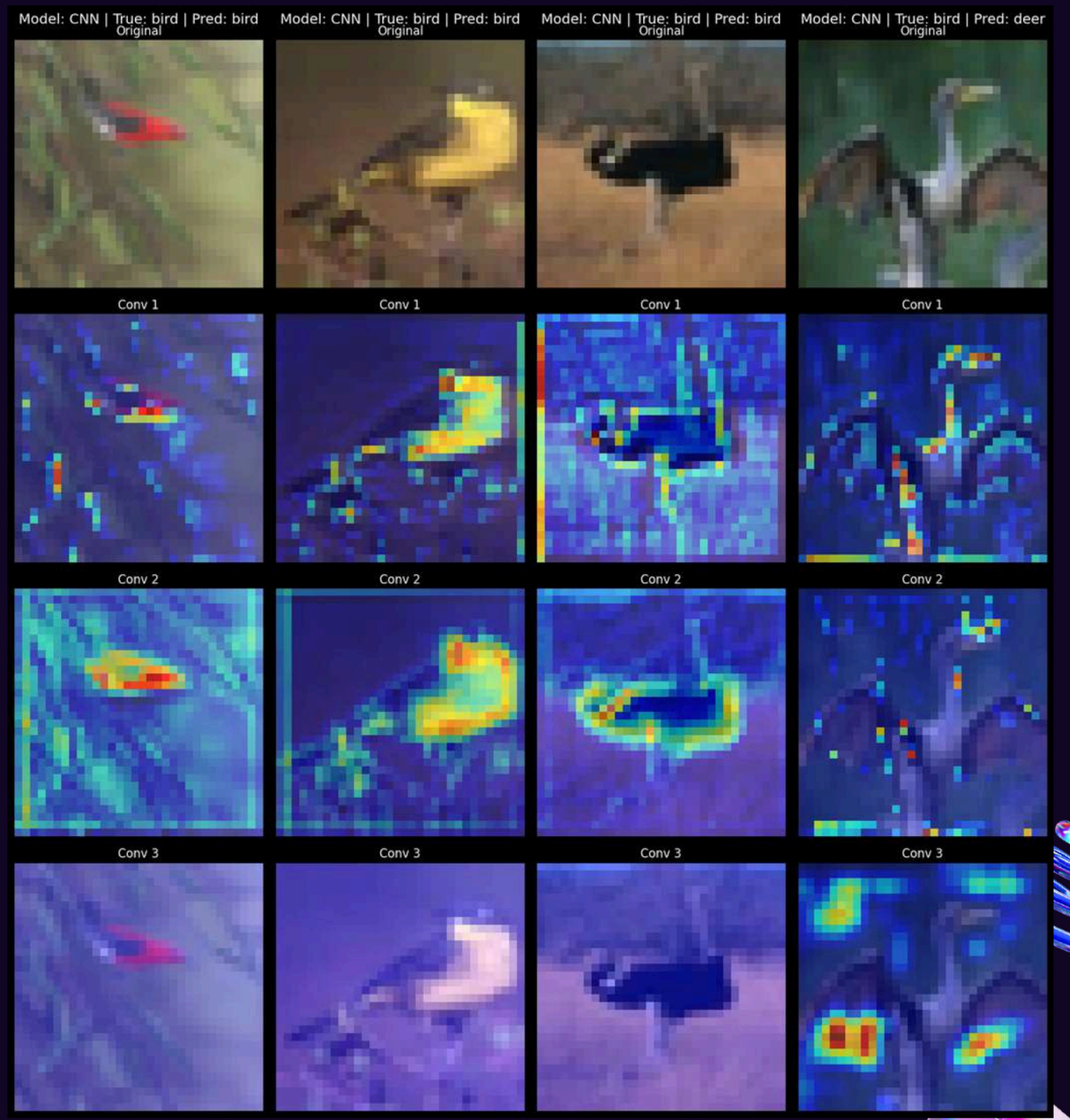
ALEXNET


Model: alexNet | True: deer | Pred: deer
Model: alexNet | True: deer | Pred: deer
Model: alexNet | True: deer | Pred: deer
Model: alexNet | True: deer | Pred: deer
Model: alexNet | True: deer | Pred: cat
Original
Conv 1
Conv 2
Conv 3
Conv 4
Conv 5

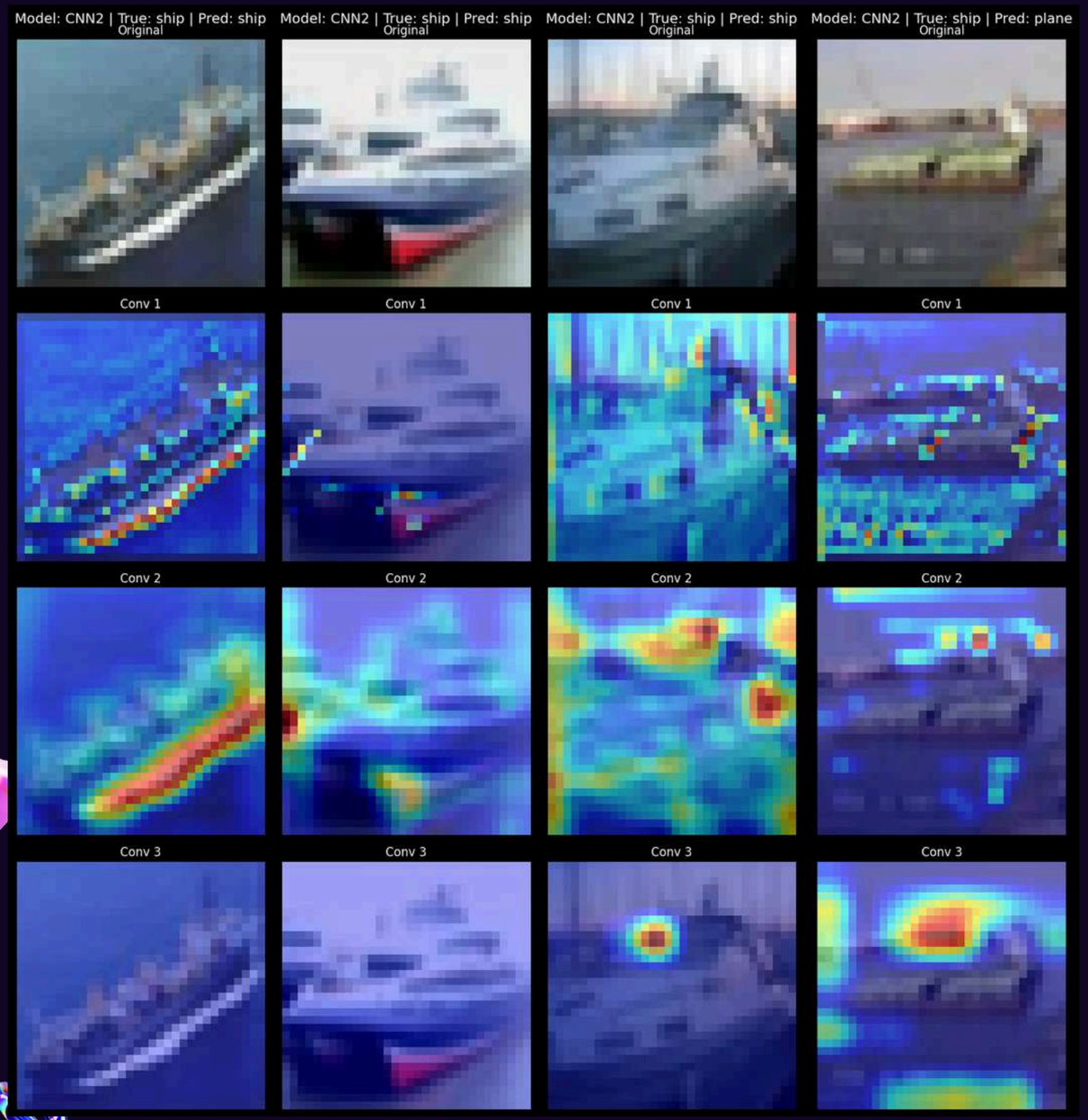

Model: alexNet | True: frog | Pred: frog
Model: alexNet | True: frog | Pred: frog
Model: alexNet | True: frog | Pred: frog
Model: alexNet | True: frog | Pred: dog
Model: alexNet | True: frog | Pred: plane
Original
Conv 1
Conv 2
Conv 3
Conv 4
Conv 5

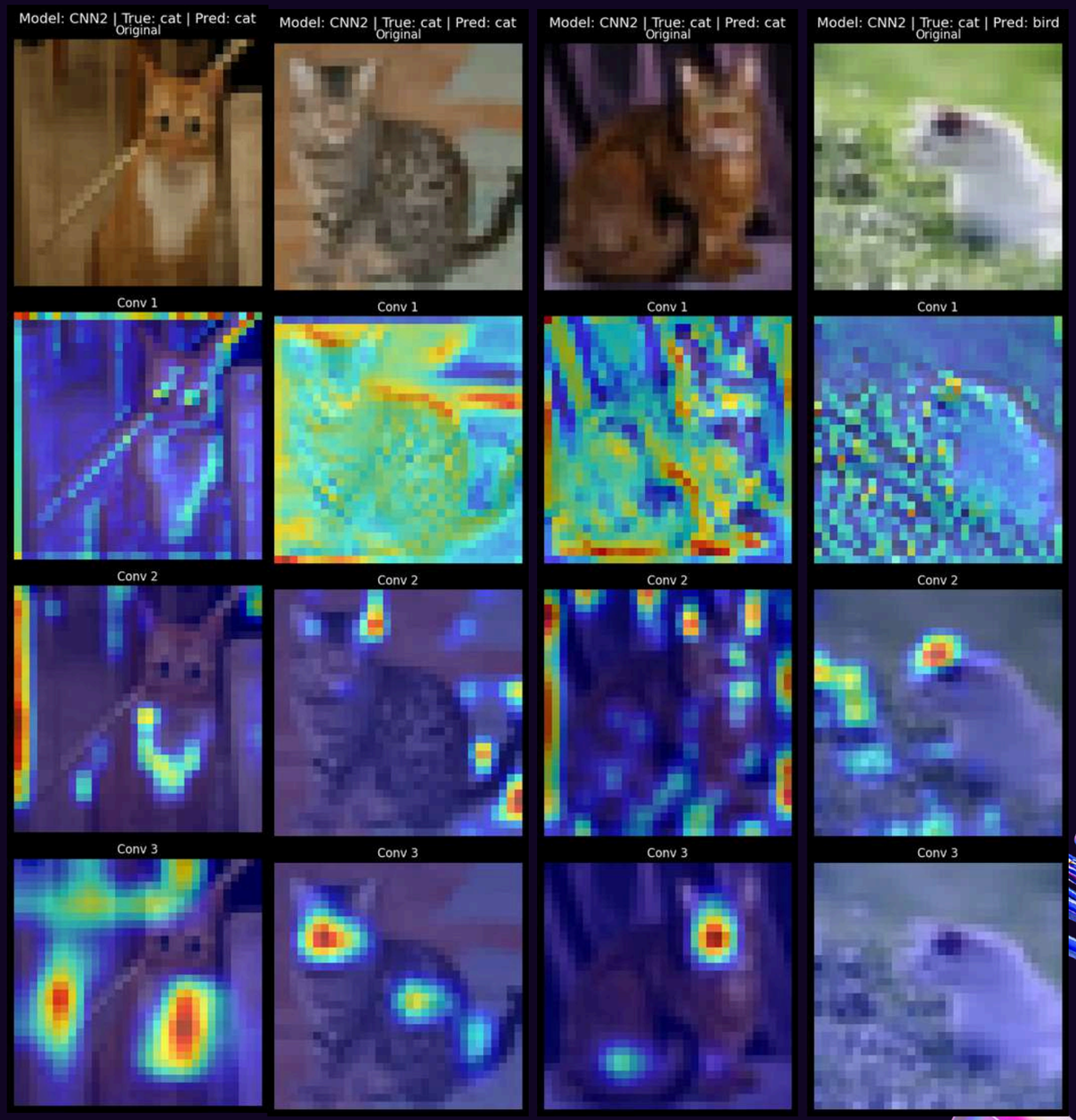
CNN-1


Model: CNN | True: truck | Pred: truck
Original
Model: CNN | True: truck | Pred: truck
Original
Model: CNN | True: truck | Pred: truck
Original
Model: CNN | True: truck | Pred: car
Original
Conv 1
Conv 1
Conv 1
Conv 1
Conv 2
Conv 2
Conv 2
Conv 2
Conv 3
Conv 3
Conv 3
Conv 3

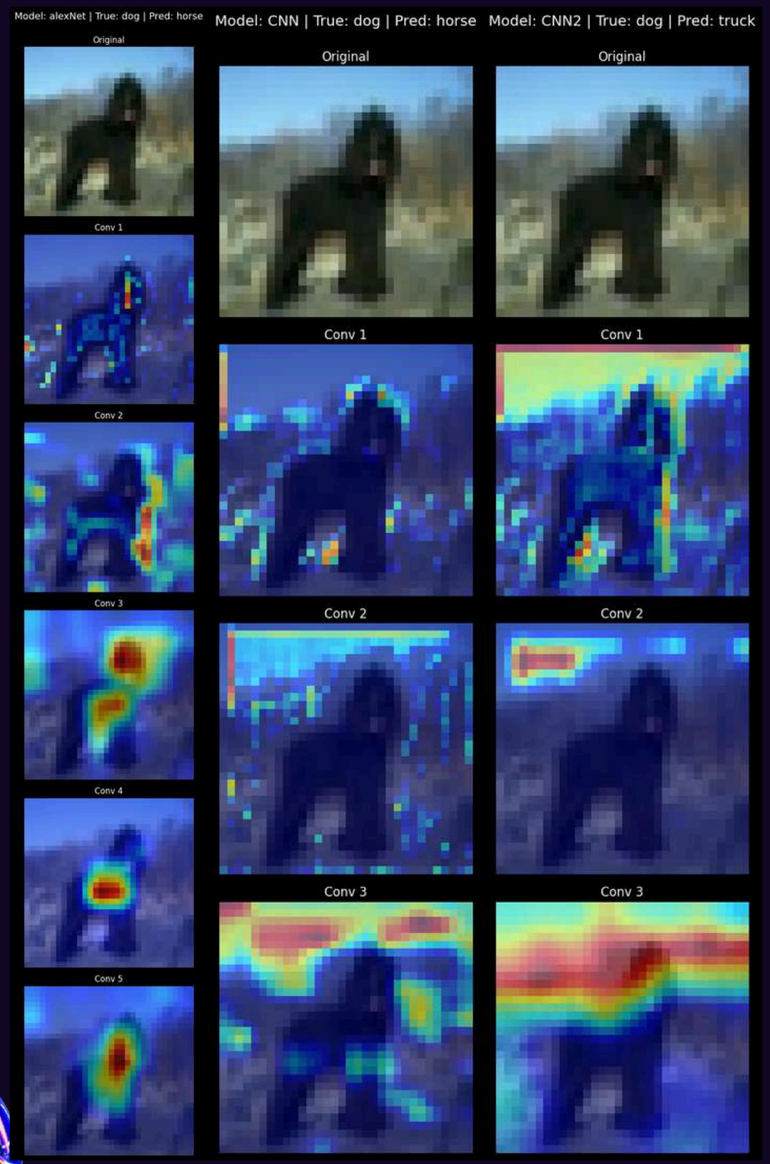

Model: CNN | True: bird | Pred: bird
Original
Model: CNN | True: bird | Pred: bird
Original
Model: CNN | True: bird | Pred: bird
Original
Model: CNN | True: bird | Pred: deer
Original
Conv 1
Conv 1
Conv 1
Conv 1
Conv 2
Conv 2
Conv 2
Conv 2
Conv 3
Conv 3
Conv 3
Conv 3

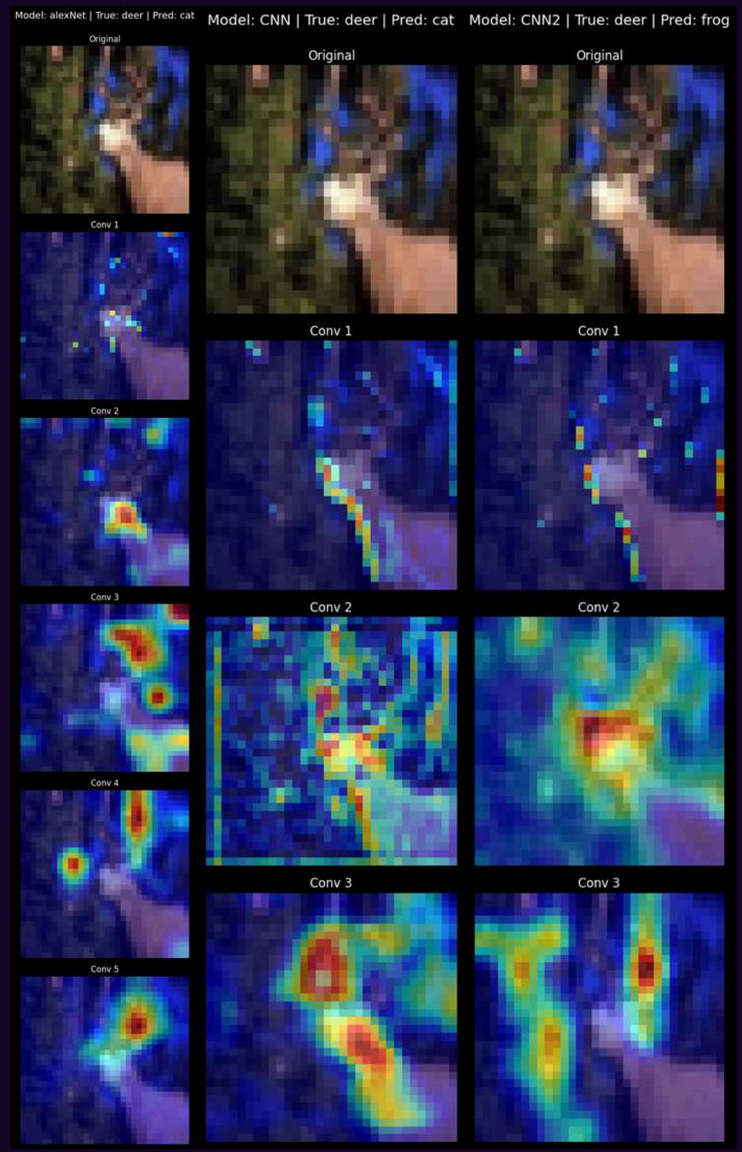
CNN-2

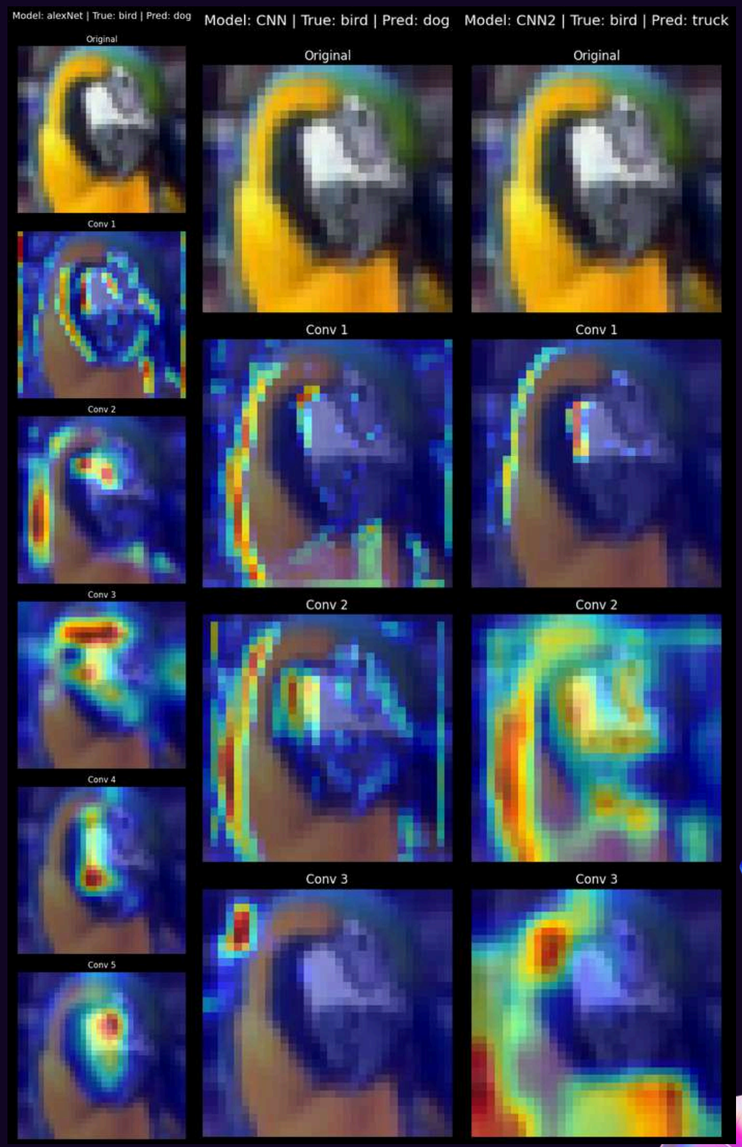

Model: CNN2 | True: ship | Pred: ship
Original
Conv 1
Conv 2
Conv 3
Model: CNN2 | True: ship | Pred: ship
Original
Conv 1
Conv 2
Conv 3
Model: CNN2 | True: ship | Pred: ship
Original
Conv 1
Conv 2
Conv 3
Model: CNN2 | True: ship | Pred: plane
Original
Conv 1
Conv 2
Conv 3


Model: CNN2 | True: cat | Pred: cat
Original
Conv 1
Conv 2
Conv 3
Model: CNN2 | True: cat | Pred: cat
Original
Conv 1
Conv 2
Conv 3
Model: CNN2 | True: cat | Pred: cat
Original
Conv 1
Conv 2
Conv 3
Model: CNN2 | True: cat | Pred: bird
Original
Conv 1
Conv 2
Conv 3

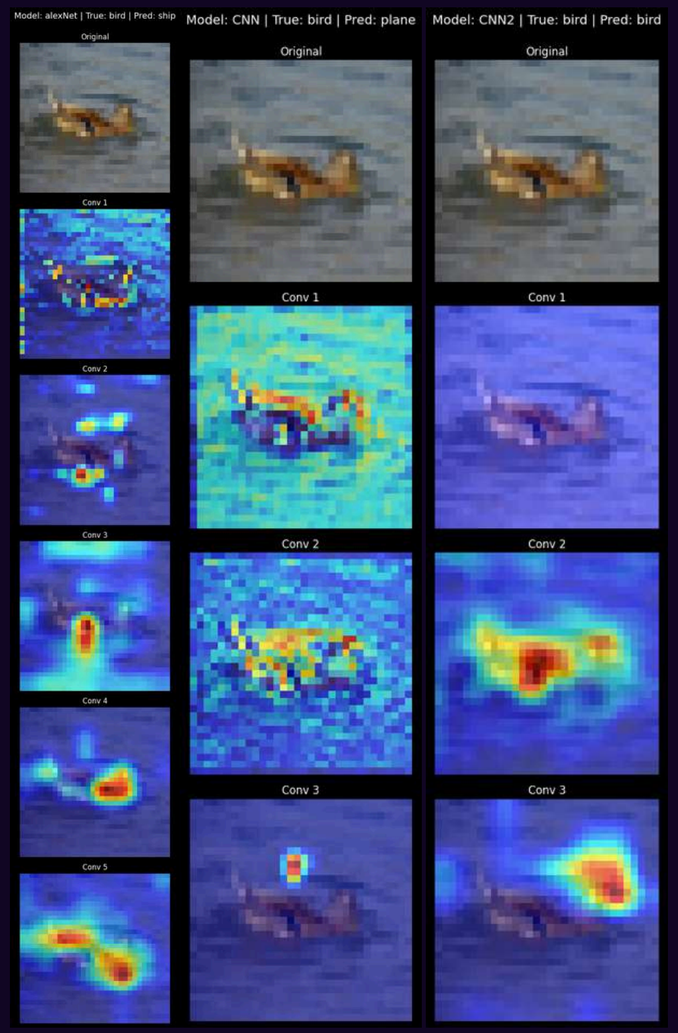
GAMBAR-GAMBAR YANG SALAH DIPREDIKSI OLEH KETIGA MODEL

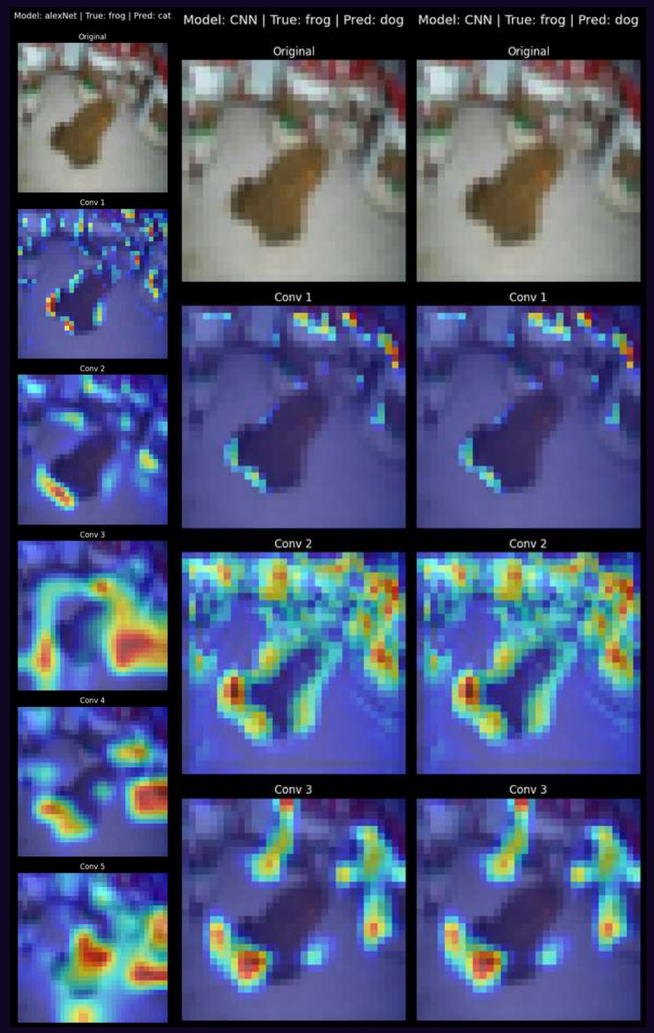


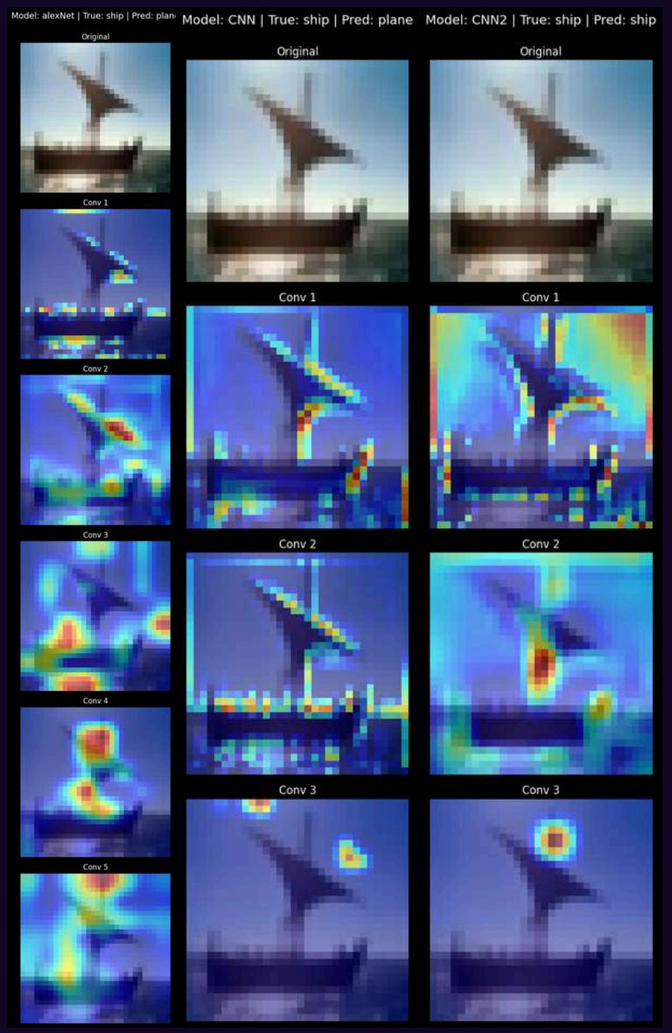




GAMBAR-GAMBAR YANG BENAR DIPREDIKSI OLEH CNN-2 NAMUN SALAH DIPREDIKSI OLEH CNN-1 DAN ALEXNET







KESIMPULAN

CNN2 adalah model terbaik dalam eksperimen ini, karena:

- **Arsitektur lebih dalam & stabil**
- **Performa evaluasi yang unggul**
- **Interpretasi visual Grad-CAM yang meyakinkan**


TERIMAKASIH