Inception, *ResNet* and *DenseNet*

Andi Ilhamsyah Idris
Computer Science
Department of Mathematics
Hasanuddin University

Dalam technical report ini, kita akan mengimplementasikan, membahas, dan membandingkan varian arsitektur *CNN modern*. Ada banyak arsitektur yang berbeda telah diusulkan selama beberapa tahun terakhir. Beberapa yang paling berdampak, dan masih relevan hingga saat ini seperti *GoogleNet* / Arsitektur *Inception* , Pemenang ILSVRC 2014, ResNet, Pemenang ILSVRC 2015 dan DenseNet, Penghargaan best paper CVPR 2017. Ketiga arsitektur tersebut menjadi *state-of-the-art* saat diusulkan dan menjadi fondasi dari sebagian besar arsitektur-arsitektur CNN saat ini. Oleh karena itu, penting untuk memahami arsitektur ini secara detail dan mempelajari cara mengimplementasikannya.

Pertama, kita akan mengimport *library* nya terlebih dahulu

```
## Standard libraries
import os
import numpy as np
import random
from PIL import Image
from types import SimpleNamespace

## Imports for plotting
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg', 'pdf') # For export
import matplotlib
matplotlib.rcParams['lines.linewidth'] = 2.0
import seaborn as sns
sns.reset_orig()

## PyTorch
import torch
import torch.nn as nn
import torch.utils.data as data
import torch.optim as optim
# Torchvision
import torchvision
from torchvision.datasets import CIFAR10
from torchvision import transforms
```

Selanjutnya kita akan membuat variabel path, yaitu `DATASET_PATH` sebagai path untuk menyimpan dataset dan `CHECKPOINT_PATH` sebagai path untuk menyimpan model checkpoint.

Kita juga mendefinisikan fungsi `set_seed` untuk menyetel seed pada semua library yang mungkin berinteraksi dengan kode (seperti pytorch, numpy, random). Hal ini memungkinkan agar proses training dapat direproduksi karena angka random yang dihasilkan seed yang sama akan selalu sama terlepas dari kapan dan di mana prosesnya terjadi.

```
# Path folder dimana datasets didownload
DATASET_PATH = "../data"
# Path folder dimana models akan disimpan
CHECKPOINT_PATH = "../saved_models/"

# Fungsi untuk mengatur seed
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
set_seed(42)

# Memastikan semua operasi deterministik dengan GPU untuk reproduktivitas
torch.backends.cudnn.determinstic = True
torch.backends.cudnn.benchmark = False

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
```

Kita telah memiliki *pre-trained model* dan *tensorboard* sebelumnya, jadi kita akan mendownload file *pre-trained model* dan tensorboard terlebih dahulu.

```
import urllib.request
from urllib.error import HTTPError
# Github URL dimana models disimpan
base_url = "https://raw.githubusercontent.com/phlippe/saved_models/main/tutorial5/"
# File yang didownload
pretrained_files = ["GoogleNet.ckpt", "ResNet.ckpt", "ResNetPreAct.ckpt", "DenseNet.ckpt",
                    "tensorboards/GoogleNet/events.out.tfevents.googlenet",
                    "tensorboards/ResNet/events.out.tfevents.resnet",
                    "tensorboards/ResNetPreAct/events.out.tfevents.resnetpreact",
                    "tensorboards/DenseNet/events.out.tfevents.densenet"]
# Buat tanda path jika tidak dapat diakses
os.makedirs(CHECKPOINT_PATH, exist_ok=True)

# Setiap file diperiksa jika sudah ada. Jika tidak, file akan diunduh.
for file_name in pretrained_files:
    file_path = os.path.join(CHECKPOINT_PATH, file_name)
    if "/" in file_name:
        os.makedirs(file_path.rsplit("/",1)[0], exist_ok=True)
    if not os.path.isfile(file_path):
        file_url = base_url + file_name
        print(f"Downloading {file_url}...")
        try:
            urllib.request.urlretrieve(file_url, file_path)
        except HTTPError as e:
            print("Something went wrong. Please try to download the file from the GDrive folder, or contact the author with the full output including th
```

Dataset yang akan digunakan untuk membandingkan arsitektur *GoogLeNet*, *ResNet* dan *DenseNet* yaitu dataset *CIFAR10*.

```
train_dataset = CIFAR10(root=DATASET_PATH, train=True, download=True)
```

I. Data *Pre-processing*

Pertama-tama kita akan menghitung rata-rata dan simpangan baku dari data, yang nantinya akan kita gunakan untuk melakukan normalisasi.

```
DATA_MEANS = (train_dataset.data / 255.0).mean(axis=(0,1,2))
DATA_STD = (train_dataset.data / 255.0).std(axis=(0,1,2))

print("Data mean", DATA_MEANS)
print("Data std", DATA_STD)
```

Kita juga akan melakukan augmentasi data, ini akan mengurangi risiko overfitting dan membantu arsitektur CNN lebih general. Secara khusus, kita akan menerapkan dua augmentasi acak.

- Membalik setiap gambar secara horizontal dengan peluang `50%` (*transforms.RandomHorizontalFlip*)
- Me-rescale gambar dalam rentang kecil, lalu melakukan crop *transforms.RandomResizedCrop*. Ini membuat nilai piksel aktual pada gambar berubah sementara konten atau semantik keseluruhan gambar tetap sama.

Kita akan secara acak membagi data latih menjadi data latih dan validasi. Data validasi akan digunakan untuk menentukan *early stopping*. Setelah menyelesaikan pelatihan, barulah kita menguji model pada data uji CIFAR.

```
test_transform = transforms.Compose([transforms.ToTensor(),
                                     transforms.Normalize(DATA_MEANS, DATA_STD)
                                     ])
# Untuk latihan, kita menambahkan sebuah augmentasi, Jaringan terlalu kuat dan akan overfit.
train_transform = transforms.Compose([transforms.RandomHorizontalFlip(),
                                      transforms.RandomResizedCrop((32,32),scale=(0.8,1.0),ratio=(0.9,1.1)),
                                      transforms.ToTensor(),
                                      transforms.Normalize(DATA_MEANS, DATA_STD)
                                     ])
# Memuat set data pelatihan. Kita perlu membaginya menjadi bagian pelatihan dan validasi
# Kita perlu melakukan sedikit trik karena set validasi tidak boleh menggunakan augmentasi.
train_dataset = CIFAR10(root=DATASET_PATH, train=True, transform=train_transform, download=True)
val_dataset = CIFAR10(root=DATASET_PATH, train=True, transform=test_transform, download=True)
set_seed(42)
train_set, _ = torch.utils.data.random_split(train_dataset, [45000, 5000])
set_seed(42)
_, val_set = torch.utils.data.random_split(val_dataset, [45000, 5000])

# Memuat set test
test_set = CIFAR10(root=DATASET_PATH, train=False, transform=test_transform, download=True)

# Kami mendefinisikan satu set pemuat data yang dapat kami gunakan untuk berbagai keperluan nanti.
train_loader = data.DataLoader(train_set, batch_size=128, shuffle=True, drop_last=True, pin_memory=True, num_workers=4)
val_loader = data.DataLoader(val_set, batch_size=128, shuffle=False, drop_last=False, num_workers=4)
test_loader = data.DataLoader(test_set, batch_size=128, shuffle=False, drop_last=False, num_workers=4)
```

Untuk memverifikasi bahwa data benar-benar ternormalisasi, kita dapat melakukan print pada mean dan standar deviasi dari satu batch tunggal. Rata-rata harus mendekati 0 dan simpangan baku mendekati 1 untuk setiap *channel*.

```
imgs, _ = next(iter(train_loader))
print("Batch mean", imgs.mean(dim=[0,2,3]))
print("Batch std", imgs.std(dim=[0,2,3]))
```

Terakhir, mari kita visualisasikan beberapa gambar dari data latih, dan bagaimana tampilannya setelah augmentasi data acak.

```
NUM_IMAGES = 4
images = [train_dataset[idx][0] for idx in range(NUM_IMAGES)]
orig_images = [Image.fromarray(train_dataset.data[idx]) for idx in range(NUM_IMAGES)]
orig_images = [test_transform(img) for img in orig_images]

img_grid = torchvision.utils.make_grid(torch.stack(images + orig_images, dim=0), nrow=4, normalize=True, pad_value=0.5)
img_grid = img_grid.permute(1, 2, 0)

plt.figure(figsize=(8,8))
plt.title("Augmentation examples on CIFAR10")
plt.imshow(img_grid)
plt.axis('off')
plt.show()
plt.close()
```

II. *PYTORCH LIGHTNING*

Kita akan menggunakan *PyTorch Lightning* untuk proses *train* dan *evaluate*. *PyTorch Lightning* adalah *wrapper framework* untuk *pytorch* yang menyederhanakan kode yang diperlukan untuk melatih, mengevaluasi, dan menguji model. *Pytorch Lightning* juga menangani *logging* ke *TensorBoard*, *toolkit* visualisasi untuk eksperimen *Machine Learning* serta memudahkan dalam penggunaan *callback*, seperti model *checkpoint*.

```
# PyTorch Lightning
try:
    import pytorch_lightning as pl
except ModuleNotFoundError: # Google Colab tidak menyediakan PyTorch Lightning instalan default. Maka dari itu, kami melakukannya di sini jika perlu
    !pip install --quiet pytorch-lightning>=1.4
    import pytorch_lightning as pl
```

PyTorch Lightning memiliki fungsi-fungsi *high level* agar kita tidak perlu lagi menulis kode *boilerplate* (berulang) dan membuat kita lebih fokus membangun model. Misalnya untuk menyetel *seed*.

```
# Mengatur seed
pl.seed_everything(42)
```

Di *PyTorch Lightning*, model didefinisikan sebagai *class* turunan dari pl.LightningModule (yang mewarisi lagi dari torch.nn.Module). pl.LightningModule mengatur kode kita menjadi 5 bagian utama:

- Inisialisasi (`__init__`), di mana kita menginisialisasikan dan membuat semua parameter/model yang diperlukan.
- *Optimizers* (`configure_optimizers`) tempat kita membuat *optimizers*, penjadwalan *learning rate*, dll.
- *Train Loop* (`training_step`) di mana kita hanya perlu mendefinisikan perhitungan loss untuk satu batch (`loop optimizer.zero_grad()`, loss.backward() dan optimizer.step(), serta setiap logging/ operasi penyimpanan, dilakukan di latar belakang).
- *Validation Loop* (`validation_step`) sama seperti *Train Loop*, kita hanya perlu mendefinisikan perhitungan *loss* untuk satu batch validasi.
- *Test loop* (`test_step`) di mana kita hanya perlu mendefinisikan perhitungan *loss* untuk satu batch tes.

Berikut merupakan contoh pembuatan model *Pytorch Lightning*