Bab 4

Transfer Learning dan Optimasi Model

Setelah mempelajari berbagai teknik regulasi seperti dropout dan L1/L2 regularization, serta strategi optimasi melalui pemilihan optimizer dan tuning hyperparameter, ada pendekatan lain yang semakin populer untuk mengatasi keterbatasan data dan mempercepat proses pelatihan, yaitu Transfer Learning (TL). Teknik ini memungkinkan model memanfaatkan pengetahuan yang telah dipelajari dari dataset lain, sehingga proses pelatihan menjadi lebih efisien dan akurasi model dapat meningkat secara signifikan, terutama ketika data pelatihan terbatas. Pada subbab berikut ini, kita akan membahas konsep dasar, manfaat, serta implementasi TL, termasuk bagaimana menerapkannya menggunakan PyTorch.

4.1 Konsep Dasar *Transfer Learning*

Model *deep learning* dikenal dengan keunggulan dibandingkan dengan *machine learning*. Pada bab sebelumnya, kita telah membahas salah satu penerapan *deep learning* dalam kasus klasifikasi tulisan angka dengan model CNN. Performa dari model CNN itu sangat dipengaruhi oleh arsitekturnya. Semakin kompleks arsitekturnya, kinerjanya cenderung lebih bagus. Sayang-

nya, arsitektur yang kompleks itu memiliki sisi balik yaitu tingkat komputasi yang tinggi dan kebutuhan sumber daya yang lebih tinggi juga.

Selain itu, model *deep learning* itu sifatnya hanya untuk kasus spesifik [1]. Misalnya, dengan dataset MNIST, kita membangun arsitektur model *deep learning* untuk mengenali angka dari tulisan tangan. Apabila model yang kita bangun itu digunakan untuk mengenali objek, seperti kendaraan atau hewan, tentunya tidak bisa.

4.2 Domain dalam *Transfer Learning*

4.2.1 Definisi

Menurut Wang (Wang & Chen, 2023b), domain berarti subjek yang menjalankan proses belajar atau pelatihan. Domain terdiri atas dua bagian, yaitu data dan distribusi yang menghasilkan data. Secara matematis, kita bisa menuliskan domain dengan simbol $\mathcal{D}$ dengan sampel $\mathbf{x}$ dan y. Dalam konsep pembelajaran mesin, $\mathbf{x}$ ini bisa disimbolkan sebagai fitur atau variabel bebas, sedangkan y ini sebagai variabel terikatnya. Selain itu, agar pola dari hubungan $\mathbf{x}$ dan y bisa diketahui, kita modelkan distribusi peluang dengan $P(\mathbf{x}, y)$. Untuk setiap sampel data dari $\mathbf{x}$, kita memiliki $x_i \in \mathcal{X}$ dan $y_i \in \mathcal{Y}$ dengan $\mathcal{X}$ adalah ruang fitur dan $\mathcal{Y}$ sebagai ruang label. Untuk memudahkan pemahaman, mari kita analogikan dengan kasus nyata. Anggaplah $\mathbf{x}$ itu sebagai kumpulan dari citra atau gambar minuman, sedangkan y adalah label dari jenis minuman seperti teh, jus, atau kopi. Dengan kondisi ini, ada kemungkinan bahwa gambar jus lebih banyak muncul daripada kopi atau beberapa gambar teh yang mirip dengan kopi.

[1] Istilah untuk *artifisial intelligence* (AI) seperti ini disebut juga dengan *narrow AI* atau *weak AI*

4.2.2 Jenis Domain

Dalam TL, setidaknya ada dua domain yang terlibat di dalamnya, domain yang kaya dengan informasi dan satunya yang akan digunakan untuk pelatihan. Domain pertama disebut juga sebagai domain sumber, sedangkan domain kedua disebut sebagai domain target. Kedua domain ini bisa disimbolkan dengan $\mathcal{D}_s$ dan $\mathcal{D}_t$, di mana $\mathcal{D}_s$ adalah domain sumber sedangkan $\mathcal{D}_t$ domain target. Suatu metode dalam pembelajaran mesin dikatakan atau termasuk TL jika memenuhi syarat berikut:

1. Ukuran dari ruang fitur berbeda

2. Ukuran dari ruang label berbeda

3. Ukuran ruang fitur dan label sama, tapi distribusi peluang yang berbeda.

Jika tidak memenuhi syarat tersebut, maka ini bukan lagi kasus *transfer learning*.

Secara spesifik, perbedaan ruang fitur, yaitu $\mathcal{X}_s \neq \mathcal{X}_t$, menunjukkan keberadaan dua domain dengan ukuran fitur dan/atau dimensi yang berbeda. Sebagai contoh, ketika ada domain sumber yang merupakan data citra RGB dan domain label/target merupakan data citra grayscale, kita bisa mengatakan bahwa dua domain ini memiliki perbedaan ruang fitur. Contoh untuk perbedaan ruang label, pada kasus klasifikasi citra, domain sumber bertujuan untuk mengelompokkan gambar sapi dan kambing, sedangkan domain target bertujuan untuk mengklasifikasikan bunga. Yang berikutnya contoh untuk perbedaan peluang distribusi yaitu di mana domain sumber adalah kumpulan citra dari sketsa dan domain target adalah kumpulan citra karya seni. Meskipun keduanya sama-sama dalam bentuk RGB tapi memiliki peluang distribusi yang berbeda.

4.2.3 Adaptasi Domain

Pembahasan domain tidak lepas dari studi khusus yang merupakan inti dari TL, yaitu adaptasi domain. Kondisi adaptasi domain muncul ketika domain sumber dan target memiliki ruang fitur dan label yang identik, namun distribusi peluang gabungannya berbeda (sebagaimana dijelaskan pada syarat ketiga dalam Subbab 4.2.2). Definisi dari adaptasi domain adalah pendekatan dalam TL yang digunakan ketika domain sumber dan domain target memiliki ruang fitur dan label yang sama namun memiliki distribusi gabungan yang berbeda. Tujuan dari adaptasi domain tidak lain adalah meningkatkan performa pada domain target sehingga meminimalkan risiko kesalahan prediksi.

Adaptasi domain bisa dikategorikan menjadi tiga:

1. *Supervised Domain Adaptation*(SDA), yaitu di mana domain target yang sudah terlabeli semua, disimbolkan dengan $\mathcal{D}_t = \{\mathbf{x}_j, y_j\}_j^{N_t} = 1$

2. *Semi-supervised Domain Adaptation* (SSDA), yaitu data pada domain target dilabeli sebagian saja, dan sisanya tidak dilabeli. Disimbolkan dengan $\mathcal{D}_t = \mathcal{A} \cup \mathcal{B}$ di mana $\mathcal{A} = \{\mathbf{x}_j, y_j\}_{j=1}^{N_{tb}}$ dan $\mathcal{B} = \{\mathbf{x}_j, y_j\}_{j=1}^{N_{ts}}$. N_{tb} adalah jumlah sampel yang belum dilabeli, sedangkan N_{ts} adalah jumlah data yang sudah dilabeli.

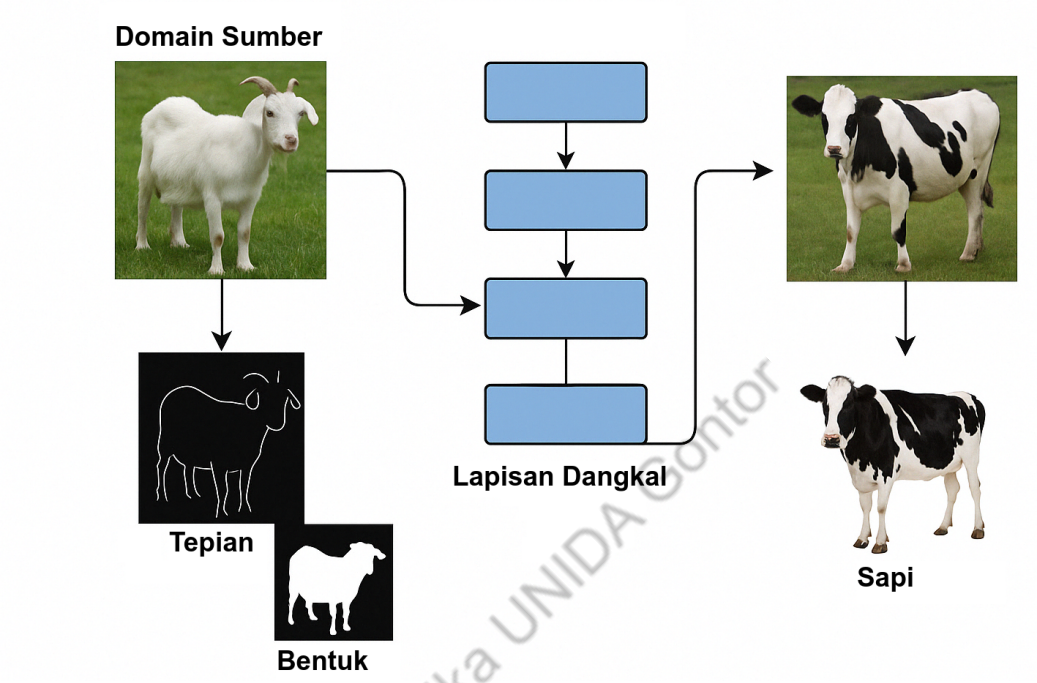
3. *Unsupervised Domain Adaptation* (UDA), yaitu tidak ada pelabelan pada target domain sama sekali. Disimbolkan dengan $\mathcal{D}_t = \{\mathbf{x}_j\}_{j=1}^{N_t}$

Di antara ketiga jenis adaptasi domain, yang paling menantang adalah UDA karena domain target yang sama sekali tidak dilabeli. Hal ini membuat proses pelatihan model menjadi lebih sulit karena tidak tersedia sinyal supervisi untuk mengevaluasi atau memperbaiki kesalahan prediksi secara langsung pada domain target. Selain itu, pencocokan distribusi antara domain sumber dan domain target harus dilakukan secara tidak langsung, misalnya melalui pendekatan adversarial, *self-training*, atau alignment representasi, dan tentunya masing-masing memiliki tantangan tersendiri. Pada prinsipnya, untuk *deep learning* dengan struktur

4.3 Strategi *Transfer Learning*

Sebelum melakukan TL, alangkah baiknya mengenal lebih dulu jaringan syaraf tiruan yang mendalam/*deep learning*. Sebagaimana yang kita ketahui, *deep learning* semakin tenar semenjak munculnya kompetisi ImageNet (J. Deng et al., 2009). Namun, seiring bertambahnya kedalaman atau jumlah layer dalam model deep learning, tingkat kompleksitasnya juga meningkat. Hal ini menyebabkan model menjadi semakin sulit untuk diinterpretasikan (Chen et al., 2019), karena fitur yang dipelajari menjadi semakin kompleks dan sulit untuk mengetahui alasan utama di balik keputusan yang diambil oleh model. Salah satu cara yang paling umum adalah dengan mengamati output dari layer aktivasi dari setiap layer model *deep learning*.

4.3.1 Ekstraksi Fitur



Gambar 4.1: Strategi transfer learning melalui pemanfaatan fitur umum pada lapisan dangkal. Model belajar dari domain sumber (kambing) kemudian ditransfer ke domain target (sapi) dengan mengekstraksi fitur-fitur seperti tepian dan bentuk, serta menyesuaikan lapisan akhir melalui fine-tuning.

Pada jaringan saraf tiruan yang mendalam (Deep Neural Network / DNN), lapisan awal atau dangkal berperan dalam mengekstraksi fitur-fitur umum seperti tepian atau bentuk dasar. Sementara itu, lapisan yang lebih dalam bertugas menangkap informasi yang lebih kompleks dan spesifik, seperti bentuk wajah atau tangan. Dengan demikian, seiring bertambahnya kedalaman jaringan, fitur yang dipelajari oleh model berkembang dari yang bersifat umum pada lapisan awal, menjadi semakin spesifik pada lapisan-lapisan akhir.

Jika kita dapat mengidentifikasi lapisan-lapisan dalam jaringan yang menangkap fitur umum (misalnya tepian dan tekstur), maka lapisan tersebut

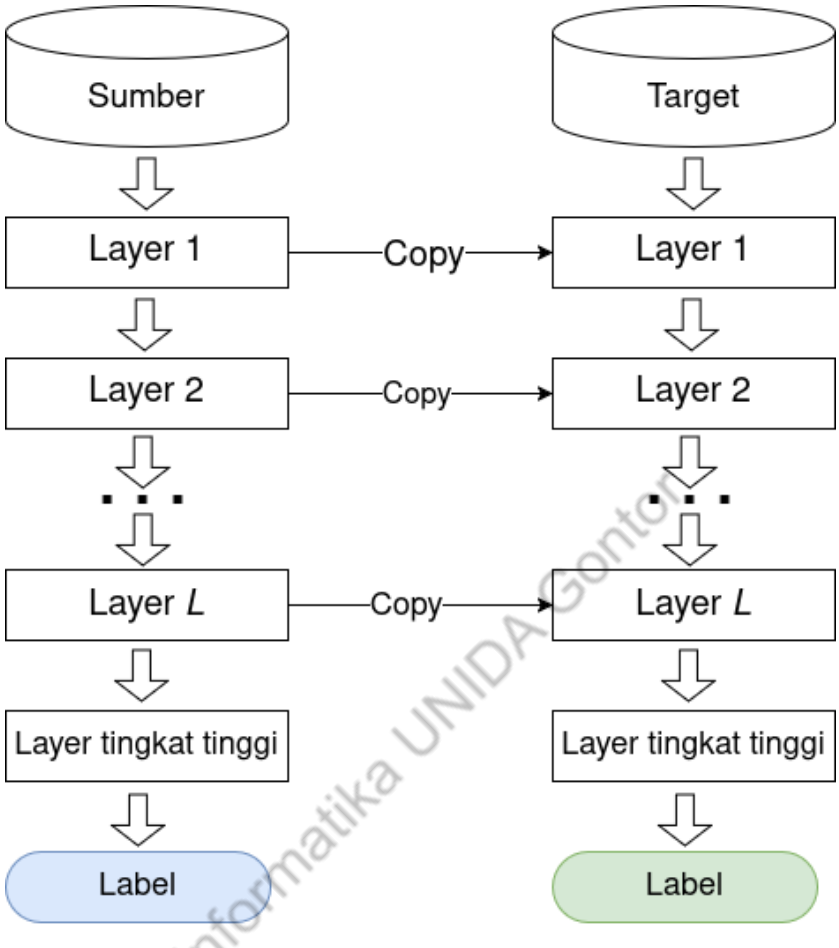
dapat digunakan ulang dalam skenario TL.

Sebagai contoh, misalnya kita memiliki dataset citra kambing sebagai domain sumber. Model yang telah dilatih pada dataset ini dapat menangkap fitur visual dasar seperti bentuk tubuh, tekstur bulu, dan kontur telinga. Fitur-fitur ini juga muncul dalam citra hewan lain seperti sapi. Oleh karena itu, pada domain target yang serupa (misalnya klasifikasi sapi), kita dapat menggunakan kembali lapisan awal dari model sumber, karena representasi fitur yang dipelajari masih relevan.

Gambar 4.1 menggambarkan proses TL dari domain sumber (kambing) ke domain target (sapi) dengan memanfaatkan ekstraksi fitur dari lapisan dangkal. Fitur-fitur umum seperti tepian dan bentuk yang diperoleh dari lapisan awal digunakan kembali di domain target yang memiliki kemiripan visual. Proses ini memungkinkan efisiensi pelatihan ulang melalui strategi *fine-tuning* hanya pada lapisan akhir.

4.3.2 Pralatih dan *Fine-Tuning*

Strategi *fine-tuning* memiliki perbedaan mendasar dengan adaptasi domain. Pada *fine-tuning*, domain sumber dan domain target tidak harus memiliki kelas atau kategori yang identik. Dalam praktiknya, ketika model akan digunakan pada tugas baru, pelatihan biasanya tidak dilakukan dari awal. Melatih model dari awal (dari parameter acak) membutuhkan waktu dan sumber daya komputasi yang besar, terutama ketika data berlabel di domain target jumlahnya terbatas. Oleh karena itu, strategi *fine-tuning* digunakan untuk menyesuaikan model pralatih agar mampu menangani tugas baru secara efisien, dengan memanfaatkan pengetahuan yang telah dipelajari sebelumnya. Secara definisi pralatih adalah melatih model pada dataset yang besar dan bersifat generik seperti ImageNet, untuk memperlajari representasi fitur. Adapun *fine-tuning* adalah pemanfaatan model pralatih dan mengadaptasikannya pada kasus atau dataset tertentu yang umumnya berukuran lebih kecil dan spesifik pada domain tertentu.



Gambar 4.2: Ilustrasi model pralatih dan *fine-tuning*.

Gambar 4.2 menunjukkan ilustrasi dari model pralatih dan *fine-tuning* di mana model sumber dilatih dengan dataset yang kaya dan telah diberi label. Parameter yang telah dilataih dari layer bawah hingga menengah (Layer 1 hingga layer L) lalu disalin pada jaringan target. Sementara itu, pada lapisan atas (layer tingkat tinggi) diinisiasi ulang secara acak dan dilatih ulang (*retrain*). Skema ini memunculkan sejumlah pertanyaan penting yang layak dipertimbangkan: Lapisan mana yang sebaiknya dipertahankan (dibekukan), dan lapisan mana yang perlu di-*fine-tuning*? Apakah membekukan lebih banyak lapisan akan lebih efektif dalam mempertahankan pengetahuan umum dari domain sumber atau justru menghambat adaptasi terhadap

Tabel 4.1: Daftar Notasi pada Persamaan Fine-Tuning

Notasi	**Deskripsi**
θ	Parameter model yang sedang disesuaikan melalui *fine-tuning*.
θ_0	Parameter awal hasil *pre-training* dari tugas sebelumnya.
θ^*	Parameter optimal yang diperoleh setelah *fine-tuning*.
$\mathcal{D}$	Dataset target dengan jumlah label yang terbatas.
$\mathcal{L}(\theta \mid \theta_0, \mathcal{D})$	Fungsi kerugian terhadap data target dengan parameter awal dari θ_0.
$\arg\min_\theta$	Notasi untuk mencari nilai θ yang meminimalkan fungsi kerugian.

kasus/tugas baru? Di sisi lain, apakah dengan melakukuan *fine-tuning* pada lapisan yang lebih banyak akan berdampak pada meningkatnya kemampuan adaptasi dengan risiko terjadinya *overfitting*, jika datanya terbatas? Bagaimana strategi regularisasi untuk menanggulangi *overfitting* pada kondisi data yang terbatas?

Secara matematis, proses *fine-tuning* dan pra-pelatihan dapat dirumuskan sebagai berikut. Diberikan himpunan data target $\mathcal{D}$ yang memiliki label terbatas, strategi pra-pelatihan dan fine-tuning bertujuan untuk mempelajari fungsi f yang diparameterisasi oleh θ, dengan memanfaatkan pengetahuan awal (parameter θ_0) yang diperoleh dari tugas sebelumnya. Parameter optimal θ^* dicari dengan meminimalkan fungsi kerugian terhadap data target sebagai berikut:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta \mid \theta_0, \mathcal{D}) \tag{4.1}$$

Adapun rincian dari simbol-simbol pada Persamaan 4.1 ada pada Tabel 4.1.

Model yang dilatih pada data lain bisa jadi tidak cocok untuk tugas yang akan Anda kerjakan. Ketidakcocokan ini bisa jadi karena distribusi dataset pada data orang lain berbeda dengan data yang dimiliki atau struktur dataset yang terlalu besar. Sebagai contoh, jika ingin melatih jaringan saraf tiruan untuk mengklasifikasi anjing dan kucing, kita bisa menggunakan model jaringan saraf yang dilatih sebelumnya pada datase CIFAR-100. Akan tetapi, CIFAR-100 memiliki 100 kelas. Padahal, kita hanya memerlukan dua

saja, anjing dan kucing. Oleh karena itu, kita harus memperbaiki lapisan yang umum dari jaringan saraf pre-training dan memodifikasi lapisan luaran (output layer) untuk menyesuaikan dengan klasifikasi anjing dan kucing.

Secara umum, kita bisa menyebutkan keunggulan dari fine-tuning, yaitu kita tidak perlu melatih model dari dasar (train from scratch) yang tentu bisa menghemat waktu dan biaya. Lalu, model pre-train sering dilatih pada dataset skala besar. Pelatihan semacam ini memperbesar kemampuan generalisasi dari model ketika akan digunakan untuk kasus lain. Selain itu, fine-tuning sangat mudah untuk diimplementasikan.

4.4 Model Pre-Train untuk Ekstraksi Fitur

Fine-tuning tidak hanya berguna untuk pembelajaran mendalam, tapi juga untuk pembelajaran mesin sederhana. Kita bisa menggunakan jaringan pre-train untuk mengekstrak fitur yang lebih baik dari pada memilih atau membuat fitur secara manual berdasarkan pengetahuan manusia atau *rules of thumbs*. Sebagai contoh, teknik ekstraksi fitur seperti Scale-Invariant Feature Transform (SIFT) sangat bagus ketika sebelum era pembelajaran mendalam. Beberapa peneliti telah mulai menggunakan pembelajaran mendalam, bernama DeCAF (Donahue et al., 2014), untuk mengekstrak fitur yang tentunya menaikkan performa dibandingkan pembelajaran mesin tradisional.

Ada juga peneliti yang menggabungkan teknik tradisional dengan mendalam untuk transfer learning. Ide ini dicetuskan oleh (Wang, Chen, Yu, Huang, & Yang, 2019) yang mana mengusulkan teknik bernama EasyTL. Teknik ini menggabukan ResNet (He et al., 2016) dengan teknik klasifikasi sederhana pada layer akhir. (Wang et al., 2019) merekomendasikan beberapa pilihan dalam transfer learning:

1. Langsung menerapkan model pre-train pada kasus baru

2. Melakukan pre-train dulu dan fine-tuning kemudian

3. Menggunakan model pre-train sebagai fitur ekstraksi dan membangun pengklasifikasi pada fitur hasil pre-train.

4.5 Cara Pre-Training dan Fine-Tuning

Masalah utama yang sering muncul pada transfer learning adalah pemilihan jumlah lapisan yang akan ditransfer dan jumlah layer yang akan di-fine-tune untuk kasus baru. Beberapa peneliti menggunakan analisis yang rinci terkait kapan untuk fine tuning (Zhang, Zhang, & Yang, 2019) dan studi fine-tuning dengan meta-learning (Jang, Lee, Hwang, & Shin, 2019).

Secara matematis, pemetaan transfer learning dari sumber ke target bisa dituliskan dengan:

$$\|r_\theta\left(T_\theta^n(\mathbf{x})\right) - S^m(\mathbf{x})\|_2^2, \tag{4.2}$$

di mana $T_\theta^n(\mathbf{x})$ adalah jumlah ke n-th representasi fitur dari jaringan saraf target, $S^m(\mathbf{x})$ m-th representasi dari jaringan saraf sumber, θ adalah parameter yang dilatih. Adapun $\mathbf{x}$ adalah data input berupa matrix, tensor, atau vektor. Sedangkan r_θ adalah transformasi linier yang memetakan secara langsung representasi fitur dari sumber ke target. Persamaan 4.2 bisa dikatakan juga sebagai persamaan yang menghitung perbedaan fitur sumber dan target dengan L2-norm dan juga sebagai bukti bahwa kita bisa melakukan transfer learning dari lapisan m-th dari sumber ke n-th target.

Kadangkala, tidak semua fitur dari sumber diperlukan untuk ditransfer ke target. Oleh karena itu, diperlukan sebuah bobot yang bisa dilatih (*learnable weights*) untuk menentukan fitur map (*channel*) mana yang penting. Untuk mencari fitur ini, digunakan hitungan sebagai berikut:

$$\mathcal{L}_{channel}^{m,n}(\theta|x, w^{m,n}) = \frac{1}{HW}\sum_c w_c^{m,n}\sum_{i,j}(r_\theta(T_\theta^n(x))_{c,i,j} - S^m(x)_{c,i,j})^2 \tag{4.3}$$

Di sini, $w_c^{m,n}$ adalah nilai bobot pada tiap kanal c, dan nilainya dinormalisasi sehingga total skornya adalah 1. Kemudian, dicek mana saja layer m yang berguna untuk ditransfer ke layer n. Bobot yang digunakan untuk menentukan layer yang akan ditransfer diprediksi dengan pendekatan meta-learning

Tabel 4.2: Tabel notasi dari Persamaan 4.3 dan 4.4

Simbol	Penjelasan
x	Data input (misalnya gambar).
$T_\theta^n(x)$	Fitur (feature map) dari layer ke-n pada jaringan target.
$S^m(x)$	Fitur dari layer ke-m pada sumber
$r_\theta(\cdot)$	Fungsi transformasi
(c, i, j)	Indeks channel ke-c dalam feature map.
$w_c^{m,n}$	Bobot pentingnya channel ke-c untuk pasangan layer $m \to n$
$H \times W$	Ukuran spasial (tinggi dan lebar) dari setiap channel.

pada $S^m(x)$. Model, kemudian, menggabungkan nilai loss dari tiap pasangan layer m, n sehingga menjadi total transfer loss. Transfer loss diformulasikan dengan:

$$\mathcal{L}_{total}(\theta|x, y, \phi) = \mathcal{L}_{ce}(\theta|x, y) + \beta\mathcal{L}_{channel}(\theta|x, \phi) \tag{4.4}$$

di mana $\mathcal{L}_{ce}$ adalah fungsi cross-entropy, β sebagai variabel kontrol terhadap nilai loss, sedangkan $\mathcal{L}_{channel}$ adalah fungsi loss yang menghitung seberapa baik jaringan target meniru fitur dari jaringan sumber, dengan fokus hanya pada kanal-kanal yang dianggap penting. Terkait strategi transfer learning mana yang cocok, (Wang & Chen, 2023a) merumuskan tiga hal sebagai berikut:

1. Single: transfer learning dengan cara menggunakan layer terakhir dari sumber dan diterapkan pada layer tertentu pada target

2. One-to-one: Tiap-tiap layer, sebelum layer pooling, ditransfer pada layer tertentu di sisi target.

3. All-to-all: layer ke-m dari sumber ditransfer ke layer ke-n pada jaringan target.

4.6 Implementasi *Transfer Learning* di PyTorch

Di bagian ini, saya tunjukkan contoh implementasi pre-training dan fine-tuning dengan PyTorch. Adapun kode dari implementasi ini, silakan dilihat di sini https://github.com/virgantara/Deep-Learning-Course/tree/master/courses/week04

4.7 Kesimpulan

Dengan menguasai teknik regulasi, optimasi, dan strategi lanjutan seperti Transfer Learning, kita dapat merancang model deep learning yang lebih robust, efisien, dan mampu mengatasi tantangan nyata di lapangan seperti overfitting, keterbatasan data, dan waktu pelatihan yang lama. Pada bab-bab berikutnya, kita akan fokus pada penerapan teknik-teknik ini dalam berbagai studi kasus dan proyek nyata untuk memperdalam pemahaman serta kemampuan praktis dalam membangun model deep learning yang optimal.