Transfer Learning

2026-05-06 · cheerful mango Haubentaucher

The Flywheel of Generalized Embeddings.

The Why

Machine Learning Engineer

In the Chapter on Variational Auto Encoders we trained an Encoder that mapped inputs into a structured latent space, where nearby points were similar and the bottleneck, our embedding, distilled what mattered. The space was useful, but it belonged to one model trained for one task. By the same logic, every new task would mean training that encoder again, from scratch, on a freshly and sufficiently labelled dataset.

Deep Learning Engineer

Let's reuse or transfer embeddings. Almost no production model is trained from scratch. They start from a backbone someone else pretrained at scale, freeze or adapt it, and bolt a small task-specific head on top. The encoder is the asset, the head is a thin wrapper, for a narrow task or domain, and the same backbone is shared across many downstream tasks.

Train once, reuse forever. Almost every modern vision and language system in production starts from someone else's pretrained encoder. The encoder is the expensive part, the task-specific head on top is cheap, and the choice of encoder determines almost everything that follows.

Training an encoder per task is not viable. Labels are expensive to collect, domains splinter into modalities that each deserve their own representation, pretraining a competitive backbone costs hundreds of GPU years, and entire classes of inputs have no labelled examples to begin with. The encoder therefore has to come from somewhere else and the cost of obtaining it has to be amortised across many downstream tasks.

Scarce labels. A medical imaging team may need months to label five hundred chest X-rays, while a competitive vision encoder is trained on millions.

Reuse is the mechanism. A backbone trained at scale on a broad task supplies features that work, often surprisingly well, for tasks it was never trained on. Push the scale far enough and the backbone becomes a foundation model that handles new tasks without any

Splintered domains. Satellite imagery, manufacturing defects, document layouts, and X-ray modalities each have their own vocabulary of inputs and in principle each deserves its own encoder.

Enormous compute. Pretraining a modern vision transformer or large language model costs hundreds of GPU years, which few teams can afford once

task-specific training at all, because its embeddings already cover the world that the new task lives in. Once we found that the long tail of any one task already sits inside the backbones embedding [1].

[1]

On Modalities. Foundation models such as CLIP, GPT, and SAM push this idea to the extreme: A single representation, learned once on hundreds of millions of image-text pairs or tokens, can be reused on classification, retrieval, or segmentation tasks it was never explicitly trained for, with zero or only a handful of labelled examples. Natural language is specifically applicable to task transfer as the input itself is adaptable and directs the task. The same property that makes the input the task specification is also the surface on which prompt injection attacks operate, since adversarial text inserted into the input can redirect the model to a task its operator never intended [2].

[2]

In order to move from one-task-one-model to many-tasks-one-backbone we have good reason to find ways to,

- reuse features learned at scale for downstream tasks with little or no labelled data.
- adapt pretrained representations efficiently, without paying full retraining cost.
- bridge modalities so that text descriptions can serve as classifiers for visual inputs.
- diagnose when transfer fails because source and target distributions are too far apart.

LoRA factors the correction as two thin matrices that slot onto a frozen layer. With W frozen, $\partial L/\partial W$ is never formed and Adam's optimiser state collapses onto the rank-r adapters. Fine-tuning just got cheap.

The fundamental question this chapter answers is how a model trained on one corpus can serve a task it never saw, with little or no additional supervision, and how to pick the right adaptation strategy for the data and compute budget at hand.

SELF-REFLECTION questions to guide your thinking:

- Why does one shot or few shot learning on top of a frozen backbone work, when training the same network from scratch on the same handful of examples does not?
- What role do class centroids play as anchor points in the embedding, and why does averaging many embeddings into one prototype still represent the class faithfully?
- In Lampert zero shot, the bridge from attributes to centroids is fitted only on the seen classes, yet it locates unseen classes correctly. What does the bridge actually learn that lets it generalise, and what role does source breadth play in making that work?
- Why does the Lampert bridge stall when two source classes share an identical attribute row, and what does that aliasing tell you about where the ceiling on attribute based zero shot lives?
- What changes when the hand built bridge is replaced by a contrastively trained text encoder, and why is the resulting structure of two encoders meeting in a shared space the same shape as Lampert?
- Why does cosine similarity work cleanly as the decision rule in both Lampert and CLIP, and what does it imply about how the two sides of the comparison have to be normalised?
- Why does composing pretrained pieces at inference time, a frozen backbone with a learned bridge and a retrieval or clustering step, produce useful behaviour on tasks that none of the pieces was specifically trained for?
- The CLIP contrastive loss encourages the diagonal of the image caption similarity matrix to dominate each row and each column. Why does that single objective produce both the alignment of matched pairs and the separation of unmatched ones at the same time, without a separate negative term?
- Why is the classification head discarded the moment you transfer to a new task, and what does the act of throwing it away tell you about where the value of pretraining actually accumulates inside the network?
- Why does the choice between freezing the backbone and fine tuning it end up depending on both how much labelled target data is available and how far the target distribution sits from the source, rather than being a fixed best practice?

Recap of Key Concepts:

- A backbone is a pretrained encoder reused across tasks while the head supplies only the decision, so pretraining cost is amortised across many downstream uses rather than paid each time.
- Linear probes succeed at one and few shot because the frozen embedding has already separated the classes, so the probe only has to pick a side of a hyperplane that the backbone effectively pre-drew.
- Class centroids act as anchor points in the embedding: averaging the embeddings of a class gives a single prototype, and classification reduces to nearest centroid under cosine similarity once both sides are unit normalised.
- Zero shot means reaching unseen classes through a bridge between class semantics and image features, hand built and fitted by least squares in Lampert, learned contrastively from natural language pairs in CLIP; the bridge structure of two encoders meeting in a shared space is the same in both, only the fitting criterion changes.
- Foundation models are backbones whose scale and breadth push the embedding to cover downstream input distributions, so composition of frozen pieces at inference time can replace task specific training, and transfer only fails when source and target distributions diverge beyond what the embedding describes.

The backbone arrives ready-made. Throughout this chapter we have taken pretrained encoders as given and asked how to reuse them. ImageNet supervised pretraining works for natural images because someone else once labelled fourteen million examples, but no second ImageNet is being built for every new domain. Foundation models need to find a source of supervision that hides in plain sight.

Self-supervised learning generalises this idea. Instead of asking humans for labels, the model is given a task that can be solved using only the structure of the data itself: predict an image from itself, predict the next token from the previous ones. The labels come from the data, and the representations that emerge are the very backbones this chapter has taught us to reuse.

Teaser. Where in the data does implicit labels, cues and supervision hide?

feedback