Transfer Learning and Fine-Tuning

Applied Deep Learning — Chapter 7

Yin Yang

Foundation Books Project

Roadmap

Why Transfer Learning

Vocabulary and Setup

What `vision_learner` Builds

Fine-Tuning in Two Stages

Same Idea in Keras 3

Augmentation and Practice

Failure Modes

Summary

Why Transfer Learning

The Applied Shortcut

- Training Food-101 from scratch wastes early epochs relearning generic visual patterns: edges, color contrasts, textures, parts.
- A pretrained ImageNet ResNet already encodes many such patterns.
- Spend the target-task budget *adapting* those features.

Food-101: 101 dish categories, separate apple pie from sushi, but also ravioli vs. dumplings vs. gnocchi.

Replace the head. Reuse visual features, drop the source-task label vocabulary.

Body and Head Decomposition



$$f(x) = h_{\theta_{\text{head}}}(g_{\theta_{\text{body}}}(x)).$$

- Body: pretrained convolutional feature extractor.
- Head: task-specific. $512 \cdot 101 + 101 = 51{,}813$ params — tiny next to a ResNet body.

Vocabulary and Setup

Source, Target, Pretraining, Fine-Tuning

Source task data and objective used *before* target training (here: ImageNet classification).

Target task new data and objective (here: Food-101 dishes).

Pretraining training on the source task.

Transfer learning reusing source-trained weights or features for the target task.

Fine-tuning target-task training that adapts the transferred model.

Frozen parameter not updated by gradient descent in this stage.

Validation decides transfer. Source similarity is plausibility; target validation is evidence.

Frozen vs. Unfrozen Stages

Frozen-head training:

$$\min_{\theta_{\text{head}}} \frac{1}{n} \sum_i \ell\Big(h_{\theta_{\text{head}}}\big(g_{\theta^{\text{pre}}_{\text{body}}}(x_i)\big),\, y_i\Big)\,.$$

Unfrozen stage: also update some or all of θ_{body}.

- Same forward pass; the optimizer's parameter set is what changes.
- In fast.ai, batch-norm layers can stay trainable under `train_bn` even when the splitter freezes other groups.

Food-101: Data, Split, Metric

- 101 categories, 101,000 images.
- Standard split: 750 train and 250 test per class.
- Train: $101 \cdot 750 = 75{,}750$. Test: $101 \cdot 250 = 25{,}250$.
- Carve a 20% validation set from the training split: $\approx 60{,}600$ training and $\approx 15{,}150$ validation images.

Primary metric: top-1 accuracy.

Reserve the official test split for one final evaluation after model selection.

What `vision_learner` Builds

Inside `vision_learner`

- Loads the requested architecture with pretrained weights.
- Cuts off the source-task classifier (e.g. ResNet ImageNet head).
- Attaches a new head sized to the data's class count.
- Installs a splitter; freezes pretrained groups by default.
- Adds pretrained image normalization when `normalize=True`.

Splitter and parameter group: a splitter maps the model to ordered groups; freezing, progressive unfreezing, and discriminative learning rates act on these groups.

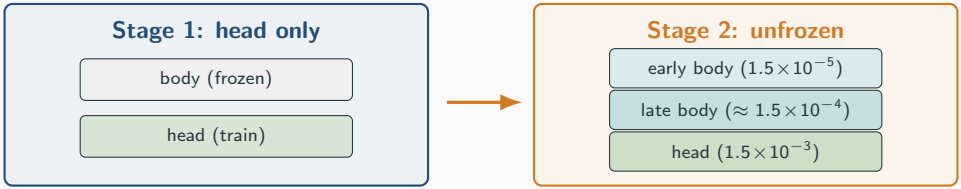
The fast.ai Head Shape

Component	Example shape / role	Why
Input batch	`(64, 3, 224, 224)`	RGB images, PyTorch order
ResNet body	feature maps, many channels	reusable visual representation
Adaptive concat pool	$2 \cdot 512 = 1024$ values	fixed-size vector
Hidden head layer	e.g. 512 hidden units	task-specific remap
Final head layer	$512 \cdot 101 + 101 = 51{,}813$ params	101 dish logits

CIFAR-10 changed the stem; Food-101 keeps it. Food images are resized to ImageNet-scale 224, so preserving the stem preserves the transferred representation.

Fine-Tuning in Two Stages

What `fine_tune` Does



- Example: `freeze_epochs=1`, `epochs=4`, `base_lr=0.003`, `lr_mult=100`.
- LR high $= 0.003/2 = 0.0015$; LR low $= 0.0015/100 = 1.5\times10^{-5}$.

Progressive Unfreezing

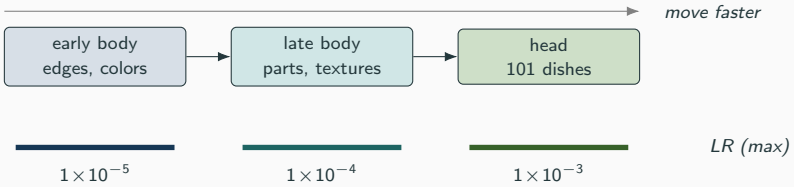
Stage in steps instead of all-at-once:

1. `learn.freeze(); fit_one_cycle(1, 3e-3)` — head only.
2. `learn.freeze_to(-2); fit_one_cycle(2, slice(1e-5, 1e-4))` — last two groups.
3. `learn.unfreeze(); fit_one_cycle(3, slice(1e-6, 1e-4))` — all groups.

- Each stage adds adaptability and overfitting risk.
- Earn the next stage with a validation gain large enough to justify runtime and risk.

Unfreeze gradually; record the schedule.

Discriminative Learning Rates



- Update size scales linearly with LR; early filters move slowly.
- Late blocks and head adapt to Food-101 quickly.
- One-cycle varies LRs through training; relative ratio preserved.

Same Idea in Keras 3

Keras 3 Translation

1. Set `KERAS_BACKEND`, then import `keras`.
2. Load `ResNet50(include_top=False, weights="imagenet")`; set `base_model.trainable = False`.
3. Build a new model: augmentations → `preprocess_input` → frozen body → dropout → `Dense(101)`.
4. `compile` + `fit` for the frozen-head stage.
5. `base_model.trainable = True`; *recompile* with a smaller LR; `fit` for the unfrozen stage.

Recompile after changing trainability. It updates Keras's view of which weights move during `fit`.

fast.ai vs. Keras Mapping

Concept	fast.ai	Keras 3
Pretrained body	`vision_learner(...)`	`ResNet50(include_top=False)`
New head	built from data vocab	explicit `Dense(101)`
Frozen stage	`learn.freeze()`	`trainable=False; compile; fit`
Unfrozen stage	`learn.unfreeze()`	`trainable=True; recompile; fit`
Discr. LRs	`slice(low, high)`	custom training logic

fast.ai supplies more transfer-learning policy in one line; Keras exposes more moving parts. Same invariant: replace the head, control which parts move.

Augmentation and Practice

Default `aug_transforms(size=224)`

Two preprocessing stages:

- Item: `Resize(460)` on PIL images (random crop in train, center crop in val).
- Batch: `aug_transforms(size=224)` on tensors, on the GPU.

Defaults: horizontal flip ($p = 0.5$, no vertical flip), affine geometry ($p_{\text{aff}} = 0.75$, max rotate $10°$, max zoom $1.1\times$), warp (≤ 0.2), lighting (brightness and contrast ≤ 0.2), reflection padding.

Augmentation rule: a transform is regularization only if the label should still be true after it.

Practical Recipe

1. Pretrained ResNet34 or ResNet50; carve 20% validation; reserve test.
2. Use fast.ai defaults for the first baseline.
3. `learn.lr_find()` as a candidate, not a guarantee.
4. Change one major factor at a time: model size, image size, schedule, freeze policy.
5. Record runtime alongside accuracy.
6. Evaluate the official test split *once* after model selection.

Progressive resizing: smaller crops first, then 224. Treat it as an experiment knob, not a free coupon.

Measured Food-101 ResNet34 Baseline

Field	Value
Architecture	ResNet34 + pretrained TorchVision weights
Split	`RandomSplitter(valid_pct=0.2, seed=42)`
Train / val items	60,600 / 15,150
Batch size, image size	64, 224 (resize 460)
Total params	21,863,744
Trainable, frozen stage	$\approx$ 596,096
Schedule	1 frozen + 5 unfrozen, base LR 3×10^{-3}, lr_mult 100
Frozen-stage val acc	52.73% (492.9s train)
Unfrozen-stage val acc	**77.30**% (2594.3s train)
Test	25,250 images reserved

Failure Modes

Common Failure Modes

Symptom	Likely cause	Controlled response
Wrong labels in batch	path parsing or label fn	inspect paths, vocab
Val loss jumps after unfreeze	LR too high; too many groups unfrozen	lower LR; smaller slices; progressive
Out of memory	batch / image too large	shrink, mixed precision
Fast train, weak val	head underfit; bad transforms	longer head stage; check augs
Strong val, weak test	val overfitting; repeated test use	one final test, locked
Accuracy without context	no reproducibility record	log versions, seed, command

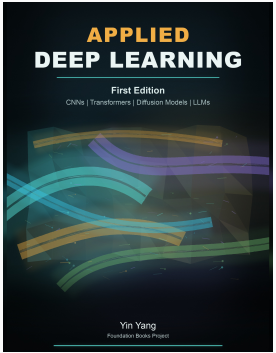
Change one thing at a time; controlled debugging keeps transfer learning fast.

Summary

Summary

- Decompose: $f(x) = h_{\theta_{\text{head}}}(g_{\theta_{\text{body}}}(x))$.
- Replace the source-task head; reuse the pretrained body.
- fast.ai: `vision_learner` builds it; `fine_tune` runs frozen-head + unfrozen stages with discriminative LRs.
- Keras 3 expresses the same idea with explicit `trainable` flags and recompile.
- Augmentation, validation split, runtime, hardware, seeds, versions belong beside the accuracy number.
- A pretrained body is a starting bet, not a free accuracy coupon.

Next: feature vectors as a language — embeddings, IMDB sentiment, CLIP.



Applied Deep Learning

Chapter overview slides for the book by Yin Yang.

Book page	`foundation-books.github.io/applied-deep-learning`
Code	`github.com/foundation-books/applied-deep-learning-code`
Print	`amazon.com/dp/B0GZF7QRSH`
Kindle	`amazon.com/dp/B0GZF9221X`
License	CC BY-NC-ND 4.0

These free overview slides may be shared non-commercially with attribution and without modifications. Full explanations, exercises, and companion-code context are in the book and linked repository.