Comprehensive Reference for Keras tf.keras.callbacks

This document provides an in-depth overview of the most commonly used built-in callbacks in TensorFlow/Keras. For each callback, you will find:

- **Description**: What the callback does.
- **Parameters**: List of all constructor arguments, with types, defaults, options.
- **Usage Guidance & Use Cases**: When and why to use it.
- **Pros & Cons**: Trade-offs, caveats, and best practices.

---

1. `EarlyStopping`

Description: Stops training when a monitored metric has stopped improving.

Constructor Parameters:

Parameter	Type	Default	Options / Notes
`monitor`	`str`	`'val_loss'`	Any model metric: `'loss'`, `'accuracy'`, `'val_accuracy'`, etc.
`min_delta`	`float`	`0`	Minimum change to qualify as improvement.
`patience`	`int`	`0`	Epochs with no improvement before stopping.
`verbose`	`int`	`0`	0 = silent, 1 = message on stop.
`mode`	`str`	`auto`	`min`, `max`, `auto` (infer from metric name).
`baseline`	`float`	`None`	Training stops if metric never exceeds this value.
`restore_best_weights`	`bool`	`False`	Restore model weights from the epoch with best metric.

Usage Guidance:

- **When to use**: Prevent overfitting and save training time by halting when no further gain.
- **Key tips**:
  - Set `monitor` to a validation metric (e.g. `'val_accuracy'`).
  - Use `restore_best_weights=True` to retain the best model.
  - Choose `patience` based on expected noise: 5–10 for stable metrics.

Pros & Cons:

- **Pros**:
  - Automatically halts training to avoid wasted epochs.
  - Can preserve best weights.
- **Cons**:
  - Too small `patience` may stop before true optimum.
  - Too large wastes compute.

---

2. `ModelCheckpoint`

Description: Saves model (or weights-only) at specified intervals (epochs).

Constructor Parameters:

Parameter	Type	Default	Options / Notes
`filepath`	`str`	**Required**	Format can include formatting options, e.g. `{epoch:02d}-{val_loss:.2f}.h5`.
`monitor`	`str`	`'val_loss'`	Metric to monitor for saving best.
`verbose`	`int`	`0`	0 = silent, 1 = message on save.
`save_best_only`	`bool`	`False`	Only save when monitored metric improves.

Parameter	Type	Default	Options / Notes
`save_weights_only`	`bool`	`False`	Save only model weights, not the entire model.
`mode`	`str`	`auto`	`min`, `max`, `auto`.
`save_freq`	`str` / `int`	`'epoch'`	`'epoch'` or integer number of samples between saves.
`options`	`tf.saved_model.SaveOptions`	`None`	Extra saving options.

Usage Guidance:

- **When to use**: Keep checkpoints, recover from crashes, or save best model.
- **Key tips**:
  - Use `save_best_only=True` with a validation metric to only keep the best.
  - Choose `save_weights_only=True` if you plan to rebuild architecture separately.
  - Use formatted `filepath` to track epoch and metric values.

Pros & Cons:

- **Pros**:
  - Protects against data or hardware failures.
  - Enables model selection based on validation performance.
- **Cons**:
  - Disk space usage if saving frequently or without `save_best_only`.

---

3. `ReduceLROnPlateau`

Description: Reduces learning rate when a monitored metric has stopped improving.

Constructor Parameters:

Parameter	Type	Default	Options / Notes
`monitor`	`str`	`'val_loss'`	Metric to monitor.
`factor`	`float`	`0.1`	Factor by which to reduce LR. New_lr = lr * factor.
`patience`	`int`	`10`	Epochs with no improvement before reducing.
`verbose`	`int`	`0`	Print message on LR change.
`mode`	`str`	`auto`	`min`, `max`, `auto`.
`min_delta`	`float`	`1e-4`	Minimum change to qualify as improvement.
`cooldown`	`int`	`0`	Epochs to wait after LR reduction before resuming normal operation.
`min_lr`	`float`	`0`	Lower bound on LR.

Usage Guidance:

- **When to use**: Fine-tune learning rate schedule based on performance plateau.
- **Key tips**:
  - Set `patience` slightly larger than for `EarlyStopping` to allow adjustment.
  - Use `min_lr` to prevent LR from becoming too small.

Pros & Cons:

- **Pros**:
  - Automates LR scheduling without manual callbacks.
  - Helps model converge after plateau.
- **Cons**:
  - Complex interplay with optimizers’ built-in LR schedules.
  - Overly aggressive reduction can stall training.

---

4. `LearningRateScheduler`

Description: Schedules learning rate according to user-defined function.

Constructor Parameters:

Parameter	Type	Default	Options / Notes
`schedule`	`function(epoch, lr)` → new_lr	**Required**	Custom function that returns new LR.
`verbose`	`int`	`0`	0 = silent, 1 = print LR each epoch.

Usage Guidance:

- **When to use**: Implement custom LR decay (e.g., step decay, cosine annealing).
- **Key tips**:
  - Ensure `schedule` returns a positive float.
  - Combine with `ModelCheckpoint` or `TensorBoard` to monitor LR changes.

Pros & Cons:

- **Pros**:
  - Full control over LR over epochs.
- **Cons**:
  - Requires careful design of schedule function.

---

5. `TensorBoard`

Description: Streams logs to TensorBoard for visualization.

Constructor Parameters:

Parameter	Type	Default	Options / Notes
`log_dir`	`str`	**Required**	Directory where to save logs.
`histogram_freq`	`int`	`0`	Frequency (in epochs) at which to compute activation and weight histograms.
`write_graph`	`bool`	`True`	Whether to log the graph.
`write_images`	`bool`	`False`	Write model weights to visualize as images.
`update_freq`	`str` / `int`	`'epoch'`	`'batch'`, `'epoch'`, or integer (samples).
`profile_batch`	`int` / `tuple`	`2`	Batch(es) to profile. E.g. `(10, 20)`.

Usage Guidance:

- **When to use**: Visualize training metrics, graphs, histograms, embeddings.
- **Key tips**:
  - Use `histogram_freq>0` for deep introspection (slower).
  - Launch TensorBoard with `tensorboard --logdir=...`.

Pros & Cons:

- **Pros**:
  - Rich visual insights into training and model internals.
- **Cons**:
  - Writing histograms/images can slow down training.

---

6. `CSVLogger`

Description: Streams epoch results to a CSV file.

Constructor Parameters:

Parameter	Type	Default	Options / Notes
`filename`	`str`	**Required**	Path to CSV file.
`separator`	`str`	`','`	Column separator.
`append`	`bool`	`False`	Append to existing file if True.

Usage Guidance:

- **When to use**: Record training history for later analysis or plotting.

Pros & Cons:

- **Pros**:
  - Simple record of metrics without needing Python objects.
- **Cons**:
  - Limited to tabular logs (no histograms or images).

---

7. `TerminateOnNaN`

Description: Stops training when a NaN loss is encountered.

Constructor Parameters: None

Usage Guidance:

- **When to use**: Protect against exploding gradients or invalid loss calculations.

Pros & Cons:

- **Pros**: Immediately halts to avoid wasted compute.
- **Cons**: Doesn't provide recovery strategy—combine with `ModelCheckpoint`.

---

8. `LambdaCallback`

Description: Create custom callbacks from simple functions.

Constructor Parameters:

Parameter	Type	Default	Notes
`on_epoch_begin`	`func`	`None`	Called at start of each epoch.
`on_epoch_end`	`func`	`None`	Called at end of each epoch.
`on_batch_begin`	`func`	`None`	Called at start of each batch.
`on_batch_end`	`func`	`None`	Called at end of each batch.
`on_train_begin`	`func`	`None`	Called at start of training.
`on_train_end`	`func`	`None`	Called at end of training.

Usage Guidance:

- **When to use**: Quick ad-hoc actions (e.g. dynamic printouts, custom logging).

Pros & Cons:

- **Pros**: Lightweight, no need to subclass.
- **Cons**: Limited to simple callbacks; complex logic may merit subclassing.

> **Tip**: For most projects, a small suite of callbacks— `EarlyStopping`, `ModelCheckpoint`, `ReduceLROnPlateau`, and `TensorBoard` — covers the majority of monitoring and control needs. Others (like `CSVLogger` or `LambdaCallback`) are useful for customized workflows

> or lightweight logging.