tf.keras.Model

✓ See Stable See Nightly

TensorFlow 1 version (/versions/r1.15/api_docs/python/tf/keras/Model)

View source on GitHub (https://github.com/tensorflow/tensorflow/blob/v2.3.0/tensorflow/python/keras/engine/training.py#L2634)

`Model` groups layers into an object with training and inference features.

Inherits From: `Layer` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer)

+ View aliases

Main aliases

`tf.keras.models.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model)

Compat aliases for migration

See Migration guide (https://www.tensorflow.org/guide/migrate) for more details.

`tf.compat.v1.keras.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model), `tf.compat.v1.keras.models.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model)

```
ras.Model(
args, **kwargs
```

Used in the notebooks

Used in the guide	Used in the tutorials

Used in the guide	Used in the tutorials
• The Functional API (https://www.tensorflow.org/guide/keras/functional) • Save and load Keras models (https://www.tensorflow.org/guide/keras/save_and_serialize) • Training and evaluation with the built-in methods (https://www.tensorflow.org/guide/keras/train_and_evaluate) • Making new Layers and Models via subclassing (https://www.tensorflow.org/guide/keras/custom_layers_and_models) • Customize what happens in Model.fit (https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit)	• Time series forecasti (https://www.tensorf • Image captioning witl (https://www.tensorf • Intro to Autoencoders (https://www.tensorf • Load CSV data (https://www.tensorf • Save and load a mod (https://www.tensorf

Arguments

`inputs`	The input(s) of the model: a `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) object or list of `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) objects.
`outputs`	The output(s) of the model. See Functional API example below.
`name`	String, the name of the model.

There are two ways to instantiate a `Model`:

1 - With the "Functional API", where you start from `Input`, you chain layer calls to specify the model's forward pass, and finally you create your model from inputs and outputs:

```
t tensorflow as tf

s = tf.keras.Input(shape=(3,))
f.keras.layers.Dense(4, activation=tf.nn.relu)(inputs)
ts = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)
 = tf.keras.Model(inputs=inputs, outputs=outputs)
```

2 - By subclassing the `Model` class: in that case, you should define your layers in `__init__` and you should implement the model's forward pass in `call`.

```
t tensorflow as tf

 MyModel(tf.keras.Model):

 __init__(self):
uper(MyModel, self).__init__()
elf.dense1 = tf.keras.layers.Dense(4, activation=tf.nn.relu)
elf.dense2 = tf.keras.layers.Dense(5, activation=tf.nn.softmax)

 call(self, inputs):
 = self.dense1(inputs)
eturn self.dense2(x)

 = MyModel()
```

If you subclass **`Model`**, you can optionally have a **`training`** argument (boolean) in **`call`**, which you can use to specify a different behavior in training and inference:

```
t tensorflow as tf

 MyModel(tf.keras.Model):

 __init__(self):
uper(MyModel, self).__init__()
elf.dense1 = tf.keras.layers.Dense(4, activation=tf.nn.relu)
elf.dense2 = tf.keras.layers.Dense(5, activation=tf.nn.softmax)
elf.dropout = tf.keras.layers.Dropout(0.5)

 call(self, inputs, training=False):
 = self.dense1(inputs)
f training:
  x = self.dropout(x, training=training)
eturn self.dense2(x)

 = MyModel()
```

Once the model is created, you can config the model with losses and metrics with **`model.compile()`**, train the model with **`model.fit()`**, or use the model to do prediction with **`model.predict()`**.

Attributes

`distribute_strategy`	The **`tf.distribute.Strategy`**

(https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) this model was created under.

`layers`	

`metrics_names`	Returns the model's display labels for all outputs.
	
	Note: **`metrics_names`** are available only after a **`keras.Model`** (https://www.tensorflow.org/api_docs/python/tf/keras/Model) has been trained/evaluated on actual data.
	
	`>>> inputs = tf.keras.layers.Input(shape=(3,))`
	`>>> outputs = tf.keras.layers.Dense(2)(inputs)`
	`>>> model = tf.keras.models.Model(inputs=inputs, out`
	`>>> model.compile(optimizer="Adam", loss="mse", metr`
	`>>> model.metrics_names`
	`[]`
	
	`>>> x = np.random.random((2, 3))`
	`>>> y = np.random.randint(0, 2, (2, 2))`
	`>>> model.fit(x, y)`
	`>>> model.metrics_names`
	`['loss', 'mae']`
	
	`>>> inputs = tf.keras.layers.Input(shape=(3,))`
	`>>> d = tf.keras.layers.Dense(2, name='out')`
	`>>> output_1 = d(inputs)`
	`>>> output_2 = d(inputs)`
	`>>> model = tf.keras.models.Model(`
	`...  inputs=inputs, outputs=[output_1, output_2])`
	`>>> model.compile(optimizer="Adam", loss="mse", metr`
	`>>> model.fit(x, (y, y))`
	`>>> model.metrics_names`
	`['loss', 'out_loss', 'out_1_loss', 'out_mae', 'out_a`
	`'out_1_acc']`

`run_eagerly`	Settable attribute indicating whether the model should run eagerly.

	Running eagerly means that your model will be run step by step, like Python code. Your model might run slower, but it should become easier for you to debug it by stepping into individual layer calls. By default, we will attempt to compile your model to a static graph to deliver the best execution performance.

Methods

`compile`

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L462-L555)

```
.le(
ptimizer='rmsprop', loss=None, metrics=None, loss_weights=None,
eighted_metrics=None, run_eagerly=None, **kwargs
```

Configures the model for training.

Arguments

`optimizer`	String (name of optimizer) or optimizer instance. See `tf.keras.optimizers` (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers).
`loss`	String (name of objective function), objective function or `tf.keras.losses.Loss` (https://www.tensorflow.org/api_docs/python/tf/keras/losses/Loss) instance. See `tf.keras.losses` (https://www.tensorflow.org/api_docs/python/tf/keras/losses). An objective function is any callable with the signature `loss = fn(y_true, y_pred)`, where y_true = ground truth values with shape = `[batch_size, d0, .. dN]`, except sparse loss functions such as sparse categorical crossentropy where shape = `[batch_size, d0, .. dN-1]`. y_pred = predicted values with shape = `[batch_size, d0, .. dN]`. It returns a weighted loss float tensor. If a custom `Loss` instance is used and reduction is set to NONE, return value has the shape [batch_size, d0, .. dN-1] ie. per-sample or per-timestep loss values; otherwise, it is a scalar. If the model has multiple outputs, you can use a different loss on each

	output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses.
`metrics`	List of metrics to be evaluated by the model during training and testing. Each of this can be a string (name of a built-in function), function or a `tf.keras.metrics.Metric` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Metric) instance. See `tf.keras.metrics` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics). Typically you will use `metrics=['accuracy']`. A function is any callable with the signature `result = fn(y_true, y_pred)`. To specify different metrics for different outputs of a multi-output model, you could also pass a dictionary, such as `metrics={'output_a': 'accuracy', 'output_b': ['accuracy', 'mse']}`. You can also pass a list (len = len(outputs)) of lists of metrics such as `metrics=[['accuracy'], ['accuracy', 'mse']]` or `metrics=['accuracy', ['accuracy', 'mse']]`. When you pass the strings 'accuracy' or 'acc', we convert this to one of `tf.keras.metrics.BinaryAccuracy` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/BinaryAccuracy), `tf.keras.metrics.CategoricalAccuracy` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy), `tf.keras.metrics.SparseCategoricalAccuracy` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy) based on the loss function used and the model output shape. We do a similar conversion for the strings 'crossentropy' and 'ce' as well.
`loss_weights`	Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the *weighted sum* of all individual losses, weighted by the `loss_weights` coefficients. If a list, it is expected to have a 1:1 mapping to the model's outputs. If a dict, it is expected to map output names (strings) to scalar coefficients.
`weighted_metrics`	List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
`run_eagerly`	Bool. Defaults to `False`. If `True`, this `Model`'s logic will not be wrapped in a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function). Recommended to leave this as `None` unless your `Model` cannot be run inside a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).

`**kwargs`	Any additional arguments. Supported arguments: • `experimental_steps_per_execution`: Int. The number of batches to run during each `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) call. Running multiple batches inside a single `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) call can greatly improve performance on TPUs or small models with a large Python overhead. Note that if this value is set to `N`, `Callback.on_batch` methods will only be called every `N` batches. This currently defaults to `1`. At most, one full epoch will be run each execution. If a number larger than the size of the epoch is passed, the execution will be truncated to the size of the epoch. • `sample_weight_mode` for backward compatibility.

Raises

`ValueError`	In case of invalid arguments for `optimizer`, `loss` or `metrics`.

evaluate

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1242-L1394)

```
ıate(
:=None, y=None, batch_size=None, verbose=1, sample_weight=None, steps=None,
:allbacks=None, max_queue_size=10, workers=1, use_multiprocessing=False,
'eturn_dict=False
```

Returns the loss value & metrics values for the model in test mode.

Computation is done in batches (see the `batch_size` arg.)

Arguments

x	Input data. It could be: • A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).

	• A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs). • A dict mapping input names to the corresponding array/tensors, if the model has named inputs. • A `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset. Should return a tuple of either `(inputs, targets)` or `(inputs, targets, sample_weights)`. • A generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) returning `(inputs, targets)` or `(inputs, targets, sample_weights)`. A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the `Unpacking behavior for iterator-like inputs` section of `Model.fit`.
`y`	Target data. Like the input data `x`, it could be either Numpy array(s) or TensorFlow tensor(s). It should be consistent with `x` (you cannot have Numpy inputs and tensor targets, or inversely). If `x` is a dataset, generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instance, `y` should not be specified (since targets will be obtained from the iterator/dataset).
`batch_size`	Integer or `None`. Number of samples per batch of computation. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of a dataset, generators, or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instances (since they generate batches).
`verbose`	0 or 1. Verbosity mode. 0 = silent, 1 = progress bar.
`sample_weight`	Optional Numpy array of weights for the test samples, used for weighting the loss function. You can either pass a flat (1D) Numpy array with the same length as the input samples (1:1 mapping between weights and samples), or in the case of temporal data, you can pass a 2D array with shape `(samples, sequence_length)`, to apply a different weight to every timestep of every sample. This argument is not supported when `x` is a dataset, instead pass sample weights as the third element of `x`.
`steps`	Integer or `None`. Total number of steps (batches of samples) before declaring the evaluation round finished. Ignored with the default value of `None`. If x is a `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset and

	`steps` is None, 'evaluate' will run until the dataset is exhausted. This argument is not supported with array inputs.
`callbacks`	List of `keras.callbacks.Callback` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback) instances. List of callbacks to apply during evaluation. See callbacks (/api_docs/python/tf/keras/callbacks).
`max_queue_size`	Integer. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. Maximum size for the generator queue. If unspecified, `max_queue_size` will default to 10.
`workers`	Integer. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.
`use_multiprocessing`	Boolean. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.
`return_dict`	If `True`, loss and metric results are returned as a dict, with each key being the name of the metric. If `False`, they are returned as a list.

See the discussion of `Unpacking behavior for iterator-like inputs` for `Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit).

Returns

Scalar test loss (if the model has a single output and no metrics) or list of scalars (if the model has multiple outputs and/or metrics). The attribute `model.metrics_names` will give you the display labels for the scalar outputs.

Raises

`RuntimeError`	If `model.evaluate` is wrapped in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).

`ValueError`	in case of invalid arguments.

evaluate_generator

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1831-L1857)

```
ıate_generator(
ıenerator, steps=None, callbacks=None, max_queue_size=10, workers=1,
ıse_multiprocessing=False, verbose=0
```

Evaluates the model on a data generator. (deprecated)

ıg: THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Plea ɔdel.evaluate, which supports generators.

DEPRECATED:

`Model.evaluate` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate) now supports generators, so there is no longer any need to use this endpoint.

fit

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L824-L1146)

```
:=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None,
'alidation_split=0.0, validation_data=None, shuffle=True, class_weight=None,
;ample_weight=None, initial_epoch=0, steps_per_epoch=None,
'alidation_steps=None, validation_batch_size=None, validation_freq=1,
ıax_queue_size=10, workers=1, use_multiprocessing=False
```

Trains the model for a fixed number of epochs (iterations on a dataset).

Arguments	
x	Input data. It could be: • A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs). • A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs). • A dict mapping input names to the corresponding array/tensors, if the model has named inputs. • A `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset. Should return a tuple of either `(inputs, targets)` or `(inputs, targets, sample_weights)`. • A generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) returning `(inputs, targets)` or `(inputs, targets, sample_weights)`. A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given below.
y	Target data. Like the input data `x`, it could be either Numpy array(s) or TensorFlow tensor(s). It should be consistent with `x` (you cannot have Numpy inputs and tensor targets, or inversely). If `x` is a dataset, generator, or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instance, `y` should not be specified (since targets will be obtained from `x`).
batch_size	Integer or `None`. Number of samples per gradient update. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of datasets, generators, or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instances (since they generate batches).
epochs	Integer. Number of epochs to train the model. An epoch is an iteration over the entire `x` and `y` data provided. Note that in conjunction with `initial_epoch`, `epochs` is to be understood as "final epoch". The model is not trained for a number of iterations given by `epochs`, but merely until the epoch of index `epochs` is reached.
verbose	0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch. Note that the progress bar is not particularly useful when

	logged to a file, so verbose=2 is recommended when not running interactively (eg, in a production environment).
`callbacks`	List of `keras.callbacks.Callback` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback) instances. List of callbacks to apply during training. See `tf.keras.callbacks` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks).
`validation_split`	Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the `x` and `y` data provided, before shuffling. This argument is not supported when `x` is a dataset, generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instance.
`validation_data`	Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. Thus, note the fact that the validation loss of data provided using `validation_split` or `validation_data` is not affected by regularization layers like noise and dropuout. `validation_data` will override `validation_split`. `validation_data` could be: • tuple `(x_val, y_val)` of Numpy arrays or tensors • tuple `(x_val, y_val, val_sample_weights)` of Numpy arrays • dataset For the first two cases, `batch_size` must be provided. For the last case, `validation_steps` could be provided. Note that `validation_data` does not support all the data types that are supported in `x`, eg, dict, generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence).
`shuffle`	Boolean (whether to shuffle the training data before each epoch) or str (for 'batch'). This argument is ignored when `x` is a generator. 'batch' is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when `steps_per_epoch` is not `None`.
`class_weight`	Optional dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class.

`sample_weight`	Optional Numpy array of weights for the training samples, used for weighting the loss function (during training only). You can either pass a flat (1D) Numpy array with the same length as the input samples (1:1 mapping between weights and samples), or in the case of temporal data, you can pass a 2D array with shape `(samples, sequence_length)`, to apply a different weight to every timestep of every sample. This argument is not supported when `x` is a dataset, generator, or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instance, instead provide the sample_weights as the third element of `x`.
`initial_epoch`	Integer. Epoch at which to start training (useful for resuming a previous training run).
`steps_per_epoch`	Integer or `None`. Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default `None` is equal to the number of samples in your dataset divided by the batch size, or 1 if that cannot be determined. If x is a `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset, and 'steps_per_epoch' is None, the epoch will run until the input dataset is exhausted. When passing an infinitely repeating dataset, you must specify the `steps_per_epoch` argument. This argument is not supported with array inputs.
`validation_steps`	Only relevant if `validation_data` is provided and is a `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset. Total number of steps (batches of samples) to draw before stopping when performing validation at the end of every epoch. If 'validation_steps' is None, validation will run until the `validation_data` dataset is exhausted. In the case of an infinitely repeated dataset, it will run into an infinite loop. If 'validation_steps' is specified and only part of the dataset will be consumed, the evaluation will start from the beginning of the dataset at each epoch. This ensures that the same validation samples are used every time.
`validation_batch_size`	Integer or `None`. Number of samples per validation batch. If unspecified, will default to `batch_size`. Do not specify the `validation_batch_size` if your data is in the form of datasets, generators, or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instances (since they generate batches).
`validation_freq`	Only relevant if validation data is provided. Integer or `collections_abc.Container` instance (e.g. list, tuple, etc.). If an integer, specifies how many training epochs to run before a new validation run is

	performed, e.g. `validation_freq=2` runs validation every 2 epochs. If a Container, specifies the epochs on which to run validation, e.g. `validation_freq=[1, 2, 10]` runs validation at the end of the 1st, 2nd, and 10th epochs.
`max_queue_size`	Integer. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. Maximum size for the generator queue. If unspecified, `max_queue_size` will default to 10.
`workers`	Integer. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.
`use_multiprocessing`	Boolean. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

Unpacking behavior for iterator-like inputs: A common pattern is to pass a tf.data.Dataset, generator, or tf.keras.utils.Sequence to the `x` argument of fit, which will in fact yield not only features (x) but optionally targets (y) and sample weights. Keras requires that the output of such iterator-likes be unambiguous. The iterator should return a tuple of length 1, 2, or 3, where the optional second and third elements will be used for y and sample_weight respectively. Any other type provided will be wrapped in a length one tuple, effectively treating everything as 'x'. When yielding dicts, they should still adhere to the top-level tuple structure. e.g. `({"x0": x0, "x1": x1}, y)`. Keras will not attempt to separate features, targets, and weights from the keys of a single dict. A notable unsupported data type is the namedtuple. The reason is that it behaves like both an ordered datatype (tuple) and a mapping datatype (dict). So given a namedtuple of the form:
`namedtuple("example_tuple", ["y", "x"])` it is ambiguous whether to reverse the order of the elements when interpreting the value. Even worse is a tuple of the form:
`namedtuple("other_tuple", ["x", "y", "z"])` where it is unclear if the tuple was intended to be unpacked into x, y, and sample_weight or passed through as a single element to `x`. As a result the data processing code will simply raise a ValueError if it encounters a namedtuple. (Along with instructions to remedy the issue.)

Returns

A `History` object. Its `History.history` attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable).

Raises

`RuntimeError`	1. If the model was never compiled or, 2. If `model.fit` is wrapped in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).
`ValueError`	In case of mismatch between the provided input data and what the model expects.

fit_generator

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1791-L1829)

```
enerator(
enerator, steps_per_epoch=None, epochs=1, verbose=1, callbacks=None,
alidation_data=None, validation_steps=None, validation_freq=1,
lass_weight=None, max_queue_size=10, workers=1, use_multiprocessing=False,
huffle=True, initial_epoch=0
```

Fits the model on data yielded batch-by-batch by a Python generator. (deprecated)

ıg: THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Plea odel.fit, which supports generators.

DEPRECATED:

`Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) now supports generators, so there is no longer any need to use this endpoint.

get_layer

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L2365-L2399)

```
.ayer(
ıame=None, index=None
```

Retrieves a layer based on either its name (unique) or index.

If `name` and `index` are both provided, `index` will take precedence. Indices are based on order of horizontal graph traversal (bottom-up).

Arguments

`name`	String, name of layer.
`index`	Integer, index of layer.

Returns

A layer instance.

Raises

`ValueError`	In case of invalid layer name or index.

`load_weights`

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L2109-L2211)

```
.weights(
ilepath, by_name=False, skip_mismatch=False, options=None
```

Loads all layer weights, either from a TensorFlow or an HDF5 weight file.

If `by_name` is False weights are loaded based on the network's topology. This means the architecture should be the same as when the weights were saved. Note that layers that don't have weights are not taken into account in the topological ordering, so adding or removing layers is fine as long as they don't have weights.

If `by_name` is True, weights are loaded into layers only if they share the same name. This is useful for fine-tuning or transfer-learning models where some of the layers have changed.

Only topological loading (`by_name=False`) is supported when loading weights from the TensorFlow format. Note that topological loading differs slightly between TensorFlow and HDF5 formats for user-defined classes inheriting from `tf.keras.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model): HDF5 loads based on a flattened list of weights, while the TensorFlow format loads based on the object-local names of attributes to which layers are assigned in the `Model`'s constructor.

Arguments

`filepath`	String, path to the weights file to load. For weight files in TensorFlow format, this is the file prefix (the same as was passed to `save_weights`).
`by_name`	Boolean, whether to load weights by name or by topological order. Only topological loading is supported for weight files in TensorFlow format.
`skip_mismatch`	Boolean, whether to skip loading of layers where there is a mismatch in the number of weights, or a mismatch in the shape of the weight (only valid when `by_name=True`).
`options`	Optional `tf.train.CheckpointOptions` (https://www.tensorflow.org/api_docs/python/tf/train/CheckpointOptions) object that specifies options for loading weights.

Returns

When loading a weight file in TensorFlow format, returns the same status object as `tf.train.Checkpoint.restore` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint#restore). When graph building, restore ops are run automatically as soon as the network is built (on first call for user-defined classes inheriting from `Model`, immediately if it is already built).

When loading weights in HDF5 format, returns `None`.

Raises

`ImportError`	If h5py is not available and the weight file is in HDF5 format.
`ValueError`	If `skip_mismatch` is set to `True` when `by_name` is `False`.

make_predict_function

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1420-L1484)

```
.predict_function()
```

Creates a function that executes one step of inference.

This method can be overridden to support custom inference logic. This method is called by `Model.predict` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict) and `Model.predict_on_batch` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict_on_batch).

Typically, this method directly controls `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) and `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) settings, and delegates the actual evaluation logic to `Model.predict_step` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict_step).

This function is cached the first time `Model.predict` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict) or `Model.predict_on_batch` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict_on_batch) is called. The cache is cleared whenever `Model.compile` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile) is called.

Returns

Function. The function created by this method should accept a `tf.data.Iterator` (https://www.tensorflow.org/api_docs/python/tf/data/Iterator), and return the outputs of the `Model`.

make_test_function

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1182-L1240)

```
.test_function()
```

Creates a function that executes one step of evaluation.

This method can be overridden to support custom evaluation logic. This method is called by `Model.evaluate` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate) and `Model.test_on_batch` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#test_on_batch).

Typically, this method directly controls `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) and `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) settings, and delegates the actual evaluation logic to `Model.test_step` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#test_step).

This function is cached the first time `Model.evaluate` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate) or `Model.test_on_batch` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#test_on_batch) is called. The cache is cleared whenever `Model.compile` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile) is called.

Returns

Function. The function created by this method should accept a `tf.data.Iterator` (https://www.tensorflow.org/api_docs/python/tf/data/Iterator), and return a `dict` containing values that will be passed to `tf.keras.Callbacks.on_test_batch_end`.

make_train_function

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L762-L822)

```
.train_function()
```

Creates a function that executes one step of training.

This method can be overridden to support custom training logic. This method is called by `Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) and `Model.train_on_batch` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#train_on_batch).

Typically, this method directly controls `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) and `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) settings, and delegates the actual training logic to `Model.train_step` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#train_step).

This function is cached the first time `Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) or `Model.train_on_batch` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#train_on_batch) is called. The cache is cleared whenever `Model.compile` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile) is called.

Returns

Function. The function created by this method should accept a `tf.data.Iterator` (https://www.tensorflow.org/api_docs/python/tf/data/Iterator), and return a `dict` containing values that will be passed to `tf.keras.Callbacks.on_train_batch_end`, such as `{'loss': 0.2, 'accuracy': 0.7}`.

predict

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1486-L1615)

```
.ct(
:, batch_size=None, verbose=0, steps=None, callbacks=None, max_queue_size=10,
orkers=1, use_multiprocessing=False
```

Generates output predictions for the input samples.

Computation is done in batches. This method is designed for performance in large scale inputs. For small amount of inputs that fit in one batch, directly using `__call__` is

recommended for faster execution, e.g., `model(x)`, or `model(x, training=False)` if you have layers such as `tf.keras.layers.BatchNormalization` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization) that behaves differently during inference. Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Arguments

x	Input samples. It could be: • A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs). • A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs). • A `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset. • A generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instance. A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the `Unpacking behavior for iterator-like inputs` section of `Model.fit`.
batch_size	Integer or `None`. Number of samples per batch. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of dataset, generators, or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) instances (since they generate batches).
verbose	Verbosity mode, 0 or 1.
steps	Total number of steps (batches of samples) before declaring the prediction round finished. Ignored with the default value of `None`. If x is a `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data) dataset and `steps` is None, `predict` will run until the input dataset is exhausted.
callbacks	List of `keras.callbacks.Callback` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback) instances. List of callbacks to apply during prediction. See callbacks (/api_docs/python/tf/keras/callbacks).

`max_queue_size`	Integer. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. Maximum size for the generator queue. If unspecified, `max_queue_size` will default to 10.
`workers`	Integer. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.
`use_multiprocessing`	Boolean. Used for generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) input only. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

See the discussion of `Unpacking behavior for iterator-like inputs` for `Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit). Note that Model.predict uses the same interpretation rules as `Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) and `Model.evaluate` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate), so inputs must be unambiguous for all three methods.

Returns

Numpy array(s) of predictions.

Raises

`RuntimeError`	If `model.predict` is wrapped in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).
`ValueError`	In case of mismatch between the provided input data and the model's expectations, or in case a stateful model receives a number of samples that is not a multiple of the batch size.

`predict_generator`

View source
(https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1859-L1883)

```
.ct_generator(
enerator, steps=None, callbacks=None, max_queue_size=10, workers=1,
ıse_multiprocessing=False, verbose=0
```

Generates predictions for the input samples from a data generator. (deprecated)

ıg: THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Plea
ɔdel.predict, which supports generators.

DEPRECATED:

`Model.predict` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict) now supports generators, so there is no longer any need to use this endpoint.

predict_on_batch

View source
(https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1767-L1789)

```
.ct_on_batch(
:
```

Returns predictions for a single batch of samples.

Arguments	
x	Input data. It could be: - A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs). - A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).

Returns

Numpy array(s) of predictions.

Raises

`RuntimeError`	If `model.predict_on_batch` is wrapped in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).
`ValueError`	In case of mismatch between given number of inputs and expectations of the model.

`predict_step`

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1396-L1418)

```
.ct_step(
lata
```

The logic for one inference step.

This method can be overridden to support custom inference logic. This method is called by `Model.make_predict_function` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#make_predict_function).

This method should contain the mathemetical logic for one step of inference. This typically includes the forward pass.

Configuration details for *how* this logic is run (e.g. `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) and `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) settings), should be left to `Model.make_predict_function` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#make_predict_function), which can also be overridden.

Arguments

`data`	A nested structure of `Tensor`s.

Returns

The result of one inference step, typically the output of calling the `Model` on data.

reset_metrics

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1617-L1637)

```
_metrics()
```

Resets the state of all the metrics in the model.

Examples:

```
nputs = tf.keras.layers.Input(shape=(3,))
utputs = tf.keras.layers.Dense(2)(inputs)
odel = tf.keras.models.Model(inputs=inputs, outputs=outputs)
odel.compile(optimizer="Adam", loss="mse", metrics=["mae"])
```

```
 = np.random.random((2, 3))
 = np.random.randint(0, 2, (2, 2))
 = model.fit(x, y, verbose=0)
ssert all(float(m.result()) for m in model.metrics)
```

```
odel.reset_metrics()
ssert all(float(m.result()) == 0 for m in model.metrics)
```

reset_states

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L2283-L2286)

```
_states()
```

save

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1918-L1979)

```
filepath, overwrite=True, include_optimizer=True, save_format=None,
signatures=None, options=None
```

Saves the model to Tensorflow SavedModel or a single HDF5 file.

The savefile includes:

- The model architecture, allowing to re-instantiate the model.
- The model weights.
- The state of the optimizer, allowing to resume training exactly where you left off.

This allows you to save the entirety of the state of a model in a single file.

Saved models can be reinstantiated via `keras.models.load_model` (https://www.tensorflow.org/api_docs/python/tf/keras/models/load_model). The model returned by `load_model` is a compiled model ready to be used (unless the saved model was never compiled in the first place).

Models built with the Sequential and Functional API can be saved to both the HDF5 and SavedModel formats. Subclassed models can only be saved with the SavedModel format.

Note that the model weights may have different scoped names after being loaded. Scoped names include the model/layer names, such as `"dense_1/kernel:0"`. It is recommended that you use the layer properties to access specific variables, e.g. `model.get_layer("dense_1").kernel`.

Arguments

`filepath`	String, PathLike, path to SavedModel or H5 file to save the model.
`overwrite`	Whether to silently overwrite any existing file at the target location, or

	provide the user with a manual prompt.
`include_optimizer`	If True, save optimizer's state together.
`save_format`	Either `'tf'` or `'h5'`, indicating whether to save the model to Tensorflow SavedModel or HDF5. Defaults to 'tf' in TF 2.X, and 'h5' in TF 1.X.
`signatures`	Signatures to save with the SavedModel. Applicable to the 'tf' format only. Please see the `signatures` argument in `tf.saved_model.save` (https://www.tensorflow.org/api_docs/python/tf/saved_model/save) for details.
`options`	Optional `tf.saved_model.SaveOptions` (https://www.tensorflow.org/api_docs/python/tf/saved_model/SaveOptions) object that specifies options for saving to SavedModel.

Example:

```
keras.models import load_model

..save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
iodel  # deletes the existing model

:urns a compiled model
:ntical to the previous one
. = load_model('my_model.h5')
```

save_weights

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1981-L2107)

```
.weights(
'ilepath, overwrite=True, save_format=None, options=None
```

Saves all layer weights.

Either saves in HDF5 or in TensorFlow format based on the `save_format` argument.

When saving in HDF5 format, the weight file has:

- `layer_names` (attribute), a list of strings (ordered names of model layers).
- For every layer, a `group` named `layer.name`
    - For every such layer group, a group attribute `weight_names`, a list of strings (ordered names of weights tensor of the layer).
    - For every weight in the layer, a dataset storing the weight value, named after the weight tensor.

When saving in TensorFlow format, all objects referenced by the network are saved in the same format as `tf.train.Checkpoint` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint), including any `Layer` instances or `Optimizer` instances assigned to object attributes. For networks constructed from inputs and outputs using `tf.keras.Model(inputs, outputs)`, `Layer` instances used by the network are tracked/saved automatically. For user-defined classes which inherit from `tf.keras.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model), `Layer` instances must be assigned to object attributes, typically in the constructor. See the documentation of `tf.train.Checkpoint` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint) and `tf.keras.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model) for details.

While the formats are the same, do not mix `save_weights` and `tf.train.Checkpoint` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint). Checkpoints saved by `Model.save_weights` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#save_weights) should be loaded using `Model.load_weights` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#load_weights). Checkpoints saved using `tf.train.Checkpoint.save` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint#save) should be restored using the corresponding `tf.train.Checkpoint.restore` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint#restore). Prefer `tf.train.Checkpoint` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint) over `save_weights` for training checkpoints.

The TensorFlow format matches objects and variables by starting at a root object, `self` for `save_weights`, and greedily matching attribute names. For `Model.save` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#save) this is the `Model`, and for `Checkpoint.save` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint#save) this is the `Checkpoint` even if the `Checkpoint` has a model attached. This means saving a `tf.keras.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model) using

`save_weights` and loading into a `tf.train.Checkpoint` (https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint) with a `Model` attached (or vice versa) will not match the `Model`'s variables. See the guide to training checkpoints (https://www.tensorflow.org/guide/checkpoint) for details on the TensorFlow format.

Arguments

`filepath`	String or PathLike, path to the file to save the weights to. When saving in TensorFlow format, this is the prefix used for checkpoint files (multiple files are generated). Note that the '.h5' suffix causes weights to be saved in HDF5 format.
`overwrite`	Whether to silently overwrite any existing file at the target location, or provide the user with a manual prompt.
`save_format`	Either 'tf' or 'h5'. A `filepath` ending in '.h5' or '.keras' will default to HDF5 if `save_format` is `None`. Otherwise `None` defaults to 'tf'.
`options`	Optional `tf.train.CheckpointOptions` (https://www.tensorflow.org/api_docs/python/tf/train/CheckpointOptions) object that specifies options for saving weights.

Raises

`ImportError`	If h5py is not available when attempting to save in HDF5 format.
`ValueError`	For invalid/unknown format arguments.

summary

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L2332-L2359)

```
ıry(
.ine_length=None, positions=None, print_fn=None
```

Prints a string summary of the network.

Arguments

`line_length`	Total length of printed lines (e.g. set this to adapt the display to different terminal window sizes).
`positions`	Relative or absolute positions of log elements in each line. If not provided, defaults to `[.33, .55, .67, 1.]`.
`print_fn`	Print function to use. Defaults to `print`. It will be called on each line of the summary. You can set it to a custom function in order to capture the string summary.

Raises

`ValueError`	if `summary()` is called before the model is built.

test_on_batch

View source
(https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1708-L1765)

```
.on_batch(
:, y=None, sample_weight=None, reset_metrics=True, return_dict=False
```

Test the model on a single batch of samples.

Arguments

x	Input data. It could be: - A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs). - A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs). • A dict mapping input names to the corresponding array/tensors, if the model has named inputs.
y	Target data. Like the input data `x`, it could be either Numpy array(s) or TensorFlow tensor(s). It should be consistent with `x` (you cannot have Numpy inputs and tensor targets, or inversely).
`sample_weight`	Optional array of the same length as x, containing weights to apply to the model's loss for each sample. In the case of temporal data, you can pass a 2D array with shape (samples, sequence_length), to apply a different weight to every timestep of every sample.

`reset_metrics`	If `True`, the metrics returned will be only for this batch. If `False`, the metrics will be statefully accumulated across batches.
`return_dict`	If `True`, loss and metric results are returned as a dict, with each key being the name of the metric. If `False`, they are returned as a list.

Returns

Scalar test loss (if the model has a single output and no metrics) or list of scalars (if the model has multiple outputs and/or metrics). The attribute `model.metrics_names` will give you the display labels for the scalar outputs.

Raises

`RuntimeError`	If `model.test_on_batch` is wrapped in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).
`ValueError`	In case of invalid user-provided arguments.

test_step

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1148-L1180)

```
.step(
lata
```

The logic for one evaluation step.

This method can be overridden to support custom evaluation logic. This method is called by `Model.make_test_function` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#make_test_function).

This function should contain the mathemetical logic for one step of evaluation. This typically includes the forward pass, loss calculation, and metrics updates.

Configuration details for *how* this logic is run (e.g. `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) and `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) settings), should be left to `Model.make_test_function`

(https://www.tensorflow.org/api_docs/python/tf/keras/Model#make_test_function), which can also be overridden.

Arguments

`data`	A nested structure of `Tensor`s.

Returns

A `dict` containing values that will be passed to `tf.keras.callbacks.CallbackList.on_train_batch_end` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/CallbackList#on_train_batch_end). Typically, the values of the `Model`'s metrics are returned.

to_json

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L2241-L2256)

```
son(
**kwargs
```

Returns a JSON string containing the network configuration.

To load a network from a JSON save file, use `keras.models.model_from_json(json_string, custom_objects={})` (https://www.tensorflow.org/api_docs/python/tf/keras/models/model_from_json).

Arguments

`**kwargs`	Additional keyword arguments to be passed to `json.dumps()`.

Returns

A JSON string.

to_yaml

View source
(https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L2258-L2281)

```
ml(
*kwargs
```

Returns a yaml string containing the network configuration.

To load a network from a yaml save file, use
`keras.models.model_from_yaml(yaml_string, custom_objects={})`
(https://www.tensorflow.org/api_docs/python/tf/keras/models/model_from_yaml).

`custom_objects` should be a dictionary mapping the names of custom losses / layers / etc to the corresponding functions / classes.

Arguments

`**kwargs`	Additional keyword arguments to be passed to `yaml.dump()`.

Returns

A YAML string.

Raises

`ImportError`	if yaml module is not found.

train_on_batch

View source
(https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L1639-L1706)

```
_on_batch(
, y=None, sample_weight=None, class_weight=None, reset_metrics=True,
eturn_dict=False
```

Runs a single gradient update on a single batch of data.

Arguments

x	Input data. It could be: • A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs). • A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs). • A dict mapping input names to the corresponding array/tensors, if the model has named inputs.
y	Target data. Like the input data `x`, it could be either Numpy array(s) or TensorFlow tensor(s). It should be consistent with `x` (you cannot have Numpy inputs and tensor targets, or inversely).
`sample_weight`	Optional array of the same length as x, containing weights to apply to the model's loss for each sample. In the case of temporal data, you can pass a 2D array with shape (samples, sequence_length), to apply a different weight to every timestep of every sample.
`class_weight`	Optional dictionary mapping class indices (integers) to a weight (float) to apply to the model's loss for the samples from this class during training. This can be useful to tell the model to "pay more attention" to samples from an under-represented class.
`reset_metrics`	If `True`, the metrics returned will be only for this batch. If `False`, the metrics will be statefully accumulated across batches.
`return_dict`	If `True`, loss and metric results are returned as a dict, with each key being the name of the metric. If `False`, they are returned as a list.

Returns

Scalar training loss (if the model has a single output and no metrics) or list of scalars (if the model has multiple outputs and/or metrics). The attribute `model.metrics_names` will give you the display labels for the scalar outputs.

Raises

`RuntimeError`	If `model.train_on_batch` is wrapped in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).
`ValueError`	In case of invalid user-provided arguments.

train_step

View source (https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/engine/training.py#L716-L760)

```
_step(
ata
```

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by `Model.make_train_function` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#make_train_function).

This method should contain the mathemetical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) and `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) settings), should be left to `Model.make_train_function` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#make_train_function), which can also be overridden.

Arguments

`data`	A nested structure of `Tensor`s.

Returns

A `dict` containing values that will be passed to `tf.keras.callbacks.CallbackList.on_train_batch_end` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/CallbackList#on_train_batch_end). Typically, the values of the `Model`'s metrics are returned. Example: `{'loss': 0.2, 'accuracy': 0.7}`.

Last updated 2020-10-15 UTC.