tf.keras.Model

View source on GitHub (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L70-L3991)

A model grouping layers into an object with training/inference features.

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

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)

```
tf.keras.Model(
    *args, **kwargs
)
```

Used in the notebooks

Used in the guide	Used in the tutorials
• Use TF1.x models in TF2 workflows (https://www.tensorflow.org/guide/migrate/model_mapping) • Migrate `tf.feature_column`s to Keras preprocessing layers (https://www.tensorflow.org/guide/migrate/migrating_feature_columns) • Debug a TensorFlow 2 migrated training pipeline (https://www.tensorflow.org/guide/migrate/migration_debugging)	• Time series forecasting (https://www.tensorflow. • Load a pandas DataFram (https://www.tensorflow. • Parameter server training (https://www.tensorflow.

Used in the guide	Used in the tutorials
• Introduction to modules, layers, and models (https://www.tensorflow.org/guide/intro_to_modules) • Basic training loops (https://www.tensorflow.org/guide/basic_training_loops)	• Intro to Autoencoders (https://www.tensorflow. • Learned data compressio (https://www.tensorflow.

Args

`inputs`	The input(s) of the model: a `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) object or a combination of `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) objects in a dict, list or tuple.
`outputs`	The output(s) of the model: a tensor that originated from `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) objects or a combination of such tensors in a dict, list or tuple. 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:

```
import tensorflow as tf

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

Note: Only dicts, lists, and tuples of input tensors are supported. Nested inputs are not supported (e.g. lists of list or dicts of dict).

A new Functional API model can also be created by using the intermediate tensors. This enables you to quickly extract sub-components of the model.

Example:

```
inputs = keras.Input(shape=(None, None, 3))
processed = keras.layers.RandomCrop(width=32, height=32)(inputs)
conv = keras.layers.Conv2D(filters=2, kernel_size=3)(processed)
pooling = keras.layers.GlobalAveragePooling2D()(conv)
feature = keras.layers.Dense(10)(pooling)

full_model = keras.Model(inputs, feature)
backbone = keras.Model(processed, conv)
activations = keras.Model(conv, feature)
```

Note that the `backbone` and `activations` models are not created with `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) objects, but with the tensors that are originated from `keras.Input` (https://www.tensorflow.org/api_docs/python/tf/keras/Input) objects. Under the hood, the layers and weights will be shared across these models, so that user can train the `full_model`, and use `backbone` or `activations` to do feature extraction. The inputs and outputs of the model can be nested structures of tensors as well, and the created models are standard Functional API models that support all the existing APIs.

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()`.

```
import tensorflow as tf

class MyModel(tf.keras.Model):

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

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

model = 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:

```
import tensorflow as tf

class MyModel(tf.keras.Model):

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

  def call(self, inputs, training=False):
    x = self.dense1(inputs)
    if training:
      x = self.dropout(x, training=training)
    return self.dense2(x)

model = 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_reduction_method`	The method employed to reduce per-replica values during training. Unless specified, the value "auto" will be assumed, indicating that the reduction strategy should be chosen based on the current running environment. See `reduce_per_replica` function for more details.
`distribute_strategy`	The `tf.distribute.Strategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy) this model was created under.
`jit_compile`	Specify whether to compile the model with XLA. XLA (https://www.tensorflow.org/xla) is an optimizing compiler for machine learning. `jit_compile` is not enabled by default. Note that `jit_compile=True` may not necessarily work for all models. For more information on supported operations please refer to the XLA documentation (https://www.tensorflow.org/xla). Also refer to known XLA issues (https://www.tensorflow.org/xla/known_issues) for more details.
`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

call

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L571-L602)

```
call(
    inputs, training=None, mask=None
)
```

Calls the model on new inputs and returns the outputs as tensors.

In this case `call()` just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Note: This method should not be called directly. It is only meant to be overridden when subclassing **`tf.keras.Model`** (https://www.tensorflow.org/api_docs/python/tf/keras/Model). To call a model on an input, always use the **`__call__()`** method, i.e. **`model(inputs)`**, which relies on the underlying **`call()`** method.

Args

`inputs`	Input tensor, or dict/list/tuple of input tensors.
`training`	Boolean or boolean scalar tensor, indicating whether to run the **`Network`** in training mode or inference mode.
`mask`	A mask or list of masks. A mask can be either a boolean tensor or None (no mask). For more details, check the guide here (https://www.tensorflow.org/guide/keras/masking_and_padding).

Returns

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

compile

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L604-L790)

```
compile(
    optimizer='rmsprop',
    loss=None,
    metrics=None,
    loss_weights=None,
    weighted_metrics=None,
    run_eagerly=None,
    steps_per_execution=None,
    jit_compile=None,
    pss_evaluation_shards=0,
    **kwargs
)
```

Configures the model for training.

Example:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(),
                       tf.keras.metrics.FalseNegatives()])
```

Args	
`optimizer`	String (name of optimizer) or optimizer instance. See `tf.keras.optimizers` (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers).
`loss`	Loss function. May be a string (name of loss function), or a `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). A loss function is any callable with the signature `loss = fn(y_true, y_pred)`, where `y_true` are the ground truth values, and `y_pred` are the model's predictions. `y_true` should have shape `(batch_size, d0, .. dN)` (except in the case of sparse loss functions such as sparse categorical crossentropy which expects integer arrays of shape `(batch_size, d0, .. dN-1)`). `y_pred` should have shape `(batch_size, d0, .. dN)`. The loss function should return a float tensor. If a custom `Loss` instance is used and reduction is set to `None`, return value has shape `(batch_size, d0, .. dN-1)` i.e. 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, unless `loss_weights` is specified.
`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 to specify a metric or a list of metrics for each output, 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 shapes of the targets and of the model output. We do a similar conversion for the strings 'crossentropy' and 'ce' as well. The metrics passed here are evaluated without sample weighting; if you would like sample weighting to apply, you can specify your metrics via the `weighted_metrics` argument instead.
`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. 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). `run_eagerly=True` is not supported when using `tf.distribute.experimental.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy) . Defaults to `False`.
`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. 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. Note that if `steps_per_execution` is set to `N`, `Callback.on_batch_begin` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback#on_batch_begin) and `Callback.on_batch_end` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback#on_batch_end) methods will only be called every `N` batches (i.e. before/after each `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) execution). Defaults to `1`.
`jit_compile`	If `True`, compile the model training step with XLA. XLA (https://www.tensorflow.org/xla) is an optimizing compiler for machine learning. `jit_compile` is not enabled for by default. Note that `jit_compile=True` may not necessarily work for all models. For more information on supported operations please refer to the XLA documentation (https://www.tensorflow.org/xla). Also refer to known XLA issues (https://www.tensorflow.org/xla/known_issues) for more details.
`pss_evaluation_shards`	Integer or 'auto'. Used for `tf.distribute.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy) training only. This arg sets the number of shards to split the dataset into, to enable an exact visitation guarantee for evaluation, meaning the model will be applied to each dataset element exactly once, even if workers fail. The dataset must be sharded to ensure separate workers do not process the same data. The number of shards should

	be at least the number of workers for good performance. A value of 'auto' turns on exact evaluation and uses a heuristic for the number of shards based on the number of workers. 0, meaning no visitation guarantee is provided. NOTE: Custom implementations of `Model.test_step` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#test_step) will be ignored when doing exact evaluation. Defaults to `0`.
`**kwargs`	Arguments supported for backwards compatibility only.

compile_from_config

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3557-L3582)

```
compile_from_config(
    config
)
```

Compiles the model with the information given in config.

This method uses the information in the config (optimizer, loss, metrics, etc.) to compile the model.

Args

`config`	Dict containing information for compiling the model.

compute_loss

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1087-L1141)

```
compute_loss(
    x=None, y=None, y_pred=None, sample_weight=None
)
```

Compute the total loss, validate it, and return it.

Subclasses can optionally override this method to provide custom loss computation logic.

Example:

```
class MyModel(tf.keras.Model):

  def __init__(self, *args, **kwargs):
    super(MyModel, self).__init__(*args, **kwargs)
    self.loss_tracker = tf.keras.metrics.Mean(name='loss')

  def compute_loss(self, x, y, y_pred, sample_weight):
    loss = tf.reduce_mean(tf.math.squared_difference(y_pred, y))
    loss += tf.add_n(self.losses)
    self.loss_tracker.update_state(loss)
    return loss

  def reset_metrics(self):
    self.loss_tracker.reset_states()

  @property
  def metrics(self):
    return [self.loss_tracker]

tensors = tf.random.uniform((10, 10)), tf.random.uniform((10,))
dataset = tf.data.Dataset.from_tensor_slices(tensors).repeat().batch(1)

inputs = tf.keras.layers.Input(shape=(10,), name='my_input')
outputs = tf.keras.layers.Dense(10)(inputs)
model = MyModel(inputs, outputs)
model.add_loss(tf.reduce_sum(outputs))

optimizer = tf.keras.optimizers.SGD()
model.compile(optimizer, loss='mse', steps_per_execution=10)
model.fit(dataset, epochs=2, steps_per_epoch=10)
print('My custom loss: ', model.loss_tracker.result().numpy())
```

Args

`x`	Input data.
`y`	Target data.
`y_pred`	Predictions returned by the model (output of `model(x)`)
`sample_weight`	Sample weights for weighting the loss function.

Returns

The total loss as a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor), or **None** if no loss results (which is the case when called by `Model.test_step` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#test_step)).

compute_metrics

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1143-L1180)

```
compute_metrics(
    x, y, y_pred, sample_weight
)
```

Update metric states and collect all metrics to be returned.

Subclasses can optionally override this method to provide custom metric updating and collection logic.

Example:

```
class MyModel(tf.keras.Sequential):

  def compute_metrics(self, x, y, y_pred, sample_weight):

    # This super call updates `self.compiled_metrics` and returns
    # results for all metrics listed in `self.metrics`.
    metric_results = super(MyModel, self).compute_metrics(
        x, y, y_pred, sample_weight)

    # Note that `self.custom_metric` is not listed in `self.metrics`.
    self.custom_metric.update_state(x, y, y_pred, sample_weight)
    metric_results['custom_metric_name'] = self.custom_metric.result()
    return metric_results
```

Args

x	Input data.

`y`	Target data.
`y_pred`	Predictions returned by the model (output of `model.call(x)`)
`sample_weight`	Sample weights for weighting the loss function.

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 metrics listed in `self.metrics` are returned. Example: `{'loss': 0.2, 'accuracy': 0.7}`.

evaluate

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2023-L2218)

```
evaluate(
    x=None,
    y=None,
    batch_size=None,
    verbose='auto',
    sample_weight=None,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    return_dict=False,
    **kwargs
)
```

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

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

Args

`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`	"`auto`", 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = single line. "`auto`" becomes 1 for most cases, and to 2 when used with `ParameterServerStrategy`. Note that the progress bar is not particularly useful when logged to a file, so `verbose=2` is recommended when not running interactively (e.g. in a production environment). Defaults to 'auto'.
`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 (https://www.tensorflow.org/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.
`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.
`**kwargs`	Unused at this time.

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 a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).

export

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3584-L3617)

```
export(
    filepath
)
```

Create a SavedModel artifact for inference (e.g. via TF-Serving).

This method lets you export a model to a lightweight SavedModel artifact that contains the model's forward pass only (its `call()` method) and can be served via e.g. TF-Serving. The forward pass is registered under the name `serve()` (see example below).

The original code of the model (including any custom layers you may have used) is *no longer* necessary to reload the artifact -- it is entirely standalone.

Args

`filepath`	`str` or `pathlib.Path` object. Path where to save the artifact.

Example:

```
# Create the artifact
model.export("path/to/location")

# Later, in a different process / environment...
reloaded_artifact = tf.saved_model.load("path/to/location")
```

```
predictions = reloaded_artifact.serve(input_data)
```

If you would like to customize your serving endpoints, you can use the lower-level `keras.export.ExportArchive` (https://www.tensorflow.org/api_docs/python/tf/keras/export/ExportArchive) class. The `export()` method relies on `ExportArchive` internally.

fit

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1392-L1823)

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

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

Args

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 `tf.keras.utils.experimental.DatasetCreator` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/experimental/DatasetCreator), which wraps a callable that takes a single argument of type `tf.distribute.InputContext` (https://www.tensorflow.org/api_docs/python/tf/distribute/InputContext), and returns a `tf.data.Dataset` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset). `DatasetCreator` should be used when users prefer to specify the per-replica batching and sharding logic for the `Dataset`. See `tf.keras.utils.experimental.DatasetCreator` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/experimental/DatasetCreator) doc for more information. A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given below. If these include `sample_weights` as a third component, note that sample weighting applies to the `weighted_metrics` argument but not the `metrics` argument in `compile()`. If using `tf.distribute.experimental.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy), only `DatasetCreator` type is supported for `x`.
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 (unless the `steps_per_epoch` flag is set to something other than None). 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`	'auto', 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch. 'auto' becomes 1 for most cases, but 2 when used with `ParameterServerStrategy`. 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). Defaults to 'auto'.
`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). Note `tf.keras.callbacks.ProgbarLogger` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ProgbarLogger) and `tf.keras.callbacks.History` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/History) callbacks are created automatically and need not be passed into `model.fit`. `tf.keras.callbacks.ProgbarLogger` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ProgbarLogger) is created or not based on `verbose` argument to `model.fit`. Callbacks with batch-level calls are currently unsupported with `tf.distribute.experimental.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy), and users are advised to implement epoch-level calls instead with an appropriate `steps_per_epoch` value.
`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. If both `validation_data` and `validation_split` are provided, `validation_data` will override `validation_split`. `validation_split` is not yet supported with `tf.distribute.experimental.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy).
`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 dropout. `validation_data` will override `validation_split`. `validation_data` could be: • A tuple `(x_val, y_val)` of Numpy arrays or tensors. • A tuple `(x_val, y_val, val_sample_weights)` of NumPy arrays. • A `tf.data.Dataset` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset). • A Python generator or `keras.utils.Sequence` (https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence) returning `(inputs, targets)` or `(inputs, targets, sample_weights)`. `validation_data` is not yet supported with `tf.distribute.experimental.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy).
`shuffle`	Boolean (whether to shuffle the training data before each epoch) or str (for 'batch'). This argument is ignored when `x` is a generator or an object of tf.data.Dataset. '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. When `class_weight` is

	specified and targets have a rank of 2 or greater, either `y` must be one-hot encoded, or an explicit final dimension of `1` must be included for sparse class labels.
`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`. Note that sample weighting does not apply to metrics specified via the `metrics` argument in `compile()`. To apply sample weighting to your metrics, you can specify them via the `weighted_metrics` in `compile()` instead.
`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. If `steps_per_epoch=-1` the training will run indefinitely with an infinitely repeating dataset. This argument is not supported with array inputs. When using `tf.distribute.experimental.ParameterServerStrategy` (https://www.tensorflow.org/api_docs/python/tf/distribute/experimental/ParameterServerStrategy): • `steps_per_epoch=None` is not supported.
`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.
`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 or when the input data is empty.

get_compile_config

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3545-L3555)

```
get_compile_config()
```

Returns a serialized config with information for compiling the model.

This method returns a config dictionary containing all the information (optimizer, loss, metrics, etc.) with which the model was compiled.

Returns

A dict containing information for compiling the model.

get_layer

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3429-L3470)

```
get_layer(
    name=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).

Args

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

Returns

A layer instance.

get_metrics_result

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1182-L1201)

```
get_metrics_result()
```

Returns the model's metrics values as a dict.

If any of the metric result is a dict (containing multiple metrics), each of them gets added to the top level returned dict of this method.

Returns

A `dict` containing values of the metrics listed in `self.metrics`.

`Example` `{'loss': 0.2, 'accuracy': 0.7}`.

get_weight_paths

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3472-L3543)

```
get_weight_paths()
```

Retrieve all the variables and their paths for the model.

The variable path (string) is a stable key to identify a `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable) instance owned by the model. It can be used to specify variable-specific configurations (e.g. DTensor, quantization) from a global view.

This method returns a dict with weight object paths as keys and the corresponding `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable) instances as values.

Note that if the model is a subclassed model and the weights haven't been initialized, an empty dict will be returned.

Returns

A dict where keys are variable paths and values are `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable) instances.

Example:

```
class SubclassModel(tf.keras.Model):

  def __init__(self, name=None):
    super().__init__(name=name)
    self.d1 = tf.keras.layers.Dense(10)
    self.d2 = tf.keras.layers.Dense(20)
```

```
  def call(self, inputs):
    x = self.d1(inputs)
    return self.d2(x)

model = SubclassModel()
model(tf.zeros((10, 10)))
weight_paths = model.get_weight_paths()
# weight_paths:
# {
#    'd1.kernel': model.d1.kernel,
#    'd1.bias': model.d1.bias,
#    'd2.kernel': model.d2.kernel,
#    'd2.bias': model.d2.bias,
# }

# Functional model
inputs = tf.keras.Input((10,), batch_size=10)
x = tf.keras.layers.Dense(20, name='d1')(inputs)
output = tf.keras.layers.Dense(30, name='d2')(x)
model = tf.keras.Model(inputs, output)
d1 = model.layers[1]
d2 = model.layers[2]
weight_paths = model.get_weight_paths()
# weight_paths:
# {
#    'd1.kernel': d1.kernel,
#    'd1.bias': d1.bias,
#    'd2.kernel': d2.kernel,
#    'd2.bias': d2.bias,
# }
```

load_weights

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3080-L3138)

```
load_weights(
    filepath, skip_mismatch=False, by_name=False, options=None
)
```

Loads all layer weights from a saved files.

The saved file could be a SavedModel file, a `.keras` file (v3 saving format), or a file created via `model.save_weights()`.

By default, 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.

Partial weight loading

If you have modified your model, for instance by adding a new layer (with weights) or by changing the shape of the weights of a layer, you can choose to ignore errors and continue loading by setting `skip_mismatch=True`. In this case any layer with mismatching weights will be skipped. A warning will be displayed for each skipped layer.

Weight loading by name

If your weights are saved as a `.h5` file created via `model.save_weights()`, you can use the argument `by_name=True`.

In this case, 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.

Note that only topological loading (`by_name=False`) is supported when loading weights from the `.keras` v3 format or from the TensorFlow SavedModel format.

Args

`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()`). This can also be a path to a SavedModel or a `.keras` file (v3 saving format) saved via `model.save()`.
`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 weights.
`by_name`	Boolean, whether to load weights by name or by topological order. Only topological loading is supported for weight files in the `.keras` v3 format or in the TensorFlow SavedModel format.
`options`	Optional `tf.train.CheckpointOptions` (https://www.tensorflow.org/api_docs/python/tf/train/CheckpointOptions) object that specifies options for loading weights (only valid for a SavedModel file).

`make_predict_function`

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2285-L2374)

```
make_predict_function(
    force=False
)
```

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. You can skip the cache and generate again the function with `force=True`.

Args

`force`	Whether to regenerate the predict function and skip the cached function if available.

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/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1913-L2021)

```
make_test_function(
    force=False
)
```

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. You can skip the cache and generate again the function with `force=True`.

Args

`force`	Whether to regenerate the test function and skip the cached function if available.

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/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1271-L1390)

```
make_train_function(
    force=False
)
```

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. You can skip the cache and generate again the function with `force=True`.

Args

`force`	Whether to regenerate the train function and skip the cached function if available.

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/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2376-L2598)

```
predict(
    x,
    batch_size=None,
    verbose='auto',
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

Generates output predictions for the input samples.

Computation is done in batches. This method is designed for batch processing of large numbers of inputs. It is not intended for use inside of loops that iterate over your data and process small numbers of inputs at a time.

For small numbers of inputs that fit in one batch, directly use `__call__()` 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 behave differently during inference. You may pair the individual model call with a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) for additional performance inside your inner loop. If you need access to numpy array values instead of tensors after your model call, you can use `tensor.numpy()` to get the numpy array value of an eager tensor.

Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Note: See this FAQ entry (https://keras.io/getting_started/faq/#whats-the-difference-between-model-methods-predict-and-call) for more details about the difference between **`Model`** methods **`predict()`** and **`__call__()`**.

Args

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`	`"auto"`, 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = single line. `"auto"` becomes 1 for most cases, and to 2 when used with `ParameterServerStrategy`. Note that the progress bar is not particularly useful when logged to a file, so `verbose=2` is recommended when not running interactively (e.g. in a production environment). Defaults to 'auto'.
`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 (https://www.tensorflow.org/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.
`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 a `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_on_batch

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2753-L2778)

```
predict_on_batch(
    x
```

```
)
```

Returns predictions for a single batch of samples.

Args

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 a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).

`predict_step`

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2262-L2283)

```
predict_step(
    data
)
```

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 mathematical 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.

Args

`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/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2600-L2620)

```
reset_metrics()
```

Resets the state of all the metrics in the model.

Examples:

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

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

```
>>> model.reset_metrics()
>>> assert all(float(m.result()) == 0 for m in model.metrics)
```

reset_states

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3307-L3312)

```
reset_states()
```

save

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2946-L3006)

```
save(
    filepath, overwrite=True, save_format=None, **kwargs
)
```

Saves a model as a TensorFlow SavedModel or HDF5 file.

See the Serialization and Saving guide (https://keras.io/guides/serialization_and_saving/) for details.

Args

`model`	Keras model instance to be saved.
`filepath`	`str` or `pathlib.Path` object. Path where to save the model.
`overwrite`	Whether we should overwrite any existing model at the target location, or instead ask the user via an interactive prompt.
`save_format`	Either `"keras"`, `"tf"`, `"h5"`, indicating whether to save the model in the native Keras format (`.keras`), in the TensorFlow SavedModel format (referred to as "SavedModel" below), or in the legacy HDF5 format (`.h5`). Defaults to `"tf"` in TF 2.X, and `"h5"` in TF 1.X.

SavedModel format arguments: include_optimizer: Only applied to SavedModel and legacy HDF5 formats. If False, do not save the optimizer state. Defaults to `True`. signatures: Only applies to SavedModel format. Signatures to save with the SavedModel. See the `signatures` argument in `tf.saved_model.save` (https://www.tensorflow.org/api_docs/python/tf/saved_model/save) for details. options: Only applies to SavedModel format. `tf.saved_model.SaveOptions` (https://www.tensorflow.org/api_docs/python/tf/saved_model/SaveOptions) object that specifies SavedModel saving options. save_traces: Only applies to SavedModel format. When enabled, the SavedModel will store the function traces for each layer. This can be disabled, so that only the configs of each layer are stored. Defaults to `True`. Disabling this will decrease serialization time and reduce file size, but it requires that all custom layers/models implement a `get_config()` method.

Example:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, input_shape=(3,)),
    tf.keras.layers.Softmax()])
model.save("model.keras")
loaded_model = tf.keras.models.load_model("model.keras")
x = tf.random.uniform((10, 3))
assert np.allclose(model.predict(x), loaded_model.predict(x))
```

Note that `model.save()` is an alias for `tf.keras.models.save_model()` (https://www.tensorflow.org/api_docs/python/tf/keras/saving/save_model).

save_spec

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3661-L3701)

```
save_spec(
    dynamic_batch=True
)
```

Returns the `tf.TensorSpec` (https://www.tensorflow.org/api_docs/python/tf/TensorSpec) of call args as a tuple `(args, kwargs)`.

This value is automatically defined after calling the model for the first time. Afterwards, you can use it when exporting the model for serving:

```
model = tf.keras.Model(...)

@tf.function
def serve(*args, **kwargs):
  outputs = model(*args, **kwargs)
  # Apply postprocessing steps, or add additional outputs.
  ...
  return outputs

# arg_specs is `[tf.TensorSpec(...), ...]`. kwarg_specs, in this
# example, is an empty dict since functional models do not use keyword
# arguments.
arg_specs, kwarg_specs = model.save_spec()

model.save(path, signatures={
  'serving_default': serve.get_concrete_function(*arg_specs,
                                                 **kwarg_specs)
})
```

Args

`dynamic_batch`	Whether to set the batch sizes of all the returned `tf.TensorSpec` (https://www.tensorflow.org/api_docs/python/tf/TensorSpec) to `None`. (Note that when defining functional or Sequential models with `tf.keras.Input([...], batch_size=X)`, the batch size will always be preserved). Defaults to `True`.

Returns

If the model inputs are defined, returns a tuple `(args, kwargs)`. All elements in `args` and `kwargs` are `tf.TensorSpec` (https://www.tensorflow.org/api_docs/python/tf/TensorSpec). If the model inputs are not defined, returns `None`. The model inputs are automatically set when calling the model, `model.fit`, `model.evaluate` or `model.predict`.

save_weights

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3008-L3078)

```
save_weights(
    filepath, 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)` (https://www.tensorflow.org/api_docs/python/tf/keras/Model), `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.

Args

`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` becomes 'tf'. Defaults to `None`.
`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.

summary

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3363-L3416)

```
summary(
    line_length=None,
    positions=None,
    print_fn=None,
    expand_nested=False,
    show_trainable=False,
    layer_range=None
)
```

Prints a string summary of the network.

Args

`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, becomes `[0.3, 0.6, 0.70, 1.]`. Defaults to `None`.
`print_fn`	Print function to use. By default, prints to `stdout`. If `stdout` doesn't work in your environment, change 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.
`expand_nested`	Whether to expand the nested models. Defaults to `False`.
`show_trainable`	Whether to show if a layer is trainable. Defaults to `False`.
`layer_range`	a list or tuple of 2 strings, which is the starting layer name and ending layer name (both inclusive) indicating the range of layers to be printed in summary. It also accepts regex patterns instead of exact name. In such case, start predicate will be the first element it matches to `layer_range[0]` and the end predicate will be the last element it matches to `layer_range[1]`. By default `None` which considers all layers of model.

Raises

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

test_on_batch

View source
(https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2692-L2751)

```
test_on_batch(
    x, y=None, sample_weight=None, reset_metrics=True, return_dict=False
)
```

Test the model on a single batch of samples.

Args

`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 a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).

test_step

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1825-L1853)

```
test_step(
    data
)
```

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 mathematical 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.

Args

`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/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3261-L3277)

```
to_json(
    **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).

Args

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

Returns

A JSON string.

to_yaml

View source
(https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L3279-L3305)

```
to_yaml(
    **kwargs
)
```

Returns a yaml string containing the network configuration.

Note: Since TF 2.6, this method is no longer supported and will raise a RuntimeError.

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.

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

Returns
A YAML string.

Raises	
`RuntimeError`	announces that the method poses a security risk

`train_on_batch`

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L2622-L2690)

```
train_on_batch(
    x,
    y=None,
    sample_weight=None,
    class_weight=None,
    reset_metrics=True,
    return_dict=False
)
```

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

Args	
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).
`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. When `class_weight` is specified and targets have a rank of 2 or greater, either `y` must be one-hot encoded, or an explicit final dimension of `1` must be included for sparse class labels.
`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 a `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function).

`train_step`

View source (https://github.com/keras-team/keras/tree/v2.13.1/keras/engine/training.py#L1051-L1085)

```
train_step(
    data
)
```

The logic for one training step.

This method can be overridden to support custom training logic. For concrete examples of how to override this method see Customizing what happens in fit (https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit). 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 mathematical 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.

Args

`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}`.