Introduction to Callbacks

Callbacks

- Provides some functionality at various stages of training
- Subclasses `tf.keras.callbacks.Callback`
- Useful in understanding a model's state during training
  - internal states
  - statistics e.g., losses and metrics

Training specific methods

```python
class Callback(object):
  def __init__(self):
    self.validation_data = None
    self.model = None

  def on_epoch_begin(self, epoch, logs=None):
    """Called at the beginning of an epoch during training."""

  def on_epoch_end(self, epoch, logs=None):
    """Called at the end of an epoch during training."""
```

Training specific methods

```
class Callback(object):
  def __init__(self):
    self.validation_data = None
    self.model = None

  def on_epoch_begin(self, epoch, logs=None):
    """Called at the beginning of an epoch during training."""

  def on_epoch_end(self, epoch, logs=None):
    """Called at the end of an epoch during training."""
```

you can overwrite these two when you want to customize the function

Training specific methods

```
class Callback(object):
  def __init__(self):
    self.validation_data = None
    self.model = None

  def on_epoch_begin(self, epoch, logs=None):
    """Called at the beginning of an epoch during training."""

  def on_epoch_end(self, epoch, logs=None):
    """Called at the end of an epoch during training."""
```

Common methods for training/testing/predicting

```
class Callback(object):
  ...
  def on_(train|test|predict)_begin(self, logs=None):
    """Called at the begin of fit/evaluate/predict."""

  def on_(train|test|predict)_end(self, logs=None):
    """Called at the end of fit/evaluate/predict."""

  def on_(train|test|predict)_batch_begin(self, batch, logs=None):
    """Called right before processing a batch during training/testing/predicting."""

  def on_(train|test|predict)_batch_end(self, batch, logs=None):
    """Called at the end of training/testing/predicting a batch."""
```

← we can specify these callbacks at specific level during training and testing predicting

Common methods for training/testing/predicting

```python
class Callback(object):
  ...
  def on_(train|test|predict)_begin(self, logs=None):
    """Called at the begin of fit/evaluate/predict."""

  def on_(train|test|predict)_end(self, logs=None):
    """Called at the end of fit/evaluate/predict."""

  def on_(train|test|predict)_batch_begin(self, batch, logs=None):
    """Called right before processing a batch during training/testing/predicting."""

  def on_(train|test|predict)_batch_end(self, batch, logs=None):
    """Called at the end of training/testing/predicting a batch."""
```

Common methods for training/testing/predicting

```
class Callback(object):
  ...
  def on_(train|test|predict)_begin(self, logs=None):
    """Called at the begin of fit/evaluate/predict."""

  def on_(train|test|predict)_end(self, logs=None):
    """Called at the end of fit/evaluate/predict."""

  def on_(train|test|predict)_batch_begin(self, batch, logs=None):
    """Called right before processing a batch during training/testing/predicting."""

  def on_(train|test|predict)_batch_end(self, batch, logs=None):
    """Called at the end of training/testing/predicting a batch."""
```

Where can you use them?

Model methods that take callbacks

- `fit(..., callbacks=[...])`
- `fit_generator(..., callbacks=[...])`
- `evaluate(..., callbacks=[...])`
- `evaluate_generator(..., callbacks=[...])`
- `predict(..., callbacks=[...])`
- `predict_generator(..., callbacks=[...])`

TensorBoard Callback

- Visualize machine learning experiments
- Track metrics (e.g., loss, accuracy)
- View the model graph

can be used area

across these

```
TensorBoard(log_dir='./logs', update_freq='epoch', **kwargs)
```

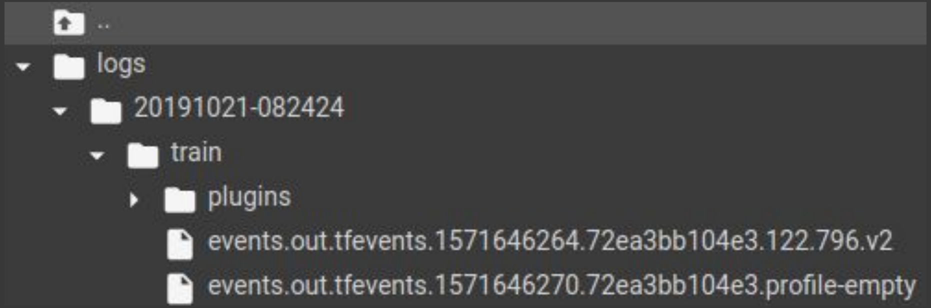
https://www.tensorflow.org/tensorboard

TensorBoard

Define the callback and start training

```
log_dir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
model.fit(train_batches, epochs=10, callbacks=[tensorboard])
```

it is initialized at the TensorBoard callback.

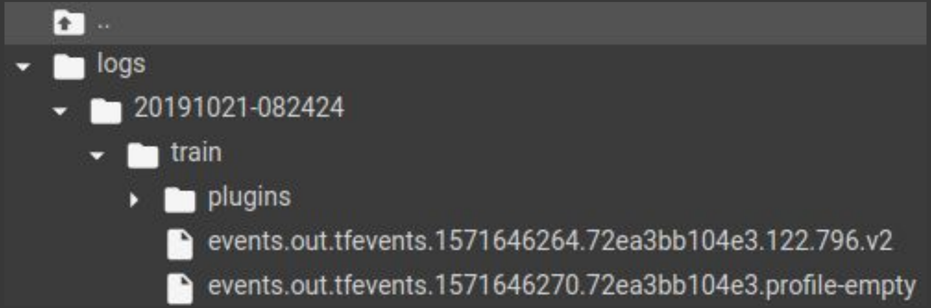


```
..
logs
  20191021-082424
    train
      plugins
      events.out.tfevents.1571646264.72ea3bb104e3.122.796.v2
      events.out.tfevents.1571646270.72ea3bb104e3.profile-empty
```

TensorBoard

Define the callback and start training

```
log_dir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
model.fit(train_batches, epochs=10, callbacks=[tensorboard])
```

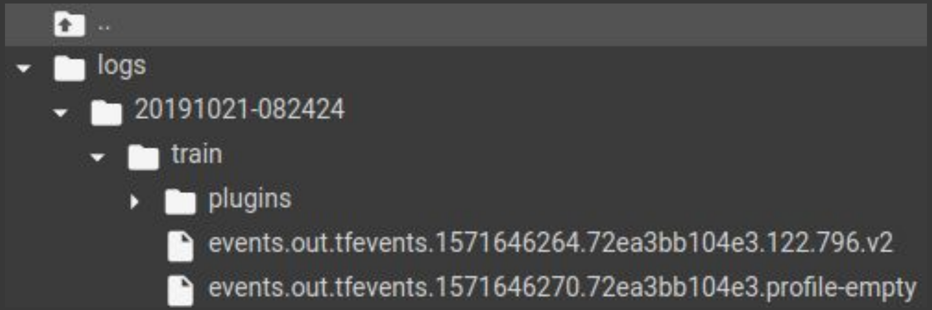


```
..
logs
  20191021-082424
    train
      plugins
      events.out.tfevents.1571646264.72ea3bb104e3.122.796.v2
      events.out.tfevents.1571646270.72ea3bb104e3.profile-empty
```

TensorBoard

Define the callback and start training

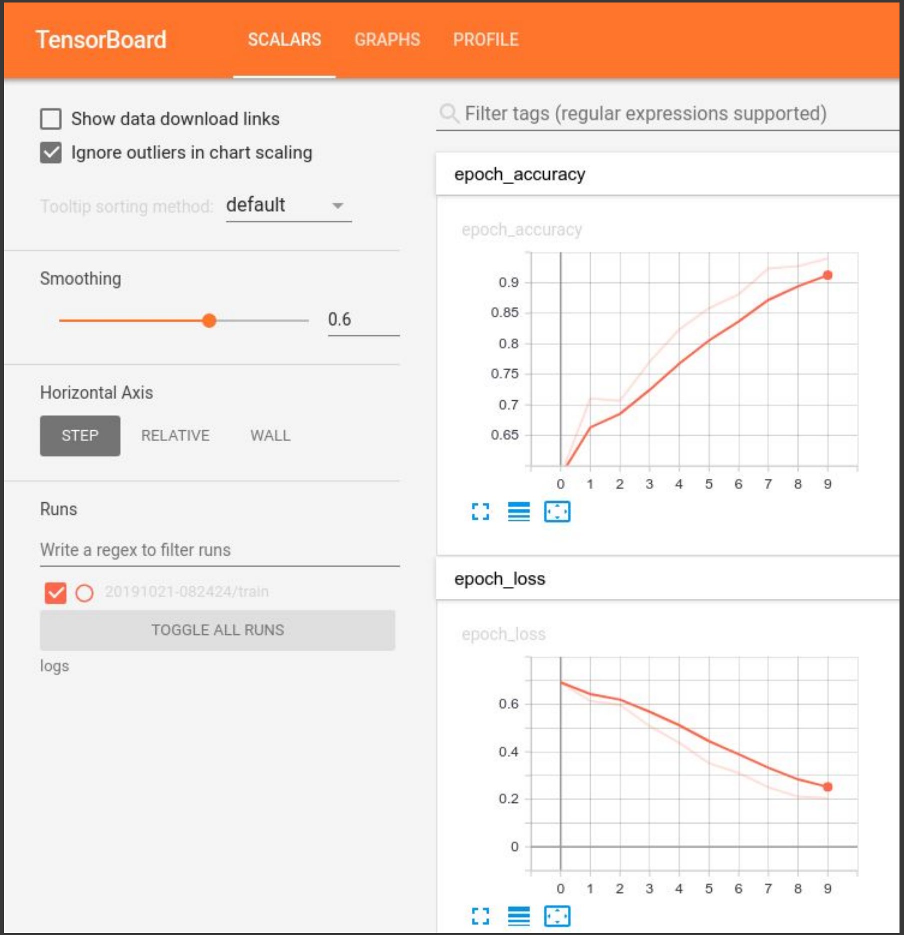
```
log_dir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
model.fit(train_batches, epochs=10, callbacks=[tensorboard])
```



```
..
▾ logs
  ▾ 20191021-082424
    ▾ train
      ▸ plugins
        events.out.tfevents.1571646264.72ea3bb104e3.122.796.v2
        events.out.tfevents.1571646270.72ea3bb104e3.profile-empty
```

TensorBoard in Colab

```
# Load the extension
%load_ext tensorboard

# Run TensorBoard
%tensorboard --logdir logs
```



□ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ 20191021-082424/train

logs

Filter tags (regular expressions supported)

Model Checkpoints

ModelCheckpoint

- Saves the model every so often

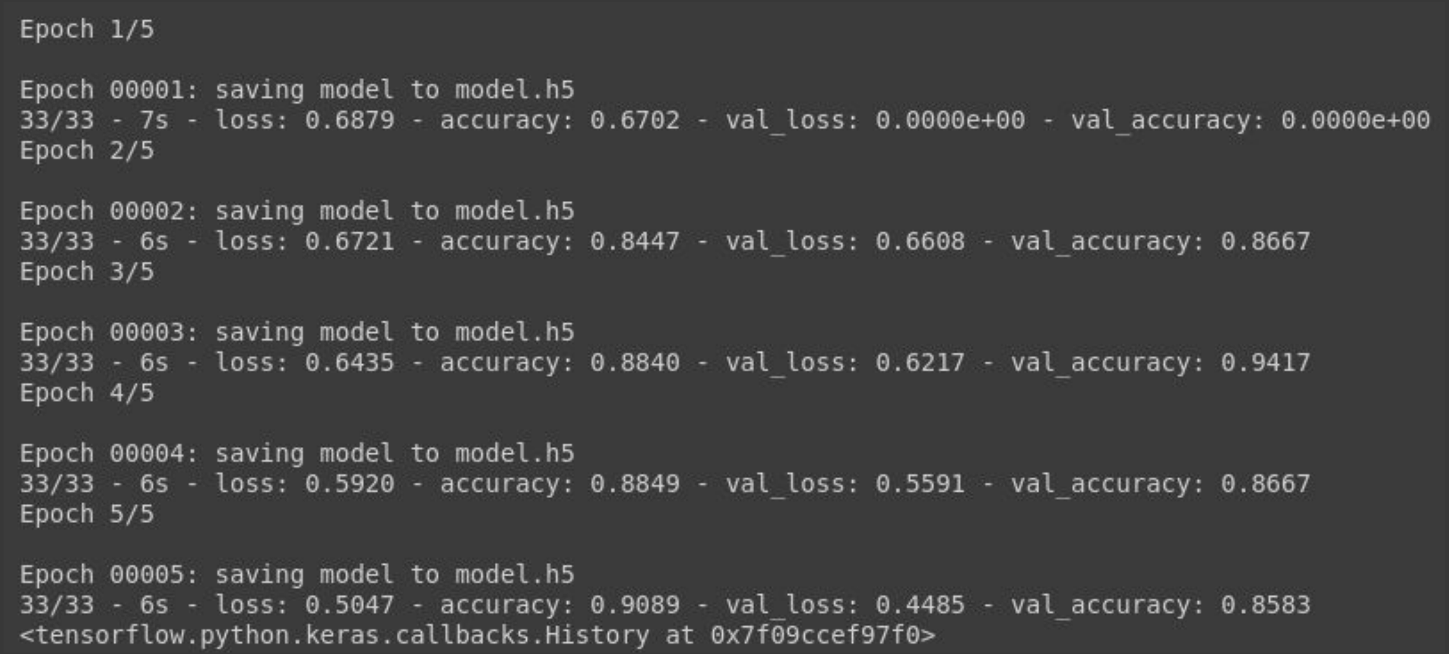
- Choose to save only the best checkpoints / weights

```
ModelCheckpoint(filepath, monitor='val_loss', mode='auto',
                save_best_only=False, save_weights_only=False,
                verbose=0, save_freq=1, **kwargs)
```

`ModelCheckpoint`

Saving model checkpoints

```
model.fit(train_batches, epochs=5, validation_data=validation_batches,
          callbacks=[ModelCheckpoint('model.h5', verbose=1)])
```

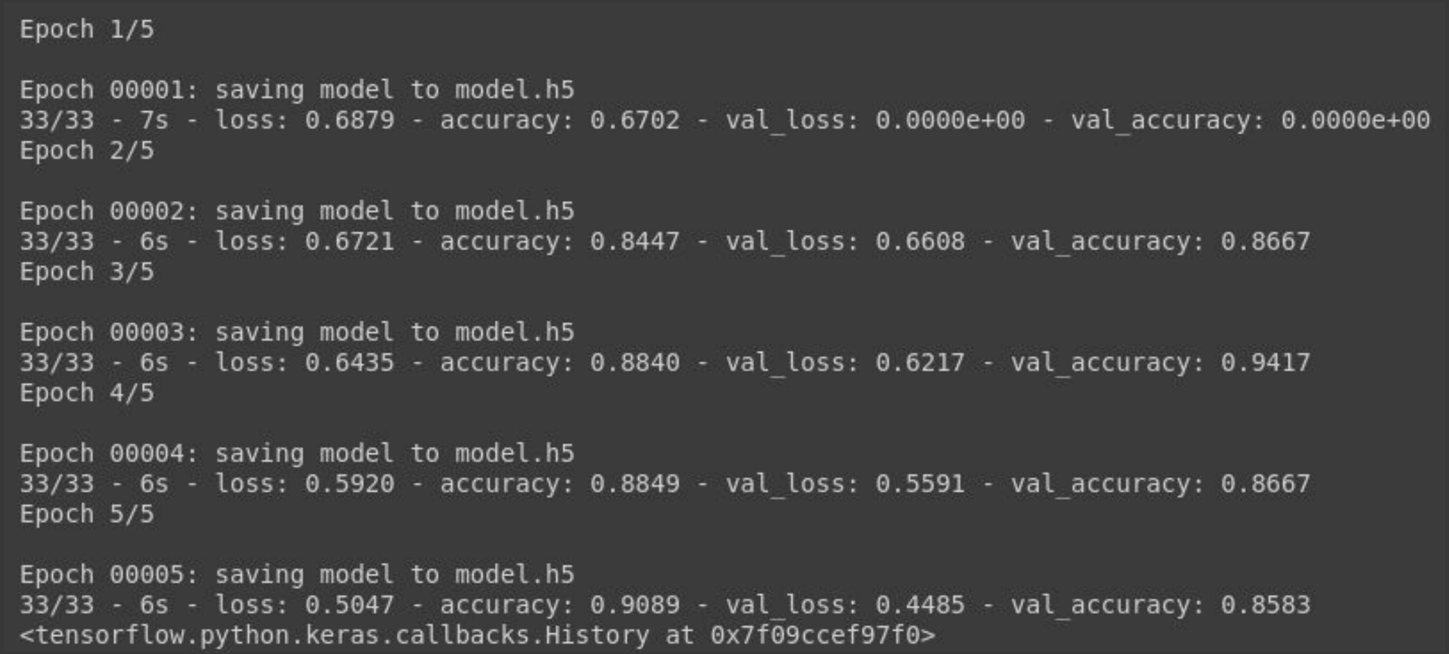


```
Epoch 1/5

Epoch 00001: saving model to model.h5
33/33 - 7s - loss: 0.6879 - accuracy: 0.6702 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: saving model to model.h5
33/33 - 6s - loss: 0.6721 - accuracy: 0.8447 - val_loss: 0.6608 - val_accuracy: 0.8667
Epoch 3/5

Epoch 00003: saving model to model.h5
33/33 - 6s - loss: 0.6435 - accuracy: 0.8840 - val_loss: 0.6217 - val_accuracy: 0.9417
Epoch 4/5

Epoch 00004: saving model to model.h5
33/33 - 6s - loss: 0.5920 - accuracy: 0.8849 - val_loss: 0.5591 - val_accuracy: 0.8667
Epoch 5/5

Epoch 00005: saving model to model.h5
33/33 - 6s - loss: 0.5047 - accuracy: 0.9089 - val_loss: 0.4485 - val_accuracy: 0.8583
<tensorflow.python.keras.callbacks.History at 0x7f09ccef97f0>
```

ModelCheckpoint

Saving model checkpoints

```
model.fit(train_batches, epochs=5, validation_data=validation_batches,
          callbacks=[ModelCheckpoint('model.h5', verbose=1)])
```

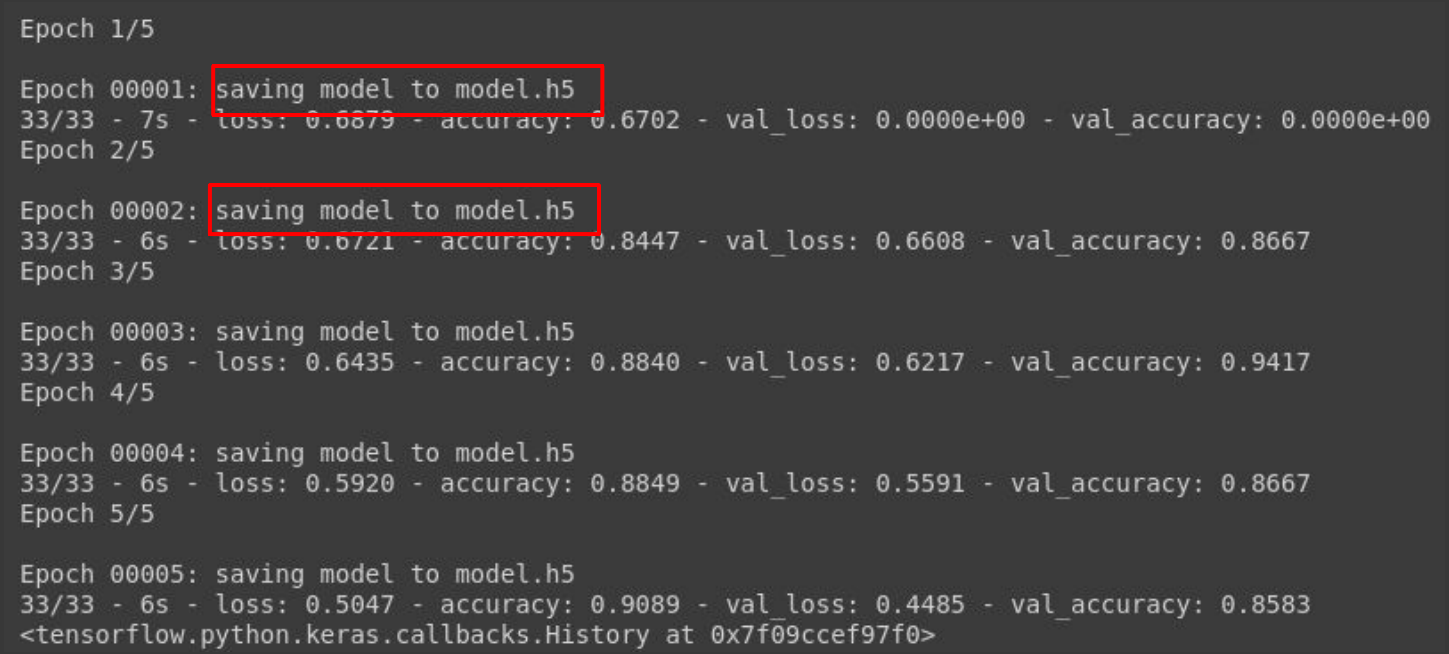


```
Epoch 1/5

Epoch 00001: saving model to model.h5
33/33 - 7s - loss: 0.6879 - accuracy: 0.6702 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: saving model to model.h5
33/33 - 6s - loss: 0.6721 - accuracy: 0.8447 - val_loss: 0.6608 - val_accuracy: 0.8667
Epoch 3/5

Epoch 00003: saving model to model.h5
33/33 - 6s - loss: 0.6435 - accuracy: 0.8840 - val_loss: 0.6217 - val_accuracy: 0.9417
Epoch 4/5

Epoch 00004: saving model to model.h5
33/33 - 6s - loss: 0.5920 - accuracy: 0.8849 - val_loss: 0.5591 - val_accuracy: 0.8667
Epoch 5/5

Epoch 00005: saving model to model.h5
33/33 - 6s - loss: 0.5047 - accuracy: 0.9089 - val_loss: 0.4485 - val_accuracy: 0.8583
<tensorflow.python.keras.callbacks.History at 0x7f09ccef97f0>
```

ModelCheckpoint

Saving model checkpoints

```
model.fit(train_batches, epochs=5, validation_data=validation_batches,
          callbacks=[ModelCheckpoint('model.h5', verbose=1)])
```

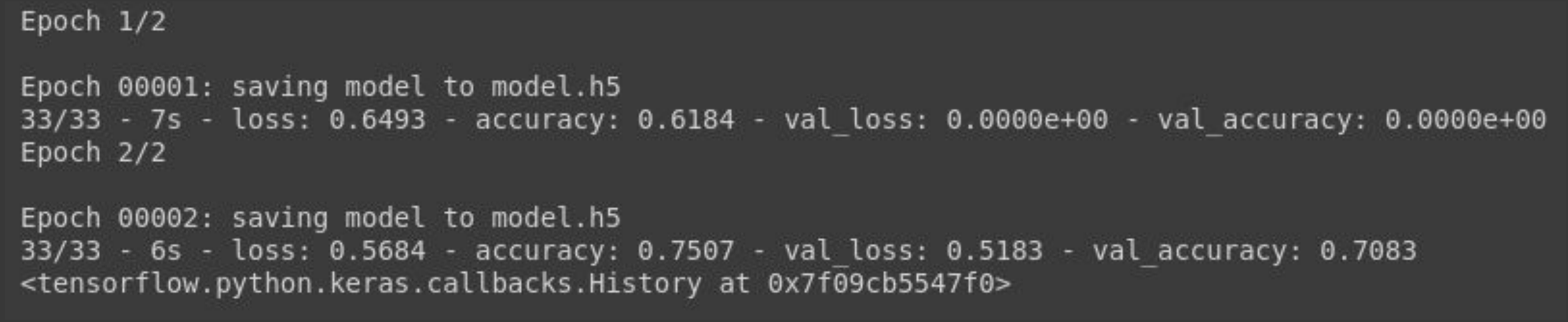


```
Epoch 1/5

Epoch 00001: saving model to model.h5
33/33 - 7s - loss: 0.6879 - accuracy: 0.6702 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: saving model to model.h5
33/33 - 6s - loss: 0.6721 - accuracy: 0.8447 - val_loss: 0.6608 - val_accuracy: 0.8667
Epoch 3/5

Epoch 00003: saving model to model.h5
33/33 - 6s - loss: 0.6435 - accuracy: 0.8840 - val_loss: 0.6217 - val_accuracy: 0.9417
Epoch 4/5

Epoch 00004: saving model to model.h5
33/33 - 6s - loss: 0.5920 - accuracy: 0.8849 - val_loss: 0.5591 - val_accuracy: 0.8667
Epoch 5/5

Epoch 00005: saving model to model.h5
33/33 - 6s - loss: 0.5047 - accuracy: 0.9089 - val_loss: 0.4485 - val_accuracy: 0.8583
<tensorflow.python.keras.callbacks.History at 0x7f09ccef97f0>
```

ModelCheckpoint

Save only the weights

```
model.fit(train_batches, epochs=5, validation_data=validation_batches, verbose=2,
          callbacks=[ModelCheckpoint('model.h5', save_weights_only=True, verbose=1)])
```

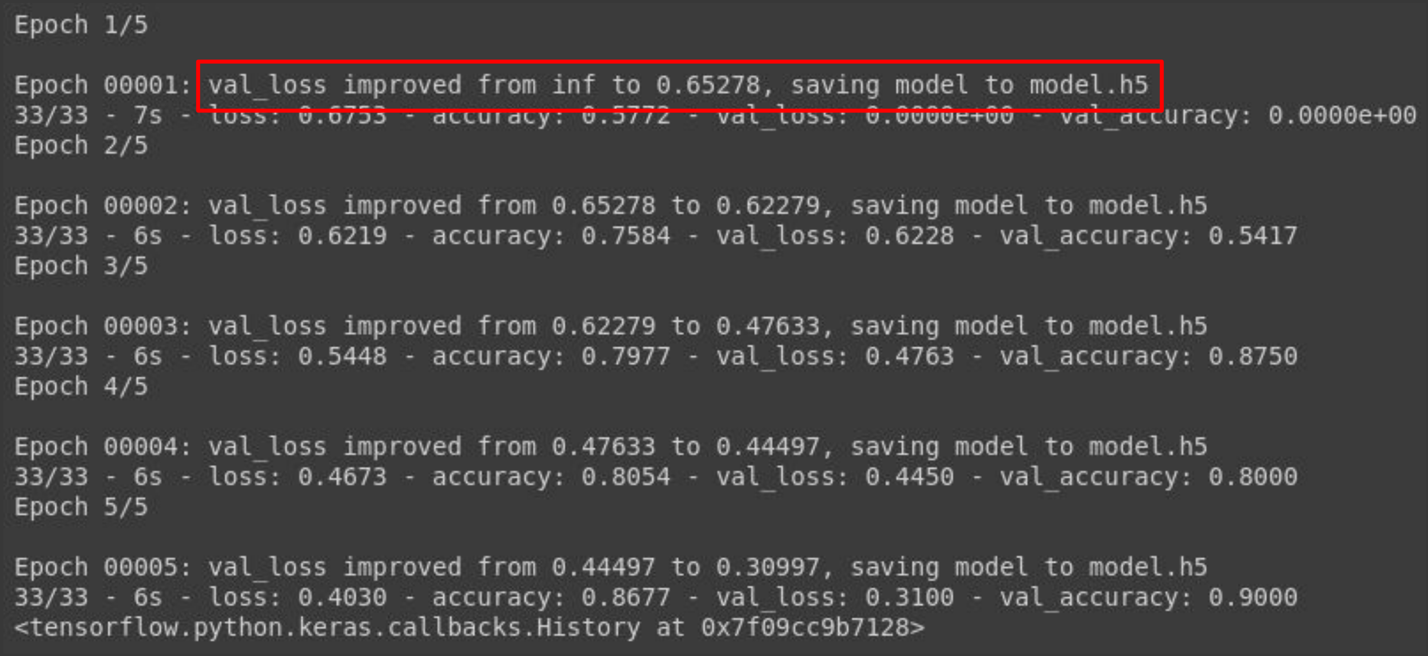


```
Epoch 1/2

Epoch 00001: saving model to model.h5
33/33 - 7s - loss: 0.6493 - accuracy: 0.6184 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/2

Epoch 00002: saving model to model.h5
33/33 - 6s - loss: 0.5684 - accuracy: 0.7507 - val_loss: 0.5183 - val_accuracy: 0.7083
<tensorflow.python.keras.callbacks.History at 0x7f09cb5547f0>
```

ModelCheckpoint

Save only the best checkpoints

```
model.fit(train_batches, epochs=5, validation_data=validation_batches, verbose=2,
          callbacks=[ModelCheckpoint('model.h5', monitor='val_loss',
                                     save_best_only=True, verbose=1)])
```

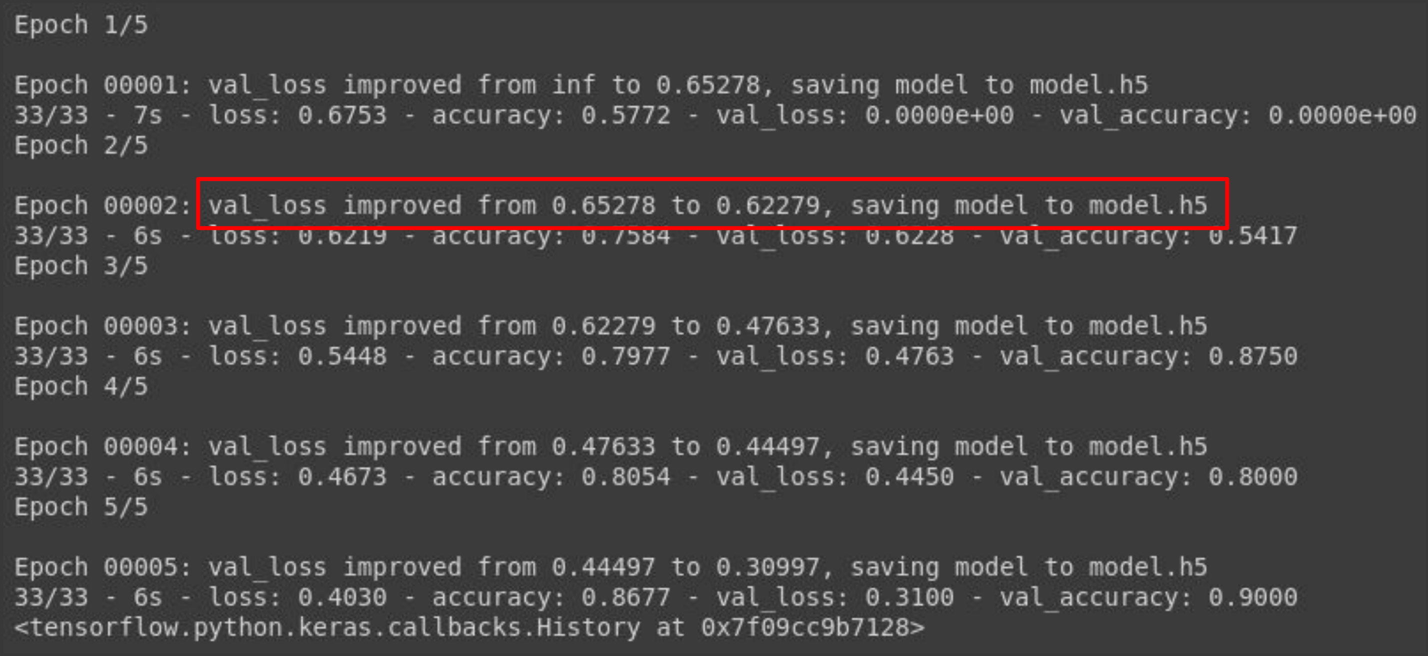


```
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.65278, saving model to model.h5
33/33 - 7s - loss: 0.6753 - accuracy: 0.5772 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: val_loss improved from 0.65278 to 0.62279, saving model to model.h5
33/33 - 6s - loss: 0.6219 - accuracy: 0.7584 - val_loss: 0.6228 - val_accuracy: 0.5417
Epoch 3/5

Epoch 00003: val_loss improved from 0.62279 to 0.47633, saving model to model.h5
33/33 - 6s - loss: 0.5448 - accuracy: 0.7977 - val_loss: 0.4763 - val_accuracy: 0.8750
Epoch 4/5

Epoch 00004: val_loss improved from 0.47633 to 0.44497, saving model to model.h5
33/33 - 6s - loss: 0.4673 - accuracy: 0.8054 - val_loss: 0.4450 - val_accuracy: 0.8000
Epoch 5/5

Epoch 00005: val_loss improved from 0.44497 to 0.30997, saving model to model.h5
33/33 - 6s - loss: 0.4030 - accuracy: 0.8677 - val_loss: 0.3100 - val_accuracy: 0.9000
<tensorflow.python.keras.callbacks.History at 0x7f09cc9b7128>
```

ModelCheckpoint

Save only the best checkpoints

```
model.fit(train_batches, epochs=5, validation_data=validation_batches, verbose=2,
          callbacks=[ModelCheckpoint('model.h5', monitor='val_loss',
                                     save_best_only=True, verbose=1)])
```

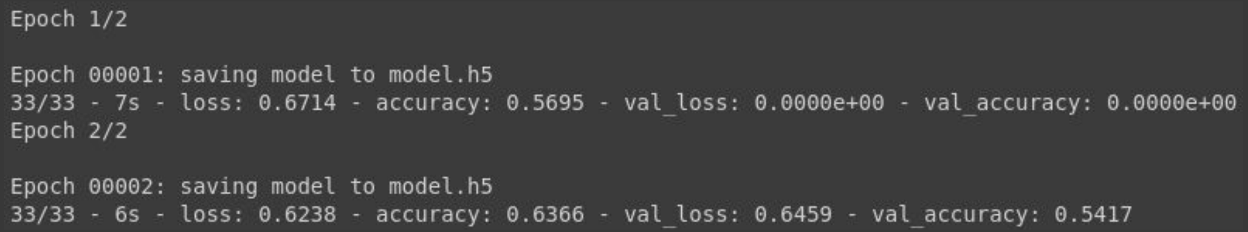


```
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.65278, saving model to model.h5
33/33 - 7s - loss: 0.6753 - accuracy: 0.5772 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: val_loss improved from 0.65278 to 0.62279, saving model to model.h5
33/33 - 6s - loss: 0.6219 - accuracy: 0.7584 - val_loss: 0.6228 - val_accuracy: 0.5417
Epoch 3/5

Epoch 00003: val_loss improved from 0.62279 to 0.47633, saving model to model.h5
33/33 - 6s - loss: 0.5448 - accuracy: 0.7977 - val_loss: 0.4763 - val_accuracy: 0.8750
Epoch 4/5

Epoch 00004: val_loss improved from 0.47633 to 0.44497, saving model to model.h5
33/33 - 6s - loss: 0.4673 - accuracy: 0.8054 - val_loss: 0.4450 - val_accuracy: 0.8000
Epoch 5/5

Epoch 00005: val_loss improved from 0.44497 to 0.30997, saving model to model.h5
33/33 - 6s - loss: 0.4030 - accuracy: 0.8677 - val_loss: 0.3100 - val_accuracy: 0.9000
<tensorflow.python.keras.callbacks.History at 0x7f09cc9b7128>
```

ModelCheckpoint

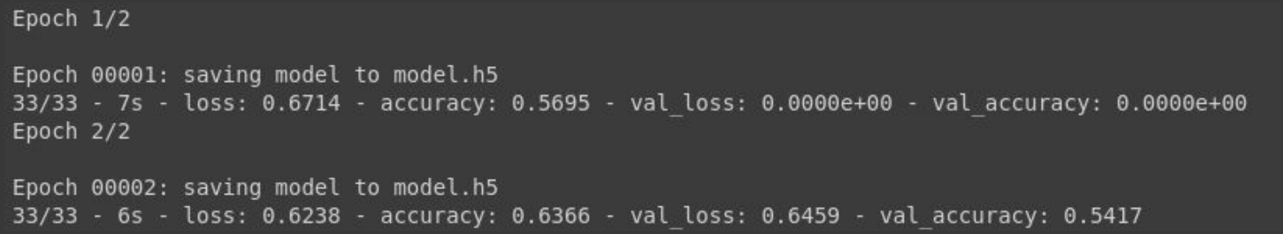
Choose your model format (SavedModel / H5)

```
model.fit(..., callbacks=[ModelCheckpoint('saved_model', ...)])
```

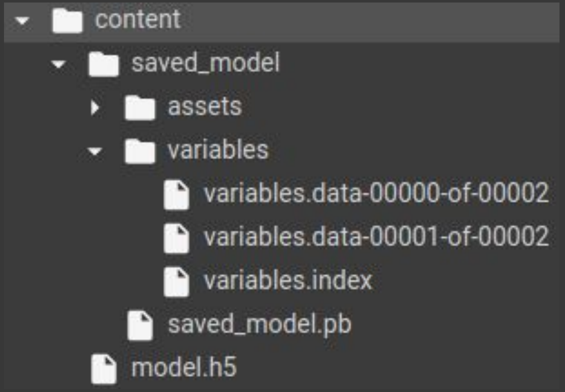


```
Epoch 1/2

Epoch 00001: saving model to model.h5
33/33 - 7s - loss: 0.6714 - accuracy: 0.5695 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/2

Epoch 00002: saving model to model.h5
33/33 - 6s - loss: 0.6238 - accuracy: 0.6366 - val_loss: 0.6459 - val_accuracy: 0.5417
```

```
model.fit(..., callbacks=[ModelCheckpoint('model.h5', ...)])
```



```
Epoch 1/2

Epoch 00001: saving model to model.h5
33/33 - 7s - loss: 0.6714 - accuracy: 0.5695 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/2

Epoch 00002: saving model to model.h5
33/33 - 6s - loss: 0.6238 - accuracy: 0.6366 - val_loss: 0.6459 - val_accuracy: 0.5417
```

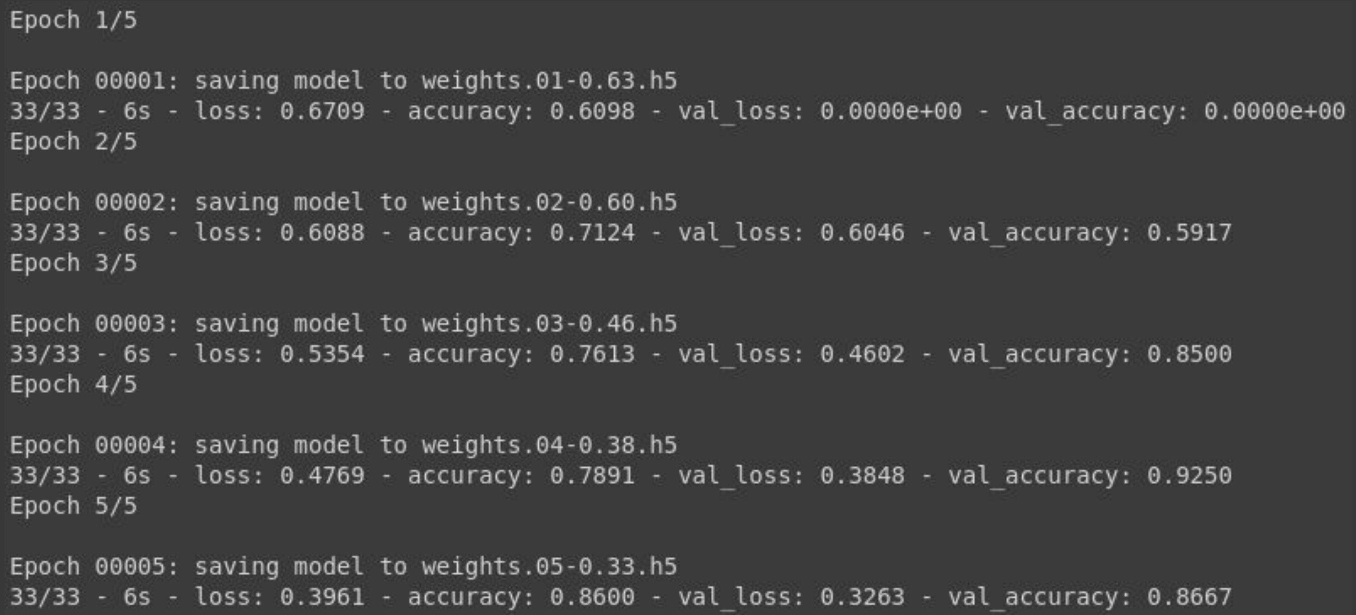


```
▾ content
    ▾ saved_model
        ▸ assets
        ▾ variables
            variables.data-00000-of-00002
            variables.data-00001-of-00002
            variables.index
          saved_model.pb
      model.h5
```

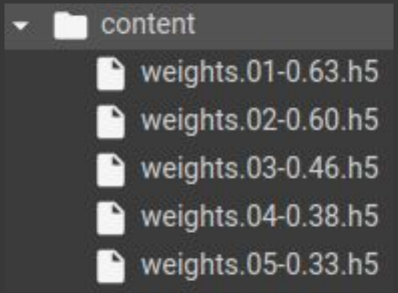
ModelCheckpoint

Track epoch #, losses, metrics

```
model.fit(..., callbacks=[ModelCheckpoint('weights.{epoch:02d}-{val_loss:.2f}.h5', verbose=1)])
```



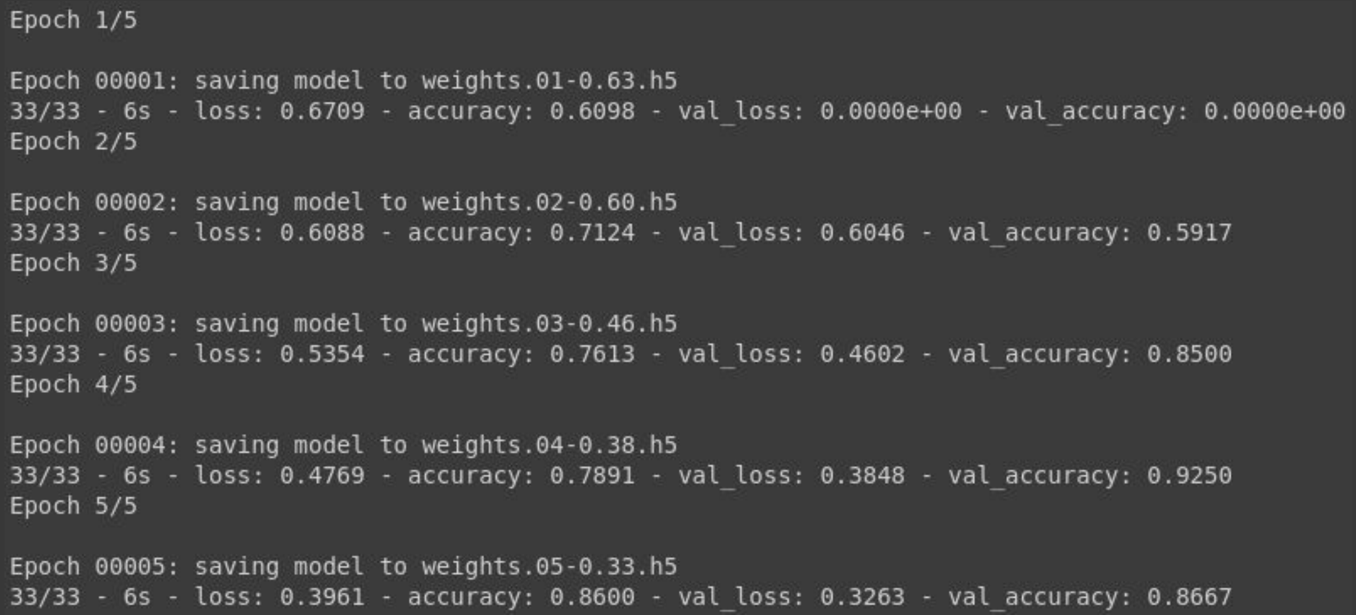
```
Epoch 1/5

Epoch 00001: saving model to weights.01-0.63.h5
33/33 - 6s - loss: 0.6709 - accuracy: 0.6098 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: saving model to weights.02-0.60.h5
33/33 - 6s - loss: 0.6088 - accuracy: 0.7124 - val_loss: 0.6046 - val_accuracy: 0.5917
Epoch 3/5

Epoch 00003: saving model to weights.03-0.46.h5
33/33 - 6s - loss: 0.5354 - accuracy: 0.7613 - val_loss: 0.4602 - val_accuracy: 0.8500
Epoch 4/5

Epoch 00004: saving model to weights.04-0.38.h5
33/33 - 6s - loss: 0.4769 - accuracy: 0.7891 - val_loss: 0.3848 - val_accuracy: 0.9250
Epoch 5/5

Epoch 00005: saving model to weights.05-0.33.h5
33/33 - 6s - loss: 0.3961 - accuracy: 0.8600 - val_loss: 0.3263 - val_accuracy: 0.8667
```



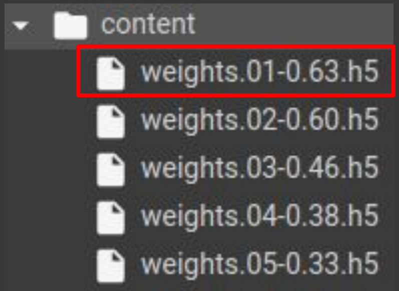
ModelCheckpoint

Track epoch #, losses, metrics

```
model.fit(..., callbacks=[ModelCheckpoint('weights.{epoch:02d}-{val_loss:.2f}.h5', verbose=1)])
```



```
Epoch 1/5

Epoch 00001: saving model to weights.01-0.63.h5
33/33 - 6s - loss: 0.6709 - accuracy: 0.6098 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5

Epoch 00002: saving model to weights.02-0.60.h5
33/33 - 6s - loss: 0.6088 - accuracy: 0.7124 - val_loss: 0.6046 - val_accuracy: 0.5917
Epoch 3/5

Epoch 00003: saving model to weights.03-0.46.h5
33/33 - 6s - loss: 0.5354 - accuracy: 0.7613 - val_loss: 0.4602 - val_accuracy: 0.8500
Epoch 4/5

Epoch 00004: saving model to weights.04-0.38.h5
33/33 - 6s - loss: 0.4769 - accuracy: 0.7891 - val_loss: 0.3848 - val_accuracy: 0.9250
Epoch 5/5

Epoch 00005: saving model to weights.05-0.33.h5
33/33 - 6s - loss: 0.3961 - accuracy: 0.8600 - val_loss: 0.3263 - val_accuracy: 0.8667
```



```
content
    weights.01-0.63.h5
    weights.02-0.60.h5
    weights.03-0.46.h5
    weights.04-0.38.h5
    weights.05-0.33.h5
```

EarlyStopping

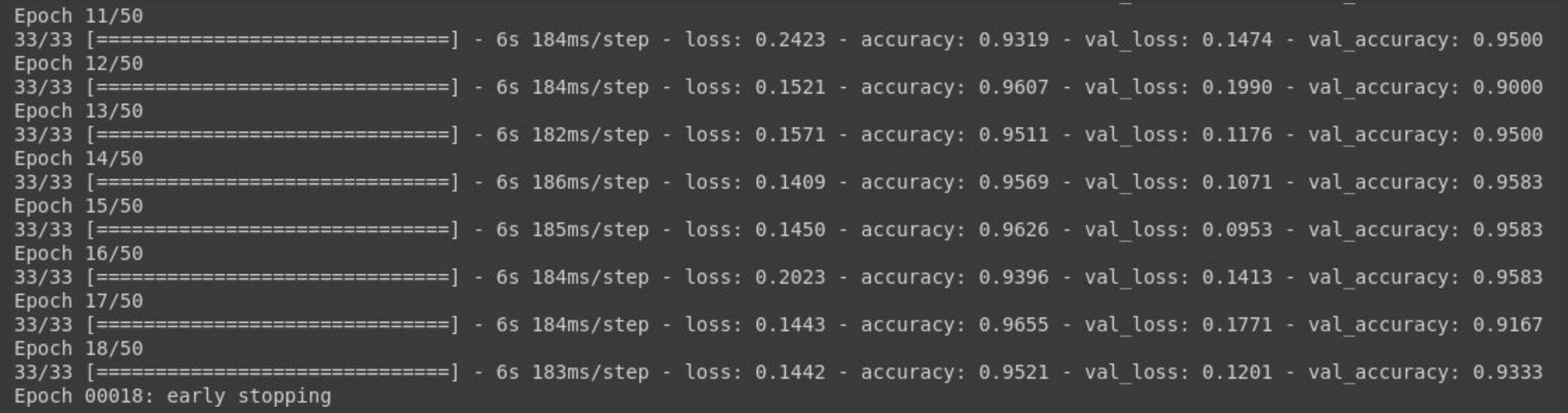
- Helps you keep track of a certain metric/loss and change training behavior accordingly
- Stops training when there’s no improvement observed

```
EarlyStopping(monitor='val_loss', verbose=0,
              min_delta=0, patience=0, mode='auto',
              baseline=None, restore_best_weights=False,
              **kwargs)
```

EarlyStopping

Prevent your model from overfitting

```
model.fit(train_batches, epochs=50, validation_data=validation_batches,
          callbacks=[EarlyStopping(patience=3, monitor='val_loss')])
```

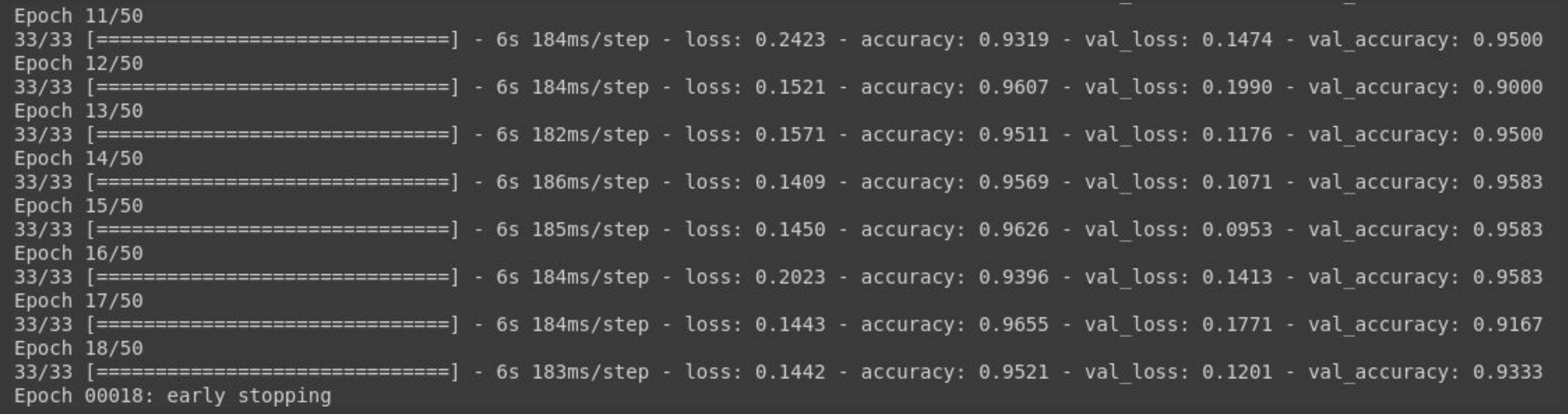


```
Epoch 11/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2423 - accuracy: 0.9319 - val_loss: 0.1474 - val_accuracy: 0.9500
Epoch 12/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1521 - accuracy: 0.9607 - val_loss: 0.1990 - val_accuracy: 0.9000
Epoch 13/50
33/33 [==============================] - 6s 182ms/step - loss: 0.1571 - accuracy: 0.9511 - val_loss: 0.1176 - val_accuracy: 0.9500
Epoch 14/50
33/33 [==============================] - 6s 186ms/step - loss: 0.1409 - accuracy: 0.9569 - val_loss: 0.1071 - val_accuracy: 0.9583
Epoch 15/50
33/33 [==============================] - 6s 185ms/step - loss: 0.1450 - accuracy: 0.9626 - val_loss: 0.0953 - val_accuracy: 0.9583
Epoch 16/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2023 - accuracy: 0.9396 - val_loss: 0.1413 - val_accuracy: 0.9583
Epoch 17/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1443 - accuracy: 0.9655 - val_loss: 0.1771 - val_accuracy: 0.9167
Epoch 18/50
33/33 [==============================] - 6s 183ms/step - loss: 0.1442 - accuracy: 0.9521 - val_loss: 0.1201 - val_accuracy: 0.9333
Epoch 00018: early stopping
```

EarlyStopping

Prevent your model from overfitting

```
model.fit(train_batches, epochs=50, validation_data=validation_batches,
          callbacks=[EarlyStopping(patience=3, monitor='val_loss')])
```

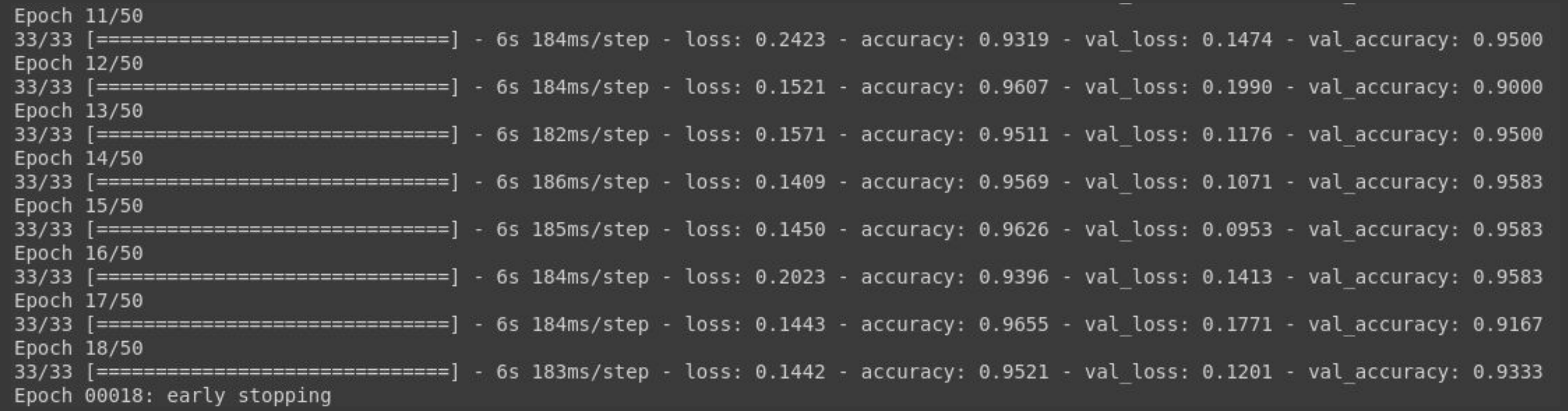


```
Epoch 11/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2423 - accuracy: 0.9319 - val_loss: 0.1474 - val_accuracy: 0.9500
Epoch 12/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1521 - accuracy: 0.9607 - val_loss: 0.1990 - val_accuracy: 0.9000
Epoch 13/50
33/33 [==============================] - 6s 182ms/step - loss: 0.1571 - accuracy: 0.9511 - val_loss: 0.1176 - val_accuracy: 0.9500
Epoch 14/50
33/33 [==============================] - 6s 186ms/step - loss: 0.1409 - accuracy: 0.9569 - val_loss: 0.1071 - val_accuracy: 0.9583
Epoch 15/50
33/33 [==============================] - 6s 185ms/step - loss: 0.1450 - accuracy: 0.9626 - val_loss: 0.0953 - val_accuracy: 0.9583
Epoch 16/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2023 - accuracy: 0.9396 - val_loss: 0.1413 - val_accuracy: 0.9583
Epoch 17/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1443 - accuracy: 0.9655 - val_loss: 0.1771 - val_accuracy: 0.9167
Epoch 18/50
33/33 [==============================] - 6s 183ms/step - loss: 0.1442 - accuracy: 0.9521 - val_loss: 0.1201 - val_accuracy: 0.9333
Epoch 00018: early stopping
```

EarlyStopping

Prevent your model from overfitting

```
model.fit(train_batches, epochs=50, validation_data=validation_batches,
          callbacks=[EarlyStopping(patience=3, monitor='val_loss')])
```

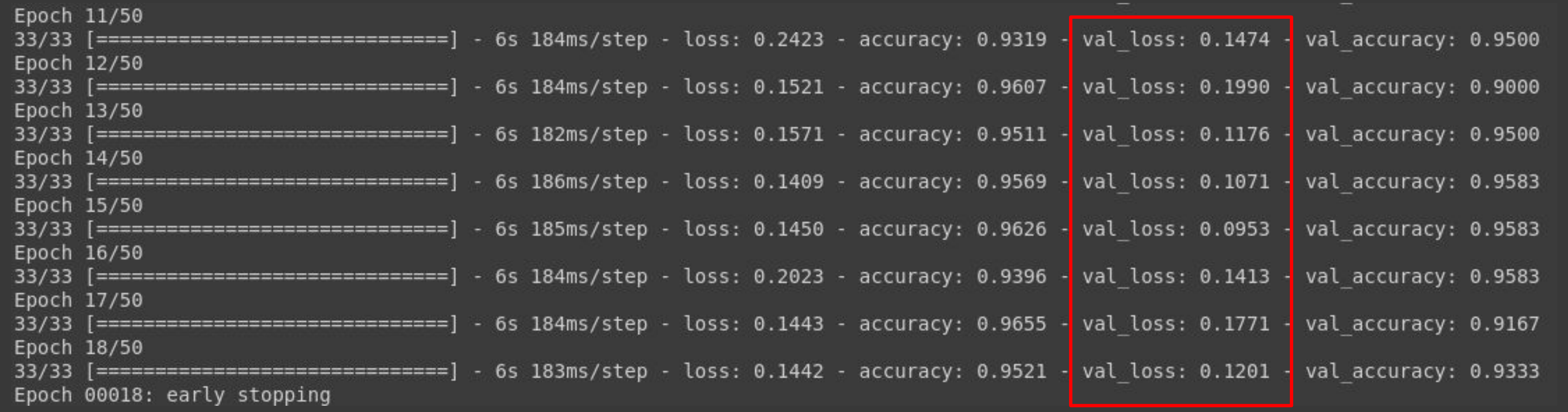


```
Epoch 11/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2423 - accuracy: 0.9319 - val_loss: 0.1474 - val_accuracy: 0.9500
Epoch 12/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1521 - accuracy: 0.9607 - val_loss: 0.1990 - val_accuracy: 0.9000
Epoch 13/50
33/33 [==============================] - 6s 182ms/step - loss: 0.1571 - accuracy: 0.9511 - val_loss: 0.1176 - val_accuracy: 0.9500
Epoch 14/50
33/33 [==============================] - 6s 186ms/step - loss: 0.1409 - accuracy: 0.9569 - val_loss: 0.1071 - val_accuracy: 0.9583
Epoch 15/50
33/33 [==============================] - 6s 185ms/step - loss: 0.1450 - accuracy: 0.9626 - val_loss: 0.0953 - val_accuracy: 0.9583
Epoch 16/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2023 - accuracy: 0.9396 - val_loss: 0.1413 - val_accuracy: 0.9583
Epoch 17/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1443 - accuracy: 0.9655 - val_loss: 0.1771 - val_accuracy: 0.9167
Epoch 18/50
33/33 [==============================] - 6s 183ms/step - loss: 0.1442 - accuracy: 0.9521 - val_loss: 0.1201 - val_accuracy: 0.9333
Epoch 00018: early stopping
```

EarlyStopping

Prevent your model from overfitting

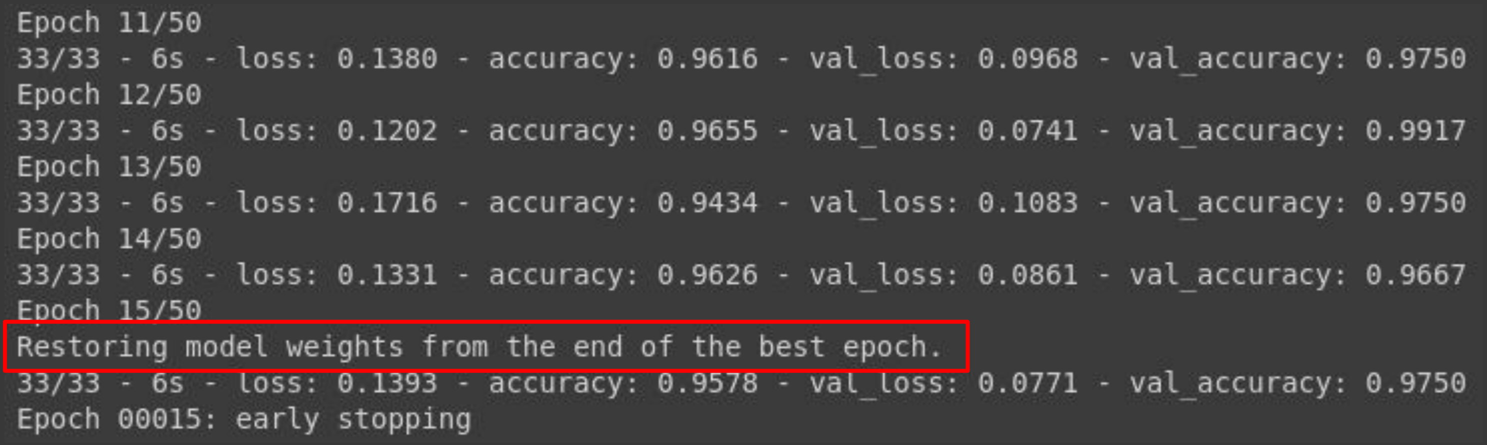
```
model.fit(train_batches, epochs=50, validation_data=validation_batches,
          callbacks=[EarlyStopping(patience=3, monitor='val_loss')])
```



```
Epoch 11/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2423 - accuracy: 0.9319 - val_loss: 0.1474 - val_accuracy: 0.9500
Epoch 12/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1521 - accuracy: 0.9607 - val_loss: 0.1990 - val_accuracy: 0.9000
Epoch 13/50
33/33 [==============================] - 6s 182ms/step - loss: 0.1571 - accuracy: 0.9511 - val_loss: 0.1176 - val_accuracy: 0.9500
Epoch 14/50
33/33 [==============================] - 6s 186ms/step - loss: 0.1409 - accuracy: 0.9569 - val_loss: 0.1071 - val_accuracy: 0.9583
Epoch 15/50
33/33 [==============================] - 6s 185ms/step - loss: 0.1450 - accuracy: 0.9626 - val_loss: 0.0953 - val_accuracy: 0.9583
Epoch 16/50
33/33 [==============================] - 6s 184ms/step - loss: 0.2023 - accuracy: 0.9396 - val_loss: 0.1413 - val_accuracy: 0.9583
Epoch 17/50
33/33 [==============================] - 6s 184ms/step - loss: 0.1443 - accuracy: 0.9655 - val_loss: 0.1771 - val_accuracy: 0.9167
Epoch 18/50
33/33 [==============================] - 6s 183ms/step - loss: 0.1442 - accuracy: 0.9521 - val_loss: 0.1201 - val_accuracy: 0.9333
Epoch 00018: early stopping
```

Restoring best weights

```
model.fit(...,
          callbacks=[EarlyStopping(patience=3, restore_best_weights=True,
                                   monitor='val_loss', verbose=1)])
```



```
Epoch 11/50
33/33 - 6s - loss: 0.1380 - accuracy: 0.9616 - val_loss: 0.0968 - val_accuracy: 0.9750
Epoch 12/50
33/33 - 6s - loss: 0.1202 - accuracy: 0.9655 - val_loss: 0.0741 - val_accuracy: 0.9917
Epoch 13/50
33/33 - 6s - loss: 0.1716 - accuracy: 0.9434 - val_loss: 0.1083 - val_accuracy: 0.9750
Epoch 14/50
33/33 - 6s - loss: 0.1331 - accuracy: 0.9626 - val_loss: 0.0861 - val_accuracy: 0.9667
Epoch 15/50
Restoring model weights from the end of the best epoch.
33/33 - 6s - loss: 0.1393 - accuracy: 0.9578 - val_loss: 0.0771 - val_accuracy: 0.9750
Epoch 00015: early stopping
```

EarlyStopping

More customization

```
model.fit(...,
          callbacks=[EarlyStopping(
                             patience=3,
                             min_delta=0.05,
                             baseline=0.8,
                             mode='min',
                             monitor='val_loss',
                             verbose=1
                     )])
```

CSVLogger

Logging training results

```
model.fit(..., callbacks=[CSVLogger('training.csv')])
```

epoch	accuracy	loss	val_accuracy	val_loss
0	0.574305	0.682536	0.775000	0.655427
1	0.760307	0.633610	0.675000	0.595201
2	0.758389	0.573186	0.850000	0.503174
3	0.835091	0.472031	0.808333	0.416691
4	0.854267	0.419491	0.916667	0.309128

Multiple callbacks

```
model.fit(..., callbacks=[EarlyStopping(...),
model.evaluate(...        ModelCheckpoint(...),
model.predict(...         TensorBoard(...),
                          ...
                         ])
```

Build a simple model

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

model.add(tf.keras.layers.Dense(units=1,
                                activation='linear',
                                input_dim=(784,)))

model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.1),
              loss='mean_squared_error', metrics=['mae'])
```

How a custom callback looks

```
import datetime

class MyCustomCallback(tf.keras.callbacks.Callback):
  def on_train_batch_begin(self, batch, logs=None):
    print('Training: batch {} begins at {}'
          .format(batch, datetime.datetime.now().time()))

  def on_train_batch_end(self, batch, logs=None):
    print('Training: batch {} ends at {}'
          .format(batch, datetime.datetime.now().time()))
```

How a custom callback looks

```
import datetime

class MyCustomCallback(tf.keras.callbacks.Callback):
  def on_train_batch_begin(self, batch, logs=None):
    print('Training: batch {} begins at {}'
          .format(batch, datetime.datetime.now().time()))

  def on_train_batch_end(self, batch, logs=None):
    print('Training: batch {} ends at {}'
          .format(batch, datetime.datetime.now().time()))
```

```
my_custom_callback = MyCustomCallback()

model.fit(x_train, y_train, batch_size=64, epochs=1, verbose=0,
          callbacks=[my_custom_callback])
```

```
my_custom_callback = MyCustomCallback()

model.fit(x_train, y_train, batch_size=64, epochs=1, verbose=0,
          callbacks=[my_custom_callback])
```

```
class DetectOverfittingCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold):
    super(DetectOverfittingCallback, self).__init__()
    self.threshold = threshold

  def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))

    if ratio>threshold:
      print("Stopping training...")
      self.model.stop_training = True

model.fit(..., callbacks=[DetectOverfittingCallback(threshold=1.3)])
```

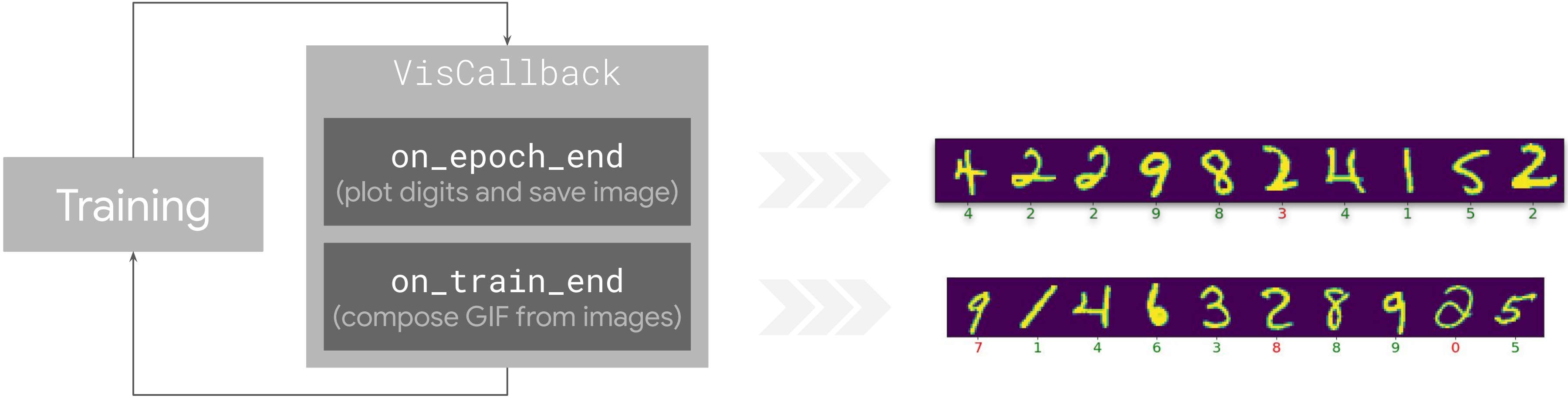
```
class DetectOverfittingCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold):
    super(DetectOverfittingCallback, self).__init__()
    self.threshold = threshold

  def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))

    if ratio>threshold:
      print("Stopping training...")
      self.model.stop_training = True

model.fit(..., callbacks=[DetectOverfittingCallback(threshold=1.3)])
```

```python
class DetectOverfittingCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold):
    super(DetectOverfittingCallback, self).__init__()
    self.threshold = threshold

  def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))

    if ratio>threshold:
      print("Stopping training...")
      self.model.stop_training = True

model.fit(..., callbacks=[DetectOverfittingCallback(threshold=1.3)])
```

```python
class DetectOverfittingCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold):
    super(DetectOverfittingCallback, self).__init__()
    self.threshold = threshold

  def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))

    if ratio>threshold:
      print("Stopping training...")
      self.model.stop_training = True

model.fit(..., callbacks=[DetectOverfittingCallback(threshold=1.3)])
```

```
class DetectOverfittingCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold):
    super(DetectOverfittingCallback, self).__init__()
    self.threshold = threshold

  def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))

    if ratio>threshold:
      print("Stopping training...")
      self.model.stop_training = True

model.fit(..., callbacks=[DetectOverfittingCallback(threshold=1.3)])
```

```
class DetectOverfittingCallback(tf.keras.callbacks.Callback):
  def __init__(self, threshold):
    super(DetectOverfittingCallback, self).__init__()
    self.threshold = threshold

  def on_epoch_end(self, epoch, logs=None):
    ratio = logs["val_loss"] / logs["loss"]
    print("Epoch: {}, Val/Train loss ratio: {:.2f}".format(epoch, ratio))

    if ratio>threshold:
      print("Stopping training...")
      self.model.stop_training = True

model.fit(..., callbacks=[DetectOverfittingCallback(threshold=1.3)])
```



Training

VisCallback

on_epoch_end
(plot digits and save image)

on_train_end
(compose GIF from images)

```python
class VisCallback(tf.keras.callbacks.Callback):
  def __init__(self, inputs, ground_truth, display_freq=10,
                                           n_samples=10):
    self.inputs = inputs
    self.ground_truth = ground_truth
    self.images = []
    self.display_freq = display_freq
    self.n_samples = n_samples
```

```
class VisCallback(tf.keras.callbacks.Callback):
  def __init__(self, inputs, ground_truth, display_freq=10,
                                           n_samples=10):
    self.inputs = inputs
    self.ground_truth = ground_truth
    self.images = []
    self.display_freq = display_freq
    self.n_samples = n_samples
```

```
class VisCallback(tf.keras.callbacks.Callback):
  def __init__(self, inputs, ground_truth, display_freq=10,
                                           n_samples=10):
    self.inputs = inputs
    self.ground_truth = ground_truth
    self.images = []
    self.display_freq = display_freq
    self.n_samples = n_samples
```

```python
class VisCallback(tf.keras.callbacks.Callback):
  def __init__(self, inputs, ground_truth, display_freq=10,
                                           n_samples=10):
    self.inputs = inputs
    self.ground_truth = ground_truth
    self.images = []
    self.display_freq = display_freq
    self.n_samples = n_samples
```

```python
class VisCallback(tf.keras.callbacks.Callback):
  def __init__(self, inputs, ground_truth, display_freq=10,
                                           n_samples=10):
    self.inputs = inputs
    self.ground_truth = ground_truth
    self.images = []
    self.display_freq = display_freq
    self.n_samples = n_samples
```

```
class VisCallback(tf.keras.callbacks.Callback):
  def __init__(self, inputs, ground_truth, display_freq=10,
                                           n_samples=10):
    self.inputs = inputs
    self.ground_truth = ground_truth
    self.images = []
    self.display_freq = display_freq
    self.n_samples = n_samples
```

```
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    # Randomly sample data
    indexes = np.random.choice(len(self.inputs), size=self.n_samples)
    X_test, y_test = self.inputs[indexes], self.ground_truth[indexes]
    predictions = np.argmax(self.model.predict(X_test), axis=1)
```

```
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    # Randomly sample data
    indexes = np.random.choice(len(self.inputs), size=self.n_samples)
    X_test, y_test = self.inputs[indexes], self.ground_truth[indexes]
    predictions = np.argmax(self.model.predict(X_test), axis=1)
```

```
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    # Randomly sample data
    indexes = np.random.choice(len(self.inputs), size=self.n_samples)
    X_test, y_test = self.inputs[indexes], self.ground_truth[indexes]
    predictions = np.argmax(self.model.predict(X_test), axis=1)
```

```python
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    # Randomly sample data
    indexes = np.random.choice(len(self.inputs), size=self.n_samples)
    X_test, y_test = self.inputs[indexes], self.ground_truth[indexes]
    predictions = np.argmax(self.model.predict(X_test), axis=1)
```

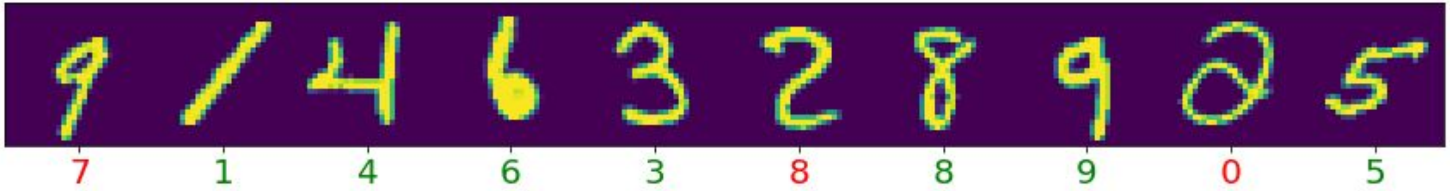
```python
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    # Randomly sample data
    indexes = np.random.choice(len(self.inputs), size=self.n_samples)
    X_test, y_test = self.inputs[indexes], self.ground_truth[indexes]
    predictions = np.argmax(self.model.predict(X_test), axis=1)

    # Plot the digits
    display_digits(X_test, predictions, y_test, epoch, n=self.display_freq)
    ...
```

```python
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    ...
    # Save the figure
    buf = io.BytesIO()
    plt.savefig(buf, format='png')
    buf.seek(0)
    image = Image.open(buf)
    self.images.append(np.array(image))

    # Display the digits every now and then
    if epoch % self.display_freq == 0:
      plt.show()
```

```python
class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_epoch_end(self, epoch, logs=None):
    ...
    # Save the figure
    buf = io.BytesIO()
    plt.savefig(buf, format='png')
    buf.seek(0)
    image = Image.open(buf)
    self.images.append(np.array(image))

    # Display the digits every now and then
    if epoch % self.display_freq == 0:
      plt.show()
```

```
import imageio

class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_train_end(self, logs=None):
    imageio.mimsave('animation.gif', self.images, fps=1)

# Train the model
model.fit(..., callbacks=[VisCallback(x_test, y_test)])
```

```
import imageio

class VisCallback(tf.keras.callbacks.Callback):
  ...
  def on_train_end(self, logs=None):
    imageio.mimsave('animation.gif', self.images, fps=1)

# Train the model
model.fit(..., callbacks=[VisCallback(x_test, y_test)])
```



7
1
4
6
3
8
8
9
0
5