```
import os
import re
import shutil
import string
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import losses
from tensorflow.keras import preprocessing


data_dir = '/content/drive/MyDrive/NEWS'

train_dir = os.path.join(data_dir, 'train')

batch_size = 32
seed = 42


raw_train_dataset = tf.keras.preprocessing.text_dataset_from_directory('/content/drive/MyDri
                                                                       batch_size = batch_si
                                                                                  seed=seed)


raw_test_dataset = tf.keras.preprocessing.text_dataset_from_directory('/content/drive/MyDriv
                                                                      batch_size=batch_size)
```

```
Found 88 files belonging to 2 classes.
Found 25 files belonging to 2 classes.
```

```
max_features = 10000
sequence_length = 250

vectorize_layer = layers.TextVectorization(max_tokens = max_features,
                                           output_mode = 'int',
                                           output_sequence_length = sequence_length)

train_text = raw_train_dataset.map(lambda x, y: x)
vectorize_layer.adapt(train_text) ## Calling adapt to create a vocabulary and frequency from
```

```
WARNING:tensorflow:5 out of the last 13 calls to <function PreprocessingLayer.make_adapt_fun
```

```
def vectorize_text(text, label):
  text = tf.expand_dims(text, -1)
  return vectorize_layer(text), label

train_data = raw_train_dataset.map(vectorize_text)
test_data = raw_test_dataset.map(vectorize_text)
```

```
AUTOTUNE = tf.data.AUTOTUNE         ## OPTIONAL, we are prefetching the next data while we a
                                    ## Autotune will set the number of elements to prefetch
train_data = train_data.cache().prefetch(buffer_size=AUTOTUNE)
test_data = test_data.cache().prefetch(buffer_size=AUTOTUNE)
```

```
embedding_dim = 16 ##Embedding layer dimsensions for vocabulary

## Customize neural network for complexity
model = tf.keras.Sequential([
  layers.Embedding(max_features + 1, embedding_dim),
  layers.Dropout(0.2),
  layers.Dense(5),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(5),
  layers.Dense(1)]) ## Has to stay as one due to binary classification. Only need one neuron

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 16)          160016

_________________________________________________________________
dropout_8 (Dropout)          (None, None, 16)          0

_________________________________________________________________
dense_7 (Dense)              (None, None, 5)           85

_________________________________________________________________
global_average_pooling1d_4 ( (None, 5)                 0

_________________________________________________________________
dropout_9 (Dropout)          (None, 5)                 0

_________________________________________________________________
dense_8 (Dense)              (None, 5)                 30

_________________________________________________________________
dense_9 (Dense)              (None, 12)                72
=================================================================
Total params: 160,203
Trainable params: 160,203
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss=losses.BinaryCrossentropy(from_logits=True),optimizer='adam',metrics=tf.m
```

```
epochs = 50
history = model.fit(
    train_data,
    epochs=epochs)

Epoch 1/50

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-63a4eb0f112a> in <module>()
      2 history = model.fit(
      3     train_data,
----> 4     epochs=epochs)

/usr/local/lib/python3.7/dist-packages/keras/engine/training.py in fit(self, x, y, batch_siz
   1182                 _r=1):
   1183               callbacks.on_train_batch_begin(step)
-> 1184               tmp_logs = self.train_function(iterator)
   1185               if data_handler.should_sync:
   1186                 context.async_wait()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in __call__(s
    883 
    884       with OptionalXlaContext(self._jit_compile):
--> 885         result = self._call(*args, **kwds)
    886 
    887       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in _call(self
    931       # This is the first call of __call__, so we have to initialize.
    932       initializers = []
--> 933       self._initialize(args, kwds, add_initializers_to=initializers)
    934     finally:
    935       # At this point we know that the initialization is complete (or less

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in _initializ
    758     self._concrete_stateful_fn = (
    759         self._stateful_fn._get_concrete_function_internal_garbage_collected(  # pyli
--> 760             *args, **kwds))
    761 
    762     def invalid_creator_scope(*unused_args, **unused_kwds):

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _get_concrete_
   3064       args, kwargs = None, None
   3065     with self._lock:
-> 3066       graph_function, _ = self._maybe_define_function(args, kwargs)
   3067     return graph_function
   3068 

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _maybe_define_
```

```
   3461 
   3462           self._function_cache.missed.add(call_context_key)
-> 3463           graph_function = self._create_graph_function(args, kwargs)
   3464           self._function_cache.primary[cache_key] = graph_function
   3465 

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _create_graph_
   3306             arg_names=arg_names,
   3307             override_flat_arg_shapes=override_flat_arg_shapes,
-> 3308             capture_by_value=self._capture_by_value),
   3309         self._function_attributes,
   3310         function_spec=self.function_spec,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/func_graph.py in func_gra
   1005         _, original_func = tf_decorator.unwrap(python_func)
   1006 
-> 1007       func_outputs = python_func(*func_args, **func_kwargs)
   1008 
   1009       # invariant: `func_outputs` contains only Tensors, CompositeTensors,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in wrapped_fn
    666         # the function a weak reference to itself to avoid a reference cycle.
    667         with OptionalXlaContext(compile_with_xla):
--> 668           out = weak_wrapped_fn().__wrapped__(*args, **kwds)
    669         return out
    670 

/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/func_graph.py in wrapper(
    992           except Exception as e:  # pylint:disable=broad-except
    993             if hasattr(e, "ag_error_metadata"):
--> 994               raise e.ag_error_metadata.to_exception(e)
    995             else:
    996               raise

ValueError: in user code:

    /usr/local/lib/python3.7/dist-packages/keras/engine/training.py:853 train_function  *
        return step_function(self, iterator)
    /usr/local/lib/python3.7/dist-packages/keras/engine/training.py:842 step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/distribute/distribute_lib.py:12
        return self._extended.call_for_each_replica(fn, args=args, kwargs=kwargs)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/distribute/distribute_lib.py:28
        return self._call_for_each_replica(fn, args, kwargs)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/distribute/distribute_lib.py:36
        return fn(*args, **kwargs)
    /usr/local/lib/python3.7/dist-packages/keras/engine/training.py:835 run_step  **
        outputs = model.train_step(data)
    /usr/local/lib/python3.7/dist-packages/keras/engine/training.py:789 train_step
        y, y_pred, sample_weight, regularization_losses=self.losses)
    /usr/local/lib/python3.7/dist-packages/keras/engine/compile_utils.py:201 __call__
        loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    /usr/local/lib/python3.7/dist-packages/keras/losses.py:141 __call__
        losses = call_fn(y_true, y_pred)
    /usr/local/lib/python3.7/dist-packages/keras/losses.py:245 call  **
        return ag_fn(y_true, y_pred, **self._fn_kwargs)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:206 wrapper
        return target(*args, **kwargs)
    /usr/local/lib/python3.7/dist-packages/keras/losses.py:1809 binary_crossentropy
        backend.binary_crossentropy(y_true, y_pred, from_logits=from_logits),
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:206 wrapper
        return target(*args, **kwargs)
    /usr/local/lib/python3.7/dist-packages/keras/backend.py:5000 binary_crossentropy
         return tf.nn.sigmoid_cross_entropy_with_logits(labels=target, logits=output)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:206 wrapper
        return target(*args, **kwargs)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/nn_impl.py:246 sigmoid_cros
        logits=logits, labels=labels, name=name)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:206 wrapper
        return target(*args, **kwargs)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/nn_impl.py:133 sigmoid_cros
        (logits.get_shape(), labels.get_shape()))

    ValueError: logits and labels must have the same shape ((None, 12) vs (None, 1))
```

```
loss, accuracy = model.evaluate(test_data)
print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
1/1 [==============================] - 0s 170ms/step - loss: 0.5813 - binary_accuracy: 0.720
Loss:  0.5813011527061462
Accuracy:  0.7200000286102295
```