```
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np

print("Loading 'ag_news_subset' dataset...")

# Load the dataset
(raw_train, raw_test), metadata = tfds.load(
    'ag_news_subset',
    split=['train', 'test'],
    with_info=True,
    as_supervised=True  # Loads as (description_text, label)
)

print("Dataset loaded successfully.")
```

```
Loading 'ag_news_subset' dataset...
WARNING:absl:Variant folder /root/tensorflow_datasets/ag_news_subset/1.0.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/t
```

Dl Completed...: 100% 1/1 [00:04<00:00, 4.28s/ url]

Dl Size...: 100% 11/11 [00:04<00:00, 3.86s/ MiB]

Extraction completed...: 100% 4/4 [00:04<00:00, 4.47s/ file]

```
Dataset ag_news_subset downloaded and prepared to /root/tensorflow_datasets/ag_news_subset/1.0.0. Subsequent calls will reuse th
Dataset loaded successfully.
```

```
# Get the class names from metadata
class_names = metadata.features['label'].names
print("Class names:", class_names)
# You should see: ['World', 'Sports', 'Business', 'Sci/Tech']

print("\nHere's an example article:")
for review, label in raw_train.take(1):
    review_text = review.numpy().decode('utf-8')
    review_label = class_names[label.numpy()]

    print(f"LABEL: {review_label}")
    print(f"ARTICLE: {review_text[:500]}...")
```

```
Class names: ['World', 'Sports', 'Business', 'Sci/Tech']

Here's an example article:
LABEL: Sci/Tech
ARTICLE: AMD #39;s new dual-core Opteron chip is designed mainly for corporate computing applications, including databases, Web
```

```
VOCAB_SIZE = 10000
MAX_SEQUENCE_LENGTH = 100

# Create the vectorization layer
vectorize_layer = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE,
    output_mode='int',
    output_sequence_length=MAX_SEQUENCE_LENGTH
)

# Adapt the layer to the training text
print("Building the vocabulary...")
train_text = raw_train.map(lambda text, label: text)
vectorize_layer.adapt(train_text)
print("Vocabulary built.")
```

```
Building the vocabulary...
Vocabulary built.
```

```
# This dictionary will hold our final datasets
datasets = {}

# --- Create a validation split (20% of train data) ---
num_train = metadata.splits['train'].num_examples
num_val = int(num_train * 0.2)  # 20% for validation

val_set = raw_train.take(num_val)
```

```
train_set = raw_train.skip(num_val)

# --- Create the preprocessing function ---
def vectorize_text(text, label):
    text = vectorize_layer(text)
    return text, label

# --- Apply the function and batch the datasets ---
datasets['train'] = train_set.map(vectorize_text).batch(64).prefetch(tf.data.AUTOTUNE)
datasets['val'] = val_set.map(vectorize_text).batch(64).prefetch(tf.data.AUTOTUNE)
datasets['test'] = raw_test.map(vectorize_text).batch(64).prefetch(tf.data.AUTOTUNE)

print("All datasets are vectorized and batched.")
print(f"New training set size: {num_train - num_val}")
print(f"New validation set size: {num_val}")
```

```
All datasets are vectorized and batched.
New training set size: 96000
New validation set size: 24000
```

```
EMBEDDING_DIM = 64
LSTM_UNITS = 64

model = tf.keras.Sequential([
    # 1. The Embedding layer
    tf.keras.layers.Embedding(VOCAB_SIZE, EMBEDDING_DIM),

    # 2. The LSTM layer
    tf.keras.layers.LSTM(LSTM_UNITS),

    # 3. The classification layers
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),

    # --- KEY CHANGE HERE ---
    # Final output layer
    # 4 units (one for each class)
    # 'softmax' activation for a probability distribution
    tf.keras.layers.Dense(4, activation='softmax')
])

model.summary()
```

Model: "sequential"

Layer (type)	**Output Shape**	**Param #**
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)

```
model.compile(
    optimizer='adam',
    # --- KEY CHANGE HERE ---
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

print("Model compiled.")
```

```
Model compiled.
```

```
EPOCHS = 10

print("Starting training...")

history = model.fit(
    datasets['train'],
```

```
    epochs=EPOCHS,
    validation_data=datasets['val']
)

print("Training finished.")
```

```
Starting training...
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 41s 22ms/step - accuracy: 0.2530 - loss: 1.3865 - val_accuracy: 0.2535 - val_loss: 1.3857
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 36s 22ms/step - accuracy: 0.2502 - loss: 1.3852 - val_accuracy: 0.4222 - val_loss: 1.1604
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 33s 22ms/step - accuracy: 0.7030 - loss: 0.7239 - val_accuracy: 0.8961 - val_loss: 0.3195
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 33s 22ms/step - accuracy: 0.9078 - loss: 0.3082 - val_accuracy: 0.8976 - val_loss: 0.3258
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 32s 21ms/step - accuracy: 0.9273 - loss: 0.2468 - val_accuracy: 0.8970 - val_loss: 0.3256
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 33s 22ms/step - accuracy: 0.9390 - loss: 0.2034 - val_accuracy: 0.8963 - val_loss: 0.3750
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 40s 21ms/step - accuracy: 0.9494 - loss: 0.1667 - val_accuracy: 0.8952 - val_loss: 0.4105
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 42s 22ms/step - accuracy: 0.9592 - loss: 0.1339 - val_accuracy: 0.8903 - val_loss: 0.4627
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 40s 21ms/step - accuracy: 0.9665 - loss: 0.1078 - val_accuracy: 0.8857 - val_loss: 0.5360
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 33s 22ms/step - accuracy: 0.9724 - loss: 0.0895 - val_accuracy: 0.8851 - val_loss: 0.5813
Training finished.
```

```
print("Evaluating on test data...")
loss, accuracy = model.evaluate(datasets['test'])

print(f"\nTest Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Evaluating on test data...
119/119 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8901 - loss: 0.5920

Test Loss: 0.6165
Test Accuracy: 88.36%
```