```
# Cell 1 - Setup Environment and Libraries
# Install the correct TensorFlow version and import required packages

# Uninstall conflicting versions
!pip uninstall -y tensorflow tf-nightly -q

# Install TensorFlow 2.19 and TensorFlow Datasets
!pip install tensorflow==2.19.0 tensorflow-datasets -q

# Import libraries
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import os

# Verify TensorFlow installation and GPU availability
print("TensorFlow version:", tf.__version__)
print("GPU available:", tf.config.list_physical_devices('GPU'))

# Note: In Colab, restart runtime if version mismatch occurs
if tf.__version__ != "2.19.0":
    print("\n⚠️ TensorFlow version is not 2.19.0. Restart runtime and rerun this cell.")
```

```
WARNING: Skipping tf-nightly as it is not installed.
TensorFlow version: 2.19.0
GPU available: []
```

```
# Cell 2 - Download SMS Spam Collection Dataset
# Short: Get training and validation TSV files

!wget https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
!wget https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv

train_file_path = "train-data.tsv"
test_file_path = "valid-data.tsv"
```

```
--2025-10-15 09:10:36--  https://cdn.freecodecamp.org/project-data/sms/train-data.tsv
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)... 104.26.2.33, 104.26.3.33, 172.67.70.149, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|104.26.2.33|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 358233 (350K) [text/tab-separated-values]
Saving to: ‘train-data.tsv.4’

train-data.tsv.4    100%[===================>] 349.84K  --.-KB/s    in 0.04s

2025-10-15 09:10:36 (8.33 MB/s) - ‘train-data.tsv.4’ saved [358233/358233]

--2025-10-15 09:10:36--  https://cdn.freecodecamp.org/project-data/sms/valid-data.tsv
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)... 104.26.2.33, 104.26.3.33, 172.67.70.149, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|104.26.2.33|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 118774 (116K) [text/tab-separated-values]
Saving to: ‘valid-data.tsv.4’

valid-data.tsv.4    100%[===================>] 115.99K  --.-KB/s    in 0.01s

2025-10-15 09:10:37 (8.26 MB/s) - ‘valid-data.tsv.4’ saved [118774/118774]
```

```
# Cell 3 - Load data using pandas and inspect
# Short: Load TSV files into DataFrames and explore shape and label distribution

train_df = pd.read_csv(train_file_path, sep='\t', header=None, names=['label', 'message'])
test_df = pd.read_csv(test_file_path, sep='\t', header=None, names=['label', 'message'])

print("Training set shape:", train_df.shape)
print("Validation set shape:", test_df.shape)

print("\nLabel distribution (train):")
print(train_df['label'].value_counts())

print("\nLabel distribution (validation):")
print(test_df['label'].value_counts())

# Display sample messages
print("\nSample messages:")
print(train_df.head())
```

```
Training set shape: (4179, 2)
Validation set shape: (1392, 2)

Label distribution (train):
label
ham     3619
spam     560
Name: count, dtype: int64

Label distribution (validation):
label
ham     1205
spam     187
Name: count, dtype: int64

Sample messages:
  label                                            message
0   ham  ahhhh...just woken up!had a bad dream about u ...
1   ham                          you can never do nothing
2   ham  now u sound like manky scouse boy steve,like! ...
3   ham  mum say we wan to go then go... then she can s...
4   ham  never y lei... i v lazy... got wat? dat day ü ...
```

```
# Cell 4 - Encode labels as 0 (ham) and 1 (spam)
# Short: Convert text labels to numeric for model training

le = LabelEncoder()
y_train = le.fit_transform(train_df['label'])  # ham=0, spam=1
y_test = le.transform(test_df['label'])

X_train = train_df['message'].values
X_test = test_df['message'].values

# Inspect a few samples
print("Training labels sample:", y_train[:10])
print("\nTraining messages sample:")
print(X_train[:5])
```

```
Training labels sample: [0 0 0 0 0 0 0 0 1 0]

Training messages sample:
['ahhhh...just woken up!had a bad dream about u tho,so i dont like u right now :) i didnt know anything about comedy night but i gu
 'you can never do nothing'
 'now u sound like manky scouse boy steve,like! i is travelling on da bus home.wot has u inmind 4 recreation dis eve?'
 'mum say we wan to go then go... then she can shun bian watch da glass exhibition...'
 'never y lei... i v lazy... got wat? dat day ü send me da url cant work one...']
```

```
# Cell 5 - Tokenize text messages and pad sequences
# Short: Convert words to integers and pad sequences to fixed length

vocab_size = 2000     # increased vocab size to better capture spam words
max_length = 100
oov_token = "<OOV>"

# Initialize tokenizer and fit on training messages
tokenizer = keras.preprocessing.text.Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(X_train)

# Convert text to sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Pad sequences to max_length
X_train_padded = keras.preprocessing.sequence.pad_sequences(
    X_train_seq, maxlen=max_length, padding='post', truncating='post'
)
X_test_padded = keras.preprocessing.sequence.pad_sequences(
    X_test_seq, maxlen=max_length, padding='post', truncating='post'
)

# Inspect shapes and example tokenized sequence
print("Training padded shape:", X_train_padded.shape)
print("Validation padded shape:", X_test_padded.shape)
print("\nExample tokenized & padded sequence (first 20 tokens of first message):")
print(X_train_padded[0][:20])
```

```
Training padded shape: (4179, 100)
Validation padded shape: (1392, 100)

Example tokenized & padded sequence (first 20 tokens of first message):
[  1  37   1  45 143   5 402 767  79   7 726  24   2  94  56   7 163  20
   2 461]
```

```
# Cell 6 - Build Neural Network
# Short: Define a sequential model with embedding and dense layers for binary classification

model = keras.Sequential([
    keras.layers.Embedding(input_dim=vocab_size, output_dim=16),
    keras.layers.GlobalAveragePooling1D(),
    keras.layers.Dense(16, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')  # sigmoid output for binary (ham/spam)
])

# Compile the model
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Show model architecture
model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	?	0 (unbuilt)
global_average_pooling1d_4 (GlobalAveragePooling1D)	?	0
dense_8 (Dense)	?	0 (unbuilt)
dense_9 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

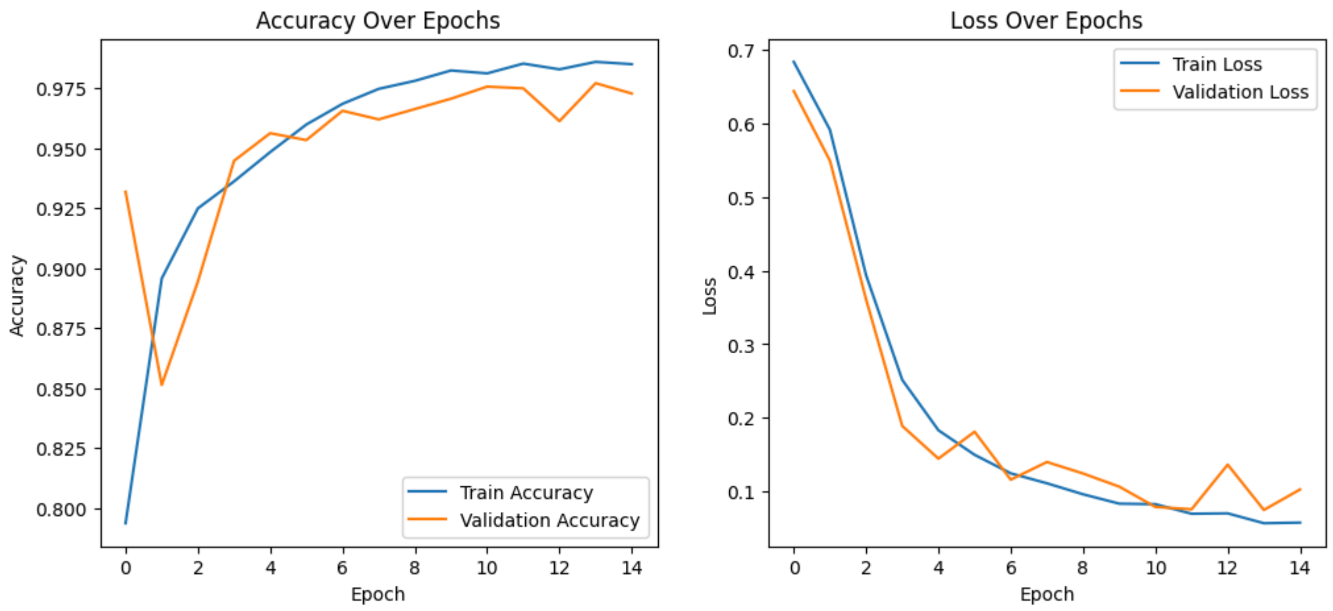
```
# Cell 7 - Train the model with class weighting to handle imbalance
# Short: Train with balanced class weights and more epochs

from sklearn.utils import class_weight

# Compute class weights to balance ham vs spam
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
class_weights_dict = dict(enumerate(class_weights))

# Train model
history = model.fit(
    X_train_padded,
    y_train,
    epochs=15,          # more epochs for better learning
    batch_size=32,
    validation_data=(X_test_padded, y_test),
    class_weight=class_weights_dict,
    verbose=2
)
```

```
Epoch 1/15
131/131 - 4s - 27ms/step - accuracy: 0.7937 - loss: 0.6837 - val_accuracy: 0.9318 - val_loss: 0.6439
Epoch 2/15
131/131 - 2s - 17ms/step - accuracy: 0.8957 - loss: 0.5913 - val_accuracy: 0.8513 - val_loss: 0.5493
Epoch 3/15
131/131 - 1s - 8ms/step - accuracy: 0.9249 - loss: 0.3938 - val_accuracy: 0.8944 - val_loss: 0.3608
Epoch 4/15
```

```
131/131 - 1s - 9ms/step - accuracy: 0.9361 - loss: 0.2516 - val_accuracy: 0.9447 - val_loss: 0.1890
Epoch 5/15
131/131 - 1s - 10ms/step - accuracy: 0.9483 - loss: 0.1832 - val_accuracy: 0.9562 - val_loss: 0.1447
Epoch 6/15
131/131 - 2s - 16ms/step - accuracy: 0.9598 - loss: 0.1499 - val_accuracy: 0.9533 - val_loss: 0.1812
Epoch 7/15
131/131 - 1s - 10ms/step - accuracy: 0.9684 - loss: 0.1247 - val_accuracy: 0.9655 - val_loss: 0.1161
Epoch 8/15
131/131 - 1s - 5ms/step - accuracy: 0.9746 - loss: 0.1112 - val_accuracy: 0.9619 - val_loss: 0.1401
Epoch 9/15
131/131 - 1s - 4ms/step - accuracy: 0.9780 - loss: 0.0963 - val_accuracy: 0.9662 - val_loss: 0.1245
Epoch 10/15
131/131 - 0s - 4ms/step - accuracy: 0.9823 - loss: 0.0835 - val_accuracy: 0.9705 - val_loss: 0.1067
Epoch 11/15
131/131 - 1s - 4ms/step - accuracy: 0.9811 - loss: 0.0828 - val_accuracy: 0.9756 - val_loss: 0.0791
Epoch 12/15
131/131 - 1s - 4ms/step - accuracy: 0.9852 - loss: 0.0698 - val_accuracy: 0.9749 - val_loss: 0.0759
Epoch 13/15
131/131 - 1s - 4ms/step - accuracy: 0.9828 - loss: 0.0703 - val_accuracy: 0.9612 - val_loss: 0.1365
Epoch 14/15
131/131 - 1s - 4ms/step - accuracy: 0.9859 - loss: 0.0570 - val_accuracy: 0.9770 - val_loss: 0.0750
Epoch 15/15
131/131 - 1s - 4ms/step - accuracy: 0.9849 - loss: 0.0578 - val_accuracy: 0.9727 - val_loss: 0.1028
```

```python
# Cell 8 - Evaluate model performance on validation set
# Short: Check accuracy and plot training history

loss, accuracy = model.evaluate(X_test_padded, y_test)
print(f"\nValidation Accuracy: {accuracy:.4f}")
print(f"Validation Loss: {loss:.4f}")

# Plot training/validation accuracy and loss
plt.figure(figsize=(12,5))

# Accuracy plot
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9727 - loss: 0.1028

Validation Accuracy: 0.9727
Validation Loss: 0.1028
```



```
# Cell 9 - Predict a single SMS message
# Short: Convert text to sequence, pad it, predict probability, return label

def predict_message(pred_text):
    # Convert text to integer sequence
    seq = tokenizer.texts_to_sequences([pred_text])

    # Pad sequence
    padded = keras.preprocessing.sequence.pad_sequences(seq, maxlen=max_length, padding='post')

    # Predict probability of spam
    prob = model.predict(padded, verbose=0)[0][0]

    # Lower threshold to 0.25 for better spam detection
    label = "spam" if prob > 0.25 else "ham"

    return [float(prob), label]

# Example usage
pred_text = "how are you doing today?"
prediction = predict_message(pred_text)
print(prediction)
```

```
[0.04660388454794884, 'ham']
```

```
# Cell 10 - Test predict_message function on sample messages
# Short: Automated test for FreeCodeCamp challenge

def test_predictions():
    test_messages = [
        "how are you doing today",
        "sale today! to stop texts call 98912460324",
        "i dont want to go. can we try it a different day? available sat",
        "our new mobile video service is live. just install on your phone to start watching.",
        "you have won £1000 cash! call to claim your prize.",
        "i'll bring it tomorrow. don't forget the milk.",
        "wow, is your arm alright. that happened to me one time too"
    ]

    test_answers = ["ham", "spam", "ham", "spam", "spam", "ham", "ham"]
    passed = True

    for msg, ans in zip(test_messages, test_answers):
        prediction = predict_message(msg)
        if prediction[1] != ans:
            passed = False
    if passed:
        print("You passed the challenge. Great job!")
    else:
        print("You haven't passed yet. Keep trying.")
```