```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.5.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (14.0.0)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (202
Requirement already satisfied: markdown>=2.6.8 in /usr/lib/python3/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.3.6)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow) (3
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->te
```

```
import os, pathlib, shutil, random
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
# Download and extract IMDB dataset
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  16.7M      0  0:00:04  0:00:04 --:--:-- 16.7M
```

```
import os

def review_summary(data_dir="aclImdb", sample_count=5):
    for dataset_type in ["train", "test"]:
        print(f"\nSummary of '{dataset_type}' split:")
        for category in ["pos", "neg"]:
            print(f"  Sentiment: {category}")
            path_to_folder = os.path.join(data_dir, dataset_type, category)
            review_files = os.listdir(path_to_folder)[:sample_count]
            for index, review_file in enumerate(review_files):
                review_path = os.path.join(path_to_folder, review_file)
                with open(review_path, "r", encoding="utf-8") as review:
                    review_text = review.readlines()
                print(f"\n  File {index + 1}: {review_file}")
                print(f"    Lines in file: {len(review_text)}")
                print(f"    First 5 lines (or fewer):")
            print("    " + "\n    ".join(review_text[:5]).strip())


review_summary() # Call the correct function name: review_summary
```

```
      Lines in file: 1
      First 5 lines (or fewer):
      I'm trying to picture the pitch for Dark Angel. "I'm thinking Matrix, I'm thinking Bladerunner, I'm thinking that chick that play

  Summary of 'test' split:
    Sentiment: pos

    File 1: 2917_9.txt
      Lines in file: 1
      First 5 lines (or fewer):
      Having read all of the comments on this film I am still amazed at Fox's reluctance to release a full screen restored version in D

    File 2: 11565_7.txt
      Lines in file: 1
      First 5 lines (or fewer):
      This is a cute little French silent comedy about a man who bets another that he can't stay in this castle for one hour due to its

    File 3: 11076_10.txt
      Lines in file: 1
      First 5 lines (or fewer):
      I just wanted to inform anyone who may be interested that the the movie "New Jersey Drive" was my personal favorite off alltime.

    File 4: 3058_10.txt
      Lines in file: 1
      First 5 lines (or fewer):
      I saw this film when it was released to the minor cinemas in the UK some 50 years ago; and the memory remains of a great musical

    File 5: 8510_8.txt
      Lines in file: 1
      First 5 lines (or fewer):
      I'm sick and tired of people complaining that Never Say Never Again is just a weak remake of Thunderball. Yes, that movie's influ
    Sentiment: neg

    File 1: 12278_3.txt
      Lines in file: 1
      First 5 lines (or fewer):
      Tony Curtis and Skip Homier both are wearing black with white trim canvas shoes in the scenes just before and after the swimming

    File 2: 2590_1.txt
      Lines in file: 1
      First 5 lines (or fewer):
      I couldn't believe how bad this film was, and trust me, I was not expecting a masterpiece from a made-for-cable film. I taped it

    File 3: 9522_1.txt
      Lines in file: 1
      First 5 lines (or fewer):
      I'm a bit spooked by some of these reviews praising A.K.A. Not only do they sound as if they were written by the same person, but

    File 4: 10329_1.txt
      Lines in file: 1
      First 5 lines (or fewer):
      This movie is #1 in the list of worst movies I have ever seen, with "Lessons for an Assassin" on the #2 spot.<br /><br />The acti

    File 5: 12149_2.txt
      Lines in file: 1
      First 5 lines (or fewer):
      Well Folks, this is another stereotypic portrayla of Gay life however, the additional downside includes poor acting, horrible scr
```

```
import os
import shutil
import random
import pathlib

sample_size = 32
root_path = pathlib.Path("aclImdb")
validation_path = root_path / "val"
training_path = root_path / "train"

for label in ("neg", "pos"):
    os.makedirs(validation_path / label, exist_ok=True)
    file_list = os.listdir(training_path / label)
    random.Random(1337).shuffle(file_list)
    split_count = int(0.2 * len(file_list))
    validation_files = file_list[-split_count:]
    for filename in validation_files:
        source = training_path / label / filename
        destination = validation_path / label / filename
        if not os.path.exists(destination):
            shutil.move(source, destination)
```

```
from tensorflow import keras

# Load datasets with new variable names
training_data = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=sample_size
)

validation_data = keras.utils.text_dataset_from_directory(
    "aclImdb/val", batch_size=sample_size
)

testing_data = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=sample_size
)

text_inputs_only = training_data.map(lambda features, labels: features)
```

```
Found 25000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

1) Cutoff reviews after 150 words

```
# Configure preprocessing parameters
sequence_len = 150
vocab_limit = 10000

vectorizer = layers.TextVectorization(
    max_tokens=vocab_limit,
    output_mode="int",
    output_sequence_length=sequence_len,
)

vectorizer.adapt(text_inputs_only)
```

```
# Vectorized datasets
tokenized_train = training_data.map(
    lambda text, label: (vectorizer(text), label),
    num_parallel_calls=4).take(100)

tokenized_val = validation_data.map(
    lambda text, label: (vectorizer(text), label),
    num_parallel_calls=4).take(10000)

tokenized_test = testing_data.map(
    lambda text, label: (vectorizer(text), label),
    num_parallel_calls=4)
```

```
# Model using an Embedding Layer
input_layer = keras.Input(shape=(None,), dtype="int64")
embedding_output = layers.Embedding(input_dim=vocab_limit, output_dim=128)(input_layer)
bi_lstm = layers.Bidirectional(layers.LSTM(32))(embedding_output)
dropout_layer = layers.Dropout(0.2)(bi_lstm)
final_output = layers.Dense(1, activation="sigmoid")(dropout_layer)

text_classifier = keras.Model(input_layer, final_output)

text_classifier.compile(optimizer="rmsprop",
                        loss="binary_crossentropy",
                        metrics=["accuracy"])

text_classifier.summary()
```

```
Model: "functional_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_3 (InputLayer)      │ (None, None)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ embedding_2 (Embedding)         │ (None, None, 128)      │     1,280,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional_3 (Bidirectional) │ (None, 64)             │        41,216 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_3 (Dropout)             │ (None, 64)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 1)              │            65 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,321,281 (5.04 MB)
 Trainable params: 1,321,281 (5.04 MB)
```

```
model_callbacks = [
    keras.callbacks.ModelCheckpoint("text_classifier.keras", save_best_only=True)
]
```

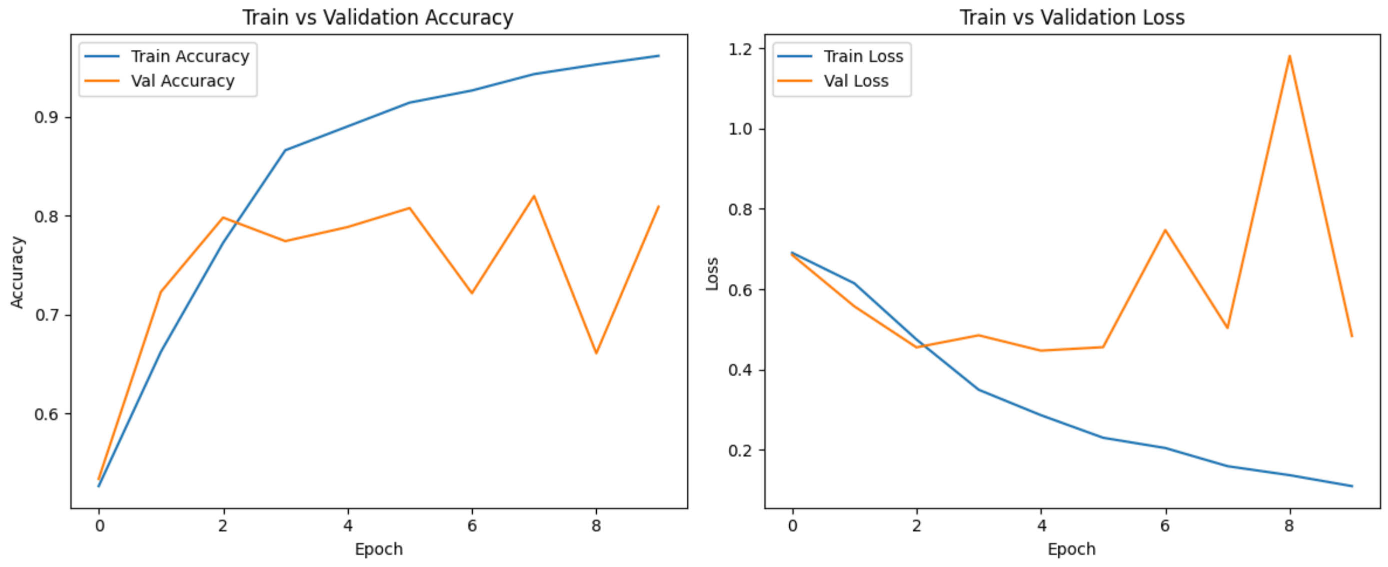
```
training_history = text_classifier.fit(
    tokenized_train, validation_data=tokenized_val, epochs=10, callbacks=model_callbacks
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 89ms/step - accuracy: 0.5034 - loss: 0.6928 - val_accuracy: 0.5342 - val_loss: 0.6856
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.6233 - loss: 0.6487 - val_accuracy: 0.7230 - val_loss: 0.5575
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.7435 - loss: 0.5129 - val_accuracy: 0.7980 - val_loss: 0.4553
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.8462 - loss: 0.3794 - val_accuracy: 0.7742 - val_loss: 0.4853
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.8739 - loss: 0.3100 - val_accuracy: 0.7884 - val_loss: 0.4471
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9122 - loss: 0.2429 - val_accuracy: 0.8076 - val_loss: 0.4558
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - accuracy: 0.9318 - loss: 0.1874 - val_accuracy: 0.7216 - val_loss: 0.7474
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9453 - loss: 0.1553 - val_accuracy: 0.8198 - val_loss: 0.5035
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9527 - loss: 0.1224 - val_accuracy: 0.6610 - val_loss: 1.1811
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9537 - loss: 0.1208 - val_accuracy: 0.8090 - val_loss: 0.4835
```

2) Restrict training samples to 100 3) Validate on 10,000 samples

```
import matplotlib.pyplot as plt

# Extract metrics from training history
metrics_log = training_history.history

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

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

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

# Display the plots
```

```
plt.tight_layout()
plt.show()
```



```
# Load Pretrained Word Embeddings (GloVe)

# Download GloVe vectors
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2025-04-15 20:41:07--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-04-15 20:41:07--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-04-15 20:41:07--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip.2’

glove.6B.zip.2      100%[===================>] 822.24M  5.03MB/s    in 2m 41s

2025-04-15 20:43:49 (5.10 MB/s) - ‘glove.6B.zip.2’ saved [862182613/862182613]

replace glove.6B.50d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
# Build embedding matrix from GloVe
embedding_size = 100
glove_file_path = "glove.6B.100d.txt"

pretrained_vectors = {}
with open(glove_file_path) as file:
    for entry in file:
        token, values = entry.split(maxsplit=1)
        values = np.fromstring(values, "f", sep=" ")
        pretrained_vectors[token] = values

token_list = vectorizer.get_vocabulary()
token_to_index = dict(zip(token_list, range(len(token_list))))

glove_matrix = np.zeros((vocab_limit, embedding_size))
for token, idx in token_to_index.items():
    if idx < vocab_limit:
        vector = pretrained_vectors.get(token)
        if vector is not None:
            glove_matrix[idx] = vector
```

5. Before the layers.Bidirectional layer, consider a) an embedding layer, .

```
# Model using pretrained GloVe embeddings
frozen_embedding = layers.Embedding(
    vocab_limit,
    embedding_size,
    embeddings_initializer=keras.initializers.Constant(glove_matrix),
    trainable=False,
    mask_zero=True,
)

input_tensor = keras.Input(shape=(None,), dtype="int64")
embedded_input = frozen_embedding(input_tensor)
lstm_output = layers.Bidirectional(layers.LSTM(32))(embedded_input)
dropout_output = layers.Dropout(0.2)(lstm_output)
final_prediction = layers.Dense(1, activation="sigmoid")(dropout_output)
glove_model = keras.Model(input_tensor, final_prediction)



input_seq = keras.Input(shape=(None,), dtype="int64")
embedded_seq = frozen_embedding(input_seq)
rnn_output = layers.Bidirectional(layers.LSTM(32))(embedded_seq)
dropout_layer = layers.Dropout(0.2)(rnn_output)
prediction = layers.Dense(1, activation="sigmoid")(dropout_layer)
glove_based_model = keras.Model(input_seq, prediction)

glove_based_model.compile(optimizer="rmsprop",
                          loss="binary_crossentropy",
                          metrics=["accuracy"])

glove_based_model.summary()
```

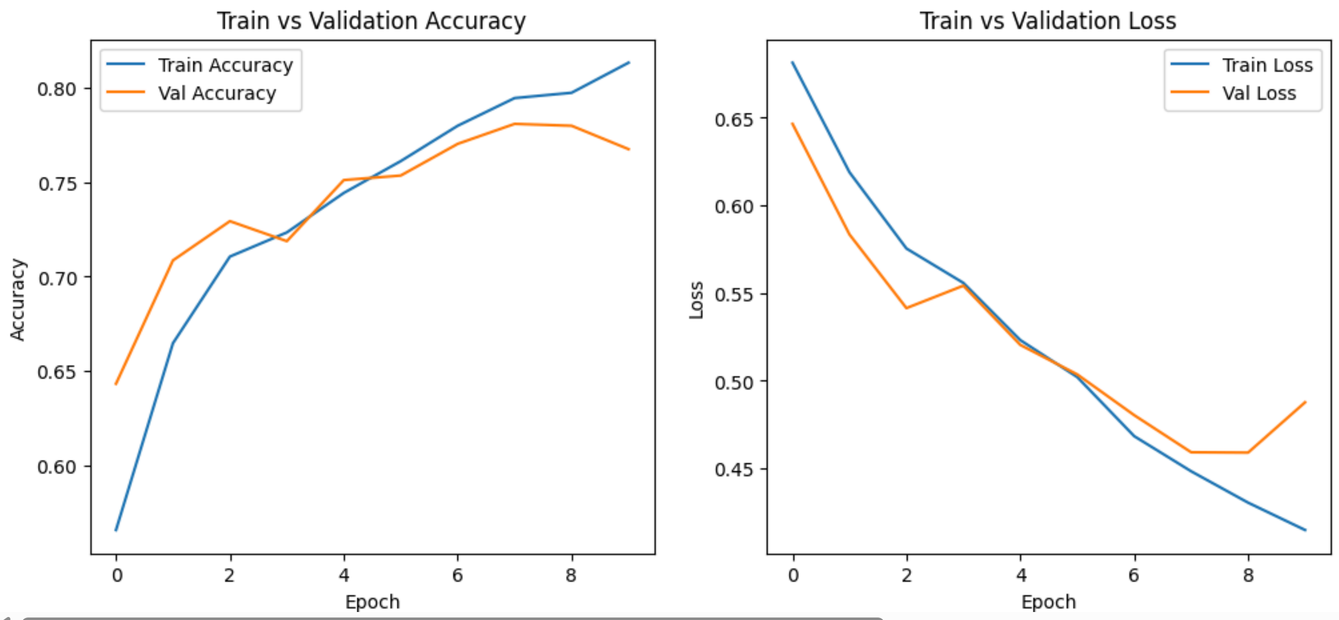
```
Model: "functional_5"
┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)         ┃ Output Shape      ┃    Param # ┃ Connected to     ┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━┩
│ input_layer_5        │ (None, None)      │          0 │ -                │
│ (InputLayer)         │                   │            │                  │
├──────────────────────┼───────────────────┼────────────┼──────────────────┤
│ embedding_3          │ (None, None, 100) │  1,000,000 │ input_layer_5[0]… │
│ (Embedding)          │                   │            │                  │
├──────────────────────┼───────────────────┼────────────┼──────────────────┤
│ not_equal_3          │ (None, None)      │          0 │ input_layer_5[0]… │
│ (NotEqual)           │                   │            │                  │
├──────────────────────┼───────────────────┼────────────┼──────────────────┤
│ bidirectional_5      │ (None, 64)        │     34,048 │ embedding_3[1][0… │
│ (Bidirectional)      │                   │            │ not_equal_3[0][0] │
├──────────────────────┼───────────────────┼────────────┼──────────────────┤
│ dropout_5 (Dropout)  │ (None, 64)        │          0 │ bidirectional_5[… │
├──────────────────────┼───────────────────┼────────────┼──────────────────┤
│ dense_5 (Dense)      │ (None, 1)         │         65 │ dropout_5[0][0]  │
└──────────────────────┴───────────────────┴────────────┴──────────────────┘
 Total params: 1,034,113 (3.94 MB)
```

```
glove_callbacks = [
    keras.callbacks.ModelCheckpoint("glove_based_model.keras", save_best_only=True)
]

glove_training_history = glove_based_model.fit(
    tokenized_train, validation_data=tokenized_val, epochs=10, callbacks=glove_callbacks
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 16s 125ms/step - accuracy: 0.5323 - loss: 0.6982 - val_accuracy: 0.6432 - val_loss: 0.6465
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 124ms/step - accuracy: 0.6501 - loss: 0.6330 - val_accuracy: 0.7086 - val_loss: 0.5834
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 125ms/step - accuracy: 0.7061 - loss: 0.5787 - val_accuracy: 0.7294 - val_loss: 0.5414
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 107ms/step - accuracy: 0.7159 - loss: 0.5569 - val_accuracy: 0.7188 - val_loss: 0.5542
Epoch 5/10
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 123ms/step - accuracy: 0.7305 - loss: 0.5396 - val_accuracy: 0.7512 - val_loss: 0.5205
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 120ms/step - accuracy: 0.7585 - loss: 0.5058 - val_accuracy: 0.7536 - val_loss: 0.5036
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 126ms/step - accuracy: 0.7662 - loss: 0.4855 - val_accuracy: 0.7704 - val_loss: 0.4804
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 126ms/step - accuracy: 0.7886 - loss: 0.4535 - val_accuracy: 0.7810 - val_loss: 0.4592
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 123ms/step - accuracy: 0.7949 - loss: 0.4322 - val_accuracy: 0.7800 - val_loss: 0.4591
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8046 - loss: 0.4222 - val_accuracy: 0.7676 - val_loss: 0.4877
```

```
import matplotlib.pyplot as plt

# Extract metrics from training history
glove_metrics = glove_training_history.history

# Create plot figure
plt.figure(figsize=(12, 5))

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

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

# Display plots
plt.show()
```



b) a pretrained word embedding.

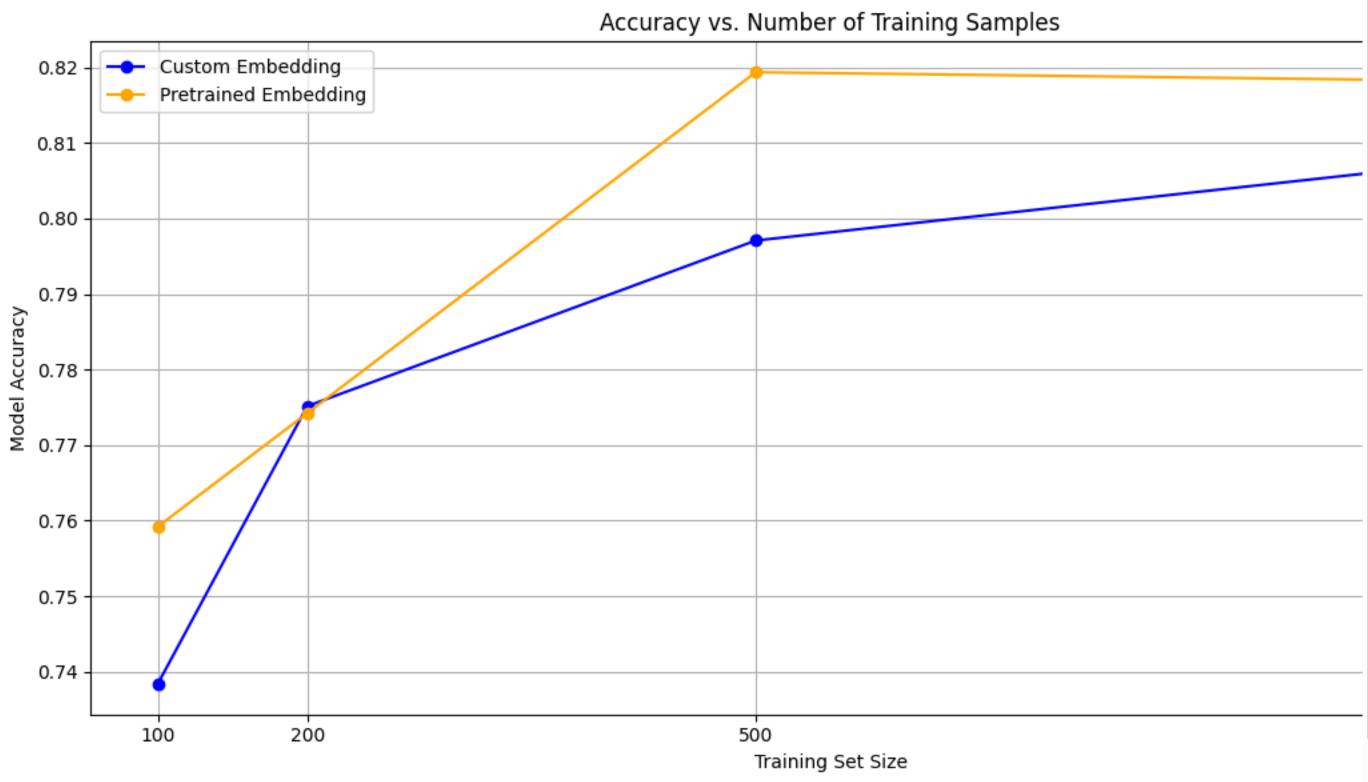
```
import numpy as np
import matplotlib.pyplot as plt
import time

# Sample size settings
sample_sizes = [100, 200, 500, 1000]
```

```
custom_acc_results = []
pretrained_acc_results = []

# Initialize plot
plt.figure(figsize=(12, 6))
plt.title('Accuracy vs. Number of Training Samples')
plt.xlabel('Training Set Size')
plt.ylabel('Model Accuracy')
plt.grid(True)

# Loop over different training sample sizes
for idx, sample_count in enumerate(sample_sizes):
    print(f"\n### Running training with {sample_count} samples ###\n")

    # Prepare subset of training data
    limited_train_data = training_data.map(
        lambda text, label: (vectorizer(text), label)).take(sample_count)

    # Train model with custom embeddings
    print(f"Training model with custom embeddings using {sample_count} samples:")
    text_classifier.fit(
        limited_train_data,
        validation_data=tokenized_val,
        epochs=10,
        verbose=1
    )
    custom_acc = text_classifier.evaluate(tokenized_test, verbose=1)[1]
    custom_acc_results.append(custom_acc)
    print(f"Custom Embedding Accuracy: {custom_acc:.4f}\n")

    # Train model with pretrained embeddings
    print(f"Training model with pretrained embeddings using {sample_count} samples:")
    glove_based_model.fit(
        limited_train_data,
        validation_data=tokenized_val,
        epochs=10,
        verbose=1
    )
    pretrained_acc = glove_based_model.evaluate(tokenized_test, verbose=1)[1]
    pretrained_acc_results.append(pretrained_acc)
    print(f"Pretrained Embedding Accuracy: {pretrained_acc:.4f}\n")

    # Final plot generation after last iteration
    if idx == len(sample_sizes) - 1:
        plt.plot(sample_sizes, custom_acc_results, marker='o', label='Custom Embedding', color='blue')
        plt.plot(sample_sizes, pretrained_acc_results, marker='o', label='Pretrained Embedding', color='orange')

# Final touches to plot
plt.title('Accuracy vs. Number of Training Samples')
plt.xlabel('Training Set Size')
plt.ylabel('Model Accuracy')
plt.xticks(sample_sizes)
plt.grid(True)
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
```

```
### Running training with 100 samples ###

Training model with custom embeddings using 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - accuracy: 0.9906 - loss: 0.0420 - val_accuracy: 0.7452 - val_loss: 1.3427
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9773 - loss: 0.0788 - val_accuracy: 0.7930 - val_loss: 0.7012
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9923 - loss: 0.0291 - val_accuracy: 0.7760 - val_loss: 0.6147
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9965 - loss: 0.0258 - val_accuracy: 0.7872 - val_loss: 0.8526
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9913 - loss: 0.0306 - val_accuracy: 0.7956 - val_loss: 0.8215
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 81ms/step - accuracy: 0.9958 - loss: 0.0179 - val_accuracy: 0.7808 - val_loss: 0.7895
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - accuracy: 0.9949 - loss: 0.0213 - val_accuracy: 0.8006 - val_loss: 0.8624
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - accuracy: 0.9961 - loss: 0.0141 - val_accuracy: 0.7742 - val_loss: 0.9019
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9948 - loss: 0.0143 - val_accuracy: 0.7812 - val_loss: 0.7886
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9901 - loss: 0.0252 - val_accuracy: 0.7444 - val_loss: 0.8845
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.7349 - loss: 0.9239
Custom Embedding Accuracy: 0.7384

Training model with pretrained embeddings using 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8260 - loss: 0.3983 - val_accuracy: 0.7656 - val_loss: 0.4956
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8341 - loss: 0.3715 - val_accuracy: 0.7688 - val_loss: 0.4871
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 107ms/step - accuracy: 0.8444 - loss: 0.3575 - val_accuracy: 0.7688 - val_loss: 0.4792
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8568 - loss: 0.3416 - val_accuracy: 0.7766 - val_loss: 0.4723
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 107ms/step - accuracy: 0.8678 - loss: 0.3212 - val_accuracy: 0.7570 - val_loss: 0.5248
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 102ms/step - accuracy: 0.8696 - loss: 0.3027 - val_accuracy: 0.7710 - val_loss: 0.4955
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8880 - loss: 0.2857 - val_accuracy: 0.7762 - val_loss: 0.4927
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 106ms/step - accuracy: 0.8907 - loss: 0.2647 - val_accuracy: 0.7710 - val_loss: 0.5052
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 106ms/step - accuracy: 0.9095 - loss: 0.2345 - val_accuracy: 0.7662 - val_loss: 0.5435
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 109ms/step - accuracy: 0.9022 - loss: 0.2258 - val_accuracy: 0.7642 - val_loss: 0.5486
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.7598 - loss: 0.5583
Pretrained Embedding Accuracy: 0.7592


### Running training with 200 samples ###

Training model with custom embeddings using 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 69ms/step - accuracy: 0.9566 - loss: 0.1169 - val_accuracy: 0.7814 - val_loss: 0.4749
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9657 - loss: 0.1109 - val_accuracy: 0.7882 - val_loss: 0.5287
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9743 - loss: 0.0830 - val_accuracy: 0.8002 - val_loss: 0.5015
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9821 - loss: 0.0615 - val_accuracy: 0.8020 - val_loss: 0.5103
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9849 - loss: 0.0484 - val_accuracy: 0.8016 - val_loss: 0.5764
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9883 - loss: 0.0379 - val_accuracy: 0.7994 - val_loss: 0.5914
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9937 - loss: 0.0255 - val_accuracy: 0.7924 - val_loss: 0.7174
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9951 - loss: 0.0211 - val_accuracy: 0.7776 - val_loss: 0.8178
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 70ms/step - accuracy: 0.9959 - loss: 0.0178 - val_accuracy: 0.8022 - val_loss: 0.7881
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9977 - loss: 0.0097 - val_accuracy: 0.7896 - val_loss: 0.7773
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.7703 - loss: 0.8046
Custom Embedding Accuracy: 0.7752

Training model with pretrained embeddings using 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.8949 - loss: 0.2624 - val_accuracy: 0.7922 - val_loss: 0.4432
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.8866 - loss: 0.2660 - val_accuracy: 0.7734 - val_loss: 0.4909
Epoch 3/10
```

```
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.8984 - loss: 0.2476 - val_accuracy: 0.7934 - val_loss: 0.4359
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9085 - loss: 0.2364 - val_accuracy: 0.7994 - val_loss: 0.4412
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9140 - loss: 0.2165 - val_accuracy: 0.7944 - val_loss: 0.4497
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.9217 - loss: 0.1987 - val_accuracy: 0.8022 - val_loss: 0.4579
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.9254 - loss: 0.1870 - val_accuracy: 0.8058 - val_loss: 0.4604
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9305 - loss: 0.1803 - val_accuracy: 0.8068 - val_loss: 0.4732
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.9386 - loss: 0.1625 - val_accuracy: 0.7962 - val_loss: 0.4817
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.9472 - loss: 0.1476 - val_accuracy: 0.7744 - val_loss: 0.5143
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.7713 - loss: 0.5116
Pretrained Embedding Accuracy: 0.7742


### Running training with 500 samples ###

Training model with custom embeddings using 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9526 - loss: 0.1282 - val_accuracy: 0.8246 - val_loss: 0.3993
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9559 - loss: 0.1285 - val_accuracy: 0.8344 - val_loss: 0.3852
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9685 - loss: 0.0922 - val_accuracy: 0.8078 - val_loss: 0.5088
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9780 - loss: 0.0658 - val_accuracy: 0.8256 - val_loss: 0.4839
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9834 - loss: 0.0489 - val_accuracy: 0.8214 - val_loss: 0.5119
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9914 - loss: 0.0306 - val_accuracy: 0.8250 - val_loss: 0.6281
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9928 - loss: 0.0244 - val_accuracy: 0.8250 - val_loss: 0.7846
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9945 - loss: 0.0180 - val_accuracy: 0.8106 - val_loss: 0.7615
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9958 - loss: 0.0143 - val_accuracy: 0.8026 - val_loss: 0.8521
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9974 - loss: 0.0103 - val_accuracy: 0.8156 - val_loss: 0.9101
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.7976 - loss: 1.0127
Custom Embedding Accuracy: 0.7971

Training model with pretrained embeddings using 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.9051 - loss: 0.2298 - val_accuracy: 0.7854 - val_loss: 0.4497
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.8882 - loss: 0.2638 - val_accuracy: 0.8092 - val_loss: 0.4108
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9012 - loss: 0.2408 - val_accuracy: 0.8008 - val_loss: 0.4289
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 43s 85ms/step - accuracy: 0.9044 - loss: 0.2286 - val_accuracy: 0.8160 - val_loss: 0.4049
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9131 - loss: 0.2161 - val_accuracy: 0.8194 - val_loss: 0.3908
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.9228 - loss: 0.1962 - val_accuracy: 0.8138 - val_loss: 0.4027
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9256 - loss: 0.1862 - val_accuracy: 0.8136 - val_loss: 0.4180
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 83ms/step - accuracy: 0.9308 - loss: 0.1736 - val_accuracy: 0.8184 - val_loss: 0.4100
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.9377 - loss: 0.1572 - val_accuracy: 0.8110 - val_loss: 0.4249
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9421 - loss: 0.1492 - val_accuracy: 0.8224 - val_loss: 0.4494
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.8174 - loss: 0.4550
Pretrained Embedding Accuracy: 0.8194


### Running training with 1000 samples ###

Training model with custom embeddings using 1000 samples:
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9936 - loss: 0.0240 - val_accuracy: 0.8278 - val_loss: 0.4735
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9925 - loss: 0.0289 - val_accuracy: 0.8194 - val_loss: 0.5397
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 40s 64ms/step - accuracy: 0.9963 - loss: 0.0153 - val_accuracy: 0.8222 - val_loss: 0.6670
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9973 - loss: 0.0117 - val_accuracy: 0.8194 - val_loss: 0.7993
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9976 - loss: 0.0087 - val_accuracy: 0.8102 - val_loss: 0.8882
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 40s 63ms/step - accuracy: 0.9984 - loss: 0.0062 - val_accuracy: 0.8068 - val_loss: 1.0041
```

```
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 62ms/step - accuracy: 0.9992 - loss: 0.0042 - val_accuracy: 0.8162 - val_loss: 1.1607
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 40s 64ms/step - accuracy: 0.9994 - loss: 0.0023 - val_accuracy: 0.8250 - val_loss: 1.2519
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9996 - loss: 0.0020 - val_accuracy: 0.8150 - val_loss: 1.2353
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 40s 63ms/step - accuracy: 0.9991 - loss: 0.0034 - val_accuracy: 0.8262 - val_loss: 1.3711
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.8047 - loss: 1.5448
Custom Embedding Accuracy: 0.8080

Training model with pretrained embeddings using 1000 samples:
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 52s 83ms/step - accuracy: 0.9407 - loss: 0.1547 - val_accuracy: 0.8276 - val_loss: 0.3747
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 52s 82ms/step - accuracy: 0.9401 - loss: 0.1601 - val_accuracy: 0.8340 - val_loss: 0.3830
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 81ms/step - accuracy: 0.9431 - loss: 0.1489 - val_accuracy: 0.8326 - val_loss: 0.3945
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 82ms/step - accuracy: 0.9460 - loss: 0.1401 - val_accuracy: 0.8330 - val_loss: 0.4032
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 80ms/step - accuracy: 0.9516 - loss: 0.1304 - val_accuracy: 0.8310 - val_loss: 0.4169
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 53s 84ms/step - accuracy: 0.9575 - loss: 0.1180 - val_accuracy: 0.8300 - val_loss: 0.4303
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 81ms/step - accuracy: 0.9590 - loss: 0.1151 - val_accuracy: 0.8154 - val_loss: 0.4661
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 81ms/step - accuracy: 0.9621 - loss: 0.1065 - val_accuracy: 0.8226 - val_loss: 0.4718
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 81ms/step - accuracy: 0.9638 - loss: 0.0992 - val_accuracy: 0.8204 - val_loss: 0.4877
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 82ms/step - accuracy: 0.9675 - loss: 0.0928 - val_accuracy: 0.8194 - val_loss: 0.5192
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.8141 - loss: 0.5293
Pretrained Embedding Accuracy: 0.8182
```

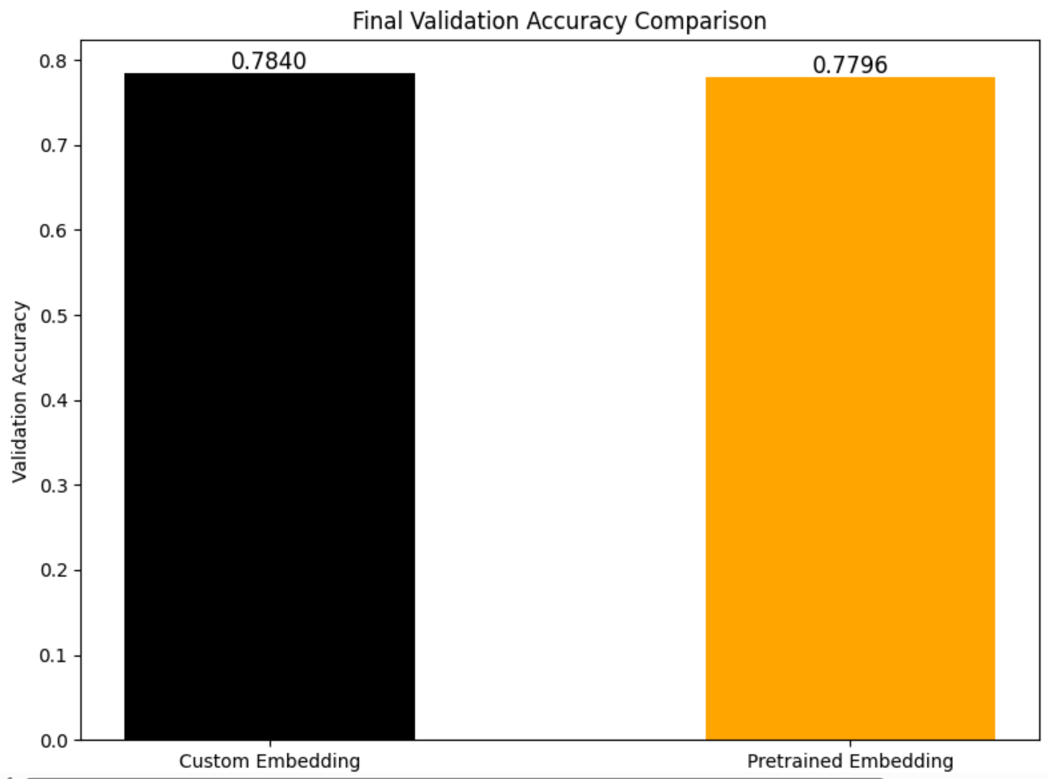


4.) Consider only the top 10,000 words

```
import pandas as pd

# Store results for custom embedding model
custom_results = {
    "Sample Size": sample_sizes,
    "Custom Embedding Accuracy": custom_acc_results,
}

# Store results for pretrained embedding model
pretrained_results = {
    "Sample Size": sample_sizes,
    "Pretrained Embedding Accuracy": pretrained_acc_results,
}

# Combine results into a single DataFrame
results_summary = pd.DataFrame({
    "Sample Size": sample_sizes,
    "Custom Embedding Accuracy": custom_acc_results,
    "Pretrained Embedding Accuracy": pretrained_acc_results
})

# Output the summary table
print("Accuracy Comparison Summary:")
print(results_summary)
```

```
Accuracy Comparison Summary:
   Sample Size  Custom Embedding Accuracy  Pretrained Embedding Accuracy
0          100                    0.73840                        0.75920
1          200                    0.77516                        0.77424
2          500                    0.79708                        0.81936
3         1000                    0.80796                        0.81816
```

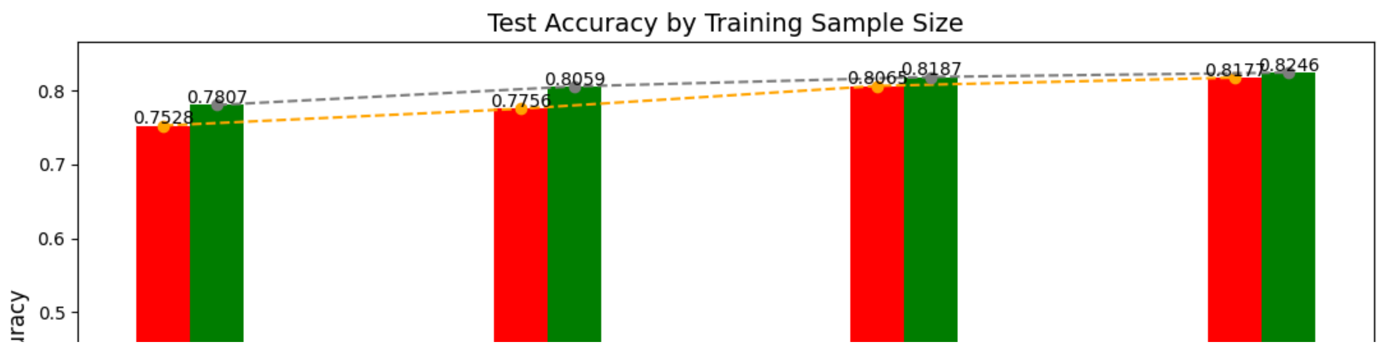
```
import numpy as np
import matplotlib.pyplot as plt

# Model labels and validation accuracies
model_labels = ['Custom Embedding', 'Pretrained Embedding']
final_accuracies = [
    training_history.history['val_accuracy'][-1],     # Last val accuracy from custom model
    glove_training_history.history['val_accuracy'][-1]  # Last val accuracy from pretrained model
]

# Create bar chart
plt.figure(figsize=(8, 6))
plt.bar(model_labels, final_accuracies, color=['black', 'orange'], width=0.5)

# Add chart labels and title
plt.ylabel('Validation Accuracy')
plt.title('Final Validation Accuracy Comparison')

# Annotate bars with accuracy values
for idx, accuracy in enumerate(final_accuracies):
    plt.text(idx, accuracy + 0.005, f'{accuracy:.4f}', ha='center', fontsize=12)

# Display chart
plt.tight_layout()
plt.show()
```



b) a pretrained word embedding.

```
import numpy as np
import matplotlib.pyplot as plt

# Input data
sample_counts = [100, 200, 500, 1000]
custom_scores = [0.75280, 0.77556, 0.80652, 0.81772]        # Custom embedding accuracy values
pretrained_scores = [0.78072, 0.80588, 0.81868, 0.82456]    # Pretrained embedding accuracy values

# Bar settings
bar_width = 0.15
x_indices = np.arange(len(sample_counts))  # X-axis positions

# Create figure
plt.figure(figsize=(11, 6))

# Plot bars for both models
plt.bar(x_indices - bar_width / 2, custom_scores,
        width=bar_width, label='Custom Embedding', color='red')
plt.bar(x_indices + bar_width / 2, pretrained_scores,
        width=bar_width, label='Pretrained Embedding', color='green')

# Overlay accuracy trend lines
plt.plot(x_indices - bar_width / 2, custom_scores,
         marker='o', color='orange', linestyle='--', label='Custom Embedding (Line)')
plt.plot(x_indices + bar_width / 2, pretrained_scores,
         marker='o', color='grey', linestyle='--', label='Pretrained Embedding (Line)')

# Add chart labels and title
plt.xlabel('Training Sample Size', fontsize=12)
plt.ylabel('Test Accuracy', fontsize=12)
plt.title('Test Accuracy by Training Sample Size', fontsize=14)
plt.xticks(x_indices, sample_counts)
plt.legend()

# Annotate bars with accuracy values
for j in range(len(sample_counts)):
    plt.text(x_indices[j] - bar_width / 2, custom_scores[j] + 0.002,
             f'{custom_scores[j]:.4f}', ha='center', fontsize=10)
    plt.text(x_indices[j] + bar_width / 2, pretrained_scores[j] + 0.002,
             f'{pretrained_scores[j]:.4f}', ha='center', fontsize=10)
```

```
# Final layout
plt.tight_layout()
plt.show()
```


Test Accuracy by Training Sample Size
uracy