```
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/python3.11/dist-packages (from tensorflow) (5.29.4)
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) (0.45.1)
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) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (3.4.1)
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) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (2025.1.31)
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~=2.19.0-
>tensorflow) (0.7.2)
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->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0-
>tensorflow) (3.0.0)
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) (2.19.1)
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->tensorflow) (0.1.2)
```

```python
import os
import pathlib
import shutil
import random

import tensorflow
```

```
from tensorflow import keras
from tensorflow.keras import layers

import numpy as np
import matplotlib.pyplot as plt

# Detailed shell scripts for setting up datasets
dataset_url =
"https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
archive_file = "aclImdb_v1.tar.gz"
unsup_folder = "aclImdb/train/unsup"

!curl {dataset_url} -O
!tar -xf {archive_file}
!rm -r {unsup_folder}
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100 80.2M  100 80.2M    0     0  18.8M      0  0:00:04  0:00:04
--:--:-- 18.8M
```

```
import os

def display_imdb_summary(dataset_path="aclImdb", sample_count=5):
    for data_split in ["train", "test"]:
        print(f"\nSummary of '{data_split}' split:")
        for label in ["pos", "neg"]:
            print(f"  Sentiment: {label}")
            dir_path = os.path.join(dataset_path, data_split, label)
            sample_files = os.listdir(dir_path)[:sample_count]
            for index, sample_name in enumerate(sample_files):
                sample_path = os.path.join(dir_path, sample_name)
                with open(sample_path, "r", encoding="utf-8") as file:
                    sample_lines = file.readlines()
                print(f"\n  File {index + 1}: {sample_name}")
                print(f"    Lines in file: {len(sample_lines)}")
                print(f"    First 5 lines (or fewer):")
                print("    " + "\n
".join(sample_lines[:5]).strip())

display_imdb_summary()
```

```
Summary of 'train' split:
  Sentiment: pos

  File 1: 7880_8.txt
    Lines in file: 1
```

```
    First 5 lines (or fewer):
    The plot of this enjoyable MGM musical is contrived and only
occasionally amusing, dealing with espionage and romance but the focus
of the film is properly pointed upon the tuneful interludes showcasing
the enormously talented and athletic tap dancing Eleanor Powell,
abetted by Tommy Dorsey and his orchestra, featuring Ziggy Elman,
Buddy Rich and Frank Sinatra. Red Skelton shares top billing with
Powell, and he and sidekick Bert Lahr are given most of the comedic
minutes, although Skelton is more effective when he, if it can be
believed, performs as Powell's love interest, with Virginia O'Brien
actually providing most of the film's humor as the dancer's companion.
The technical brilliance of Powell is evidenced during one incredible
scene within which Buddy Rich contributes his drumming skills, and
which must be viewed several times in order to permit one's breathing
to catch up with her precision. Director Edward Buzzell utilizes his
large cast well to move the action nicely along despite the rather
disjointed script with which he must deal, and permits Powell's
cotangent impossibilities to rule the affair, as is appropriate.

  File 2: 3815_7.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Corean cinema can be quite surprising for an occidental audience,
because of the multiplicity of the tones and genres you can find in
the same movie. In a Coreen drama such as this "Secret Sunshine",
you'll also find some comical parts, thriller scenes and romantic
times. "There's not only tragedy in life, there's also tragic-comedy"
says at one point of the movie the character interpreted by Song Kang-
ho, summing up the mixture of the picture. But don't get me wrong,
this heterogeneity of the genres the movie deals with, adds veracity
to the experience this rich movie offers to its spectators. That
doesn't mean that it lacks unity : on the contrary, it's rare to see
such a dense and profound portrait of a woman in pain.<br /><br
/>Shin-ae, who's in quest for a quiet life with her son in the native
town of her late husband, really gives, by all the different faces of
suffering she's going through, unity to this movie. It's realistic
part is erased by the psychological descriptions of all the phases the
poor mother is going through. Denial, lost, anger, faith, pert of
reality : the movie fallows all the steps the character crosses, and
looks like a psychological catalog of all the suffering phases a woman
can experience.<br /><br />The only thing is to accept what may look
like a conceptual experience (the woman wears the mask of tragedy, the
man represents the comical interludes) and to let the artifices of the
movie touch you. I must say that some parts of the movie really did
move me (especialy in the beginning), particularly those concerning
the unability of Chang Joan to truly help the one he loves, but also
that the accumulation of suffering emotionally tired me towards the
end. Nevertheless, some cinematographic ideas are really breathtaking
and surprising (the scene where a body is discovered in a large shot
```

```
is for instance amazing). This kind of scenes makes "Secret Sunshine"
the melo equivalent of "The Host" for horror movies or "Memories of
murder" for thrillers. These movies are indeed surprising, most
original, aesthetically incredible, and manage to give another
dimension to the genres they deal with. The only thing that "Secret
Sunshine" forgets, as "The host" forgot to be scary, is to make its
audience cry : bad point for a melodrama, but good point for a good
film.

  File 3: 7472_9.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I really liked this movie. I've read a few of the other comments,
and although I pity those who did not understand it, I do agree with
some of the criticisms. Which, in a strange way, makes me like this
movie all the more. I accept that they have got a pretty cast to
remake an intelligent movie for the general public, yet it has so many
levels and is still great to watch. I also love the movies, such as
this one, which provoke so many debates, theories, possible endings
and hidden subtext. Congratulations Mr.Crowe, definitely in my Top
Ten.<br /><br />P.S. Saw this when it first came out whilst I was
backpacking in Mexico, it was late at night and I had to get back to
my hotel and I had a major paranoia trip! Where does the dream end and
the real begin?

  File 4: 11969_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I loved this show growing up and I still watch the first season
DVD at age 19 today. What can I say? I grew up in a house much like
the one on Full House. I had a dad, two sisters, and a dog. I guess
the only difference was that I did not live with my uncle and my dad's
best friend. Also, I grew up with my mom in the house. I don't know
what I would have done without Full House on television. I think that
Stephanie (played by Jodie Sweetin), D.J. (played by Kirk Cameron's
sister Candace), and Michelle (Played by Mary-Kate and Ashley Olsen)
are my favorite characters. I can relate to each of them because I am
the middle child of my family like Steph, I am a younger sister like
Michelle, and I am an older sister like D.J. I really like how the
show always has moral values because I don't really like any of the
O.C.-like shows today. I like the comedy of Full House, too. Uncle
Jesse (John Stamos), Joey (Dave Coulier), and Danny (Bob Saget) are
hilarious as the girls' uncle, dad's friend, and dad, respectively.
The story goes that, after the girls' mom dies, Danny's best friend
Joey and his brother-in-law, Jesse move in to help raise the kids.
Three men trying to raise three young girls=hilarious. Each character
on Full House is full of heart, funny, and genuinely believable. Joey
is an aspiring comedian with a kid's heart and soul. Jesse is the
cool, motorcycle riding, tough-guy uncle who is softened by his three
```

```
nieces, and later, his wife Becky (Laurie Laughlin, from Summerville).
Both kids and adults will love this show. Guaranteed.

  File 5: 4182_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I saw this Australian film about 10 years ago and have never
forgotten it. The movie shows the horror of war in a way that
Hollywood usually glosses over. The relationship between the soldiers
of the two warring countries is highlighted by the differences in
culture and the ultimate knowledge that in the end we are all really
not different on the inside. If you can find any type of copy of
this--buy or rent it. You won't be disappointed, just awed.
  Sentiment: neg

  File 1: 2308_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    OK, the box looks promising. Whoopi Goldberg standing next to
Danny Glover parodying the famous farmer and his wife painting. Then
you pop this baby in the DVD player and all hope is lost in less then
five minutes. Supposed to be a comedy. And I must admit I did laugh
once about ten minutes before the ending. This movie has the following
elements: A battered and abused next door neighbor, a boring legal
trial, racisim, talk of lynchings, and death and arson. Hilarious,
huh? No, please, if you never listen to anyone's reviews, please do
here. You cannot even force yourself to watch this crap. CRAP! I said
it, CRAP! Whoever put there name on this should indeed sue.

  File 2: 10837_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I can't tell you how angry I was after seing this movie. The
characters are not the slightest bit interesting, and the plot is non-
existant. So after waiting to see how the lives of these characters
affected each other, hoping that the past 2 and a half hours were
leading up to some significant finish, what do we get??? A storm of
frogs. Now yes, I understand the references to the bible (Exodus) and
the underlying theme, but first of all, it was presented with
absolutely no resolution, and second of all it would be lost to anyone
who has not read the bible (a significant portion of the population)
or Charles Fort (a still larger portion). As a somewhat well read
person, I thought this movie was a self indulgent poor imitation of a
seinfeld episode.<br /><br />Don't waste your time. It would be better
spent reading...<br /><br />...well anything to be honest

  File 3: 890_2.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I saw this regurgitated pile of vignettes tonight at a preview
```

```
screening and I was straight up blown away by how bad it was. <br
/><br />First off, the film practically flaunted its gaping blind
spots. There are no black or gay New Yorkers in love? Or who, say,
know the self-involved white people in love? I know it's not the love
Crash of anvil-tastic inclusiveness but you can't pretend to have a
cinematic New York with out these fairly prevalent members of society.
Plus, you know the people who produced this ish thought Crash deserved
that ham-handed Oscar, so where is everyone? <br /><br />Possibly
worse than the bizarre and willful socioeconomic ignorance were the
down right offensive chapters (remember when you were in high school
and people were openly disgusted with pretty young women in
wheelchairs? Me either). This movie ran the gamut of ways to be the
worst. Bad acting, bad writing, bad directing -- all spanning every
possible genre ever to concern wealthy white people who smoke
cigarettes outside fancy restaurants. <br /><br />But thank god they
finally got powerhouses Hayden Christensen and Rachel Bilson back
together for that Jumper reunion. And, side note, Uma dodged a bullet;
Ethan Hawke looks ravaged. This, of course, is one thing in terms of
his looks, but added an incredibly creepy extra vibe of horribleness
to his terrifyingly scripted scene opposite poor, lovely Maggie
Q.<br /><br />I had a terrible time choosing my least favorite scene
for the end of film questionnaire, but it has to be the Anton Yelchin/
Olivia Thirlby bit for the sheer lack of taste, which saddens me
because I really like those two actors. I don't consider myself easily
offended, but all I could do was scoff and look around with disgust
like someone's 50 year old aunt. <br /><br />A close second place in
this incredibly tight contest of terrible things is Shia LaBeouf's
tone deaf portrayal of what it means for a former Disney Channel star
to act against Julie Christie. I don't mean opposite, I mean against.
Against is the only explanation. I realize now that the early sequence
with Orlando Bloom is a relative highlight. HIGHLIGHT. Please keep
that in mind when your brain begins to leak out your ear soon after
the opening credits, which seem to be a nod to the first New York Real
World. This film is embarrassing, strangely dated, inarticulate,
ineffective, pretentious and, in the end, completely divorced from any
real idea of New York at all. <br /><br />(The extra star is for the
Cloris Leachman/ Eli Wallach sequence, as it is actually quite sweet,
but it is only one bright spot in what feels like hours of pointless,
masturbatory torment.)

  File 4: 7974_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    It's interesting at first. A naive park ranger (Colin Firth)
marries a pretty, mysterious woman (Lisa Zane) he's only known for a
short time. They seem to be happy, then she disappears without
warning. He searches for her and, after a few dead ends, stumbles upon
some of her abused childhood and sleazy recent past, which may include
criminal activity. And then, it seems the filmmakers didn't know what
```

```
to do with the story. The beginning, while not as suspenseful as it
sounds, is at least watchable. Then it ceases to be interesting or
even make much sense. And the ending is so lame, so dull, and so
devoid of any excitement or intelligence, you'll think the
screenwriters didn't know what to do with it and got bored trying.
What a sorry waste of a good idea!

  File 5: 3619_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Basically there's a family where a little boy (Jake) thinks
there's a zombie in his closet & his parents are fighting all the
time.<br /><br />This movie is slower than a soap opera... and
suddenly, Jake decides to become Rambo and kill the zombie.<br
/><br />OK, first of all when you're going to make a film you must
Decide if its a thriller or a drama! As a drama the movie is
watchable. Parents are divorcing & arguing like in real life. And then
we have Jake with his closet which totally ruins all the film! I
expected to see a BOOGEYMAN similar movie, and instead i watched a
drama with some meaningless thriller spots.<br /><br />3 out of 10
just for the well playing parents & descent dialogs. As for the shots
with Jake: just ignore them.

Summary of 'test' split:
  Sentiment: pos

  File 1: 9554_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Watch out! This is not a gross out comedy like American Pie or you
know the comedies of the new millenium. This is a sweet satire of dog
shows focused on a bunch of completely different characters. Oh and
the commentator is hilarious!<br /><br />10/10

  File 2: 2027_8.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I seen this movie when it came out. I thought what an average
movie. I have now realized that this director was ahead of his time.
This is a great movie and great soundtrack. I have seen my share of
rock films but although this is far from spinal tap (which I did not
like)> This film does take us into the life of an 80s rocker wanting
to be nothing but. This is nothing more than our inner child wanting
to grow up and to be a *ROCK STAR* Yeah I said it. Everyone wants to
grow up and be on the spot light( Weather said or not). This movie
just puts you in the core of emotions and you can almost feel the
excitement of Izzy. I must admit the acting was less par but still the
music and story was enough to hold you in to it, till the credits
rolled. Worth the watch especially if you are a fan of ye Ole mighty
hair bands.
```

```
  File 3: 9194_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Like in "Les amants du Pont-Neuf" two outsiders lives a love story
without concessions. The film consists out of a lot of interesting
conversation and a lot of sweet moments. The best one comes in a
listening booth. They listen to a record together and once in a while
they look at each other. They talk, they like each other. She suggests
a change in their lives but he is out of hope. The realistic
stylestrokes over the realistic (but) emotive dialogs. A really
mathematic screenwriter's work for this film. Spanish novel director
Jesus Ponce creates one of the most perfect gallery from the latest
year of Spanish cinema.

  File 4: 7472_9.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I saw this movie yesterday. I must admit - I love it! It's like
early Tarantino, but better. Really a must, but... don't show this
movie to anyone younger than 18. It's full of blood and sex (rape
scene is great :) ). Now I'm just waiting for other movies of this
director and a DVD release.

  File 5: 11969_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Scary in places though the effects did leave something to be
desired unless you have bad eyesight or are afraid of the dark.
However most of the acting was convincing and most of the effects were
well done. I thought the creature looked a bit too much like a man in
a gorilla suit for my liking. It reminded me of the original pink
panther film.
  Sentiment: neg

  File 1: 4050_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Skip all the subjective "this is a great film" reviews and read
the IMDB trailer or the back the KINO videobox (which includes both
versions of this flick) which I'll paraphrase: "To the tune of sci-fi
score by George Antheil, the camera goes on a sleepwalk through B-
Movie hell, all photographed by Will Thompson (who did 'Plan 9 from
outer space' & 'Maniac')." You don't know whether to laugh AT the film
or WITH it. So if you like self-produced B or C-grade noir-wannabe
actors and effects with pretensions of surrealism, this could be for
you! Otherwise, get a copy of "Screamplay", a modern low-budget
expressionist masterpiece.

  File 2: 10837_1.txt
```

```
    Lines in file: 1
    First 5 lines (or fewer):
    Possibly the worst movie I have ever seen. Pathetic in almost
every way.<br /><br />I threw the DVD straight in the bin - I didn't
even think it was fair to give it to the local thrift shop.<br
/><br />The effects are beyond a joke. The dam control room looks like
cardboard. The water looks way out of scale with the backgrounds -
nothing works.<br /><br />Then there is the limp plot - about as much
depth as a Scooby Doo cartoon.<br /><br />I couldn't wait for them all
to drown.

  File 3: 8906_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I got hold of this film on DVD with the title Evil Never Sleeps,
it gives front cover billing to Carrie Ann Moss, but she plays such a
minor character that I didn't really notice her in the film.<br
/><br />I'm afraid that I consider this one of the worst purchases I
have ever made. The dialogue was stilted and the delivery wooden, I
found the acting to be disconnected from the plot. Graham's
performance to me was of someone who's wondering whether she's left
the gas on at home.<br /><br />All in all both my wife and I found
this film painful to watch, and it is not a valuable addition to my
collection, watch it at your peril, but spending 90 minutes having
your fingernails pulled out would probably be a better way to spend
your time.

  File 4: 8644_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    William Petersen (that C.S.I guy) has a small uncredited role but
it's the best part of the movie. His character comes across smart ass
and tough, and it's a fun surprise to see him in this. He has a range
that allows him to play just about anything. After his 5 minutes, it
goes from looking cool to just nothing much. It leaves you hoping that
his character will reappear in the movie but after 20 minutes you give
up hope. The movie itself is pretty poor. Worth a watch on TMN or a
pick up at the library but not much more. Too much of it reminds you
of L.A Confidential except that where that movie starts to get
complicated upon itself, this one is so loose, it steers everywhere
but where it should. 2 out of 5 stars

  File 5: 5276_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    The German regional-broadcast-station WDR has shown both "The
General" and ODC. On Saturday I've seen "The General" and I thought,
it wasn't very bad, but not very good too. But yesterday I've seen ODC
and I switched it off after about an hour. Although Kevin Spacey was
the main actor the movie was totally confusing and seems restless.
```

```
"The General" told the story straight and ordered, but ODC just wanted
to be cool. There is a reference on the Guy Ritchie Movies "Lock,
Stock and Two Smoking Barrels" and "Snatch", but doesn't have the
Coolness of these movies.<br /><br />So, in the end I would rate it 3
of 10!
```

```
# Get the directories ready for the split validation
batch_size = 32
base_path = pathlib.Path("aclImdb")
validation_path = base_path / "val"
training_path = base_path / "train"

for label in ("neg", "pos"):
    os.makedirs(validation_path / label, exist_ok=True)
    all_files = os.listdir(training_path / label)
    random.Random(1337).shuffle(all_files)
    val_count = int(0.2 * len(all_files))
    validation_files = all_files[-val_count:]

    for file in validation_files:
        source_path = training_path / label / file
        target_path = validation_path / label / file
        if not os.path.exists(target_path):
            shutil.move(source_path, target_path)

# Loading the datasets from the directories
training_dataset = keras.utils.text_dataset_from_directory(
    base_path / "train", batch_size=batch_size
)

validation_dataset = keras.utils.text_dataset_from_directory(
    base_path / "val", batch_size=batch_size
)

test_dataset = keras.utils.text_dataset_from_directory(
    base_path / "test", batch_size=batch_size
)

# Creating a dataset containing only text (without any labels)
text_only_training = training_dataset.map(lambda x, y: x)
```

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

```
# Setting the vectorization parameters
sequence_length = 150
vocab_size = 10000

text_vectorizer = layers.TextVectorization(
    max_tokens=vocab_size,
```

```
    output_mode="int",
    output_sequence_length=sequence_length,
)

# Using the training text to adjust the vectorizer
text_vectorizer.adapt(text_only_training)

# Use the text vectorizer to tokenize datasets
tokenized_train = training_dataset.map(
    lambda x, y: (text_vectorizer(x), y),
    num_parallel_calls=4).take(100)  # Restricting the training
samples to 100

tokenized_val = validation_dataset.map(
    lambda x, y: (text_vectorizer(x), y),
    num_parallel_calls=4).take(10000)  # Restricting the validation
samples to 10,000

tokenized_test = test_dataset.map(
    lambda x, y: (text_vectorizer(x), y),
    num_parallel_calls=4)

# Establish a simple text classification model with LSTM and embedding
layers
input_layer = keras.Input(shape=(None,), dtype="int64")
embedding_output = layers.Embedding(input_dim=vocab_size,
output_dim=128)(input_layer)
lstm_output = layers.Bidirectional(layers.LSTM(32))(embedding_output)
dropout_output = layers.Dropout(rate=0.3)(lstm_output)
final_output = layers.Dense(1, activation="sigmoid")(dropout_output)

sentiment_model = keras.Model(inputs=input_layer,
outputs=final_output)

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

sentiment_model.summary()
```

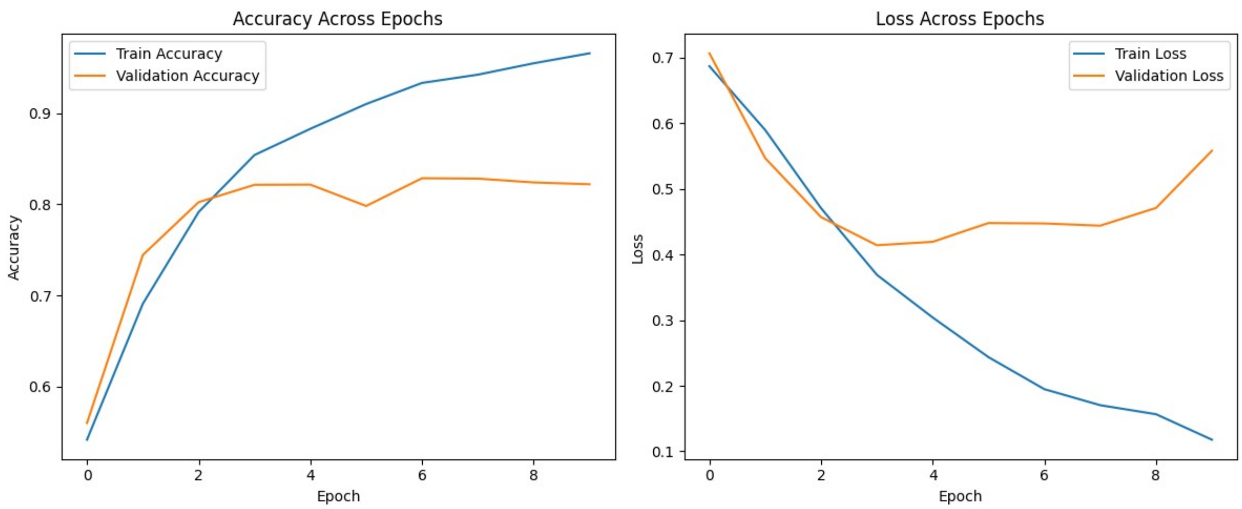
```
Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ 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)

 Non-trainable params: 0 (0.00 B)
```

```
# Establish a checkpoint to preserve the model's peak performance
callbacks = [

keras.callbacks.ModelCheckpoint(filepath="embedding_model.keras",save_
best_only=True)
]

# Create the model and save the training data for further examination
history_embedded = sentiment_model.fit(
    tokenized_train,
    validation_data=tokenized_val,
    epochs=10,
    callbacks=callbacks
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 91ms/step - accuracy: 0.5221 - loss:
0.6926 - val_accuracy: 0.5602 - val_loss: 0.7068
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 88ms/step - accuracy: 0.6584 - loss:
0.6201 - val_accuracy: 0.7444 - val_loss: 0.5469
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.7822 - loss:
0.4927 - val_accuracy: 0.8024 - val_loss: 0.4571
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.8497 - loss:
```

```
0.3820 - val_accuracy: 0.8214 - val_loss: 0.4142
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 88ms/step - accuracy: 0.8841 - loss:
0.3054 - val_accuracy: 0.8216 - val_loss: 0.4195
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.9095 - loss:
0.2499 - val_accuracy: 0.7982 - val_loss: 0.4481
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.9353 - loss:
0.1967 - val_accuracy: 0.8286 - val_loss: 0.4474
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9378 - loss:
0.1772 - val_accuracy: 0.8282 - val_loss: 0.4440
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9543 - loss:
0.1520 - val_accuracy: 0.8240 - val_loss: 0.4711
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9685 - loss:
0.1058 - val_accuracy: 0.8220 - val_loss: 0.5582

# Retrieve metrics from the model's training process
training_stats = history_embedded.history

# Preparing the canvas for two plots adjacent to one other
plt.figure(figsize=(12, 5))

# Accuracy Plotting for Training and Validation
plt.subplot(1, 2, 1)
plt.plot(training_stats['accuracy'], label='Train Accuracy')
plt.plot(training_stats['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Across Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plotting Loss for Training and Validation
plt.subplot(1, 2, 2)
plt.plot(training_stats['loss'], label='Train Loss')
plt.plot(training_stats['val_loss'], label='Validation Loss')
plt.title('Loss Across Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Presenting everything clean and neatly
plt.tight_layout()
plt.show()
```




```
# Retrieve pre-trained GloVe word vectors from the NLP website at
Stanford
!wget http://nlp.stanford.edu/data/glove.6B.zip

# Silently extract the downloaded archive
!unzip -q glove.6B.zip
```

```
--2025-04-08 15:23:12--  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]
--2025-04-08 15:23:12--  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]
--2025-04-08 15:23:12--
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.3’

glove.6B.zip.3      100%[===================>] 822.24M  5.02MB/s    in
2m 39s

2025-04-08 15:25:52 (5.16 MB/s) - ‘glove.6B.zip.3’ saved
```

```
[862182613/862182613]

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

```
# Use pre-trained GloVe embeddings to set up the matrix
glove_dim = 100
glove_path = "glove.6B.100d.txt"

# The GloVe vectors are loaded into a dictionary
glove_index = {}
with open(glove_path, "r", encoding="utf8") as file:
    for line in file:
        token, vector = line.strip().split(maxsplit=1)
        glove_index[token] = np.fromstring(vector, dtype="f", sep=" ")

# Create an embedding matrix that matches the language of our model
vocab_list = text_vectorizer.get_vocabulary()
vocab_lookup = {word: idx for idx, word in enumerate(vocab_list)}

embedding_matrix = np.zeros((vocab_size, glove_dim))
for word, idx in vocab_lookup.items():
    if idx < vocab_size:
        vector = glove_index.get(word)
        if vector is not None:
            embedding_matrix[idx] = vector

# Use preloaded GloVe weights (frozen) to create an embedding layer
glove_embedding = layers.Embedding(
    input_dim=vocab_size,
    output_dim=glove_dim,

embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False, # When training, keep the GloVe weights fixed
    mask_zero=True
)

# Use the pretrained embeddings to define the model structure
input_seq = keras.Input(shape=(None,), dtype="int64")
embedded_seq = glove_embedding(input_seq)
lstm_out = layers.Bidirectional(layers.LSTM(32))(embedded_seq)
dropout_out = layers.Dropout(0.3)(lstm_out)
final_output = layers.Dense(1, activation="sigmoid")(dropout_out)

glove_model = keras.Model(inputs=input_seq, outputs=final_output)

# Use pre-trained GloVe embeddings to construct and assemble the
sentiment model
input_layer = keras.Input(shape=(None,), dtype="int64")
embedded_output = glove_embedding(input_layer)
lstm_output = layers.Bidirectional(layers.LSTM(32))(embedded_output)
dropout_layer = layers.Dropout(0.3)(lstm_output)
```

```python
prediction = layers.Dense(1, activation="sigmoid")(dropout_layer)

pretrained_model = keras.Model(inputs=input_layer, outputs=prediction)

# Using binary classification parameters to compile the model
pretrained_model.compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

# Printing the model's structure
pretrained_model.summary()
```

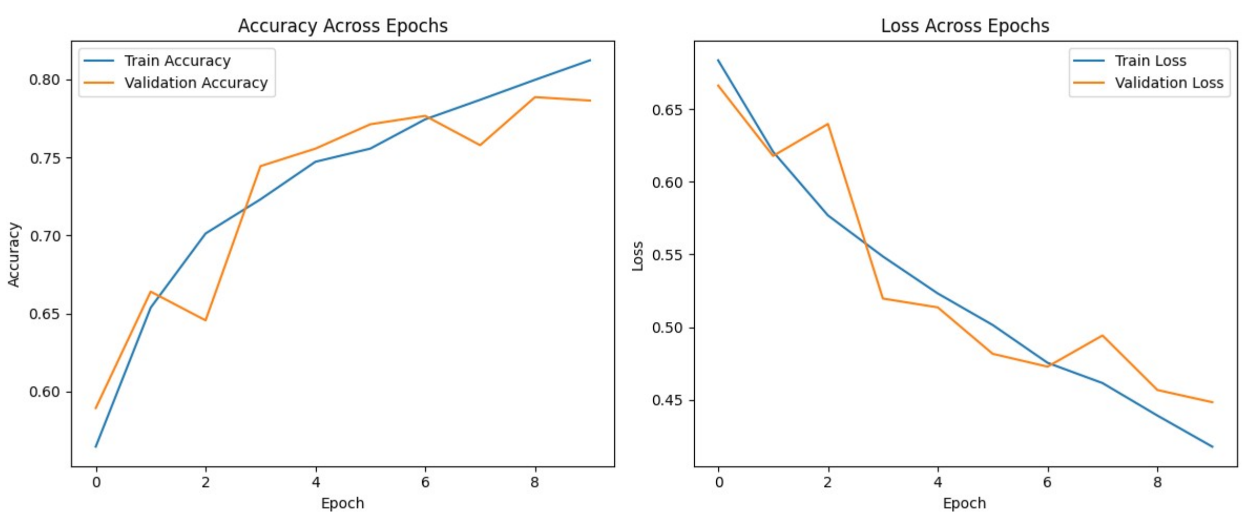
```
Model: "functional_5"
```

Layer (type) Connected to	Output Shape	Param #
input_layer_5 - (InputLayer)	(None, None)	0
embedding_3 (Embedding) input_layer_5[0][0]	(None, None, 100)	1,000,000
not_equal_3 (NotEqual) input_layer_5[0][0]	(None, None)	0
bidirectional_5 embedding_3[1][0], (Bidirectional) not_equal_3[0][0]	(None, 64)	34,048
dropout_5 (Dropout) bidirectional_5[0][0]	(None, 64)	0
dense_5 (Dense) dropout_5[0][0]	(None, 1)	65

```
 Total params: 1,034,113 (3.94 MB)

 Trainable params: 34,113 (133.25 KB)

 Non-trainable params: 1,000,000 (3.81 MB)

# Configure a callback to preserve the optimal GloVe-based model
version
glove_callbacks = [
    keras.callbacks.ModelCheckpoint("pretrained_model.keras",
save_best_only=True)
]

# Use the tokenized data to train the model
history_glove = pretrained_model.fit(
    tokenized_train,
    validation_data=tokenized_val,
    epochs=10,
    callbacks=glove_callbacks
)

Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 16s 129ms/step - accuracy: 0.5537 - loss:
0.6958 - val_accuracy: 0.5894 - val_loss: 0.6661
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 124ms/step - accuracy: 0.6172 - loss:
0.6431 - val_accuracy: 0.6640 - val_loss: 0.6177
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 110ms/step - accuracy: 0.6790 - loss:
0.5965 - val_accuracy: 0.6456 - val_loss: 0.6397
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 126ms/step - accuracy: 0.7028 - loss:
0.5684 - val_accuracy: 0.7444 - val_loss: 0.5197
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 124ms/step - accuracy: 0.7392 - loss:
0.5337 - val_accuracy: 0.7556 - val_loss: 0.5136
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 120ms/step - accuracy: 0.7439 - loss:
0.5133 - val_accuracy: 0.7712 - val_loss: 0.4817
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 127ms/step - accuracy: 0.7576 - loss:
0.4942 - val_accuracy: 0.7766 - val_loss: 0.4729
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 110ms/step - accuracy: 0.7706 - loss:
0.4931 - val_accuracy: 0.7578 - val_loss: 0.4943
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 122ms/step - accuracy: 0.7976 - loss:
0.4509 - val_accuracy: 0.7886 - val_loss: 0.4568
Epoch 10/10
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 118ms/step - accuracy: 0.8091 - loss:
0.4228 - val_accuracy: 0.7864 - val_loss: 0.4484
```

```python
import matplotlib.pyplot as plt

# Extracting the training history from GloVe-based model
metrics = history_glove.history

# Setting up a wide figure with two subplots
plt.figure(figsize=(12, 5))

# Plot accuracy: training vs validation
plt.subplot(1, 2, 1)
plt.plot(metrics['accuracy'], label='Train Accuracy')
plt.plot(metrics['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Across Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss: training vs validation
plt.subplot(1, 2, 2)
plt.plot(metrics['loss'], label='Train Loss')
plt.plot(metrics['val_loss'], label='Validation Loss')
plt.title('Loss Across Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Showing both the plots
plt.tight_layout()
plt.show()
```



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

# Defining the sample counts to use for training
sample_counts = [100, 200, 500, 1000]
custom_acc_list = []
glove_acc_list = []

# Setting up the graph
plt.figure(figsize=(12, 6))
plt.title('Accuracy vs Number of Training Samples')
plt.xlabel('Sample Count')
plt.ylabel('Accuracy')
plt.grid(True)

# Loop through various sample sizes
for i, count in enumerate(sample_counts):
    print(f"\n### Training with {count} samples ###\n")

    # Restricting the training set size
    # Replacing 'train_ds' with 'training_dataset' and
'text_vectorization' with 'text_vectorizer'
    limited_train_data = training_dataset.map(
        lambda x, y: (text_vectorizer(x), y)).take(count)

    # Training model with custom embeddings
    print(f"Training Custom Embedding Model on {count} samples:")
    sentiment_model.fit( # Replacing 'embedding_model' with
'sentiment_model'
        limited_train_data,
        validation_data=tokenized_val, # Replacing 'int_val_ds' with
'tokenized_val'
        epochs=10,
        verbose=1
    )
    # Replacing 'embedding_model' with 'sentiment_model' and
'int_test_ds' with 'tokenized_test'
    acc_custom = sentiment_model.evaluate(tokenized_test, verbose=1)
[1]
    custom_acc_list.append(acc_custom)
    print(f"Custom Embedding Model Accuracy: {acc_custom:.4f}\n")

    # Training the model with pretrained GloVe embeddings
    print(f"Training Pretrained Embedding Model on {count} samples:")
    pretrained_model.fit(
        limited_train_data,
        validation_data=tokenized_val, # Replacing 'int_val_ds' with
'tokenized_val'
        epochs=10,
```

```
        verbose=1
    )
    # Replacing 'int_test_ds' with 'tokenized_test'
    acc_glove = pretrained_model.evaluate(tokenized_test, verbose=1)
[1]
    glove_acc_list.append(acc_glove)
    print(f"Pretrained Embedding Model Accuracy: {acc_glove:.4f}\n")

    # Plotting once all sample sizes are doen running
    if i == len(sample_counts) - 1:
        plt.plot(sample_counts, custom_acc_list, marker='o',
label='Custom Embedding', color='blue')
        plt.plot(sample_counts, glove_acc_list, marker='o',
label='Pretrained Embedding', color='orange')

# Final designing of the chart
plt.title('Accuracy vs Number of Training Samples')
plt.xlabel('Sample Count')
plt.ylabel('Accuracy')
plt.xticks(sample_counts)
plt.grid(True)
plt.legend()

# Displaying the chart
plt.tight_layout()
plt.show()


### Training with 100 samples ###

Training Custom Embedding Model on 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9605 - loss:
0.1176 - val_accuracy: 0.8344 - val_loss: 0.5587
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9831 - loss:
0.0663 - val_accuracy: 0.7908 - val_loss: 0.5306
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9757 - loss:
0.0799 - val_accuracy: 0.8058 - val_loss: 0.5644
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9797 - loss:
0.0816 - val_accuracy: 0.8138 - val_loss: 0.6496
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.9952 - loss:
0.0265 - val_accuracy: 0.8258 - val_loss: 0.6288
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9854 - loss:
0.0489 - val_accuracy: 0.8308 - val_loss: 0.7016
Epoch 7/10
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.9977 - loss:
0.0143 - val_accuracy: 0.8126 - val_loss: 0.6292
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9940 - loss:
0.0284 - val_accuracy: 0.8070 - val_loss: 0.6300
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9974 - loss:
0.0124 - val_accuracy: 0.6648 - val_loss: 1.7311
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.9778 - loss:
0.0673 - val_accuracy: 0.8170 - val_loss: 0.8740
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step - accuracy: 0.7722 - loss:
1.1256
Custom Embedding Model Accuracy: 0.7732

Training Pretrained Embedding Model on 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 110ms/step - accuracy: 0.8149 - loss:
0.4125 - val_accuracy: 0.8026 - val_loss: 0.4364
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8310 - loss:
0.3836 - val_accuracy: 0.7816 - val_loss: 0.4783
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 102ms/step - accuracy: 0.8264 - loss:
0.3850 - val_accuracy: 0.7856 - val_loss: 0.4604
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 109ms/step - accuracy: 0.8385 - loss:
0.3746 - val_accuracy: 0.8086 - val_loss: 0.4367
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8457 - loss:
0.3411 - val_accuracy: 0.7716 - val_loss: 0.4841
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 106ms/step - accuracy: 0.8533 - loss:
0.3330 - val_accuracy: 0.8096 - val_loss: 0.4342
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 109ms/step - accuracy: 0.8762 - loss:
0.3046 - val_accuracy: 0.7984 - val_loss: 0.4629
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 104ms/step - accuracy: 0.8795 - loss:
0.2841 - val_accuracy: 0.8188 - val_loss: 0.4445
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.8898 - loss:
0.2796 - val_accuracy: 0.8196 - val_loss: 0.4384
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 108ms/step - accuracy: 0.9020 - loss:
0.2567 - val_accuracy: 0.8200 - val_loss: 0.4386
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.7851 - loss:
0.5020
Pretrained Embedding Model Accuracy: 0.7862
```

```
### Training with 200 samples ###

Training Custom Embedding Model on 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 72ms/step - accuracy: 0.9581 - loss:
0.1107 - val_accuracy: 0.8378 - val_loss: 0.3845
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 73ms/step - accuracy: 0.9600 - loss:
0.1202 - val_accuracy: 0.8594 - val_loss: 0.3630
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 74ms/step - accuracy: 0.9723 - loss:
0.0923 - val_accuracy: 0.8104 - val_loss: 0.4578
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9774 - loss:
0.0712 - val_accuracy: 0.8454 - val_loss: 0.3930
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9842 - loss:
0.0586 - val_accuracy: 0.8618 - val_loss: 0.4157
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 73ms/step - accuracy: 0.9890 - loss:
0.0368 - val_accuracy: 0.8678 - val_loss: 0.4119
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9918 - loss:
0.0344 - val_accuracy: 0.8442 - val_loss: 0.5465
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9942 - loss:
0.0234 - val_accuracy: 0.8548 - val_loss: 0.4650
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 73ms/step - accuracy: 0.9929 - loss:
0.0272 - val_accuracy: 0.8630 - val_loss: 0.5182
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 73ms/step - accuracy: 0.9959 - loss:
0.0145 - val_accuracy: 0.8064 - val_loss: 0.9331
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.7154 - loss:
1.4618
Custom Embedding Model Accuracy: 0.7169

Training Pretrained Embedding Model on 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 92ms/step - accuracy: 0.8846 - loss:
0.2764 - val_accuracy: 0.8146 - val_loss: 0.4202
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.8903 - loss:
0.2815 - val_accuracy: 0.8158 - val_loss: 0.4187
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 92ms/step - accuracy: 0.8992 - loss:
0.2687 - val_accuracy: 0.7802 - val_loss: 0.4642
Epoch 4/10
```

```
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.9002 - loss:
0.2532 - val_accuracy: 0.8334 - val_loss: 0.3851
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9118 - loss:
0.2364 - val_accuracy: 0.8340 - val_loss: 0.3855
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.9203 - loss:
0.2141 - val_accuracy: 0.8300 - val_loss: 0.3937
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.9237 - loss:
0.2073 - val_accuracy: 0.8266 - val_loss: 0.4063
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.9322 - loss:
0.1900 - val_accuracy: 0.8266 - val_loss: 0.4018
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 96ms/step - accuracy: 0.9378 - loss:
0.1730 - val_accuracy: 0.8292 - val_loss: 0.4074
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9410 - loss:
0.1605 - val_accuracy: 0.8322 - val_loss: 0.4040
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.7891 - loss:
0.4917
Pretrained Embedding Model Accuracy: 0.7894


### Training with 500 samples ###

Training Custom Embedding Model on 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9501 - loss:
0.1334 - val_accuracy: 0.8994 - val_loss: 0.2711
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9584 - loss:
0.1199 - val_accuracy: 0.9062 - val_loss: 0.2517
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 66ms/step - accuracy: 0.9680 - loss:
0.0919 - val_accuracy: 0.9258 - val_loss: 0.1993
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9810 - loss:
0.0639 - val_accuracy: 0.9344 - val_loss: 0.2010
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 66ms/step - accuracy: 0.9848 - loss:
0.0473 - val_accuracy: 0.9178 - val_loss: 0.2470
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9910 - loss:
0.0321 - val_accuracy: 0.9360 - val_loss: 0.2263
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9943 - loss:
0.0212 - val_accuracy: 0.9300 - val_loss: 0.2450
Epoch 8/10
```

```
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 65ms/step - accuracy: 0.9948 - loss:
0.0187 - val_accuracy: 0.8532 - val_loss: 0.7217
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9956 - loss:
0.0158 - val_accuracy: 0.9394 - val_loss: 0.2903
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 33s 65ms/step - accuracy: 0.9974 - loss:
0.0092 - val_accuracy: 0.9304 - val_loss: 0.2946
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step - accuracy: 0.7912 - loss:
0.9677
Custom Embedding Model Accuracy: 0.7924

Training Pretrained Embedding Model on 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9037 - loss:
0.2363 - val_accuracy: 0.8390 - val_loss: 0.3714
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 83ms/step - accuracy: 0.8955 - loss:
0.2557 - val_accuracy: 0.8506 - val_loss: 0.3417
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.8993 - loss:
0.2466 - val_accuracy: 0.8532 - val_loss: 0.3343
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9073 - loss:
0.2327 - val_accuracy: 0.8588 - val_loss: 0.3235
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 43s 85ms/step - accuracy: 0.9161 - loss:
0.2144 - val_accuracy: 0.8620 - val_loss: 0.3180
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9229 - loss:
0.2006 - val_accuracy: 0.8684 - val_loss: 0.3087
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.9266 - loss:
0.1914 - val_accuracy: 0.8742 - val_loss: 0.2932
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 43s 85ms/step - accuracy: 0.9315 - loss:
0.1796 - val_accuracy: 0.8704 - val_loss: 0.3075
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9376 - loss:
0.1680 - val_accuracy: 0.8854 - val_loss: 0.2767
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9395 - loss:
0.1546 - val_accuracy: 0.8854 - val_loss: 0.2773
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.8221 - loss:
0.4415
Pretrained Embedding Model Accuracy: 0.8230


### Training with 1000 samples ###
```

```
Training Custom Embedding Model on 1000 samples:
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 50s 64ms/step - accuracy: 0.9841 - loss:
0.0497 - val_accuracy: 0.9410 - val_loss: 0.1570
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 50s 63ms/step - accuracy: 0.9811 - loss:
0.0567 - val_accuracy: 0.9650 - val_loss: 0.1115
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 50s 64ms/step - accuracy: 0.9887 - loss:
0.0365 - val_accuracy: 0.9906 - val_loss: 0.0419
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.9923 - loss:
0.0234 - val_accuracy: 0.9822 - val_loss: 0.0518
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.9946 - loss:
0.0176 - val_accuracy: 0.9858 - val_loss: 0.0410
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.9964 - loss:
0.0121 - val_accuracy: 0.9950 - val_loss: 0.0163
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.9979 - loss:
0.0074 - val_accuracy: 0.9946 - val_loss: 0.0194
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.9981 - loss:
0.0073 - val_accuracy: 0.9944 - val_loss: 0.0204
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.9981 - loss:
0.0074 - val_accuracy: 0.9962 - val_loss: 0.0125
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 62ms/step - accuracy: 0.9989 - loss:
0.0045 - val_accuracy: 0.9976 - val_loss: 0.0076
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.8102 - loss:
1.3481
Custom Embedding Model Accuracy: 0.8109

Training Pretrained Embedding Model on 1000 samples:
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 61s 78ms/step - accuracy: 0.9288 - loss:
0.1823 - val_accuracy: 0.8740 - val_loss: 0.2916
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 64s 81ms/step - accuracy: 0.9202 - loss:
0.1988 - val_accuracy: 0.8522 - val_loss: 0.3268
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 64s 82ms/step - accuracy: 0.9292 - loss:
0.1867 - val_accuracy: 0.8984 - val_loss: 0.2487
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 64s 82ms/step - accuracy: 0.9304 - loss:
0.1739 - val_accuracy: 0.8974 - val_loss: 0.2356
Epoch 5/10
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 64s 81ms/step - accuracy: 0.9356 - loss: 0.1641 - val_accuracy: 0.8504 - val_loss: 0.3255
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 63s 81ms/step - accuracy: 0.9382 - loss: 0.1585 - val_accuracy: 0.9070 - val_loss: 0.2208
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 63s 81ms/step - accuracy: 0.9447 - loss: 0.1472 - val_accuracy: 0.9138 - val_loss: 0.2127
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 64s 82ms/step - accuracy: 0.9506 - loss: 0.1348 - val_accuracy: 0.9152 - val_loss: 0.2067
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 62s 79ms/step - accuracy: 0.9487 - loss: 0.1334 - val_accuracy: 0.9160 - val_loss: 0.1979
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 62s 79ms/step - accuracy: 0.9525 - loss: 0.1264 - val_accuracy: 0.9330 - val_loss: 0.1720
782/782 ━━━━━━━━━━━━━━━━━━━━ 15s 19ms/step - accuracy: 0.8292 - loss: 0.4327
Pretrained Embedding Model Accuracy: 0.8294
```

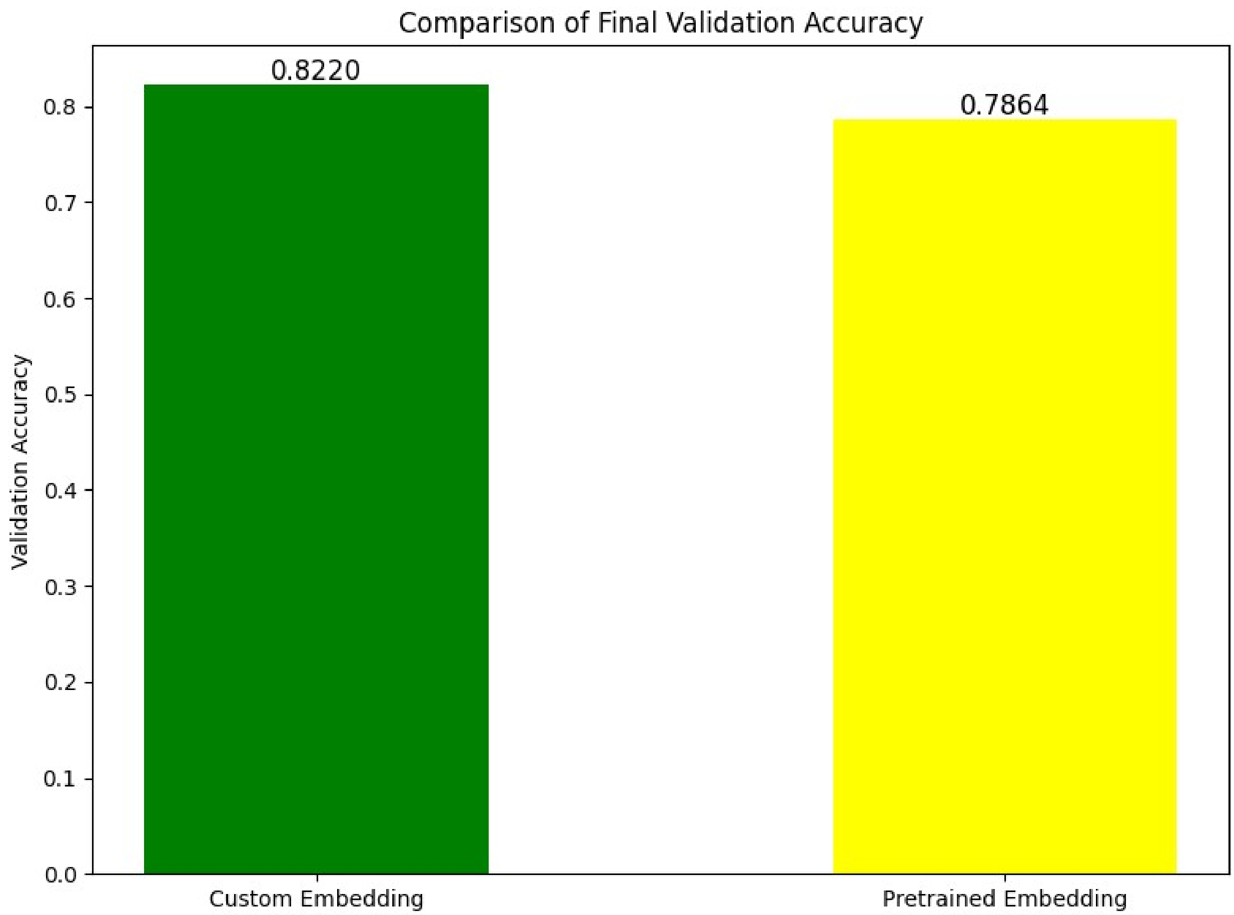

```python
import pandas as pd

# Save the self-trained embedding model's findings
custom_results = {
    "Data Count": sample_counts,
    "Accuracy (Custom Embedding)": custom_acc_list,
}
```

```python
# Save the GloVe model's pretrained findings
pretrained_results = {
    "Data Count": sample_counts,
    "Accuracy (Pretrained Embedding)": glove_acc_list,
}

# Combine both sets of results into a single DataFrame
results_table = pd.DataFrame({
    "Data Count": sample_counts,
    "Accuracy (Custom Embedding)": custom_acc_list,
    "Accuracy (Pretrained Embedding)": glove_acc_list
})

# Printing the final comparison table
print("Final Comparison Summary:")
print(results_table)
```

```
Final Comparison Summary:
   Data Count  Accuracy (Custom Embedding)  Accuracy (Pretrained 
Embedding)
0         100                      0.77316
0.78616
1         200                      0.71692
0.78936
2         500                      0.79244
0.82304
3        1000                      0.81092
0.82944
```

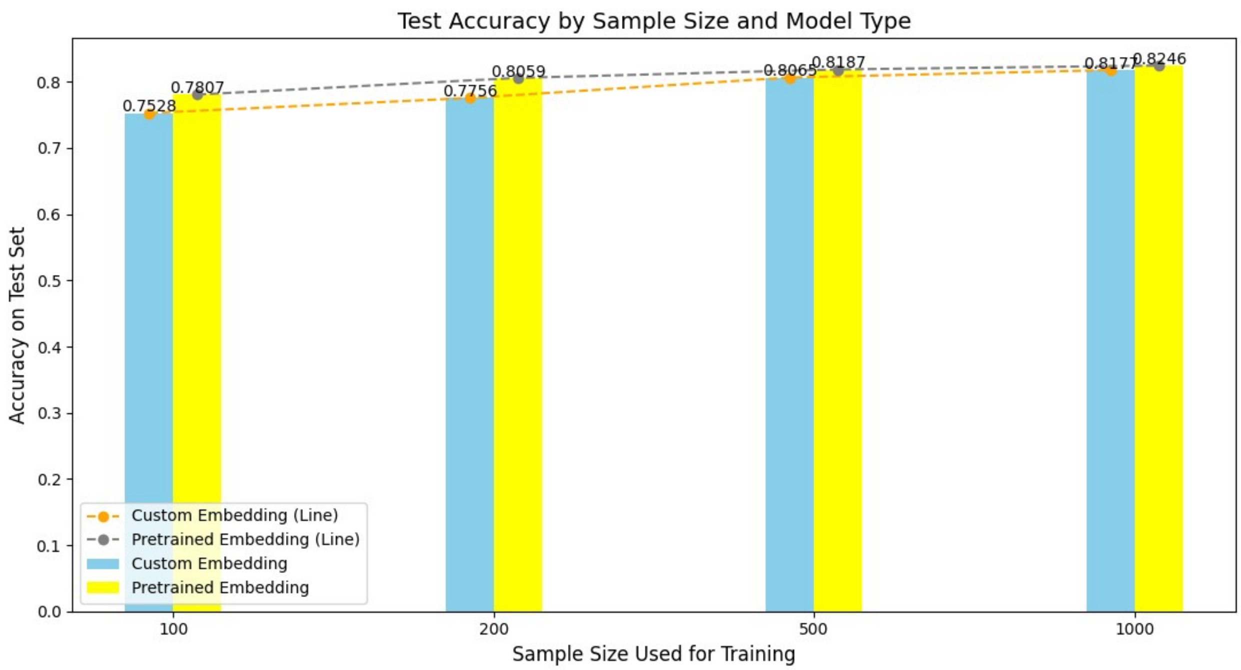
```python
import numpy as np
import matplotlib.pyplot as plt

# Defining the model types
model_labels = ['Custom Embedding', 'Pretrained Embedding']
accuracy_scores = [
    history_embedded.history['val_accuracy'][-1],   # Last validation 
accuracy for custom model
    history_glove.history['val_accuracy'][-1]       # Last validation 
accuracy for pretrained model
]

# Creating the bar chart
plt.figure(figsize=(8, 6))
plt.bar(model_labels, accuracy_scores, color=['green', 'yellow'], 
width=0.5)

# Adding axis labels and chart titles
plt.ylabel('Validation Accuracy')
plt.title('Comparison of Final Validation Accuracy')
```

```python
# Displaying values above each bar
for idx, score in enumerate(accuracy_scores):
    plt.text(idx, score + 0.005, f'{score:.4f}', ha='center',
fontsize=12)

# Rendering the final chart
plt.tight_layout()
plt.show()
```




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

# Training the sample counts
train_sizes = [100, 200, 500, 1000]
custom_model_scores = [0.75280, 0.77556, 0.80652, 0.81772]      #
Custom model accuracy
glove_model_scores = [0.78072, 0.80588, 0.81868, 0.82456]       #
Pretrained model accuracy

# Bar positioning setting up
bar_width = 0.15
```

```
index_positions = np.arange(len(train_sizes))  # Bar centers

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

# Plotting the side-by-side bars
plt.bar(index_positions - bar_width / 2, custom_model_scores,
        width=bar_width, label='Custom Embedding', color='skyblue')
plt.bar(index_positions + bar_width / 2, glove_model_scores,
        width=bar_width, label='Pretrained Embedding', color='yellow')

# Adding the line overlays for visual tracking
plt.plot(index_positions - bar_width / 2, custom_model_scores,
         marker='o', color='orange', linestyle='--', label='Custom
Embedding (Line)')
plt.plot(index_positions + bar_width / 2, glove_model_scores,
         marker='o', color='grey', linestyle='--', label='Pretrained
Embedding (Line)')

# labels and title of the charts
plt.xlabel('Sample Size Used for Training', fontsize=12)
plt.ylabel('Accuracy on Test Set', fontsize=12)
plt.title('Test Accuracy by Sample Size and Model Type', fontsize=14)
plt.xticks(index_positions, train_sizes)
plt.legend()

# Annotation of each bar with its accuracy
for i in range(len(train_sizes)):
    plt.text(index_positions[i] - bar_width / 2,
custom_model_scores[i] + 0.002,
             f'{custom_model_scores[i]:.4f}', ha='center',
fontsize=10)
    plt.text(index_positions[i] + bar_width / 2, glove_model_scores[i]
+ 0.002,
             f'{glove_model_scores[i]:.4f}', ha='center', fontsize=10)

# Final display of the charts
plt.tight_layout()
plt.show()
```


Test Accuracy by Sample Size and Model Type
Accuracy on Test Set
Sample Size Used for Training
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
100
200
500
1000
0.7528
0.7807
0.7756
0.8059
0.8065
0.8187
0.8177
0.8246
Custom Embedding (Line)
Pretrained Embedding (Line)
Custom Embedding
Pretrained Embedding