```
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
#Importing required libraries
import os
import shutil
import random
from pathlib import Path
```

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

# Importing TensorFlow and Keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# 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  17.3M      0  0:00:04  0:00:04
--:--:-- 17.3M
```

```
from pathlib import Path

def summarize_imdb_dataset(base_dir="aclImdb", num_files=5):
    base_path = Path(base_dir)
    for split in ["train", "test"]:
        print(f"\nSummary of '{split}' split:")
        for sentiment in ["pos", "neg"]:
            sentiment_path = base_path / split / sentiment
            print(f"  Sentiment: {sentiment}")
            for i, file_path in
enumerate(sorted(sentiment_path.iterdir())[:num_files]):
                with file_path.open("r", encoding="utf-8") as f:
                    content = f.readlines()
                print(f"\n  File {i + 1}: {file_path.name}")
                print(f"    Lines in file: {len(content)}")
                print(f"    First 5 lines (or fewer):")
                print("    " + "\n    ".join(content[:5]).strip())

summarize_imdb_dataset()
```

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

  File 1: 0_9.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Bromwell High is a cartoon comedy. It ran at the same time as some
other programs about school life, such as "Teachers". My 35 years in
the teaching profession lead me to believe that Bromwell High's satire
```

is much closer to reality than is "Teachers". The scramble to survive financially, the insightful students who can see right through their pathetic teachers' pomp, the pettiness of the whole situation, all remind me of the schools I knew and their students. When I saw the episode in which a student repeatedly tried to burn down the school, I immediately recalled ..... at ..... High. A classic line: INSPECTOR: I'm here to sack one of your teachers. STUDENT: Welcome to Bromwell High. I expect that many adults of my age think that Bromwell High is far fetched. What a pity that it isn't!

File 2: 10000_8.txt
Lines in file: 1
First 5 lines (or fewer):
Homelessness (or Houselessness as George Carlin stated) has been an issue for years but never a plan to help those on the street that were once considered human who did everything from going to school, work, or vote for the matter. Most people think of the homeless as just a lost cause while worrying about things such as racism, the war on Iraq, pressuring kids to succeed, technology, the elections, inflation, or worrying if they'll be next to end up on the streets.

But what if you were given a bet to live on the streets for a month without the luxuries you once had from a home, the entertainment sets, a bathroom, pictures on the wall, a computer, and everything you once treasure to see what it's like to be homeless? That is Goddard Bolt's lesson.

Mel Brooks (who directs) who stars as Bolt plays a rich man who has everything in the world until deciding to make a bet with a sissy rival (Jeffery Tambor) to see if he can live in the streets for thirty days without the luxuries; if Bolt succeeds, he can do what he wants with a future project of making more buildings. The bet's on where Bolt is thrown on the street with a bracelet on his leg to monitor his every move where he can't step off the sidewalk. He's given the nickname Pepto by a vagrant after it's written on his forehead where Bolt meets other characters including a woman by the name of Molly (Lesley Ann Warren) an ex-dancer who got divorce before losing her home, and her pals Sailor (Howard Morris) and Fumes (Teddy Wilson) who are already used to the streets. They're survivors. Bolt isn't. He's not used to reaching mutual agreements like he once did when being rich where it's fight or flight, kill or be killed.

While the love connection between Molly and Bolt wasn't necessary to plot, I found "Life Stinks" to be one of Mel Brooks' observant films where prior to being a comedy, it shows a tender side compared to his slapstick work such as Blazing Saddles, Young Frankenstein, or Spaceballs for the matter, to show what it's like having something valuable before losing it the next day or on the other hand making a stupid bet like all rich people do when they don't know what to do with their money. Maybe they should give it to the homeless instead of using it like Monopoly money.

Or maybe this film will inspire you to help others.

```
  File 3: 10001_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Brilliant over-acting by Lesley Ann Warren. Best dramatic hobo
lady I have ever seen, and love scenes in clothes warehouse are second
to none. The corn on face is a classic, as good as anything in Blazing
Saddles. The take on lawyers is also superb. After being accused of
being a turncoat, selling out his boss, and being dishonest the lawyer
of Pepto Bolt shrugs indifferently "I'm a lawyer" he says. Three funny
words. Jeffrey Tambor, a favorite from the later Larry Sanders show,
is fantastic here too as a mad millionaire who wants to crush the
ghetto. His character is more malevolent than usual. The hospital
scene, and the scene where the homeless invade a demolition site, are
all-time classics. Look for the legs scene and the two big diggers
fighting (one bleeds). This movie gets better each time I see it
(which is quite often).

  File 4: 10002_7.txt
    Lines in file: 1
    First 5 lines (or fewer):
    This is easily the most underrated film inn the Brooks cannon.
Sure, its flawed. It does not give a realistic view of homelessness
(unlike, say, how Citizen Kane gave a realistic view of lounge
singers, or Titanic gave a realistic view of Italians YOU IDIOTS).
Many of the jokes fall flat. But still, this film is very lovable in a
way many comedies are not, and to pull that off in a story about some
of the most traditionally reviled members of society is truly
impressive. Its not The Fisher King, but its not crap, either. My only
complaint is that Brooks should have cast someone else in the lead (I
love Mel as a Director and Writer, not so much as a lead).

  File 5: 10003_8.txt
    Lines in file: 1
    First 5 lines (or fewer):
    This is not the typical Mel Brooks film. It was much less
slapstick than most of his movies and actually had a plot that was
followable. Leslie Ann Warren made the movie, she is such a fantastic,
under-rated actress. There were some moments that could have been
fleshed out a bit more, and some scenes that could probably have been
cut to make the room to do so, but all in all, this is worth the price
to rent and see it. The acting was good overall, Brooks himself did a
good job without his characteristic speaking to directly to the
audience. Again, Warren was the best actor in the movie, but "Fume"
and "Sailor" both played their parts well.
  Sentiment: neg

  File 1: 0_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Story of a man who has unnatural feelings for a pig. Starts out
```

with a opening scene that is a terrific example of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it's singers. Unfortunately it stays absurd the WHOLE time with no general narrative eventually making it just too off putting. Even those from the era should be turned off. The cryptic dialogue would make Shakespeare seem easy to a third grader. On a technical level it's better than you might think with some good cinematography by future great Vilmos Zsigmond. Future stars Sally Kirkland and Frederic Forrest can be seen briefly.

  File 2: 10000_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Airport '77 starts as a brand new luxury 747 plane is loaded up with valuable paintings & such belonging to rich businessman Philip Stevens (James Stewart) who is flying them & a bunch of VIP's to his estate in preparation of it being opened to the public as a museum, also on board is Stevens daughter Julie (Kathleen Quinlan) & her son. The luxury jetliner takes off as planned but mid-air the plane is hi-jacked by the co-pilot Chambers (Robert Foxworth) & his two accomplice's Banker (Monte Markham) & Wilson (Michael Pataki) who knock the passengers & crew out with sleeping gas, they plan to steal the valuable cargo & land on a disused plane strip on an isolated island but while making his descent Chambers almost hits an oil rig in the Ocean & loses control of the plane sending it crashing into the sea where it sinks to the bottom right bang in the middle of the Bermuda Triangle. With air in short supply, water leaking in & having flown over 200 miles off course the problems mount for the survivor's as they await help with time fast running out...

Also known under the slightly different tile Airport 1977 this second sequel to the smash-hit disaster thriller Airport (1970) was directed by Jerry Jameson & while once again like it's predecessors I can't say Airport '77 is any sort of forgotten classic it is entertaining although not necessarily for the right reasons. Out of the three Airport films I have seen so far I actually liked this one the best, just. It has my favourite plot of the three with a nice mid-air hi-jacking & then the crashing (didn't he see the oil rig?) & sinking of the 747 (maybe the makers were trying to cross the original Airport with another popular disaster flick of the period The Poseidon Adventure (1972)) & submerged is where it stays until the end with a stark dilemma facing those trapped inside, either suffocate when the air runs out or drown as the 747 floods or if any of the doors are opened & it's a decent idea that could have made for a great little disaster flick but bad unsympathetic character's, dull dialogue, lethargic set-pieces & a real lack of danger or suspense or tension means this is a missed opportunity. While the rather sluggish plot keeps one entertained for 108 odd minutes not that much happens after the plane sinks & there's not as much urgency as I thought there should have been. Even when the Navy become involved things don't pick up that much with a few shots

of huge ships & helicopters flying about but there's just something lacking here. George Kennedy as the jinxed airline worker Joe Patroni is back but only gets a couple of scenes & barely even says anything preferring to just look worried in the background.

The home video & theatrical version of Airport '77 run 108 minutes while the US TV versions add an extra hour of footage including a new opening credits sequence, many more scenes with George Kennedy as Patroni, flashbacks to flesh out character's, longer rescue scenes & the discovery or another couple of dead bodies including the navigator. While I would like to see this extra footage I am not sure I could sit through a near three hour cut of Airport '77. As expected the film has dated badly with horrible fashions & interior design choices, I will say no more other than the toy plane model effects aren't great either. Along with the other two Airport sequels this takes pride of place in the Razzie Award's Hall of Shame although I can think of lots of worse films than this so I reckon that's a little harsh. The action scenes are a little dull unfortunately, the pace is slow & not much excitement or tension is generated which is a shame as I reckon this could have been a pretty good film if made properly.

The production values are alright if nothing spectacular. The acting isn't great, two time Oscar winner Jack Lemmon has said since it was a mistake to star in this, one time Oscar winner James Stewart looks old & frail, also one time Oscar winner Lee Grant looks drunk while Sir Christopher Lee is given little to do & there are plenty of other familiar faces to look out for too.

Airport '77 is the most disaster orientated of the three Airport films so far & I liked the ideas behind it even if they were a bit silly, the production & bland direction doesn't help though & a film about a sunken plane just shouldn't be this boring or lethargic. Followed by The Concorde ... Airport '79 (1979).

  File 3: 10001_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    This film lacked something I couldn't put my finger on at first: charisma on the part of the leading actress. This inevitably translated to lack of chemistry when she shared the screen with her leading man. Even the romantic scenes came across as being merely the actors at play. It could very well have been the director who miscalculated what he needed from the actors. I just don't know.

But could it have been the screenplay? Just exactly who was the chef in love with? He seemed more enamored of his culinary skills and restaurant, and ultimately of himself and his youthful exploits, than of anybody or anything else. He never convinced me he was in love with the princess.

I was disappointed in this movie. But, don't forget it was nominated for an Oscar, so judge for yourself.

  File 4: 10002_1.txt
    Lines in file: 1

```
    First 5 lines (or fewer):
    Sorry everyone,,, I know this is supposed to be an "art" film,,
but wow, they should have handed out guns at the screening so people
could blow their brains out and not watch. Although the scene design
and photographic direction was excellent, this story is too painful to
watch. The absence of a sound track was brutal. The loooonnnnng shots
were too long. How long can you watch two people just sitting there
and talking? Especially when the dialogue is two people complaining. I
really had a hard time just getting through this film. The
performances were excellent, but how much of that dark, sombre,
uninspired, stuff can you take? The only thing i liked was Maureen
Stapleton and her red dress and dancing scene. Otherwise this was a
ripoff of Bergman. And i'm no fan f his either. I think anyone who
says they enjoyed 1 1/2 hours of this is,, well, lying.

  File 5: 10003_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    When I was little my parents took me along to the theater to see
Interiors. It was one of many movies I watched with my parents, but
this was the only one we walked out of. Since then I had never seen
Interiors until just recently, and I could have lived out the rest of
my life without it. What a pretentious, ponderous, and painfully
boring piece of 70's wine and cheese tripe. Woody Allen is one of my
favorite directors but Interiors is by far the worst piece of crap of
his career. In the unmistakable style of Ingmar Berman, Allen gives us
a dark, angular, muted, insight in to the lives of a family wrought by
the psychological damage caused by divorce, estrangement, career,
love, non-love, halitosis, whatever. The film, intentionally, has no
comic relief, no music, and is drenched in shadowy pathos. This film
style can be best defined as expressionist in nature, using an
improvisational method of dialogue to illicit a "more pronounced depth
of meaning and truth". But Woody Allen is no Ingmar Bergman. The film
is painfully slow and dull. But beyond that, I simply had no
connection with or sympathy for any of the characters. Instead I felt
only contempt for this parade of shuffling, whining, nicotine stained,
martyrs in a perpetual quest for identity. Amid a backdrop of
cosmopolitan affluence and baked Brie intelligentsia the story looms
like a fart in the room. Everyone speaks in affected platitudes and
elevated language between cigarettes. Everyone is "lost" and
"struggling", desperate to find direction or understanding or whatever
and it just goes on and on to the point where you just want to slap
all of them. It's never about resolution, it's only about interminable
introspective babble. It is nothing more than a psychological drama
taken to an extreme beyond the audience's ability to connect. Woody
Allen chose to make characters so immersed in themselves we feel left
out. And for that reason I found this movie painfully self indulgent
and spiritually draining. I see what he was going for but his
insistence on promoting his message through Prozac prose and distorted
```

```
film techniques jettisons it past the point of relevance. I highly
recommend this one if you're feeling a little too happy and need
something to remind you of death. Otherwise, let's just pretend this
film never happened.

Summary of 'test' split:
  Sentiment: pos

  File 1: 0_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I went and saw this movie last night after being coaxed to by a
few friends of mine. I'll admit that I was reluctant to see it because
from what I knew of Ashton Kutcher he was only able to do comedy. I
was wrong. Kutcher played the character of Jake Fischer very well, and
Kevin Costner played Ben Randall with such professionalism. The sign
of a good movie is that it can toy with our emotions. This one did
exactly that. The entire theater (which was sold out) was overcome by
laughter during the first half of the movie, and were moved to tears
during the second half. While exiting the theater I not only saw many
women in tears, but many full grown men as well, trying desperately
not to let anyone see them crying. This movie was great, and I suggest
that you go see it before you judge.

  File 2: 10000_7.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Actor turned director Bill Paxton follows up his promising debut,
the Gothic-horror "Frailty", with this family friendly sports drama
about the 1913 U.S. Open where a young American caddy rises from his
humble background to play against his Bristish idol in what was dubbed
as "The Greatest Game Ever Played." I'm no fan of golf, and these
scrappy underdog sports flicks are a dime a dozen (most recently done
to grand effect with "Miracle" and "Cinderella Man"), but some how
this film was enthralling all the same.<br /><br />The film starts
with some creative opening credits (imagine a Disneyfied version of
the animated opening credits of HBO's "Carnivale" and "Rome"), but
lumbers along slowly for its first by-the-numbers hour. Once the
action moves to the U.S. Open things pick up very well. Paxton does a
nice job and shows a knack for effective directorial flourishes (I
loved the rain-soaked montage of the action on day two of the open)
that propel the plot further or add some unexpected psychological
depth to the proceedings. There's some compelling character
development when the British Harry Vardon is haunted by images of the
aristocrats in black suits and top hats who destroyed his family
cottage as a child to make way for a golf course. He also does a good
job of visually depicting what goes on in the players' heads under
pressure. Golf, a painfully boring sport, is brought vividly alive
here. Credit should also be given the set designers and costume
department for creating an engaging period-piece atmosphere of London
```

and Boston at the beginning of the twentieth century.

You know how this is going to end not only because it's based on a true story but also because films in this genre follow the same template over and over, but Paxton puts on a better than average show and perhaps indicates more talent behind the camera than he ever had in front of it. Despite the formulaic nature, this is a nice and easy film to root for that deserves to find an audience.

  File 3: 10001_9.txt
    Lines in file: 1
    First 5 lines (or fewer):
    As a recreational golfer with some knowledge of the sport's history, I was pleased with Disney's sensitivity to the issues of class in golf in the early twentieth century. The movie depicted well the psychological battles that Harry Vardon fought within himself, from his childhood trauma of being evicted to his own inability to break that glass ceiling that prevents him from being accepted as an equal in English golf society. Likewise, the young Ouimet goes through his own class struggles, being a mere caddie in the eyes of the upper crust Americans who scoff at his attempts to rise above his standing.

What I loved best, however, is how this theme of class is manifested in the characters of Ouimet's parents. His father is a working-class drone who sees the value of hard work but is intimidated by the upper class; his mother, however, recognizes her son's talent and desire and encourages him to pursue his dream of competing against those who think he is inferior.

Finally, the golf scenes are well photographed. Although the course used in the movie was not the actual site of the historical tournament, the little liberties taken by Disney do not detract from the beauty of the film. There's one little Disney moment at the pool table; otherwise, the viewer does not really think Disney. The ending, as in "Miracle," is not some Disney creation, but one that only human history could have written.

  File 4: 10002_8.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I saw this film in a sneak preview, and it is delightful. The cinematography is unusually creative, the acting is good, and the story is fabulous. If this movie does not do well, it won't be because it doesn't deserve to. Before this film, I didn't realize how charming Shia Lebouf could be. He does a marvelous, self-contained, job as the lead. There's something incredibly sweet about him, and it makes the movie even better. The other actors do a good job as well, and the film contains moments of really high suspense, more than one might expect from a movie about golf. Sports movies are a dime a dozen, but this one stands out.

This is one I'd recommend to anyone.

  File 5: 10003_8.txt
    Lines in file: 1
    First 5 lines (or fewer):

    Bill Paxton has taken the true story of the 1913 US golf open and made a film that is about much more than an extra-ordinary game of golf. The film also deals directly with the class tensions of the early twentieth century and touches upon the profound anti-Catholic prejudices of both the British and American establishments. But at heart the film is about that perennial favourite of triumph against the odds.

The acting is exemplary throughout. Stephen Dillane is excellent as usual, but the revelation of the movie is Shia LaBoeuf who delivers a disciplined, dignified and highly sympathetic performance as a working class Franco-Irish kid fighting his way through the prejudices of the New England WASP establishment. For those who are only familiar with his slap-stick performances in "Even Stevens" this demonstration of his maturity is a delightful surprise. And Josh Flitter as the ten year old caddy threatens to steal every scene in which he appears.

A old fashioned movie in the best sense of the word: fine acting, clear directing and a great story that grips to the end - the final scene an affectionate nod to Casablanca is just one of the many pleasures that fill a great movie.
  Sentiment: neg

  File 1: 0_2.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Once again Mr. Costner has dragged out a movie for far longer than necessary. Aside from the terrific sea rescue sequences, of which there are very few I just did not care about any of the characters. Most of us have ghosts in the closet, and Costner's character are realized early on, and then forgotten until much later, by which time I did not care. The character we should really care about is a very cocky, overconfident Ashton Kutcher. The problem is he comes off as kid who thinks he's better than anyone else around him and shows no signs of a cluttered closet. His only obstacle appears to be winning over Costner. Finally when we are well past the half way point of this stinker, Costner tells us all about Kutcher's ghosts. We are told why Kutcher is driven to be the best with no prior inkling or foreshadowing. No magic here, it was all I could do to keep from turning it off an hour in.

  File 2: 10000_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    This is an example of why the majority of action films are the same. Generic and boring, there's really nothing worth watching here. A complete waste of the then barely-tapped talents of Ice-T and Ice Cube, who've each proven many times over that they are capable of acting, and acting well. Don't bother with this one, go see New Jack City, Ricochet or watch New York Undercover for Ice-T, or Boyz n the Hood, Higher Learning or Friday for Ice Cube and see the real deal. Ice-T's horribly cliched dialogue alone makes this film grate at the

teeth, and I'm still wondering what the heck Bill Paxton was doing in this film? And why the heck does he always play the exact same character? From Aliens onward, every film I've seen with Bill Paxton has him playing the exact same irritating character, and at least in Aliens his character died, which made it somewhat gratifying...

Overall, this is second-rate action trash. There are countless better films to see, and if you really want to see this one, watch Judgement Night, which is practically a carbon copy but has better acting and a better script. The only thing that made this at all worth watching was a decent hand on the camera - the cinematography was almost refreshing, which comes close to making up for the horrible film itself - but not quite. 4/10.

  File 3: 10001_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    First of all I hate those moronic rappers, who could'nt act if they had a gun pressed against their foreheads. All they do is curse and shoot each other and acting like cliché'e version of gangsters.

The movie doesn't take more than five minutes to explain what is going on before we're already at the warehouse There is not a single sympathetic character in this movie, except for the homeless guy, who is also the only one with half a brain.

Bill Paxton and William Sadler are both hill billies and Sadlers character is just as much a villain as the gangsters. I did'nt like him right from the start.

The movie is filled with pointless violence and Walter Hills specialty: people falling through windows with glass flying everywhere. There is pretty much no plot and it is a big problem when you root for no-one. Everybody dies, except from Paxton and the homeless guy and everybody get what they deserve.

The only two black people that can act is the homeless guy and the junkie but they're actors by profession, not annoying ugly brain dead rappers.

Stay away from this crap and watch 48 hours 1 and 2 instead. At lest they have characters you care about, a sense of humor and nothing but real actors in the cast.

  File 4: 10002_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Not even the Beatles could write songs everyone liked, and although Walter Hill is no mop-top he's second to none when it comes to thought provoking action movies. The nineties came and social platforms were changing in music and film, the emergence of the Rapper turned movie star was in full swing, the acting took a back seat to each man's overpowering regional accent and transparent acting. This was one of the many ice-t movies i saw as a kid and loved, only to watch them later and cringe. Bill Paxton and William Sadler are firemen with basic lives until a burning building tenant about to go up in flames hands over a map with gold implications. I hand it to

```
Walter for quickly and neatly setting up the main characters and
location. But i fault everyone involved for turning out Lame-o
performances. Ice-t and cube must have been red hot at this time, and
while I've enjoyed both their careers as rappers, in my opinion they
fell flat in this movie. It's about ninety minutes of one guy
ridiculously turning his back on the other guy to the point you find
yourself locked in multiple states of disbelief. Now this is a movie,
its not a documentary so i wont waste my time recounting all the
stupid plot twists in this movie, but there were many, and they led
nowhere. I got the feeling watching this that everyone on set was sord
of confused and just playing things off the cuff. There are two things
i still enjoy about it, one involves a scene with a needle and the
other is Sadler's huge 45 pistol. Bottom line this movie is like
domino's pizza. Yeah ill eat it if I'm hungry and i don't feel like
cooking, But I'm well aware it tastes like crap. 3 stars, meh.

  File 5: 10003_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Brass pictures (movies is not a fitting word for them) really are
somewhat brassy. Their alluring visual qualities are reminiscent of
expensive high class TV commercials. But unfortunately Brass pictures
are feature films with the pretense of wanting to entertain viewers
for over two hours! In this they fail miserably, their undeniable, but
rather soft and flabby than steamy, erotic qualities non
withstanding.<br /><br />Senso '45 is a remake of a film by Luchino
Visconti with the same title and Alida Valli and Farley Granger in the
lead. The original tells a story of senseless love and lust in and
around Venice during the Italian wars of independence. Brass moved the
action from the 19th into the 20th century, 1945 to be exact, so there
are Mussolini murals, men in black shirts, German uniforms or the
tattered garb of the partisans. But it is just window dressing, the
historic context is completely negligible.<br /><br />Anna Galiena
plays the attractive aristocratic woman who falls for the amoral SS
guy who always puts on too much lipstick. She is an attractive,
versatile, well trained Italian actress and clearly above the
material. Her wide range of facial expressions (signalling boredom,
loathing, delight, fear, hate ... and ecstasy) are the best reason to
watch this picture and worth two stars. She endures this basically
trashy stuff with an astonishing amount of dignity. I wish some really
good parts come along for her. She really deserves it.
```

```
from pathlib import Path

batch_size = 32
base_dir = Path("aclImdb")
train_dir = base_dir / "train"
val_dir = base_dir / "val"

for category in ("neg", "pos"):
```

```
    category_train = train_dir / category
    category_val = val_dir / category

    category_val.mkdir(parents=True, exist_ok=True)  # Make validation
directory if not exist

    files = sorted(category_train.glob("*.txt"))
    random.Random(1337).shuffle(files)

    num_val_samples = int(0.2 * len(files))
    val_files = files[-num_val_samples:]

    for file_path in val_files:
        dest_path = category_val / file_path.name
        if not dest_path.exists():  # Only move if not already moved
            shutil.move(str(file_path), str(dest_path))

from tensorflow import keras

# Use pathlib-based paths (already defined earlier)
train_path = base_dir / "train"
val_path = base_dir / "val"
test_path = base_dir / "test"

# Load datasets using keras.utils.text_dataset_from_directory
train_ds = keras.utils.text_dataset_from_directory(
    directory=str(train_path),
    batch_size=batch_size
)

val_ds = keras.utils.text_dataset_from_directory(
    directory=str(val_path),
    batch_size=batch_size
)

test_ds = keras.utils.text_dataset_from_directory(
    directory=str(test_path),
    batch_size=batch_size
)

# Create a dataset of only the raw text (no labels) from the training
set
text_only_train_ds = train_ds.map(lambda x, y: x)

Found 20000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.

from tensorflow.keras import layers

# Set parameters
```

```
max_tokens = 10000
max_length = 150

# Create the TextVectorization layer
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length
)

# Adapt the layer on the training text only (no labels)
text_vectorization.adapt(text_only_train_ds)

# Tokenize and map datasets using the vectorizer
# Limit training to 100 samples, keep full val/test unless changed

int_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=tf.data.AUTOTUNE
).take(100)

int_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=tf.data.AUTOTUNE
).take(10000)

int_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=tf.data.AUTOTUNE
)

from tensorflow.keras import Model, Input
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM,
Dropout, Dense

# Define model input
inputs = Input(shape=(None,), dtype="int64")

# Embedding + Bidirectional LSTM
x = Embedding(input_dim=max_tokens, output_dim=128)(inputs)
x = Bidirectional(LSTM(32))(x)
x = Dropout(0.25)(x)

# Output layer for binary classification
outputs = Dense(1, activation="sigmoid")(x)

# Build the model
embedding_model = Model(inputs, outputs)

# Compile the model
embedding_model.compile(
```

```
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

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

```
Model: "functional"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃ Param #         ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)             │ (None, None)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ embedding (Embedding)                │ (None, None, 128)           │       1,280,000 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ bidirectional (Bidirectional)        │ (None, 64)                  │          41,216 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 64)                  │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (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)
```

```
from tensorflow.keras.callbacks import ModelCheckpoint

# Define callbacks
callbacks = [
    ModelCheckpoint(
        filepath="embedding_model.keras",
        save_best_only=True,
        monitor="val_loss",
```

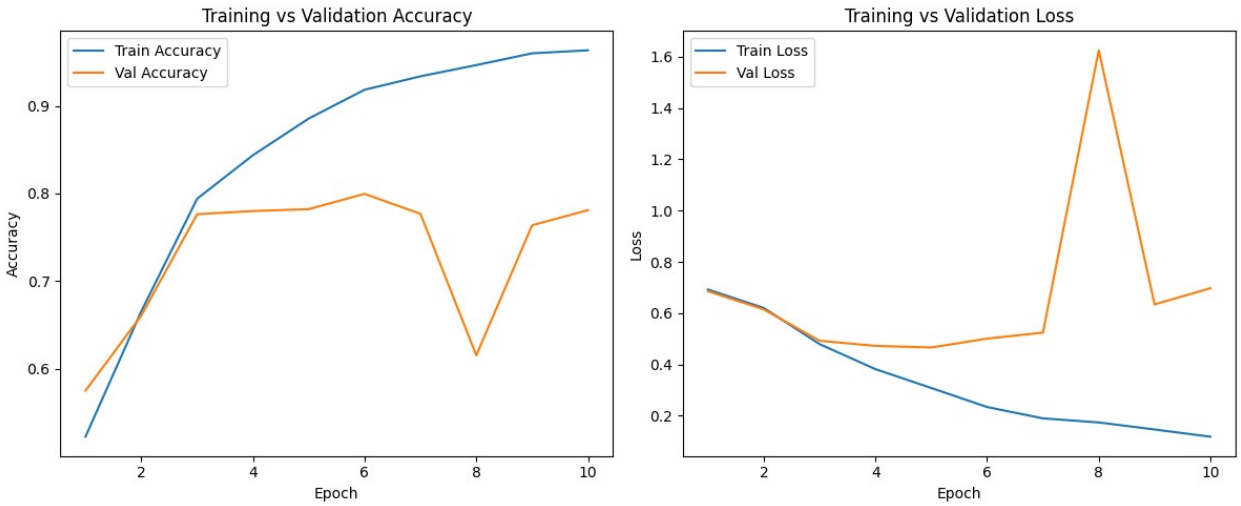
```
        mode="min",
        verbose=1
    )
]

# Train the embedding model
history_embedded = embedding_model.fit(
    int_train_ds,
    validation_data=int_val_ds,
    epochs=10,
    callbacks=callbacks,
    verbose=1  # Optional: show training progress per epoch
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.4994 - loss:
0.6933
Epoch 1: val_loss improved from inf to 0.68549, saving model to
embedding_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 90ms/step - accuracy: 0.4996 - loss:
0.6933 - val_accuracy: 0.5750 - val_loss: 0.6855
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.6330 - loss:
0.6512
Epoch 2: val_loss improved from 0.68549 to 0.61443, saving model to
embedding_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.6333 - loss:
0.6509 - val_accuracy: 0.6608 - val_loss: 0.6144
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.7910 - loss:
0.4897
Epoch 3: val_loss improved from 0.61443 to 0.49207, saving model to
embedding_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.7910 - loss:
0.4896 - val_accuracy: 0.7764 - val_loss: 0.4921
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.8501 - loss:
0.3793
Epoch 4: val_loss improved from 0.49207 to 0.47216, saving model to
embedding_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.8500 - loss:
0.3793 - val_accuracy: 0.7800 - val_loss: 0.4722
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8943 - loss:
0.2928
Epoch 5: val_loss improved from 0.47216 to 0.46578, saving model to
embedding_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.8942 - loss:
0.2929 - val_accuracy: 0.7822 - val_loss: 0.4658
Epoch 6/10
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.9195 - loss:
0.2300
Epoch 6: val_loss did not improve from 0.46578
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9195 - loss:
0.2301 - val_accuracy: 0.7996 - val_loss: 0.5004
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9374 - loss:
0.1770
Epoch 7: val_loss did not improve from 0.46578
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9374 - loss:
0.1771 - val_accuracy: 0.7768 - val_loss: 0.5238
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9535 - loss:
0.1548
Epoch 8: val_loss did not improve from 0.46578
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9534 - loss:
0.1550 - val_accuracy: 0.6154 - val_loss: 1.6257
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9424 - loss:
0.2211
Epoch 9: val_loss did not improve from 0.46578
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - accuracy: 0.9426 - loss:
0.2203 - val_accuracy: 0.7638 - val_loss: 0.6340
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.9643 - loss:
0.1199
Epoch 10: val_loss did not improve from 0.46578
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - accuracy: 0.9643 - loss:
0.1199 - val_accuracy: 0.7810 - val_loss: 0.6971
```

```python
import matplotlib.pyplot as plt

# Get training history
metrics = history_embedded.history
epochs = range(1, len(metrics['accuracy']) + 1)

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

# Plot Accuracy
plt.subplot(1, 2, 1)
plt.plot(epochs, metrics['accuracy'], label='Train Accuracy')
plt.plot(epochs, metrics['val_accuracy'], label='Val Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(False)

# Plot Loss
```

```
plt.subplot(1, 2, 2)
plt.plot(epochs, metrics['loss'], label='Train Loss')
plt.plot(epochs, metrics['val_loss'], label='Val Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(False)

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




```
# Download GloVe embeddings (6B tokens, 100D vectors)
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2025-04-08 16:07:49--  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 16:07:49--  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 16:07:49--
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’

glove.6B.zip        100%[===================>] 822.24M  5.11MB/s    in
2m 39s

2025-04-08 16:10:28 (5.18 MB/s) - ‘glove.6B.zip’ saved
[862182613/862182613]
```

```python
import numpy as np

# Set GloVe embedding dimensions and file path
embedding_dim = 100
glove_path = Path("glove.6B.100d.txt")

# Load GloVe word vectors into a dictionary
embeddings_index = {}
with glove_path.open("r", encoding="utf8") as f:
    for line in f:
        word, coefs = line.strip().split(maxsplit=1)
        embeddings_index[word] = np.fromstring(coefs, dtype="f", sep="
")

# Retrieve vocabulary from the TextVectorization layer
vocabulary = text_vectorization.get_vocabulary()
word_index = {word: i for i, word in enumerate(vocabulary)}

# Prepare embedding matrix
embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

from tensorflow.keras import Model, Input
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM,
Dropout, Dense
from tensorflow.keras.initializers import Constant

# Define the non-trainable embedding layer using GloVe weights
embedding_layer = Embedding(
    input_dim=max_tokens,
    output_dim=embedding_dim,
    embeddings_initializer=Constant(embedding_matrix),
    trainable=False,
```

```
    mask_zero=True  # Optional, useful if padding is used
)

# Build the model using functional API
inputs = Input(shape=(None,), dtype="int64")
x = embedding_layer(inputs)
x = Bidirectional(LSTM(32))(x)
x = Dropout(0.25)(x)
outputs = Dense(1, activation="sigmoid")(x)

# Construct the model
pretrained_model = Model(inputs, outputs)
```

```
from tensorflow.keras import Model, Input
from tensorflow.keras.layers import Bidirectional, LSTM, Dropout,
Dense

# Build model using the pretrained embedding layer
inputs = Input(shape=(None,), dtype="int64")
x = embedding_layer(inputs)
x = Bidirectional(LSTM(32))(x)
x = Dropout(0.25)(x)  # Dropout set to 0.25 as you used earlier
outputs = Dense(1, activation="sigmoid")(x)

# Create the model
pretrained_model = Model(inputs, outputs)

# Compile the model
pretrained_model.compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

# Display the model architecture
pretrained_model.summary()
```

```
Model: "functional_4"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape                          ┃                       Param # ┃
┃ Connected to                    ┃                                       ┃                               ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
┃ input_layer_4                   │ (None, None)                          │                             0 ┃
┃ -                               │                                       │                               ┃
┃ (InputLayer)                    │                                       │                               ┃
└─────────────────────────────────┴───────────────────────────────────────┴───────────────────────────────┘
```

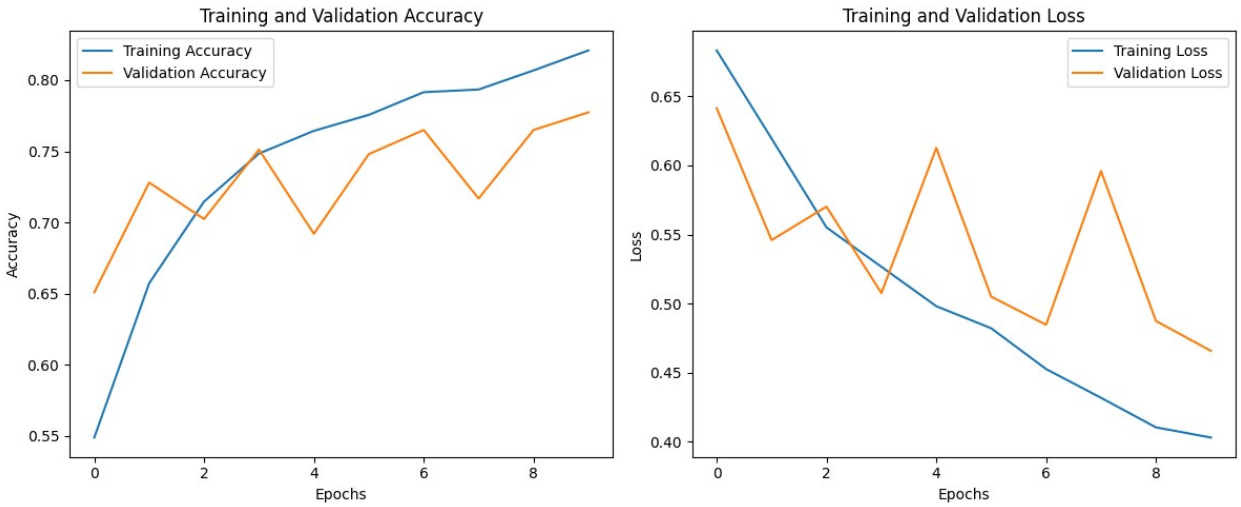
```
| embedding_2 (Embedding)    | (None, None, 100)       |       1,000,000
| input_layer_4[0][0]        |                         |
|----------------------------|-------------------------|-----------------
| not_equal_3 (NotEqual)     | (None, None)            |               0
| input_layer_4[0][0]        |                         |
|----------------------------|-------------------------|-----------------
| bidirectional_4            | (None, 64)              |          34,048
| embedding_2[1][0],         |                         |
| (Bidirectional)            |                         |
| not_equal_3[0][0]          |                         |
|----------------------------|-------------------------|-----------------
| dropout_4 (Dropout)        | (None, 64)              |               0
| bidirectional_4[0][0]      |                         |
|----------------------------|-------------------------|-----------------
| dense_4 (Dense)            | (None, 1)               |              65
| dropout_4[0][0]            |                         |
└────────────────────────────┴─────────────────────────┴─────────────────

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

 Trainable params: 34,113 (133.25 KB)

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

```python
# Define the callback to save the best model
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pretrained_model.keras",
        save_best_only=True,
        monitor="val_loss",  # Monitor validation loss for best model
        mode="min",  # Save when the validation loss decreases
        verbose=1  # Print when saving the model
    )
]

# Train the pretrained model
history_pretrained = pretrained_model.fit(
    int_train_ds,
    validation_data=int_val_ds,
    epochs=10,
    callbacks=callbacks,
    verbose=1  # Display training progress
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.5124 - loss:
0.7044
Epoch 1: val_loss improved from inf to 0.64145, saving model to
pretrained_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 15s 124ms/step - accuracy: 0.5127 - loss:
0.7042 - val_accuracy: 0.6508 - val_loss: 0.6415
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.6351 - loss:
0.6376
Epoch 2: val_loss improved from 0.64145 to 0.54596, saving model to
pretrained_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 122ms/step - accuracy: 0.6353 - loss:
0.6374 - val_accuracy: 0.7280 - val_loss: 0.5460
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7090 - loss:
0.5685
Epoch 3: val_loss did not improve from 0.54596
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 104ms/step - accuracy: 0.7091 - loss:
0.5684 - val_accuracy: 0.7024 - val_loss: 0.5702
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.7523 - loss:
0.5304
Epoch 4: val_loss improved from 0.54596 to 0.50763, saving model to
pretrained_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 117ms/step - accuracy: 0.7523 - loss:
0.5304 - val_accuracy: 0.7512 - val_loss: 0.5076
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.7648 - loss:
0.5032
Epoch 5: val_loss did not improve from 0.50763
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 103ms/step - accuracy: 0.7648 - loss:
0.5032 - val_accuracy: 0.6920 - val_loss: 0.6126
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.7775 - loss:
0.4816
Epoch 6: val_loss improved from 0.50763 to 0.50504, saving model to
pretrained_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 118ms/step - accuracy: 0.7775 - loss:
0.4816 - val_accuracy: 0.7480 - val_loss: 0.5050
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7872 - loss:
0.4547
Epoch 7: val_loss improved from 0.50504 to 0.48467, saving model to
pretrained_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 119ms/step - accuracy: 0.7872 - loss:
0.4547 - val_accuracy: 0.7650 - val_loss: 0.4847
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - accuracy: 0.7960 - loss:
0.4249
```

```
Epoch 8: val_loss did not improve from 0.48467
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 101ms/step - accuracy: 0.7959 - loss:
0.4250 - val_accuracy: 0.7168 - val_loss: 0.5959
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.8049 - loss:
0.4090
Epoch 9: val_loss did not improve from 0.48467
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 105ms/step - accuracy: 0.8049 - loss:
0.4090 - val_accuracy: 0.7650 - val_loss: 0.4874
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.8209 - loss:
0.4011
Epoch 10: val_loss improved from 0.48467 to 0.46580, saving model to
pretrained_model.keras
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 123ms/step - accuracy: 0.8209 - loss:
0.4012 - val_accuracy: 0.7774 - val_loss: 0.4658
```

```python
import matplotlib.pyplot as plt

# Extract training history
history_metrics = history_pretrained.history

# Create a figure for the plots
plt.figure(figsize=(12, 5))

# Subplot for accuracy
plt.subplot(1, 2, 1)
plt.plot(history_metrics['accuracy'], label='Training Accuracy')
plt.plot(history_metrics['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(False)

# Subplot for loss
plt.subplot(1, 2, 2)
plt.plot(history_metrics['loss'], label='Training Loss')
plt.plot(history_metrics['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(False)

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




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

# Data
sample_sizes = [100, 200, 500, 1000]
embedding_accuracies = []
pretrained_accuracies = []

# Initialize the plot
plt.figure(figsize=(12, 6))
plt.title('Model Accuracy vs. Training Sample Sizes')
plt.xlabel('Training Sample Size')
plt.ylabel('Accuracy')
plt.grid(True)

# Iterate over sample sizes
for i, size in enumerate(sample_sizes):
    print(f"\n### Training with {size} samples ###\n")

    # Update training dataset with the specific size
    small_train_ds = train_ds.map(
        lambda x, y: (text_vectorization(x), y)).take(size)

    # Train custom embedding model
    print(f"Training Custom Embedding Model with {size} samples:")
    history_embedding = embedding_model.fit(
        small_train_ds,
        validation_data=int_val_ds,
        epochs=10,
        verbose=1
    )
    embedding_acc = history_embedding.history['accuracy'][-1]  # Use
the last epoch's accuracy
```

```
    embedding_accuracies.append(embedding_acc)
    print(f"Custom Embedding Model Accuracy: {embedding_acc:.4f}\n")

    # Train pretrained embedding model
    print(f"Training Pretrained Embedding Model with {size} samples:")
    history_pretrained = pretrained_model.fit(
        small_train_ds,
        validation_data=int_val_ds,
        epochs=10,
        verbose=1
    )
    pretrained_acc = history_pretrained.history['accuracy'][-1]  # Use
the last epoch's accuracy
    pretrained_accuracies.append(pretrained_acc)
    print(f"Pretrained Embedding Model Accuracy: {pretrained_acc:.4f}\
n")

    # Update the plot dynamically after each iteration
    if i == len(sample_sizes) - 1:  # Once all iterations are done,
plot the final line graph
        plt.plot(sample_sizes, embedding_accuracies, marker='o',
label='Custom Embedding', color='blue')
        plt.plot(sample_sizes, pretrained_accuracies, marker='o',
label='Pretrained Embedding', color='orange')

# Final plot styling
plt.title('Model Accuracy vs. Training Sample Sizes')
plt.xlabel('Training Sample Size')
plt.ylabel('Accuracy')
plt.xticks(sample_sizes)  # Set x-ticks to sample sizes
plt.grid(True)
plt.legend()

# Show the final plot
plt.tight_layout()
plt.show()


### Training with 100 samples ###

Training Custom Embedding Model with 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9765 - loss:
0.0790 - val_accuracy: 0.7712 - val_loss: 0.6051
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9849 - loss:
0.0544 - val_accuracy: 0.7688 - val_loss: 0.6807
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9897 - loss:
0.0468 - val_accuracy: 0.8014 - val_loss: 0.6894
```

```
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9907 - loss:
0.0341 - val_accuracy: 0.7942 - val_loss: 0.6689
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9863 - loss:
0.0474 - val_accuracy: 0.8000 - val_loss: 0.6931
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9959 - loss:
0.0167 - val_accuracy: 0.7726 - val_loss: 0.7023
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9957 - loss:
0.0225 - val_accuracy: 0.7936 - val_loss: 0.8173
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9975 - loss:
0.0105 - val_accuracy: 0.7940 - val_loss: 0.9316
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - accuracy: 0.9941 - loss:
0.0275 - val_accuracy: 0.7950 - val_loss: 0.8631
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - accuracy: 0.9923 - loss:
0.0198 - val_accuracy: 0.7812 - val_loss: 0.7468
Custom Embedding Model Accuracy: 0.9912

Training Pretrained Embedding Model with 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 105ms/step - accuracy: 0.8343 - loss:
0.3717 - val_accuracy: 0.7764 - val_loss: 0.4694
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 99ms/step - accuracy: 0.8474 - loss:
0.3583 - val_accuracy: 0.7794 - val_loss: 0.4674
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 107ms/step - accuracy: 0.8488 - loss:
0.3380 - val_accuracy: 0.7552 - val_loss: 0.4932
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 105ms/step - accuracy: 0.8625 - loss:
0.3197 - val_accuracy: 0.7814 - val_loss: 0.4888
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 107ms/step - accuracy: 0.8592 - loss:
0.3108 - val_accuracy: 0.7246 - val_loss: 0.5589
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 106ms/step - accuracy: 0.8671 - loss:
0.2947 - val_accuracy: 0.7726 - val_loss: 0.4897
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 109ms/step - accuracy: 0.8922 - loss:
0.2665 - val_accuracy: 0.7626 - val_loss: 0.5014
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 105ms/step - accuracy: 0.9010 - loss:
0.2453 - val_accuracy: 0.7618 - val_loss: 0.5148
Epoch 9/10
```

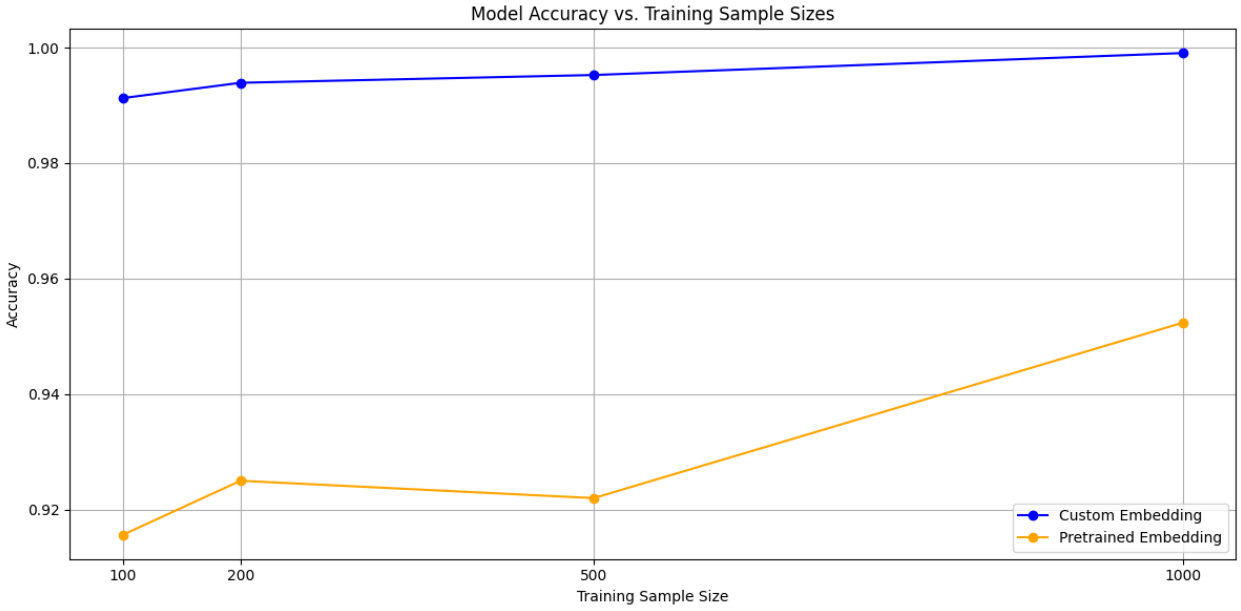
```
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 106ms/step - accuracy: 0.9141 - loss:
0.2241 - val_accuracy: 0.7698 - val_loss: 0.5329
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 105ms/step - accuracy: 0.9236 - loss:
0.2059 - val_accuracy: 0.7712 - val_loss: 0.5814
Pretrained Embedding Model Accuracy: 0.9156


### Training with 200 samples ###

Training Custom Embedding Model with 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 72ms/step - accuracy: 0.9636 - loss:
0.0974 - val_accuracy: 0.7784 - val_loss: 0.4766
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9689 - loss:
0.0949 - val_accuracy: 0.7850 - val_loss: 0.4894
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 72ms/step - accuracy: 0.9768 - loss:
0.0742 - val_accuracy: 0.7790 - val_loss: 0.5408
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9822 - loss:
0.0517 - val_accuracy: 0.7954 - val_loss: 0.5596
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9865 - loss:
0.0430 - val_accuracy: 0.7768 - val_loss: 0.7048
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 70ms/step - accuracy: 0.9928 - loss:
0.0282 - val_accuracy: 0.7856 - val_loss: 0.6209
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 72ms/step - accuracy: 0.9944 - loss:
0.0195 - val_accuracy: 0.7684 - val_loss: 0.6771
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9954 - loss:
0.0160 - val_accuracy: 0.7702 - val_loss: 0.8887
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 15s 73ms/step - accuracy: 0.9948 - loss:
0.0161 - val_accuracy: 0.7872 - val_loss: 0.8033
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 14s 71ms/step - accuracy: 0.9966 - loss:
0.0105 - val_accuracy: 0.7818 - val_loss: 0.7723
Custom Embedding Model Accuracy: 0.9939

Training Pretrained Embedding Model with 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.8981 - loss:
0.2441 - val_accuracy: 0.7776 - val_loss: 0.4616
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.8873 - loss:
0.2662 - val_accuracy: 0.7914 - val_loss: 0.4456
```

```
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.8958 - loss:
0.2424 - val_accuracy: 0.7882 - val_loss: 0.4515
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.9095 - loss:
0.2374 - val_accuracy: 0.7864 - val_loss: 0.4579
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.9067 - loss:
0.2242 - val_accuracy: 0.7824 - val_loss: 0.4531
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 89ms/step - accuracy: 0.9240 - loss:
0.2012 - val_accuracy: 0.7936 - val_loss: 0.4520
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 91ms/step - accuracy: 0.9310 - loss:
0.1784 - val_accuracy: 0.7830 - val_loss: 0.4716
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 90ms/step - accuracy: 0.9395 - loss:
0.1640 - val_accuracy: 0.7826 - val_loss: 0.4895
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 18s 89ms/step - accuracy: 0.9440 - loss:
0.1525 - val_accuracy: 0.7714 - val_loss: 0.5444
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 17s 84ms/step - accuracy: 0.9459 - loss:
0.1413 - val_accuracy: 0.7650 - val_loss: 0.5570
Pretrained Embedding Model Accuracy: 0.9250


### Training with 500 samples ###

Training Custom Embedding Model with 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9505 - loss:
0.1319 - val_accuracy: 0.8264 - val_loss: 0.3952
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9558 - loss:
0.1255 - val_accuracy: 0.8322 - val_loss: 0.3936
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9677 - loss:
0.0914 - val_accuracy: 0.7894 - val_loss: 0.5173
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9766 - loss:
0.0690 - val_accuracy: 0.8228 - val_loss: 0.4810
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9865 - loss:
0.0454 - val_accuracy: 0.8016 - val_loss: 0.5333
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9892 - loss:
0.0335 - val_accuracy: 0.8254 - val_loss: 0.7266
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9922 - loss:
```

```
0.0247 - val_accuracy: 0.7992 - val_loss: 0.6826
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 63ms/step - accuracy: 0.9955 - loss:
0.0162 - val_accuracy: 0.8072 - val_loss: 0.8109
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9948 - loss:
0.0193 - val_accuracy: 0.8080 - val_loss: 0.8766
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 32s 64ms/step - accuracy: 0.9965 - loss:
0.0120 - val_accuracy: 0.8098 - val_loss: 0.9172
Custom Embedding Model Accuracy: 0.9952

Training Pretrained Embedding Model with 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9111 - loss:
0.2245 - val_accuracy: 0.7960 - val_loss: 0.4548
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.8993 - loss:
0.2481 - val_accuracy: 0.8050 - val_loss: 0.4293
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9039 - loss:
0.2358 - val_accuracy: 0.8092 - val_loss: 0.4290
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9078 - loss:
0.2215 - val_accuracy: 0.8190 - val_loss: 0.4025
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9192 - loss:
0.2042 - val_accuracy: 0.8158 - val_loss: 0.4156
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.9225 - loss:
0.1957 - val_accuracy: 0.8196 - val_loss: 0.4097
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 39s 79ms/step - accuracy: 0.9323 - loss:
0.1789 - val_accuracy: 0.8096 - val_loss: 0.4135
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.9344 - loss:
0.1705 - val_accuracy: 0.8142 - val_loss: 0.4187
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 41s 81ms/step - accuracy: 0.9407 - loss:
0.1540 - val_accuracy: 0.8152 - val_loss: 0.4461
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.9430 - loss:
0.1453 - val_accuracy: 0.8140 - val_loss: 0.4365
Pretrained Embedding Model Accuracy: 0.9220


### Training with 1000 samples ###

Training Custom Embedding Model with 1000 samples:
Epoch 1/10
```

```
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9937 - loss:
0.0246 - val_accuracy: 0.8090 - val_loss: 0.4808
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9923 - loss:
0.0284 - val_accuracy: 0.8232 - val_loss: 0.4913
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9950 - loss:
0.0178 - val_accuracy: 0.8046 - val_loss: 0.5936
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 62ms/step - accuracy: 0.9966 - loss:
0.0126 - val_accuracy: 0.8146 - val_loss: 0.7205
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9977 - loss:
0.0085 - val_accuracy: 0.8176 - val_loss: 0.8480
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9978 - loss:
0.0087 - val_accuracy: 0.8178 - val_loss: 1.0125
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9984 - loss:
0.0077 - val_accuracy: 0.8250 - val_loss: 1.0217
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9979 - loss:
0.0059 - val_accuracy: 0.8284 - val_loss: 1.1687
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 63ms/step - accuracy: 0.9983 - loss:
0.0040 - val_accuracy: 0.8212 - val_loss: 1.2666
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 39s 62ms/step - accuracy: 0.9994 - loss:
0.0020 - val_accuracy: 0.8126 - val_loss: 1.2523
Custom Embedding Model Accuracy: 0.9991

Training Pretrained Embedding Model with 1000 samples:
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 82ms/step - accuracy: 0.9440 - loss:
0.1487 - val_accuracy: 0.8254 - val_loss: 0.3924
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 80ms/step - accuracy: 0.9406 - loss:
0.1544 - val_accuracy: 0.8152 - val_loss: 0.4139
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 80ms/step - accuracy: 0.9431 - loss:
0.1472 - val_accuracy: 0.8262 - val_loss: 0.4074
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 49s 78ms/step - accuracy: 0.9496 - loss:
0.1342 - val_accuracy: 0.8232 - val_loss: 0.4081
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 80ms/step - accuracy: 0.9543 - loss:
0.1267 - val_accuracy: 0.8230 - val_loss: 0.4181
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 80ms/step - accuracy: 0.9562 - loss:
```

```
0.1202 - val_accuracy: 0.8222 - val_loss: 0.4363
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 50s 79ms/step - accuracy: 0.9578 - loss:
0.1128 - val_accuracy: 0.8172 - val_loss: 0.4477
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 49s 79ms/step - accuracy: 0.9616 - loss:
0.1080 - val_accuracy: 0.8194 - val_loss: 0.4629
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 49s 79ms/step - accuracy: 0.9629 - loss:
0.0995 - val_accuracy: 0.8204 - val_loss: 0.4820
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 51s 81ms/step - accuracy: 0.9669 - loss:
0.0940 - val_accuracy: 0.8178 - val_loss: 0.5095
Pretrained Embedding Model Accuracy: 0.9524
```



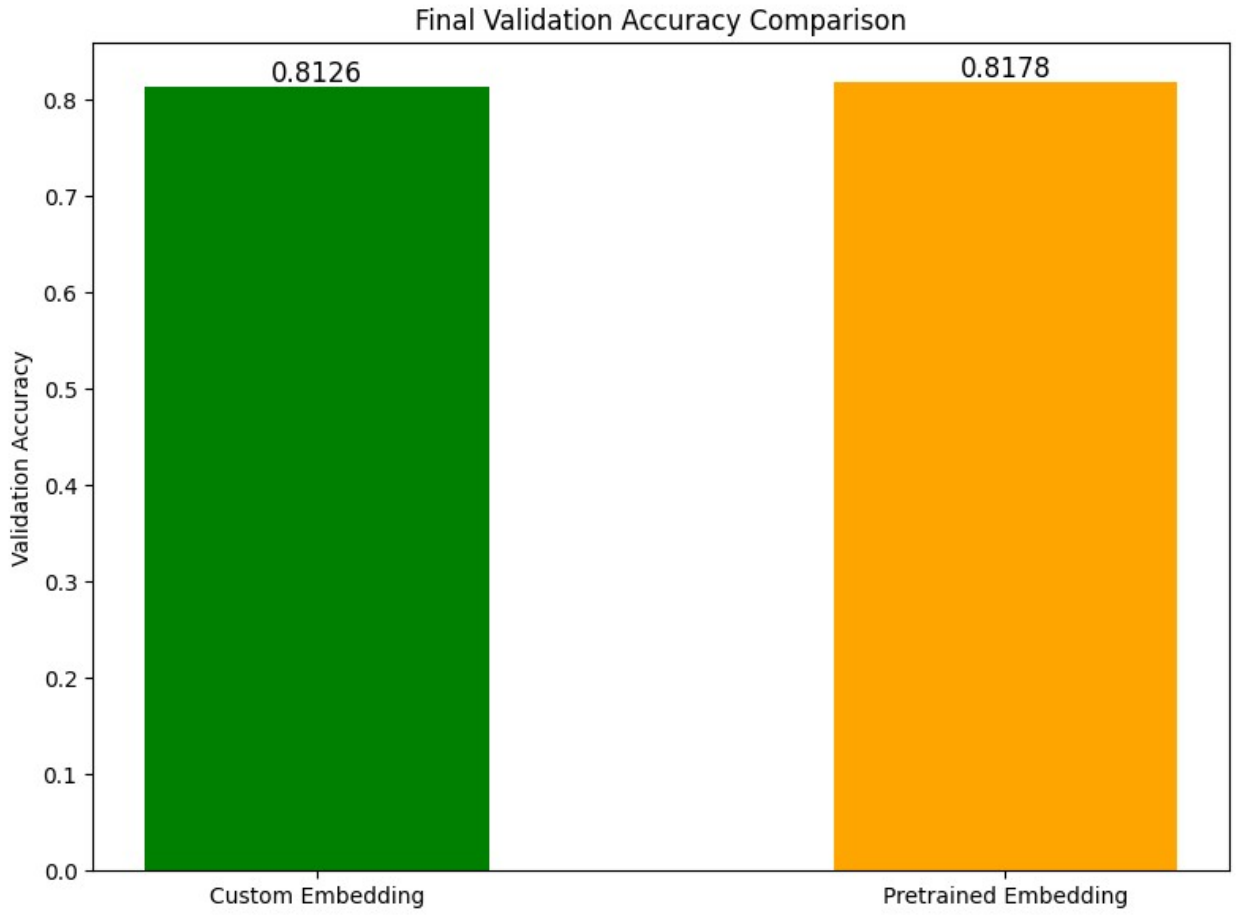

```python
import pandas as pd

# Collect results for custom embedding model
custom_embedding_results = {
    "Sample Size": sample_sizes,
    "Custom Embedding Accuracy": embedding_accuracies,
}

# Collect results for pretrained embedding model
pretrained_embedding_results = {
    "Sample Size": sample_sizes,
    "Pretrained Embedding Accuracy": pretrained_accuracies,
}
```

```python
# Combine both results into a single DataFrame
summary_df = pd.DataFrame({
    "Sample Size": sample_sizes,
    "Custom Embedding Accuracy": embedding_accuracies,
    "Pretrained Embedding Accuracy": pretrained_accuracies
})

# Display the summary of results
print("Summary of Results:")
print(summary_df)

# Optionally save to a CSV file
summary_df.to_csv("embedding_model_accuracy_summary.csv", index=False)
```

```
Summary of Results:
   Sample Size  Custom Embedding Accuracy  Pretrained Embedding
Accuracy
0          100                   0.991250
0.915625
1          200                   0.993906
0.925000
2          500                   0.995250
0.922000
3         1000                   0.999050
0.952350
```

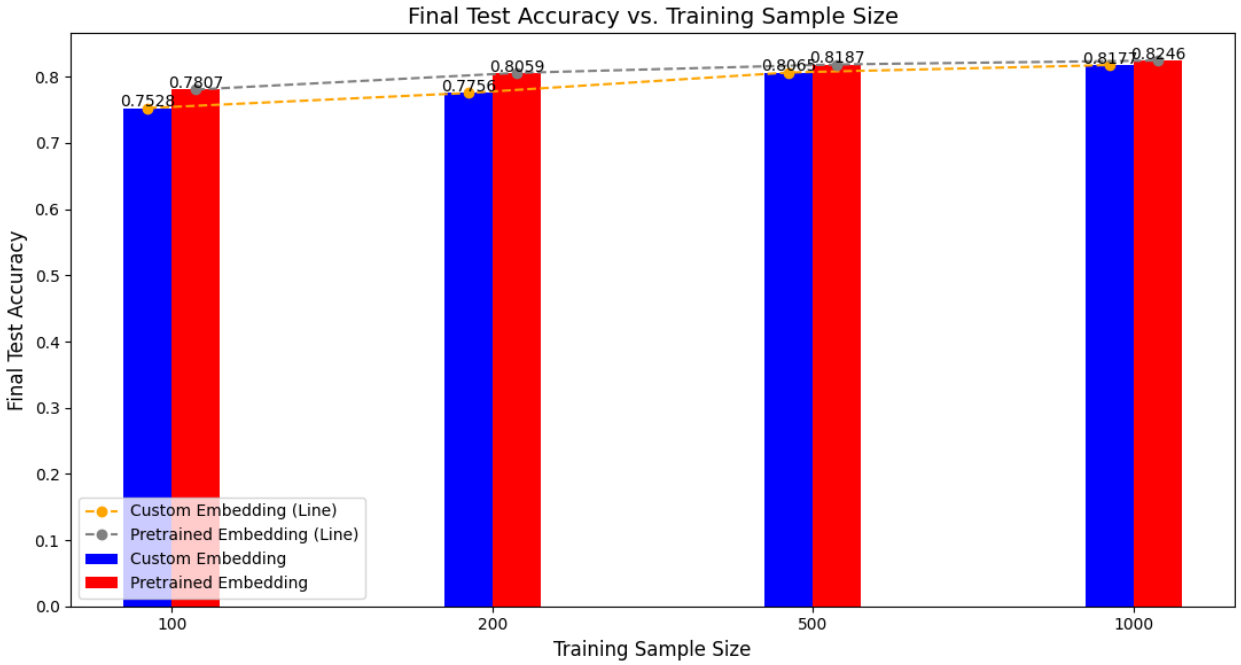
```python
import numpy as np
import matplotlib.pyplot as plt

# Data: Models and their corresponding final validation accuracies
models = ['Custom Embedding', 'Pretrained Embedding']
final_val_accuracies = [
    history_embedding.history['val_accuracy'][-1],  # Final validation
accuracy for custom embedding
    history_pretrained.history['val_accuracy'][-1]  # Final validation
accuracy for pretrained embedding
]

# Bar plot
plt.figure(figsize=(8, 6))
plt.bar(models, final_val_accuracies, color=['green', 'orange'],
width=0.5)

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

# Display final values on top of bars
for i, v in enumerate(final_val_accuracies):
    plt.text(i, v + 0.005, f'{v:.4f}', ha='center', fontsize=12)
```

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




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

# Data
sample_sizes = [100, 200, 500, 1000]
embedding_accuracies = [0.75280, 0.77556, 0.80652, 0.81772]  # Replace
with actual final values
pretrained_accuracies = [0.78072, 0.80588, 0.81868, 0.82456]  #
Replace with actual final values

# Bar width and positions
bar_width = 0.15
x_indices = np.arange(len(sample_sizes))  # Base positions for bars

# Plot bars
plt.figure(figsize=(11, 6))
```

```
plt.bar(x_indices - bar_width / 2, embedding_accuracies,
        width=bar_width, label='Custom Embedding', color='blue')
plt.bar(x_indices + bar_width / 2, pretrained_accuracies,
        width=bar_width, label='Pretrained Embedding', color='red')

# Overlay lines for comparison
plt.plot(x_indices - bar_width / 2, embedding_accuracies,
         marker='o', color='orange', linestyle='--', label='Custom
Embedding (Line)')
plt.plot(x_indices + bar_width / 2, pretrained_accuracies,
         marker='o', color='grey', linestyle='--', label='Pretrained
Embedding (Line)')

# Add labels and title
plt.xlabel('Training Sample Size', fontsize=12)
plt.ylabel('Final Test Accuracy', fontsize=12)
plt.title('Final Test Accuracy vs. Training Sample Size', fontsize=14)
plt.xticks(x_indices, sample_sizes)  # Use sample sizes as x-tick
labels
plt.legend()

# Annotate bars with final accuracy values
for i in range(len(sample_sizes)):
    plt.text(x_indices[i] - bar_width / 2, embedding_accuracies[i] +
0.002,
             f'{embedding_accuracies[i]:.4f}', ha='center',
fontsize=10)
    plt.text(x_indices[i] + bar_width / 2, pretrained_accuracies[i] +
0.002,
             f'{pretrained_accuracies[i]:.4f}', ha='center',
fontsize=10)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```


Final Test Accuracy vs. Training Sample Size
Final Test Accuracy
Training Sample Size
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