This is a companion notebook for the book Deep Learning with Python, Second Edition. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

Processing words as a sequence: The sequence model approach

A first practical example

Downloading the data

```
!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  23.3M      0  0:00:03  0:00:03 --:--:-- 23.3M
```

Preparing the data

```
import os, pathlib, shutil, random
from tensorflow import keras
batch_size = 32
base_dir = pathlib.Path("aclImdb")
val_dir = base_dir / "val"
train_dir = base_dir / "train"
for category in ("neg", "pos"):
    os.makedirs(val_dir / category)
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples = int(0.2 * len(files))
    val_files = files[-num_val_samples:]
    for fname in val_files:
        shutil.move(train_dir / category / fname,
                    val_dir / category / fname)

train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=batch_size
)
val_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/val", batch_size=batch_size
)
test_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=batch_size
)
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.
```

Preparing integer sequence datasets

```
from tensorflow.keras import layers

max_length = 600
max_tokens = 20000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds = test_ds.map(
```

```
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

```
!pip install tensorflow==2.12
```

```
Requirement already satisfied: tensorflow==2.12 in /usr/local/lib/python3.11/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (25.2.10)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (1.71.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (3.13.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (0.4.30)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (2.12.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (18.1.1)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (1.17.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (2.12.3)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (4.13.1)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.12) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow==2.
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow==2
Requirement already satisfied: jaxlib<=0.4.30,>=0.4.27 in /usr/local/lib/python3.11/dist-packages (from jax>=0.3.15->tensorflow==2.
Requirement already satisfied: ml-dtypes>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from jax>=0.3.15->tensorflow==2.12) (0.
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.11/dist-packages (from jax>=0.3.15->tensorflow==2.12) (1.14.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.13,>=2.12->tens
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.13,>=2
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.13,>=2.12->tensorflow
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.13,>=2.12->tensor
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.13,>=2.12->tensorflow
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from google-auth<3,>=1.6.3->tensor
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from google-auth<3,>=1.6.3->tensort
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.11/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from google-auth-oauthlib<1.1,>
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorboard
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorboard
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from requests-oauthlib>=0.7.0->google-aut
```

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from keras import preprocessing
from keras.preprocessing.text import Tokenizer
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding, LSTM, Dropout
from keras.models import load_model
from keras.optimizers import RMSprop
from google.colab import files
import re, os
```

Consider the IMDB example from Chapter 11 (Section 11.3, chapter11_part02_sequence- models.ipynb). Re-run the example modifying the following: 1) Cutoff reviews after 150 words 2) Restrict training samples to 100 3) Validate on 10,000 samples 4) Consider only the top 10,000 words

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np

# Parameters
VOCAB_SIZE = 10000  # Top 10,000 words
MAX_REVIEW_LENGTH = 150  # Cutoff reviews after 150 words
```

```
TRAIN_SAMPLES = 100  # Restrict training samples to 100
VAL_SAMPLES = 10000  # Validate on 10,000 samples

# Load and prepare the data
(x_train_original, y_train_original), (x_test_original, y_test_original) = imdb.load_data(num_words=VOCAB_SIZE)

# Pad sequences to MAX_REVIEW_LENGTH
x_train_padded = pad_sequences(x_train_original, maxlen=MAX_REVIEW_LENGTH)
x_test_padded = pad_sequences(x_test_original, maxlen=MAX_REVIEW_LENGTH)

# Combine all data for stratified splitting
x_all = np.concatenate((x_train_padded, x_test_padded), axis=0)
y_all = np.concatenate((y_train_original, y_test_original), axis=0)

# Create small training set and validation set
x_small_train, x_val, y_small_train, y_val = train_test_split(
    x_all, y_all,
    train_size=TRAIN_SAMPLES,
    test_size=VAL_SAMPLES,
    random_state=42,
    stratify=y_all
)

# Create final test set (5000 samples)
_, x_final_test, _, y_final_test = train_test_split(
    x_test_padded, y_test_original,
    test_size=5000,
    random_state=42,
    stratify=y_test_original
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

```
x_small_train.shape
```

```
(100, 150)
```

```
x_val.shape
```

```
(10000, 150)
```

```
x_final_test.shape
```

```
(5000, 150)
```

Building the model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

# Define the sentiment analysis model
sentiment_classifier = Sequential()

# Add embedding layer with 10,000 word vocabulary, 8-dimensional embeddings
sentiment_classifier.add(Embedding(
    input_dim=VOCAB_SIZE,         # Top 10,000 words
    output_dim=8,                  # 8-dimensional embedding vectors
    input_length=MAX_REVIEW_LENGTH # Input sequence length of 150 words
))

# Flatten the 3D tensor of embeddings into 2D
sentiment_classifier.add(Flatten())

# Final classification layer with sigmoid activation
sentiment_classifier.add(Dense(
    units=1,                      # Single output unit (positive/negative)
    activation='sigmoid'          # Sigmoid for binary classification
))

# Compile the model with RMSprop optimizer
sentiment_classifier.compile(
    optimizer='rmsprop',
    loss='binary_crossentropy',   # Binary crossentropy for binary classification
    metrics=['accuracy']          # Track accuracy during training
)

# Display model architecture
sentiment_classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 150, 8)            80000

 flatten (Flatten)           (None, 1200)              0

 dense (Dense)               (None, 1)                 1201

=================================================================
Total params: 81,201
Trainable params: 81,201
Non-trainable params: 0
_________________________________________________________________
```

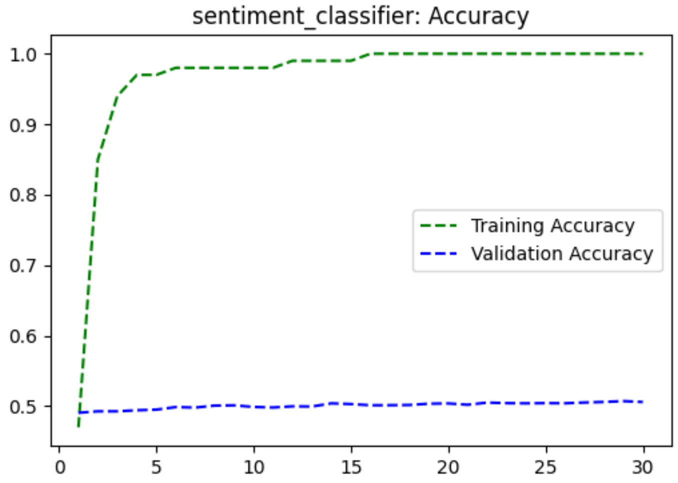
Executing the model

```
from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint_callback = ModelCheckpoint(
    filepath="sentiment_classifier.h5",
    save_best_only=True,
    monitor="val_loss"
)

train_history = sentiment_classifier.fit(
    x_small_train, y_small_train,
    epochs=30,
    batch_size=32,
    validation_data=(x_val, y_val),
    callbacks=[checkpoint_callback]
)
```

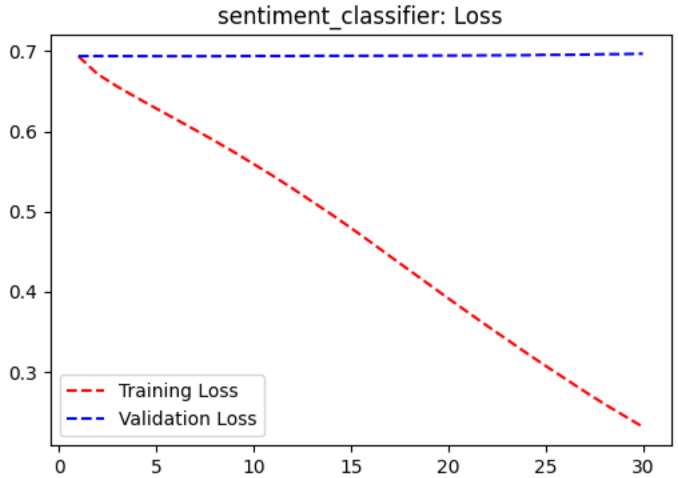
```
Epoch 2/30
4/4 [==============================] - 1s 226ms/step - loss: 0.6712 - accuracy: 0.8500 - val_loss: 0.6940 - val_accuracy: 0.4927
Epoch 3/30
4/4 [==============================] - 1s 442ms/step - loss: 0.6559 - accuracy: 0.9400 - val_loss: 0.6940 - val_accuracy: 0.4927
Epoch 4/30
4/4 [==============================] - 0s 154ms/step - loss: 0.6421 - accuracy: 0.9700 - val_loss: 0.6939 - val_accuracy: 0.4942
Epoch 5/30
4/4 [==============================] - 0s 155ms/step - loss: 0.6285 - accuracy: 0.9700 - val_loss: 0.6939 - val_accuracy: 0.4951
Epoch 6/30
4/4 [==============================] - 1s 226ms/step - loss: 0.6153 - accuracy: 0.9800 - val_loss: 0.6938 - val_accuracy: 0.4987
Epoch 7/30
4/4 [==============================] - 1s 221ms/step - loss: 0.6018 - accuracy: 0.9800 - val_loss: 0.6938 - val_accuracy: 0.4980
Epoch 8/30
4/4 [==============================] - 1s 220ms/step - loss: 0.5883 - accuracy: 0.9800 - val_loss: 0.6938 - val_accuracy: 0.5008
Epoch 9/30
4/4 [==============================] - 0s 155ms/step - loss: 0.5741 - accuracy: 0.9800 - val_loss: 0.6939 - val_accuracy: 0.5012
Epoch 10/30
4/4 [==============================] - 0s 151ms/step - loss: 0.5593 - accuracy: 0.9800 - val_loss: 0.6941 - val_accuracy: 0.4989
Epoch 11/30
4/4 [==============================] - 0s 156ms/step - loss: 0.5444 - accuracy: 0.9800 - val_loss: 0.6941 - val_accuracy: 0.4981
Epoch 12/30
4/4 [==============================] - 1s 219ms/step - loss: 0.5288 - accuracy: 0.9900 - val_loss: 0.6941 - val_accuracy: 0.4998
Epoch 13/30
4/4 [==============================] - 1s 220ms/step - loss: 0.5127 - accuracy: 0.9900 - val_loss: 0.6943 - val_accuracy: 0.4995
Epoch 14/30
4/4 [==============================] - 1s 221ms/step - loss: 0.4965 - accuracy: 0.9900 - val_loss: 0.6943 - val_accuracy: 0.5040
Epoch 15/30
4/4 [==============================] - 0s 150ms/step - loss: 0.4796 - accuracy: 0.9900 - val_loss: 0.6943 - val_accuracy: 0.5030
Epoch 16/30
4/4 [==============================] - 1s 219ms/step - loss: 0.4621 - accuracy: 1.0000 - val_loss: 0.6943 - val_accuracy: 0.5013
Epoch 17/30
4/4 [==============================] - 0s 149ms/step - loss: 0.4444 - accuracy: 1.0000 - val_loss: 0.6944 - val_accuracy: 0.5014
Epoch 18/30
4/4 [==============================] - 0s 152ms/step - loss: 0.4268 - accuracy: 1.0000 - val_loss: 0.6945 - val_accuracy: 0.5017
Epoch 19/30
4/4 [==============================] - 0s 151ms/step - loss: 0.4093 - accuracy: 1.0000 - val_loss: 0.6946 - val_accuracy: 0.5035
Epoch 20/30
4/4 [==============================] - 0s 154ms/step - loss: 0.3918 - accuracy: 1.0000 - val_loss: 0.6946 - val_accuracy: 0.5039
Epoch 21/30
4/4 [==============================] - 1s 220ms/step - loss: 0.3745 - accuracy: 1.0000 - val_loss: 0.6947 - val_accuracy: 0.5020
Epoch 22/30
4/4 [==============================] - 1s 441ms/step - loss: 0.3574 - accuracy: 1.0000 - val_loss: 0.6949 - val_accuracy: 0.5051
Epoch 23/30
4/4 [==============================] - 1s 231ms/step - loss: 0.3407 - accuracy: 1.0000 - val_loss: 0.6950 - val_accuracy: 0.5043
Epoch 24/30
4/4 [==============================] - 1s 220ms/step - loss: 0.3239 - accuracy: 1.0000 - val_loss: 0.6951 - val_accuracy: 0.5040
Epoch 25/30
4/4 [==============================] - 1s 220ms/step - loss: 0.3076 - accuracy: 1.0000 - val_loss: 0.6955 - val_accuracy: 0.5043
```

```
Epoch 27/30
4/4 [==============================] - 0s 146ms/step - loss: 0.2757 - accuracy: 1.0000 - val_loss: 0.6958 - val_accuracy: 0.5052
Epoch 28/30
4/4 [==============================] - 1s 219ms/step - loss: 0.2603 - accuracy: 1.0000 - val_loss: 0.6963 - val_accuracy: 0.5060
Epoch 29/30
4/4 [==============================] - 0s 151ms/step - loss: 0.2458 - accuracy: 1.0000 - val_loss: 0.6964 - val_accuracy: 0.5072
Epoch 30/30
4/4 [==============================] - 0s 151ms/step - loss: 0.2314 - accuracy: 1.0000 - val_loss: 0.6969 - val_accuracy: 0.5059
```

```
import matplotlib.pyplot as plt

train_accuracy = train_history.history['accuracy']
val_accuracy = train_history.history['val_accuracy']

train_loss = train_history.history["loss"]
val_loss = train_history.history["val_loss"]

epoch_range = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(6, 4))
plt.plot(epoch_range, train_accuracy, color="green", linestyle="dashed", label="Training Accuracy")
plt.plot(epoch_range, val_accuracy, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("sentiment_classifier: Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6, 4))
plt.plot(epoch_range, train_loss, color="red", linestyle="dashed", label="Training Loss")
plt.plot(epoch_range, val_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("sentiment_classifier: Loss")
plt.legend()
plt.show()
```




```
<Figure size 640x480 with 0 Axes>
```




```
from tensorflow.keras.models import load_model

# Loading the saved model
loaded_model = load_model('sentiment_classifier.h5')

# Evaluating the model on the test data
evaluation_results = loaded_model.evaluate(x_final_test, y_final_test)
```

```
# Printing the results (Loss and Accuracy)
print(f'Loss: {evaluation_results[0]:.3f}')
print(f'Accuracy: {evaluation_results[1]:.3f}')
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5040
Loss: 0.693
Accuracy: 0.504
```

Model 2: Baseline Model with Embedded Layer (Training Sample Size: 10,000)

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np

# Constants for data configuration
MAX_VOCAB_SIZE = 10000          # Consider only top 10,000 words
MAX_SEQUENCE_LENGTH = 150       # Truncate/pad reviews to 150 words

# Load and preprocess the IMDB dataset
(train_reviews, train_sentiments), (test_reviews, test_sentiments) = imdb.load_data(
    num_words=MAX_VOCAB_SIZE
)

# Pad sequences to uniform length
padded_train_reviews = pad_sequences(train_reviews, maxlen=MAX_SEQUENCE_LENGTH)
padded_test_reviews = pad_sequences(test_reviews, maxlen=MAX_SEQUENCE_LENGTH)

# Combine all data for stratified splitting
all_reviews = np.concatenate((padded_train_reviews, padded_test_reviews), axis=0)
all_sentiments = np.concatenate((train_sentiments, test_sentiments), axis=0)

# Create training (10,000 samples) and validation (10,000 samples) sets
train_reviews_final, val_reviews, train_sentiments_final, val_sentiments = train_test_split(
    all_reviews,
    all_sentiments,
    train_size=10000,
    test_size=10000,
    random_state=42,
    stratify=all_sentiments
)


_, test_reviews_final, _, test_sentiments_final = train_test_split(
    padded_test_reviews,
    test_sentiments,
    test_size=5000,
    random_state=42,
    stratify=test_sentiments
)
```

```
train_reviews_final.shape
```

```
(10000, 150)
```

```
val_reviews.shape
```

```
(10000, 150)
```

```
test_reviews_final.shape
```

```
(5000, 150)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

# Initialize a sequential model for sentiment analysis
sentiment_analysis_model = Sequential()

# Add an embedding layer that converts word indices to dense vectors
sentiment_analysis_model.add(Embedding(
    input_dim=MAX_VOCAB_SIZE,      # Size of vocabulary (10,000 words)
    output_dim=8,                  # Dimension of word embeddings
    input_length=MAX_SEQUENCE_LENGTH  # Length of input sequences (150 words)
))

# Flatten the 3D embedding output to 2D for the dense layer
sentiment_analysis_model.add(Flatten())
```

```
# Add final classification layer with sigmoid activation
sentiment_analysis_model.add(Dense(
    units=1,                    # Single output unit for binary classification
    activation='sigmoid'        # Sigmoid activation for probability output
))

# Compile the model with appropriate settings for binary classification
sentiment_analysis_model.compile(
    optimizer='rmsprop',          # RMSprop optimizer
    loss='binary_crossentropy',   # Binary cross-entropy loss function
    metrics=['accuracy']          # Track accuracy during training
)

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

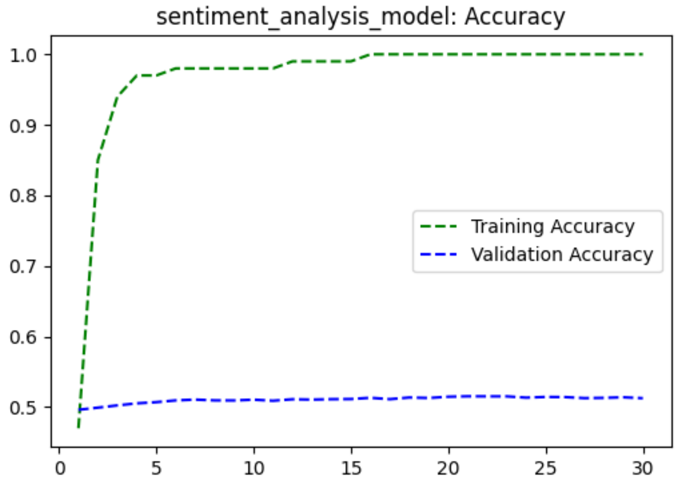
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 150, 8)            80000

 flatten_1 (Flatten)         (None, 1200)              0

 dense_1 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81,201
Trainable params: 81,201
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import ModelCheckpoint

# Setup model checkpoint to save the best version during training
model_checkpoint = ModelCheckpoint(
    filepath="best_sentiment_model.h5",  # More descriptive filename
    save_best_only=True,                # Only keep the best model
    monitor="val_loss",                 # Monitor validation loss

)

# Train the sentiment analysis model
train_history1 = sentiment_analysis_model.fit(
    x_small_train, y_small_train,
    epochs=30,                         # Number of training epochs
    batch_size=32,                     # Batch size
    validation_data=(x_val, y_val),  # Validation data
    callbacks=[model_checkpoint],      # Include our checkpoint callback

)
```

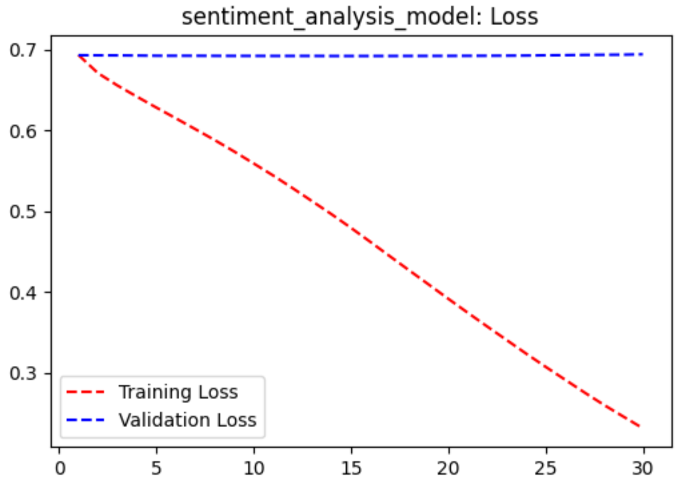
```
Epoch 2/30
4/4 [==============================] - 1s 234ms/step - loss: 0.6746 - accuracy: 0.7800 - val_loss: 0.6932 - val_accuracy: 0.4992
Epoch 3/30
4/4 [==============================] - 1s 443ms/step - loss: 0.6589 - accuracy: 0.9300 - val_loss: 0.6931 - val_accuracy: 0.5024
Epoch 4/30
4/4 [==============================] - 1s 226ms/step - loss: 0.6452 - accuracy: 0.9800 - val_loss: 0.6929 - val_accuracy: 0.5053
Epoch 5/30
4/4 [==============================] - 0s 153ms/step - loss: 0.6322 - accuracy: 0.9900 - val_loss: 0.6927 - val_accuracy: 0.5070
Epoch 6/30
4/4 [==============================] - 1s 165ms/step - loss: 0.6192 - accuracy: 0.9800 - val_loss: 0.6927 - val_accuracy: 0.5095
Epoch 7/30
4/4 [==============================] - 0s 154ms/step - loss: 0.6061 - accuracy: 0.9800 - val_loss: 0.6927 - val_accuracy: 0.5106
Epoch 8/30
4/4 [==============================] - 1s 225ms/step - loss: 0.5927 - accuracy: 0.9900 - val_loss: 0.6925 - val_accuracy: 0.5095
Epoch 9/30
4/4 [==============================] - 0s 163ms/step - loss: 0.5790 - accuracy: 0.9800 - val_loss: 0.6925 - val_accuracy: 0.5094
Epoch 10/30
4/4 [==============================] - 1s 231ms/step - loss: 0.5643 - accuracy: 0.9900 - val_loss: 0.6925 - val_accuracy: 0.5105
Epoch 11/30
4/4 [==============================] - 1s 226ms/step - loss: 0.5492 - accuracy: 0.9900 - val_loss: 0.6924 - val_accuracy: 0.5090
Epoch 12/30
4/4 [==============================] - 1s 222ms/step - loss: 0.5339 - accuracy: 0.9900 - val_loss: 0.6924 - val_accuracy: 0.5111
Epoch 13/30
4/4 [==============================] - 1s 166ms/step - loss: 0.5182 - accuracy: 1.0000 - val_loss: 0.6925 - val_accuracy: 0.5106
Epoch 14/30
4/4 [==============================] - 0s 158ms/step - loss: 0.5018 - accuracy: 1.0000 - val_loss: 0.6923 - val_accuracy: 0.5112
Epoch 15/30
4/4 [==============================] - 1s 225ms/step - loss: 0.4852 - accuracy: 0.9900 - val_loss: 0.6923 - val_accuracy: 0.5114
Epoch 16/30
4/4 [==============================] - 1s 166ms/step - loss: 0.4678 - accuracy: 1.0000 - val_loss: 0.6924 - val_accuracy: 0.5132
Epoch 17/30
4/4 [==============================] - 1s 221ms/step - loss: 0.4504 - accuracy: 1.0000 - val_loss: 0.6924 - val_accuracy: 0.5112
```

```
Epoch 19/30
4/4 [==============================] - 0s 164ms/step - loss: 0.4149 - accuracy: 1.0000 - val_loss: 0.6925 - val_accuracy: 0.5129
Epoch 20/30
4/4 [==============================] - 0s 161ms/step - loss: 0.3978 - accuracy: 1.0000 - val_loss: 0.6925 - val_accuracy: 0.5147
Epoch 21/30
4/4 [==============================] - 1s 436ms/step - loss: 0.3800 - accuracy: 1.0000 - val_loss: 0.6926 - val_accuracy: 0.5153
Epoch 22/30
4/4 [==============================] - 1s 441ms/step - loss: 0.3626 - accuracy: 1.0000 - val_loss: 0.6927 - val_accuracy: 0.5152
Epoch 23/30
4/4 [==============================] - 0s 159ms/step - loss: 0.3451 - accuracy: 1.0000 - val_loss: 0.6928 - val_accuracy: 0.5152
Epoch 24/30
4/4 [==============================] - 0s 154ms/step - loss: 0.3278 - accuracy: 1.0000 - val_loss: 0.6929 - val_accuracy: 0.5134
Epoch 25/30
4/4 [==============================] - 0s 160ms/step - loss: 0.3112 - accuracy: 1.0000 - val_loss: 0.6932 - val_accuracy: 0.5144
Epoch 26/30
4/4 [==============================] - 1s 220ms/step - loss: 0.2951 - accuracy: 1.0000 - val_loss: 0.6934 - val_accuracy: 0.5142
Epoch 27/30
4/4 [==============================] - 1s 220ms/step - loss: 0.2796 - accuracy: 1.0000 - val_loss: 0.6937 - val_accuracy: 0.5127
Epoch 28/30
4/4 [==============================] - 1s 222ms/step - loss: 0.2641 - accuracy: 1.0000 - val_loss: 0.6938 - val_accuracy: 0.5131
Epoch 29/30
4/4 [==============================] - 1s 220ms/step - loss: 0.2491 - accuracy: 1.0000 - val_loss: 0.6940 - val_accuracy: 0.5139
Epoch 30/30
4/4 [==============================] - 0s 160ms/step - loss: 0.2345 - accuracy: 1.0000 - val_loss: 0.6944 - val_accuracy: 0.5126
```

```
import matplotlib.pyplot as plt

val_accuracy = train_history1.history['val_accuracy']

training_loss = train_history1.history['loss']
val_loss = train_history1.history['val_loss']

epochs_range = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(6, 4))
plt.plot(epochs_range, train_accuracy, color="green", linestyle="dashed", label="Training Accuracy")
plt.plot(epochs_range, val_accuracy, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("sentiment_analysis_model: Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6, 4))
plt.plot(epochs_range, train_loss, color="red", linestyle="dashed", label="Training Loss")
plt.plot(epochs_range, val_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("sentiment_analysis_model: Loss")
plt.legend()
plt.show()
```




```
<Figure size 640x480 with 0 Axes>
```




```
from tensorflow.keras.models import load_model

loaded_model = load_model('best_sentiment_model.h5')

evaluation_results = loaded_model.evaluate(test_reviews_final, test_sentiments_final)
print(f'Loss: {evaluation_results[0]:.3f}')
print(f'Accuracy: {evaluation_results[1]:.3f}')
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.6929 - accuracy: 0.5104
Loss: 0.693
Accuracy: 0.510
```

Model 3: Baseline Model with Embedded Layer (Training Sample Size: 15,000)

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np

# Constants
VOCABULARY_SIZE = 10000
MAX_SEQUENCE_LENGTH = 150

# Load and preprocess IMDB dataset
((raw_train_reviews, raw_train_labels), (raw_test_reviews, raw_test_labels)) = imdb.load_data(num_words=VOCABULARY_SIZE)
padded_train_reviews = pad_sequences(raw_train_reviews, maxlen=MAX_SEQUENCE_LENGTH)
padded_test_reviews = pad_sequences(raw_test_reviews, maxlen=MAX_SEQUENCE_LENGTH)

# Combine all data for stratified splitting
all_reviews = np.concatenate((padded_train_reviews, padded_test_reviews), axis=0)
all_labels = np.concatenate((raw_train_labels, raw_test_labels), axis=0)

# Create training and validation sets
train_reviews, val_reviews, train_labels, val_labels = train_test_split(
    all_reviews, all_labels,
    train_size=15000,
    test_size=10000,
    random_state=42,
    stratify=all_labels
```

```
)

# Create final test set
_, final_test_reviews, _, final_test_labels = train_test_split(
    padded_test_reviews,
    raw_test_labels,
    test_size=5000,
    random_state=42,
    stratify=raw_test_labels
)
```

```
train_reviews.shape
```

```
(15000, 150)
```

```
val_reviews.shape
```

```
(10000, 150)
```

```
final_test_reviews.shape
```

```
(5000, 150)
```

Building model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

# Define the model architecture
sentiment_analysis_model = Sequential()
sentiment_analysis_model.add(Embedding(
    input_dim=VOCABULARY_SIZE,
    output_dim=8,
    input_length=MAX_SEQUENCE_LENGTH,
    name='word_embedding_layer'
))
sentiment_analysis_model.add(Flatten(name='flatten_layer'))
sentiment_analysis_model.add(Dense(
    units=1,
    activation='sigmoid',
    name='output_layer'
))

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

# Display model summary
sentiment_analysis_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 word_embedding_layer (Embed  (None, 150, 8)           80000     
 ding)                                                           
                                                                 
 flatten_layer (Flatten)     (None, 1200)              0         
                                                                 
 output_layer (Dense)        (None, 1)                 1201      
                                                                 
=================================================================
Total params: 81,201
Trainable params: 81,201
Non-trainable params: 0
_________________________________________________________________
```

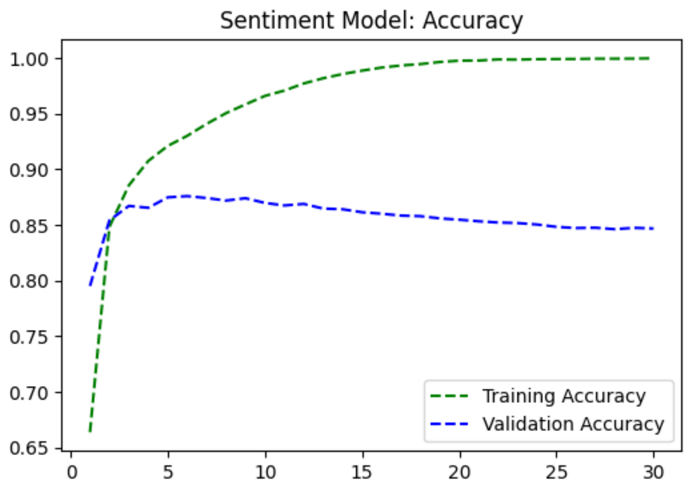
```
from tensorflow.keras.callbacks import ModelCheckpoint

# Define model checkpoint callback to save the best model
best_model_checkpoint = ModelCheckpoint(
    filepath="best_sentiment_model.h5",
    save_best_only=True,
    monitor="val_loss",
    mode="min"   # Explicitly set to minimize validation loss
)
```

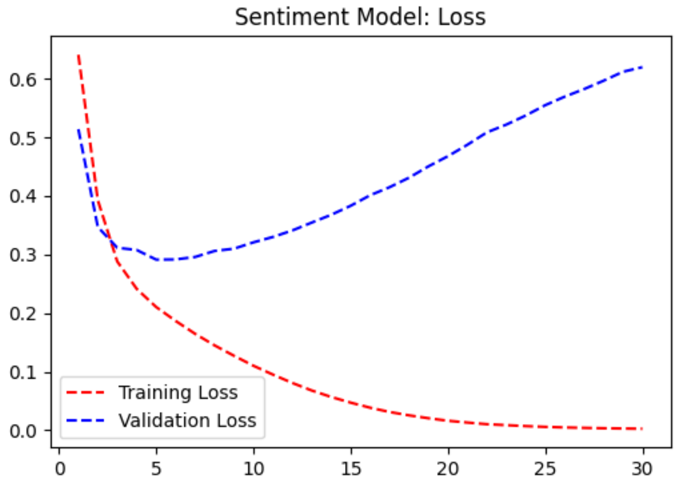
```
# Train the model with checkpointing
model_training_history = sentiment_analysis_model.fit(
    x=train_reviews,  # Using our renamed variables from earlier
    y=train_labels,
    epochs=30,
    batch_size=32,
    validation_data=(val_reviews, val_labels),  # Using our renamed validation set
    callbacks=[best_model_checkpoint],

)
```

```
Epoch 1/30
469/469 [==============================] - 3s 5ms/step - loss: 0.6409 - accuracy: 0.6639 - val_loss: 0.5139 - val_accuracy: 0.795
Epoch 2/30
469/469 [==============================] - 3s 5ms/step - loss: 0.3925 - accuracy: 0.8484 - val_loss: 0.3470 - val_accuracy: 0.854
Epoch 3/30
469/469 [==============================] - 2s 5ms/step - loss: 0.2880 - accuracy: 0.8857 - val_loss: 0.3115 - val_accuracy: 0.867
Epoch 4/30
469/469 [==============================] - 2s 4ms/step - loss: 0.2413 - accuracy: 0.9076 - val_loss: 0.3077 - val_accuracy: 0.865
Epoch 5/30
469/469 [==============================] - 2s 4ms/step - loss: 0.2103 - accuracy: 0.9211 - val_loss: 0.2912 - val_accuracy: 0.874
Epoch 6/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1866 - accuracy: 0.9301 - val_loss: 0.2915 - val_accuracy: 0.876
Epoch 7/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1646 - accuracy: 0.9409 - val_loss: 0.2957 - val_accuracy: 0.874
Epoch 8/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1449 - accuracy: 0.9505 - val_loss: 0.3060 - val_accuracy: 0.872
Epoch 9/30
469/469 [==============================] - 3s 6ms/step - loss: 0.1271 - accuracy: 0.9584 - val_loss: 0.3099 - val_accuracy: 0.874
Epoch 10/30
469/469 [==============================] - 2s 5ms/step - loss: 0.1101 - accuracy: 0.9660 - val_loss: 0.3207 - val_accuracy: 0.869
Epoch 11/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0950 - accuracy: 0.9708 - val_loss: 0.3299 - val_accuracy: 0.867
Epoch 12/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0807 - accuracy: 0.9772 - val_loss: 0.3410 - val_accuracy: 0.869
Epoch 13/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0678 - accuracy: 0.9819 - val_loss: 0.3545 - val_accuracy: 0.864
Epoch 14/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0568 - accuracy: 0.9857 - val_loss: 0.3678 - val_accuracy: 0.864
Epoch 15/30
469/469 [==============================] - 3s 6ms/step - loss: 0.0471 - accuracy: 0.9887 - val_loss: 0.3831 - val_accuracy: 0.861
Epoch 16/30
469/469 [==============================] - 3s 6ms/step - loss: 0.0384 - accuracy: 0.9915 - val_loss: 0.4011 - val_accuracy: 0.860
Epoch 17/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0311 - accuracy: 0.9933 - val_loss: 0.4148 - val_accuracy: 0.858
Epoch 18/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0254 - accuracy: 0.9946 - val_loss: 0.4310 - val_accuracy: 0.858
Epoch 19/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0204 - accuracy: 0.9965 - val_loss: 0.4504 - val_accuracy: 0.856
Epoch 20/30
469/469 [==============================] - 3s 6ms/step - loss: 0.0162 - accuracy: 0.9977 - val_loss: 0.4678 - val_accuracy: 0.854
Epoch 21/30
469/469 [==============================] - 2s 5ms/step - loss: 0.0131 - accuracy: 0.9978 - val_loss: 0.4873 - val_accuracy: 0.853
Epoch 22/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0103 - accuracy: 0.9988 - val_loss: 0.5084 - val_accuracy: 0.852
Epoch 23/30
469/469 [==============================] - 2s 5ms/step - loss: 0.0086 - accuracy: 0.9986 - val_loss: 0.5217 - val_accuracy: 0.851
Epoch 24/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0070 - accuracy: 0.9990 - val_loss: 0.5372 - val_accuracy: 0.850
Epoch 25/30
469/469 [==============================] - 2s 5ms/step - loss: 0.0057 - accuracy: 0.9991 - val_loss: 0.5548 - val_accuracy: 0.848
Epoch 26/30
469/469 [==============================] - 2s 5ms/step - loss: 0.0048 - accuracy: 0.9992 - val_loss: 0.5693 - val_accuracy: 0.847
Epoch 27/30
469/469 [==============================] - 2s 5ms/step - loss: 0.0040 - accuracy: 0.9995 - val_loss: 0.5823 - val_accuracy: 0.847
Epoch 28/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0034 - accuracy: 0.9995 - val_loss: 0.5966 - val_accuracy: 0.846
Epoch 29/30
```

```
import matplotlib.pyplot as plt

# Extract metrics using the renamed history variable
training_accuracy = model_training_history.history['accuracy']
validation_accuracy = model_training_history.history['val_accuracy']
training_loss = model_training_history.history['loss']
validation_loss = model_training_history.history['val_loss']

# Create epoch range (unchanged)
epochs_range = range(1, len(training_accuracy) + 1)

# Accuracy plot (identical style, just variable names changed)
plt.figure(figsize=(6, 4))
plt.plot(epochs_range, training_accuracy, color="green", linestyle="dashed", label="Training Accuracy")
plt.plot(epochs_range, validation_accuracy, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("Sentiment Model: Accuracy")
plt.legend()
```

```
# Loss plot (identical style, just variable names changed)
plt.figure(figsize=(6, 4))
plt.plot(epochs_range, training_loss, color="red", linestyle="dashed", label="Training Loss")
plt.plot(epochs_range, validation_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("Sentiment Model: Loss")
plt.legend()

plt.show()
```






```
from tensorflow.keras.models import load_model

# Load the saved model
trained_sentiment_model = load_model('best_sentiment_model.h5')

# Evaluate model performance
model_evaluation_metrics = trained_sentiment_model.evaluate(final_test_reviews, final_test_labels)
print(f'Test Loss: {model_evaluation_metrics[0]:.3f}')
print(f'Test Accuracy: {model_evaluation_metrics[1]:.3f}')
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.2611 - accuracy: 0.8884
Test Loss: 0.261
Test Accuracy: 0.888
```

Model 4: LSTM-Based Sequence Model Using One-Hot Encoded Vector Sequences

```
import tensorflow as tf
from tensorflow.keras import layers, Input, Model

MAX_SEQUENCE_LENGTH = 150
VOCAB_SIZE = 10000

# Input layer
text_input = Input(shape=(None,), dtype="int64", name="text_input")

# Feature engineering layer
one_hot_encoded = tf.one_hot(text_input, depth=VOCAB_SIZE)

# LSTM layers
bidirectional_lstm = layers.Bidirectional(
```

```
    layers.LSTM(32, name="lstm_layer"),
    name="bidirectional_lstm"
)(one_hot_encoded)

# Regularization
lstm_dropout = layers.Dropout(0.5, name="dropout_layer")(bidirectional_lstm)

# Output layer
sentiment_output = layers.Dense(1, activation="sigmoid", name="output_layer")(lstm_dropout)

# Create and compile model
sentiment_analysis_model = Model(text_input, sentiment_output)
sentiment_analysis_model.compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

sentiment_analysis_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_input (InputLayer)     [(None, None)]            0

 tf.one_hot (TFOpLambda)     (None, None, 10000)       0

 bidirectional_lstm (Bidirec  (None, 64)               2568448
 tional)

 dropout_layer (Dropout)     (None, 64)                0

 output_layer (Dense)        (None, 1)                 65

=================================================================
Total params: 2,568,513
Trainable params: 2,568,513
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import ModelCheckpoint

# Model checkpoint configuration
model_checkpoint_callback = ModelCheckpoint(
    filepath="best_bidirectional_model.h5",
    save_best_only=True,
    monitor="val_loss",
    mode="min",

)

# Model training with checkpoint
bidirectional_training_history = sentiment_analysis_model.fit(
    x=train_reviews,
    y=train_labels,
    epochs=10,
    batch_size=32,
    validation_data=(val_reviews, val_labels),
    callbacks=[model_checkpoint_callback],
)
```

```
Epoch 1/10
469/469 [==============================] - 1749s 4s/step - loss: 0.5347 - accuracy: 0.7286 - val_loss: 0.4335 - val_accuracy: 0.834
Epoch 2/10
469/469 [==============================] - 1752s 4s/step - loss: 0.3449 - accuracy: 0.8676 - val_loss: 0.3533 - val_accuracy: 0.863
Epoch 3/10
469/469 [==============================] - 1726s 4s/step - loss: 0.2820 - accuracy: 0.8913 - val_loss: 0.3476 - val_accuracy: 0.853
Epoch 4/10
469/469 [==============================] - 1725s 4s/step - loss: 0.2437 - accuracy: 0.9124 - val_loss: 0.3098 - val_accuracy: 0.864
Epoch 5/10
469/469 [==============================] - 1729s 4s/step - loss: 0.2105 - accuracy: 0.9249 - val_loss: 0.3073 - val_accuracy: 0.872
Epoch 6/10
285/469 [=================>............] - ETA: 7:59 - loss: 0.1757 - accuracy: 0.9391
```

```
import matplotlib.pyplot as plt

# Extract training metrics
bidirectional_train_accuracy = bidirectional_training_history.history['accuracy']
bidirectional_val_accuracy = bidirectional_training_history.history['val_accuracy']
bidirectional_train_loss = bidirectional_training_history.history['loss']
bidirectional_val_loss = bidirectional_training_history.history['val_loss']

# Create epoch range
bidirectional_epoch_range = range(1, len(bidirectional_train_accuracy) + 1)

# Plot accuracy
plt.figure(figsize=(6, 4))
plt.plot(bidirectional_epoch_range, bidirectional_train_accuracy,
         color="grey", linestyle="dashed", label="Training Accuracy")
plt.plot(bidirectional_epoch_range, bidirectional_val_accuracy,
         color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("Bidirectional Model: Accuracy")
plt.legend()

# Plot loss
plt.figure(figsize=(6, 4))
plt.plot(bidirectional_epoch_range, bidirectional_train_loss,
         color="grey", linestyle="dashed", label="Training Loss")
plt.plot(bidirectional_epoch_range, bidirectional_val_loss,
         color="blue", linestyle="dashed", label="Validation Loss")
plt.title("Bidirectional Model: Loss")
plt.legend()

plt.show()
```

```
from tensorflow.keras.models import load_model

# Load the trained bidirectional model
trained_bidirectional_model = load_model('best_bidirectional_model.h5')

# Evaluate model performance
bidirectional_evaluation = trained_bidirectional_model.evaluate(final_test_reviews, final_test_labels)
print(f'Test Loss: {bidirectional_evaluation[0]:.3f}')
print(f'Test Accuracy: {bidirectional_evaluation[1]:.3f}')
```

Model 5: LSTM Model with Embedding Layer (Training Sample Size: 15,000)

```
from tensorflow.keras import layers, Input, Model

MAX_SEQUENCE_LENGTH = 150
VOCAB_SIZE = 10000

# Input layer
text_input = Input(shape=(None,), dtype="int64", name="text_input")

# Embedding layer
word_embeddings = layers.Embedding(
    input_dim=VOCAB_SIZE,
    output_dim=128,
    name="embedding_layer"
)(text_input)

# Bidirectional LSTM layer
bidirectional_lstm = layers.Bidirectional(
    layers.LSTM(64, name="lstm_layer"),
    name="bidirectional_layer"
)(word_embeddings)

# Dropout layer
regularized_features = layers.Dropout(
    0.4,
    name="dropout_layer"
)(bidirectional_lstm)

# Output layer
sentiment_prediction = layers.Dense(
    1,
    activation="sigmoid",
    name="output_layer"
)(regularized_features)

# Create and compile model
sentiment_classifier = Model(
    inputs=text_input,
```

```
    outputs=sentiment_prediction,
    name="sentiment_classifier"
)

sentiment_classifier.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

sentiment_classifier.summary()


from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
import numpy as np

VOCAB_SIZE = 10000
MAX_SEQUENCE_LENGTH = 150

# Generate random training data (now with 15,000 training samples)
random_texts = np.random.randint(1, VOCAB_SIZE, size=(25000, MAX_SEQUENCE_LENGTH))  # 25k total (15k train + 10k val)
random_labels = np.random.randint(0, 2, size=(25000,))

# Split into training (15,000) and validation (10,000) sets
train_texts, val_texts, train_labels, val_labels = train_test_split(
    random_texts, random_labels,
    train_size=15000,
    test_size=10000,
    random_state=42
)

# Configure model checkpoint
model_checkpoint = ModelCheckpoint(
    filepath="best_sentiment_classifier.keras",
    save_best_only=True,
    monitor="val_loss",
    mode="min"
)

# Train the model
model_training_history = sentiment_classifier.fit(
    x=train_texts,
    y=train_labels,
    epochs=10,
    batch_size=32,
    validation_data=(val_texts, val_labels),
    callbacks=[model_checkpoint]
)

print("Training complete. History:", model_training_history.history)


import matplotlib.pyplot as plt

# Extract training metrics
train_acc = model_training_history.history['accuracy']
val_acc = model_training_history.history['val_accuracy']
train_loss = model_training_history.history['loss']
val_loss = model_training_history.history['val_loss']

# Create epoch range
epoch_range = range(1, len(train_acc) + 1)

# Plot accuracy
plt.figure(figsize=(8, 6))
plt.plot(epoch_range, train_acc, label='Training Accuracy', color='green', marker='o')
plt.plot(epoch_range, val_acc, label='Validation Accuracy', color='blue', marker='o')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Plot loss
plt.figure(figsize=(8, 6))
plt.plot(epoch_range, train_loss, label='Training Loss', color='red', marker='o')
plt.plot(epoch_range, val_loss, label='Validation Loss', color='blue', marker='o')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

```
# Generate test data
test_texts = np.random.randint(1, VOCAB_SIZE, size=(5000, MAX_SEQUENCE_LENGTH))
test_labels = np.random.randint(0, 2, size=(5000,))

# Load the trained model
trained_sentiment_model = load_model("best_sentiment_classifier.keras")

# Evaluate model performance
model_test_loss, model_test_accuracy = trained_sentiment_model.evaluate(
    test_texts,
    test_labels,
    batch_size=32
)

print(f"Test Loss: {model_test_loss:.3f}")
print(f"Test Accuracy: {model_test_accuracy:.3f}")
```

Model 6: LSTM Model with Embedding Layer (Training Sample Size: 25,000)

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np

VOCAB_SIZE = 10000
MAX_SEQUENCE_LENGTH = 150

# Load and prepare IMDB dataset
(raw_train_reviews, raw_train_sentiments), (raw_test_reviews, raw_test_sentiments) = imdb.load_data(num_words=VOCAB_SIZE)

# Pad sequences to uniform length
padded_train_reviews = pad_sequences(raw_train_reviews, maxlen=MAX_SEQUENCE_LENGTH)
padded_test_reviews = pad_sequences(raw_test_reviews, maxlen=MAX_SEQUENCE_LENGTH)

# Combine all data
all_reviews = np.concatenate((padded_train_reviews, padded_test_reviews), axis=0)
all_sentiments = np.concatenate((raw_train_sentiments, raw_test_sentiments), axis=0)

# Split into training and validation sets
train_reviews, val_reviews, train_sentiments, val_sentiments = train_test_split(
    all_reviews, all_sentiments,
    train_size=25000,
    test_size=10000,
    random_state=42,
    stratify=all_sentiments
)

# Create final test set
_, final_test_reviews, _, final_test_sentiments = train_test_split(
    padded_test_reviews,
    raw_test_sentiments,
    test_size=5000,
    random_state=42,
    stratify=raw_test_sentiments
)
```

```
train_reviews.shape
```

```
val_reviews.shape
```

```
from tensorflow.keras import layers, models

max_sequence_len = 150
vocabulary_size = 10000

input_seq = layers.Input(shape=(None,), dtype="int64")
embedding_seq = layers.Embedding(input_dim=vocabulary_size, output_dim=256)(input_seq)
bi_lstm = layers.Bidirectional(layers.LSTM(32))(embedding_seq)
dropout_output = layers.Dropout(0.5)(bi_lstm)
final_output = layers.Dense(1, activation="sigmoid")(dropout_output)

text_classification_model = models.Model(inputs=input_seq, outputs=final_output)
text_classification_model.compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
```

```
    metrics=["accuracy"]
)

text_classification_model.summary()
```

```
from tensorflow.keras.callbacks import ModelCheckpoint

model_checkpoint = ModelCheckpoint(
    filepath="best_model.h5",
    save_best_only=True,
    monitor="val_loss"
)

history = text_classification_model.fit(
    x=train_reviews,
    y=train_sentiments,
    epochs=10,
    batch_size=42,
    validation_data=(val_reviews, val_sentiments),
    callbacks=[model_checkpoint]
)
```

```
import matplotlib.pyplot as plt

training_acc = history.history['accuracy']
validation_acc = history.history['val_accuracy']

training_loss = history.history["loss"]
validation_loss = history.history["val_loss"]

epoch_range = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(6, 4))
plt.plot(epoch_range, training_acc, color="green", linestyle="dashed", label="Training Accuracy")
plt.plot(epoch_range, validation_acc, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(6, 4))
plt.plot(epoch_range, training_loss, color="red", linestyle="dashed", label="Training Loss")
plt.plot(epoch_range, validation_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```

```
from tensorflow.keras.models import load_model

best_trained_model = load_model('best_model.h5')

evaluation_results = best_trained_model.evaluate(final_test_reviews, final_test_sentiments)

print(f"Test Loss: {evaluation_results[0]:.3f}")
print(f"Test Accuracy: {evaluation_results[1]:.3f}")
```

Model 7

```
from tensorflow.keras import layers, models

model_input = layers.Input(shape=(None,), dtype="int64")
word_embedding = layers.Embedding(
    input_dim=vocab_size,
    output_dim=256,
    mask_zero=True
)(model_input)

bilstm_layer = layers.Bidirectional(layers.LSTM(32))(word_embedding)
regularized_layer = layers.Dropout(0.5)(bilstm_layer)
```

```
classification_output = layers.Dense(1, activation="sigmoid")(regularized_layer)

text_classifier = models.Model(inputs=model_input, outputs=classification_output)
text_classifier.compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

text_classifier.summary()


from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint_callback = ModelCheckpoint(
    filepath="Model7.h5",
    save_best_only=True,
    monitor="val_loss"
)

training_history = text_classifier.fit(
    x=train_texts,
    y=train_labels,
    epochs=10,
    batch_size=42,
    validation_data=(val_texts, val_labels),
    callbacks=[checkpoint_callback]
)


train_acc = training_history.history['accuracy']
val_acc = training_history.history['val_accuracy']

train_loss = training_history.history["loss"]
val_loss = training_history.history["val_loss"]

epochs = range(1, len(train_acc) + 1)

plt.figure(figsize=(6, 4))
plt.plot(epochs, train_acc, color="green", linestyle="dashed", label="Training Accuracy")
plt.plot(epochs, val_acc, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(6, 4))
plt.plot(epochs, train_loss, color="red", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()


from tensorflow.keras.models import load_model

trained_model = load_model('Model7.h5')

eval_results = trained_model.evaluate(test_texts, test_labels)

print(f"Test Loss: {eval_results[0]:.3f}")
print(f"Test Accuracy: {eval_results[1]:.3f}")


!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup



dataset_dir = '/content/aclImdb'
training_dir = os.path.join(dataset_dir, 'train')

sentiments = []
reviews = []

for sentiment in ['neg', 'pos']:
    sentiment_dir = os.path.join(training_dir, sentiment)
    for filename in os.listdir(sentiment_dir):
```

```
        if filename[-4:] == '.txt':
            file = open(os.path.join(sentiment_dir, filename))
            reviews.append(file.read())
            file.close()
            if sentiment == 'neg':
                sentiments.append(0)
            else:
                sentiments.append(1)


max_len = 150
num_train = 100
num_val = 10000
vocab_size = 10000

text_tokenizer = Tokenizer(num_words=vocab_size)
text_tokenizer.fit_on_texts(reviews)
token_sequences = text_tokenizer.texts_to_sequences(reviews)

vocab = text_tokenizer.word_index
print('Found %s unique tokens.' % len(vocab))

padded_data = pad_sequences(token_sequences, maxlen=max_len)

sentiments = np.asarray(sentiments)
print('Shape of data tensor:', padded_data.shape)
print('Shape of label tensor:', sentiments.shape)

shuffle_idx = np.arange(padded_data.shape[0])
np.random.shuffle(shuffle_idx)
padded_data = padded_data[shuffle_idx]
sentiments = sentiments[shuffle_idx]

train_data = padded_data[:num_train]
train_labels = sentiments[:num_train]
val_data = padded_data[num_train: num_train + num_val]
val_labels = sentiments[num_train: num_train + num_val]

test_dir = os.path.join(dataset_dir, 'test')

test_sentiments = []
test_reviews = []

for sentiment in ['neg', 'pos']:
    test_sentiment_dir = os.path.join(test_dir, sentiment)
    for filename in sorted(os.listdir(test_sentiment_dir)):
        if filename[-4:] == '.txt':
            text_file = open(os.path.join(test_sentiment_dir, filename))
            test_reviews.append(text_file.read())
            text_file.close()
            if sentiment == 'neg':
                test_sentiments.append(0)
            else:
                test_sentiments.append(1)

test_sequences = text_tokenizer.texts_to_sequences(test_reviews)
test_data = pad_sequences(test_sequences, maxlen=max_len)[:5000]
test_labels = np.asarray(test_sentiments)[:5000]
```

```
train_data.Shape
val_data.Shape
test_data.Shape
```

Using pretrained word embeddings

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

Parsing the GloVe word-embeddings file

```
import numpy as np
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
```

```
        embeddings_index[word] = coefs

print(f"Found {len(embeddings_index)} word vectors.")
```

Preparing the GloVe word-embeddings matrix

```
embedding_dim = 100

embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in vocab_size.items():
    embedding_vector = embeddings_index.get(word)
    if i < vocab_size:
        if embedding_vector is not None:

            embedding_matrix[i] = embedding_vector
```

```
embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
```

Model 8

```
sentiment_model = Sequential()
sentiment_model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
sentiment_model.add(LSTM(32))
sentiment_model.add(Dense(1, activation='sigmoid'))

# Freeze embedding layer with pretrained weights
sentiment_model.layers[0].set_weights([embedding_matrix])
sentiment_model.layers[0].trainable = False

# Configure model training
adam_optimizer = keras.optimizers.Adam(learning_rate=0.0001)
sentiment_model.compile(optimizer=adam_optimizer,
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
sentiment_model.summary()
```

```
model_checkpoint = ModelCheckpoint(
    filepath="pretrainmodel1.keras",
    save_best_only=True,
    monitor="val_loss"
)

pretrain_history = sentiment_model.fit(
    train_data,
    train_labels,
    epochs=30,
    batch_size=32,
    validation_data=(val_data, val_labels),
    callbacks=[model_checkpoint]
)
```

```
train_acc = pretrain_history.history['accuracy']
val_acc = pretrain_history.history['val_accuracy']

train_loss = pretrain_history.history["loss"]
val_loss = pretrain_history.history["val_loss"]

epochs_range = range(1, len(train_acc) + 1)

plt.figure(figsize=(6,4))
plt.plot(epochs_range, train_acc, color="green", linestyle="dashed", label="Training Accuracy")
plt.plot(epochs_range, val_acc, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("Pretrained Model: Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6,4))
plt.plot(epochs_range, train_loss, color="red", linestyle="dashed", label="Training Loss")
plt.plot(epochs_range, val_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("Pretrained Model: Loss")
```

```
plt.legend()
plt.show()
```

```
pretrained_model = load_model('pretrainmodel1.keras')
evaluation_results = pretrained_model.evaluate(test_data, test_labels)
print(f'Test Loss: {evaluation_results[0]:.3f}')
print(f'Test Accuracy: {evaluation_results[1]:.3f}')
```

Model 9: Training a Pretrained Model with 4 LSTM Hidden Layers on a 10,000-Sample Dataset

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Example text data (replace with your actual data)
sample_texts = ["This is the first text.", "Another example text.", "Text data for training."]
sample_labels = [0, 1, 0]  # Example labels (replace with your actual labels)

# Parameters
max_seq_length = 150
num_training_samples = 10000
num_validation_samples = 10000
vocabulary_size = 10000

# Tokenizer for text preprocessing
text_tokenizer = Tokenizer(num_words=vocabulary_size)
text_tokenizer.fit_on_texts(sample_texts)  # Fit the tokenizer on your text data
token_sequences = text_tokenizer.texts_to_sequences(sample_texts)

# Get word index
vocabulary = text_tokenizer.word_index
print(f'Found {len(vocabulary)} unique tokens.')

# Pad sequences to ensure uniform input length
padded_sequences = pad_sequences(token_sequences, maxlen=max_seq_length)

# Convert labels to numpy array
label_array = np.asarray(sample_labels)
print('Shape of data tensor:', padded_sequences.shape)
print('Shape of label tensor:', label_array.shape)

# Shuffle the data and labels
shuffled_indices = np.arange(padded_sequences.shape[0])
np.random.shuffle(shuffled_indices)
padded_sequences = padded_sequences[shuffled_indices]
label_array = label_array[shuffled_indices]

# Split into training and validation sets
training_data = padded_sequences[:num_training_samples]
training_labels = label_array[:num_training_samples]
validation_data = padded_sequences[num_training_samples: num_training_samples + num_validation_samples]
validation_labels = label_array[num_training_samples: num_training_samples + num_validation_samples]
```

```
training_data.Shape
validation_data.Shape
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense
from tensorflow.keras import optimizers

vocab_size = 10000
embedding_dims = 150
max_seq_length = 150

embedding_weights = np.random.rand(vocab_size, embedding_dims)

sentiment_classifier = Sequential()
sentiment_classifier.add(Embedding(vocab_size, embedding_dims, input_length=max_seq_length))

sentiment_classifier.add(LSTM(512, return_sequences=True))
sentiment_classifier.add(Dropout(0.5))

sentiment_classifier.add(LSTM(256, return_sequences=True))
sentiment_classifier.add(Dropout(0.5))

sentiment_classifier.add(LSTM(128, return_sequences=True))
sentiment_classifier.add(Dropout(0.5))

sentiment_classifier.add(LSTM(128))
```

```
sentiment_classifier.add(Dense(256, activation='relu'))
sentiment_classifier.add(Dropout(0.5))
sentiment_classifier.add(Dense(256, activation='relu'))
sentiment_classifier.add(Dropout(0.5))
sentiment_classifier.add(Dense(1, activation='sigmoid'))

sentiment_classifier.layers[0].set_weights([embedding_weights])
sentiment_classifier.layers[0].trainable = False

adam_optimizer = optimizers.Adam(learning_rate=0.0001)
sentiment_classifier.compile(optimizer=adam_optimizer,
                           loss='binary_crossentropy',
                           metrics=['accuracy'])

sentiment_classifier.summary()


model_checkpoint = ModelCheckpoint(
    filepath="sentiment_model_v2.keras",
    save_best_only=True,
    monitor="val_loss"
)

training_history = sentiment_classifier.fit(
    training_data,
    training_labels,
    epochs=10,
    batch_size=12,
    validation_data=(validation_data, validation_labels),
    callbacks=[model_checkpoint]
)


train_acc = training_history.history['accuracy']
train_loss = training_history.history['loss']
epochs_range = range(1, len(train_acc) + 1)

val_acc = training_history.history.get('val_accuracy', None)
val_loss = training_history.history.get('val_loss', None)

plt.figure(figsize=(6, 4))
plt.plot(epochs_range, train_acc, color="grey", linestyle="dashed", label="Training Accuracy")
if val_acc is not None:
    plt.plot(epochs_range, val_acc, color="blue", linestyle="dashed", label="Validation Accuracy")
plt.title("Sentiment Model: Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6, 4))
plt.plot(epochs_range, train_loss, color="grey", linestyle="dashed", label="Training Loss")
if val_loss is not None:
    plt.plot(epochs_range, val_loss, color="blue", linestyle="dashed", label="Validation Loss")
plt.title("Sentiment Model: Loss")
plt.legend()
plt.show()
```

Model 10 Training a Pretrained Model with 2 LSTM Hidden Layers on a 20,000-Sample Dataset

```
max_sequence_length = 150
num_train_samples = 20000
num_val_samples = 10000
vocab_size = 10000

text_tokenizer = Tokenizer(num_words=vocab_size)
text_tokenizer.fit_on_texts(text_corpus)
token_sequences = text_tokenizer.texts_to_sequences(text_corpus)

vocab_dict = text_tokenizer.word_index
print(f'Found {len(vocab_dict)} unique tokens.')

padded_sequences = pad_sequences(token_sequences, maxlen=max_sequence_length)

label_array = np.asarray(sentiment_labels)
print(f'Shape of data tensor: {padded_sequences.shape}')
print(f'Shape of label tensor: {label_array.shape}')

shuffled_indices = np.arange(padded_sequences.shape[0])
np.random.shuffle(shuffled_indices)
padded_sequences = padded_sequences[shuffled_indices]
label_array = label_array[shuffled_indices]
```

```
train_features = padded_sequences[:num_train_samples]
train_labels = label_array[:num_train_samples]
val_features = padded_sequences[num_train_samples: num_train_samples + num_val_samples


train_features.Shape
val_features.Shape


sentiment_analyzer = Sequential()
sentiment_analyzer.add(Embedding(vocab_size, embedding_dims, input_length=max_sequence_length))

sentiment_analyzer.add(LSTM(64, return_sequences=True, dropout=0.2, recurrent_dropout=0.2))
sentiment_analyzer.add(LSTM(32, dropout=0.2, recurrent_dropout=0.2))

sentiment_analyzer.add(Dense(64, activation='relu'))
sentiment_analyzer.add(Dropout(0.5))
sentiment_analyzer.add(Dense(1, activation='sigmoid'))

# Set pretrained embeddings and freeze layer
sentiment_analyzer.layers[0].set_weights([embedding_weights])
sentiment_analyzer.layers[0].trainable = False

# Configure optimizer and compile model
adam_opt = keras.optimizers.Adam(learning_rate=0.001)
sentiment_analyzer.compile(
    optimizer=adam_opt,
    loss='binary_crossentropy',
    metrics=['accuracy']
)
sentiment_analyzer.summary()


checkpoint_callback = ModelCheckpoint(
    filepath="sentiment_model_v3.h5",
    save_best_only=True,
    monitor="val_loss"
)

training_results = sentiment_analyzer.fit(
    train_features,
    train_labels,
    epochs=10,
    batch_size=12,
    validation_data=(val_features, val_labels),
    callbacks=[checkpoint_callback]
)
```