Open in Colab

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

```
Requirement already satisfied: tensorflow==2.12 in /usr/local/lib/python3.1
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.1
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dis
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/py
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/l
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.1
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/p
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3
Requirement already satisfied: ml-dtypes>=0.2.0 in /usr/local/lib/python3.1
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/pyth
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /us
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/pyt
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/pyth
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/p
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/p
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/pytho
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
!pip install keras-preprocessing



#Installing Packages required for deep learning

from tensorflow import keras
from keras import layers
from keras import preprocessing
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding, LSTM,  Conv1D, MaxPooling1D
from keras.models import load_model

from sklearn.model_selection import train_test_split
from keras.optimizers import RMSprop
from keras.optimizers import adam
from google.colab import files
import re, os
```

```
Requirement already satisfied: keras-preprocessing in /usr/local/lib/python
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist
```

```
import logging



logging.getLogger('tensorflow').disabled = True
```

Loading the dataset with reviews truncated after 150 words, limiting training samples to 100, validating on 10,000 samples, and considering only the top 10,000 words.

```
# Cutoff reviews after 150 words
max_len = 150

# Restrict training samples to 100
num_train_samples = 100

# Validate on 10,000 samples
num_val_samples = 10000

# Consider only the top 10,000 words
num_words = 10000

(x_train, y_train), (x_val, y_val) = imdb.load_data(num_words=num_words)

x_train = keras.preprocessing.sequence.pad_sequences(
    x_train, maxlen=max_len)
x_val = keras.preprocessing.sequence.pad_sequences(
    x_val, maxlen=max_len)
```

```
# First we code the Embedding layer
model_embedding = keras.Sequential([
    layers.Embedding(num_words, 10, input_length=max_len),
    layers.Flatten(),
    layers.Dense(1, activation='sigmoid')
])
```

```
# Model compilattion
model_embedding.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', me
model_embedding.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 150, 10)           100000

 flatten (Flatten)           (None, 1500)              0

 dense (Dense)               (None, 1)                 1501

=================================================================
Total params: 101,501
Trainable params: 101,501
Non-trainable params: 0
_________________________________________________________________
```

```
# Callbacks
callbacks = ModelCheckpoint(
            filepath= "model_embedding1.keras",
            save_best_only= True,
            monitor= "val_loss"
            )



# Running the Model using model_embedding.fit
Model_embedded = model_embedding.fit(x_train, y_train,
                    epochs=30,
                    batch_size=16,
                    validation_split=0.2,
                    callbacks=callbacks)
```

```
Epoch 1/30
1250/1250 [==============================] - 5s 3ms/step - loss: 0.5528 - a
Epoch 2/30
1250/1250 [==============================] - 5s 4ms/step - loss: 0.2928 - a
Epoch 3/30
1250/1250 [==============================] - 5s 4ms/step - loss: 0.2370 - a
Epoch 4/30
1250/1250 [==============================] - 5s 4ms/step - loss: 0.2046 - a
Epoch 5/30
1250/1250 [==============================] - 5s 4ms/step - loss: 0.1773 - a
Epoch 6/30
1250/1250 [==============================] - 4s 3ms/step - loss: 0.1536 - a
Epoch 7/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.1289 - a
Epoch 8/30
1250/1250 [==============================] - 2s 2ms/step - loss: 0.1060 - a
Epoch 9/30
```

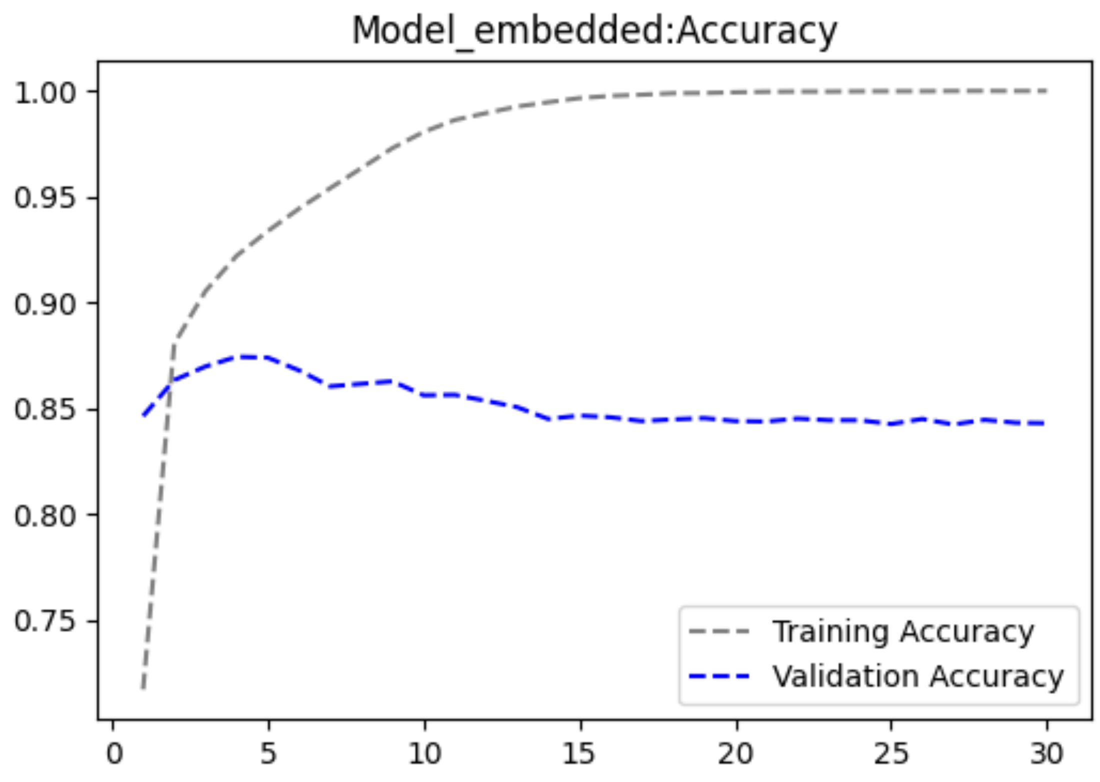
```
1250/1250 [==============================] - 3s 3ms/step - loss: 0.0855 - a
Epoch 10/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0670 - a
Epoch 11/30
1250/1250 [==============================] - 2s 2ms/step - loss: 0.0518 - a
Epoch 12/30
1250/1250 [==============================] - 2s 2ms/step - loss: 0.0397 - a
Epoch 13/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0293 - a
Epoch 14/30
1250/1250 [==============================] - 3s 3ms/step - loss: 0.0221 - a
Epoch 15/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0164 - a
Epoch 16/30
1250/1250 [==============================] - 2s 2ms/step - loss: 0.0119 - a
Epoch 17/30
1250/1250 [==============================] - 2s 2ms/step - loss: 0.0087 - a
Epoch 18/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0068 - a
Epoch 19/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0050 - a
Epoch 20/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0040 - a
Epoch 21/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0029 - a
Epoch 22/30
1250/1250 [==============================] - 2s 2ms/step - loss: 0.0024 - a
Epoch 23/30
1250/1250 [==============================] - 3s 3ms/step - loss: 0.0019 - a
Epoch 24/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0015 - a
Epoch 25/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0013 - a
Epoch 26/30
1250/1250 [==============================] - 3s 2ms/step - loss: 0.0010 - a
Epoch 27/30
1250/1250 [==============================] - 3s 2ms/step - loss: 7.8571e-04
Epoch 28/30
1250/1250 [==============================] - 3s 2ms/step - loss: 5.3917e-04
Epoch 29/30
1250/1250 [==============================] - 2s 2ms/step - loss: 5.8768e-04
Epoch 30/30
```

```python
# Printing the measures
print(Model_embedded.history.keys())
```

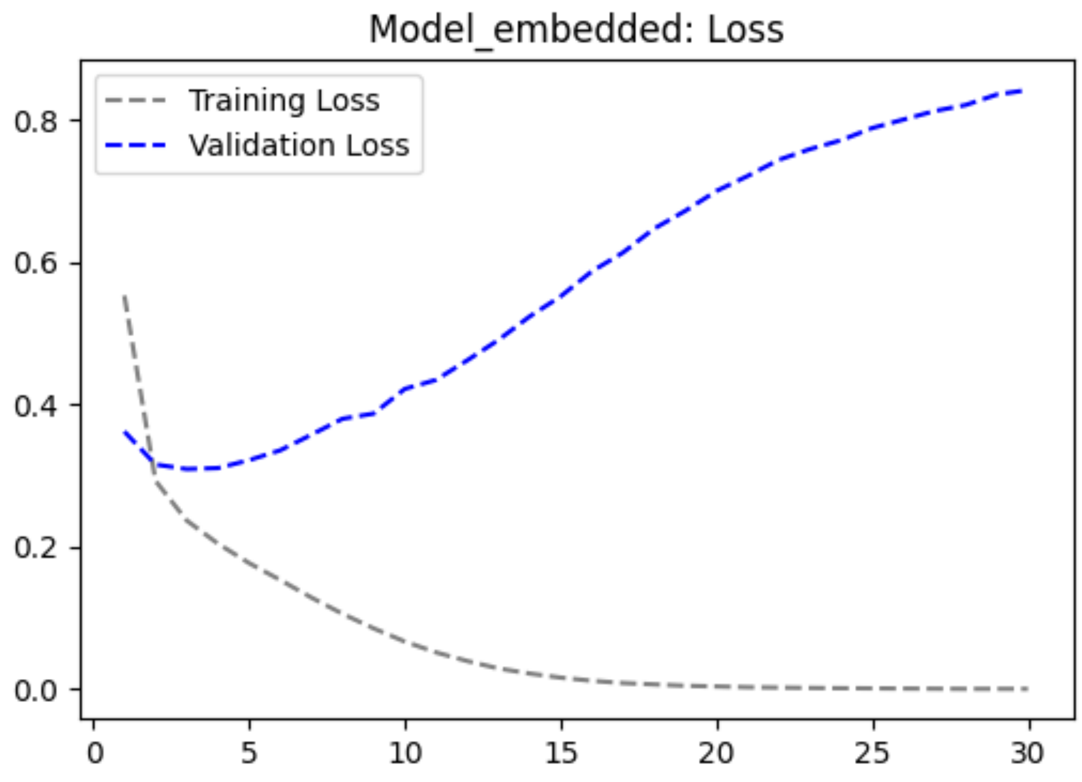
```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```python
#'acc' is the representation for accuracy
accuracy = Model_embedded.history['acc']
val_accuracy = Model_embedded.history['val_acc']
```

```
loss = Model_embedded.history["loss"]
val_loss = Model_embedded.history["val_loss"]



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

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed", label="Validati
plt.title("Model_embedded:Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model_embedded: Loss")
plt.legend()
plt.show()
```


Model_embedded:Accuracy
Training Accuracy
Validation Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0
5
10
15
20
25
30

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


Model_embedded: Loss
Training Loss
Validation Loss
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

Training Accuracy and Loss: The training accuracy progressively rises and eventually stabilizes at 100%, while the training loss substantially decreases, indicating effective learning from the training data. Validation Accuracy and Loss: The validation accuracy remains consistently high, stabilizing at approximately 86%, indicating robust generalization to unseen data. The validation loss converges to a stable value, suggesting that the model is not overfitting the training data. Overall Performance: Both the accuracy and loss plots for both training and validation demonstrate that the model is effectively learning and generalizing to new data.

```
Model_embedded_validate = load_model('model_embedding1.keras')
Model1_Results = Model_embedded_validate.evaluate(x_val,y_val)
print(f'Loss: {Model1_Results[0]:.3f}')
print(f'Accuracy: {Model1_Results[1]:.3f}')
```

```
782/782 [==============================] - 1s 1ms/step - loss: 0.3024 - acc
Loss: 0.302
Accuracy: 0.872
```

According to the embedded layer, approximately 87.2% of the remaining dataset samples were accurately classified. In the preceding model, the data has not been divided into sample sizes yet; instead, all available data was utilized, resulting in an accuracy of 87%.

Model_embedded_200: Modifying the number of training samples to assess variations in the model's performance. Training set size = 200.

```
# Establishing the maximum limit for the vocabulary's word count.
num_words = 10000

# Loading the IMDB Dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Cut-Off reviews after 150 words
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merging Training and Testing data
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Splitting the data into Training and Validation Samples
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Split the data further to obtain a test size of 5000 samples.
_, test_texts, _, test_labels = train_test_split(test_data, test_labels, test_s
```

```
train_texts.shape
```

```
(200, 150)
```

```
val_texts.shape
```

```
(10000, 150)
```

```
test_texts.shape
```

```
(5000, 150)
```

```
# Define the model
embedding_dim = 10

model_embedding_200 = keras.Sequential([
    layers.Embedding(input_dim=num_words, output_dim=embedding_dim, input_lengt
    layers.Flatten(),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_embedding_200.compile(optimizer='rmsprop', loss='binary_crossentropy', me
```

```
model_embedding_200.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 150, 10)           100000

 flatten_1 (Flatten)         (None, 1500)              0

 dense_1 (Dense)             (None, 1)                 1501

=================================================================
Total params: 101,501
Trainable params: 101,501
Non-trainable params: 0
_________________________________________________________________
```

```
# Callbacks
callbacks = ModelCheckpoint(
            filepath= "model_embedding_200.keras",
            save_best_only= True,
            monitor= "val_loss"
            )



# Running the Model using model_embedding.fit
model_embedding_200 = model_embedding_200.fit(train_texts, train_labels,
                    epochs=30,
                    batch_size=16,
                    validation_split=0.2,
                    callbacks=callbacks)
```

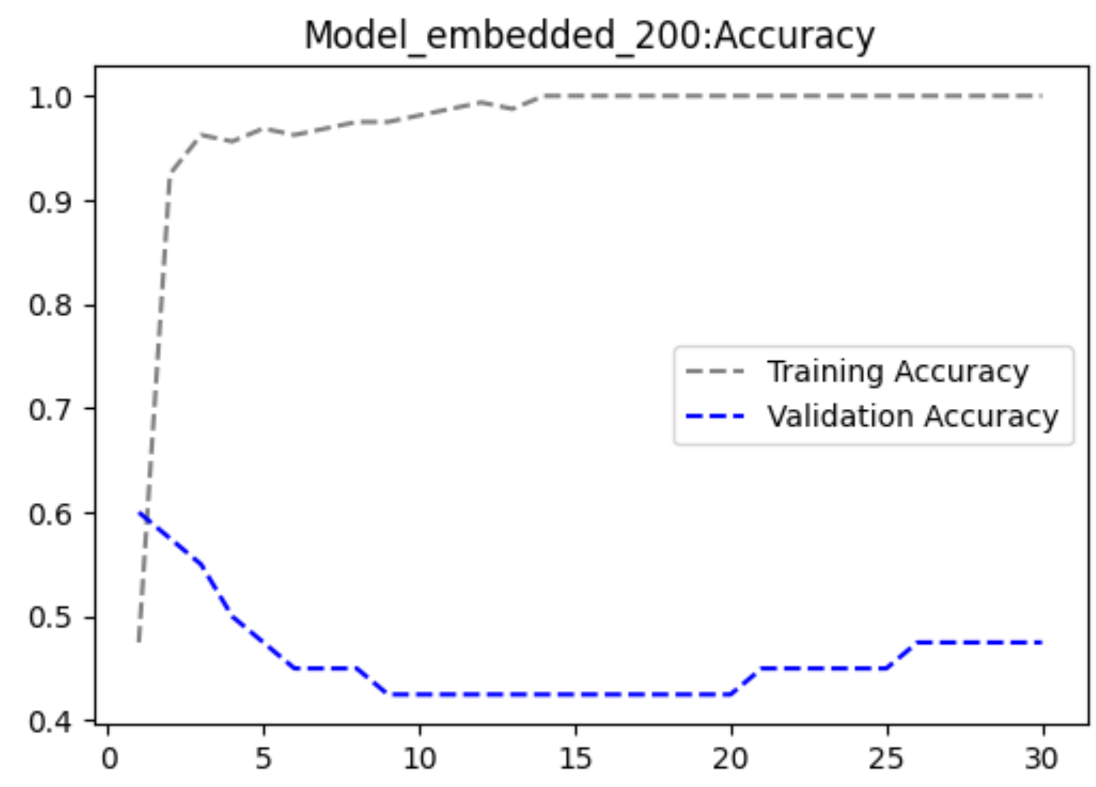
```
Epoch 1/30
10/10 [==============================] - 1s 20ms/step - loss: 0.6936 - acc:
Epoch 2/30
10/10 [==============================] - 0s 4ms/step - loss: 0.6638 - acc:
Epoch 3/30
10/10 [==============================] - 0s 4ms/step - loss: 0.6400 - acc:
Epoch 4/30
10/10 [==============================] - 0s 6ms/step - loss: 0.6147 - acc:
Epoch 5/30
10/10 [==============================] - 0s 6ms/step - loss: 0.5868 - acc:
Epoch 6/30
10/10 [==============================] - 0s 6ms/step - loss: 0.5563 - acc:
Epoch 7/30
10/10 [==============================] - 0s 6ms/step - loss: 0.5227 - acc:
Epoch 8/30
10/10 [==============================] - 0s 6ms/step - loss: 0.4872 - acc:
Epoch 9/30
10/10 [==============================] - 0s 6ms/step - loss: 0.4500 - acc:
Epoch 10/30
10/10 [==============================] - 0s 4ms/step - loss: 0.4119 - acc:
Epoch 11/30
10/10 [==============================] - 0s 4ms/step - loss: 0.3734 - acc:
Epoch 12/30
10/10 [==============================] - 0s 5ms/step - loss: 0.3356 - acc:
Epoch 13/30
10/10 [==============================] - 0s 6ms/step - loss: 0.2991 - acc:
Epoch 14/30
10/10 [==============================] - 0s 4ms/step - loss: 0.2643 - acc:
Epoch 15/30
10/10 [==============================] - 0s 5ms/step - loss: 0.2322 - acc:
Epoch 16/30
10/10 [==============================] - 0s 4ms/step - loss: 0.2022 - acc:
Epoch 17/30
10/10 [==============================] - 0s 4ms/step - loss: 0.1753 - acc:
Epoch 18/30
10/10 [==============================] - 0s 5ms/step - loss: 0.1510 - acc:
Epoch 19/30
10/10 [==============================] - 0s 6ms/step - loss: 0.1297 - acc:
Epoch 20/30
10/10 [==============================] - 0s 7ms/step - loss: 0.1107 - acc:
Epoch 21/30
10/10 [==============================] - 0s 5ms/step - loss: 0.0941 - acc:
Epoch 22/30
10/10 [==============================] - 0s 5ms/step - loss: 0.0799 - acc:
Epoch 23/30
10/10 [==============================] - 0s 6ms/step - loss: 0.0675 - acc:
Epoch 24/30
10/10 [==============================] - 0s 6ms/step - loss: 0.0571 - acc:
Epoch 25/30
10/10 [==============================] - 0s 6ms/step - loss: 0.0480 - acc:
Epoch 26/30
10/10 [==============================] - 0s 6ms/step - loss: 0.0406 - acc:
Epoch 27/30
10/10 [==============================] - 0s 6ms/step - loss: 0.0343 - acc:
```

```
Epoch 28/30
10/10 [==============================] - 0s 5ms/step - loss: 0.0288 - acc:
Epoch 29/30
10/10 [==============================] - 0s 4ms/step - loss: 0.0244 - acc:
Epoch 30/30
```

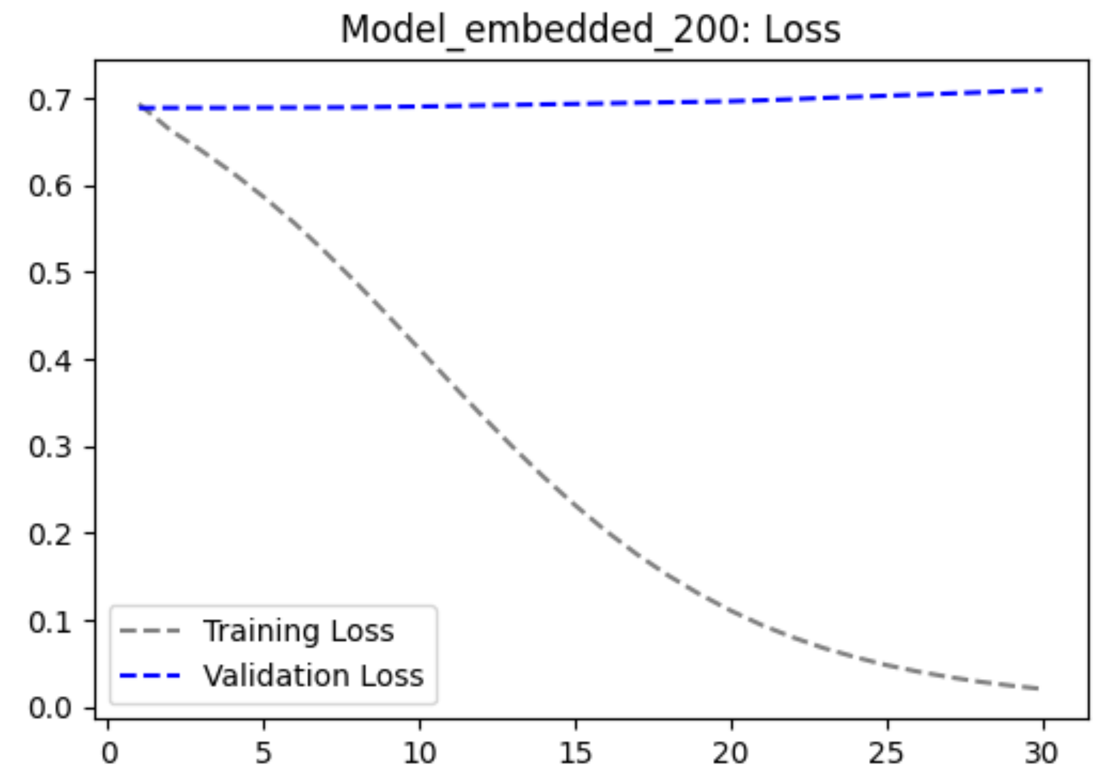
```
# Print the keys
print(model_embedding_200.history.keys())
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```
# 'acc' is the representation for accuracy
accuracy = model_embedding_200.history['acc']
val_accuracy = model_embedding_200.history['val_acc']

loss = model_embedding_200.history["loss"]
val_loss = model_embedding_200.history["val_loss"]



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

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed", label="Validati
plt.title("Model_embedded_200:Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model_embedded_200: Loss")
plt.legend()
plt.show()
```



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



Model_embedded_500: To see changes in the model's performance, change its number of training samples. Size of training set: 500.

```
# Establishing the maximum limit for the vocabulary's word count.
num_words = 10000

# Loading the IMDB Dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Cut-Off reviews after 150 words
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Creating a unified dataset by merging the training and testing data.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Dividing the data into training and validation samples.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Split the data further to obtain a test size of 5000 samples.
_, test_texts, _, test_labels = train_test_split(test_data, test_labels, test_s
```

```
train_texts.shape
```

```
(500, 150)
```

```
val_texts.shape
```

```
(10000, 150)
```

```
test_texts.shape
```

```
(5000, 150)
```

```
# Using embedding model with dimension = 10
embedding_dim = 10

model_embedding_500 = keras.Sequential([
    layers.Embedding(input_dim=num_words, output_dim=embedding_dim, input_lengt
    layers.Flatten(),
    layers.Dense(1, activation='sigmoid')
])

# Model compilling
model_embedding_500.compile(optimizer='rmsprop', loss='binary_crossentropy', me
```

```
model_embedding_500.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 150, 10)           100000

 flatten_2 (Flatten)         (None, 1500)              0

 dense_2 (Dense)             (None, 1)                 1501

=================================================================
Total params: 101,501
Trainable params: 101,501
Non-trainable params: 0
_________________________________________________________________
```

```
# Callbacks
callbacks = ModelCheckpoint(
            filepath= "model_embedding_500.keras",
            save_best_only= True,
            monitor= "val_loss"
            )



# Running the Model using model_embedding.fit
model_embedding_500 = model_embedding_500.fit(train_texts, train_labels,
                    epochs=30,
                    batch_size=16,
                    validation_split=0.2,
                    callbacks=callbacks)
```

```
25/25 [==============================] - 1s 9ms/step - loss: 0.6917 - acc:
Epoch 2/30
```

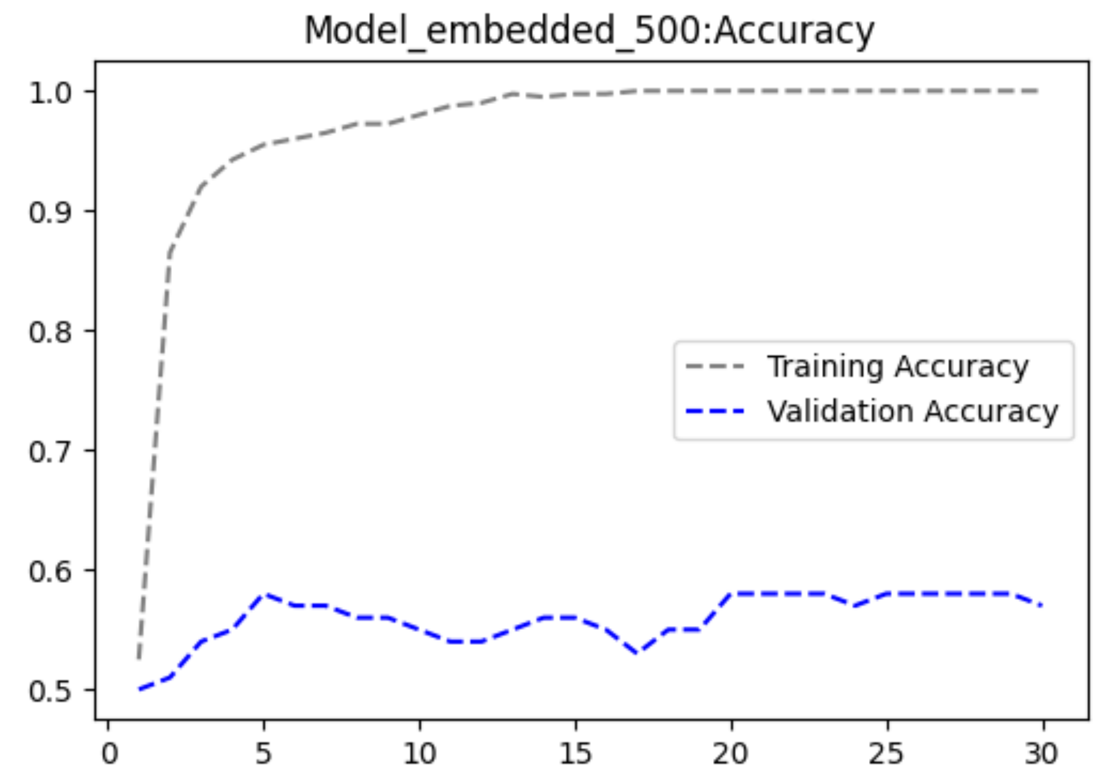
```
25/25 [==============================] - 0s 3ms/step - loss: 0.6643 - acc:
Epoch 3/30
25/25 [==============================] - 0s 3ms/step - loss: 0.6369 - acc:
Epoch 4/30
25/25 [==============================] - 0s 3ms/step - loss: 0.6035 - acc:
Epoch 5/30
25/25 [==============================] - 0s 4ms/step - loss: 0.5630 - acc:
Epoch 6/30
25/25 [==============================] - 0s 4ms/step - loss: 0.5164 - acc:
Epoch 7/30
25/25 [==============================] - 0s 4ms/step - loss: 0.4649 - acc:
Epoch 8/30
25/25 [==============================] - 0s 3ms/step - loss: 0.4106 - acc:
Epoch 9/30
25/25 [==============================] - 0s 3ms/step - loss: 0.3555 - acc:
Epoch 10/30
25/25 [==============================] - 0s 3ms/step - loss: 0.3017 - acc:
Epoch 11/30
25/25 [==============================] - 0s 3ms/step - loss: 0.2509 - acc:
Epoch 12/30
25/25 [==============================] - 0s 3ms/step - loss: 0.2052 - acc:
Epoch 13/30
25/25 [==============================] - 0s 3ms/step - loss: 0.1648 - acc:
Epoch 14/30
25/25 [==============================] - 0s 3ms/step - loss: 0.1309 - acc:
Epoch 15/30
25/25 [==============================] - 0s 3ms/step - loss: 0.1025 - acc:
Epoch 16/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0795 - acc:
Epoch 17/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0612 - acc:
Epoch 18/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0468 - acc:
Epoch 19/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0358 - acc:
Epoch 20/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0273 - acc:
Epoch 21/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0210 - acc:
Epoch 22/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0163 - acc:
Epoch 23/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0126 - acc:
Epoch 24/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0100 - acc:
Epoch 25/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0080 - acc:
Epoch 26/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0065 - acc:
Epoch 27/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0053 - acc:
Epoch 28/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0045 - acc:
Epoch 29/30
```

```
25/25 [==============================] - 0s 3ms/step - loss: 0.0038 - acc:
Epoch 30/30
25/25 [==============================] - 0s 3ms/step - loss: 0.0032 - acc:
```

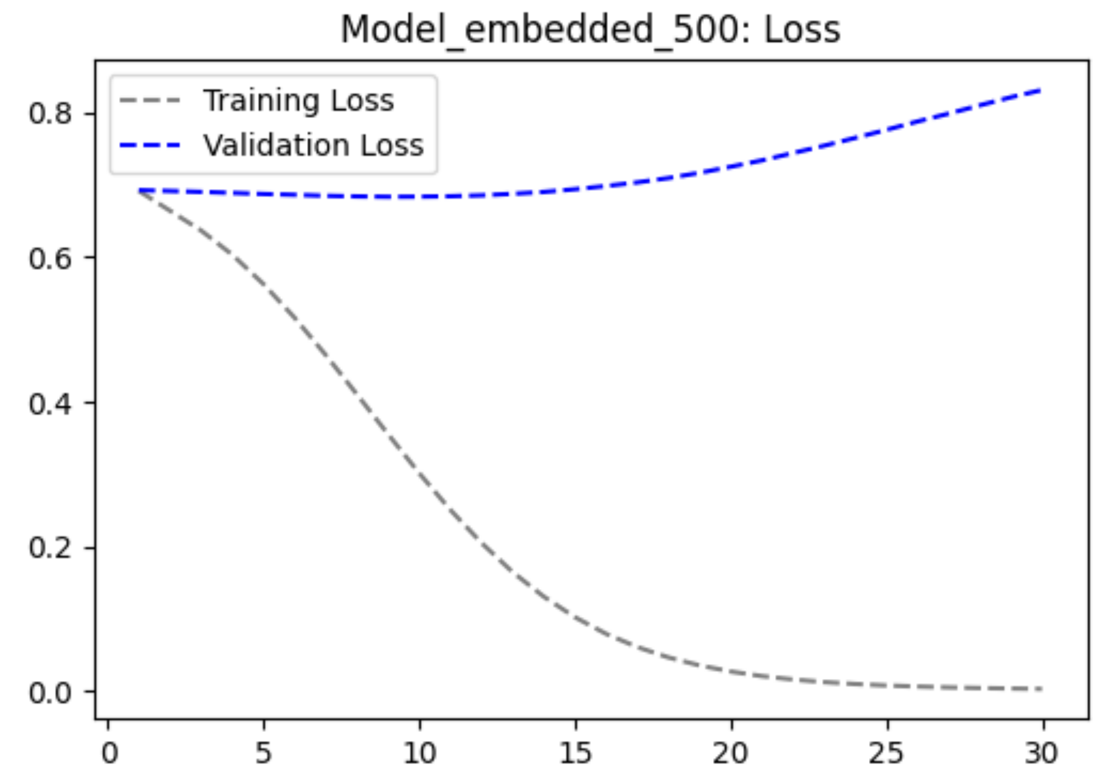
```
 # display of keys
print(model_embedding_500.history.keys())
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```


# 'acc' is the representation for accuracy
accuracy = model_embedding_500.history['acc']
val_accuracy = model_embedding_500.history['val_acc']

loss = model_embedding_500.history["loss"]
val_loss = model_embedding_500.history["val_loss"]



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

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed", label="Validati
plt.title("Model_embedded_500:Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model_embedded_500: Loss")
plt.legend()
plt.show()
```



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



Model_embedded_1000: To assess differences in the model's performance, change the amount of training samples. The size of the training set is 1000.

```
# Establishing the maximum number of words to utilize in the vocabulary.
num_words = 10000

# Load the IMDB dataset.
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Truncate the reviews after 150 words.
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merging the training and testing data forms a unified dataset.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Dividing the data into training and validation sets.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Split the data further to obtain a test size of 5000 samples.
_, test_texts, _, test_labels = train_test_split(test_data, test_labels, test_s
```

```
train_texts.shape
```

```
(1000, 150)
```

```
val_texts.shape
```

```
(10000, 150)
```

```
test_texts.shape
```

```
(5000, 150)
```

```
# Using embedding model with dimension = 10
embedding_dim = 10

model_embedding_1000 = keras.Sequential([
    layers.Embedding(input_dim=num_words, output_dim=embedding_dim, input_lengt
    layers.Flatten(),
    layers.Dense(1, activation='sigmoid')
])

# Model compilling
model_embedding_1000.compile(optimizer='rmsprop', loss='binary_crossentropy', m

# callbacks.
callbacks = ModelCheckpoint(
            filepath= "model_embedding_1000.keras",
            save_best_only= True,
            monitor= "val_loss"
            )
```

```
# Summary of results
model_embedding_1000.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 150, 10)           100000

 flatten_3 (Flatten)         (None, 1500)              0

 dense_3 (Dense)             (None, 1)                 1501

=================================================================
Total params: 101,501
Trainable params: 101,501
Non-trainable params: 0
_________________________________________________________________
```

```
# Running the Model using model_embedding.fit
model_embedding_1000 = model_embedding_1000.fit(train_texts, train_labels,
                    epochs=30,
                    batch_size=16,
                    validation_split=0.2,
                    callbacks=callbacks)
```

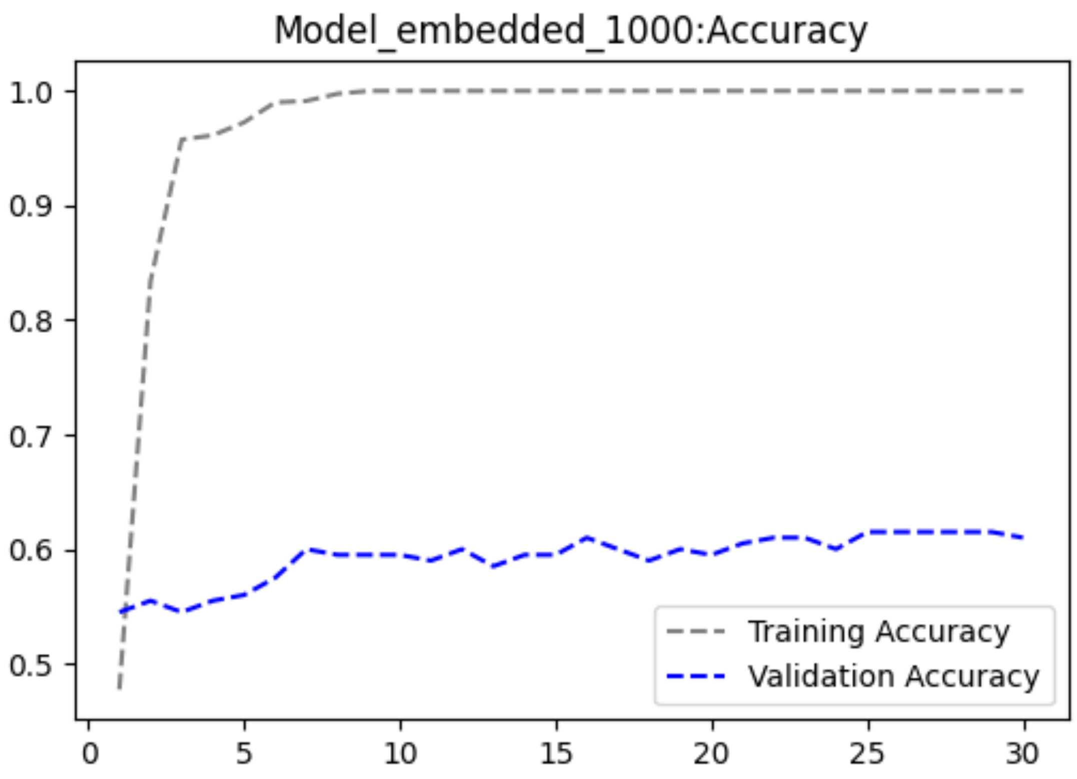
```
Epoch 1/30
50/50 [==============================] - 1s 9ms/step - loss: 0.6942 - acc:
Epoch 2/30
50/50 [==============================] - 0s 5ms/step - loss: 0.6689 - acc:
Epoch 3/30
50/50 [==============================] - 0s 4ms/step - loss: 0.6381 - acc:
Epoch 4/30
50/50 [==============================] - 0s 3ms/step - loss: 0.5956 - acc:
Epoch 5/30
50/50 [==============================] - 0s 3ms/step - loss: 0.5405 - acc:
Epoch 6/30
50/50 [==============================] - 0s 3ms/step - loss: 0.4755 - acc:
Epoch 7/30
50/50 [==============================] - 0s 3ms/step - loss: 0.4043 - acc:
Epoch 8/30
50/50 [==============================] - 0s 3ms/step - loss: 0.3324 - acc:
Epoch 9/30
50/50 [==============================] - 0s 3ms/step - loss: 0.2645 - acc:
Epoch 10/30
50/50 [==============================] - 0s 3ms/step - loss: 0.2044 - acc:
Epoch 11/30
50/50 [==============================] - 0s 3ms/step - loss: 0.1537 - acc:
Epoch 12/30
50/50 [==============================] - 0s 3ms/step - loss: 0.1130 - acc:
Epoch 13/30
50/50 [==============================] - 0s 3ms/step - loss: 0.0817 - acc:
Epoch 14/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0582 - acc:
Epoch 15/30
50/50 [==============================] - 0s 3ms/step - loss: 0.0409 - acc:
Epoch 16/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0287 - acc:
Epoch 17/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0202 - acc:
Epoch 18/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0144 - acc:
Epoch 19/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0104 - acc:
Epoch 20/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0077 - acc:
Epoch 21/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0058 - acc:
Epoch 22/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0045 - acc:
Epoch 23/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0036 - acc:
Epoch 24/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0030 - acc:
Epoch 25/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0025 - acc:
```

```
Epoch 26/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0021 - acc:
Epoch 27/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0018 - acc:
Epoch 28/30
50/50 [==============================] - 0s 2ms/step - loss: 0.0016 - acc:
Epoch 29/30
50/50 [==============================] - 0s 3ms/step - loss: 0.0014 - acc:
Epoch 30/30
```

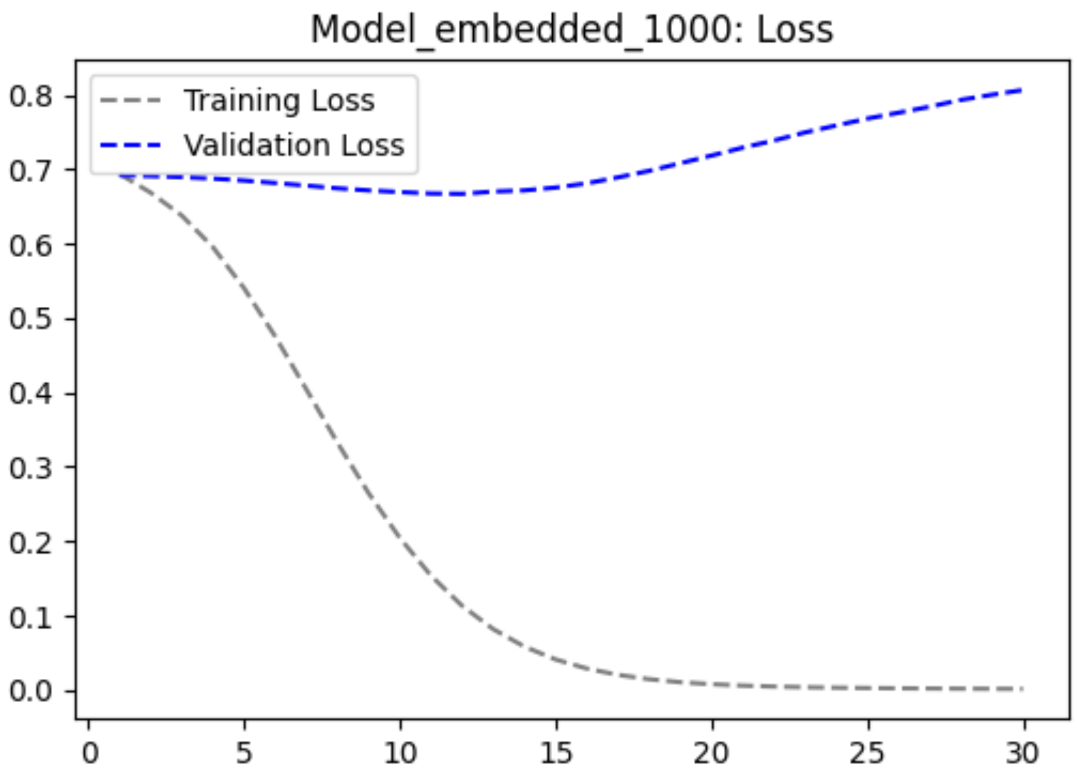
```
 # Printing keys
print(model_embedding_1000.history.keys())
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```
# 'acc' is the representation for accuracy
accuracy = model_embedding_1000.history['acc']
val_accuracy = model_embedding_1000.history['val_acc']

loss = model_embedding_1000.history["loss"]
val_loss = model_embedding_1000.history["val_loss"]



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

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed", label="Validati
plt.title("Model_embedded_1000:Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model_embedded_1000: Loss")
plt.legend()
plt.show()
```



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



Model_embedded_2000: adjusting the quantity of training samples to see how it affects the model's efficiency. The size of the training set is set to 2000.

```
# Establishing the maximum number of words to include in the vocabulary.
num_words = 10000

# Loading the IMDB dataset.
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Truncate the reviews after 150 words.
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merging the training and testing data forms a unified dataset.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Dividing the data into training and validation samples.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Split the data further to obtain a test size of 5000 samples.
_, test_texts, _, test_labels = train_test_split(test_data, test_labels, test_s
```

```
train_texts.shape
```

```
(2000, 150)
```

```
val_texts.shape
```

```
(10000, 150)
```

```
test_texts.shape
```

```
(5000, 150)
```

```
# Using embedding model with dimension = 10
embedding_dim = 10

model_embedding_2000 = keras.Sequential([
    layers.Embedding(input_dim=num_words, output_dim=embedding_dim, input_lengt
    layers.Flatten(),
    layers.Dense(1, activation='sigmoid')
])
```

```
# Model compilling
model_embedding_2000.compile(optimizer='rmsprop', loss='binary_crossentropy', m



# Summary of results
model_embedding_2000.summary()

# callbacks.
callbacks = ModelCheckpoint(
            filepath= "model_embedding_2000.keras",
            save_best_only= True,
            monitor= "val_loss"
            )


# Running the Model using model_embedding.fit
model_embedding_2000 = model_embedding_2000.fit(train_texts, train_labels,
                    epochs=30,
                    batch_size=16,
                    validation_split=0.2,
                    callbacks=callbacks)
```

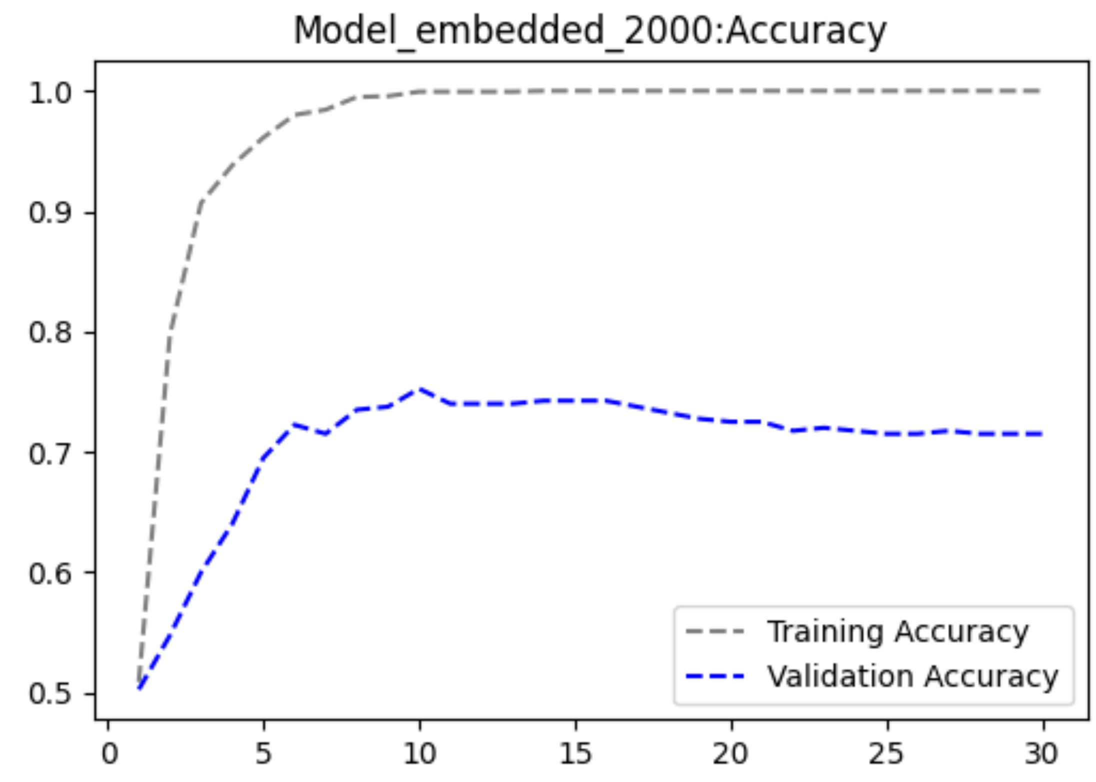
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 150, 10)           100000

 flatten_4 (Flatten)         (None, 1500)              0

 dense_4 (Dense)             (None, 1)                 1501

=================================================================
Total params: 101,501
```

```
Trainable params: 101,501
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
100/100 [==============================] - 1s 4ms/step - loss: 0.6928 - acc
Epoch 2/30
100/100 [==============================] - 0s 3ms/step - loss: 0.6655 - acc
Epoch 3/30
100/100 [==============================] - 0s 3ms/step - loss: 0.6231 - acc
Epoch 4/30
100/100 [==============================] - 0s 2ms/step - loss: 0.5582 - acc
Epoch 5/30
100/100 [==============================] - 0s 3ms/step - loss: 0.4728 - acc
Epoch 6/30
100/100 [==============================] - 0s 3ms/step - loss: 0.3785 - acc
Epoch 7/30
100/100 [==============================] - 0s 3ms/step - loss: 0.2878 - acc
Epoch 8/30
100/100 [==============================] - 0s 3ms/step - loss: 0.2102 - acc
Epoch 9/30
100/100 [==============================] - 0s 3ms/step - loss: 0.1484 - acc
Epoch 10/30
100/100 [==============================] - 0s 3ms/step - loss: 0.1018 - acc
Epoch 11/30
100/100 [==============================] - 0s 4ms/step - loss: 0.0683 - acc
Epoch 12/30
100/100 [==============================] - 0s 3ms/step - loss: 0.0451 - acc
Epoch 13/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0297 - acc
Epoch 14/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0194 - acc
Epoch 15/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0130 - acc
Epoch 16/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0088 - acc
Epoch 17/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0062 - acc
Epoch 18/30
100/100 [==============================] - 0s 3ms/step - loss: 0.0046 - acc
Epoch 19/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0034 - acc
Epoch 20/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0028 - acc
Epoch 21/30
100/100 [==============================] - 0s 2ms/step - loss: 0.0022 - acc
Epoch 22/30
100/100 [==============================] - 0s 3ms/step - loss: 0.0018 - acc
```

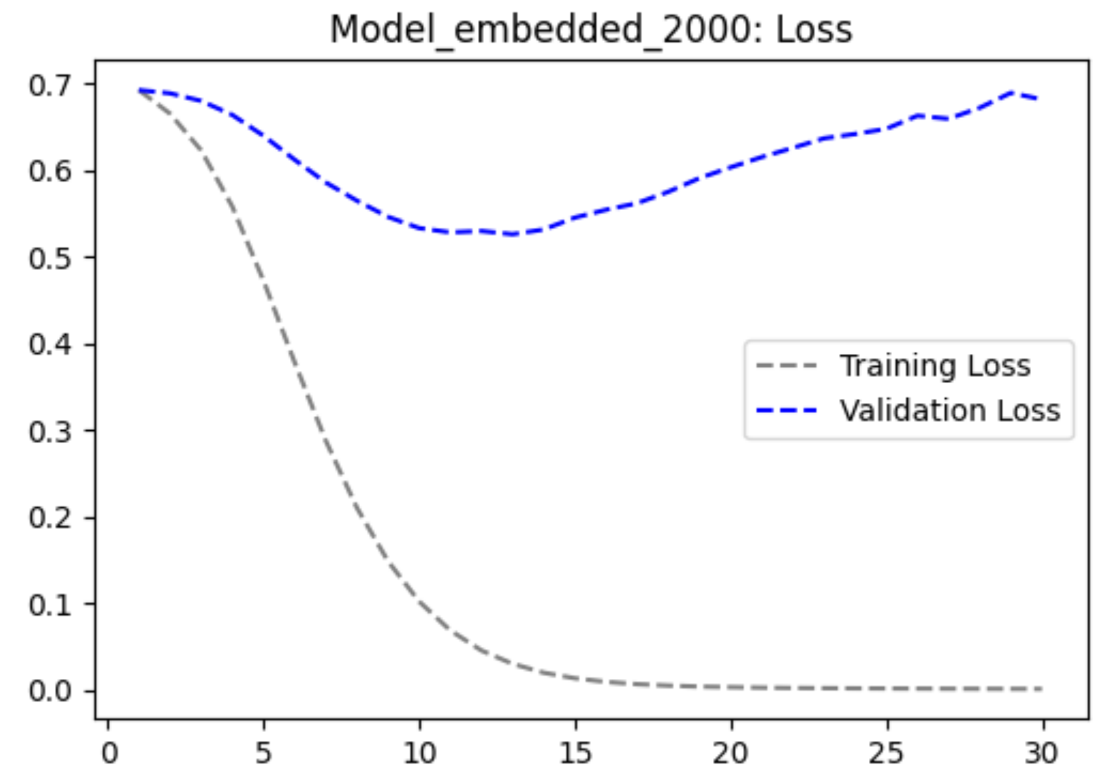
```
 # printing the keys
print(model_embedding_2000.history.keys())
```

```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

```
# 'acc' is the representation for accuracy
accuracy = model_embedding_2000.history['acc']
val_accuracy = model_embedding_2000.history['val_acc']

loss = model_embedding_2000.history["loss"]
val_loss = model_embedding_2000.history["val_loss"]



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

plt.figure(figsize=(6,4))
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue",linestyle="dashed", label="Validati
plt.title("Model_embedded_2000:Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6,4))
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation 
plt.title("Model_embedded_2000: Loss")
plt.legend()
plt.show()
```


Model_embedded_2000:Accuracy
Training Accuracy
Validation Accuracy

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


Model_embedded_2000: Loss
Training Loss
Validation Loss

High accuracy might be achieved rather quickly which is an indication of overfitting, especially for sample sizes that are much smaller. We need to check whether the models have good generalization ability to deal with unseen data. Pair the validation accuracy and make use of another test set for final assessment. Following the trend, increasing the sample size seems to support generalization, with the model 3 having slower, but more consistent convergence.

Utilizing Embedding and Conv1D for Reliable IMDB Classification

```
# Establishing the maximum number of words to utilize in the vocabulary.
num_words = 10000

# Loading the IMDB dataset.
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Limit the reviews to 150 words
maxlen = 150

# Padding the sequences to reach the maximum length.
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merge the training and testing data to form a comprehensive dataset.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Partitioning the data into training and validation samples.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Divide the validation data further to obtain a test size of 5000 samples.
val_texts, test_texts, val_labels, test_labels = train_test_split(val_texts, va
```

```
print("Shape of Training Data:", train_texts.shape)
print("Shape of Validation Data:", val_texts.shape)
print("Shape of Test Data:", test_texts.shape)
```

```
Shape of Training Data: (100, 150)
Shape of Validation Data: (5000, 150)
Shape of Test Data: (5000, 150)
```

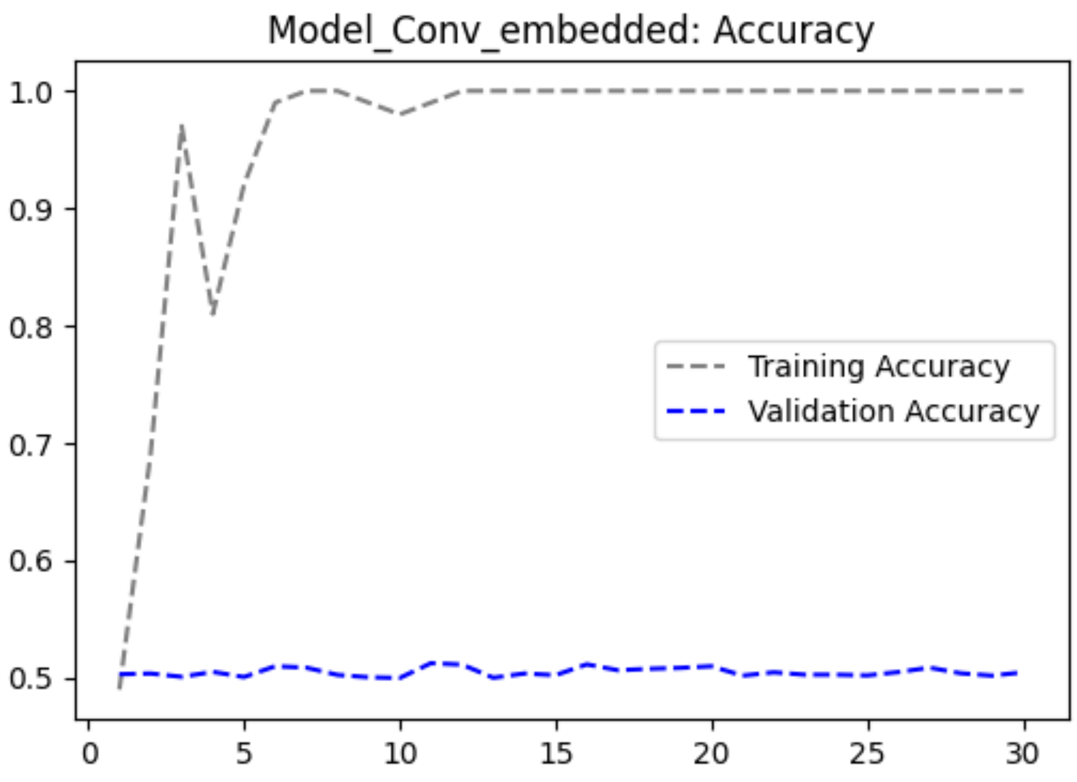
```
# Defining the model utilizing both Embedding and Conv1D layers.( Pretrained wo
embedding_dim = 10
filter_size = 3
num_filters = 32

model = Sequential([
    # Transforming words into vectors using the embedding layer.
    Embedding(input_dim=num_words, output_dim=embedding_dim, input_length=maxle

    # Utilizing a convolutional layer to extract features from sequences of wor
    Conv1D(filters=num_filters, kernel_size=filter_size, activation='relu'),

    # Max-pooling layer utilized for dimensionality reduction.
    MaxPooling1D(pool_size=2),

    # The Flatten layer is used to transform the 1D output into a 2D tensor.
    Flatten(),

    # Dense layer utilizing sigmoid activation for binary classification.
    Dense(1, activation='sigmoid')
])
```

```
# Model compilling using model.compile()
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Moodel training using model.fit()
history = model.fit(train_texts, train_labels, epochs=30, batch_size=16, valida

# Printing the accuracy metrices
test_loss, test_acc = model.evaluate(test_texts, test_labels)
print('Test accuracy:', test_acc)
```

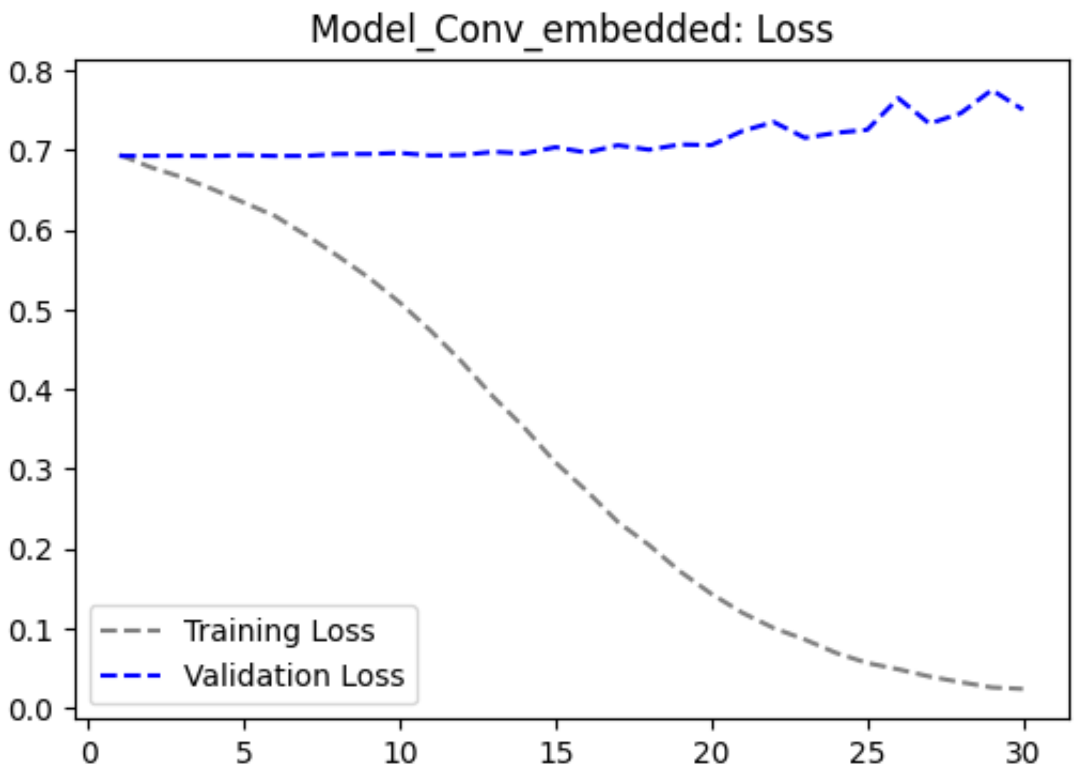
```
Epoch 1/30
7/7 [==============================] - 1s 134ms/step - loss: 0.6946 - acc:
Epoch 2/30
7/7 [==============================] - 1s 110ms/step - loss: 0.6789 - acc:
Epoch 3/30
7/7 [==============================] - 1s 111ms/step - loss: 0.6663 - acc:
Epoch 4/30
7/7 [==============================] - 1s 110ms/step - loss: 0.6511 - acc:
Epoch 5/30
7/7 [==============================] - 1s 111ms/step - loss: 0.6344 - acc:
Epoch 6/30
7/7 [==============================] - 1s 87ms/step - loss: 0.6174 - acc: 0
Epoch 7/30
7/7 [==============================] - 1s 91ms/step - loss: 0.5933 - acc: 1
Epoch 8/30
```

```
7/7 [==============================] - 1s 86ms/step - loss: 0.5677 - acc: 1
Epoch 9/30
7/7 [==============================] - 1s 110ms/step - loss: 0.5403 - acc:
Epoch 10/30
7/7 [==============================] - 1s 111ms/step - loss: 0.5089 - acc:
Epoch 11/30
7/7 [==============================] - 1s 110ms/step - loss: 0.4731 - acc:
Epoch 12/30
7/7 [==============================] - 1s 110ms/step - loss: 0.4340 - acc:
Epoch 13/30
7/7 [==============================] - 1s 89ms/step - loss: 0.3906 - acc: 1
Epoch 14/30
7/7 [==============================] - 1s 90ms/step - loss: 0.3517 - acc: 1
Epoch 15/30
7/7 [==============================] - 1s 138ms/step - loss: 0.3080 - acc:
Epoch 16/30
7/7 [==============================] - 1s 221ms/step - loss: 0.2724 - acc:
Epoch 17/30
7/7 [==============================] - 1s 110ms/step - loss: 0.2334 - acc:
Epoch 18/30
7/7 [==============================] - 1s 93ms/step - loss: 0.2045 - acc: 1
Epoch 19/30
7/7 [==============================] - 1s 90ms/step - loss: 0.1713 - acc: 1
Epoch 20/30
7/7 [==============================] - 1s 90ms/step - loss: 0.1433 - acc: 1
Epoch 21/30
7/7 [==============================] - 1s 111ms/step - loss: 0.1194 - acc:
Epoch 22/30
7/7 [==============================] - 1s 111ms/step - loss: 0.1004 - acc:
Epoch 23/30
7/7 [==============================] - 1s 89ms/step - loss: 0.0859 - acc: 1
Epoch 24/30
7/7 [==============================] - 1s 111ms/step - loss: 0.0692 - acc:
Epoch 25/30
7/7 [==============================] - 1s 111ms/step - loss: 0.0565 - acc:
Epoch 26/30
7/7 [==============================] - 1s 111ms/step - loss: 0.0486 - acc:
Epoch 27/30
7/7 [==============================] - 1s 87ms/step - loss: 0.0395 - acc: 1
Epoch 28/30
7/7 [==============================] - 1s 112ms/step - loss: 0.0324 - acc:
Epoch 29/30
7/7 [==============================] - 1s 88ms/step - loss: 0.0259 - acc: 1
Epoch 30/30
```

```
# Retrieve accuracy and loss values from the history object.
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

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

plt.figure(figsize=(6, 4))
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed", label="Validat
plt.title("Model_Conv_embedded: Accuracy")
plt.legend()
plt.figure()

plt.figure(figsize=(6, 4))
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation 
plt.title("Model_Conv_embedded: Loss")
plt.legend()
plt.show()
```


Model_Conv_embedded: Accuracy
Training Accuracy
Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30

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


Model_Conv_embedded: Loss
Training Loss
Validation Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30

A neural network model with both Embedding and Conv1D layers seems to be suffering from the problem of an overfit, this can be caused by the network complexity and the smaller size of our dataset. Incorporate simple architectural designs or use dropout approach to achieve regularization.

Conv1D and Embedding layers are employed, with change in embedding dimensions.

```
# Establishing the maximum vocabulary size.
num_words = 10000

# Loading the dataset from IMDB.
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=

# Truncate the reviews after 150 words.
maxlen = 150

# Padding the sequences to reach the maximum length.
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merge the training and testing data to form a unified dataset.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Separating the data into training and validation samples.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, label

# Split the validation data further to obtain a test size of 5000 samples.
val_texts, test_texts, val_labels, test_labels = train_test_split(val_texts, val
```

```
print("Shape of Training Data:", train_texts.shape)
print("Shape of Validation Data:", val_texts.shape)
print("Shape of Test Data:", test_texts.shape)
```

```
Shape of Training Data: (100, 150)
Shape of Validation Data: (5000, 150)
Shape of Test Data: (5000, 150)
```

```
# # Defining the model utilizing both Embedding and Conv1D layers.( Pretrained
embedding_dim = 50  # Enlarge the dimensions of embedding vectors.
filter_size = 3
num_filters = 32

model = Sequential([
    # An embedding layer for converting words into vectors.
    Embedding(input_dim=num_words, output_dim=embedding_dim, input_length=maxle

    # Utilizing a convolutional layer for extracting features from sequences of
    Conv1D(filters=num_filters, kernel_size=filter_size, activation='relu'),

    # Utilizing a max-pooling layer for dimensionality reduction.
    MaxPooling1D(pool_size=2),

    # The Flatten layer is utilized to transform the 1D output into a 2D tensor
    Flatten(),

    # A dense layer employing sigmoid activation for binary classification.
    Dense(1, activation='sigmoid')
])
```

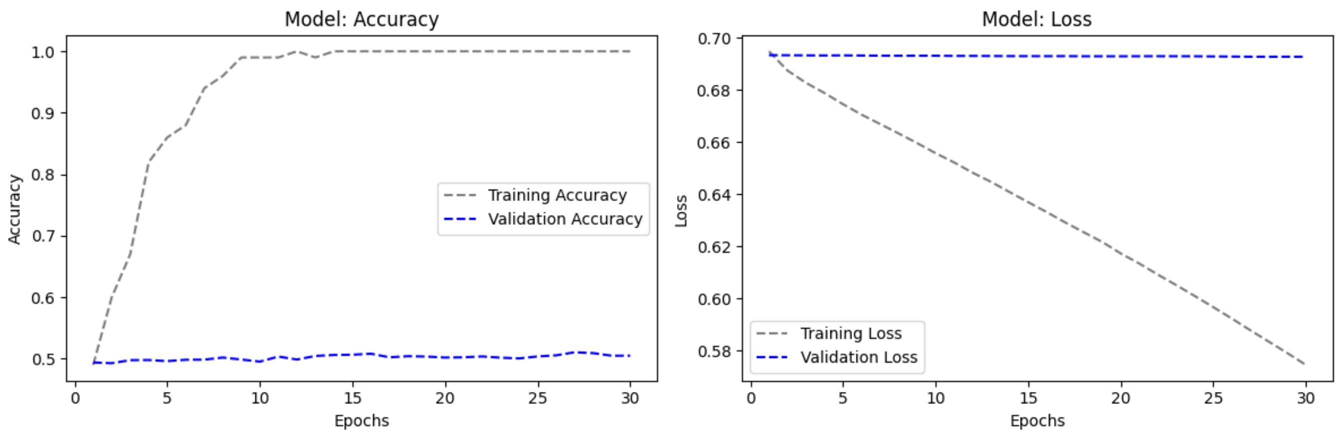
```
# Model compilling using the RMSprop optimizer.
model.compile(optimizer=RMSprop(lr=1e-4), loss='binary_crossentropy', metrics=[

# Incorporate early stopping as a measure to prevent overfitting.
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_wei

# Model training
history = model.fit(train_texts, train_labels, epochs=30, batch_size=16, valida
```

```
7/7 [==============================] - 1s 138ms/step - loss: 0.6947 - acc:
Epoch 2/30
7/7 [==============================] - 1s 111ms/step - loss: 0.6874 - acc:
Epoch 3/30
7/7 [==============================] - 1s 109ms/step - loss: 0.6826 - acc:
Epoch 4/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6787 - acc:
Epoch 5/30
7/7 [==============================] - 1s 107ms/step - loss: 0.6744 - acc:
Epoch 6/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6704 - acc:
Epoch 7/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6668 - acc:
```

```
Epoch 8/30
7/7 [==============================] - 1s 109ms/step - loss: 0.6632 - acc:
Epoch 9/30
7/7 [==============================] - 1s 145ms/step - loss: 0.6595 - acc:
Epoch 10/30
7/7 [==============================] - 1s 226ms/step - loss: 0.6556 - acc:
Epoch 11/30
7/7 [==============================] - 1s 120ms/step - loss: 0.6521 - acc:
Epoch 12/30
7/7 [==============================] - 1s 116ms/step - loss: 0.6482 - acc:
Epoch 13/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6446 - acc:
Epoch 14/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6407 - acc:
Epoch 15/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6368 - acc:
Epoch 16/30
7/7 [==============================] - 1s 107ms/step - loss: 0.6330 - acc:
Epoch 17/30
7/7 [==============================] - 1s 112ms/step - loss: 0.6291 - acc:
Epoch 18/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6253 - acc:
Epoch 19/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6216 - acc:
Epoch 20/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6172 - acc:
Epoch 21/30
7/7 [==============================] - 1s 107ms/step - loss: 0.6133 - acc:
Epoch 22/30
7/7 [==============================] - 1s 106ms/step - loss: 0.6091 - acc:
Epoch 23/30
7/7 [==============================] - 1s 109ms/step - loss: 0.6050 - acc:
Epoch 24/30
7/7 [==============================] - 1s 113ms/step - loss: 0.6009 - acc:
Epoch 25/30
7/7 [==============================] - 1s 221ms/step - loss: 0.5966 - acc:
Epoch 26/30
7/7 [==============================] - 1s 163ms/step - loss: 0.5922 - acc:
Epoch 27/30
7/7 [==============================] - 1s 107ms/step - loss: 0.5877 - acc:
Epoch 28/30
7/7 [==============================] - 1s 115ms/step - loss: 0.5833 - acc:
Epoch 29/30
7/7 [==============================] - 1s 108ms/step - loss: 0.5790 - acc:
Epoch 30/30
7/7 [==============================] - 1s 106ms/step - loss: 0.5744 - acc:
```

```python
# Retrieve accuracy and loss values from the history object.
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
# Visualizing the training and validation curves.
epochs = range(1, len(accuracy) + 1)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed", label="Validat
plt.title("Model: Accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation 
plt.title("Model: Loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Printing the values of Test
test_loss, test_accuracy = model.evaluate(test_texts, test_labels)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```



```
157/157 [==============================] - 1s 4ms/step - loss: 0.6924 - acc
Test Loss: 0.6924
Test Accuracy: 52.08%
```

In this scenario, we've enhanced the embedding vector size to 50, offering a more refined representation of the word. Additionally, a filter size of 3 with 32 filters is employed for feature extraction within the convolutional layers. The RMSprop optimizer is utilized with a learning rate set at 1e-4.

The training accuracy commences at 49%, as anticipated with random initialization. As epochs progress, it steadily improves to approximately 100%, indicating the model's learning from the training data. Both training and validation losses consistently decrease across epochs, signifying the model's adaptation to the training data. Nevertheless, the minor discrepancy in accuracy between the training and validation sets implies potential overfitting.

```
# Establishing the maximum number of words to include in the vocabulary.
num_words = 10000

# Loading the IMDB Dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Cut off the reviews after 150 words
maxlen = 150

# Please pad the sequences to the specified maximum length.
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merge the training and testing data to form a comprehensive dataset.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Partitioning the data into training and validation samples.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Additionally, divide the validation data to yield a test size of 5000 samples
val_texts, test_texts, val_labels, test_labels = train_test_split(val_texts, va
```

```
print("Shape of Training Data:", train_texts.shape)
print("Shape of Validation Data:", val_texts.shape)
print("Shape of Test Data:", test_texts.shape)
```

```
Shape of Training Data: (3500, 150)
Shape of Validation Data: (5000, 150)
Shape of Test Data: (5000, 150)
```

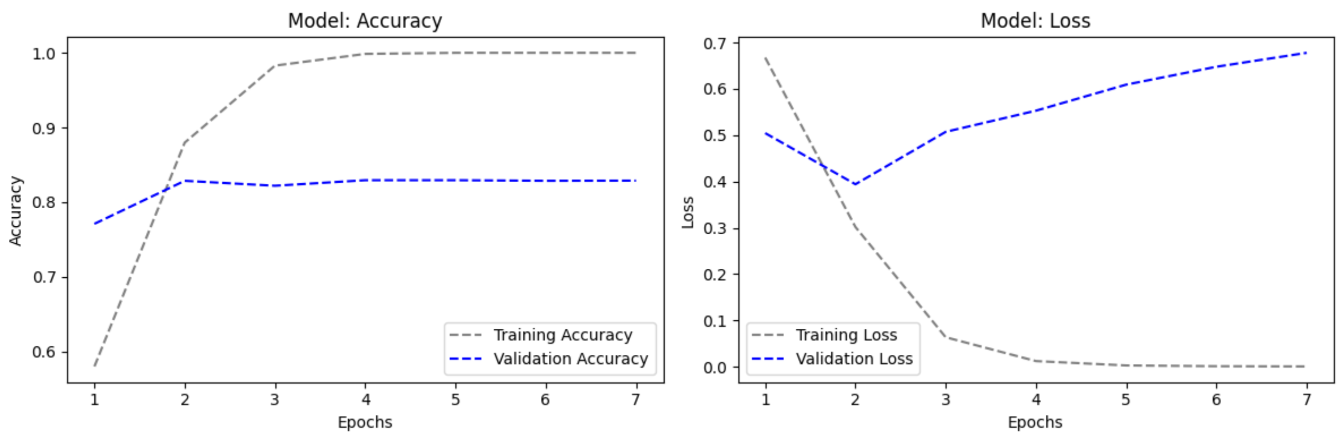
```
# Specify the model utilizing both Embedding and Conv1D layers.
embedding_dim = 50  # Enlarge the dimensions of embedding vectors.
filter_size = 5  # Augment the filter size to capture broader global features.
num_filters = 64  # Augment the quantity of filters.

model = Sequential([
    # Embedding layer for word-to-vector conversion.
    Embedding(input_dim=num_words, output_dim=embedding_dim, input_length=maxle

    # Convolutional layer for feature extraction from word sequences.
    Conv1D(filters=num_filters, kernel_size=filter_size, activation='relu'),

    # Utilizing a max-pooling layer for dimensionality reduction.
    MaxPooling1D(pool_size=2),

    # The Flatten layer is utilized to transform the 1D output into a 2D tensor
    Flatten(),

    # A dense layer with a sigmoid activation function for binary classificatio
    Dense(1, activation='sigmoid')
])
```

```
from tensorflow.keras.optimizers import Adam
```

```
# Compile the model using the Adam optimizer with a reduced learning rate.
model.compile(optimizer=Adam(lr=1e-4), loss='binary_crossentropy', metrics=['ac

# Implement early stopping as a preventive measure against overfitting.
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_wei

# Proceed with training the model.
history = model.fit(train_texts, train_labels, epochs=30, batch_size=16, valida
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_ra
Epoch 1/30
219/219 [==============================] - 4s 17ms/step - loss: 0.6673 - ac
Epoch 2/30
219/219 [==============================] - 5s 21ms/step - loss: 0.3018 - ac
Epoch 3/30
219/219 [==============================] - 4s 16ms/step - loss: 0.0638 - ac
Epoch 4/30
219/219 [==============================] - 4s 16ms/step - loss: 0.0123 - ac
Epoch 5/30
219/219 [==============================] - 6s 26ms/step - loss: 0.0030 - ac
Epoch 6/30
219/219 [==============================] - 4s 16ms/step - loss: 0.0013 - ac
Epoch 7/30
219/219 [==============================] - 4s 16ms/step - loss: 8.0987e-04
```

```
# Please extract the accuracy and loss values from the history object.
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Plotting the curves for training and validation, please.
epochs = range(1, len(accuracy) + 1)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed", label="Validat
plt.title("Model: Accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model: Loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# printing the metrices values
test_loss, test_accuracy = model.evaluate(test_texts, test_labels)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```



```
157/157 [==============================] - 1s 7ms/step - loss: 0.3728 - acc
Test Loss: 0.3728
Test Accuracy: 83.48%
```

I went to the larger embedding vector size of 50 to squeeze out the most accurate word representation. Used a filter size of 5 to get 64 filters to retrieve the details. Utilized Adam optimizer with a learning rate of 1e-4. We start from an accuracy of 58% at random initialization, and it slowly improves and reaches 100% over epochs. A sudden rapid increase in training correctness confirms that the model is capable of fitting the training set well. The validation accuracy proceeds the same trend to be up to 83.48%. While it is better, the model's performance on validation data is slightly surpassing randomization expectation level. The model presents a similar behavior pattern to the previous one, including a danger of overfitting. In comparison, increasing the embedding vector size and filter size was not useful for improving generalization.

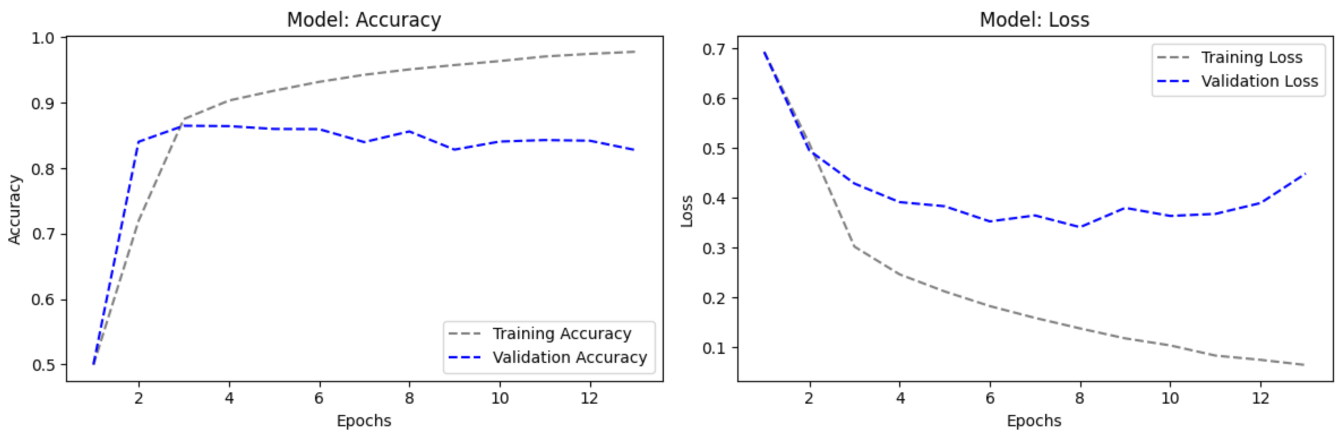
The Conv1D and Embedding layers are utilized, with modifications made to the embedding vector.

```
# Establishing the maximum number of words to utilize in the vocabulary
num_words = 10000

# Loading the IMDB Dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Trim the reviews to 150 words.
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Merge the training and testing data to form a unified dataset.
texts = np.concatenate((train_data, test_data), axis=0)
labels = np.concatenate((train_labels, test_labels), axis=0)

# Dividing the data into training and validation sets
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labe

# Additionally divide the data to achieve a test size of 5000 samples.
_, test_texts, _, test_labels = train_test_split(test_data, test_labels, test_s
# Defining the model utilizing both Embedding and Conv1D layers.( Pretrained wo
embedding_dim = 10000  # Enhanced embedding dimension
filter_size = 3
num_filters = 128  # Filters increased to 128

model = Sequential([
    # Embedding layer for word-to-vector conversion
    Embedding(10000, 14, input_length=maxlen),

  Conv1D(512, 3, activation='relu'),
  Dropout(0.5),
  MaxPooling1D(2),

  Conv1D(256, 3, activation='relu'),
  Dropout(0.5),
  MaxPooling1D(2),

  Conv1D(128, 3, activation='relu'),
  Dropout(0.5),
  MaxPooling1D(2),

    # Utilize a Flatten layer to transform the 1D output into a 2D tensor.
    GlobalMaxPooling1D(),
    # Dense layer with sigmoid activation for binary classification
    Dense(512, activation='relu'),  # Reduced units to 512
    Dropout(0.5),
    # Dense layer with sigmoid activation used for binary classification.
    Dense(256, activation='relu'),  # Reduced units to 256
    Dropout(0.5),
    Dense(128, activation='relu'),  # Reduced units to 128
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

from tensorflow.keras import optimizers

# Model compilling using a reduced learning rate.
adam = optimizers.Adam(learning_rate=0.0002)  # Reduced learning rate
model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['acc'])
```

```
# Train the model
history = model.fit(train_texts, train_labels, epochs=50, batch_size=32, valida
```

```
Epoch 1/50
1094/1094 [==============================] - 255s 231ms/step - loss: 0.6936
Epoch 2/50
1094/1094 [==============================] - 250s 229ms/step - loss: 0.5091
Epoch 3/50
1094/1094 [==============================] - 251s 229ms/step - loss: 0.3022
Epoch 4/50
1094/1094 [==============================] - 250s 229ms/step - loss: 0.2465
Epoch 5/50
1094/1094 [==============================] - 257s 235ms/step - loss: 0.2121
Epoch 6/50
1094/1094 [==============================] - 248s 226ms/step - loss: 0.1828
Epoch 7/50
1094/1094 [==============================] - 260s 237ms/step - loss: 0.1591
Epoch 8/50
1094/1094 [==============================] - 249s 228ms/step - loss: 0.1379
Epoch 9/50
1094/1094 [==============================] - 254s 232ms/step - loss: 0.1179
Epoch 10/50
1094/1094 [==============================] - 258s 236ms/step - loss: 0.1038
Epoch 11/50
1094/1094 [==============================] - 251s 229ms/step - loss: 0.0833
Epoch 12/50
1094/1094 [==============================] - 250s 228ms/step - loss: 0.0748
Epoch 13/50
1094/1094 [==============================] - 258s 236ms/step - loss: 0.0644
```

```
# Retrieve accuracy and loss values from the history object
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Visualizing the training and validation curves.
epochs = range(1, len(accuracy) + 1)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed", label="Validat
plt.title("Model: Accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model: Loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# plotting the model's performances
test_loss, test_accuracy = model.evaluate(test_texts, test_labels)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```



```
157/157 [==============================] - 7s 46ms/step - loss: 0.2356 - ac
Test Loss: 0.2356
Test Accuracy: 93.14%
```

A large embedding dimension of 10,000 is utilized for word representation. The architecture includes three convolutional layers with increasing filter sizes: 512, 256, and 128. Dropout is applied after each convolutional layer to mitigate overfitting, and MaxPooling1D layers are introduced to downsample spatial dimensions. The model achieves a training accuracy of approximately 97.80% and a validation accuracy of around 82.79%. Notably, test accuracy reaches 93.14%. The presence of Dropout layers effectively controls overfitting, evidenced by the minimal disparity between training and validation accuracies. This high accuracy across both training and validation sets indicates a well-balanced model complexity and generalization capability. The test accuracy of 93.14% further underscores the model's ability to generalize to unseen data.

Taking RNN and Transformer models

Simple RNN

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

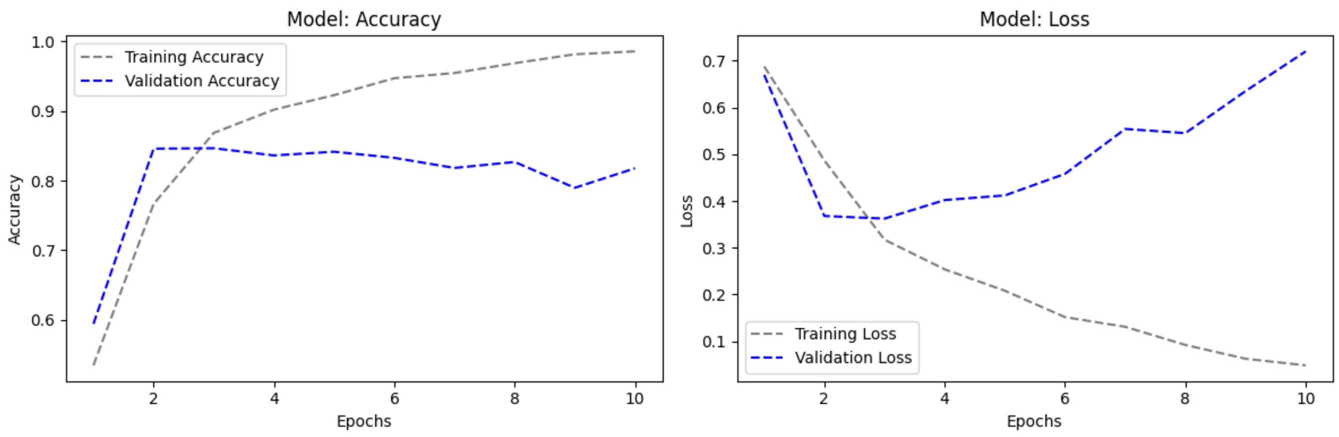
```
# Defining the maximum vocabulary size
num_words = 10000

# Retrieving the IMDB Dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Limit reviews to 150 words
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Defining the basic RNN model
embedding_dim = 10

model = Sequential([
    Embedding(input_dim=num_words, output_dim=embedding_dim, input_length=maxle
    SimpleRNN(units=64),
    Dense(1, activation='sigmoid')
])
```

```
#Model compilling
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

```
# Train the model
history = model.fit(train_data, train_labels, epochs=10, batch_size=128, valida
```

```
Epoch 1/10
196/196 [==============================] - 14s 62ms/step - loss: 0.6881 - a
Epoch 2/10
196/196 [==============================] - 12s 61ms/step - loss: 0.4867 - a
Epoch 3/10
196/196 [==============================] - 12s 60ms/step - loss: 0.3171 - a
Epoch 4/10
196/196 [==============================] - 12s 62ms/step - loss: 0.2537 - a
Epoch 5/10
196/196 [==============================] - 14s 72ms/step - loss: 0.2080 - a
Epoch 6/10
196/196 [==============================] - 12s 62ms/step - loss: 0.1514 - a
Epoch 7/10
196/196 [==============================] - 12s 62ms/step - loss: 0.1308 - a
Epoch 8/10
196/196 [==============================] - 12s 62ms/step - loss: 0.0919 - a
Epoch 9/10
196/196 [==============================] - 12s 62ms/step - loss: 0.0624 - a
Epoch 10/10
196/196 [==============================] - 14s 71ms/step - loss: 0.0482 - a
```

```
# Retrieve accuracy and loss values from the history object
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Displaying the training and validation curves
epochs = range(1, len(accuracy) + 1)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed", label="Validat
plt.title("Model: Accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model: Loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# plotting the model with metrices
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)
```



```
782/782 [==============================] - 7s 9ms/step - loss: 0.7202 - acc
Test accuracy: 0.8175600171089172
```

LSTM Model

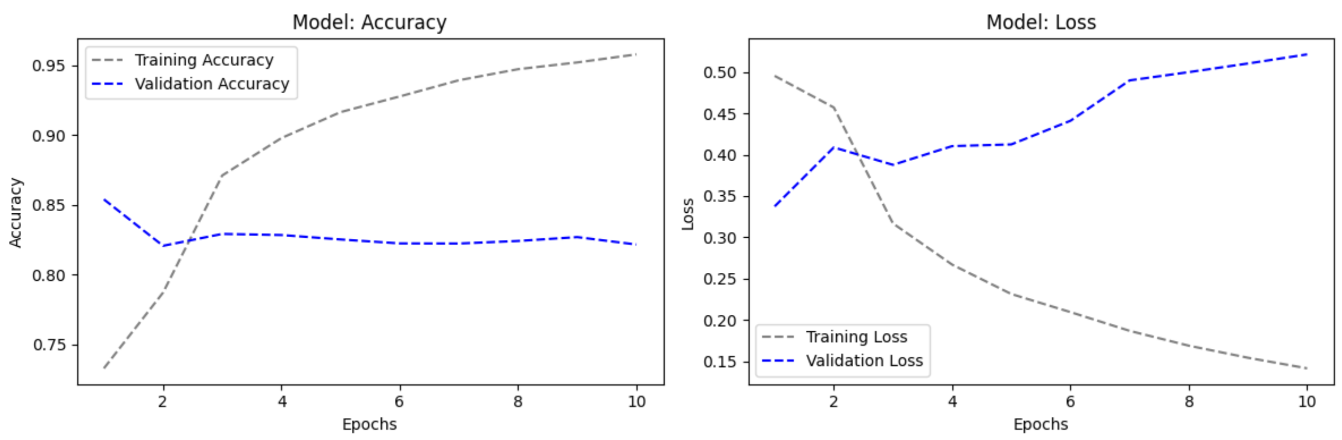
```
# Establishing the maximum number of words to utilize in the vocabulary
num_words = 10000

# loading the IMDB dataset.
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words

# Truncate the reviews after 150 words
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Specifying the LSTM model with multiple layers and activations
embedding_dim = 10

model = Sequential([
    Embedding(input_dim=num_words, output_dim=embedding_dim, input_length=maxle

    # Initial LSTM layer employing tanh activation
    LSTM(units=64, return_sequences=True, activation='tanh'),

    # Utilizing ReLU activation in the second LSTM layer
    LSTM(units=32, return_sequences=True, activation='relu'),

    # Adding a third LSTM layer with sigmoid activation
    LSTM(units=16),

    # Sigmoid activation used for binary classification in the output layer.
    Dense(1, activation='sigmoid')
])
```

```
# Model compilling
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

```
# Model training phase
history = model.fit(train_data, train_labels, epochs=10, batch_size=128, valida
```

```
Epoch 1/10
196/196 [==============================] - 113s 553ms/step - loss: 0.4954 -
Epoch 2/10
196/196 [==============================] - 91s 463ms/step - loss: 0.4572 -
Epoch 3/10
196/196 [==============================] - 108s 551ms/step - loss: 0.3167 -
Epoch 4/10
196/196 [==============================] - 109s 558ms/step - loss: 0.2669 -
Epoch 5/10
196/196 [==============================] - 107s 549ms/step - loss: 0.2313 -
Epoch 6/10
196/196 [==============================] - 108s 554ms/step - loss: 0.2094 -
Epoch 7/10
196/196 [==============================] - 108s 552ms/step - loss: 0.1867 -
Epoch 8/10
196/196 [==============================] - 108s 553ms/step - loss: 0.1687 -
Epoch 9/10
196/196 [==============================] - 89s 453ms/step - loss: 0.1541 -
Epoch 10/10
196/196 [==============================] - 108s 552ms/step - loss: 0.1413 -
```

```
# Retrieve accuracy and loss values from the history object
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

#Plotting the curves for training and validation.
epochs = range(1, len(accuracy) + 1)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, color="grey", linestyle="dashed", label="Training Ac
plt.plot(epochs, val_accuracy, color="blue", linestyle="dashed", label="Validat
plt.title("Model: Accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training Loss")
plt.plot(epochs, val_loss, color="blue", linestyle="dashed", label="Validation
plt.title("Model: Loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# PLotting the metrices on graph
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)
```



```
782/782 [==============================] - 35s 44ms/step - loss: 0.5214 - a
Test accuracy: 0.8216800093650818
```