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  6633k      0  0:00:12  0:00:12 --:--:-- 14.4M
```

Preparing the data

```
import os, pathlib, shutil, random
from tensorflow import keras
batchSize = 32
project_base_directory= pathlib.Path("/content/aclImdb")
validation_data_directory = project_base_directory/ "val"
training_data_directory = project_base_directory / "train"
for category in ("neg", "pos"):
    os.makedirs(validation_data_directory / category, exist_ok=True )

    files = os.listdir(training_data_directory / category)
    random.Random(1496).shuffle(files)
    num_val_samples = 10000
    validation_file_list = files[-num_val_samples:]
    for fname in validation_file_list:
        shutil.move(training_data_directory / category / fname,
                    validation_data_directory / category / fname)

training_dataset = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=batchSize
).take(100)  # Restrict training samples to 100

validation_dataset = keras.utils.text_dataset_from_directory(
    "/content/aclImdb/val", batch_size=batchSize
)
testing_dataset = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=batchSize
)
te_only_training_dataset = training_dataset.map(lambda x, y: x)
```

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

Setting up datasets for numeric sequences

A model that processes sequences represented as one-hot encoded vectors

```
from tensorflow.keras import layers

MAX_SEQUENCE_LENGTH = 150
MAX_VOCAB_SIZE = 10000

# Define TextVectorization layer
t_vectorization = layers.TextVectorization(
    max_tokens=MAX_VOCAB_SIZE,
    output_mode="int",
    output_sequence_length=MAX_SEQUENCE_LENGTH,
)

# Extract texts only from train_ds for vectorization adaptation
training_text_only_dataset = training_dataset.map(lambda x, y: x)
t_vectorization.adapt(training_text_only_dataset)

# Vectorize the train, validation, and test datasets
int_train_dataset = training_dataset.map(
    lambda x, y: (t_vectorization(x), y),
    num_parallel_calls=4
)
int_val_dataset = validation_dataset.map(
    lambda x, y: (t_vectorization(x), y),
    num_parallel_calls=4
)
int_testing_dataset = testing_dataset.map(
```

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

```
import tensorflow as tf  # Model with embedding layer

input_layer = keras.Input(shape=(None,), dtype="int64")
embedding_output = layers.Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=256, mask_zero=True)(input_layer)
x = layers.Bidirectional(layers.LSTM(32))(embedding_output)
x = layers.Dropout(0.5)(x)
output_layer = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(input_layer, output_layer)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, None)	0	–
embedding (Embedding)	(None, None, 256)	2,560,000	input_layer[0][0]
not_equal (NotEqual)	(None, None)	0	input_layer[0][0]
bidirectional (Bidirectional)	(None, 64)	73,984	embedding[0][0], not_equal[0][0]
dropout (Dropout)	(None, 64)	0	bidirectional[0]…
dense (Dense)	(None, 1)	65	dropout[0][0]

Total params: 2,634,049 (10.05 MB)
Trainable params: 2,634,049 (10.05 MB)
Non-trainable params: 0 (0.00 B)

Beginning with the core principles of sequencing

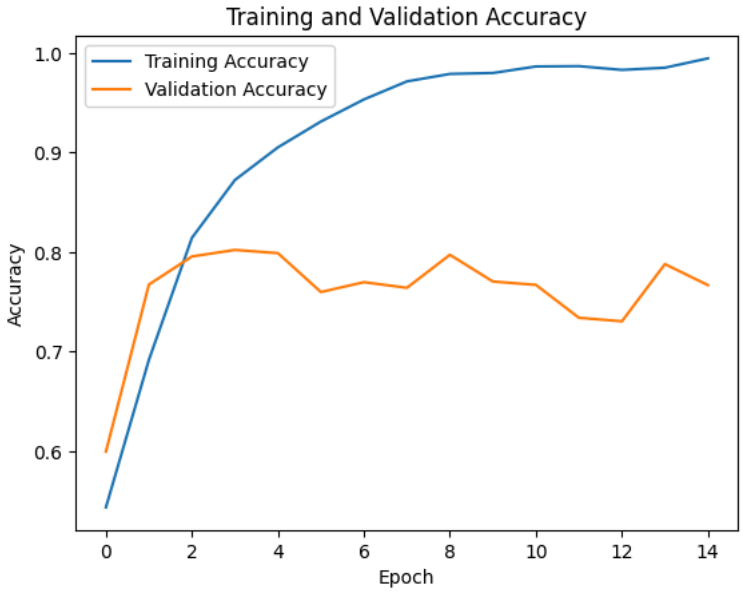
```
checkpoint_callbacks = [
    keras.callbacks.ModelCheckpoint("one_hot_bidir_lstm.keras",
                                    save_best_only=True)
]
# Ensure this is run before plotting
history = model.fit(int_train_dataset, validation_data=int_val_dataset, epochs=15, callbacks=checkpoint_callback
```

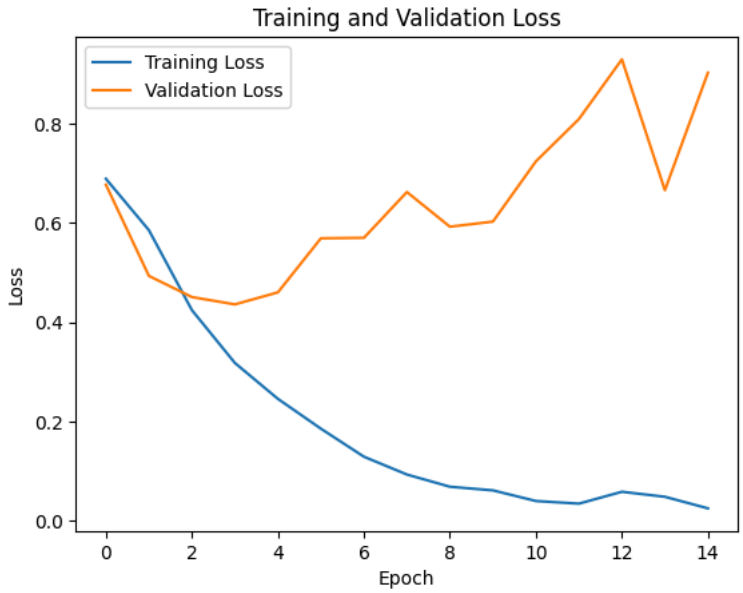
```
Epoch 1/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 75ms/step - accuracy: 0.5349 - loss: 0.6912 - val_accuracy: 0.5997 - val_loss: 0
Epoch 2/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 79ms/step - accuracy: 0.6668 - loss: 0.6216 - val_accuracy: 0.7675 - val_loss: 0.
Epoch 3/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 73ms/step - accuracy: 0.8043 - loss: 0.4381 - val_accuracy: 0.7957 - val_loss: 0.
Epoch 4/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 71ms/step - accuracy: 0.8761 - loss: 0.3185 - val_accuracy: 0.8023 - val_loss: 0
Epoch 5/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9141 - loss: 0.2290 - val_accuracy: 0.7990 - val_loss: 0.
Epoch 6/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9390 - loss: 0.1755 - val_accuracy: 0.7600 - val_loss: 0.
Epoch 7/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 69ms/step - accuracy: 0.9502 - loss: 0.1329 - val_accuracy: 0.7699 - val_loss: 0.
Epoch 8/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 74ms/step - accuracy: 0.9768 - loss: 0.0822 - val_accuracy: 0.7642 - val_loss: 0
Epoch 9/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9838 - loss: 0.0627 - val_accuracy: 0.7974 - val_loss: 0.
Epoch 10/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 70ms/step - accuracy: 0.9779 - loss: 0.0557 - val_accuracy: 0.7706 - val_loss: 0.
Epoch 11/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9908 - loss: 0.0340 - val_accuracy: 0.7672 - val_loss: 0.
Epoch 12/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 79ms/step - accuracy: 0.9921 - loss: 0.0246 - val_accuracy: 0.7341 - val_loss: 0
Epoch 13/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 69ms/step - accuracy: 0.9853 - loss: 0.0456 - val_accuracy: 0.7306 - val_loss: 0.
Epoch 14/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9871 - loss: 0.0404 - val_accuracy: 0.7880 - val_loss: 0.
Epoch 15/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9995 - loss: 0.0097 - val_accuracy: 0.7670 - val_loss: 0.
```

```
model = keras.models.load_model('one_hot_bidir_lstm.keras')
print(f"Test acc: {model.evaluate(int_testing_dataset)[1]:.3f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.7949 - loss: 0.4473
Test acc: 0.793
```

```
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Using the Embedded level to extract embedded words

Using the Embedded level to extract embedded words

```
em_layer = layers.Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=256)
```

Anchor layer system developed from the ground up

```
in1 = keras.Input(shape=(None,), dtype="int64")
em1 = layers.Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=256)(in1)
x = layers.Bidirectional(layers.LSTM(32))(em1)
x = layers.Dropout(0.5)(x)
output_layer1 = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(in1, output_layer1)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

Model: "functional_1"

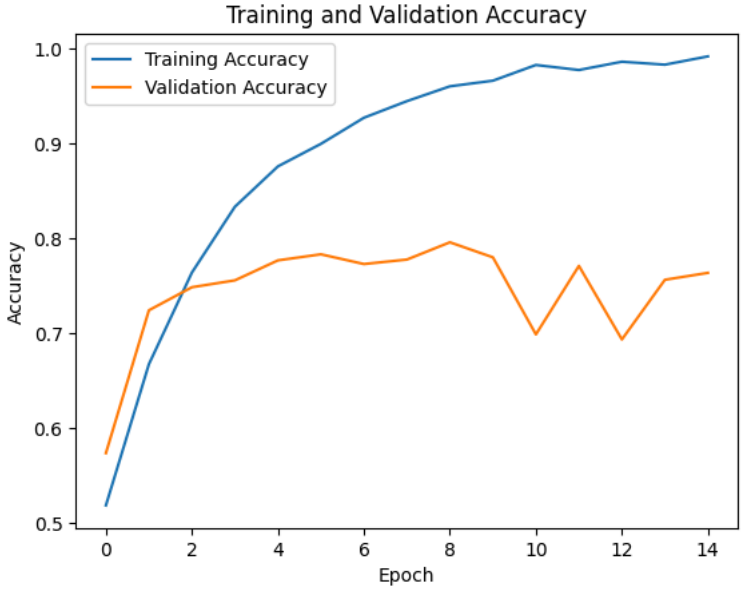
Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, None)	0
embedding_2 (Embedding)	(None, None, 256)	2,560,000
bidirectional_1 (Bidirectional)	(None, 64)	73,984
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

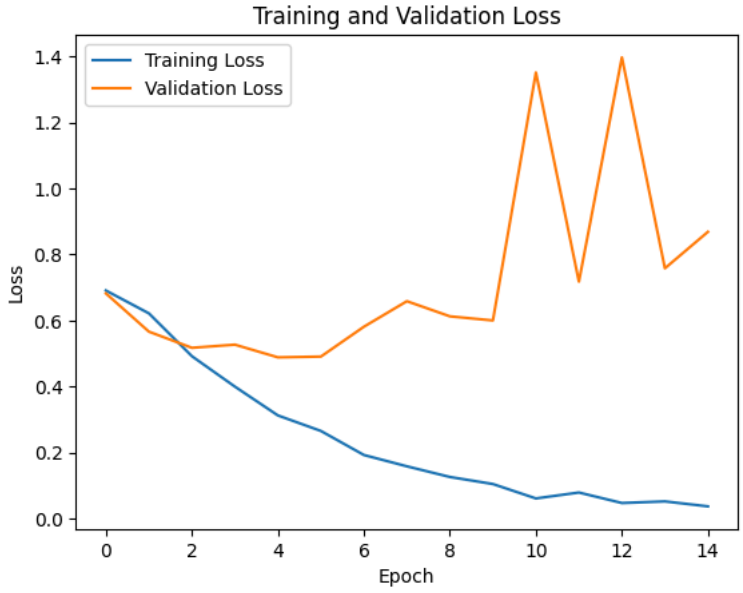
Total params: 2,634,049 (10.05 MB)
Trainable params: 2,634,049 (10.05 MB)
Non-trainable params: 0 (0.00 B)

```
checkpoint_callbacks1 = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru.keras",  # Change to .keras
                                    save_best_only=True)
]

history1 = model.fit(int_train_dataset, validation_data=int_val_dataset, epochs=15, callbacks=checkpoint_callbac

# Load the best model saved by the callback
model = keras.models.load_model("embeddings_bidir_gru.keras")

# Evaluate the model on the test dataset
print(f"Test acc: {model.evaluate(int_testing_dataset)[1]:.3f}")
```

```
Epoch 1/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 77ms/step - accuracy: 0.4960 - loss: 0.6929 - val_accuracy: 0.5738 - val_loss: 0.
Epoch 2/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 15s 121ms/step - accuracy: 0.6517 - loss: 0.6457 - val_accuracy: 0.7243 - val_loss:
Epoch 3/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 121ms/step - accuracy: 0.7557 - loss: 0.5115 - val_accuracy: 0.7486 - val_loss:
Epoch 4/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 76ms/step - accuracy: 0.8392 - loss: 0.3909 - val_accuracy: 0.7559 - val_loss: 0.
Epoch 5/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - accuracy: 0.8764 - loss: 0.3175 - val_accuracy: 0.7769 - val_loss: 0
Epoch 6/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 70ms/step - accuracy: 0.9017 - loss: 0.2587 - val_accuracy: 0.7833 - val_loss: 0
Epoch 7/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 66ms/step - accuracy: 0.9298 - loss: 0.1904 - val_accuracy: 0.7731 - val_loss: 0
Epoch 8/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 74ms/step - accuracy: 0.9553 - loss: 0.1401 - val_accuracy: 0.7778 - val_loss: 0.
Epoch 9/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 69ms/step - accuracy: 0.9624 - loss: 0.1204 - val_accuracy: 0.7959 - val_loss: 0.
Epoch 10/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 70ms/step - accuracy: 0.9657 - loss: 0.0956 - val_accuracy: 0.7801 - val_loss: 0.
Epoch 11/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 74ms/step - accuracy: 0.9859 - loss: 0.0529 - val_accuracy: 0.6989 - val_loss: 1
Epoch 12/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 70ms/step - accuracy: 0.9733 - loss: 0.0869 - val_accuracy: 0.7711 - val_loss: 0.
Epoch 13/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 67ms/step - accuracy: 0.9915 - loss: 0.0337 - val_accuracy: 0.6935 - val_loss: 1
Epoch 14/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 67ms/step - accuracy: 0.9812 - loss: 0.0606 - val_accuracy: 0.7564 - val_loss: 0
Epoch 15/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 73ms/step - accuracy: 0.9969 - loss: 0.0227 - val_accuracy: 0.7638 - val_loss: 0.
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.7673 - loss: 0.4942
Test acc: 0.769
```

```
# Visualize training and validation accuracy
plt.plot(history1.history['accuracy'], label='Training Accuracy')
plt.plot(history1.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Visualize training and validation loss
plt.plot(history1.history['loss'], label='Training Loss')
plt.plot(history1.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Detecting blur and performing inpainting Applying filters at the Anchoring level for blur recognition and filling

```
in2 = keras.Input(shape=(None,), dtype="int64")
em2 = layers.Embedding(
    input_dim=MAX_VOCAB_SIZE, output_dim=256, mask_zero=True)(in2)
x = layers.Bidirectional(layers.LSTM(32))(em2)
x = layers.Dropout(0.5)(x)
output_layer2 = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(in2, output_layer2)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

Model: "functional_2"

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, None)	0	–
embedding_3 (Embedding)	(None, None, 256)	2,560,000	input_layer_2[0]…
not_equal_2 (NotEqual)	(None, None)	0	input_layer_2[0]…
bidirectional_2 (Bidirectional)	(None, 64)	73,984	embedding_3[0][0… not_equal_2[0][0]
dropout_2 (Dropout)	(None, 64)	0	bidirectional_2[…
dense_2 (Dense)	(None, 1)	65	dropout_2[0][0]

Total params: 2,634,049 (10.05 MB)
Trainable params: 2,634,049 (10.05 MB)
Non-trainable params: 0 (0.00 B)

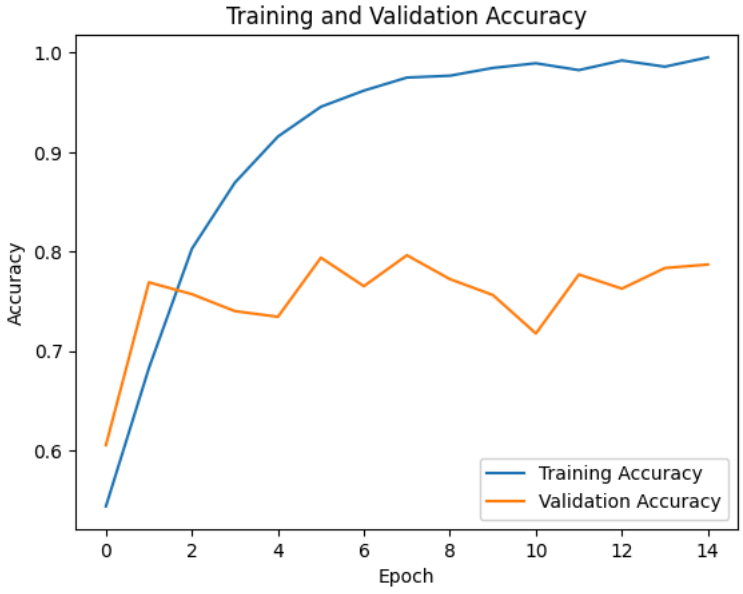
```
checkpoint_callbacks2 = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru_with_masking.keras",
                                    save_best_only=True)
]
history2=model.fit(int_train_dataset, validation_data= int_val_dataset, epochs=15, callbacks=checkpoint_callback
```

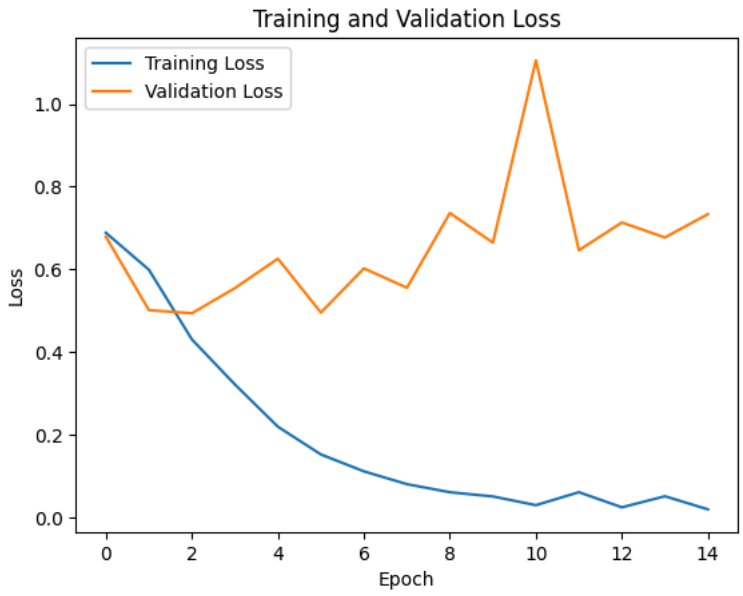
```
Epoch 1/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 91ms/step - accuracy: 0.5147 - loss: 0.6913 - val_accuracy: 0.6054 - val_loss: 0
Epoch 2/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 79ms/step - accuracy: 0.6681 - loss: 0.6346 - val_accuracy: 0.7691 - val_loss: 0.
Epoch 3/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 71ms/step - accuracy: 0.7847 - loss: 0.4635 - val_accuracy: 0.7571 - val_loss: 0.
Epoch 4/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 75ms/step - accuracy: 0.8720 - loss: 0.3204 - val_accuracy: 0.7401 - val_loss: 0
Epoch 5/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 78ms/step - accuracy: 0.9064 - loss: 0.2325 - val_accuracy: 0.7344 - val_loss: 0
Epoch 6/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 120ms/step - accuracy: 0.9381 - loss: 0.1766 - val_accuracy: 0.7939 - val_loss:
Epoch 7/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9664 - loss: 0.0994 - val_accuracy: 0.7652 - val_loss: 0.
Epoch 8/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 71ms/step - accuracy: 0.9776 - loss: 0.0743 - val_accuracy: 0.7963 - val_loss: 0.
Epoch 9/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 78ms/step - accuracy: 0.9808 - loss: 0.0544 - val_accuracy: 0.7723 - val_loss: 0.
Epoch 10/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 72ms/step - accuracy: 0.9930 - loss: 0.0322 - val_accuracy: 0.7563 - val_loss: 0.
Epoch 11/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 71ms/step - accuracy: 0.9916 - loss: 0.0271 - val_accuracy: 0.7176 - val_loss: 1
Epoch 12/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 78ms/step - accuracy: 0.9863 - loss: 0.0446 - val_accuracy: 0.7770 - val_loss: 0.
Epoch 13/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 81ms/step - accuracy: 0.9966 - loss: 0.0129 - val_accuracy: 0.7627 - val_loss: 0
Epoch 14/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 120ms/step - accuracy: 0.9951 - loss: 0.0185 - val_accuracy: 0.7835 - val_loss:
Epoch 15/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 120ms/step - accuracy: 0.9995 - loss: 0.0092 - val_accuracy: 0.7870 - val_loss:
```

```
model = keras.models.load_model("embeddings_bidir_gru_with_masking.keras")
print(f"Test acc: {model.evaluate(int_testing_dataset)[1]:.3f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.7504 - loss: 0.5039
Test acc: 0.748
```

```
# Plot training and validation accuracy
plt.plot(history2.history['accuracy'], label='Training Accuracy')
plt.plot(history2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history2.history['loss'], label='Training Loss')
plt.plot(history2.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Pre-trained word embeddings are utilized

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

```
--2025-11-14 17:37:07--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-11-14 17:37:07--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
```

```
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-11-14 17:37:08--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

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

2025-11-14 17:39:49 (5.12 MB/s) - 'glove.6B.zip' saved [862182613/862182613]
```

Analyzing the word-embeddings package for One

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

glove_embeddings = {}
with open(GLOVE_FILE_PATH) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        glove_embeddings[word] = coefs

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

```
Found 400000 word vectors.
```

Constructing the matrix for GloVe-embedded words

```
em_dim = 100

vocab = t_vectorization.get_vocabulary()
word_to_index = dict(zip(vocab, range(len(vocab))))

embedding_matrix = np.zeros((MAX_VOCAB_SIZE, em_dim))
for word, i in word_to_index.items():
    if i < MAX_VOCAB_SIZE:
        em_vector = glove_embeddings.get(word)
    if em_vector is not None:
        embedding_matrix[i] = em_vector
```

```
em_layer = layers.Embedding(
    MAX_VOCAB_SIZE,
    em_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
```

Architecture incorporating a trained embedded level

```
in4 = keras.Input(shape=(None,), dtype="int64")
em4 = em_layer(in4)
x = layers.Bidirectional(layers.LSTM(32))(em4)
x = layers.Dropout(0.5)(x)
output_layer4 = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(in4, output_layer4)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

Model: "functional_3"

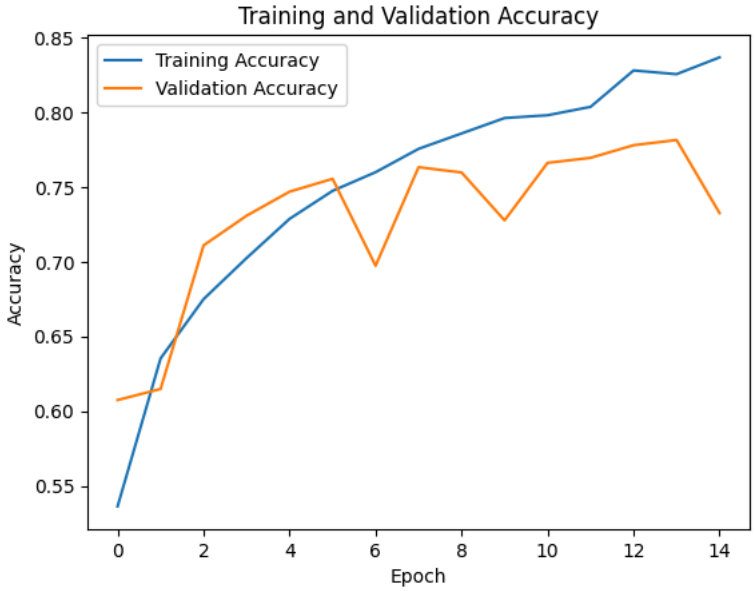
Layer (type)	Output Shape	Param #	Connected to
input_layer_3 (InputLayer)	(None, None)	0	–
embedding_4 (Embedding)	(None, None, 100)	1,000,000	input_layer_3[0]…
not_equal_4 (NotEqual)	(None, None)	0	input_layer_3[0]…
bidirectional_3 (Bidirectional)	(None, 64)	34,048	embedding_4[0][0]… not_equal_4[0][0]
dropout_3 (Dropout)	(None, 64)	0	bidirectional_3[…
dense_3 (Dense)	(None, 1)	65	dropout_3[0][0]

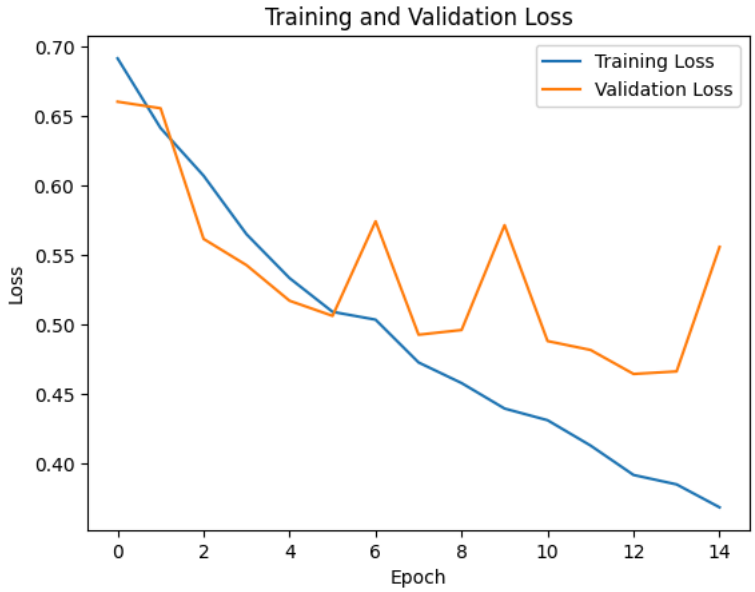
Total params: 1,034,113 (3.94 MB)
Trainable params: 34,113 (133.25 KB)
Non-trainable params: 1,000,000 (3.81 MB)

```
checkpoint_callbacks4 = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]

history4=model.fit(int_train_dataset, validation_data= int_val_dataset, epochs=15, callbacks=checkpoint_callback
model = keras.models.load_model("glove_embeddings_sequence_model.keras")
print(f"Test Accuracy: {model.evaluate(int_testing_dataset)[1]:.3f}")
```

```
Epoch 1/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 99ms/step - accuracy: 0.5161 - loss: 0.7036 - val_accuracy: 0.6075 - val_loss: 0
Epoch 2/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 104ms/step - accuracy: 0.6184 - loss: 0.6550 - val_accuracy: 0.6148 - val_loss:
Epoch 3/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 135ms/step - accuracy: 0.6641 - loss: 0.6125 - val_accuracy: 0.7110 - val_loss:
Epoch 4/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 103ms/step - accuracy: 0.6832 - loss: 0.5782 - val_accuracy: 0.7308 - val_loss:
Epoch 5/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 85ms/step - accuracy: 0.7313 - loss: 0.5306 - val_accuracy: 0.7469 - val_loss: 0.
Epoch 6/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 100ms/step - accuracy: 0.7474 - loss: 0.5133 - val_accuracy: 0.7556 - val_loss:
Epoch 7/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 119ms/step - accuracy: 0.7600 - loss: 0.4974 - val_accuracy: 0.6974 - val_loss:
Epoch 8/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 13s 135ms/step - accuracy: 0.7751 - loss: 0.4704 - val_accuracy: 0.7634 - val_loss:
Epoch 9/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.7755 - loss: 0.4603 - val_accuracy: 0.7598 - val_loss: 0.
Epoch 10/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 75ms/step - accuracy: 0.8020 - loss: 0.4279 - val_accuracy: 0.7278 - val_loss: 0.
Epoch 11/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 101ms/step - accuracy: 0.8011 - loss: 0.4272 - val_accuracy: 0.7663 - val_loss:
Epoch 12/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 92ms/step - accuracy: 0.8048 - loss: 0.4155 - val_accuracy: 0.7696 - val_loss: 0.
Epoch 13/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 94ms/step - accuracy: 0.8255 - loss: 0.3949 - val_accuracy: 0.7781 - val_loss: 0.
Epoch 14/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 120ms/step - accuracy: 0.8216 - loss: 0.3959 - val_accuracy: 0.7815 - val_loss:
Epoch 15/15
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.8437 - loss: 0.3641 - val_accuracy: 0.7326 - val_loss: 0.
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.7806 - loss: 0.4604
Test Accuracy: 0.778
```

```
# Visualize training and validation accuracy
plt.plot(history4.history['accuracy'], label='Training Accuracy')
plt.plot(history4.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Visualize training and validation loss
plt.plot(history4.history['loss'], label='Training Loss')
plt.plot(history4.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

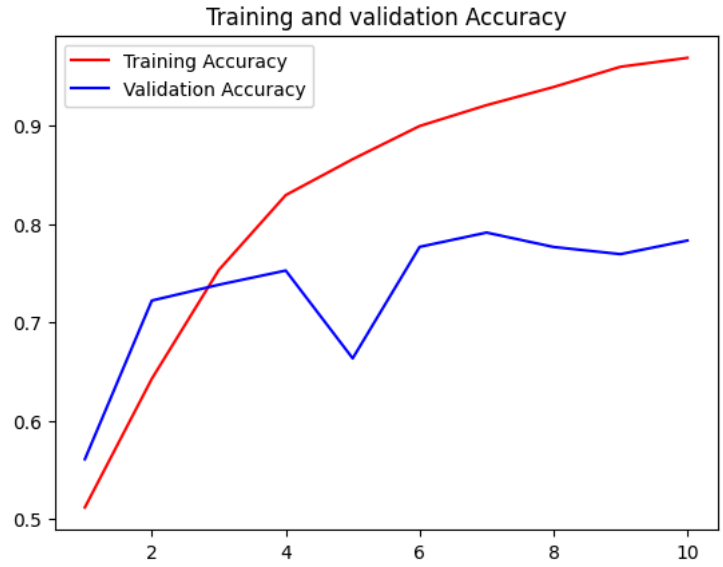




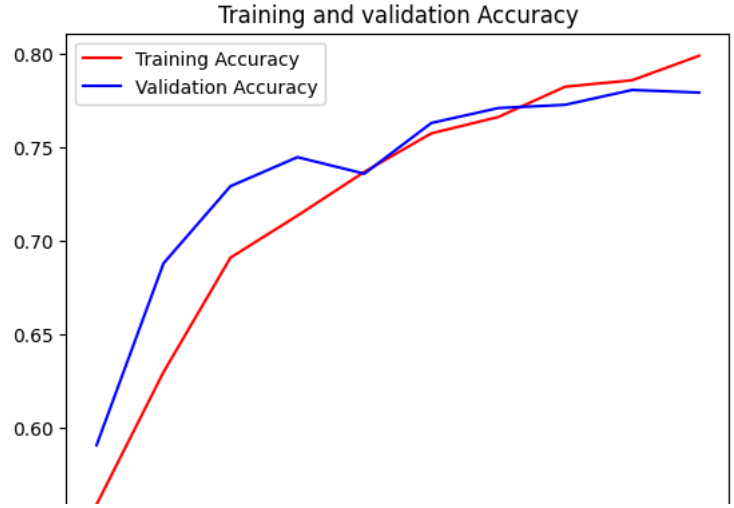

```
training_sample_size_list = [100, 500, 1000, 5000, 10000, 20000]
for train_size in training_sample_size_list:
    training_dataset = keras.utils.text_dataset_from_directory(
        "aclImdb/train", batch_size=batchSize
    ).take(train_size)

    int_training_dataset = training_dataset.map(
        lambda x, y: (t_vectorization(x), y),
        num_parallel_calls=4
    )
    int_validation_dataset = validation_dataset.map(
        lambda x, y: (t_vectorization(x), y),
        num_parallel_calls=4
    )
    int_testing_dataset = testing_dataset.map(
        lambda x, y: (t_vectorization(x), y),
        num_parallel_calls=4
    )

    # Train and evaluate the model with the embedding layer
    embedding_layer = layers.Embedding(MAX_VOCAB_SIZE, em_dim)

    inputs = keras.Input(shape=(None,), dtype="int64")
    embedded = embedding_layer(inputs)
    x = layers.Bidirectional(layers.LSTM(32))(embedded)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(1, activation="sigmoid")(x)
    model = keras.Model(inputs, outputs)
    model.compile(optimizer="rmsprop",
                  loss="binary_crossentropy",
                  metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_model.keras", save_best_only=True)
]
history = model.fit(int_training_dataset, validation_data=int_validation_dataset, epochs=10, callbacks=callb
model = keras.models.load_model("embeddings_model.keras")
embedding_layer_test_acc = model.evaluate(int_testing_dataset)[1]

loss = history.history["accuracy"]
validation_loss_value = history.history["val_accuracy"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "r", label="Training Accuracy")
plt.plot(epochs, validation_loss_value, "b", label="Validation Accuracy")
plt.title("Training and validation Accuracy")
plt.legend()
plt.show()

# Train and evaluate the model with the pretrained word embeddings
embedding_layer = layers.Embedding(
    MAX_VOCAB_SIZE,
    em_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)

inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint("pretrained_embeddings_model.keras", save_best_only=True)
]
history = model.fit(int_training_dataset, validation_data=int_validation_dataset, epochs=10, callbacks=callb
model = keras.models.load_model("pretrained_embeddings_model.keras")
pretrained_embeddings_test_acc = model.evaluate(int_testing_dataset)[1]

loss = history.history["accuracy"]
validation_loss_value = history.history["val_accuracy"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "r", label="Training Accuracy")
plt.plot(epochs, validation_loss_value, "b", label="Validation Accuracy")
plt.title("Training and validation Accuracy")
plt.legend()
plt.show()

# Compare the performance and store the results
print(f"Training samples: {train_size}")
print(f"Embedding layer test accuracy: {embedding_layer_test_acc:.3f}")
print(f"Pretrained embeddings test accuracy: {pretrained_embeddings_test_acc:.3f}")
print("-" * 50)
```

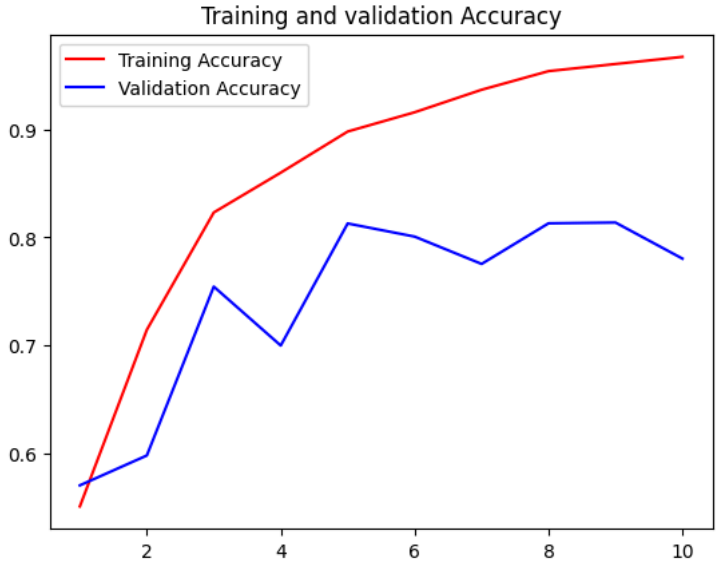
```
Found 5000 files belonging to 2 classes.
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 76ms/step - accuracy: 0.5082 - loss: 0.6934 - val_accuracy: 0.5606 - val_loss:
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 79ms/step - accuracy: 0.6214 - loss: 0.6674 - val_accuracy: 0.7221 - val_loss: (
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - accuracy: 0.7427 - loss: 0.5511 - val_accuracy: 0.7382 - val_loss: (
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 79ms/step - accuracy: 0.8165 - loss: 0.4385 - val_accuracy: 0.7526 - val_loss: (
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 65ms/step - accuracy: 0.8707 - loss: 0.3359 - val_accuracy: 0.6633 - val_loss: (
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 80ms/step - accuracy: 0.8925 - loss: 0.2773 - val_accuracy: 0.7767 - val_loss: (
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - accuracy: 0.9404 - loss: 0.1878 - val_accuracy: 0.7912 - val_loss: (
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 79ms/step - accuracy: 0.9467 - loss: 0.1777 - val_accuracy: 0.7766 - val_loss: (
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - accuracy: 0.9703 - loss: 0.1116 - val_accuracy: 0.7693 - val_loss: (
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.9747 - loss: 0.0813 - val_accuracy: 0.7832 - val_loss: (
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.7631 - loss: 0.4926
```




```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 105ms/step - accuracy: 0.5543 - loss: 0.6939 - val_accuracy: 0.5905 - val_loss
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 86ms/step - accuracy: 0.6100 - loss: 0.6495 - val_accuracy: 0.6879 - val_loss: (
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 99ms/step - accuracy: 0.7005 - loss: 0.5921 - val_accuracy: 0.7292 - val_loss:
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 99ms/step - accuracy: 0.7187 - loss: 0.5534 - val_accuracy: 0.7448 - val_loss:
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 100ms/step - accuracy: 0.7434 - loss: 0.5119 - val_accuracy: 0.7359 - val_loss
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 88ms/step - accuracy: 0.7729 - loss: 0.4835 - val_accuracy: 0.7631 - val_loss: (
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 100ms/step - accuracy: 0.7751 - loss: 0.4844 - val_accuracy: 0.7710 - val_loss
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 99ms/step - accuracy: 0.7875 - loss: 0.4557 - val_accuracy: 0.7728 - val_loss:
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - accuracy: 0.7916 - loss: 0.4431 - val_accuracy: 0.7807 - val_loss: (
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 16s 158ms/step - accuracy: 0.8031 - loss: 0.4270 - val_accuracy: 0.7793 - val_loss
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 9ms/step - accuracy: 0.7752 - loss: 0.4710
```




```
Training samples: 100
Embedding layer test accuracy: 0.761
Pretrained embeddings test accuracy: 0.775
--------------------------------------------------
Found 5000 files belonging to 2 classes.
Epoch 1/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 11s 58ms/step - accuracy: 0.5305 - loss: 0.6912 - val_accuracy: 0.5709 - val_loss:
Epoch 2/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 9s 56ms/step - accuracy: 0.6862 - loss: 0.6022 - val_accuracy: 0.5985 - val_loss: (
Epoch 3/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - accuracy: 0.8082 - loss: 0.4441 - val_accuracy: 0.7545 - val_loss: (
Epoch 4/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - accuracy: 0.8484 - loss: 0.3668 - val_accuracy: 0.7000 - val_loss: (
Epoch 5/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 10s 57ms/step - accuracy: 0.8945 - loss: 0.2855 - val_accuracy: 0.8128 - val_loss:
Epoch 6/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - accuracy: 0.9190 - loss: 0.2227 - val_accuracy: 0.8007 - val_loss: (
Epoch 7/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - accuracy: 0.9406 - loss: 0.1769 - val_accuracy: 0.7754 - val_loss: (
Epoch 8/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 14s 81ms/step - accuracy: 0.9513 - loss: 0.1558 - val_accuracy: 0.8130 - val_loss:
Epoch 9/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - accuracy: 0.9594 - loss: 0.1243 - val_accuracy: 0.8137 - val_loss: (
Epoch 10/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - accuracy: 0.9672 - loss: 0.1094 - val_accuracy: 0.7804 - val_loss: (
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.8022 - loss: 0.4534
```




```
Epoch 1/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 12s 70ms/step - accuracy: 0.5496 - loss: 0.6895 - val_accuracy: 0.6528 - val_loss:
Epoch 2/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 8s 52ms/step - accuracy: 0.6540 - loss: 0.6187 - val_accuracy: 0.6572 - val_loss: (
Epoch 3/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 11s 69ms/step - accuracy: 0.7108 - loss: 0.5704 - val_accuracy: 0.6647 - val_loss:
Epoch 4/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 11s 69ms/step - accuracy: 0.7391 - loss: 0.5330 - val_accuracy: 0.7452 - val_loss:
Epoch 5/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 11s 69ms/step - accuracy: 0.7670 - loss: 0.4920 - val_accuracy: 0.7746 - val_loss:
Epoch 6/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 8s 54ms/step - accuracy: 0.7873 - loss: 0.4680 - val_accuracy: 0.7220 - val_loss: (
Epoch 7/10
157/157 ━━━━━━━━━━━━━━━━━━━━ 11s 61ms/step - accuracy: 0.8054 - loss: 0.4390 - val_accuracy: 0.7864 - val_loss:
Epoch 8/10
```