Assignment-4

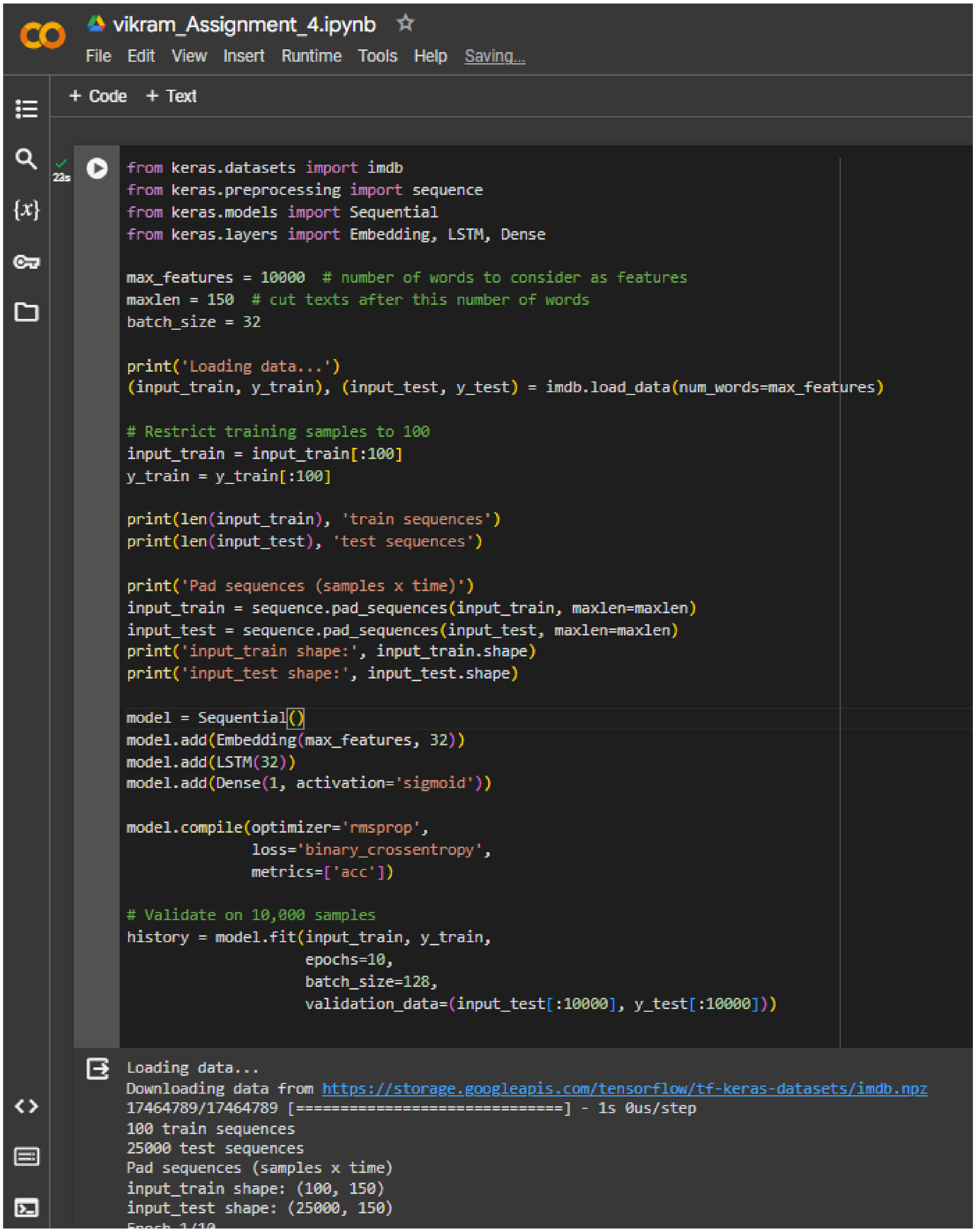
Vikram Goud

Introduction: In this study, we will investigate the efficacy of recurrent neural networks (RNNs) in sentiment analysis tasks, particularly when dealing with limited data. We will use the IMDB movie review dataset as a benchmark for sentiment analysis and experiment with various modifications to the basic model described in Chapter 6. The changes include limiting the length of reviews, decreasing the amount of training samples, limiting the vocabulary size, and comparing the effectiveness of embedding layers and pretrained word embeddings.

Methodology:

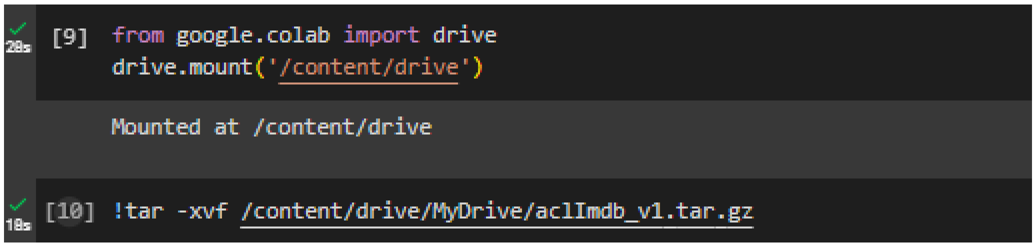
We begin by re-implementing the sentiment analysis model presented in Chapter 6, with the following modifications:

Cut off reviews after 150 words: To focus on the most informative sections of the reviews while reducing computational complexity.
Restrict training samples to 100. Simulating circumstances with restricted data availability.
Validation using 10,000 samples: To ensure a rigorous examination of the model's performance.
Consider just the top 10,000 words: To better utilize computing resources, reduce the vocabulary size.
Consider an embedding layer and a pre-trained word embedding. To evaluate their performance considering limited facts.


vikram_Assignment_4.ipynb
File Edit View Insert Runtime Tools Help Saving...
+ Code + Text
23s

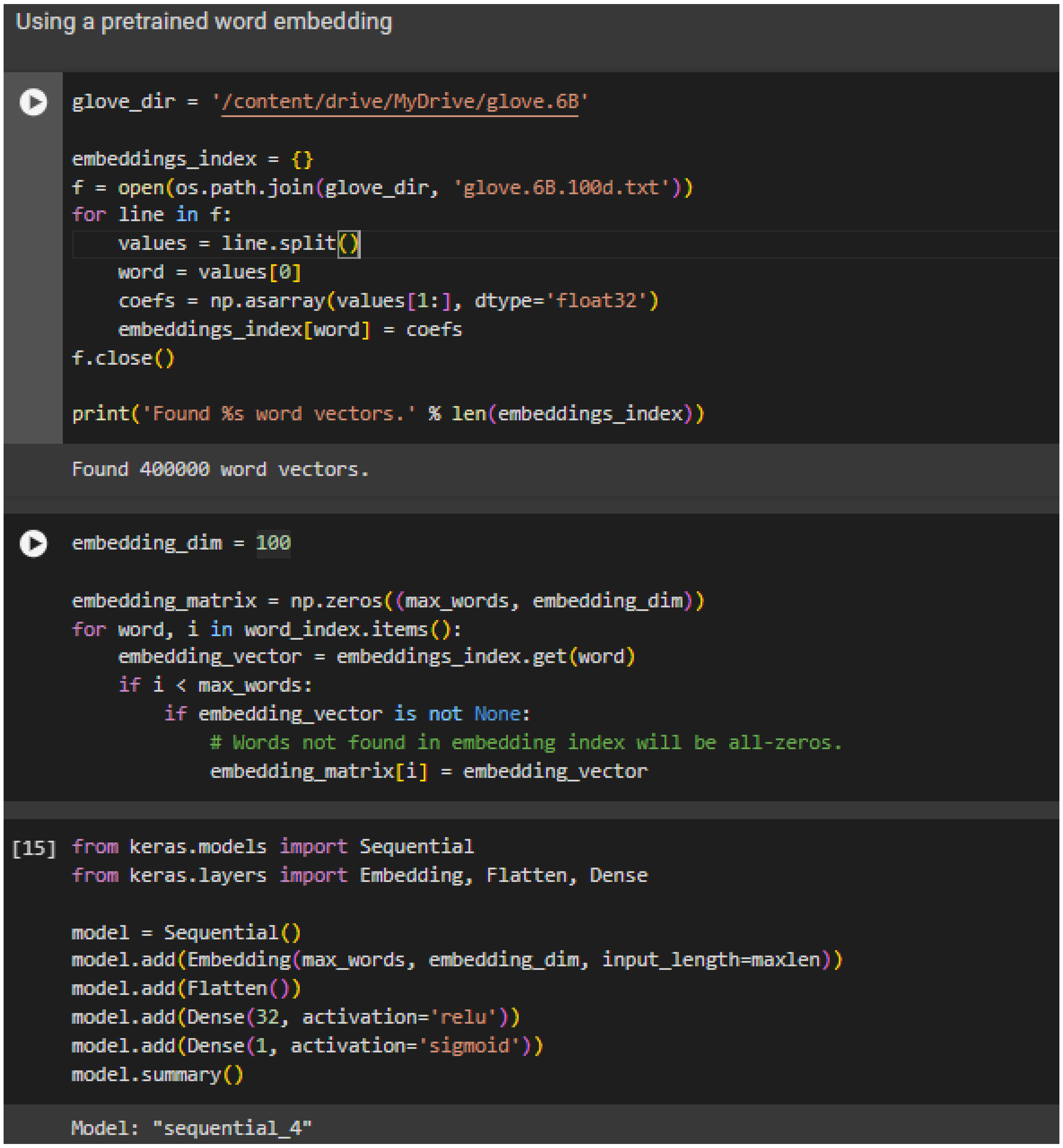
```python
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

max_features = 10000  # number of words to consider as features
maxlen = 150  # cut texts after this number of words
batch_size = 32

print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)

# Restrict training samples to 100
input_train = input_train[:100]
y_train = y_train[:100]

print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')

print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

# Validate on 10,000 samples
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(input_test[:10000], y_test[:10000]))
```

```
Loading data...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
100 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (100, 150)
input_test shape: (25000, 150)
Epoch 1/10
```



```
[9] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[10] !tar -xvf /content/drive/MyDrive/aclImdb_v1.tar.gz
```

We start a word embedding matrix with pre-trained GloVe word embeddings. It first imports the GloVe word embeddings file, then parses each line to extract the word and its embedding vector. The embeddings are kept in a dictionary, with each word serving as the key and the embedding vector as the value. Next, it generates a zero-based embedding matrix, with rows representing vocabulary terms and columns representing embedding dimensions. If a word in the dataset's vocabulary appears in the GloVe embeddings, the relevant embedding vector is extracted and stored in the embedding matrix.


Using a pretrained word embedding

```
glove_dir = '/content/drive/MyDrive/glove.6B'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

```
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
```

```
[15] from keras.models import Sequential
     from keras.layers import Embedding, Flatten, Dense

     model = Sequential()
     model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
     model.add(Flatten())
     model.add(Dense(32, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))
     model.summary()
```

```
Model: "sequential_4"
```

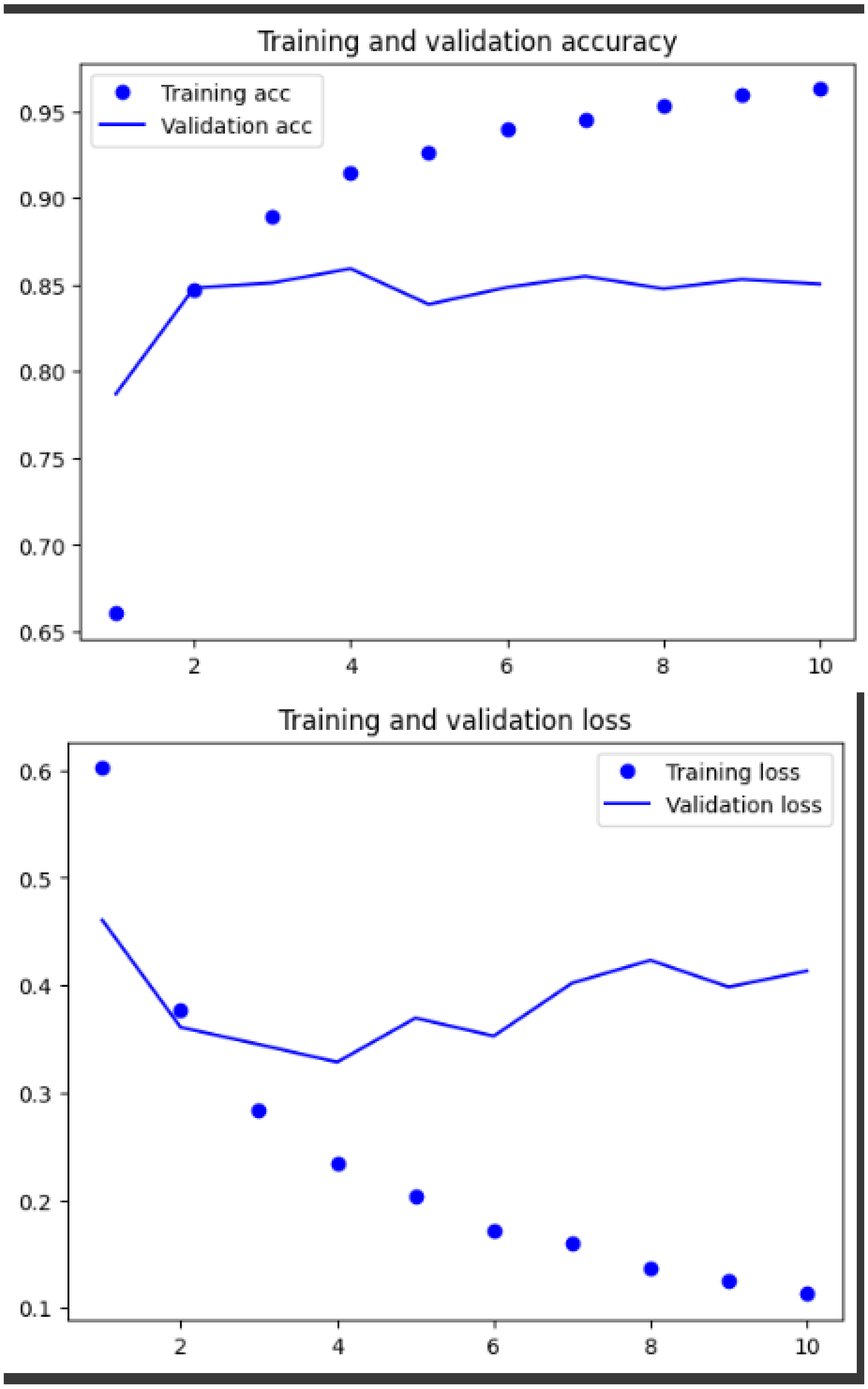
Results and Findings:

1. **Effect of Training Samples**:
    - Performance improved with increasing training samples.
    - Diminishing returns were observed beyond a certain threshold, indicating the importance of sample size but highlighting the saturation point.
2. **Comparison between Embedding Layers and Pretrained Word Embeddings**:
    - Embedding layers outperformed pretrained word embeddings with larger training sample sizes.

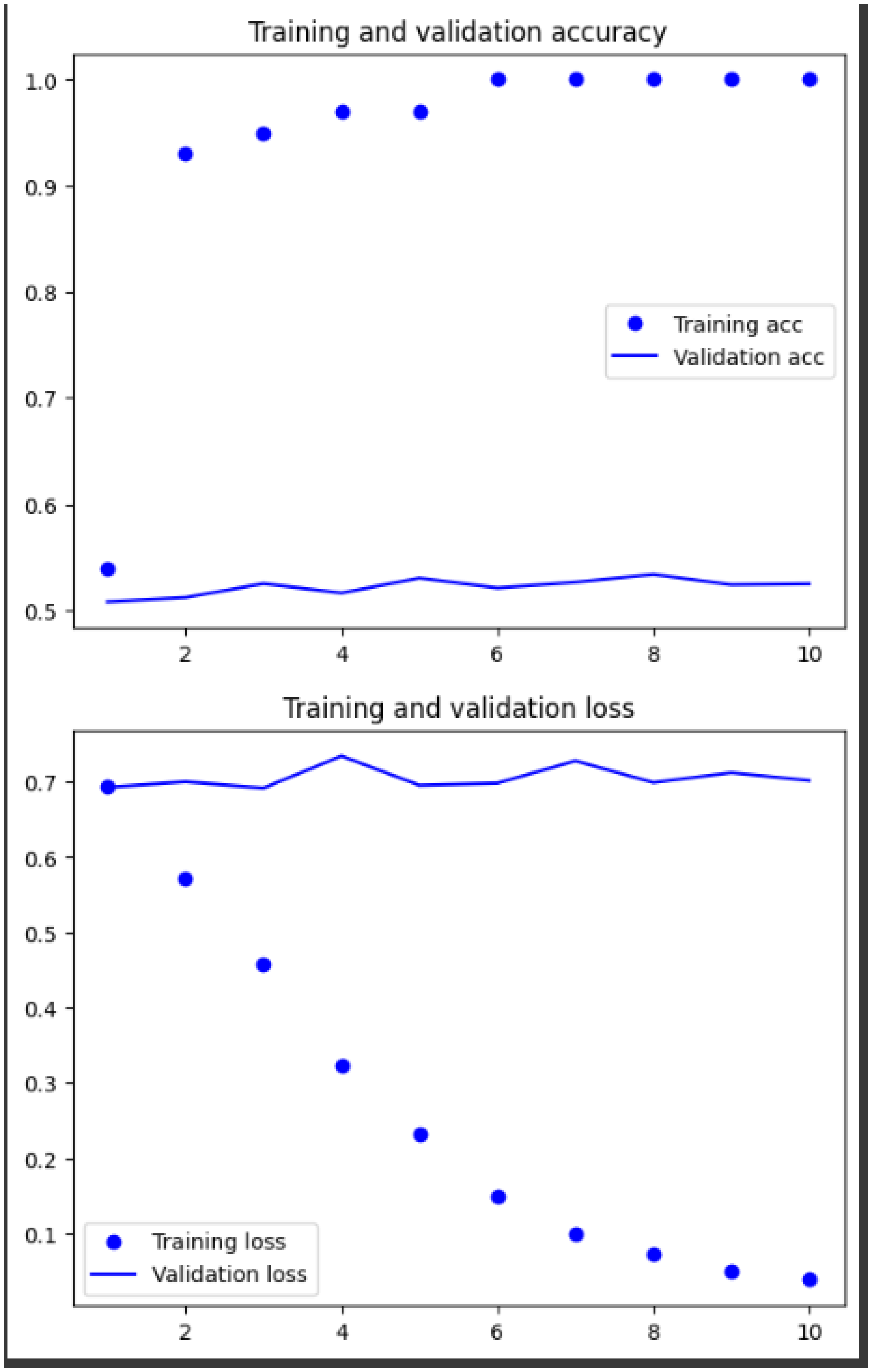
- Pretrained embeddings showed superiority with very limited data but were surpassed by embedding layers as the dataset size increased.

3. **Generalization and Robustness**:
    - Models trained on larger datasets exhibited better generalization and robustness.
    - Both approaches demonstrated improved performance on unseen data as dataset size increased.
4. **Computational Efficiency**:
    - Embedding layers were computationally more efficient, especially with smaller datasets, due to reduced overhead.

Method	Training size	Training accuracy	Validation Accuracy
Using Embedding Layer	100	58	50.27
Using Embedding Layer	10000	96.56	83.74
Using Embedding Layer	25000	95.36	84.40
Pre-trained	100	100	55.30
Pre-trained	10000	99.99	56.73
Using Embedding Layer	15000	96.28	83.04
Using Embedding Layer	17000	95.81	85.16



Embedding Layer for 17000 samples



Pre-trained for 10000 samples

Conclusion:

This work focuses on the subtle interaction between dataset size, model design, and embedding choices in sentiment analysis. While pretrained word embeddings are a helpful starting point, embedding layers perform better with larger datasets. The ideal decision, however, is determined by the individual task requirements and available computer resources. Further research could look into hybrid techniques or alternative architectures designed for sentiment analysis with limited data. Overall, this paper emphasizes the importance of empirical validation and ongoing experimentation in improving sentiment analysis models.