Importing Packages

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,
Dropout
```

Task params

```
max_features = 7000
maxlen = 450
batch_size = 32
embedding_dims = 50
filters = 250
kernel_size = 3
hidden_dims = 250
epochs = 3
```

Data Loading

```
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)
```

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

```
print(f"Training data shape: {x_train.shape}, {y_train.shape}")
print(f"Testing data shape: {x_test.shape}, {y_test.shape}")
```

```
Training data shape: (25000,), (25000,)
Testing data shape: (25000,), (25000,)
```

Pre-Processing

```
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

Model Building

```
model = Sequential()
model.add(Embedding(max_features, embedding_dims,
input_length=maxlen))
model.add(Dropout(0.2))
model.add(Conv1D(filters, kernel_size, padding='valid',
activation='relu', strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(hidden_dims, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 450, 50)           350000    
                                                                 
 dropout (Dropout)           (None, 450, 50)           0         
                                                                 
 conv1d (Conv1D)             (None, 448, 250)          37750     
                                                                 
 global_max_pooling1d (Glob  (None, 250)               0         
 alMaxPooling1D)                                                 
                                                                 
 dense (Dense)               (None, 250)               62750     
                                                                 
 dropout_1 (Dropout)         (None, 250)               0         
                                                                 
 dense_1 (Dense)             (None, 1)                 251       
                                                                 
=================================================================
Total params: 450751 (1.72 MB)
Trainable params: 450751 (1.72 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Training the Model

```
history = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_data=(x_test, y_test))
```

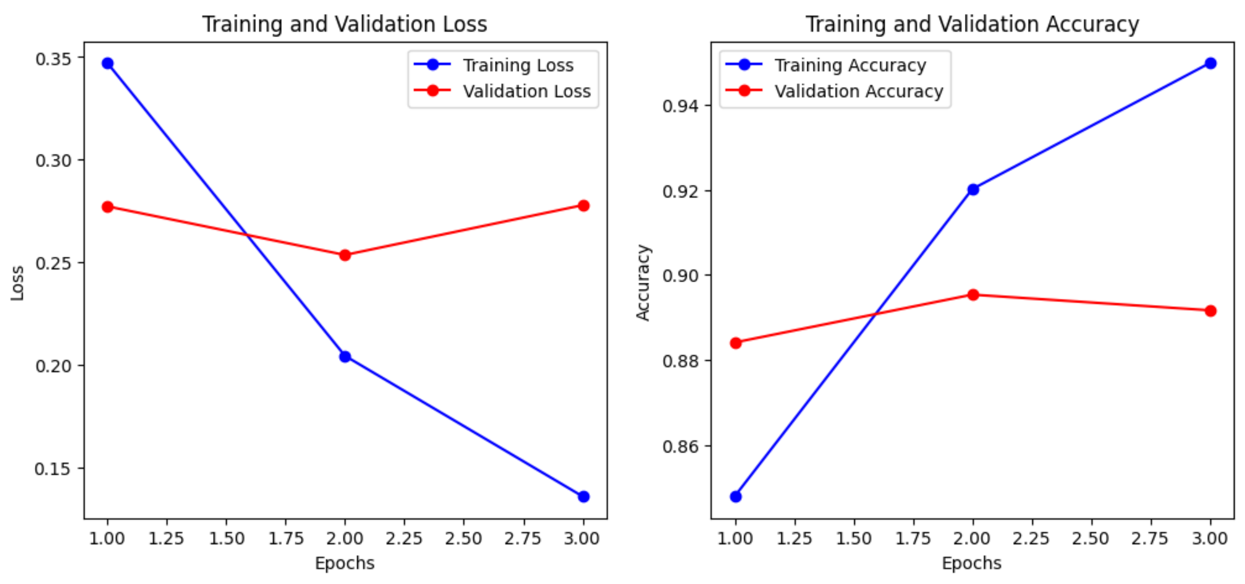
```
Epoch 1/3
782/782 [==============================] - 61s 78ms/step - loss:
0.3471 - accuracy: 0.8479 - val_loss: 0.2773 - val_accuracy: 0.8841
Epoch 2/3
782/782 [==============================] - 61s 79ms/step - loss:
0.2046 - accuracy: 0.9201 - val_loss: 0.2536 - val_accuracy: 0.8953
Epoch 3/3
782/782 [==============================] - 60s 77ms/step - loss:
0.1361 - accuracy: 0.9498 - val_loss: 0.2779 - val_accuracy: 0.8916

training_loss = history.history['loss']
validation_loss = history.history['val_loss']
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
epochs = range(1, len(training_loss) + 1)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, training_loss, 'bo-', label='Training Loss')
plt.plot(epochs, validation_loss, 'ro-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()


plt.subplot(1, 2, 2)
plt.plot(epochs, training_accuracy, 'bo-', label='Training Accuracy')
plt.plot(epochs, validation_accuracy, 'ro-', label='Validation
Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Inference

```
text = "This movie was amazing The acting was superb and the plot kept
me on the edge of my seat"

seq = imdb.get_word_index()
words = text.lower().split()
sequence = [seq[word] for word in words]
sequence = pad_sequences([sequence], maxlen=maxlen)

sequence
```

```
array([[    0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,
           0,    0,   11,   17,   13,  477,    1,  113,   13,  894,
2,
           1,  111,  825,   69,   20,    1, 1286,    4,   58, 2221]],
      dtype=int32)
```

```
prediction = model.predict(sequence)
```

```
1/1 [==============================] - 0s 87ms/step
```

```
prediction
```

```
array([[0.95971715]], dtype=float32)
```

```
print("Predicted sentiment: ", "Positive" if prediction > 0.5 else
"Negative")
```

```
Predicted sentiment:  Positive
```