In [1]:
```
import os
import numpy as np

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds

print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.config.list_physical_
```

```
Version:  2.15.0
Eager mode:  True
Hub version:  0.16.1
GPU is available
```

In [2]:
```
train_data, validation_data, test_data = tfds.load(
    name="imdb_reviews",
    split=('train[:60%]', 'train[60%:]', 'test'),
    as_supervised=True)
```

In [3]:
```
embedding = "https://tfhub.dev/google/nnlm-en-dim50/2"
hub_layer = hub.KerasLayer(embedding, input_shape=[],
                           dtype=tf.string, trainable=T
```

In [4]:
```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad
from tensorflow.keras.utils import to_categorical
```

In [5]:
```
num_words = 10000
max_len = 250
(x_train, y_train), (x_test, y_test) = imdb.load_data(n

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

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [6]:
```
validation_split = 0.2
indices = np.arange(x_train.shape[0])
np.random.shuffle(indices)
x_train = x_train[indices]
y_train = y_train[indices]
num_validation_samples = int(validation_split * x_train
x_val = x_train[:num_validation_samples]
y_val = y_train[:num_validation_samples]
x_train = x_train[num_validation_samples:]
y_train = y_train[num_validation_samples:]
```

In [7]:
```
from tensorflow.keras.layers import Layer, InputSpec
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, F
from tensorflow.keras.layers import Conv1D, GlobalMaxPo
```

In [8]:
```
class Global_k_MaxPooling1D(Layer):
    def __init__(self, k=1, **kwargs):
        super().__init__(**kwargs)
        self.input_spec = InputSpec(ndim=3)
        self.k = k
    def compute_output_shape(self, input_shape):
        return (input_shape[0], (input_shape[1] * self.
    def call(self, inputs):
        inputs = tf.transpose(inputs, [0, 2, 1])
        top_k = tf.nn.top_k(inputs, k=self.k, sorted=Tr
        top_k = tf.transpose(top_k, [0, 2, 1])
        return Flatten()(top_k)
```

In [9]:
```
def CNN():
    model = Sequential()
    model.add(Embedding(num_words, embedding_dim, input
    model.add(Conv1D(num_filters, kernel_size, padding=
    model.add(GlobalMaxPooling1D())
    model.add(Dense(500, activation = 'relu'))
    model.add(Dense(2, activation = 'sigmoid'))
    model.summary()
    return model
```

In [10]:
```
def KCNN(k):
    model = Sequential()
    model.add(Embedding(num_words, embedding_dim, input
    model.add(Conv1D(num_filters, kernel_size, padding=
    model.add(Global_k_MaxPooling1D(k))
    model.add(Dense(500, activation = 'relu'))
    model.add(Dense(2, activation = 'softmax'))
    model.summary()
    return model
```

In [11]:
```
def LossPlot():
    fig, loss_ax = plt.subplots()
    acc_ax = loss_ax.twinx()
    loss_ax.plot(hist.history['loss'], 'c', linestyle =
    loss_ax.plot(hist.history['val_loss'], 'c', linesty
    loss_ax.set_ylim([0.0, 3.0])
    acc_ax.plot(hist.history['accuracy'], 'k', linestyl
    acc_ax.plot(hist.history['val_accuracy'], 'k', line
    acc_ax.set_ylim([0.0, 1.0])
    loss_ax.set_xlabel('Epoch', fontsize = 12)
    loss_ax.set_ylabel('Loss', fontsize = 12)
    acc_ax.set_ylabel('Accuracy', fontsize = 12)
    loss_ax.legend(loc='lower left')
    acc_ax.legend(loc='upper left')
    plt.show()
```

In [12]:
```
num_words     = 10000
max_len       = 250

embedding_dim = 64
num_filters   = 128
kernel_size   = 3
```

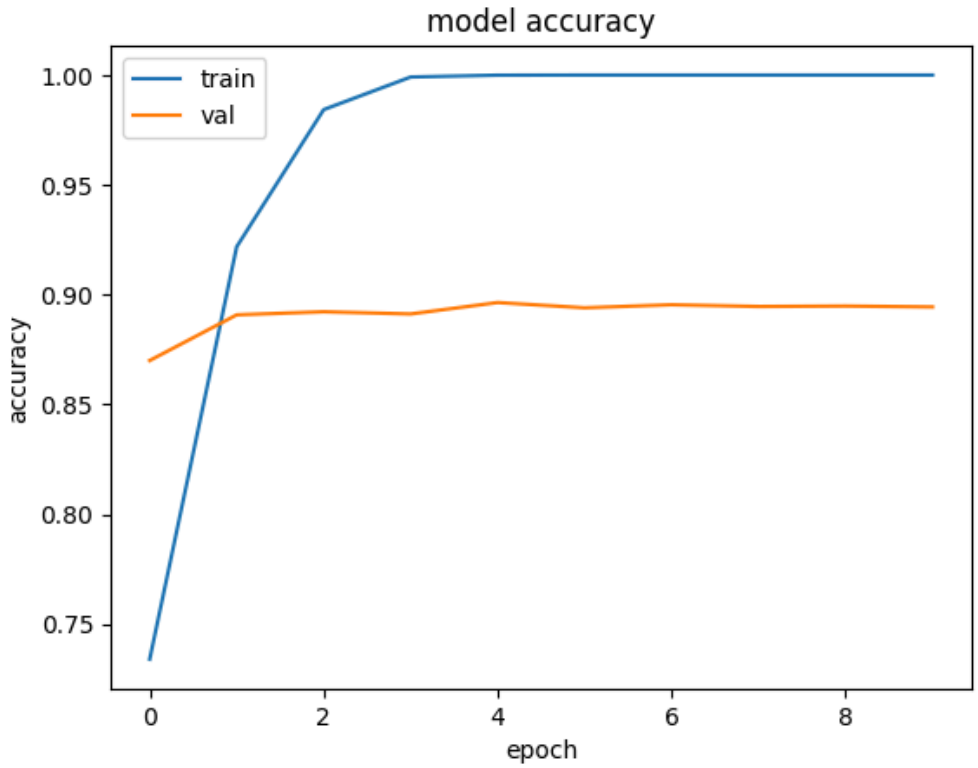
In [13]:
```
model = CNN()
model.compile(loss = 'categorical_crossentropy', optimi
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 250, 64)           640000

 conv1d (Conv1D)             (None, 250, 128)          24704

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense (Dense)               (None, 500)               64500

 dense_1 (Dense)             (None, 2)                 1002

=================================================================
Total params: 730206 (2.79 MB)
Trainable params: 730206 (2.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [14]:
```
hist = model.fit(x_train, y_train, epochs=10, batch_siz
```

```
Epoch 1/10
157/157 [==============================] - 31s 156ms/s
tep - loss: 0.4918 - accuracy: 0.7340 - val_loss: 0.30
61 - val_accuracy: 0.8700
Epoch 2/10
157/157 [==============================] - 15s 95ms/st
ep - loss: 0.2019 - accuracy: 0.9218 - val_loss: 0.275
0 - val_accuracy: 0.8908
Epoch 3/10
157/157 [==============================] - 11s 73ms/st
ep - loss: 0.0618 - accuracy: 0.9843 - val_loss: 0.313
6 - val_accuracy: 0.8922
Epoch 4/10
157/157 [==============================] - 8s 52ms/ste
p - loss: 0.0112 - accuracy: 0.9991 - val_loss: 0.3663
- val_accuracy: 0.8912
Epoch 5/10
157/157 [==============================] - 6s 36ms/ste
p - loss: 0.0023 - accuracy: 0.9999 - val_loss: 0.3911
- val_accuracy: 0.8964
Epoch 6/10
157/157 [==============================] - 6s 39ms/ste
p - loss: 8.5967e-04 - accuracy: 1.0000 - val_loss: 0.
4122 - val_accuracy: 0.8940
Epoch 7/10
157/157 [==============================] - 7s 42ms/ste
p - loss: 5.3536e-04 - accuracy: 1.0000 - val_loss: 0.
4279 - val_accuracy: 0.8954
Epoch 8/10
157/157 [==============================] - 4s 28ms/ste
p - loss: 3.6806e-04 - accuracy: 1.0000 - val_loss: 0.
4423 - val_accuracy: 0.8946
Epoch 9/10
157/157 [==============================] - 3s 20ms/ste
p - loss: 2.6562e-04 - accuracy: 1.0000 - val_loss: 0.
4550 - val_accuracy: 0.8948
Epoch 10/10
157/157 [==============================] - 2s 13ms/ste
p - loss: 1.9507e-04 - accuracy: 1.0000 - val_loss: 0.
4683 - val_accuracy: 0.8944
```

In [15]:
```
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [16]:
```
model = KCNN(3)
model.compile(loss = 'categorical_crossentropy', optimi
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 250, 64)           640000

 conv1d_1 (Conv1D)           (None, 250, 128)          24704

 global_k__max_pooling1d (G  (None, 384)               0
 lobal_k_MaxPooling1D)

 dense_2 (Dense)             (None, 500)               192500

 dense_3 (Dense)             (None, 2)                 1002

=================================================================
Total params: 858206 (3.27 MB)
Trainable params: 858206 (3.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [17]:
```
hist = model.fit(x_train, y_train, epochs=10, batch_siz
```

```
Epoch 1/10
157/157 [==============================] - 22s 133ms/s
tep - loss: 0.4601 - accuracy: 0.7621 - val_loss: 0.27
75 - val_accuracy: 0.8814
Epoch 2/10
157/157 [==============================] - 13s 84ms/st
ep - loss: 0.1823 - accuracy: 0.9304 - val_loss: 0.291
7 - val_accuracy: 0.8828
Epoch 3/10
157/157 [==============================] - 14s 88ms/st
ep - loss: 0.0639 - accuracy: 0.9813 - val_loss: 0.345
6 - val_accuracy: 0.8818
Epoch 4/10
157/157 [==============================] - 9s 60ms/ste
p - loss: 0.0137 - accuracy: 0.9980 - val_loss: 0.3815
- val_accuracy: 0.8950
Epoch 5/10
157/157 [==============================] - 8s 50ms/ste
p - loss: 0.0026 - accuracy: 0.9998 - val_loss: 0.4252
- val_accuracy: 0.8916
Epoch 6/10
157/157 [==============================] - 8s 52ms/ste
p - loss: 7.9865e-04 - accuracy: 1.0000 - val_loss: 0.
4477 - val_accuracy: 0.8944
Epoch 7/10
157/157 [==============================] - 8s 49ms/ste
p - loss: 4.5807e-04 - accuracy: 1.0000 - val_loss: 0.
4680 - val_accuracy: 0.8942
Epoch 8/10
157/157 [==============================] - 7s 45ms/ste
p - loss: 3.1481e-04 - accuracy: 1.0000 - val_loss: 0.
4840 - val_accuracy: 0.8934
Epoch 9/10
157/157 [==============================] - 8s 50ms/ste
p - loss: 2.3161e-04 - accuracy: 1.0000 - val_loss: 0.
5000 - val_accuracy: 0.8932
Epoch 10/10
157/157 [==============================] - 7s 43ms/ste
p - loss: 1.3246e-04 - accuracy: 1.0000 - val_loss: 0.
5501 - val_accuracy: 0.8946
```

In [18]:

```
from matplotlib import pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```