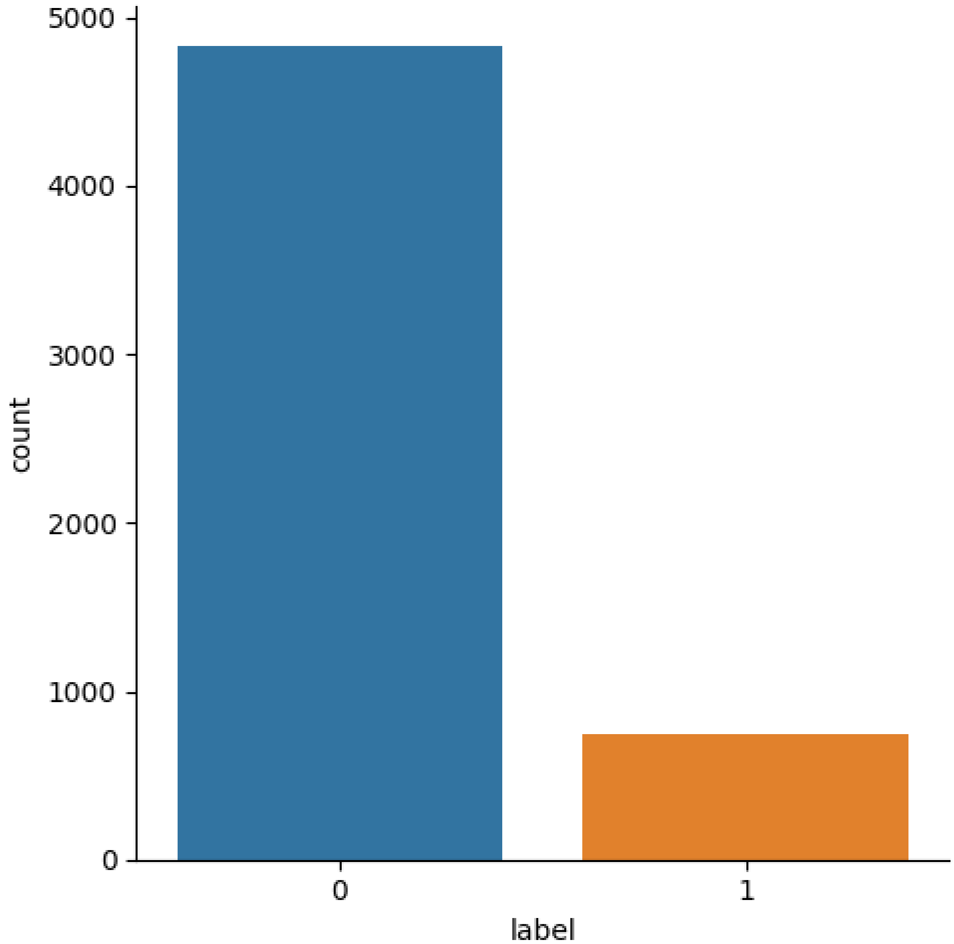
Assignment 9: Text Classification with Keras

In [ ]:
```python
# some necessary packages
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np
import pandas as pd
import seaborn as sb

# set seed for reproducibility
np.random.seed(1234)
```

In [ ]:
```python
df = pd.read_csv('train.csv', header=0, encoding='latin-1')
print('rows and columns:', df.shape)
print(df.head())

# Create a graph showing the distribution of the target classes using seaborn
sb.catplot(x="label", kind='count', data=df)
```

```
rows and columns: (5574, 2)
                                                 sms  label
0  Go until jurong point, crazy.. Available only ...      0
1                      Ok lar... Joking wif u oni...\n      0
2  Free entry in 2 a wkly comp to win FA Cup fina...      1
3  U dun say so early hor... U c already then say...      0
4  Nah I don't think he goes to usf, he lives aro...      0
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f8ef3086c70>
```



This data set holds SMS labeled messages that have been collected for mobile phone spam research. Data labeled as '0' is Not Spam and data labeled as '1' is Spam.

The model should be able to predict whether a piece of text is Not Spam or is Spam.

In [ ]:
```
# split df into train and test
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (4465, 2)
test data size:  (1109, 2)
```

In [ ]:
```
# set up X and Y
num_labels = 2
vocab_size = 25000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.sms)

x_train = tokenizer.texts_to_matrix(train.sms, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.sms, mode='tfidf')
```

```
encoder = LabelEncoder()
encoder.fit(train.label)
y_train = encoder.transform(train.label)
y_test = encoder.transform(test.label)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (4465, 25000) (4465,)
test shapes: (1109, 25000) (1109,)
test first five labels: [0 1 1 1 0]
```

Using a Sequential Model

In [ ]:

```
# fit model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activ
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

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

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 32)                800032

 dense_14 (Dense)            (None, 1)                 33

=================================================================
Total params: 800,065
Trainable params: 800,065
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

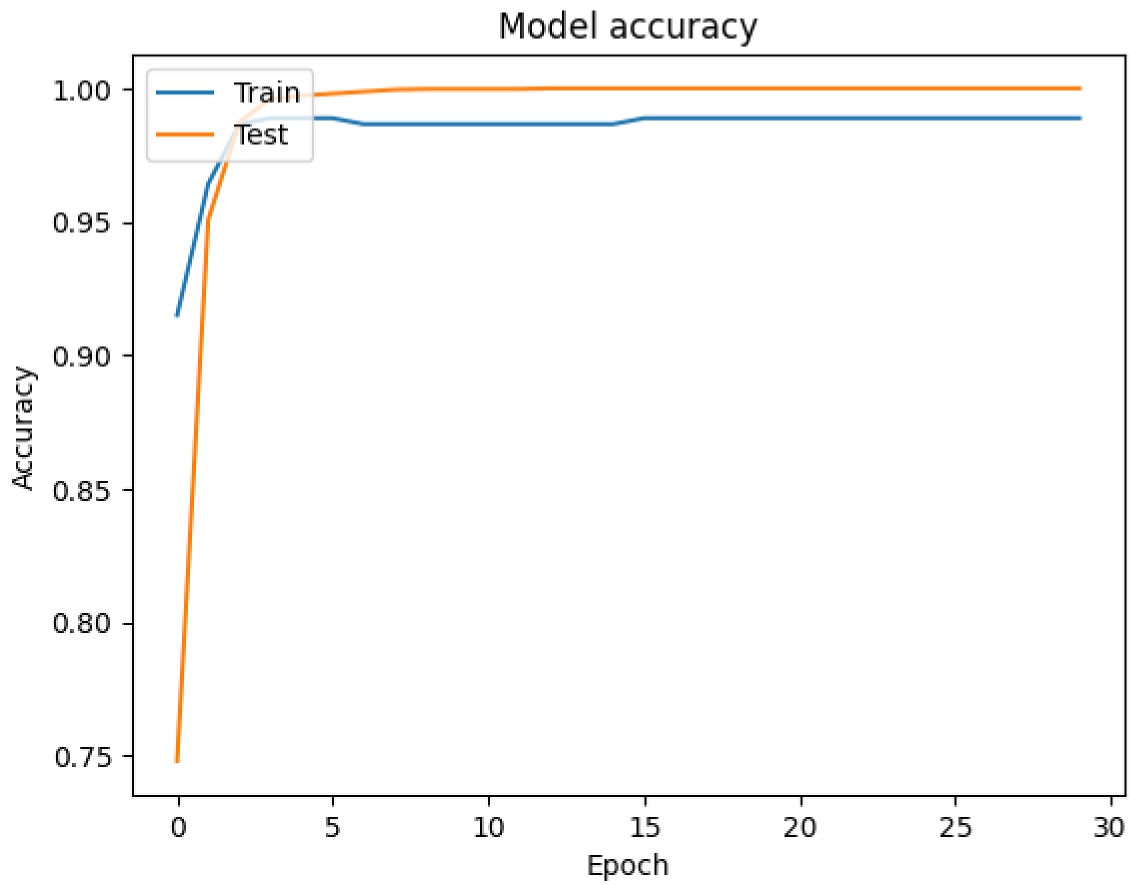
```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/30
41/41 [==============================] - 2s 29ms/step - loss: 0.5891 - accuracy:
0.7481 - val_loss: 0.4153 - val_accuracy: 0.9150
Epoch 2/30
41/41 [==============================] - 1s 24ms/step - loss: 0.2821 - accuracy:
0.9507 - val_loss: 0.2022 - val_accuracy: 0.9642
Epoch 3/30
41/41 [==============================] - 1s 26ms/step - loss: 0.1221 - accuracy:
0.9873 - val_loss: 0.1157 - val_accuracy: 0.9866
Epoch 4/30
41/41 [==============================] - 1s 35ms/step - loss: 0.0610 - accuracy:
0.9960 - val_loss: 0.0868 - val_accuracy: 0.9888
Epoch 5/30
41/41 [==============================] - 1s 33ms/step - loss: 0.0360 - accuracy:
0.9973 - val_loss: 0.0752 - val_accuracy: 0.9888
Epoch 6/30
41/41 [==============================] - 1s 24ms/step - loss: 0.0238 - accuracy:
0.9980 - val_loss: 0.0703 - val_accuracy: 0.9888
Epoch 7/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0167 - accuracy:
0.9988 - val_loss: 0.0676 - val_accuracy: 0.9866
Epoch 8/30
41/41 [==============================] - 1s 24ms/step - loss: 0.0121 - accuracy:
0.9995 - val_loss: 0.0667 - val_accuracy: 0.9866
Epoch 9/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0094 - accuracy:
0.9998 - val_loss: 0.0666 - val_accuracy: 0.9866
Epoch 10/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0074 - accuracy:
0.9998 - val_loss: 0.0661 - val_accuracy: 0.9866
Epoch 11/30
41/41 [==============================] - 1s 22ms/step - loss: 0.0060 - accuracy:
0.9998 - val_loss: 0.0666 - val_accuracy: 0.9866
Epoch 12/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0049 - accuracy:
0.9998 - val_loss: 0.0673 - val_accuracy: 0.9866
Epoch 13/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0041 - accuracy:
1.0000 - val_loss: 0.0680 - val_accuracy: 0.9866
Epoch 14/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0035 - accuracy:
1.0000 - val_loss: 0.0687 - val_accuracy: 0.9866
Epoch 15/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0030 - accuracy:
1.0000 - val_loss: 0.0699 - val_accuracy: 0.9866
Epoch 16/30
41/41 [==============================] - 1s 30ms/step - loss: 0.0026 - accuracy:
1.0000 - val_loss: 0.0704 - val_accuracy: 0.9888
Epoch 17/30
41/41 [==============================] - 1s 36ms/step - loss: 0.0023 - accuracy:
1.0000 - val_loss: 0.0713 - val_accuracy: 0.9888
Epoch 18/30
41/41 [==============================] - 1s 29ms/step - loss: 0.0020 - accuracy:
1.0000 - val_loss: 0.0720 - val_accuracy: 0.9888
Epoch 19/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0018 - accuracy:
```

```
1.0000 - val_loss: 0.0730 - val_accuracy: 0.9888
Epoch 20/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 0.0737 - val_accuracy: 0.9888
Epoch 21/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 0.0745 - val_accuracy: 0.9888
Epoch 22/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 0.0754 - val_accuracy: 0.9888
Epoch 23/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 0.0759 - val_accuracy: 0.9888
Epoch 24/30
41/41 [==============================] - 1s 24ms/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 0.0767 - val_accuracy: 0.9888
Epoch 25/30
41/41 [==============================] - 1s 23ms/step - loss: 0.0010 - accuracy:
1.0000 - val_loss: 0.0775 - val_accuracy: 0.9888
Epoch 26/30
41/41 [==============================] - 1s 23ms/step - loss: 9.5575e-04 - accurac
y: 1.0000 - val_loss: 0.0784 - val_accuracy: 0.9888
Epoch 27/30
41/41 [==============================] - 1s 23ms/step - loss: 8.8575e-04 - accurac
y: 1.0000 - val_loss: 0.0790 - val_accuracy: 0.9888
Epoch 28/30
41/41 [==============================] - 1s 24ms/step - loss: 8.2408e-04 - accurac
y: 1.0000 - val_loss: 0.0798 - val_accuracy: 0.9888
Epoch 29/30
41/41 [==============================] - 1s 34ms/step - loss: 7.6887e-04 - accurac
y: 1.0000 - val_loss: 0.0805 - val_accuracy: 0.9888
Epoch 30/30
41/41 [==============================] - 1s 35ms/step - loss: 7.1874e-04 - accurac
y: 1.0000 - val_loss: 0.0812 - val_accuracy: 0.9888
```

In [ ]:

```python
# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:
```
# Evaluate on test data
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
print('Loss: ', score[0])
```

```
12/12 [==============================] - 0s 8ms/step - loss: 0.1104 - accuracy: 0.
9838
Accuracy:  0.9837691783905029
Loss:  0.11036085337400436
```

In [ ]:
```
# Get predictions so we can calculate recall, f1, etc.
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]

print(pred[:10])
print(pred_labels[:10])

print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
35/35 [==============================] - 0s 4ms/step
[[2.7375409e-06]
 [9.9996001e-01]
 [9.9993151e-01]
 [3.3593246e-01]
 [6.7555270e-06]
 [6.9552794e-11]
 [7.2406023e-05]
 [2.3160602e-03]
 [2.7215751e-12]
 [3.8534995e-06]]
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
accuracy score:  0.9837691614066727
precision score:  1.0
recall score:  0.8860759493670886
f1 score:  0.9395973154362416
```

Using a CNN Model

In [ ]:

```
embedding_dim = 100

model = models.Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, None, 100)         2500000

 conv1d (Conv1D)             (None, None, 128)         64128

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense_15 (Dense)            (None, 10)                1290

 dense_16 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,565,429
Trainable params: 2,565,429
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

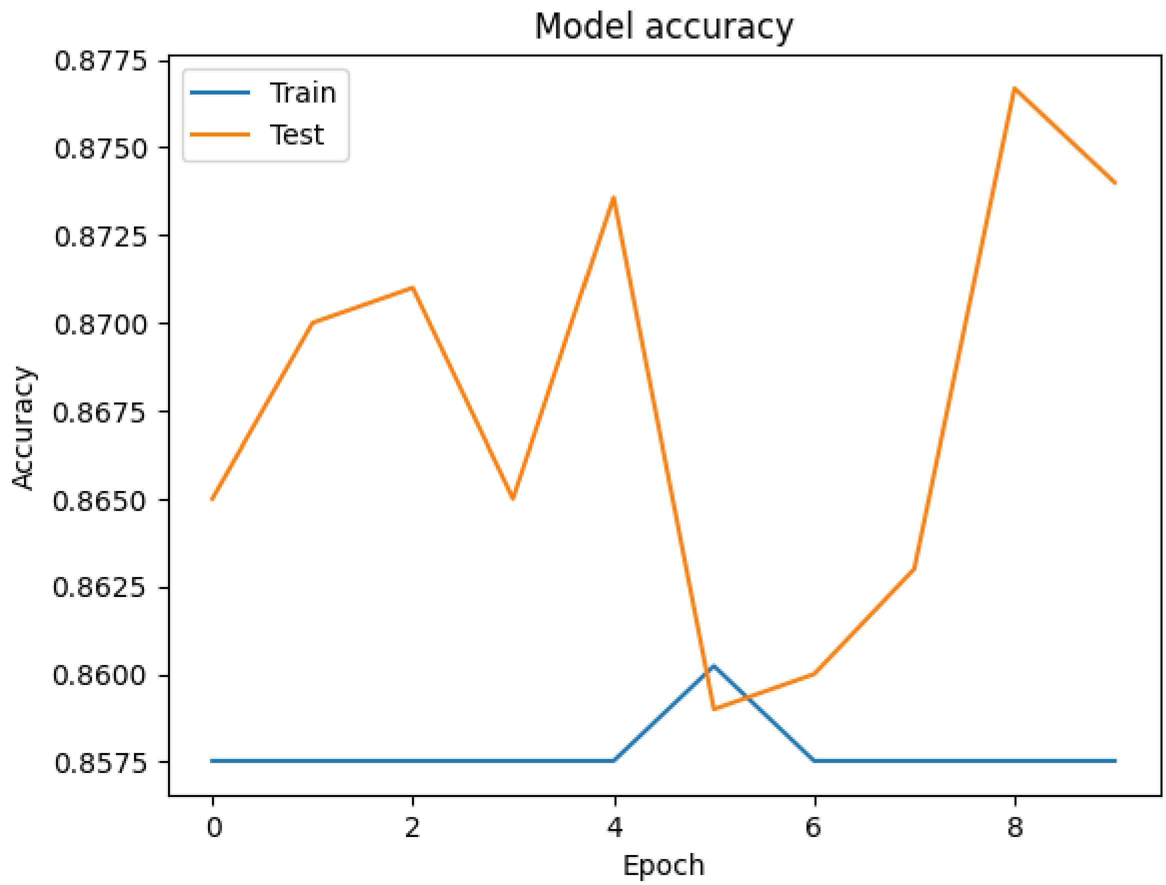
```
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
```

```
                    epochs=10,
                    steps_per_epoch=10,
                    verbose=1,
                    validation_split=0.1,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/10
10/10 [==============================] - 350s 35s/step - loss: 0.5523 - accuracy:
0.8650 - val_loss: 0.4416 - val_accuracy: 0.8575
Epoch 2/10
10/10 [==============================] - 334s 34s/step - loss: 0.4032 - accuracy:
0.8700 - val_loss: 0.4316 - val_accuracy: 0.8575
Epoch 3/10
10/10 [==============================] - 347s 35s/step - loss: 0.3909 - accuracy:
0.8710 - val_loss: 0.4118 - val_accuracy: 0.8575
Epoch 4/10
10/10 [==============================] - 317s 33s/step - loss: 0.3734 - accuracy:
0.8650 - val_loss: 0.3895 - val_accuracy: 0.8575
Epoch 5/10
10/10 [==============================] - 314s 32s/step - loss: 0.3418 - accuracy:
0.8736 - val_loss: 0.3859 - val_accuracy: 0.8575
Epoch 6/10
10/10 [==============================] - 325s 33s/step - loss: 0.3387 - accuracy:
0.8590 - val_loss: 0.3728 - val_accuracy: 0.8602
Epoch 7/10
10/10 [==============================] - 329s 34s/step - loss: 0.3271 - accuracy:
0.8600 - val_loss: 0.3678 - val_accuracy: 0.8575
Epoch 8/10
10/10 [==============================] - 328s 34s/step - loss: 0.3280 - accuracy:
0.8630 - val_loss: 0.3709 - val_accuracy: 0.8575
Epoch 9/10
10/10 [==============================] - 318s 33s/step - loss: 0.3120 - accuracy:
0.8767 - val_loss: 0.3679 - val_accuracy: 0.8575
Epoch 10/10
10/10 [==============================] - 323s 33s/step - loss: 0.3168 - accuracy:
0.8740 - val_loss: 0.3687 - val_accuracy: 0.8575
```

In [ ]:

```
# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:
```
# Evaluate on test data
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
print('Loss: ', score[0])
```

```
12/12 [==============================] - 73s 6s/step - loss: 0.3687 - accuracy: 0.
8575
Accuracy:  0.8575292825698853
Loss:  0.36866673827171326
```

Save the model for future use

In [ ]:
```
model.save('functional_model')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _updat
e_step_xla while saving (showing 2 of 2). These functions will not be directly cal
lable after loading.
```

Different Embedding Approaches

Here, I am modifying the normal Sequential model to have a middle dense layer with 64 nodes instead of the normal 32 and increasing the epoch number to 50 from 30. to see if the accuracy increases or if it overfits the training data.

In [ ]:
```
# fit model
embedding_dim = 50
```

```
model = models.Sequential()
model.add(layers.Dense(64, input_dim=vocab_size, kernel_initializer='normal', activ
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_20 (Dense)            (None, 64)                1600064   
                                                                 
 dense_21 (Dense)            (None, 1)                 65        
                                                                 
=================================================================
Total params: 1,600,129
Trainable params: 1,600,129
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
history = model.fit(x_train, y_train,
                    epochs=50,
                    verbose=1,
                    validation_data=(x_test, y_test),
                    batch_size=batch_size)
```

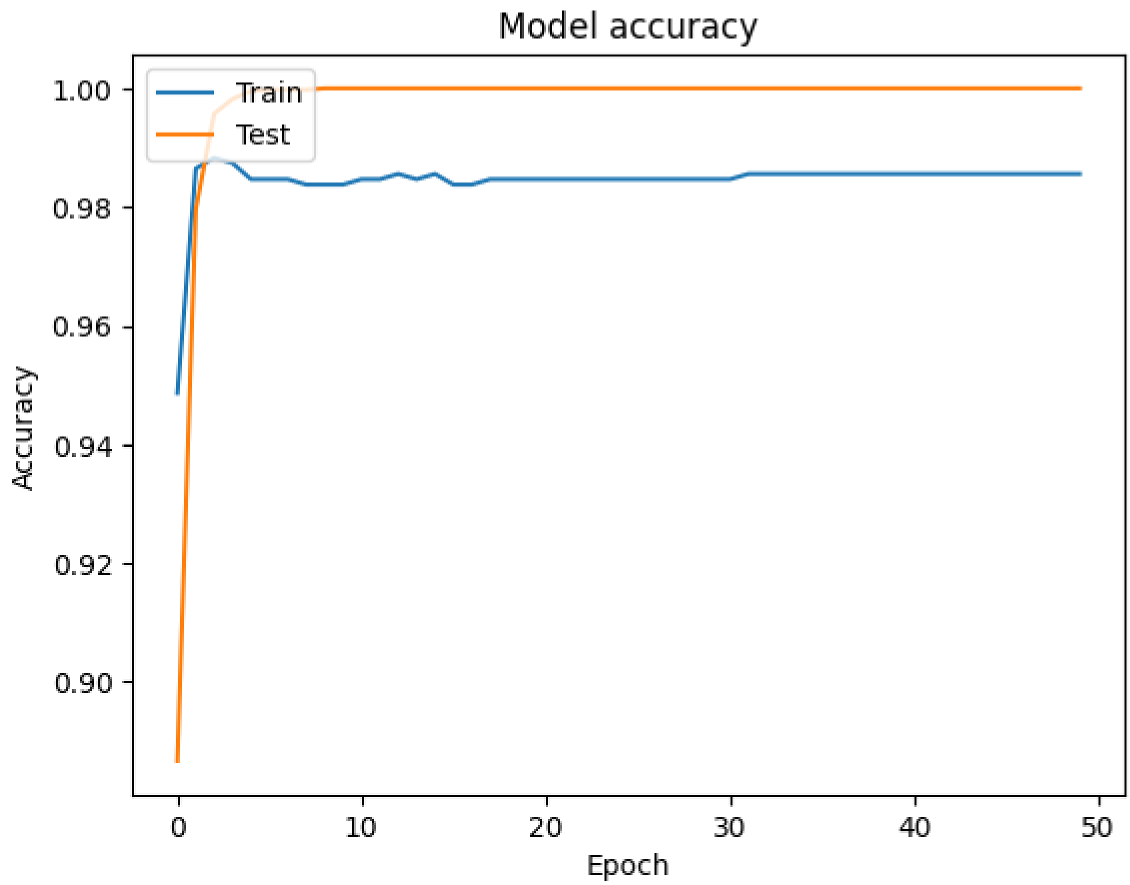
```
Epoch 1/50
45/45 [==============================] - 6s 114ms/step - loss: 0.4096 - accuracy:
0.8867 - val_loss: 0.2232 - val_accuracy: 0.9486
Epoch 2/50
45/45 [==============================] - 4s 86ms/step - loss: 0.1209 - accuracy:
0.9796 - val_loss: 0.1009 - val_accuracy: 0.9865
Epoch 3/50
45/45 [==============================] - 2s 52ms/step - loss: 0.0432 - accuracy:
0.9957 - val_loss: 0.0794 - val_accuracy: 0.9883
Epoch 4/50
45/45 [==============================] - 2s 45ms/step - loss: 0.0206 - accuracy:
0.9982 - val_loss: 0.0761 - val_accuracy: 0.9874
Epoch 5/50
45/45 [==============================] - 2s 49ms/step - loss: 0.0118 - accuracy:
0.9996 - val_loss: 0.0762 - val_accuracy: 0.9847
Epoch 6/50
45/45 [==============================] - 3s 67ms/step - loss: 0.0078 - accuracy:
0.9998 - val_loss: 0.0777 - val_accuracy: 0.9847
Epoch 7/50
45/45 [==============================] - 2s 44ms/step - loss: 0.0054 - accuracy:
0.9998 - val_loss: 0.0795 - val_accuracy: 0.9847
Epoch 8/50
45/45 [==============================] - 2s 41ms/step - loss: 0.0040 - accuracy:
0.9998 - val_loss: 0.0814 - val_accuracy: 0.9838
Epoch 9/50
45/45 [==============================] - 2s 41ms/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 0.0836 - val_accuracy: 0.9838
Epoch 10/50
45/45 [==============================] - 2s 42ms/step - loss: 0.0025 - accuracy:
1.0000 - val_loss: 0.0851 - val_accuracy: 0.9838
Epoch 11/50
45/45 [==============================] - 2s 45ms/step - loss: 0.0020 - accuracy:
1.0000 - val_loss: 0.0873 - val_accuracy: 0.9847
Epoch 12/50
45/45 [==============================] - 3s 71ms/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 0.0889 - val_accuracy: 0.9847
Epoch 13/50
45/45 [==============================] - 2s 43ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 0.0907 - val_accuracy: 0.9856
Epoch 14/50
45/45 [==============================] - 2s 46ms/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 0.0921 - val_accuracy: 0.9847
Epoch 15/50
45/45 [==============================] - 2s 42ms/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 0.0939 - val_accuracy: 0.9856
Epoch 16/50
45/45 [==============================] - 2s 41ms/step - loss: 9.5315e-04 - accurac
y: 1.0000 - val_loss: 0.0953 - val_accuracy: 0.9838
Epoch 17/50
45/45 [==============================] - 2s 45ms/step - loss: 8.4657e-04 - accurac
y: 1.0000 - val_loss: 0.0966 - val_accuracy: 0.9838
Epoch 18/50
45/45 [==============================] - 3s 70ms/step - loss: 7.5776e-04 - accurac
y: 1.0000 - val_loss: 0.0980 - val_accuracy: 0.9847
Epoch 19/50
45/45 [==============================] - 2s 43ms/step - loss: 6.8219e-04 - accurac
```

```
y: 1.0000 - val_loss: 0.0990 - val_accuracy: 0.9847
Epoch 20/50
45/45 [==============================] - 2s 45ms/step - loss: 6.1730e-04 - accurac
y: 1.0000 - val_loss: 0.1001 - val_accuracy: 0.9847
Epoch 21/50
45/45 [==============================] - 2s 41ms/step - loss: 5.6313e-04 - accurac
y: 1.0000 - val_loss: 0.1015 - val_accuracy: 0.9847
Epoch 22/50
45/45 [==============================] - 2s 42ms/step - loss: 5.1530e-04 - accurac
y: 1.0000 - val_loss: 0.1027 - val_accuracy: 0.9847
Epoch 23/50
45/45 [==============================] - 2s 41ms/step - loss: 4.7376e-04 - accurac
y: 1.0000 - val_loss: 0.1036 - val_accuracy: 0.9847
Epoch 24/50
45/45 [==============================] - 3s 67ms/step - loss: 4.3739e-04 - accurac
y: 1.0000 - val_loss: 0.1047 - val_accuracy: 0.9847
Epoch 25/50
45/45 [==============================] - 2s 45ms/step - loss: 4.0521e-04 - accurac
y: 1.0000 - val_loss: 0.1058 - val_accuracy: 0.9847
Epoch 26/50
45/45 [==============================] - 2s 41ms/step - loss: 3.7591e-04 - accurac
y: 1.0000 - val_loss: 0.1066 - val_accuracy: 0.9847
Epoch 27/50
45/45 [==============================] - 2s 44ms/step - loss: 3.5080e-04 - accurac
y: 1.0000 - val_loss: 0.1075 - val_accuracy: 0.9847
Epoch 28/50
45/45 [==============================] - 2s 44ms/step - loss: 3.2755e-04 - accurac
y: 1.0000 - val_loss: 0.1084 - val_accuracy: 0.9847
Epoch 29/50
45/45 [==============================] - 2s 40ms/step - loss: 3.0703e-04 - accurac
y: 1.0000 - val_loss: 0.1095 - val_accuracy: 0.9847
Epoch 30/50
45/45 [==============================] - 3s 59ms/step - loss: 2.8851e-04 - accurac
y: 1.0000 - val_loss: 0.1103 - val_accuracy: 0.9847
Epoch 31/50
45/45 [==============================] - 2s 53ms/step - loss: 2.7153e-04 - accurac
y: 1.0000 - val_loss: 0.1111 - val_accuracy: 0.9847
Epoch 32/50
45/45 [==============================] - 2s 44ms/step - loss: 2.5613e-04 - accurac
y: 1.0000 - val_loss: 0.1121 - val_accuracy: 0.9856
Epoch 33/50
45/45 [==============================] - 2s 41ms/step - loss: 2.4197e-04 - accurac
y: 1.0000 - val_loss: 0.1127 - val_accuracy: 0.9856
Epoch 34/50
45/45 [==============================] - 2s 44ms/step - loss: 2.2901e-04 - accurac
y: 1.0000 - val_loss: 0.1135 - val_accuracy: 0.9856
Epoch 35/50
45/45 [==============================] - 2s 41ms/step - loss: 2.1724e-04 - accurac
y: 1.0000 - val_loss: 0.1142 - val_accuracy: 0.9856
Epoch 36/50
45/45 [==============================] - 2s 50ms/step - loss: 2.0618e-04 - accurac
y: 1.0000 - val_loss: 0.1150 - val_accuracy: 0.9856
Epoch 37/50
45/45 [==============================] - 3s 64ms/step - loss: 1.9605e-04 - accurac
y: 1.0000 - val_loss: 0.1157 - val_accuracy: 0.9856
Epoch 38/50
```

```
45/45 [==============================] - 2s 41ms/step - loss: 1.8680e-04 - accurac
y: 1.0000 - val_loss: 0.1165 - val_accuracy: 0.9856
Epoch 39/50
45/45 [==============================] - 2s 43ms/step - loss: 1.7809e-04 - accurac
y: 1.0000 - val_loss: 0.1173 - val_accuracy: 0.9856
Epoch 40/50
45/45 [==============================] - 2s 41ms/step - loss: 1.6985e-04 - accurac
y: 1.0000 - val_loss: 0.1179 - val_accuracy: 0.9856
Epoch 41/50
45/45 [==============================] - 2s 40ms/step - loss: 1.6229e-04 - accurac
y: 1.0000 - val_loss: 0.1185 - val_accuracy: 0.9856
Epoch 42/50
45/45 [==============================] - 2s 44ms/step - loss: 1.5515e-04 - accurac
y: 1.0000 - val_loss: 0.1193 - val_accuracy: 0.9856
Epoch 43/50
45/45 [==============================] - 3s 72ms/step - loss: 1.4861e-04 - accurac
y: 1.0000 - val_loss: 0.1199 - val_accuracy: 0.9856
Epoch 44/50
45/45 [==============================] - 2s 45ms/step - loss: 1.4251e-04 - accurac
y: 1.0000 - val_loss: 0.1206 - val_accuracy: 0.9856
Epoch 45/50
45/45 [==============================] - 2s 44ms/step - loss: 1.3673e-04 - accurac
y: 1.0000 - val_loss: 0.1211 - val_accuracy: 0.9856
Epoch 46/50
45/45 [==============================] - 2s 51ms/step - loss: 1.3130e-04 - accurac
y: 1.0000 - val_loss: 0.1218 - val_accuracy: 0.9856
Epoch 47/50
45/45 [==============================] - 2s 47ms/step - loss: 1.2624e-04 - accurac
y: 1.0000 - val_loss: 0.1225 - val_accuracy: 0.9856
Epoch 48/50
45/45 [==============================] - 2s 46ms/step - loss: 1.2132e-04 - accurac
y: 1.0000 - val_loss: 0.1231 - val_accuracy: 0.9856
Epoch 49/50
45/45 [==============================] - 3s 77ms/step - loss: 1.1680e-04 - accurac
y: 1.0000 - val_loss: 0.1237 - val_accuracy: 0.9856
Epoch 50/50
45/45 [==============================] - 2s 43ms/step - loss: 1.1254e-04 - accurac
y: 1.0000 - val_loss: 0.1243 - val_accuracy: 0.9856
```

In [ ]:

```
# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:

```
# Evaluate on test data
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
print('Loss: ', score[0])
```

```
12/12 [==============================] - 0s 16ms/step - loss: 0.1243 - accuracy:
0.9856
Accuracy:  0.9855725765228271
Loss:  0.12431249022483826
```

In [ ]:

```
# Get predictions so we can calculate recall, f1, etc.
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]

print(pred[:10])
print(pred_labels[:10])

print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
35/35 [==============================] - 0s 7ms/step
[[8.6301537e-07]
 [9.9998462e-01]
 [9.9999744e-01]
 [5.1109916e-01]
 [2.9832279e-08]
 [4.0621752e-14]
 [2.9252993e-07]
 [1.5760394e-05]
 [7.5378643e-16]
 [5.4969966e-09]]
[0, 1, 1, 1, 0, 0, 0, 0, 0, 0]
accuracy score:  0.9855725879170424
precision score:  1.0
recall score:  0.8987341772151899
f1 score:  0.9466666666666668
```

Performance Analysis

- The regular Sequential model outperformed the other models by a very large margin, both in speed and accuracy. In a total of 40 seconds, it produced a final testing accuracy of 98.38% with a loss of .1104. During training, the model had epochs where it performed with 100% accuracy.
- The CNN model performed the worst in both accuracy and time to train. The first CNN architecture I worked with produced an accuracy of 87% after 4 hours of training, and the final model produces an accuracy of 85.75% and loss of .3687 after about 55 minutes. With a 12% reduction in accuracy and a .26 increase in loss plus an increase of 54 minutes in training time, the CNN model definitely underperformed the regular Sequential model.
- The Sequential model with modified embeddings performed slightly better than the Sequential model, with an accuracy of 98.56% and a loss of .1243. The major downside to this model is that it took about 2 minutes to finish training, which is about 3 times longer than the original Sequential model. Overall, the modified embeddings model performed the best, with only a slight downside in training time.