Text Classification for this assignmnt, I used the link https://www.kaggle.com/datasets/mfaisalqureshi/spam-email . this data contains ham and spam emails. themodel should be able to predict if a given text is real or spam.

```
import pandas as pd
import io
from sklearn.model_selection import train_test_split
```

```
# Load the CSV file into a pandas dataframe
from google.colab import files
uploaded = files.upload()
```

```
Choose Files spam.csv
  • spam.csv(text/csv) - 480130 bytes, last modified: 4/1/2023 - 100% done
Saving spam.csv to spam.csv
```

```
df = pd.read_csv(io.BytesIO(uploaded['spam.csv']))
```

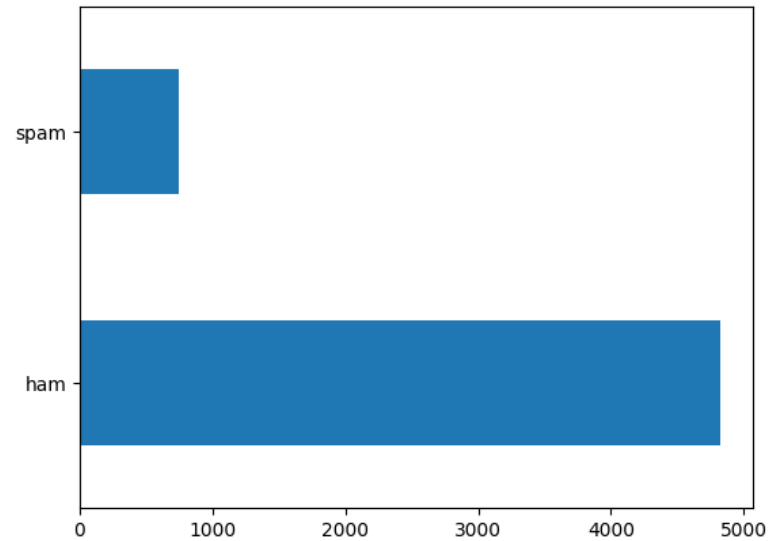
```
print(df)
```

```
     Category                                            Message
0         ham  Go until jurong point, crazy.. Available only ...
1         ham                      Ok lar... Joking wif u oni...
2        spam  Free entry in 2 a wkly comp to win FA Cup fina...
3         ham  U dun say so early hor... U c already then say...
4         ham  Nah I don't think he goes to usf, he lives aro...
...       ...                                                ...
5567     spam  This is the 2nd time we have tried 2 contact u...
5568      ham              Will ü b going to esplanade fr home?
5569      ham  Pity, * was in mood for that. So...any other s...
5570      ham  The guy did some bitching but I acted like i'd...
5571      ham                        Rofl. Its true to its name

[5572 rows x 2 columns]
```

```
#Graph target distribution
df['Category'].value_counts().plot(kind = 'barh')
```

```
<Axes: >
```




```
#divide into training and testing data
import numpy as np
np.random.seed(80085)
i = np.random.rand(len(df))<  0.8
train = df[i]
test = df[~i]
```

```
#Print the train shape and test shape
print('Train Shape: ', train.shape)
print('Test Shape: ', test.shape)
```

```
Train Shape:  (4441, 2)
Test Shape:  (1131, 2)
```

```
from keras.preprocessing.text import Tokenizer


tokenizer = Tokenizer()
tokenizer.fit_on_texts(train.Message)


import tensorflow as tf
from sklearn.preprocessing import LabelEncoder



x_train = tokenizer.texts_to_matrix(train.Message, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Message, mode='tfidf')
encode = LabelEncoder()
encode.fit(train.Category)
y_train = tf.keras.utils.to_categorical(encode.transform(train.Category), 2)
y_test = tf.keras.utils.to_categorical(encode.transform(test.Category), 2)
print("train shape: ", x_train.shape, y_train.shape)
print("test shape: ", x_test.shape, y_test.shape)
print("first five test labels: ", y_test[:5])
```

```
train shape:  (4441, 7951) (4441, 2)
test shape:  (1131, 7951) (1131, 2)
first five test labels:  [[1. 0.]
 [1. 0.]
 [0. 1.]
 [0. 1.]]
```

Sequential

```
from keras import layers, models


sequential = models.Sequential()
sequential.add(layers.Dense(32, input_dim=7951, kernel_initializer='normal', activation='relu'))
sequential.add(layers.Dense(32, input_dim=7951, kernel_initializer='normal', activation='relu'))
sequential.add(layers.Dense(2, kernel_initializer='normal', activation='softmax'))
sequential.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = sequential.fit(x_train, y_train, epochs=30, batch_size=128)
```

```
Epoch 2/30
35/35 [==============================] - 1s 14ms/step - loss: 0.2149 - accuracy: 0.8899
Epoch 3/30
35/35 [==============================] - 1s 14ms/step - loss: 0.0724 - accuracy: 0.9896
Epoch 4/30
35/35 [==============================] - 1s 15ms/step - loss: 0.0191 - accuracy: 0.9975
Epoch 5/30
35/35 [==============================] - 1s 15ms/step - loss: 0.0082 - accuracy: 0.9982
Epoch 6/30
35/35 [==============================] - 1s 14ms/step - loss: 0.0043 - accuracy: 0.9991
Epoch 7/30
35/35 [==============================] - 1s 15ms/step - loss: 0.0029 - accuracy: 0.9993
Epoch 8/30
35/35 [==============================] - 0s 14ms/step - loss: 0.0021 - accuracy: 0.9995
Epoch 9/30
35/35 [==============================] - 1s 15ms/step - loss: 0.0016 - accuracy: 1.0000
Epoch 10/30
35/35 [==============================] - 0s 14ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 11/30
35/35 [==============================] - 1s 15ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 12/30
35/35 [==============================] - 1s 15ms/step - loss: 8.8310e-04 - accuracy: 1.0000
Epoch 13/30
35/35 [==============================] - 1s 16ms/step - loss: 7.6221e-04 - accuracy: 1.0000
Epoch 14/30
35/35 [==============================] - 0s 14ms/step - loss: 6.6198e-04 - accuracy: 1.0000
```

```
Epoch 16/30
35/35 [==============================] - 1s 22ms/step - loss: 5.1928e-04 - accuracy: 1.0000
Epoch 17/30
35/35 [==============================] - 1s 18ms/step - loss: 4.6499e-04 - accuracy: 1.0000
Epoch 18/30
35/35 [==============================] - 1s 17ms/step - loss: 4.1754e-04 - accuracy: 1.0000
Epoch 19/30
35/35 [==============================] - 1s 18ms/step - loss: 3.7609e-04 - accuracy: 1.0000
Epoch 20/30
35/35 [==============================] - 0s 11ms/step - loss: 3.4143e-04 - accuracy: 1.0000
Epoch 21/30
35/35 [==============================] - 0s 11ms/step - loss: 3.1176e-04 - accuracy: 1.0000
Epoch 22/30
35/35 [==============================] - 0s 11ms/step - loss: 2.8372e-04 - accuracy: 1.0000
Epoch 23/30
35/35 [==============================] - 0s 11ms/step - loss: 2.6065e-04 - accuracy: 1.0000
Epoch 24/30
35/35 [==============================] - 0s 11ms/step - loss: 2.3953e-04 - accuracy: 1.0000
Epoch 25/30
35/35 [==============================] - 1s 15ms/step - loss: 2.1991e-04 - accuracy: 1.0000
Epoch 26/30
35/35 [==============================] - 1s 15ms/step - loss: 2.0436e-04 - accuracy: 1.0000
Epoch 27/30
35/35 [==============================] - 0s 12ms/step - loss: 1.8862e-04 - accuracy: 1.0000
Epoch 28/30
35/35 [==============================] - 0s 11ms/step - loss: 1.7510e-04 - accuracy: 1.0000
Epoch 29/30
35/35 [==============================] - 0s 11ms/step - loss: 1.6197e-04 - accuracy: 1.0000
Epoch 30/30
35/35 [==============================] - 0s 12ms/step - loss: 1.5180e-04 - accuracy: 1.0000
```

```
score = sequential.evaluate(x_test, y_test, batch_size=100, verbose=1)
print('Accuracy: ', score[1])
```

```
12/12 [==============================] - 0s 9ms/step - loss: 0.0871 - accuracy: 0.9894
Accuracy:  0.9893898963928223
```

CNN

```
cnn = models.Sequential()
cnn.add(layers.Embedding(10000, 128, input_length=7951))
cnn.add(layers.Conv1D(32, 7, activation='relu'))
cnn.add(layers.MaxPooling1D(5))
cnn.add(layers.Conv1D(32, 7, activation='relu'))
cnn.add(layers.GlobalMaxPooling1D())
cnn.add(layers.Dense(2))
cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = cnn.fit(x_train, y_train, epochs=5, batch_size=64)
```

```
Epoch 1/5
70/70 [==============================] - 291s 4s/step - loss: 0.3786 - accuracy: 0.1493
Epoch 2/5
70/70 [==============================] - 278s 4s/step - loss: 0.3707 - accuracy: 0.1385
Epoch 3/5
70/70 [==============================] - 265s 4s/step - loss: 0.5425 - accuracy: 0.1353
Epoch 4/5
70/70 [==============================] - 265s 4s/step - loss: 0.3870 - accuracy: 0.1385
Epoch 5/5
70/70 [==============================] - 265s 4s/step - loss: 0.3624 - accuracy: 0.1376
```

```
score = cnn.evaluate(x_test, y_test, batch_size=100, verbose=1)
print('Accuracy: ', score[1])
```

```
12/12 [==============================] - 15s 1s/step - loss: 0.3891 - accuracy: 0.1167
Accuracy:  0.11671087890863419
```

Embedding

```
embed = models.Sequential()
embed.add(layers.Embedding(10000, 8, input_length=7951))
embed.add(layers.Flatten())
embed.add(layers.Dense(32, activation='relu'))
embed.add(layers.Dense(2, activation='relu'))
```

```
embed.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = embed.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=.2)
```

```
Epoch 1/10
111/111 [==============================] - 8s 61ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 2/10
111/111 [==============================] - 6s 53ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 3/10
111/111 [==============================] - 7s 59ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 4/10
111/111 [==============================] - 5s 44ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 5/10
111/111 [==============================] - 6s 56ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 6/10
111/111 [==============================] - 5s 49ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 7/10
111/111 [==============================] - 5s 45ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 8/10
111/111 [==============================] - 7s 61ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 9/10
111/111 [==============================] - 5s 46ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
Epoch 10/10
111/111 [==============================] - 7s 66ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8
```

```
score = embed.evaluate(x_test, y_test, batch_size=100, verbose=1)
print('Accuracy: ', score[1])
```

```
12/12 [==============================] - 1s 44ms/step - loss: nan - accuracy: 0.8833
Accuracy:  0.883289098739624
```

Analysis

The highest accuracy that I achieved was with the sequential model, followed by the embedding, with CNN having the lowest accuracy by far. My sequential model achieved an accuracy of 99%, while the embedding only achieved 88%. The CNN was very low, at only 11.6% accuracy.

✓ 0s completed at 12:59 PM