The dataset I am using is a spam text message classification dataset. The model should be able to predict if a given text is real or spam.

Ham is real text messages, while spam is fake messages

```
import io
import pandas as pd
import tensorflow as tf
import pickle
import numpy as np
from google.colab import files
from tensorflow import keras
from keras.preprocessing.text import Tokenizer
from keras import layers, models
from sklearn.preprocessing import LabelEncoder
from tensorflow.python import metrics
from keras import preprocessing
```

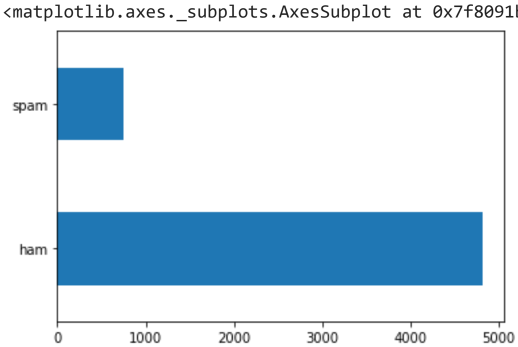
```
data = pd.read_csv('spamtextdata.csv', header=0, usecols=[0,1])
display(data)
```

	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 × 2 columns

```
data['Category'].value_counts().plot(kind='barh')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f8091b63bb0>




```
np.random.seed(80085)
i = np.random.rand(len(data)) < 0.8
train = data[i]
test = data[~i]
print('Train shape: ', train.shape)
print('Test shape: ', test.shape)
```

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

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

```
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.]
 [1. 0.]
 [0. 1.]
 [0. 1.]]
```

Sequential

```
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 [==============================] - 0s 10ms/step - loss: 0.2225 - accuracy: 0.8854
Epoch 3/30
35/35 [==============================] - 0s 10ms/step - loss: 0.0923 - accuracy: 0.9827
Epoch 4/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0263 - accuracy: 0.9957
Epoch 5/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0101 - accuracy: 0.9980
Epoch 6/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0053 - accuracy: 0.9998
Epoch 7/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0033 - accuracy: 0.9998
Epoch 8/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0023 - accuracy: 0.9998
Epoch 9/30
35/35 [==============================] - 0s 10ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 10/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 11/30
35/35 [==============================] - 0s 9ms/step - loss: 0.0010 - accuracy: 1.0000
Epoch 12/30
35/35 [==============================] - 0s 9ms/step - loss: 8.4217e-04 - accuracy: 1.0000
Epoch 13/30
35/35 [==============================] - 0s 9ms/step - loss: 6.9547e-04 - accuracy: 1.0000
Epoch 14/30
35/35 [==============================] - 0s 9ms/step - loss: 5.8967e-04 - accuracy: 1.0000
Epoch 15/30
35/35 [==============================] - 0s 10ms/step - loss: 4.9998e-04 - accuracy: 1.0000
Epoch 16/30
35/35 [==============================] - 0s 9ms/step - loss: 4.3307e-04 - accuracy: 1.0000
Epoch 17/30
35/35 [==============================] - 0s 9ms/step - loss: 3.7568e-04 - accuracy: 1.0000
Epoch 18/30
35/35 [==============================] - 0s 9ms/step - loss: 3.3122e-04 - accuracy: 1.0000
Epoch 19/30
35/35 [==============================] - 0s 9ms/step - loss: 2.9274e-04 - accuracy: 1.0000
Epoch 20/30
35/35 [==============================] - 0s 9ms/step - loss: 2.6384e-04 - accuracy: 1.0000
Epoch 21/30
35/35 [==============================] - 0s 10ms/step - loss: 2.3657e-04 - accuracy: 1.0000
Epoch 22/30
35/35 [==============================] - 0s 9ms/step - loss: 2.1242e-04 - accuracy: 1.0000
Epoch 23/30
35/35 [==============================] - 0s 9ms/step - loss: 1.9376e-04 - accuracy: 1.0000
Epoch 24/30
35/35 [==============================] - 0s 9ms/step - loss: 1.7544e-04 - accuracy: 1.0000
Epoch 25/30
```

```
35/35 [==============================] - 0s 9ms/step - loss: 1.4698e-04 - accuracy: 1.0000
Epoch 27/30
35/35 [==============================] - 0s 10ms/step - loss: 1.3448e-04 - accuracy: 1.0000
Epoch 28/30
35/35 [==============================] - 0s 9ms/step - loss: 1.2428e-04 - accuracy: 1.0000
Epoch 29/30
35/35 [==============================] - 0s 9ms/step - loss: 1.1481e-04 - accuracy: 1.0000
Epoch 30/30
35/35 [==============================] - 0s 9ms/step - loss: 1.0659e-04 - accuracy: 1.0000
```

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

```
12/12 [==============================] - 0s 3ms/step - loss: 0.0833 - accuracy: 0.9903
Accuracy:  0.9902740716934204
```

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 [==============================] - 212s 3s/step - loss: 13.6829 - accuracy: 0.1385
Epoch 2/5
70/70 [==============================] - 211s 3s/step - loss: 13.8860 - accuracy: 0.1385
Epoch 3/5
70/70 [==============================] - 209s 3s/step - loss: 13.8860 - accuracy: 0.1385
Epoch 4/5
70/70 [==============================] - 209s 3s/step - loss: 13.8860 - accuracy: 0.1385
Epoch 5/5
70/70 [==============================] - 214s 3s/step - loss: 13.8860 - accuracy: 0.1385
```

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

```
12/12 [==============================] - 11s 888ms/step - loss: 14.2369 - 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 [==============================] - 4s 36ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 2/10
111/111 [==============================] - 4s 38ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 3/10
111/111 [==============================] - 4s 38ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 4/10
111/111 [==============================] - 4s 38ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 5/10
111/111 [==============================] - 4s 39ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 6/10
111/111 [==============================] - 4s 39ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 7/10
111/111 [==============================] - 4s 39ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 8/10
111/111 [==============================] - 4s 39ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
Epoch 9/10
111/111 [==============================] - 4s 40ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
```

```
Epoch 10/10
111/111 [==============================] - 4s 40ms/step - loss: nan - accuracy: 0.8606 - val_loss: nan - val_accuracy: 0.8650
```

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

```
12/12 [==============================] - 0s 23ms/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.

My final project for my Machine Learning class was to analyze the performance of various machine learning models when trained on different datasets. My findings were the CNNs really relied on a large dataset and a decent number of epochs to achieve great performance. Their benefits are also much more apparent with more complex datasets. I believe this is a large part of why it was the worst performer here. The dataset that I used for this assignment was pretty small, under 8000 entries. It was also a very simple dataset, with only two classifications.