Jonathan Ho

CS 4375

Text Classification 2

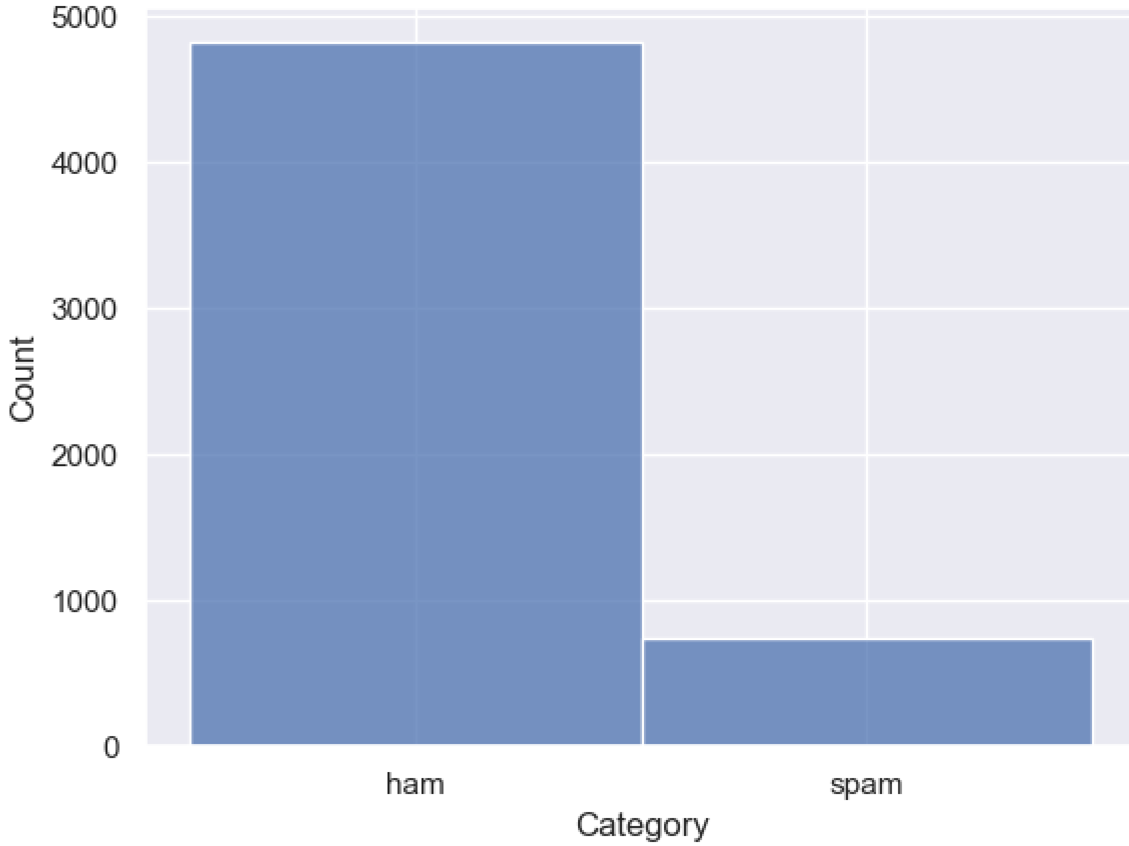
Import Packages

In [1]:
```
from tensorflow import keras
from keras.layers import SimpleRNN, Embedding, Dense, LSTM
from keras.models import Sequential
from keras import layers, models
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from sklearn.metrics import classification_report

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns; sns.set()

import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import PorterStemmer
from sklearn import preprocessing
```

In [2]:
```
# Load data and print categorical graph
data = pd.read_csv("spam_data.csv", encoding='utf-8')
print(data)

sns.histplot(data['Category'])
```

```
     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]
```

Out[2]:
```
<Axes: xlabel='Category', ylabel='Count'>
```



The dataset I chose is of texts that belong to one of two targets: spam and ham. Spam are messages that you would receive unsolicited, while ham are messages that you would typically receive from friends. The model should be able to differentiate messages into the two categories.

In [3]:
```
# Encode ham and spam as 0 and 1 respectively for binary classification
texts = []
labels = []
for i, label in enumerate(data['Category']):
    texts.append(data['Message'][i])
    if label == 'ham':
        labels.append(0)
    else:
        labels.append(1)

texts = np.asarray(texts)
labels = np.asarray(labels)


print("number of texts :" , len(texts))
print("number of labels: ", len(labels))
```

```
number of texts : 5572
number of labels:  5572
```

In [4]:
```
# Number of words used as features
max_features = 10000
# Max 500 words
maxlen = 500
```

```
# 80% Training, 20% Validation
training_samples = int(5572 * .8)
validation_samples = int(5572 - training_samples)

# Tokenize the data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Create word index
word_index = tokenizer.word_index

# Pad the data
data = pad_sequences(sequences, maxlen=maxlen)

np.random.seed(1234)

# Shuffle data
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_test = data[training_samples:]
y_test = labels[training_samples:]
```

In [5]:
```
# Sequential Model, must be 500 for the shape
model = Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(500,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

# Compile
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train
history_seq = model.fit(x_train, y_train, epochs=10, batch_size=60, validation_split=
```

```
Epoch 1/10
60/60 [==============================] - 1s 5ms/step - loss: 13.5596 - accuracy: 0.79
27 - val_loss: 5.8023 - val_accuracy: 0.7455
Epoch 2/10
60/60 [==============================] - 0s 2ms/step - loss: 4.5898 - accuracy: 0.807
3 - val_loss: 2.4691 - val_accuracy: 0.8318
Epoch 3/10
60/60 [==============================] - 0s 2ms/step - loss: 2.1962 - accuracy: 0.837
6 - val_loss: 1.5908 - val_accuracy: 0.8531
Epoch 4/10
60/60 [==============================] - 0s 2ms/step - loss: 1.2125 - accuracy: 0.848
8 - val_loss: 1.1677 - val_accuracy: 0.8531
Epoch 5/10
60/60 [==============================] - 0s 2ms/step - loss: 0.7840 - accuracy: 0.859
7 - val_loss: 1.0031 - val_accuracy: 0.8509
Epoch 6/10
60/60 [==============================] - 0s 2ms/step - loss: 0.6430 - accuracy: 0.864
5 - val_loss: 0.9819 - val_accuracy: 0.8632
Epoch 7/10
60/60 [==============================] - 0s 2ms/step - loss: 0.5621 - accuracy: 0.871
2 - val_loss: 1.0168 - val_accuracy: 0.8643
Epoch 8/10
60/60 [==============================] - 0s 2ms/step - loss: 0.4978 - accuracy: 0.879
4 - val_loss: 0.8900 - val_accuracy: 0.8587
Epoch 9/10
60/60 [==============================] - 0s 2ms/step - loss: 0.4460 - accuracy: 0.880
2 - val_loss: 0.9036 - val_accuracy: 0.8565
Epoch 10/10
60/60 [==============================] - 0s 2ms/step - loss: 0.4168 - accuracy: 0.881
9 - val_loss: 0.8228 - val_accuracy: 0.8430
```

In [6]:

```
# Sequential training and validation accuracy
plt.plot(history_seq.history['accuracy'])
plt.plot(history_seq.history['val_accuracy'])
plt.title('Sequential Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```

In [7]:
```
# Prediction accuracy on test data
pred = model.predict(x_test)
pred = [1.0 if p>=0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

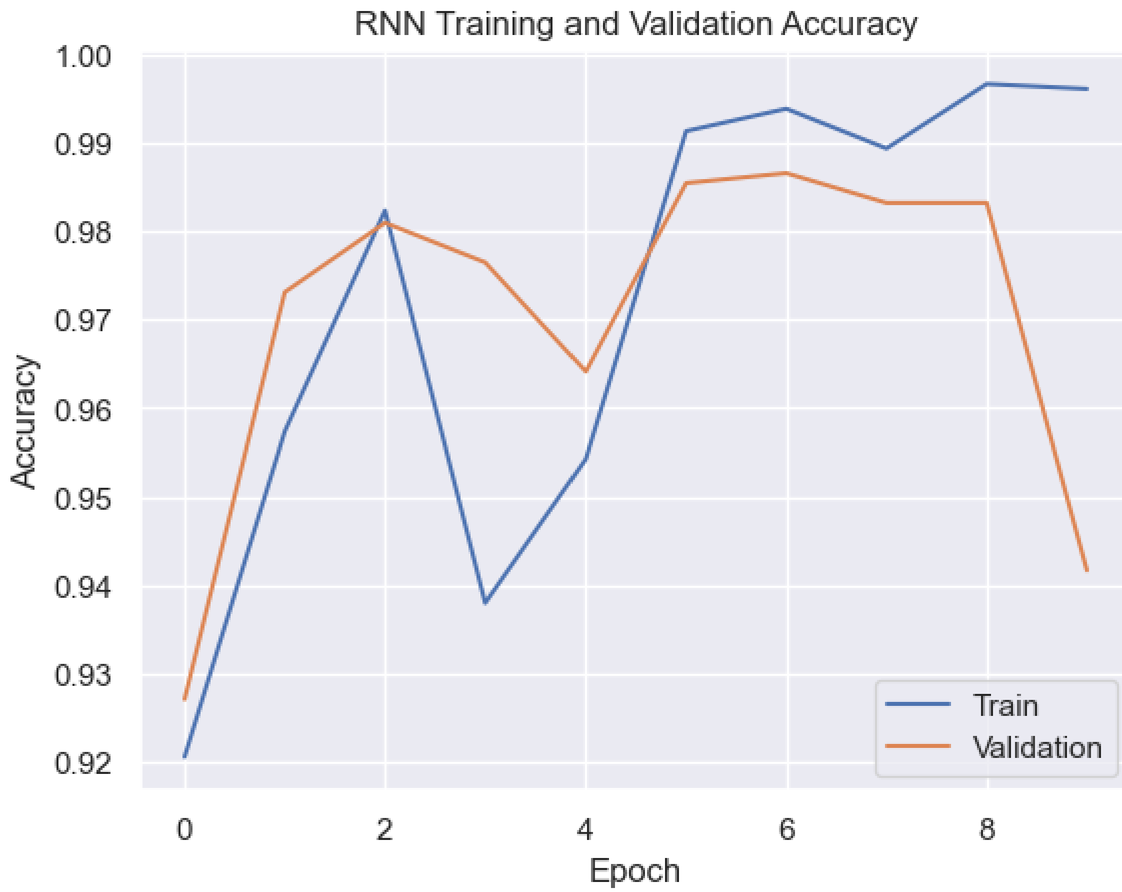
```
35/35 [==============================] - 0s 913us/step
              precision    recall  f1-score   support

           0       0.93      0.86      0.89       950
           1       0.43      0.61      0.51       165

    accuracy                           0.82      1115
   macro avg       0.68      0.74      0.70      1115
weighted avg       0.85      0.82      0.84      1115
```

In [8]:
```
# RNN Model
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

# Compile
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train
history_rnn = model.fit(x_train, y_train, epochs=10, batch_size=60, validation_split=0
```

```
Epoch 1/10
60/60 [==============================] - 5s 64ms/step - loss: 0.2548 - accuracy: 0.92
06 - val_loss: 0.2126 - val_accuracy: 0.9271
Epoch 2/10
60/60 [==============================] - 4s 60ms/step - loss: 0.1419 - accuracy: 0.95
74 - val_loss: 0.0946 - val_accuracy: 0.9731
Epoch 3/10
60/60 [==============================] - 4s 60ms/step - loss: 0.0641 - accuracy: 0.98
23 - val_loss: 0.0703 - val_accuracy: 0.9809
Epoch 4/10
60/60 [==============================] - 4s 62ms/step - loss: 0.1536 - accuracy: 0.93
80 - val_loss: 0.0923 - val_accuracy: 0.9765
Epoch 5/10
60/60 [==============================] - 4s 60ms/step - loss: 0.1179 - accuracy: 0.95
43 - val_loss: 0.1083 - val_accuracy: 0.9641
Epoch 6/10
60/60 [==============================] - 4s 60ms/step - loss: 0.0345 - accuracy: 0.99
13 - val_loss: 0.0629 - val_accuracy: 0.9854
Epoch 7/10
60/60 [==============================] - 4s 62ms/step - loss: 0.0240 - accuracy: 0.99
38 - val_loss: 0.0641 - val_accuracy: 0.9865
Epoch 8/10
60/60 [==============================] - 4s 62ms/step - loss: 0.0350 - accuracy: 0.98
93 - val_loss: 0.0610 - val_accuracy: 0.9832
Epoch 9/10
60/60 [==============================] - 4s 62ms/step - loss: 0.0114 - accuracy: 0.99
66 - val_loss: 0.0702 - val_accuracy: 0.9832
Epoch 10/10
60/60 [==============================] - 4s 60ms/step - loss: 0.0114 - accuracy: 0.99
61 - val_loss: 0.1676 - val_accuracy: 0.9417
```

In [9]:

```python
# RNN training and validation accuracy
plt.plot(history_rnn.history['accuracy'])
plt.plot(history_rnn.history['val_accuracy'])
plt.title('RNN Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



In [10]:
```
# Prediction accuracy on test data
pred = model.predict(x_test)
pred = [1.0 if p>=0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

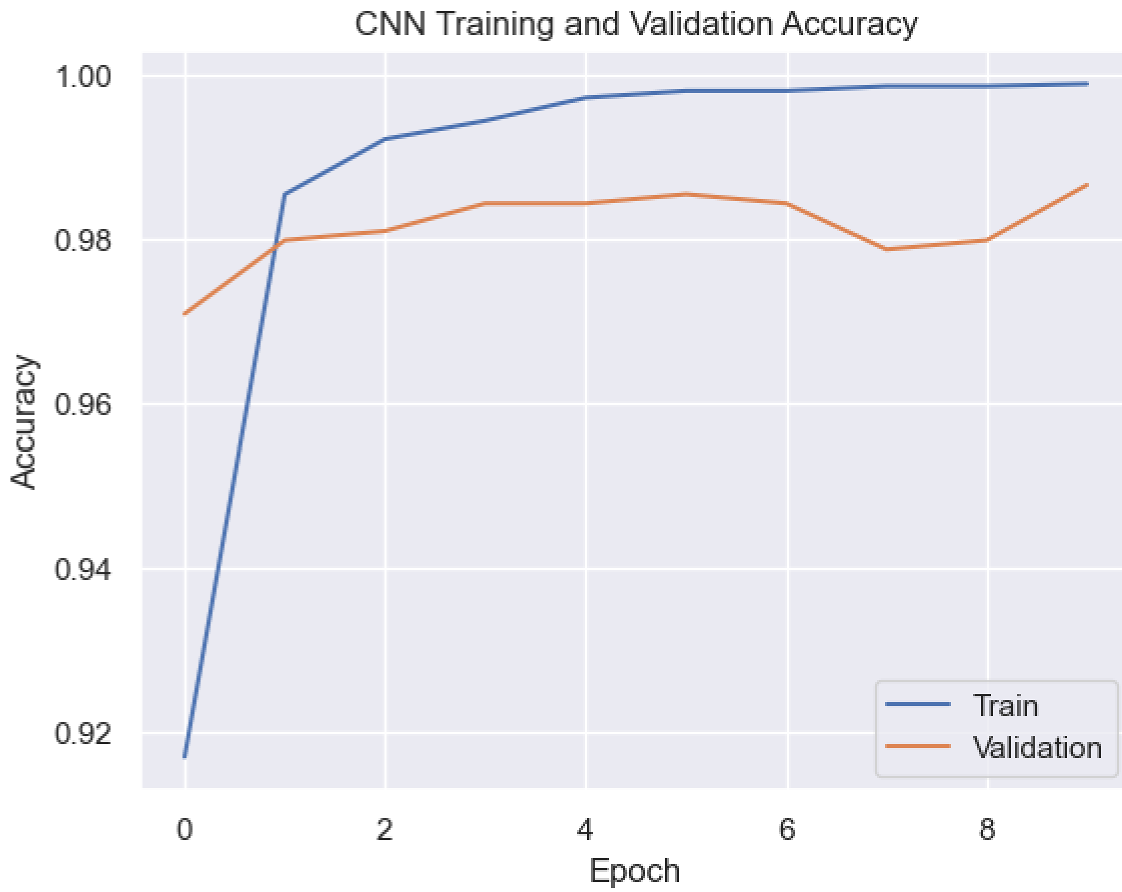
```
35/35 [==============================] - 1s 14ms/step
              precision    recall  f1-score   support

           0       0.98      0.94      0.96       950
           1       0.73      0.92      0.81       165

    accuracy                           0.94      1115
   macro avg       0.86      0.93      0.89      1115
weighted avg       0.95      0.94      0.94      1115
```

In [11]:
```
# CNN
model = Sequential()
model.add(layers.Embedding(max_features, 64, input_length=maxlen))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

# Compile
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train
history_cnn = model.fit(x_train, y_train, epochs=10, batch_size=60, validation_split=0
```

```
Epoch 1/10
60/60 [==============================] - 2s 20ms/step - loss: 0.2215 - accuracy: 0.91
70 - val_loss: 0.0811 - val_accuracy: 0.9709
Epoch 2/10
60/60 [==============================] - 1s 18ms/step - loss: 0.0609 - accuracy: 0.98
54 - val_loss: 0.1131 - val_accuracy: 0.9798
Epoch 3/10
60/60 [==============================] - 1s 17ms/step - loss: 0.0356 - accuracy: 0.99
21 - val_loss: 0.1187 - val_accuracy: 0.9809
Epoch 4/10
60/60 [==============================] - 1s 17ms/step - loss: 0.0270 - accuracy: 0.99
44 - val_loss: 0.1419 - val_accuracy: 0.9843
Epoch 5/10
60/60 [==============================] - 1s 17ms/step - loss: 0.0218 - accuracy: 0.99
72 - val_loss: 0.1551 - val_accuracy: 0.9843
Epoch 6/10
60/60 [==============================] - 1s 19ms/step - loss: 0.0157 - accuracy: 0.99
80 - val_loss: 0.1288 - val_accuracy: 0.9854
Epoch 7/10
60/60 [==============================] - 1s 20ms/step - loss: 0.0155 - accuracy: 0.99
80 - val_loss: 0.1423 - val_accuracy: 0.9843
Epoch 8/10
60/60 [==============================] - 1s 18ms/step - loss: 0.0143 - accuracy: 0.99
86 - val_loss: 0.1335 - val_accuracy: 0.9787
Epoch 9/10
60/60 [==============================] - 1s 20ms/step - loss: 0.0140 - accuracy: 0.99
86 - val_loss: 0.1190 - val_accuracy: 0.9798
Epoch 10/10
60/60 [==============================] - 1s 21ms/step - loss: 0.0140 - accuracy: 0.99
89 - val_loss: 0.1594 - val_accuracy: 0.9865
```

In [12]:

```python
# CNN training and validation accuracy
plt.plot(history_cnn.history['accuracy'])
plt.plot(history_cnn.history['val_accuracy'])
plt.title('CNN Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



In [13]:
```
# Prediction accuracy on test data
pred = model.predict(x_test)
pred = [1.0 if p>=0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

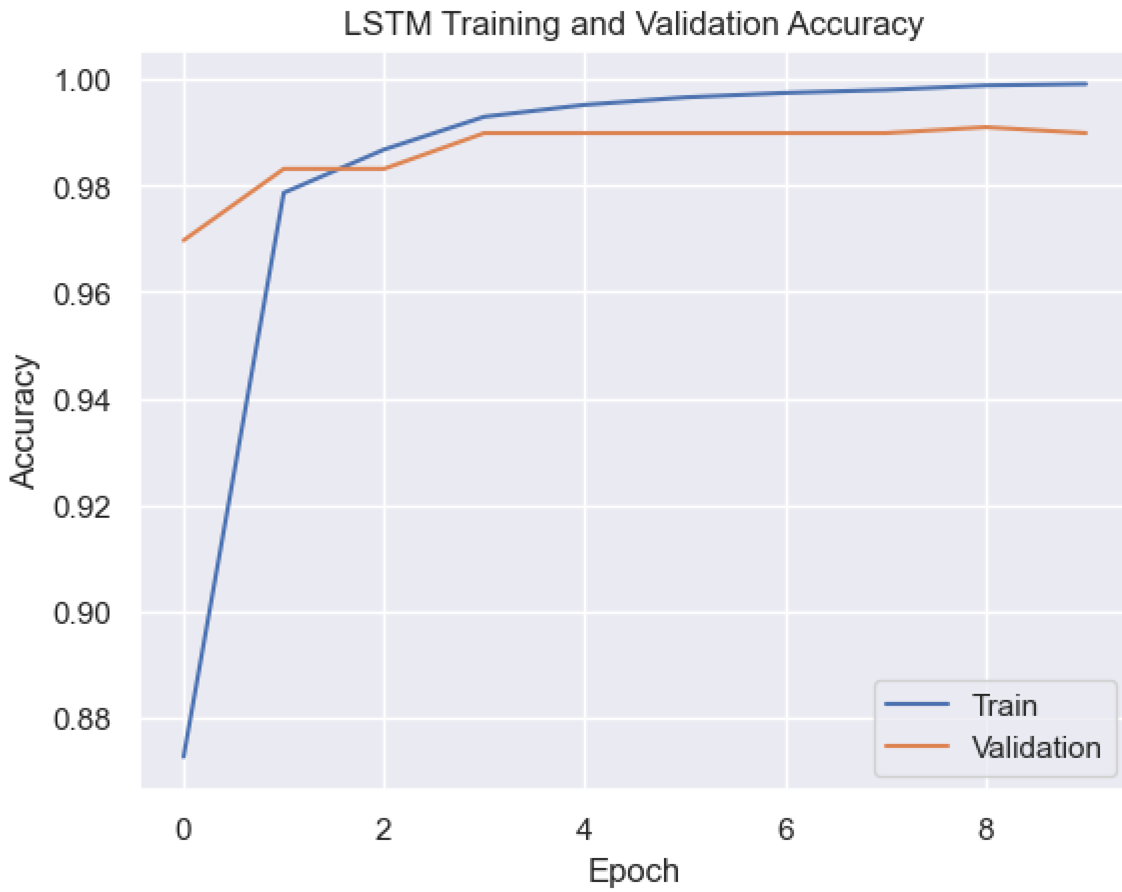
```
35/35 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       950
           1       0.96      0.90      0.93       165

    accuracy                           0.98      1115
   macro avg       0.97      0.95      0.96      1115
weighted avg       0.98      0.98      0.98      1115
```

In [14]:
```
# LSTM
model = Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

# Compile
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train
history_lstm = model.fit(x_train, y_train, epochs=10, batch_size=60, validation_split=
```

```
Epoch 1/10
60/60 [==============================] - 11s 159ms/step - loss: 0.3025 - accuracy: 0.
8727 - val_loss: 0.1719 - val_accuracy: 0.9697
Epoch 2/10
60/60 [==============================] - 9s 151ms/step - loss: 0.0990 - accuracy: 0.9
787 - val_loss: 0.0750 - val_accuracy: 0.9832
Epoch 3/10
60/60 [==============================] - 9s 158ms/step - loss: 0.0592 - accuracy: 0.9
868 - val_loss: 0.0775 - val_accuracy: 0.9832
Epoch 4/10
60/60 [==============================] - 9s 153ms/step - loss: 0.0373 - accuracy: 0.9
930 - val_loss: 0.0597 - val_accuracy: 0.9899
Epoch 5/10
60/60 [==============================] - 9s 154ms/step - loss: 0.0204 - accuracy: 0.9
952 - val_loss: 0.0557 - val_accuracy: 0.9899
Epoch 6/10
60/60 [==============================] - 9s 153ms/step - loss: 0.0144 - accuracy: 0.9
966 - val_loss: 0.0520 - val_accuracy: 0.9899
Epoch 7/10
60/60 [==============================] - 9s 155ms/step - loss: 0.0101 - accuracy: 0.9
975 - val_loss: 0.0538 - val_accuracy: 0.9899
Epoch 8/10
60/60 [==============================] - 9s 152ms/step - loss: 0.0069 - accuracy: 0.9
980 - val_loss: 0.0632 - val_accuracy: 0.9899
Epoch 9/10
60/60 [==============================] - 9s 153ms/step - loss: 0.0052 - accuracy: 0.9
989 - val_loss: 0.0566 - val_accuracy: 0.9910
Epoch 10/10
60/60 [==============================] - 9s 155ms/step - loss: 0.0039 - accuracy: 0.9
992 - val_loss: 0.0593 - val_accuracy: 0.9899
```

In [15]:

```python
# LSTM training and validation accuracy
plt.plot(history_lstm.history['accuracy'])
plt.plot(history_lstm.history['val_accuracy'])
plt.title('LSTM Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



In [16]:

```
# Prediction accuracy on test data
pred = model.predict(x_test)
pred = [1.0 if p>=0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
35/35 [==============================] - 1s 28ms/step
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       950
           1       0.97      0.93      0.95       165

    accuracy                           0.99      1115
   macro avg       0.98      0.96      0.97      1115
weighted avg       0.99      0.99      0.99      1115
```

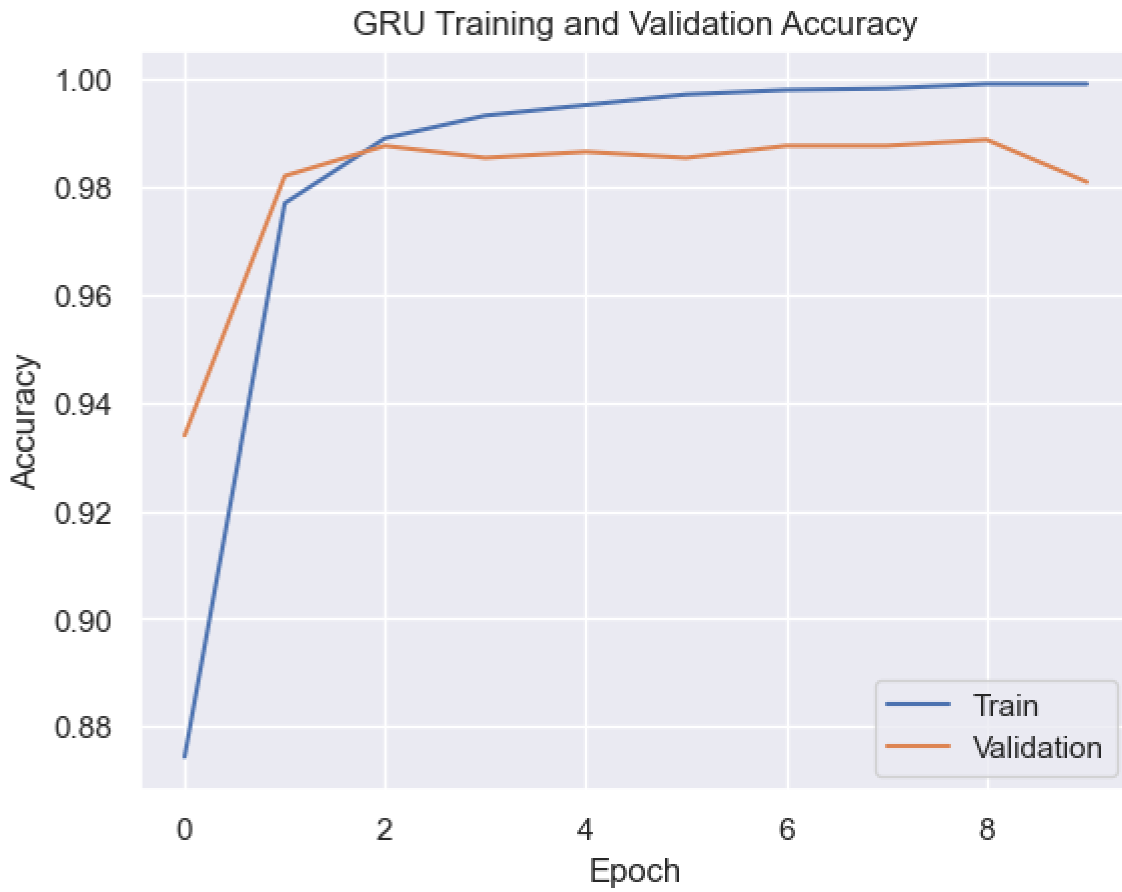
In [17]:

```
# GRU
model = Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.GRU(32))
model.add(layers.Dense(1, activation='sigmoid'))

# Compile
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# Train
history_gru = model.fit(x_train, y_train, epochs=10, batch_size=60, validation_split=0
```

```
Epoch 1/10
60/60 [==============================] - 13s 183ms/step - loss: 0.3384 - accuracy: 0.
8743 - val_loss: 0.1547 - val_accuracy: 0.9339
Epoch 2/10
60/60 [==============================] - 10s 165ms/step - loss: 0.0827 - accuracy: 0.
9770 - val_loss: 0.0696 - val_accuracy: 0.9821
Epoch 3/10
60/60 [==============================] - 10s 168ms/step - loss: 0.0364 - accuracy: 0.
9891 - val_loss: 0.0531 - val_accuracy: 0.9877
Epoch 4/10
60/60 [==============================] - 10s 171ms/step - loss: 0.0222 - accuracy: 0.
9933 - val_loss: 0.0537 - val_accuracy: 0.9854
Epoch 5/10
60/60 [==============================] - 10s 167ms/step - loss: 0.0161 - accuracy: 0.
9952 - val_loss: 0.0553 - val_accuracy: 0.9865
Epoch 6/10
60/60 [==============================] - 10s 169ms/step - loss: 0.0115 - accuracy: 0.
9972 - val_loss: 0.0596 - val_accuracy: 0.9854
Epoch 7/10
60/60 [==============================] - 10s 169ms/step - loss: 0.0080 - accuracy: 0.
9980 - val_loss: 0.0630 - val_accuracy: 0.9877
Epoch 8/10
60/60 [==============================] - 10s 172ms/step - loss: 0.0069 - accuracy: 0.
9983 - val_loss: 0.0637 - val_accuracy: 0.9877
Epoch 9/10
60/60 [==============================] - 11s 177ms/step - loss: 0.0039 - accuracy: 0.
9992 - val_loss: 0.0798 - val_accuracy: 0.9888
Epoch 10/10
60/60 [==============================] - 10s 167ms/step - loss: 0.0038 - accuracy: 0.
9992 - val_loss: 0.0702 - val_accuracy: 0.9809
```

In [18]:

```python
# GRU training and validation accuracy
plt.plot(history_gru.history['accuracy'])
plt.plot(history_gru.history['val_accuracy'])
plt.title('GRU Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



In [19]:
```
# Prediction accuracy on test data
pred = model.predict(x_test)
pred = [1.0 if p>=0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
35/35 [==============================] - 2s 34ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       950
           1       0.93      0.90      0.91       165

    accuracy                           0.97      1115
   macro avg       0.95      0.95      0.95      1115
weighted avg       0.97      0.97      0.97      1115
```

GloVe Embedding

In [31]:
```
import os

path_to_glove_file = os.path.join(
    os.path.expanduser("~"), "Desktop/glove.6B.100d.txt"
)

embeddings_index = {}
with open(path_to_glove_file, encoding='utf-8') as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep = ",")
```

```
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
C:\Users\Jonathan\AppData\Local\Temp\ipykernel_22076\4155444208.py:11: DeprecationWar
ning: string or file could not be read to its end due to unmatched data; this will ra
ise a ValueError in the future.
  coefs = np.fromstring(coefs, "f", sep = ",")
Found 400000 word vectors.
```

In [32]:

```
embedding_dim = 100
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((max_features, embedding_dim))
for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1

print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 6550 words (2454 misses)
```

In [33]:

```
# Create embedding layer from GloVe model
embedding_layer = Embedding(
    max_features,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
)
```

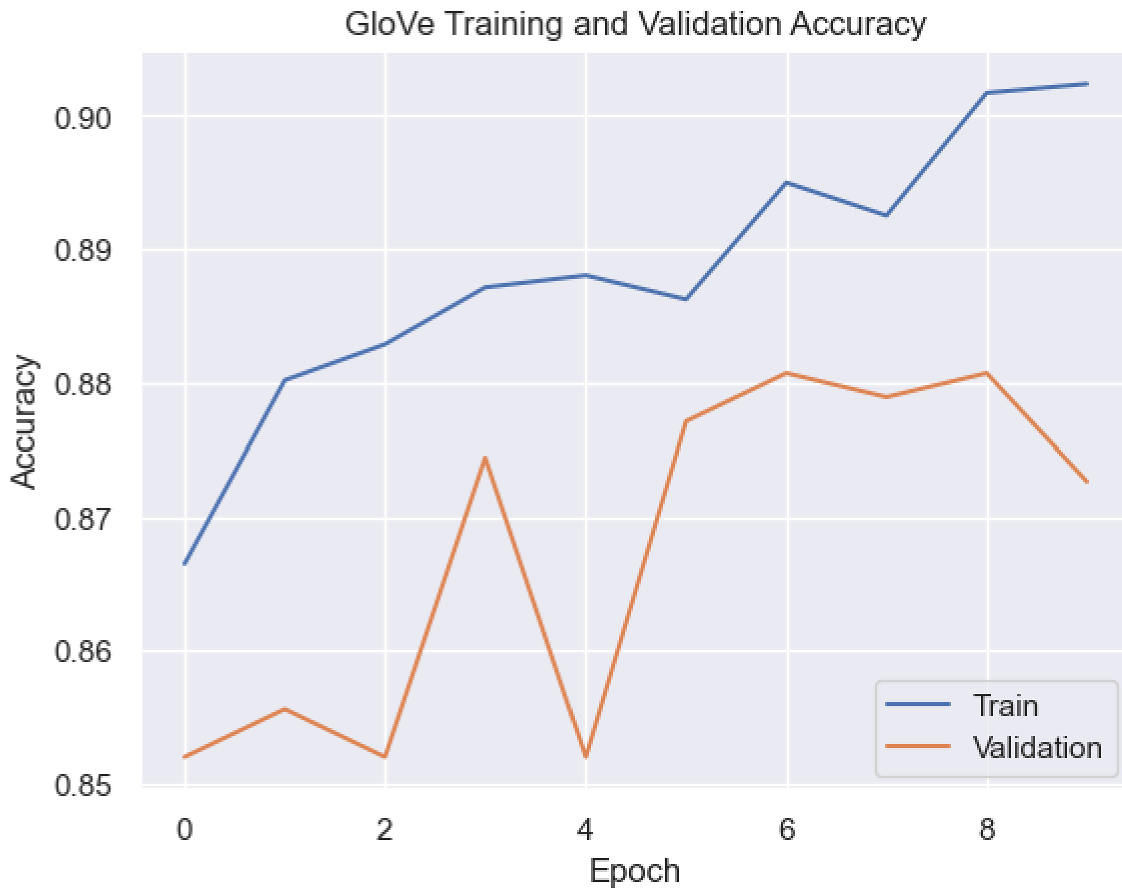
In [34]:

```
# GloVe Embedding
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 5, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dropout(0.5)(x)
preds = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(int_sequences_input, preds)

# Compile
model.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=["accuracy"])

# Train
history_glove = model.fit(x_train, y_train, batch_size=60, epochs=10, validation_data=
```

```
Epoch 1/10
75/75 [==============================] - 5s 60ms/step - loss: 0.3237 - accuracy: 0.86
65 - val_loss: 0.3143 - val_accuracy: 0.8520
Epoch 2/10
75/75 [==============================] - 4s 60ms/step - loss: 0.2684 - accuracy: 0.88
02 - val_loss: 0.2978 - val_accuracy: 0.8556
Epoch 3/10
75/75 [==============================] - 4s 58ms/step - loss: 0.2631 - accuracy: 0.88
29 - val_loss: 0.2906 - val_accuracy: 0.8520
Epoch 4/10
75/75 [==============================] - 4s 53ms/step - loss: 0.2600 - accuracy: 0.88
71 - val_loss: 0.2779 - val_accuracy: 0.8744
Epoch 5/10
75/75 [==============================] - 4s 54ms/step - loss: 0.2515 - accuracy: 0.88
80 - val_loss: 0.4312 - val_accuracy: 0.8520
Epoch 6/10
75/75 [==============================] - 5s 61ms/step - loss: 0.2526 - accuracy: 0.88
62 - val_loss: 0.2729 - val_accuracy: 0.8771
Epoch 7/10
75/75 [==============================] - 4s 59ms/step - loss: 0.2466 - accuracy: 0.89
50 - val_loss: 0.2862 - val_accuracy: 0.8807
Epoch 8/10
75/75 [==============================] - 4s 54ms/step - loss: 0.2384 - accuracy: 0.89
25 - val_loss: 0.2877 - val_accuracy: 0.8789
Epoch 9/10
75/75 [==============================] - 4s 56ms/step - loss: 0.2352 - accuracy: 0.90
17 - val_loss: 0.2761 - val_accuracy: 0.8807
Epoch 10/10
75/75 [==============================] - 5s 61ms/step - loss: 0.2329 - accuracy: 0.90
24 - val_loss: 0.2991 - val_accuracy: 0.8726
```

In [35]:

```
# GloVe training and validation accuracy
plt.plot(history_glove.history['accuracy'])
plt.plot(history_glove.history['val_accuracy'])
plt.title('GloVe Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



In [36]:
```python
# Prediction accuracy on test data
pred = model.predict(x_test)
pred = [1.0 if p>=0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
35/35 [==============================] - 0s 9ms/step
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       950
           1       0.63      0.34      0.44       165

    accuracy                           0.87      1115
   macro avg       0.76      0.65      0.68      1115
weighted avg       0.85      0.87      0.86      1115
```

Model Analysis

Overall, the models did well to predict whether text was ham or spam. Relatively though, CNN, LSTM, and GRU did good while sequential, RNN, and GloVe did worse. Specifically, sequential did the worst and LSTM did the best.

It was no surprise that sequential did the worst since modifications other than Dense layers were used. The results yielded about an 82% accuracy since it was not able to predict spam messages well (only a 52% success rate).

CNN, LSTM, and GRU all did really well, although it is a bit concerning how well. I am not sure if it is because of overfitting, or because it is a small dataset, but all three models did really well in identifying spam/ham messages (all above 90% success rates).

Lastly is testing the embedding technique of utilizing a GloVe pretrained model. I had expected results to be on par of the models other than sequential, but it only did better than sequential. It again suffered with trying to identify spam messages from the validation dataset.

Referenced Code for Models:

https://www.kaggle.com/code/kentata/rnn-for-spam-detection/notebook