Text Classification 2

About the Dataset

The dataset is made up of headlines from posts of 3 different subreddits dedicated to their respective popular Asian Netflix series: Squid Games, Physical 100, and Alice in Borderland. My model will attempt to classify which show ('title') a headline is referring to.

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import datasets, layers, models, preprocessing, callbacks
from keras.optimizers import SGD
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.preprocessing import LabelEncoder

!pip install --upgrade tensorflow_hub
import tensorflow_hub as hub
!pip install tensorflow-text
import tensorflow_text as text
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow_hub in /usr/local/lib/python3.9/dist-packages (0.13.0)
```

```
Requirement already satisfied: protobuf>=3.19.6 in /usr/local/lib/python3.9/dist-packages (from tensorflow_hub) (3.20.3)
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow_hub) (1.22.4)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow-text in /usr/local/lib/python3.9/dist-packages (2.12.1)
Requirement already satisfied: tensorflow<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow-text) (2
Requirement already satisfied: tensorflow-hub>=0.8.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow-text) (0.13
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->tensor
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->t
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflo
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->t
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/loc
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->tensorf
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->t
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->tensorfl
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.9/dist-packages (from tensorf
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->tensor
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->tensor
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0->ter
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<2.13,>=2.12.0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.9/dist-packages (from astunparse>=1.6.0->tensor
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.9/dist-packages (from jax>=0.3.15->tensorflow<2.13,>=2.
Requirement already satisfied: ml-dtypes>=0.0.3 in /usr/local/lib/python3.9/dist-packages (from jax>=0.3.15->tensorflow<2.1
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->ter
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from tensor
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.1
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12->ter
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.13,>=2.12-
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorb
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from google-auth-oauthli
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.9/dist-packages (from markdown>=2.6.8->ter
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboar
```

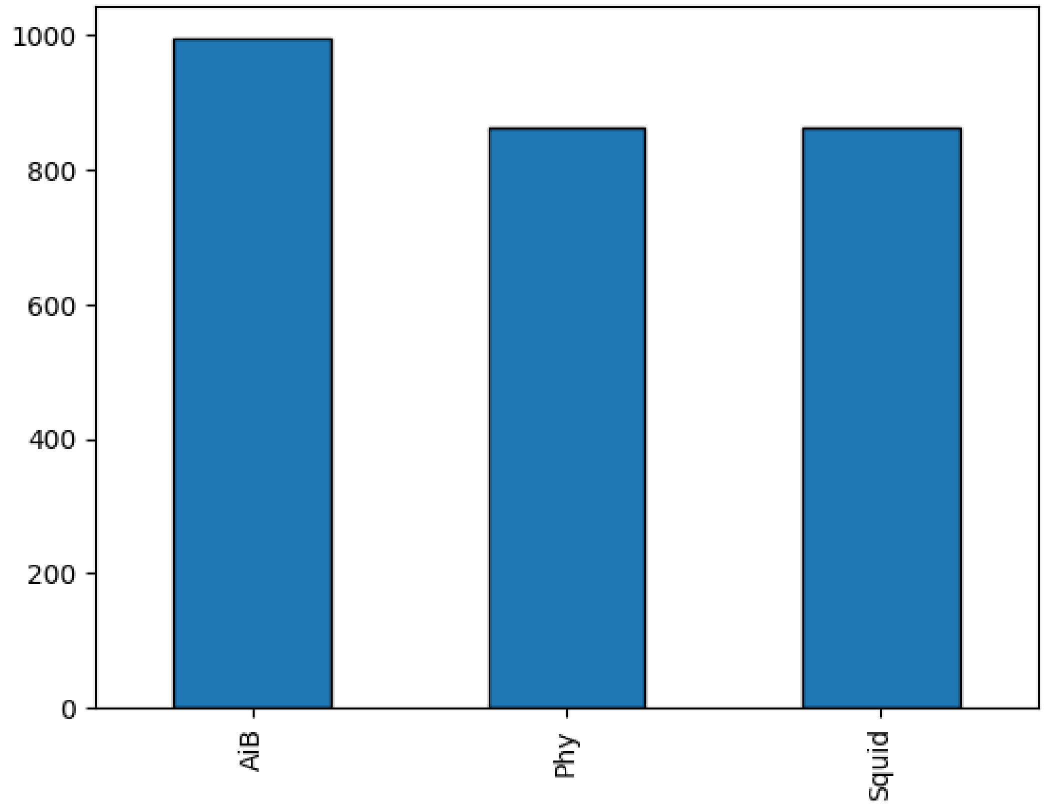
```
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->t
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.9/dist-packages (from werkzeug>=1.0.1->tensorboa
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=4.4->markdown>
```

Preparing the Data

Distribution analysis, preprocessing

```
# Create df from each csv file then concat together
alice_url = 'https://drive.google.com/file/d/1OHA3QLfzCPWq1uRnNzzukeZBkW-rsJox/view?usp=sharing'
alice_path = 'https://drive.google.com/uc?export=download&id='+alice_url.split('/')[-2]
alice_df = pd.read_csv(alice_path)
alice_df['title'] = 'AiB'

phy_url = 'https://drive.google.com/file/d/1qhrhsl1caJIesMiTcEvO8vSFMGMCOQeJ/view?usp=sharing'
phy_path = 'https://drive.google.com/uc?export=download&id='+phy_url.split('/')[-2]
phy_df = pd.read_csv(phy_path)
phy_df['title'] = 'Phy'

squid_url = 'https://drive.google.com/file/d/1puXcSc-IJ1EUeUR1_IsqADD_Tt67SfZx/view?usp=sharing'
squid_path = 'https://drive.google.com/uc?export=download&id='+squid_url.split('/')[-2]
squid_df = pd.read_csv(squid_path)
squid_df['title'] = 'Squid'

df = pd.concat([alice_df, phy_df, squid_df])

# Display distribution of target classes
df['title'].value_counts().plot(kind='bar', edgecolor='black')

# Remove stop words from data frame
stop = stopwords.words('english')
df['headlines'] = df['headlines'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))

# Split into training and testing data
i = np.random.rand(len(df)) < 0.8
```

```
train = df[i]
test = df[~i]
```



Sequential Modeling

The simplest type of model that is a linear stack of layers. Keras will be used to implement the model.

```
# Tokenizer vectorizes a text corpus by some specified measure; tf-idf is chosen
max_vocab = 25000
batch_size = 250
tokenizer = Tokenizer(num_words=max_vocab)
tokenizer.fit_on_texts(train.headlines)

X_train = tokenizer.texts_to_matrix(train.headlines, mode='tfidf')
X_test = tokenizer.texts_to_matrix(test.headlines, mode='tfidf')
```

```
# Convert target classes to numerical value
encoder = LabelEncoder()
encoder.fit(train.title)
# Fit the tokenizer
y_train = encoder.transform(train.title)
y_test = encoder.transform(test.title)

# One-hot encode target classes
one_hot_y_train = np_utils.to_categorical(y_train)
one_hot_y_test = np_utils.to_categorical(y_test)

# Fit the model
model = models.Sequential()
model.add(layers.Dense(8, input_dim=max_vocab, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(16, input_dim=max_vocab, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(24, input_dim=max_vocab, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(3, kernel_initializer='normal', activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, one_hot_y_train, batch_size=batch_size, epochs=15, verbose=1, validation_split=0.1)

# Evaluate the model
score = model.evaluate(X_test, one_hot_y_test, batch_size=batch_size, verbose=1)
print('Accuracy:', score[1])
```

```
Epoch 1/15
195/195 [==============================] - 3s 8ms/step - loss: 1.0404 - accuracy: 0.4415 - val_loss: 1.3508 - val_accuracy: 0
Epoch 2/15
195/195 [==============================] - 1s 7ms/step - loss: 0.6556 - accuracy: 0.6971 - val_loss: 1.4271 - val_accuracy: 0
Epoch 3/15
195/195 [==============================] - 1s 6ms/step - loss: 0.4414 - accuracy: 0.7444 - val_loss: 1.2889 - val_accuracy: 0
Epoch 4/15
195/195 [==============================] - 1s 7ms/step - loss: 0.2876 - accuracy: 0.8881 - val_loss: 1.0139 - val_accuracy: 0
Epoch 5/15
195/195 [==============================] - 2s 11ms/step - loss: 0.1554 - accuracy: 0.9584 - val_loss: 1.0383 - val_accuracy: 0
Epoch 6/15
195/195 [==============================] - 2s 13ms/step - loss: 0.0786 - accuracy: 0.9795 - val_loss: 1.5226 - val_accuracy: 0
Epoch 7/15
195/195 [==============================] - 1s 7ms/step - loss: 0.0505 - accuracy: 0.9882 - val_loss: 1.3563 - val_accuracy: 0
Epoch 8/15
195/195 [==============================] - 1s 6ms/step - loss: 0.0378 - accuracy: 0.9882 - val_loss: 1.6181 - val_accuracy: 0
```

```
Epoch 9/15
195/195 [==============================] - 1s 6ms/step - loss: 0.0315 - accuracy: 0.9902 - val_loss: 1.6821 - val_accuracy: 0
Epoch 10/15
195/195 [==============================] - 1s 6ms/step - loss: 0.0272 - accuracy: 0.9902 - val_loss: 2.0473 - val_accuracy: 0
Epoch 11/15
195/195 [==============================] - 1s 6ms/step - loss: 0.0218 - accuracy: 0.9933 - val_loss: 1.8574 - val_accuracy: 0
Epoch 12/15
195/195 [==============================] - 2s 8ms/step - loss: 0.0186 - accuracy: 0.9933 - val_loss: 2.3449 - val_accuracy: 0
Epoch 13/15
195/195 [==============================] - 2s 9ms/step - loss: 0.0179 - accuracy: 0.9928 - val_loss: 2.2351 - val_accuracy: 0
Epoch 14/15
195/195 [==============================] - 2s 12ms/step - loss: 0.0144 - accuracy: 0.9944 - val_loss: 1.7834 - val_accuracy: (
Epoch 15/15
195/195 [==============================] - 1s 7ms/step - loss: 0.0136 - accuracy: 0.9944 - val_loss: 2.2774 - val_accuracy: 0
56/56 [==============================] - 0s 3ms/step - loss: 1.6267 - accuracy: 0.7514
Accuracy: 0.7513513565063477
```

I had tried to make a couple of modifications to achieve better accuracy. But no matter what it was consistently around 42% accurate with the validation data. I tried to modify the number of layers as well the units within each. I increased and decreased the number of epochs. I kept the activation functions as ReLu, the output function as sigmoid, and loss function as binary crossentropy because these are optimal for binary classification.

Later I came back to revisit this because I realized my problem was in fact multi-class classification. I found that using one-hot encoding with categorical_crossentropy and softmax output greatly increased my accuracy to 77%.

Recurrent Neural Network

They are good for sequential data such as texts, so we will give it a try. I specifically chose gated recurrent unit (GRU) because it has better performance than the SimpleRNN but does not take as long as the LTSM variation.

```
maxlen = 500
max_features = 10000
max_vocab = 25000 ###
batch_size = 250 ###

# Sequentialize features
tokenizer = Tokenizer(num_words=max_vocab)
```

```
tokenizer.fit_on_texts(df['headlines'].values)
X = tokenizer.texts_to_sequences(df['headlines'].values)
X = preprocessing.sequence.pad_sequences(X, maxlen=maxlen)

# Convert target labels
encoder = LabelEncoder()
Y = encoder.fit(df['title'].values)
Y = encoder.transform(df['title'].values)
Y = np_utils.to_categorical(Y)

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42)

# Build model
model = models.Sequential()
model.add(layers.Embedding(max_vocab, 100, input_length=X.shape[1]))
model.add(layers.SpatialDropout1D(0.2))
model.add(layers.GRU(100, dropout=0.2, recurrent_dropout=0.2))
model.add(layers.Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train, Y_train, epochs=8, batch_size=batch_size, validation_split=0.2)

# Evaluate model
accuracy = model.evaluate(X_test,Y_test)
print('Accuracy:', accuracy[1])
```

```
(2720, 3)
Epoch 1/8
7/7 [==============================] - 36s 5s/step - loss: 1.0929 - accuracy: 0.3805 - val_loss: 1.0872 - val_accuracy: 0.3394
Epoch 2/8
7/7 [==============================] - 31s 4s/step - loss: 1.0634 - accuracy: 0.3885 - val_loss: 1.0665 - val_accuracy: 0.3417
Epoch 3/8
7/7 [==============================] - 34s 5s/step - loss: 1.0178 - accuracy: 0.4092 - val_loss: 1.0222 - val_accuracy: 0.4128
Epoch 4/8
7/7 [==============================] - 32s 5s/step - loss: 0.9206 - accuracy: 0.6684 - val_loss: 0.9231 - val_accuracy: 0.6697
Epoch 5/8
7/7 [==============================] - 33s 5s/step - loss: 0.7408 - accuracy: 0.8155 - val_loss: 0.7589 - val_accuracy: 0.6950
Epoch 6/8
7/7 [==============================] - 39s 6s/step - loss: 0.5422 - accuracy: 0.8414 - val_loss: 0.6491 - val_accuracy: 0.7385
Epoch 7/8
7/7 [==============================] - 34s 5s/step - loss: 0.3766 - accuracy: 0.8989 - val_loss: 0.5248 - val_accuracy: 0.7936
```

```
Epoch 8/8
7/7 [==============================] - 33s 5s/step - loss: 0.2569 - accuracy: 0.9270 - val_loss: 0.4843 - val_accuracy: 0.8028
17/17 [==============================] - 2s 135ms/step - loss: 0.5701 - accuracy: 0.7500
Accuracy: 0.75
```

While the performance of the Recurrent Neural Network GRU model is very comparable to that of the basic Sequential model; the achieved the same validation set accuracy. I had tried different setups and hyperparameters (epochs, dropout) and this is the best I could achieve.

BERT Embedding

An embedding is a fixed-length vector. It is a more compact way to represent text in comparison to binary or word-count matrices. BERT stands for Bidirectional Encoder Representations from Transformers. It can embed words in sequential context and predict masked words.

```
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'
tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1'
bert_preprocess_model = hub.KerasLayer(tfhub_handle_preprocess)
bert_model = hub.KerasLayer(tfhub_handle_encoder)

def build_model():
  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
  encoder_inputs = preprocessing_layer(text_input)
  encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
  outputs = encoder(encoder_inputs)
  net = outputs['pooled_output']
  net = tf.keras.layers.Dropout(0.1)(net)
  net = tf.keras.layers.Dense(3, activation='softmax', name='classifier')(net)
  return tf.keras.Model(text_input, net)

# Prepare data
X = df['headlines'].values
Y = df['title'].values
encoder = LabelEncoder()
Y = encoder.fit(df['title'].values)
Y = encoder.transform(df['title'].values)
Y = np_utils.to_categorical(Y)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42)

# Build model
model = build_model()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=15, batch_size=batch_size, validation_split=0.2)

# Evaluate model
accuracy = model.evaluate(X_test,Y_test)
print('Accuracy:', accuracy[1])
```

```
Epoch 1/15
7/7 [==============================] - 42s 5s/step - loss: 1.2038 - accuracy: 0.3885 - val_loss: 0.9718 - val_accuracy: 0.5940
Epoch 2/15
7/7 [==============================] - 34s 5s/step - loss: 0.9164 - accuracy: 0.5718 - val_loss: 0.6411 - val_accuracy: 0.7408
Epoch 3/15
7/7 [==============================] - 32s 5s/step - loss: 0.5767 - accuracy: 0.7638 - val_loss: 0.4737 - val_accuracy: 0.8056
Epoch 4/15
7/7 [==============================] - 34s 5s/step - loss: 0.3458 - accuracy: 0.8609 - val_loss: 0.4967 - val_accuracy: 0.8188
Epoch 5/15
7/7 [==============================] - 33s 5s/step - loss: 0.2198 - accuracy: 0.9247 - val_loss: 0.5402 - val_accuracy: 0.8188
Epoch 6/15
7/7 [==============================] - 33s 5s/step - loss: 0.1314 - accuracy: 0.9563 - val_loss: 0.5968 - val_accuracy: 0.8188
Epoch 7/15
7/7 [==============================] - 36s 5s/step - loss: 0.0830 - accuracy: 0.9747 - val_loss: 0.6777 - val_accuracy: 0.8257
Epoch 8/15
7/7 [==============================] - 33s 5s/step - loss: 0.0616 - accuracy: 0.9845 - val_loss: 0.7181 - val_accuracy: 0.8142
Epoch 9/15
7/7 [==============================] - 34s 5s/step - loss: 0.0585 - accuracy: 0.9787 - val_loss: 0.8019 - val_accuracy: 0.8073
Epoch 10/15
7/7 [==============================] - 32s 5s/step - loss: 0.0527 - accuracy: 0.9845 - val_loss: 0.8080 - val_accuracy: 0.8119
Epoch 11/15
7/7 [==============================] - 33s 5s/step - loss: 0.0338 - accuracy: 0.9902 - val_loss: 0.8580 - val_accuracy: 0.8119
Epoch 12/15
7/7 [==============================] - 33s 5s/step - loss: 0.0393 - accuracy: 0.9885 - val_loss: 0.8284 - val_accuracy: 0.8234
Epoch 13/15
7/7 [==============================] - 33s 5s/step - loss: 0.0216 - accuracy: 0.9920 - val_loss: 0.8590 - val_accuracy: 0.8326
Epoch 14/15
7/7 [==============================] - 33s 5s/step - loss: 0.0260 - accuracy: 0.9920 - val_loss: 0.8340 - val_accuracy: 0.8211
Epoch 15/15
7/7 [==============================] - 32s 5s/step - loss: 0.0160 - accuracy: 0.9948 - val_loss: 0.8679 - val_accuracy: 0.8211
```

```
17/17 [==============================] - 3s 181ms/step - loss: 1.1107 - accuracy: 0.7941
Accuracy: 0.7941176295280457
```

The BERT embedding seemed to have a pretty good impact on validating accuracy. The percentage increased by about 4%. The time it takes to train is only slightly longer than GRU.

Evaluation of Various Approaches

From these trials it seems that all of them have similar performance. Granted, more fine tuning might reveal that certain models are stronger than others *for different applications*. The simple Sequential model was the quickest to train. While the computing requirements for GRU and BERT are similar, BERT provides tooling (preprocessor, encoder) that were otherwise done manually with GRU. BERT also performed slightly better than Sequential and GRU in my trials.