# Text Classification

Using neural networks, we will classify pieces of text according to the emotions they invoke. The dataset can be obtained here. The data consists of two columns: the first column is the text, and the second column is a manually annotated emotion associated with each text. We will make models that take the text as input and predict what category of emotion that text should be associated with.

## Data Exploration

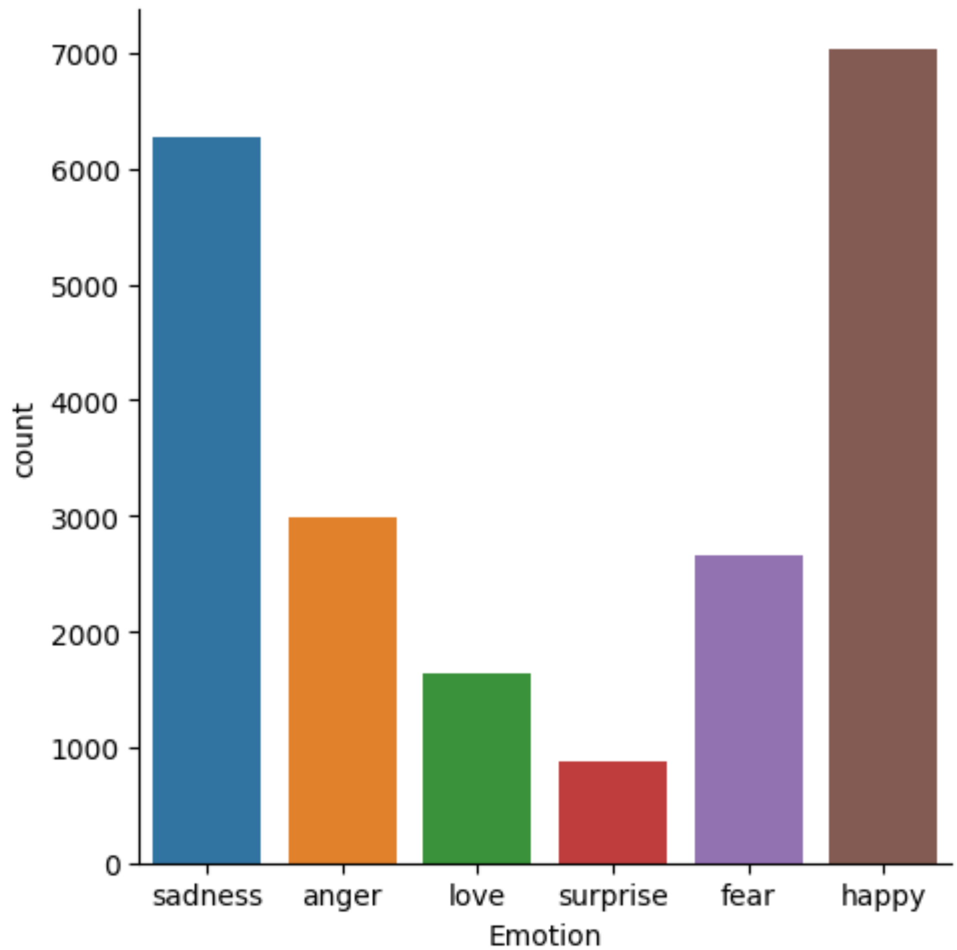
First, we'll read in the data.

In [ ]:
```
import pandas as pd

df = pd.read_csv('data/Emotion_final.csv')
df.dropna() # drop rows with missing data
df['EmotionAsFactor'] = pd.factorize(df.Emotion)[0] # convert Emotion column to int
```

We'll make a frequency graph of all the classes.

In [ ]:
```
import seaborn as sb

sb.catplot(x='Emotion', kind='count', data=df[['Emotion']])
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x22d8b39ac50>
```




The data is slightly unbalanced in favor of happiness and sadness. Fortunately, the other categories are represented enough that any model cannot achieve an accuracy of over 70% by exclusively guessing those two categories.

## Model Training

First, we will divide the data into train and test sets. Every model we create will use the same label data, so we also go ahead and encode the target columns in both sets. The input data will have to be separately encoded depending on the type of model being created.

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
import numpy as np

# setting random seeds
tf.keras.utils.set_random_seed(1234)
np.random.seed(1234)

# train/test split
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]

num_classes = df.EmotionAsFactor.nunique()
vocab_size = 20000

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Text)

# encode label column
encoder = LabelEncoder()
encoder.fit(train.EmotionAsFactor)
y_train = encoder.transform(train.EmotionAsFactor)
y_test = encoder.transform(test.EmotionAsFactor)
```

## Sequential Neural Network

We will make a dense sequential neural network and evaluate it on the test data. For this model, the input data is transformed to matrices according to the tf-idf frequency measure.

In [ ]:

```
from tensorflow.keras import layers, models
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

batch_size = 128
num_epochs = 30

# transform input
x_train = tokenizer.texts_to_matrix(train.Text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Text, mode='tfidf')

# define model topology
model_seq = models.Sequential()
model_seq.add(layers.Dense(16, kernel_initializer='normal', activation='sigmoid'))
model_seq.add(layers.Dense(16, kernel_initializer='normal', activation='relu'))
model_seq.add(layers.Dense(num_classes, input_dim=vocab_size, kernel_initializer='n

# train
model_seq.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['sparse_categorical_accuracy']
)

# apply to test data
model_seq.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2
)

pred_seq = model_seq.predict(x_test) # get predictions as label probabilities
pred_seq = np.argmax(pred_seq, axis=1) # get most likely label from probabilities

print('\naccuracy: ', accuracy_score(y_test, pred_seq))
```

```
Epoch 1/30
108/108 [==============================] - 3s 20ms/step - loss: 1.6612 - sparse_cat
egorical_accuracy: 0.3303 - val_loss: 1.6081 - val_sparse_categorical_accuracy: 0.2
958
Epoch 2/30
108/108 [==============================] - 2s 16ms/step - loss: 1.5411 - sparse_cat
egorical_accuracy: 0.3395 - val_loss: 1.5832 - val_sparse_categorical_accuracy: 0.2
958
Epoch 3/30
108/108 [==============================] - 2s 16ms/step - loss: 1.4608 - sparse_cat
egorical_accuracy: 0.3395 - val_loss: 1.5434 - val_sparse_categorical_accuracy: 0.2
964
Epoch 4/30
108/108 [==============================] - 2s 16ms/step - loss: 1.3499 - sparse_cat
egorical_accuracy: 0.3461 - val_loss: 1.4970 - val_sparse_categorical_accuracy: 0.3
074
Epoch 5/30
108/108 [==============================] - 2s 16ms/step - loss: 1.2130 - sparse_cat
egorical_accuracy: 0.4070 - val_loss: 1.4270 - val_sparse_categorical_accuracy: 0.4
006
Epoch 6/30
108/108 [==============================] - 2s 15ms/step - loss: 1.0520 - sparse_cat
egorical_accuracy: 0.5453 - val_loss: 1.3698 - val_sparse_categorical_accuracy: 0.5
272
Epoch 7/30
108/108 [==============================] - 2s 16ms/step - loss: 0.8561 - sparse_cat
egorical_accuracy: 0.6945 - val_loss: 1.2697 - val_sparse_categorical_accuracy: 0.5
683
Epoch 8/30
108/108 [==============================] - 2s 16ms/step - loss: 0.6502 - sparse_cat
egorical_accuracy: 0.7673 - val_loss: 1.2006 - val_sparse_categorical_accuracy: 0.6
274
Epoch 9/30
108/108 [==============================] - 2s 15ms/step - loss: 0.4981 - sparse_cat
egorical_accuracy: 0.8535 - val_loss: 1.1787 - val_sparse_categorical_accuracy: 0.6
687
Epoch 10/30
108/108 [==============================] - 2s 16ms/step - loss: 0.3860 - sparse_cat
egorical_accuracy: 0.8975 - val_loss: 1.1717 - val_sparse_categorical_accuracy: 0.6
801
Epoch 11/30
108/108 [==============================] - 2s 16ms/step - loss: 0.2987 - sparse_cat
egorical_accuracy: 0.9174 - val_loss: 1.1794 - val_sparse_categorical_accuracy: 0.6
891
Epoch 12/30
108/108 [==============================] - 2s 23ms/step - loss: 0.2356 - sparse_cat
egorical_accuracy: 0.9253 - val_loss: 1.1846 - val_sparse_categorical_accuracy: 0.6
964
Epoch 13/30
108/108 [==============================] - 2s 19ms/step - loss: 0.1918 - sparse_cat
egorical_accuracy: 0.9343 - val_loss: 1.1989 - val_sparse_categorical_accuracy: 0.6
961
Epoch 14/30
108/108 [==============================] - 2s 16ms/step - loss: 0.1592 - sparse_cat
egorical_accuracy: 0.9515 - val_loss: 1.2229 - val_sparse_categorical_accuracy: 0.7
031
```

```
Epoch 15/30
108/108 [==============================] - 2s 17ms/step - loss: 0.1332 - sparse_cat
egorical_accuracy: 0.9650 - val_loss: 1.2542 - val_sparse_categorical_accuracy: 0.7
112
Epoch 16/30
108/108 [==============================] - 2s 16ms/step - loss: 0.1126 - sparse_cat
egorical_accuracy: 0.9720 - val_loss: 1.2978 - val_sparse_categorical_accuracy: 0.7
100
Epoch 17/30
108/108 [==============================] - 2s 17ms/step - loss: 0.0965 - sparse_cat
egorical_accuracy: 0.9762 - val_loss: 1.2987 - val_sparse_categorical_accuracy: 0.7
095
Epoch 18/30
108/108 [==============================] - 2s 17ms/step - loss: 0.0836 - sparse_cat
egorical_accuracy: 0.9792 - val_loss: 1.3139 - val_sparse_categorical_accuracy: 0.7
135
Epoch 19/30
108/108 [==============================] - 2s 17ms/step - loss: 0.0737 - sparse_cat
egorical_accuracy: 0.9810 - val_loss: 1.3397 - val_sparse_categorical_accuracy: 0.7
138
Epoch 20/30
108/108 [==============================] - 2s 16ms/step - loss: 0.0656 - sparse_cat
egorical_accuracy: 0.9832 - val_loss: 1.3787 - val_sparse_categorical_accuracy: 0.7
112
Epoch 21/30
108/108 [==============================] - 2s 17ms/step - loss: 0.0590 - sparse_cat
egorical_accuracy: 0.9849 - val_loss: 1.4459 - val_sparse_categorical_accuracy: 0.7
112
Epoch 22/30
108/108 [==============================] - 2s 18ms/step - loss: 0.0532 - sparse_cat
egorical_accuracy: 0.9856 - val_loss: 1.4692 - val_sparse_categorical_accuracy: 0.7
135
Epoch 23/30
108/108 [==============================] - 2s 18ms/step - loss: 0.0489 - sparse_cat
egorical_accuracy: 0.9861 - val_loss: 1.5086 - val_sparse_categorical_accuracy: 0.7
135
Epoch 24/30
108/108 [==============================] - 2s 16ms/step - loss: 0.0441 - sparse_cat
egorical_accuracy: 0.9874 - val_loss: 1.5271 - val_sparse_categorical_accuracy: 0.7
045
Epoch 25/30
108/108 [==============================] - 2s 16ms/step - loss: 0.0403 - sparse_cat
egorical_accuracy: 0.9885 - val_loss: 1.5533 - val_sparse_categorical_accuracy: 0.7
103
Epoch 26/30
108/108 [==============================] - 2s 15ms/step - loss: 0.0371 - sparse_cat
egorical_accuracy: 0.9895 - val_loss: 1.5524 - val_sparse_categorical_accuracy: 0.7
068
Epoch 27/30
108/108 [==============================] - 2s 15ms/step - loss: 0.0339 - sparse_cat
egorical_accuracy: 0.9902 - val_loss: 1.6183 - val_sparse_categorical_accuracy: 0.7
048
Epoch 28/30
108/108 [==============================] - 2s 15ms/step - loss: 0.0313 - sparse_cat
egorical_accuracy: 0.9907 - val_loss: 1.6403 - val_sparse_categorical_accuracy: 0.7
086
```

```
Epoch 29/30
108/108 [==============================] - 2s 15ms/step - loss: 0.0288 - sparse_cat
egorical_accuracy: 0.9909 - val_loss: 1.6565 - val_sparse_categorical_accuracy: 0.7
080
Epoch 30/30
108/108 [==============================] - 2s 15ms/step - loss: 0.0270 - sparse_cat
egorical_accuracy: 0.9915 - val_loss: 1.7001 - val_sparse_categorical_accuracy: 0.7
039
134/134 [==============================] - 1s 4ms/step

accuracy:  0.8332555970149254
```

We get a fairly good accuracy of just over 80% with this simple network. This is fairly promising.

Also note that we used the softmax activation function in the last layer of the model, and the number of output nodes is equal to the number of classes. This has to be done to make the network topology compatible with multiclass classification, and the same will be done for all future models. For similar reasons, 'sparse_categorical_crossentropy' must be the loss function used during each model's compilation step.

## Convolutional Neural Network

We will now create a convolutional network and evaluate it. For this model, the inputs are transformed to sequences with an arbitrary maximum length of 500. The number of epochs has been reduced in the interest of training the model more quickly.

In [ ]:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len = 500
batch_size = 128
num_epochs = 20

x_train = pad_sequences(tokenizer.texts_to_sequences(train.Text), maxlen=max_len)
x_test  = pad_sequences(tokenizer.texts_to_sequences(test.Text), maxlen=max_len)

model_cnn = models.Sequential()
model_cnn.add(layers.Embedding(vocab_size, 128, input_length=max_len))
model_cnn.add(layers.Conv1D(32, 7, activation='relu'))
model_cnn.add(layers.MaxPooling1D(5))
model_cnn.add(layers.Conv1D(32, 7, activation='relu'))
model_cnn.add(layers.GlobalMaxPooling1D())
model_cnn.add(layers.Dense(num_classes, activation='softmax'))

model_cnn.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['sparse_categorical_accuracy']
)

model_cnn.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2
)

pred_cnn = model_cnn.predict(x_test)
pred_cnn = np.argmax(pred_cnn, axis=1)

print('\naccuracy: ', accuracy_score(y_test, pred_cnn))
```

```
Epoch 1/20
108/108 [==============================] - 31s 260ms/step - loss: 1.5820 - sparse_c
ategorical_accuracy: 0.3342 - val_loss: 1.6207 - val_sparse_categorical_accuracy:
0.3071
Epoch 2/20
108/108 [==============================] - 31s 287ms/step - loss: 1.5690 - sparse_c
ategorical_accuracy: 0.3438 - val_loss: 1.6353 - val_sparse_categorical_accuracy:
0.2958
Epoch 3/20
108/108 [==============================] - 34s 311ms/step - loss: 1.5508 - sparse_c
ategorical_accuracy: 0.3613 - val_loss: 1.6015 - val_sparse_categorical_accuracy:
0.3493
Epoch 4/20
108/108 [==============================] - 34s 315ms/step - loss: 1.4283 - sparse_c
ategorical_accuracy: 0.4512 - val_loss: 1.4726 - val_sparse_categorical_accuracy:
0.4224
Epoch 5/20
108/108 [==============================] - 33s 304ms/step - loss: 1.1790 - sparse_c
ategorical_accuracy: 0.5614 - val_loss: 1.3101 - val_sparse_categorical_accuracy:
0.5013
Epoch 6/20
108/108 [==============================] - 28s 262ms/step - loss: 0.9636 - sparse_c
ategorical_accuracy: 0.6402 - val_loss: 1.2338 - val_sparse_categorical_accuracy:
0.5418
Epoch 7/20
108/108 [==============================] - 28s 256ms/step - loss: 0.7839 - sparse_c
ategorical_accuracy: 0.7085 - val_loss: 1.1907 - val_sparse_categorical_accuracy:
0.5552
Epoch 8/20
108/108 [==============================] - 28s 262ms/step - loss: 0.6590 - sparse_c
ategorical_accuracy: 0.7592 - val_loss: 1.1912 - val_sparse_categorical_accuracy:
0.5665
Epoch 9/20
108/108 [==============================] - 28s 261ms/step - loss: 0.5706 - sparse_c
ategorical_accuracy: 0.7958 - val_loss: 1.2532 - val_sparse_categorical_accuracy:
0.5738
Epoch 10/20
108/108 [==============================] - 29s 265ms/step - loss: 0.5071 - sparse_c
ategorical_accuracy: 0.8195 - val_loss: 1.2710 - val_sparse_categorical_accuracy:
0.5825
Epoch 11/20
108/108 [==============================] - 29s 273ms/step - loss: 0.4557 - sparse_c
ategorical_accuracy: 0.8402 - val_loss: 1.3253 - val_sparse_categorical_accuracy:
0.5825
Epoch 12/20
108/108 [==============================] - 37s 346ms/step - loss: 0.4117 - sparse_c
ategorical_accuracy: 0.8540 - val_loss: 1.3622 - val_sparse_categorical_accuracy:
0.5805
Epoch 13/20
108/108 [==============================] - 38s 352ms/step - loss: 0.3715 - sparse_c
ategorical_accuracy: 0.8668 - val_loss: 1.4371 - val_sparse_categorical_accuracy:
0.5793
Epoch 14/20
108/108 [==============================] - 38s 348ms/step - loss: 0.3420 - sparse_c
ategorical_accuracy: 0.8749 - val_loss: 1.6312 - val_sparse_categorical_accuracy:
0.5776
```

```
Epoch 15/20
108/108 [==============================] - 38s 349ms/step - loss: 0.3153 - sparse_c
ategorical_accuracy: 0.8852 - val_loss: 1.6846 - val_sparse_categorical_accuracy:
0.5718
Epoch 16/20
108/108 [==============================] - 38s 349ms/step - loss: 0.2939 - sparse_c
ategorical_accuracy: 0.8907 - val_loss: 1.6600 - val_sparse_categorical_accuracy:
0.5726
Epoch 17/20
108/108 [==============================] - 38s 347ms/step - loss: 0.2750 - sparse_c
ategorical_accuracy: 0.8964 - val_loss: 1.7408 - val_sparse_categorical_accuracy:
0.5761
Epoch 18/20
108/108 [==============================] - 37s 347ms/step - loss: 0.2585 - sparse_c
ategorical_accuracy: 0.9016 - val_loss: 1.8199 - val_sparse_categorical_accuracy:
0.5654
Epoch 19/20
108/108 [==============================] - 38s 351ms/step - loss: 0.2460 - sparse_c
ategorical_accuracy: 0.9055 - val_loss: 1.8998 - val_sparse_categorical_accuracy:
0.5575
Epoch 20/20
108/108 [==============================] - 35s 327ms/step - loss: 0.2356 - sparse_c
ategorical_accuracy: 0.9080 - val_loss: 1.9785 - val_sparse_categorical_accuracy:
0.5706
134/134 [==============================] - 2s 16ms/step

accuracy:  0.6450559701492538
```

The accuracy of this CNN is unfortunately much lower than our original sequential network. The accuracy seems to have been increasing at a steady rate between all the epochs, so it is likely that more epochs or a greater batch size would result in a more accurate model.

## Embeddings

Now we will try different embedding schemes to try to improve our results. For these, we will not have to transform our existing sequence representations of the input data.

First, we'll try using pretrained embeddings from GloVe. Here we will use the 100 dimension embeddings.

In [ ]:
```
embedding_dim = 100

embeddings_index = {}
with open('data/glove.6B/glove.6B.100d.txt', 'r', 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

embedding_matrix = np.zeros((vocab_size, 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
```

In [ ]:
```
batch_size = 128
num_epochs = 20

model_em1 = models.Sequential()
model_em1.add(layers.Embedding(
    vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix),
    trainable=False
))
model_em1.add(layers.Conv1D(128, 5, activation='relu'))
model_em1.add(layers.MaxPooling1D(5))
model_em1.add(layers.Conv1D(128, 5, activation='relu'))
model_em1.add(layers.MaxPooling1D(5))
model_em1.add(layers.GlobalMaxPooling1D())
model_em1.add(layers.Dropout(0.5))
model_em1.add(layers.Dense(128, activation="relu"))
model_em1.add(layers.Dense(num_classes, activation='softmax'))

model_em1.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['sparse_categorical_accuracy']
)

model_em1.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2
)

pred_em1 = model_em1.predict(x_test)
pred_em1 = np.argmax(pred_em1, axis=1)

print('\naccuracy: ', accuracy_score(y_test, pred_em1))
```

```
Epoch 1/20
108/108 [==============================] - 35s 313ms/step - loss: 1.5724 - sparse_c
ategorical_accuracy: 0.3604 - val_loss: 1.5046 - val_sparse_categorical_accuracy:
0.4279
Epoch 2/20
108/108 [==============================] - 33s 305ms/step - loss: 1.3523 - sparse_c
ategorical_accuracy: 0.4895 - val_loss: 1.3443 - val_sparse_categorical_accuracy:
0.4836
Epoch 3/20
108/108 [==============================] - 39s 360ms/step - loss: 1.1245 - sparse_c
ategorical_accuracy: 0.5918 - val_loss: 1.2351 - val_sparse_categorical_accuracy:
0.5313
Epoch 4/20
108/108 [==============================] - 63s 584ms/step - loss: 0.9499 - sparse_c
ategorical_accuracy: 0.6553 - val_loss: 1.0616 - val_sparse_categorical_accuracy:
0.6015
Epoch 5/20
108/108 [==============================] - 52s 481ms/step - loss: 0.8097 - sparse_c
ategorical_accuracy: 0.7046 - val_loss: 1.1271 - val_sparse_categorical_accuracy:
0.5936
Epoch 6/20
108/108 [==============================] - 39s 364ms/step - loss: 0.7187 - sparse_c
ategorical_accuracy: 0.7394 - val_loss: 1.0057 - val_sparse_categorical_accuracy:
0.6303
Epoch 7/20
108/108 [==============================] - 43s 400ms/step - loss: 0.6236 - sparse_c
ategorical_accuracy: 0.7721 - val_loss: 0.9603 - val_sparse_categorical_accuracy:
0.6472
Epoch 8/20
108/108 [==============================] - 44s 411ms/step - loss: 0.5581 - sparse_c
ategorical_accuracy: 0.7980 - val_loss: 1.2342 - val_sparse_categorical_accuracy:
0.5555
Epoch 9/20
108/108 [==============================] - 39s 364ms/step - loss: 0.5054 - sparse_c
ategorical_accuracy: 0.8179 - val_loss: 1.0519 - val_sparse_categorical_accuracy:
0.6282
Epoch 10/20
108/108 [==============================] - 47s 438ms/step - loss: 0.4555 - sparse_c
ategorical_accuracy: 0.8339 - val_loss: 1.0877 - val_sparse_categorical_accuracy:
0.6512
Epoch 11/20
108/108 [==============================] - 52s 483ms/step - loss: 0.4117 - sparse_c
ategorical_accuracy: 0.8510 - val_loss: 0.9938 - val_sparse_categorical_accuracy:
0.6777
Epoch 12/20
108/108 [==============================] - 54s 501ms/step - loss: 0.3854 - sparse_c
ategorical_accuracy: 0.8571 - val_loss: 1.0502 - val_sparse_categorical_accuracy:
0.6719
Epoch 13/20
108/108 [==============================] - 55s 505ms/step - loss: 0.3569 - sparse_c
ategorical_accuracy: 0.8698 - val_loss: 1.0568 - val_sparse_categorical_accuracy:
0.6798
Epoch 14/20
108/108 [==============================] - 54s 501ms/step - loss: 0.3306 - sparse_c
ategorical_accuracy: 0.8768 - val_loss: 1.2551 - val_sparse_categorical_accuracy:
0.6518
```

```
Epoch 15/20
108/108 [==============================] - 54s 501ms/step - loss: 0.3112 - sparse_c
ategorical_accuracy: 0.8871 - val_loss: 1.1873 - val_sparse_categorical_accuracy:
0.6754
Epoch 16/20
108/108 [==============================] - 54s 497ms/step - loss: 0.2987 - sparse_c
ategorical_accuracy: 0.8931 - val_loss: 1.2479 - val_sparse_categorical_accuracy:
0.6501
Epoch 17/20
108/108 [==============================] - 54s 504ms/step - loss: 0.2839 - sparse_c
ategorical_accuracy: 0.8984 - val_loss: 2.0406 - val_sparse_categorical_accuracy:
0.5916
Epoch 18/20
108/108 [==============================] - 53s 495ms/step - loss: 0.2665 - sparse_c
ategorical_accuracy: 0.9038 - val_loss: 1.2636 - val_sparse_categorical_accuracy:
0.6614
Epoch 19/20
108/108 [==============================] - 53s 493ms/step - loss: 0.2519 - sparse_c
ategorical_accuracy: 0.9067 - val_loss: 1.3850 - val_sparse_categorical_accuracy:
0.6640
Epoch 20/20
108/108 [==============================] - 44s 406ms/step - loss: 0.2464 - sparse_c
ategorical_accuracy: 0.9103 - val_loss: 1.2977 - val_sparse_categorical_accuracy:
0.6667
134/134 [==============================] - 5s 35ms/step

accuracy:  0.7437033582089553
```

This model is a clear improvement over the first CNN. This can potentially be attributed to both the more complex network topology as well as the use of the pretrained embeddings from GloVe.

We will now see if using one of GloVe's higher dimension embeddings can improve our results. We will use the 300 dimension version.

In [ ]:
```
embedding_dim = 300

embeddings_index = {}
with open('data/glove.6B/glove.6B.300d.txt', 'r', 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

embedding_matrix = np.zeros((vocab_size, 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
```

In [ ]:

```
batch_size = 128
num_epochs = 20

model_em2 = models.Sequential()
model_em2.add(layers.Embedding(
    vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix),
    trainable=False
))
model_em2.add(layers.Conv1D(128, 5, activation='relu'))
model_em2.add(layers.MaxPooling1D(5))
model_em2.add(layers.Conv1D(128, 5, activation='relu'))
model_em2.add(layers.MaxPooling1D(5))
model_em2.add(layers.GlobalMaxPooling1D())
model_em2.add(layers.Dropout(0.5))
model_em2.add(layers.Dense(128, activation="relu"))
model_em2.add(layers.Dense(num_classes, activation='softmax'))

model_em2.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['sparse_categorical_accuracy']
)

model_em2.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2
)

pred_em2 = model_em2.predict(x_test)
pred_em2 = np.argmax(pred_em2, axis=1)

print('\naccuracy: ', accuracy_score(y_test, pred_em2))
```

```
Epoch 1/20
108/108 [==============================] - 89s 800ms/step - loss: 1.5294 - sparse_c
ategorical_accuracy: 0.3947 - val_loss: 1.3807 - val_sparse_categorical_accuracy:
0.4655
Epoch 2/20
108/108 [==============================] - 95s 878ms/step - loss: 1.1716 - sparse_c
ategorical_accuracy: 0.5610 - val_loss: 1.1656 - val_sparse_categorical_accuracy:
0.5636
Epoch 3/20
108/108 [==============================] - 102s 941ms/step - loss: 0.8727 - sparse_
categorical_accuracy: 0.6883 - val_loss: 1.0776 - val_sparse_categorical_accuracy:
0.6224
Epoch 4/20
108/108 [==============================] - 102s 945ms/step - loss: 0.6758 - sparse_
categorical_accuracy: 0.7606 - val_loss: 0.9550 - val_sparse_categorical_accuracy:
0.6594
Epoch 5/20
108/108 [==============================] - 92s 853ms/step - loss: 0.5409 - sparse_c
ategorical_accuracy: 0.8063 - val_loss: 0.8576 - val_sparse_categorical_accuracy:
0.7112
Epoch 6/20
108/108 [==============================] - 84s 782ms/step - loss: 0.4466 - sparse_c
ategorical_accuracy: 0.8447 - val_loss: 1.2581 - val_sparse_categorical_accuracy:
0.6475
Epoch 7/20
108/108 [==============================] - 90s 837ms/step - loss: 0.3835 - sparse_c
ategorical_accuracy: 0.8643 - val_loss: 0.8679 - val_sparse_categorical_accuracy:
0.7083
Epoch 8/20
108/108 [==============================] - 88s 819ms/step - loss: 0.3299 - sparse_c
ategorical_accuracy: 0.8836 - val_loss: 0.9771 - val_sparse_categorical_accuracy:
0.6993
Epoch 9/20
108/108 [==============================] - 93s 863ms/step - loss: 0.2952 - sparse_c
ategorical_accuracy: 0.8929 - val_loss: 1.4610 - val_sparse_categorical_accuracy:
0.6565
Epoch 10/20
108/108 [==============================] - 97s 893ms/step - loss: 0.2805 - sparse_c
ategorical_accuracy: 0.9001 - val_loss: 1.0690 - val_sparse_categorical_accuracy:
0.7106
Epoch 11/20
108/108 [==============================] - 95s 877ms/step - loss: 0.2420 - sparse_c
ategorical_accuracy: 0.9150 - val_loss: 1.1444 - val_sparse_categorical_accuracy:
0.7057
Epoch 12/20
108/108 [==============================] - 92s 856ms/step - loss: 0.2236 - sparse_c
ategorical_accuracy: 0.9185 - val_loss: 1.5131 - val_sparse_categorical_accuracy:
0.6873
Epoch 13/20
108/108 [==============================] - 92s 849ms/step - loss: 0.2054 - sparse_c
ategorical_accuracy: 0.9261 - val_loss: 1.2661 - val_sparse_categorical_accuracy:
0.6932
Epoch 14/20
108/108 [==============================] - 93s 858ms/step - loss: 0.2096 - sparse_c
ategorical_accuracy: 0.9275 - val_loss: 1.3126 - val_sparse_categorical_accuracy:
0.7028
```

```
Epoch 15/20
108/108 [==============================] - 92s 849ms/step - loss: 0.1756 - sparse_c
ategorical_accuracy: 0.9388 - val_loss: 1.7706 - val_sparse_categorical_accuracy:
0.6771
Epoch 16/20
108/108 [==============================] - 92s 848ms/step - loss: 0.1772 - sparse_c
ategorical_accuracy: 0.9365 - val_loss: 1.4497 - val_sparse_categorical_accuracy:
0.6873
Epoch 17/20
108/108 [==============================] - 93s 858ms/step - loss: 0.1661 - sparse_c
ategorical_accuracy: 0.9423 - val_loss: 1.4122 - val_sparse_categorical_accuracy:
0.6908
Epoch 18/20
108/108 [==============================] - 89s 823ms/step - loss: 0.1610 - sparse_c
ategorical_accuracy: 0.9429 - val_loss: 1.5872 - val_sparse_categorical_accuracy:
0.6914
Epoch 19/20
108/108 [==============================] - 94s 869ms/step - loss: 0.1595 - sparse_c
ategorical_accuracy: 0.9438 - val_loss: 1.7140 - val_sparse_categorical_accuracy:
0.6862
Epoch 20/20
108/108 [==============================] - 92s 852ms/step - loss: 0.1462 - sparse_c
ategorical_accuracy: 0.9481 - val_loss: 1.6737 - val_sparse_categorical_accuracy:
0.6902
134/134 [==============================] - 12s 89ms/step
```

```
accuracy:  0.7541977611940298
```

The accuracy has slightly improved over the previous version. However, the accuracy is still less than that of the dense sequential model, indicating that the GloVe embeddings and/or CNNs are not very useful for this dataset.

We will now see if the GloVe embeddings can be used with a dense sequential network to get better results.

In [ ]:

```
batch_size = 128
num_epochs = 30

model_em3 = models.Sequential()
model_em3.add(layers.Embedding(
    vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix),
    trainable=False
))
model_em3.add(layers.GlobalMaxPooling1D())
model_em3.add(layers.Dense(32, kernel_initializer='normal', activation='relu'))
model_em3.add(layers.Dense(32, kernel_initializer='normal', activation='relu'))
model_em3.add(layers.Dense(num_classes, kernel_initializer='normal', activation='so

model_em3.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['sparse_categorical_accuracy']
)

model_em3.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2
)

pred_em3 = model_em3.predict(x_test)
pred_em3 = np.argmax(pred_em3, axis=1)

print('\naccuracy: ', accuracy_score(y_test, pred_em3))
```

```
Epoch 1/30
108/108 [==============================] - 8s 59ms/step - loss: 1.5869 - sparse_cat
egorical_accuracy: 0.3439 - val_loss: 1.6135 - val_sparse_categorical_accuracy: 0.2
958
Epoch 2/30
108/108 [==============================] - 6s 60ms/step - loss: 1.5552 - sparse_cat
egorical_accuracy: 0.3788 - val_loss: 1.6155 - val_sparse_categorical_accuracy: 0.2
993
Epoch 3/30
108/108 [==============================] - 7s 61ms/step - loss: 1.4894 - sparse_cat
egorical_accuracy: 0.4512 - val_loss: 1.5167 - val_sparse_categorical_accuracy: 0.3
991
Epoch 4/30
108/108 [==============================] - 7s 68ms/step - loss: 1.4104 - sparse_cat
egorical_accuracy: 0.4692 - val_loss: 1.5461 - val_sparse_categorical_accuracy: 0.3
857
Epoch 5/30
108/108 [==============================] - 7s 68ms/step - loss: 1.3572 - sparse_cat
egorical_accuracy: 0.4837 - val_loss: 1.4351 - val_sparse_categorical_accuracy: 0.4
521
Epoch 6/30
108/108 [==============================] - 7s 67ms/step - loss: 1.3231 - sparse_cat
egorical_accuracy: 0.4987 - val_loss: 1.3937 - val_sparse_categorical_accuracy: 0.4
658
Epoch 7/30
108/108 [==============================] - 7s 65ms/step - loss: 1.2978 - sparse_cat
egorical_accuracy: 0.5085 - val_loss: 1.4261 - val_sparse_categorical_accuracy: 0.4
480
Epoch 8/30
108/108 [==============================] - 7s 67ms/step - loss: 1.2763 - sparse_cat
egorical_accuracy: 0.5244 - val_loss: 1.4770 - val_sparse_categorical_accuracy: 0.4
099
Epoch 9/30
108/108 [==============================] - 8s 70ms/step - loss: 1.2665 - sparse_cat
egorical_accuracy: 0.5251 - val_loss: 1.3479 - val_sparse_categorical_accuracy: 0.4
969
Epoch 10/30
108/108 [==============================] - 7s 67ms/step - loss: 1.2513 - sparse_cat
egorical_accuracy: 0.5358 - val_loss: 1.3416 - val_sparse_categorical_accuracy: 0.5
022
Epoch 11/30
108/108 [==============================] - 7s 64ms/step - loss: 1.2333 - sparse_cat
egorical_accuracy: 0.5426 - val_loss: 1.3334 - val_sparse_categorical_accuracy: 0.5
063
Epoch 12/30
108/108 [==============================] - 7s 67ms/step - loss: 1.2093 - sparse_cat
egorical_accuracy: 0.5550 - val_loss: 1.4417 - val_sparse_categorical_accuracy: 0.4
591
Epoch 13/30
108/108 [==============================] - 7s 69ms/step - loss: 1.1941 - sparse_cat
egorical_accuracy: 0.5558 - val_loss: 1.3334 - val_sparse_categorical_accuracy: 0.5
007
Epoch 14/30
108/108 [==============================] - 7s 68ms/step - loss: 1.1750 - sparse_cat
egorical_accuracy: 0.5587 - val_loss: 1.3311 - val_sparse_categorical_accuracy: 0.5
089
```

```
Epoch 15/30
108/108 [==============================] - 7s 69ms/step - loss: 1.1573 - sparse_cat
egorical_accuracy: 0.5644 - val_loss: 1.2850 - val_sparse_categorical_accuracy: 0.5
130
Epoch 16/30
108/108 [==============================] - 7s 70ms/step - loss: 1.1382 - sparse_cat
egorical_accuracy: 0.5736 - val_loss: 1.3782 - val_sparse_categorical_accuracy: 0.4
585
Epoch 17/30
108/108 [==============================] - 7s 68ms/step - loss: 1.1304 - sparse_cat
egorical_accuracy: 0.5733 - val_loss: 1.3432 - val_sparse_categorical_accuracy: 0.4
812
Epoch 18/30
108/108 [==============================] - 7s 69ms/step - loss: 1.1168 - sparse_cat
egorical_accuracy: 0.5817 - val_loss: 1.3470 - val_sparse_categorical_accuracy: 0.4
771
Epoch 19/30
108/108 [==============================] - 8s 70ms/step - loss: 1.1038 - sparse_cat
egorical_accuracy: 0.5864 - val_loss: 1.2672 - val_sparse_categorical_accuracy: 0.5
092
Epoch 20/30
108/108 [==============================] - 7s 69ms/step - loss: 1.0946 - sparse_cat
egorical_accuracy: 0.5874 - val_loss: 1.2726 - val_sparse_categorical_accuracy: 0.5
103
Epoch 21/30
108/108 [==============================] - 7s 68ms/step - loss: 1.0833 - sparse_cat
egorical_accuracy: 0.5908 - val_loss: 1.3228 - val_sparse_categorical_accuracy: 0.4
929
Epoch 22/30
108/108 [==============================] - 7s 69ms/step - loss: 1.0788 - sparse_cat
egorical_accuracy: 0.5936 - val_loss: 1.2533 - val_sparse_categorical_accuracy: 0.5
211
Epoch 23/30
108/108 [==============================] - 7s 67ms/step - loss: 1.0730 - sparse_cat
egorical_accuracy: 0.5976 - val_loss: 1.2633 - val_sparse_categorical_accuracy: 0.5
191
Epoch 24/30
108/108 [==============================] - 7s 69ms/step - loss: 1.0613 - sparse_cat
egorical_accuracy: 0.5982 - val_loss: 1.2588 - val_sparse_categorical_accuracy: 0.5
138
Epoch 25/30
108/108 [==============================] - 7s 67ms/step - loss: 1.0544 - sparse_cat
egorical_accuracy: 0.6037 - val_loss: 1.3446 - val_sparse_categorical_accuracy: 0.4
608
Epoch 26/30
108/108 [==============================] - 7s 68ms/step - loss: 1.0496 - sparse_cat
egorical_accuracy: 0.6029 - val_loss: 1.2331 - val_sparse_categorical_accuracy: 0.5
226
Epoch 27/30
108/108 [==============================] - 7s 68ms/step - loss: 1.0461 - sparse_cat
egorical_accuracy: 0.6069 - val_loss: 1.3440 - val_sparse_categorical_accuracy: 0.5
045
Epoch 28/30
108/108 [==============================] - 7s 67ms/step - loss: 1.0393 - sparse_cat
egorical_accuracy: 0.6077 - val_loss: 1.2683 - val_sparse_categorical_accuracy: 0.5
095
```

```
Epoch 29/30
108/108 [==============================] - 7s 68ms/step - loss: 1.0320 - sparse_cat
egorical_accuracy: 0.6089 - val_loss: 1.2146 - val_sparse_categorical_accuracy: 0.5
380
Epoch 30/30
108/108 [==============================] - 8s 70ms/step - loss: 1.0253 - sparse_cat
egorical_accuracy: 0.6150 - val_loss: 1.4162 - val_sparse_categorical_accuracy: 0.4
926
134/134 [==============================] - 3s 18ms/step

accuracy:  0.5396455223880597
```

This model is unfortunately the worst so far. It suggests that using an embedding without a CNN or RNN does not really add value.

Now we will try to use vectorizations as a basis for our embeddings rather than tokenizations.

In [ ]:

```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

max_len = 200
batch_size = 128
num_epochs = 20
embedding_dim = 128

vectorizer = TextVectorization(max_tokens=vocab_size, output_sequence_length=max_le
text_ds = tf.data.Dataset.from_tensor_slices(train.Text).batch(batch_size)
vectorizer.adapt(text_ds)

voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))

x_train = vectorizer(np.array([[s] for s in train.Text])).numpy()
x_test = vectorizer(np.array([[s] for s in test.Text])).numpy()

model_em4 = models.Sequential()
model_em4.add(layers.Embedding(len(word_index) + 1, embedding_dim, input_length=max
model_em4.add(layers.Conv1D(128, 5, activation='relu'))
model_em4.add(layers.MaxPooling1D(5))
model_em4.add(layers.Conv1D(128, 5, activation='relu'))
model_em4.add(layers.MaxPooling1D(5))
model_em4.add(layers.GlobalMaxPooling1D())
model_em4.add(layers.Dropout(0.5))
model_em4.add(layers.Dense(128, activation="relu"))
model_em4.add(layers.Dense(num_classes, activation='softmax'))

model_em4.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['sparse_categorical_accuracy']
)

model_em4.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=num_epochs,
    validation_split=0.2
)

pred_em4 = model_em4.predict(x_test)
pred_em4 = np.argmax(pred_em4, axis=1)

print('\naccuracy: ', accuracy_score(y_test, pred_em4))
```

```
Epoch 1/20
108/108 [==============================] - 27s 227ms/step - loss: 1.5891 - sparse_c
ategorical_accuracy: 0.3292 - val_loss: 1.6143 - val_sparse_categorical_accuracy:
0.3162
Epoch 2/20
108/108 [==============================] - 21s 196ms/step - loss: 1.4435 - sparse_c
ategorical_accuracy: 0.3893 - val_loss: 1.2660 - val_sparse_categorical_accuracy:
0.4969
Epoch 3/20
108/108 [==============================] - 21s 197ms/step - loss: 0.8360 - sparse_c
ategorical_accuracy: 0.6765 - val_loss: 1.0228 - val_sparse_categorical_accuracy:
0.5991
Epoch 4/20
108/108 [==============================] - 22s 206ms/step - loss: 0.4476 - sparse_c
ategorical_accuracy: 0.8310 - val_loss: 0.8150 - val_sparse_categorical_accuracy:
0.7051
Epoch 5/20
108/108 [==============================] - 22s 201ms/step - loss: 0.2415 - sparse_c
ategorical_accuracy: 0.9208 - val_loss: 0.8044 - val_sparse_categorical_accuracy:
0.7205
Epoch 6/20
108/108 [==============================] - 21s 198ms/step - loss: 0.1758 - sparse_c
ategorical_accuracy: 0.9439 - val_loss: 0.8403 - val_sparse_categorical_accuracy:
0.7397
Epoch 7/20
108/108 [==============================] - 22s 200ms/step - loss: 0.1272 - sparse_c
ategorical_accuracy: 0.9570 - val_loss: 0.8109 - val_sparse_categorical_accuracy:
0.7354
Epoch 8/20
108/108 [==============================] - 22s 200ms/step - loss: 0.0999 - sparse_c
ategorical_accuracy: 0.9672 - val_loss: 0.8779 - val_sparse_categorical_accuracy:
0.7377
Epoch 9/20
108/108 [==============================] - 24s 223ms/step - loss: 0.0755 - sparse_c
ategorical_accuracy: 0.9750 - val_loss: 0.9467 - val_sparse_categorical_accuracy:
0.7397
Epoch 10/20
108/108 [==============================] - 26s 242ms/step - loss: 0.0604 - sparse_c
ategorical_accuracy: 0.9804 - val_loss: 0.9529 - val_sparse_categorical_accuracy:
0.7354
Epoch 11/20
108/108 [==============================] - 45s 417ms/step - loss: 0.0494 - sparse_c
ategorical_accuracy: 0.9848 - val_loss: 0.9872 - val_sparse_categorical_accuracy:
0.7389
Epoch 12/20
108/108 [==============================] - 55s 508ms/step - loss: 0.0401 - sparse_c
ategorical_accuracy: 0.9868 - val_loss: 1.0320 - val_sparse_categorical_accuracy:
0.7377
Epoch 13/20
108/108 [==============================] - 23s 210ms/step - loss: 0.0349 - sparse_c
ategorical_accuracy: 0.9890 - val_loss: 1.0991 - val_sparse_categorical_accuracy:
0.7400
Epoch 14/20
108/108 [==============================] - 20s 183ms/step - loss: 0.0328 - sparse_c
ategorical_accuracy: 0.9902 - val_loss: 1.1438 - val_sparse_categorical_accuracy:
0.7319
```

```
Epoch 15/20
108/108 [==============================] - 22s 206ms/step - loss: 0.0270 - sparse_c
ategorical_accuracy: 0.9907 - val_loss: 1.1041 - val_sparse_categorical_accuracy:
0.7394
Epoch 16/20
108/108 [==============================] - 36s 334ms/step - loss: 0.0254 - sparse_c
ategorical_accuracy: 0.9921 - val_loss: 1.1492 - val_sparse_categorical_accuracy:
0.7365
Epoch 17/20
108/108 [==============================] - 40s 368ms/step - loss: 0.0204 - sparse_c
ategorical_accuracy: 0.9936 - val_loss: 1.1962 - val_sparse_categorical_accuracy:
0.7397
Epoch 18/20
108/108 [==============================] - 30s 279ms/step - loss: 0.0193 - sparse_c
ategorical_accuracy: 0.9939 - val_loss: 1.2036 - val_sparse_categorical_accuracy:
0.7383
Epoch 19/20
108/108 [==============================] - 27s 254ms/step - loss: 0.0182 - sparse_c
ategorical_accuracy: 0.9935 - val_loss: 1.2196 - val_sparse_categorical_accuracy:
0.7336
Epoch 20/20
108/108 [==============================] - 29s 265ms/step - loss: 0.0187 - sparse_c
ategorical_accuracy: 0.9934 - val_loss: 1.2705 - val_sparse_categorical_accuracy:
0.7354
134/134 [==============================] - 3s 22ms/step
```

```
accuracy:  0.8708022388059702
```

This model slightly outperformed our original model.

## Analysis

The best models were the dense sequential network and the last CNN, which utilized vectorizations of the input data. The sequential network likely outperformed most of the CNNs because looking for the presence or absence of words already gives a solid foundation for judging the sentiment of a text.

However, the CNNs may in general have been able to perform better if they were given more training time. For all but the last CNN tested, the increase in accuracy between epochs had yet to significantly plateau. Thus, it is likely that all the CNNs in this notebook are not performing at their most optimal level. The number of epochs was cut down in the interest of time, but it is clear that CNNs simply take a longer time to achieve better results.

Using pretrained embeddings definitely provides some value. Switching from the 100 dimension to 300 dimension versions of GloVe slightly improved performance, and as stated previously, all the CNNs may perform better if given more epochs to train. The true value of the GloVe embeddings may also be more apparent if RNNs were used on this dataset. However, RNNs were avoided simply because the training time is too long.