Week 6 Final Project

August 10, 2023

1 Introduction and Objective

The objective of this project is to use deep learning techniques to classify text from news articles into four different categories. This project can be found on Github at: https://github.com/highdeltav/DeepLearningWeek6

1.1 Libraries and Helper Functions

[ ]:
```
import pandas as pd
import numpy as np

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.utils import to_categorical

from gensim.parsing.preprocessing import remove_stopwords
from gensim.parsing.preprocessing import strip_punctuation
from gensim.parsing.preprocessing  import lower_to_unicode
from gensim.models import Word2Vec
from gensim.downloader import load as gensim_downloader
from gensim.models import Phrases
from gensim.test.utils import datapath
from gensim import utils

from nltk.corpus import stopwords

from matplotlib import pyplot as plt
from seaborn import histplot
from wordcloud import WordCloud, ImageColorGenerator

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
```

```python
import json
```

[12]:

```python
def json_out(d, filename):
    """ Export a dictionary to a JSON """
    with open(f"{filename}.json", "w") as save_file:
        json.dump(d, save_file)

def json_to_dict(filename):
    """ Convert a JSON to a dictionary """
    with open(f"{filename}", 'r') as file:
        d = json.load(file)
    return d

def rm_stop_words(text, other_words = None):
    """Remove stop words from text"""
    # Add our own words
    stop_words = stopwords.words('english')
    if other_words != None:
        stop_words.extend(other_words)

    words = [word for word in text.split() if word.lower() not in stop_words]
    new_text = " ".join(words)
    return new_text

def plot_all_plots(d, plot_type = 'accuracy', cols = 3):
    """Plot stats of all histories in a dictionary on a single chart"""

    total_plots = len(d)
    rows = int(np.ceil(total_plots/cols))
    plt.figure(figsize = (5*cols,5*rows))


    # Postion differently depending on amount of rows
    title_y = .98
    if rows >= 3:
        title_y = .9
    elif rows == 2:
        title_y = .93
    #Set the amount of columns


    for i, model in enumerate(d):

         #Check to see which variables to plot
        if plot_type == 'accuracy':
            y_axis_train = d[model]['accuracy']
            y_axis_val = d[model]['val_accuracy']
```

```python
            title = 'Accuracy'
            y_limit = [.5,1]
        elif plot_type == 'loss':
            y_axis_train = d[model]['loss']
            y_axis_val = d[model]['val_loss']
            title = 'Loss'
            y_limit = [0,1.5]
        else:
            print('Avaliable options are: Loss and Accuracy')
            return


        x_epochs = np.arange(1,len(y_axis_train)+1)
        plt.subplot(rows, cols, i+1)
        plt.plot(x_epochs, y_axis_train, label = 'Training')
        plt.plot(x_epochs, y_axis_val,   label = 'Validation')
        plt.ylim(y_limit)
        plt.title(f"{model}")
        plt.xlabel('Epochs')
        plt.ylabel(title)

        plt.legend()

    plt.suptitle(title, fontsize=20, y = title_y)
    plt.show()

def plot_models_together(d):
    """Plots all of the metircs curves on 4 plots in 1 figure"""

    fig, axes = plt.subplots(2,2)
    fig.set_figheight(10)
    fig.set_figwidth(10)
    #ax.plot(range(self.epochs), val_acc,   label = 'Validation')
    plot_types = ['accuracy','val_accuracy','loss','val_loss']
    for p in plot_types:
        if p == 'val_accuracy':
            ylab = 'Accuracy'
            title = 'Validation Accuracy'
            lim = [.5,1]
            ax = axes[0,0]
        elif p == 'accuracy':
            ylab = 'Accuracy'
            title = 'Training Accuracy'
            lim = [.5,1]
            ax = axes[0,1]
        elif p =='val_loss':
            ylab = 'Loss'
```

```
            title = 'Validation Loss'
            lim = [0,1.5]
            ax = axes[1,0]
        elif p == 'loss':
            ylab = 'Loss'
            title = 'Training Loss'
            lim = [0,1.5]
            ax = axes[1,1]
        #Add data from all of the models
        for model in d:
            train_acc = (d[model][p])
            x_epochs = np.arange(1,len(d[model][p])+1)
            ax.plot(x_epochs, train_acc, label = model)

        ax.set_ylim(lim)
        ax.set_xlabel('Epoch')
        ax.set_ylabel(ylab)
        ax.set_title(title)
        ax.legend()
    plt.suptitle('Model Comparison', fontsize=20, y = .93)
    plt.show()
def predict_from_model(model, X):
    """Outputs predictions from the model output"""
    preds = model.predict(X)
    #Returns the hightsest value from the SoftMax output
    return np.argmax(preds, axis = 1)


def create_summary_table(d):
    """Creates a summary table from a dictionary"""

    max_acc = []
    min_loss = []
    max_val_acc = []
    epoch_min_val_loss = []
    min_val_loss = []
    row_names = []
    runtime = []
    for each in d:
        max_acc.append(max(d[each]['accuracy']))
        min_loss.append( min(d[each]['loss']))
        max_val_acc.append(max(d[each]['val_accuracy']))
        min_val_loss.append(min(d[each]['val_loss']))
        epoch_min_val_loss.append(np.argmin(d[each]['val_loss']))
        row_names.append(each)
    df = pd.DataFrame(list(zip(row_names, max_acc,min_loss, max_val_acc,
 ↪min_val_loss, epoch_min_val_loss)),
```

```
                   columns = ['Model','Max Acc', 'Minnimum Loss','Max Val␣
↪Acc','Min Val Loss','Epoch of Min Val Loss']).round(3)
   display(df)
```

2 Data

2.1 Information and Initial EDA

This dataset is a subset of the AG news classification dataset, which contains 1 million news articles. This subset was compiled by Xiang Zhang for his paper "Character-level Convolutional Networks for Text Classification." and uploaded to Kaggle

This dataset is composed of two different files, one us designated as training data, and one is designated as test data. There are 120,000 records in the training data, and 7,600 in the test data. Each record is a listing of a short news article for classification. It has three features, one is the 'Class Index', one is the 'Title' of the article, and one is the 'Description.' There are 30,000 articles of each class in the training dataset, and 1,900 in the test set. All of the articles are evenly distributed, so there is no class imbalance.

The longest article is 985 characters, and the shortest has only 6 words. There are also no null values in the data that needed to be addressed.

[4]:
```
# The class ids are numbered 1-4 where 1 represents World, 2 represents Sports,␣
↪3 represents Business and 4 represents Sci/Tech.
train = pd.read_csv('/kaggle/input/ag-news-classification-dataset/train.csv')
test = pd.read_csv('/kaggle/input/ag-news-classification-dataset/test.csv')
class_ids = ['World','Sports','Business','Sci/Tech']
```

[4]:
```
train.tail()
```

[4]:
```
        Class Index                                              Title  \
119995            1  Pakistan's Musharraf Says Won't Quit as Army C…
119996            2                  Renteria signing a top-shelf deal
119997            2                    Saban not going to Dolphins yet
119998            2                                  Today's NFL games
119999            2                       Nets get Carter from Raptors

                                              Description
119995   KARACHI (Reuters) - Pakistani President Perve…
119996  Red Sox general manager Theo Epstein acknowled…
119997  The Miami Dolphins will put their courtship of…
119998  PITTSBURGH at NY GIANTS Time: 1:30 p.m. Line: …
119999  INDIANAPOLIS -- All-Star Vince Carter was trad…
```

[5]:
```
print(f"Train: {train.describe()}")
print(f"Test: {test.describe()}")
```

```
Train:          Class Index
```

```
count   120000.000000
mean         2.500000
std          1.118039
min          1.000000
25%          1.750000
50%          2.500000
75%          3.250000
max          4.000000
Test:         Class Index
count   7600.000000
mean       2.500000
std        1.118108
min        1.000000
25%        1.750000
50%        2.500000
75%        3.250000
max        4.000000
```

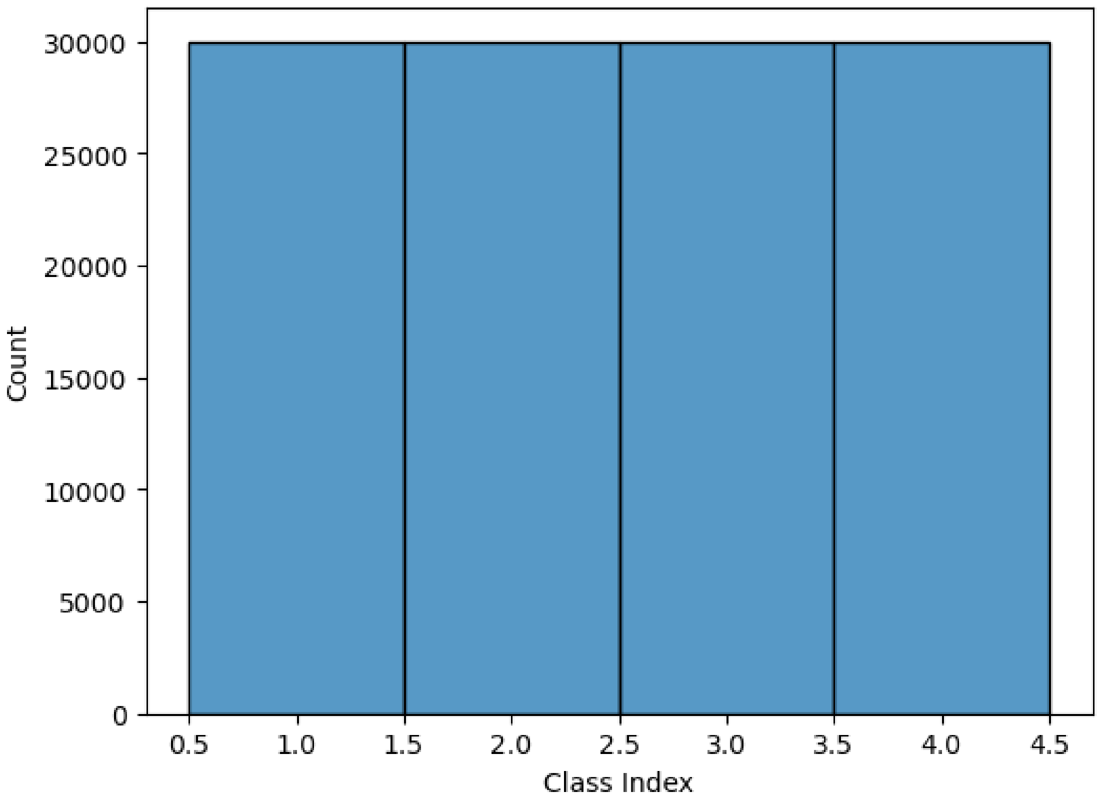
[6]:
```python
## Verify there are no null values in the description and predictions
print(train[train['Description'].isnull()])
print(train[train['Class Index'].isnull()])
```

```
Empty DataFrame
Columns: [Class Index, Title, Description]
Index: []
Empty DataFrame
Columns: [Class Index, Title, Description]
Index: []
```

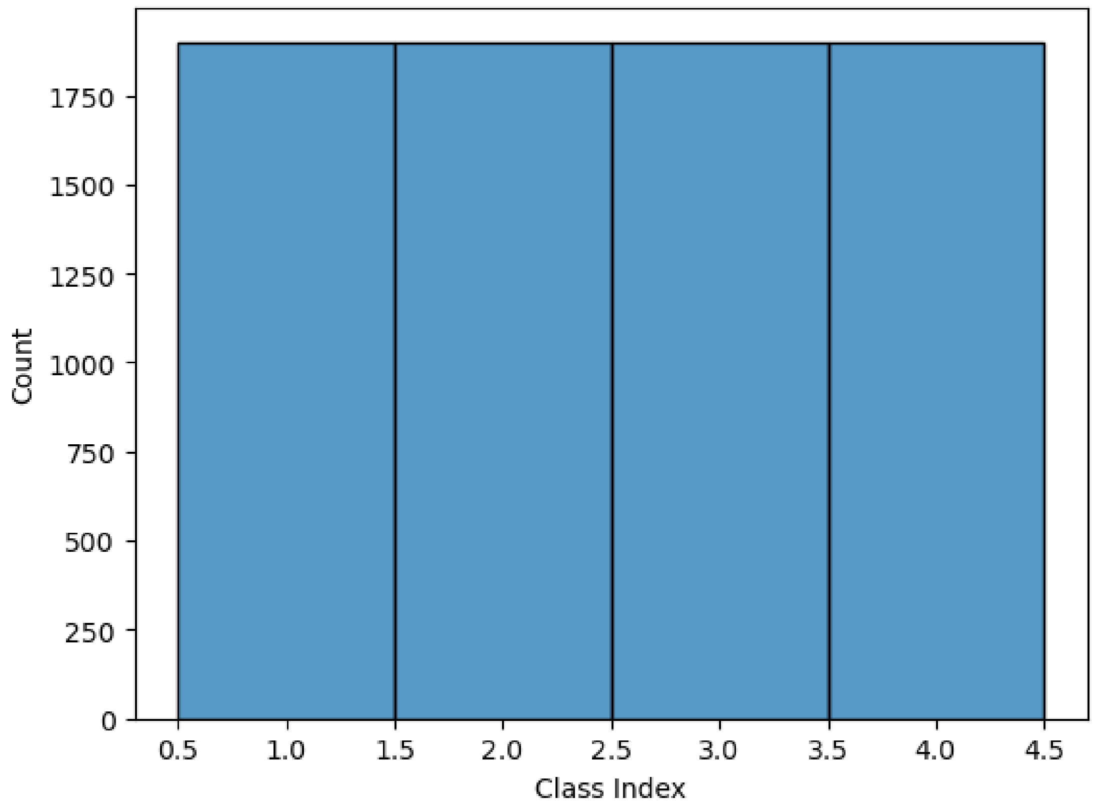
[7]:
```python
print(f"Longest article: {train['Description'].map(lambda x: len(x)).max()}␣
↪words")
print(f"Shortest article: {train['Description'].map(lambda x: len(x)).min()}␣
↪words")
```

```
Longest article: 985 words
Shortest article: 6 words
```

[6]:
```python
histplot(train['Class Index'], discrete = True)
plt.show()
```



[70]:
```
histplot(test['Class Index'], discrete = True)
plt.show()
```




```
[71]: pd.set_option('display.max_colwidth', None)
      #train[train['Description'].str.contains('[UNK]')]
```

2.2 Pre-processing

For this project, I decided to just do the text analysis on the description, and ignore the the title. I considered just using the title, or concatenating the title and the description, but in the end, decided to keep it simple, and just use the description field. Evaluating which of those options would yield the best results, could be an interesting expansion of this project, but is out of scope for this specific project.

The first thing I did was remove all of the stop words from the list gensim library. This removes words that are normally common to all the data. Words such as 'the.' This is not always the recommended approach, but in this instance, I decided it would be beneficial.

In addition to removing the stop words, I also stripped the articles of punctuation and made all of the words lowercase.

I also changed the category designations from 1 to 4, to 0 to 3, to make it easier to to use them as classifiers in the machine learning process.

```
[7]: # Make the class designations 0 to 3 instead of 1 to 4
train['Class Index'] = train['Class Index']-1
test['Class Index'] = test['Class Index']-1
```

```
[13]: # Process train data
train['Description'] = train['Description'].apply(lower_to_unicode)
train['Description'] = train['Description'].apply(strip_punctuation)
train['Description'] = train['Description'].apply(rm_stop_words)
#train['Description'] = train['Description'].apply(rm_stop_words, args =
 ↪([remove_word_list]))

# Prccess test data
test['Description'] = test['Description'].apply(lower_to_unicode)
test['Description'] = test['Description'].apply(strip_punctuation)
test['Description'] = test['Description'].apply(rm_stop_words)
#test['Description'] = test['Description'].apply(rm_stop_words, args =
 ↪([remove_word_list]))
```

```
[14]: train['Description']
```

```
[14]: 0         reuters short sellers wall street dwindling ba…
1         reuters private investment firm carlyle group …
2         reuters soaring crude prices plus worries econ…
3         reuters authorities halted oil export flows ma…
4         afp tearaway world oil prices toppling records…
                                ...
119995    karachi reuters pakistani president pervez mus…
119996    red sox general manager theo epstein acknowled…
119997    miami dolphins put courtship lsu coach nick sa…
119998    pittsburgh ny giants time 1 30 p line steelers…
119999    indianapolis star vince carter traded toronto …
Name: Description, Length: 120000, dtype: object
```

2.3 EDA Part Two

After I had done some initial preprocessing of the data, I did some additional EDA. I used CountVectorizer to tokenize the data, so I could have a better understanding of it. I also created a word cloud, and a most common words list so I could see if there were other words that I wanted to remove.

```
[15]: # Creeate a CountVectorizer to do more data analysis
vectorizer = CountVectorizer()
vec = vectorizer.fit_transform(train['Description'])
```

```
[16]: # Create a list of words sorted by their frequency
word_sum_list = vec.sum(axis = 0).tolist()[0]
feature_list = vectorizer.get_feature_names_out()
```

```
most_words_list=sorted(zip(feature_list,word_sum_list), key = lambda x:␣
↪x[1],reverse = True)
```

[17]:
```
print(f"Total number of words: {len(word_sum_list)}")
```

```
Total number of words: 60564
```

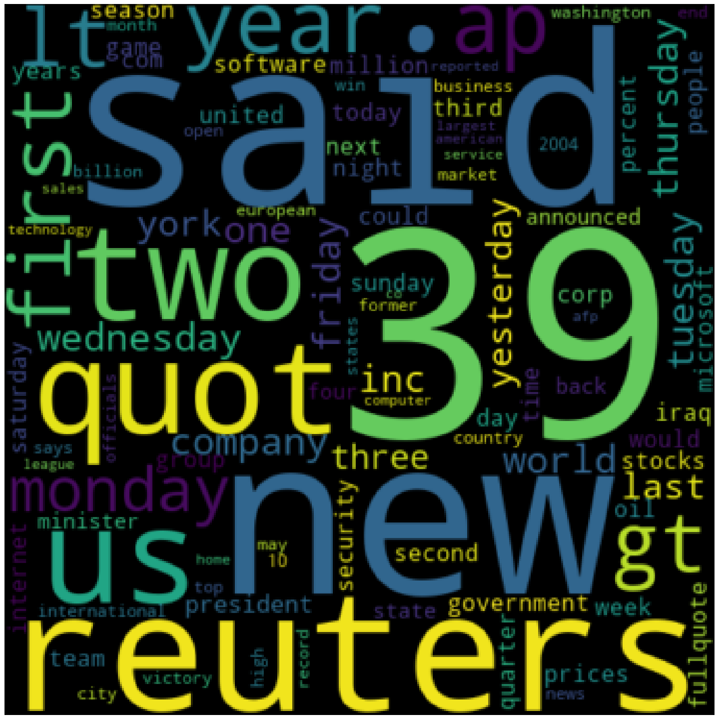
[18]:
```
most_word_dictionary = dict(most_words_list[:100])
```

[20]:
```
word_freq = {}
for i, words in enumerate(most_words_list[:10]):
    print(f"{i+1}. {words[0]}: {words[1]}")
```

```
1. 39: 31874
2. said: 20098
3. new: 17392
4. reuters: 15078
5. two: 9248
6. us: 9079
7. quot: 8941
8. year: 8923
9. first: 8596
10. ap: 8499
```

[21]:
```
# Create and generate a word cloud image:
wordcloud = WordCloud(width = 500, height = 500).
↪generate_from_frequencies(most_word_dictionary)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



2.3.1 Additional Word Removal

After looking at the tops words list and the word cloud, I decided to remove the names of the news organizations such as "Reuters." I did this because I didn't want the data to rely to much on the news organizations for classification, when additional data might be from an unseen one. In addition, I also removed some words that appeared to be artifacts in the data gathering process such as "fullquote."

2.3.2 Text Summary

There were a total of 60554 words, after the removal of the stop words. "Said" was the word that was used the most used word, used 20,098 times.

[23]:
```
# Add words to remove that were not n the default stop words list
remove_word_list = ["quot", "fullquote", "reuters", 'ap', '39', 'afp', 'lt',␣
↪'gt', 'quote', 'sn' ]

# Remove additional words
train['Description'] = train['Description'].apply(rm_stop_words, args =␣
↪([remove_word_list]))
test['Description'] = test['Description'].apply(rm_stop_words, args =␣
↪([remove_word_list]))
```

[24]:
```python
# Create a CountVectorizer to do more data analysis
vectorizer = CountVectorizer()
vec = vectorizer.fit_transform(train['Description'])

# Create a list of words sorted by their frequency
word_sum_list = vec.sum(axis = 0).tolist()[0]
feature_list = vectorizer.get_feature_names_out()

most_words_list=sorted(zip(feature_list,word_sum_list), key = lambda x:
    ↪x[1],reverse = True)
most_word_dictionary = dict(most_words_list[:100])

# Create Total Words for Models
total_words = len(word_sum_list)

word_freq = {}
for i, words in enumerate(most_words_list[:10]):
    print(f"{i+1}. {words[0]}: {words[1]}")
```

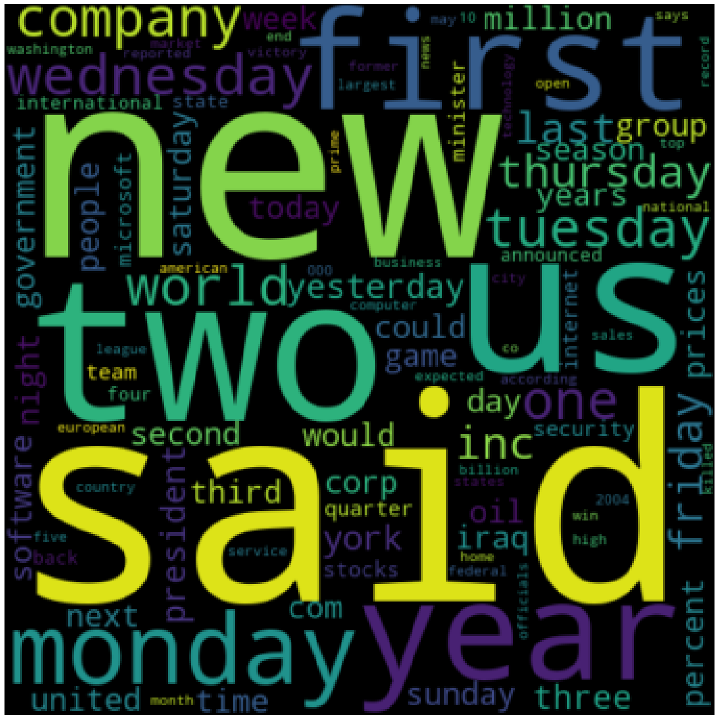
```
1. said: 20098
2. new: 17392
3. two: 9248
4. us: 9079
5. year: 8923
6. first: 8596
7. monday: 7506
8. wednesday: 7461
9. tuesday: 7388
10. one: 7365
```

[33]:
```python
print(f"Total Words: {total_words}")
```

```
Total Words: 60554
```

[25]:
```python
# Create and generate a word cloud image:
wordcloud = WordCloud(width = 500, height = 500).
    ↪generate_from_frequencies(most_word_dictionary)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



2.4 Split train/test

I split the train set into a train and validation set, and kept the test set for model testing at the end.

[28]:
```
X_train, X_val, y_train, y_val = train_test_split(train['Description'],␣
↪train['Class Index'], random_state = 42)
X_test = test['Description']
y_test = test['Class Index']
```

2.5 Create Text Vectorization Layer

In order to pass the vectorized text to Keras, I created a layer that took the X_train data, and vectorized it. This is the layer that our text data will be passed to for training and testing purposes. I did make a concious choice to not pass the validation data to this layer as well. While it would help the validation data have better accuracy, and add additional words to the vocabulary, it would be sacrificing the ability in our testing to see what completely new data looks like, where not all of its words have been fit with the text vectorization layer.

[31]:
```
# Create the enocder for the text vectorizaton layer
#number_of_words = 1000

vectorize_layer = tf.keras.layers.TextVectorization()
vectorize_layer.adapt(list(X_train))
```

```
word_count = vectorize_layer.vocabulary_size()
```

[30]:
```
print(f"Word Count: {word_count}")
```

```
Word Count: 54560
```

2.6 Word2Vec

I decided to got a step beyond just tokenizing the data and create a Word2Vec model. This is a model that tries to estimate the context of words, instead of just going by the quantity of each word. It often uses a metric like cosine distance too see if a word is similar. Word vectors are passed to a model and this trained with a simple neural net. This information is then passed to an embedding layer as weights, for model training.

For training this WordVec model, I used the Gensim library to generate a model based on the training data. After it generated the model, I then had to transfer this to a matrix, that could be used as the weights for the embedding layer. Saturn Clouds blog post was invaluable in learning how to do this.

[49]:
```
# Create a dictionary of the vectors for training the WordVec Model
word_idx = {}
for i, word in enumerate(vectorize_layer.get_vocabulary()):
    word_idx[word] = i
```

[54]:
```
#https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html

from gensim.test.utils import datapath
from gensim import utils

class MyCorpus:
    """An iterator that yields sentences (lists of str)."""
    def __init__(self,texts):
        self.texts = texts
    def __iter__(self):

        for line in self.texts:
            # assume there's one document per line, tokens separated by␣
 ↪whitespace
            yield utils.simple_preprocess(line)
```

[56]:
```
w2v_model = Word2Vec(sentences=MyCorpus(X_train), vector_size = 256)
print('Finished')
```

```
Finished
```

[57]:
```
# See what my model thinks is similar to the word 'score.'
w2v_model.wv.most_similar('score')
```

[57]:
```
[('scoring', 0.9402279257774353),
 ('ball', 0.9141973853111267),
 ('yard', 0.895178496837616),
 ('seconds', 0.8881029486656189),
 ('touchdown', 0.8861629366874695),
 ('goal', 0.8822108507156372),
 ('stubblefield', 0.8820501565933228),
 ('tied', 0.872307300567627),
 ('offense', 0.8630639314651489),
 ('passes', 0.8618302345275879)]
```

[58]:
```
for index, word in enumerate(w2v_model.wv.index_to_key):
    if index == 10:
        break
    print(f"word #{index}/{len(w2v_model.wv.index_to_key)} is {word}")
```

```
word #0/20690 is said
word #1/20690 is new
word #2/20690 is two
word #3/20690 is us
word #4/20690 is year
word #5/20690 is first
word #6/20690 is wednesday
word #7/20690 is monday
word #8/20690 is tuesday
word #9/20690 is one
```

[59]:
```
#Modified from code at https://saturncloud.io/blog/
↪using-pretrained-gensim-word2vec-embeddings-in-keras-a-comprehensive-guide/

# Create the embeding matrix fromt he WordVec Model
max_words = word_count
embed_size = 256
nb_words = min(max_words, len(word_idx))
embedding_matrix = np.zeros((nb_words, embed_size))

# Loop through the words and assign their values to the weight matrix
for word, i in word_idx.items():
    if i >= max_words:
        continue
    if word in w2v_model.wv.key_to_index:
        embedding_matrix[i] = w2v_model.wv.get_vector(word)
```

3 Model Building

3.1 Class Creation

I created a custom class that inherits the Keras model, and added a custom callback that would save the data after every epoch, instead of just at the end of the model run.

[50]:
```
class CustomCallback(keras.callbacks.Callback):
    """Saves data to a dictionary in the class"""
    def __init__(self, model):
        super().__init__()
        self.model = model
    def on_epoch_end(self, epoch, logs=None):
        keys = list(logs.keys())
        stats = self.model.stats[self.model.name]
        stats['epochs'] += 1
        for key in keys:
            if key not in stats:
                stats[key] = []
            stats[key].append(logs[key])

        #print(f"Total Epochs Run: {stats['epochs']}")
```

[51]:
```
# Model Class
class Model_Seq(Sequential):
    """ Inherits the Sequential class"""
    def __init__(self, model_layers, name = 'model',
                 X_train = X_train,
                 y_train = y_train,
                 X_val = X_val,
                 y_val = y_val,
                 inital_lr = .001,
                 chart_title = None):
        super().__init__()

        tf.random.set_seed(846)
        self.call_backs = CustomCallback(self)
        self.add_layers(model_layers)
        self._name = name
        self.stats = {self.name:{'epochs': 0}}
        self.X_train = X_train
        self.y_train = y_train
        self.X_val = X_val
        self.y_val = y_val
        self.loss_fn = tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False)
        self.metrics1 = ['accuracy']
```

```
        self.lr = inital_lr
        if chart_title == None:
            self.chart_title = self.name
        else:
            self.chart_title = chart_title
        self.compile_model(self.lr)

    def add_layers(self,layers1):
        """Adds layers to the model"""
        for layer in layers1:
            self.add(layer)

    def compile_model(self, learning_rate = None):
        """ Compiles the model, with an optional learning rate"""
        if learning_rate == None:
            learning_rate = self.lr
        self.compile(loss=self.loss_fn,
              optimizer=tf.keras.optimizers.Adam(learning_rate =learning_rate),
              metrics=self.metrics1)

    def train_model(self, epochs, batch_size = 64):
        """Trains the model"""
        self.fit(
            x=self.X_train,
            y=self.y_train,
            validation_data = (self.X_val, self.y_val),
            batch_size=batch_size,
            epochs=epochs,
            callbacks=self.call_backs,
            shuffle=True,
            class_weight=None,
            sample_weight=None,
            initial_epoch=self.stats[self.name]['epochs']
)

    def rename(self,new_name):
        """Renames the Model"""
        self.stats[new_name] = self.stats.pop(self.name)
        self._name = new_name
```

3.2 Architecture

It was at this point that I began building the models that I wanted to test. I decided to start off with two different neural network architectures. I chose to do a LSTM network and a Convolutional Neural Network. I would then train both networks with data that had been passed through a WordVec model and data that where the text vectors were just passed through a trainable embedding later. After training each model, I would have a better idea about which model would work well for this specific data, and then I could tune the hyperparameters of one or two models.

4 Model Training

I decided to use fifty epochs for each test. After seeing the results, this was probably too much, and I could have reduced it, but the models were running fast enough, I decided to keep using fifty epochs for all of my tuning for consistency in evaluation.

4.1 Inital LTSM Model

[ ]:
```
lstm_wordvec_model = Model_Seq([vectorize_layer,
                                layers.Embedding(nb_words,
                                                 embed_size,
                                                 weights=[embedding_matrix],
                                                 trainable=False),
                                layers.LSTM(64,),
                                layers.Dense(128, activation='relu'),
                                layers.Dense(4,  activation='softmax')
                                ],
                               name = 'LSTM_with_WordVec', X_train = X_train,␣
 ↪y_train = y_train, X_val = X_val, y_val = y_val, inital_lr=.00005)
```

[63]:
```
lstm_wordvec_model.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 13s 10ms/step - loss: 0.9838 -
accuracy: 0.5487 - val_loss: 0.6627 - val_accuracy: 0.7530
Epoch 2/50
704/704 [==============================] - 6s 9ms/step - loss: 0.4978 -
accuracy: 0.8392 - val_loss: 0.4415 - val_accuracy: 0.8548
Epoch 3/50
704/704 [==============================] - 6s 9ms/step - loss: 0.4180 -
accuracy: 0.8619 - val_loss: 0.4127 - val_accuracy: 0.8622
Epoch 4/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3960 -
accuracy: 0.8675 - val_loss: 0.3948 - val_accuracy: 0.8675
Epoch 5/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3830 -
accuracy: 0.8717 - val_loss: 0.3861 - val_accuracy: 0.8702
Epoch 6/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3736 -
accuracy: 0.8747 - val_loss: 0.3792 - val_accuracy: 0.8710
Epoch 7/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3669 -
accuracy: 0.8765 - val_loss: 0.3743 - val_accuracy: 0.8739
Epoch 8/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3602 -
```

```
accuracy: 0.8790 - val_loss: 0.3682 - val_accuracy: 0.8748
Epoch 9/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3555 -
accuracy: 0.8796 - val_loss: 0.3704 - val_accuracy: 0.8737
Epoch 10/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3509 -
accuracy: 0.8812 - val_loss: 0.3605 - val_accuracy: 0.8777
Epoch 11/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3469 -
accuracy: 0.8829 - val_loss: 0.3619 - val_accuracy: 0.8770
Epoch 12/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3431 -
accuracy: 0.8842 - val_loss: 0.3575 - val_accuracy: 0.8784
Epoch 13/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3394 -
accuracy: 0.8847 - val_loss: 0.3555 - val_accuracy: 0.8795
Epoch 14/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3363 -
accuracy: 0.8865 - val_loss: 0.3532 - val_accuracy: 0.8796
Epoch 15/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3332 -
accuracy: 0.8868 - val_loss: 0.3534 - val_accuracy: 0.8800
Epoch 16/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3311 -
accuracy: 0.8874 - val_loss: 0.3497 - val_accuracy: 0.8811
Epoch 17/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3282 -
accuracy: 0.8890 - val_loss: 0.3468 - val_accuracy: 0.8816
Epoch 18/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3255 -
accuracy: 0.8899 - val_loss: 0.3479 - val_accuracy: 0.8819
Epoch 19/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3230 -
accuracy: 0.8906 - val_loss: 0.3460 - val_accuracy: 0.8827
Epoch 20/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3211 -
accuracy: 0.8918 - val_loss: 0.3440 - val_accuracy: 0.8833
Epoch 21/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3192 -
accuracy: 0.8917 - val_loss: 0.3432 - val_accuracy: 0.8828
Epoch 22/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3164 -
accuracy: 0.8926 - val_loss: 0.3419 - val_accuracy: 0.8833
Epoch 23/50
704/704 [==============================] - 8s 11ms/step - loss: 0.3143 -
accuracy: 0.8937 - val_loss: 0.3415 - val_accuracy: 0.8834
Epoch 24/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3121 -
```

```
accuracy: 0.8944 - val_loss: 0.3410 - val_accuracy: 0.8846
Epoch 25/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3107 -
accuracy: 0.8949 - val_loss: 0.3375 - val_accuracy: 0.8854
Epoch 26/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3088 -
accuracy: 0.8956 - val_loss: 0.3386 - val_accuracy: 0.8850
Epoch 27/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3067 -
accuracy: 0.8963 - val_loss: 0.3378 - val_accuracy: 0.8847
Epoch 28/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3051 -
accuracy: 0.8968 - val_loss: 0.3376 - val_accuracy: 0.8864
Epoch 29/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3038 -
accuracy: 0.8975 - val_loss: 0.3377 - val_accuracy: 0.8843
Epoch 30/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3022 -
accuracy: 0.8978 - val_loss: 0.3394 - val_accuracy: 0.8852
Epoch 31/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2997 -
accuracy: 0.8986 - val_loss: 0.3346 - val_accuracy: 0.8872
Epoch 32/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2988 -
accuracy: 0.8989 - val_loss: 0.3341 - val_accuracy: 0.8854
Epoch 33/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2980 -
accuracy: 0.8989 - val_loss: 0.3341 - val_accuracy: 0.8860
Epoch 34/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2953 -
accuracy: 0.9006 - val_loss: 0.3319 - val_accuracy: 0.8872
Epoch 35/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2944 -
accuracy: 0.8998 - val_loss: 0.3344 - val_accuracy: 0.8863
Epoch 36/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2935 -
accuracy: 0.9004 - val_loss: 0.3340 - val_accuracy: 0.8863
Epoch 37/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2914 -
accuracy: 0.9019 - val_loss: 0.3318 - val_accuracy: 0.8875
Epoch 38/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2901 -
accuracy: 0.9019 - val_loss: 0.3299 - val_accuracy: 0.8872
Epoch 39/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2896 -
accuracy: 0.9024 - val_loss: 0.3324 - val_accuracy: 0.8874
Epoch 40/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2872 -
```

```
accuracy: 0.9027 - val_loss: 0.3390 - val_accuracy: 0.8837
Epoch 41/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2861 -
accuracy: 0.9035 - val_loss: 0.3318 - val_accuracy: 0.8881
Epoch 42/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2857 -
accuracy: 0.9034 - val_loss: 0.3311 - val_accuracy: 0.8872
Epoch 43/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2840 -
accuracy: 0.9045 - val_loss: 0.3296 - val_accuracy: 0.8870
Epoch 44/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2822 -
accuracy: 0.9044 - val_loss: 0.3296 - val_accuracy: 0.8878
Epoch 45/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2818 -
accuracy: 0.9041 - val_loss: 0.3292 - val_accuracy: 0.8885
Epoch 46/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2803 -
accuracy: 0.9044 - val_loss: 0.3292 - val_accuracy: 0.8881
Epoch 47/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2792 -
accuracy: 0.9052 - val_loss: 0.3296 - val_accuracy: 0.8876
Epoch 48/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2781 -
accuracy: 0.9058 - val_loss: 0.3270 - val_accuracy: 0.8889
Epoch 49/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2767 -
accuracy: 0.9056 - val_loss: 0.3287 - val_accuracy: 0.8899
Epoch 50/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2761 -
accuracy: 0.9059 - val_loss: 0.3285 - val_accuracy: 0.8890
```

[65]:
```python
ltsm_1_token_model = Model_Seq([vectorize_layer,
                layers.Embedding(
                    input_dim=word_count,
                    output_dim=256,
                    mask_zero=True),
                layers.LSTM(64),
                layers.Dense(128, activation='relu'),
        layers.Dense(4,  activation='softmax')
         ], name = 'LSTM_token', X_train = X_train, y_train = y_train, X_val =
 ↪X_val, y_val = y_val, inital_lr=.00005)
ltsm_1_token_model.summary()
```

```
Model: "LSTM_token_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 text_vectorization_1 (TextV  (None, None)             0
 ectorization)

 embedding_1 (Embedding)     (None, None, 256)         13967360

 lstm_1 (LSTM)               (None, 64)                82176

 dense_2 (Dense)             (None, 128)               8320

 dense_3 (Dense)             (None, 4)                 516

=================================================================
Total params: 14,058,372
Trainable params: 14,058,372
Non-trainable params: 0
_________________________________________________________________
```

[66]:

```
ltsm_1_token_model.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 61s 79ms/step - loss: 0.9389 -
accuracy: 0.6267 - val_loss: 0.5220 - val_accuracy: 0.8311
Epoch 2/50
704/704 [==============================] - 14s 20ms/step - loss: 0.3976 -
accuracy: 0.8737 - val_loss: 0.3849 - val_accuracy: 0.8761
Epoch 3/50
704/704 [==============================] - 13s 19ms/step - loss: 0.3019 -
accuracy: 0.9067 - val_loss: 0.3569 - val_accuracy: 0.8836
Epoch 4/50
704/704 [==============================] - 13s 18ms/step - loss: 0.2537 -
accuracy: 0.9221 - val_loss: 0.3470 - val_accuracy: 0.8864
Epoch 5/50
704/704 [==============================] - 12s 17ms/step - loss: 0.2188 -
accuracy: 0.9337 - val_loss: 0.3475 - val_accuracy: 0.8876
Epoch 6/50
704/704 [==============================] - 11s 16ms/step - loss: 0.1913 -
accuracy: 0.9426 - val_loss: 0.3584 - val_accuracy: 0.8855
Epoch 7/50
704/704 [==============================] - 13s 18ms/step - loss: 0.1680 -
accuracy: 0.9498 - val_loss: 0.3759 - val_accuracy: 0.8848
Epoch 8/50
704/704 [==============================] - 13s 18ms/step - loss: 0.1488 -
accuracy: 0.9556 - val_loss: 0.3896 - val_accuracy: 0.8824
Epoch 9/50
704/704 [==============================] - 12s 17ms/step - loss: 0.1319 -
accuracy: 0.9611 - val_loss: 0.4082 - val_accuracy: 0.8794
Epoch 10/50
704/704 [==============================] - 11s 16ms/step - loss: 0.1182 -
```

```
accuracy: 0.9647 - val_loss: 0.4259 - val_accuracy: 0.8770
Epoch 11/50
704/704 [==============================] - 11s 16ms/step - loss: 0.1050 -
accuracy: 0.9691 - val_loss: 0.4507 - val_accuracy: 0.8749
Epoch 12/50
704/704 [==============================] - 11s 16ms/step - loss: 0.0951 -
accuracy: 0.9715 - val_loss: 0.4695 - val_accuracy: 0.8729
Epoch 13/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0856 -
accuracy: 0.9744 - val_loss: 0.5015 - val_accuracy: 0.8713
Epoch 14/50
704/704 [==============================] - 11s 15ms/step - loss: 0.0767 -
accuracy: 0.9770 - val_loss: 0.5151 - val_accuracy: 0.8691
Epoch 15/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0702 -
accuracy: 0.9794 - val_loss: 0.5389 - val_accuracy: 0.8657
Epoch 16/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0634 -
accuracy: 0.9811 - val_loss: 0.5575 - val_accuracy: 0.8651
Epoch 17/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0590 -
accuracy: 0.9824 - val_loss: 0.6005 - val_accuracy: 0.8639
Epoch 18/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0534 -
accuracy: 0.9838 - val_loss: 0.6189 - val_accuracy: 0.8619
Epoch 19/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0491 -
accuracy: 0.9850 - val_loss: 0.6528 - val_accuracy: 0.8614
Epoch 20/50
704/704 [==============================] - 11s 16ms/step - loss: 0.0444 -
accuracy: 0.9861 - val_loss: 0.6696 - val_accuracy: 0.8606
Epoch 21/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0426 -
accuracy: 0.9868 - val_loss: 0.6922 - val_accuracy: 0.8599
Epoch 22/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0384 -
accuracy: 0.9879 - val_loss: 0.7317 - val_accuracy: 0.8574
Epoch 23/50
704/704 [==============================] - 11s 16ms/step - loss: 0.0360 -
accuracy: 0.9885 - val_loss: 0.7209 - val_accuracy: 0.8588
Epoch 24/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0339 -
accuracy: 0.9890 - val_loss: 0.7728 - val_accuracy: 0.8571
Epoch 25/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0307 -
accuracy: 0.9899 - val_loss: 0.7984 - val_accuracy: 0.8563
Epoch 26/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0289 -
```

```
accuracy: 0.9903 - val_loss: 0.7986 - val_accuracy: 0.8554
Epoch 27/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0271 -
accuracy: 0.9909 - val_loss: 0.8503 - val_accuracy: 0.8535
Epoch 28/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0254 -
accuracy: 0.9914 - val_loss: 0.8455 - val_accuracy: 0.8545
Epoch 29/50
704/704 [==============================] - 11s 15ms/step - loss: 0.0236 -
accuracy: 0.9918 - val_loss: 0.8669 - val_accuracy: 0.8527
Epoch 30/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0231 -
accuracy: 0.9920 - val_loss: 0.8709 - val_accuracy: 0.8539
Epoch 31/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0205 -
accuracy: 0.9929 - val_loss: 0.9175 - val_accuracy: 0.8524
Epoch 32/50
704/704 [==============================] - 11s 16ms/step - loss: 0.0198 -
accuracy: 0.9931 - val_loss: 0.9230 - val_accuracy: 0.8516
Epoch 33/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0197 -
accuracy: 0.9932 - val_loss: 0.9545 - val_accuracy: 0.8480
Epoch 34/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0180 -
accuracy: 0.9937 - val_loss: 0.9567 - val_accuracy: 0.8506
Epoch 35/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0176 -
accuracy: 0.9937 - val_loss: 0.9765 - val_accuracy: 0.8504
Epoch 36/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0181 -
accuracy: 0.9937 - val_loss: 0.9881 - val_accuracy: 0.8492
Epoch 37/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0168 -
accuracy: 0.9940 - val_loss: 0.9765 - val_accuracy: 0.8487
Epoch 38/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0154 -
accuracy: 0.9944 - val_loss: 1.0020 - val_accuracy: 0.8474
Epoch 39/50
704/704 [==============================] - 11s 15ms/step - loss: 0.0145 -
accuracy: 0.9948 - val_loss: 1.0145 - val_accuracy: 0.8461
Epoch 40/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0168 -
accuracy: 0.9939 - val_loss: 0.9755 - val_accuracy: 0.8482
Epoch 41/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0145 -
accuracy: 0.9949 - val_loss: 1.0106 - val_accuracy: 0.8471
Epoch 42/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0130 -
```

```
accuracy: 0.9952 - val_loss: 1.0208 - val_accuracy: 0.8476
Epoch 43/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0130 -
accuracy: 0.9952 - val_loss: 1.0302 - val_accuracy: 0.8468
Epoch 44/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0122 -
accuracy: 0.9954 - val_loss: 1.0644 - val_accuracy: 0.8454
Epoch 45/50
704/704 [==============================] - 10s 15ms/step - loss: 0.0118 -
accuracy: 0.9953 - val_loss: 1.0756 - val_accuracy: 0.8442
Epoch 46/50
704/704 [==============================] - 9s 13ms/step - loss: 0.0116 -
accuracy: 0.9954 - val_loss: 1.0803 - val_accuracy: 0.8445
Epoch 47/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0113 -
accuracy: 0.9956 - val_loss: 1.1005 - val_accuracy: 0.8460
Epoch 48/50
704/704 [==============================] - 11s 15ms/step - loss: 0.0117 -
accuracy: 0.9954 - val_loss: 1.0991 - val_accuracy: 0.8451
Epoch 49/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0111 -
accuracy: 0.9956 - val_loss: 1.1379 - val_accuracy: 0.8426
Epoch 50/50
704/704 [==============================] - 10s 14ms/step - loss: 0.0102 -
accuracy: 0.9959 - val_loss: 1.1519 - val_accuracy: 0.8430
```

4.2 Inital CNN model

[68]:
```
cnn1_model_token = Model_Seq([vectorize_layer,
       layers.Embedding(
        input_dim=word_count,
        output_dim=256,
        mask_zero=True),
     layers.Conv1D(256, 3, activation = 'relu', padding = 'valid'),
     layers.GlobalMaxPooling1D(),
     layers.Dense(4, activation='softmax')], name = 'cnn1_model_token', X_train␣
↪= X_train, y_train = y_train, X_val = X_val, y_val = y_val, inital_lr=.00005)
```

[69]:
```
cnn1_model_token.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 48s 62ms/step - loss: 1.2380 -
accuracy: 0.6844 - val_loss: 0.9152 - val_accuracy: 0.8235
Epoch 2/50
704/704 [==============================] - 9s 13ms/step - loss: 0.5798 -
accuracy: 0.8581 - val_loss: 0.4227 - val_accuracy: 0.8749
Epoch 3/50
704/704 [==============================] - 9s 13ms/step - loss: 0.3561 -
```

```
accuracy: 0.8906 - val_loss: 0.3513 - val_accuracy: 0.8880
Epoch 4/50
704/704 [==============================] - 9s 12ms/step - loss: 0.2948 -
accuracy: 0.9069 - val_loss: 0.3237 - val_accuracy: 0.8952
Epoch 5/50
704/704 [==============================] - 8s 11ms/step - loss: 0.2562 -
accuracy: 0.9186 - val_loss: 0.3088 - val_accuracy: 0.8997
Epoch 6/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2260 -
accuracy: 0.9277 - val_loss: 0.2994 - val_accuracy: 0.9024
Epoch 7/50
704/704 [==============================] - 8s 12ms/step - loss: 0.2004 -
accuracy: 0.9363 - val_loss: 0.2952 - val_accuracy: 0.9030
Epoch 8/50
704/704 [==============================] - 7s 10ms/step - loss: 0.1779 -
accuracy: 0.9437 - val_loss: 0.2940 - val_accuracy: 0.9037
Epoch 9/50
704/704 [==============================] - 8s 11ms/step - loss: 0.1579 -
accuracy: 0.9499 - val_loss: 0.2958 - val_accuracy: 0.9026
Epoch 10/50
704/704 [==============================] - 7s 10ms/step - loss: 0.1400 -
accuracy: 0.9560 - val_loss: 0.2989 - val_accuracy: 0.9026
Epoch 11/50
704/704 [==============================] - 8s 11ms/step - loss: 0.1239 -
accuracy: 0.9613 - val_loss: 0.3052 - val_accuracy: 0.9008
Epoch 12/50
704/704 [==============================] - 7s 10ms/step - loss: 0.1095 -
accuracy: 0.9662 - val_loss: 0.3123 - val_accuracy: 0.9002
Epoch 13/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0967 -
accuracy: 0.9705 - val_loss: 0.3211 - val_accuracy: 0.8982
Epoch 14/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0852 -
accuracy: 0.9739 - val_loss: 0.3318 - val_accuracy: 0.8963
Epoch 15/50
704/704 [==============================] - 7s 11ms/step - loss: 0.0751 -
accuracy: 0.9776 - val_loss: 0.3430 - val_accuracy: 0.8948
Epoch 16/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0661 -
accuracy: 0.9802 - val_loss: 0.3565 - val_accuracy: 0.8928
Epoch 17/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0582 -
accuracy: 0.9831 - val_loss: 0.3699 - val_accuracy: 0.8918
Epoch 18/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0512 -
accuracy: 0.9848 - val_loss: 0.3847 - val_accuracy: 0.8914
Epoch 19/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0450 -
```

```
accuracy: 0.9865 - val_loss: 0.4006 - val_accuracy: 0.8890
Epoch 20/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0397 -
accuracy: 0.9883 - val_loss: 0.4168 - val_accuracy: 0.8881
Epoch 21/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0350 -
accuracy: 0.9897 - val_loss: 0.4336 - val_accuracy: 0.8871
Epoch 22/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0309 -
accuracy: 0.9908 - val_loss: 0.4510 - val_accuracy: 0.8866
Epoch 23/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0275 -
accuracy: 0.9917 - val_loss: 0.4683 - val_accuracy: 0.8843
Epoch 24/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0244 -
accuracy: 0.9927 - val_loss: 0.4860 - val_accuracy: 0.8841
Epoch 25/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0219 -
accuracy: 0.9932 - val_loss: 0.5038 - val_accuracy: 0.8826
Epoch 26/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0198 -
accuracy: 0.9935 - val_loss: 0.5224 - val_accuracy: 0.8816
Epoch 27/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0180 -
accuracy: 0.9941 - val_loss: 0.5389 - val_accuracy: 0.8807
Epoch 28/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0164 -
accuracy: 0.9945 - val_loss: 0.5556 - val_accuracy: 0.8801
Epoch 29/50
704/704 [==============================] - 8s 11ms/step - loss: 0.0151 -
accuracy: 0.9947 - val_loss: 0.5725 - val_accuracy: 0.8792
Epoch 30/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0140 -
accuracy: 0.9946 - val_loss: 0.5890 - val_accuracy: 0.8782
Epoch 31/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0130 -
accuracy: 0.9952 - val_loss: 0.6042 - val_accuracy: 0.8791
Epoch 32/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0122 -
accuracy: 0.9954 - val_loss: 0.6195 - val_accuracy: 0.8778
Epoch 33/50
704/704 [==============================] - 8s 12ms/step - loss: 0.0116 -
accuracy: 0.9952 - val_loss: 0.6349 - val_accuracy: 0.8770
Epoch 34/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0109 -
accuracy: 0.9954 - val_loss: 0.6492 - val_accuracy: 0.8770
Epoch 35/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0104 -
```

```
accuracy: 0.9957 - val_loss: 0.6643 - val_accuracy: 0.8758
Epoch 36/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0099 -
accuracy: 0.9955 - val_loss: 0.6777 - val_accuracy: 0.8756
Epoch 37/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0095 -
accuracy: 0.9958 - val_loss: 0.6912 - val_accuracy: 0.8750
Epoch 38/50
704/704 [==============================] - 7s 11ms/step - loss: 0.0092 -
accuracy: 0.9958 - val_loss: 0.7035 - val_accuracy: 0.8750
Epoch 39/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0089 -
accuracy: 0.9959 - val_loss: 0.7160 - val_accuracy: 0.8739
Epoch 40/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0086 -
accuracy: 0.9958 - val_loss: 0.7286 - val_accuracy: 0.8735
Epoch 41/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0084 -
accuracy: 0.9962 - val_loss: 0.7408 - val_accuracy: 0.8731
Epoch 42/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0081 -
accuracy: 0.9960 - val_loss: 0.7531 - val_accuracy: 0.8734
Epoch 43/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0079 -
accuracy: 0.9960 - val_loss: 0.7629 - val_accuracy: 0.8728
Epoch 44/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0077 -
accuracy: 0.9963 - val_loss: 0.7741 - val_accuracy: 0.8728
Epoch 45/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0075 -
accuracy: 0.9961 - val_loss: 0.7848 - val_accuracy: 0.8727
Epoch 46/50
704/704 [==============================] - 6s 9ms/step - loss: 0.0074 -
accuracy: 0.9962 - val_loss: 0.7964 - val_accuracy: 0.8714
Epoch 47/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0073 -
accuracy: 0.9962 - val_loss: 0.8058 - val_accuracy: 0.8719
Epoch 48/50
704/704 [==============================] - 7s 9ms/step - loss: 0.0072 -
accuracy: 0.9963 - val_loss: 0.8183 - val_accuracy: 0.8707
Epoch 49/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0070 -
accuracy: 0.9964 - val_loss: 0.8247 - val_accuracy: 0.8704
Epoch 50/50
704/704 [==============================] - 7s 10ms/step - loss: 0.0069 -
accuracy: 0.9963 - val_loss: 0.8348 - val_accuracy: 0.8705
```

[70]:
```
cnn1_model_word_vec = Model_Seq([vectorize_layer,
       layers.Embedding(nb_words,
                    embed_size,
                    weights=[embedding_matrix],
                    trainable=False),
     layers.Conv1D(256, 3, activation = 'relu', padding = 'valid'),
     layers.GlobalMaxPooling1D(),
     layers.Dense(4, activation='softmax')], name = 'CNN_with_WordVec', X_train␣
 ↪= X_train, y_train = y_train, X_val = X_val, y_val = y_val, inital_lr=.00005)
```

[71]:
```
cnn1_model_word_vec.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 6s 8ms/step - loss: 0.5097 -
accuracy: 0.8244 - val_loss: 0.4351 - val_accuracy: 0.8498
Epoch 2/50
704/704 [==============================] - 5s 8ms/step - loss: 0.4073 -
accuracy: 0.8598 - val_loss: 0.4113 - val_accuracy: 0.8571
Epoch 3/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3833 -
accuracy: 0.8681 - val_loss: 0.3984 - val_accuracy: 0.8629
Epoch 4/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3678 -
accuracy: 0.8738 - val_loss: 0.3895 - val_accuracy: 0.8653
Epoch 5/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3557 -
accuracy: 0.8787 - val_loss: 0.3829 - val_accuracy: 0.8675
Epoch 6/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3459 -
accuracy: 0.8821 - val_loss: 0.3784 - val_accuracy: 0.8685
Epoch 7/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3373 -
accuracy: 0.8847 - val_loss: 0.3756 - val_accuracy: 0.8696
Epoch 8/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3297 -
accuracy: 0.8873 - val_loss: 0.3729 - val_accuracy: 0.8695
Epoch 9/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3230 -
accuracy: 0.8899 - val_loss: 0.3692 - val_accuracy: 0.8712
Epoch 10/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3167 -
accuracy: 0.8923 - val_loss: 0.3655 - val_accuracy: 0.8741
Epoch 11/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3109 -
accuracy: 0.8948 - val_loss: 0.3650 - val_accuracy: 0.8724
Epoch 12/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3054 -
```

```
accuracy: 0.8962 - val_loss: 0.3614 - val_accuracy: 0.8754
Epoch 13/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3005 -
accuracy: 0.8983 - val_loss: 0.3631 - val_accuracy: 0.8739
Epoch 14/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2957 -
accuracy: 0.9000 - val_loss: 0.3591 - val_accuracy: 0.8764
Epoch 15/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2910 -
accuracy: 0.9019 - val_loss: 0.3579 - val_accuracy: 0.8757
Epoch 16/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2865 -
accuracy: 0.9033 - val_loss: 0.3568 - val_accuracy: 0.8776
Epoch 17/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2821 -
accuracy: 0.9046 - val_loss: 0.3563 - val_accuracy: 0.8761
Epoch 18/50
704/704 [==============================] - 5s 8ms/step - loss: 0.2781 -
accuracy: 0.9063 - val_loss: 0.3558 - val_accuracy: 0.8767
Epoch 19/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2742 -
accuracy: 0.9076 - val_loss: 0.3562 - val_accuracy: 0.8750
Epoch 20/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2702 -
accuracy: 0.9088 - val_loss: 0.3533 - val_accuracy: 0.8780
Epoch 21/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2666 -
accuracy: 0.9101 - val_loss: 0.3531 - val_accuracy: 0.8771
Epoch 22/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2628 -
accuracy: 0.9114 - val_loss: 0.3531 - val_accuracy: 0.8765
Epoch 23/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2593 -
accuracy: 0.9128 - val_loss: 0.3534 - val_accuracy: 0.8773
Epoch 24/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2560 -
accuracy: 0.9141 - val_loss: 0.3527 - val_accuracy: 0.8766
Epoch 25/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2525 -
accuracy: 0.9149 - val_loss: 0.3538 - val_accuracy: 0.8764
Epoch 26/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2492 -
accuracy: 0.9161 - val_loss: 0.3525 - val_accuracy: 0.8779
Epoch 27/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2460 -
accuracy: 0.9173 - val_loss: 0.3518 - val_accuracy: 0.8785
Epoch 28/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2429 -
```

```
accuracy: 0.9184 - val_loss: 0.3530 - val_accuracy: 0.8772
Epoch 29/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2400 -
accuracy: 0.9192 - val_loss: 0.3549 - val_accuracy: 0.8772
Epoch 30/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2368 -
accuracy: 0.9204 - val_loss: 0.3539 - val_accuracy: 0.8784
Epoch 31/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2337 -
accuracy: 0.9216 - val_loss: 0.3527 - val_accuracy: 0.8778
Epoch 32/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2308 -
accuracy: 0.9228 - val_loss: 0.3536 - val_accuracy: 0.8785
Epoch 33/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2284 -
accuracy: 0.9240 - val_loss: 0.3529 - val_accuracy: 0.8772
Epoch 34/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2250 -
accuracy: 0.9247 - val_loss: 0.3551 - val_accuracy: 0.8767
Epoch 35/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2224 -
accuracy: 0.9260 - val_loss: 0.3545 - val_accuracy: 0.8776
Epoch 36/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2196 -
accuracy: 0.9269 - val_loss: 0.3554 - val_accuracy: 0.8765
Epoch 37/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2172 -
accuracy: 0.9278 - val_loss: 0.3548 - val_accuracy: 0.8766
Epoch 38/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2144 -
accuracy: 0.9289 - val_loss: 0.3549 - val_accuracy: 0.8769
Epoch 39/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2117 -
accuracy: 0.9298 - val_loss: 0.3563 - val_accuracy: 0.8769
Epoch 40/50
704/704 [==============================] - 5s 8ms/step - loss: 0.2093 -
accuracy: 0.9311 - val_loss: 0.3573 - val_accuracy: 0.8754
Epoch 41/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2069 -
accuracy: 0.9321 - val_loss: 0.3581 - val_accuracy: 0.8748
Epoch 42/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2046 -
accuracy: 0.9326 - val_loss: 0.3571 - val_accuracy: 0.8765
Epoch 43/50
704/704 [==============================] - 5s 7ms/step - loss: 0.2021 -
accuracy: 0.9332 - val_loss: 0.3577 - val_accuracy: 0.8767
Epoch 44/50
704/704 [==============================] - 5s 7ms/step - loss: 0.1995 -
```

```
accuracy: 0.9347 - val_loss: 0.3599 - val_accuracy: 0.8746
Epoch 45/50
704/704 [==============================] - 5s 7ms/step - loss: 0.1973 -
accuracy: 0.9356 - val_loss: 0.3593 - val_accuracy: 0.8747
Epoch 46/50
704/704 [==============================] - 5s 7ms/step - loss: 0.1949 -
accuracy: 0.9361 - val_loss: 0.3626 - val_accuracy: 0.8736
Epoch 47/50
704/704 [==============================] - 5s 8ms/step - loss: 0.1926 -
accuracy: 0.9380 - val_loss: 0.3620 - val_accuracy: 0.8749
Epoch 48/50
704/704 [==============================] - 5s 7ms/step - loss: 0.1905 -
accuracy: 0.9380 - val_loss: 0.3625 - val_accuracy: 0.8727
Epoch 49/50
704/704 [==============================] - 5s 7ms/step - loss: 0.1881 -
accuracy: 0.9393 - val_loss: 0.3636 - val_accuracy: 0.8741
Epoch 50/50
704/704 [==============================] - 5s 7ms/step - loss: 0.1862 -
accuracy: 0.9401 - val_loss: 0.3646 - val_accuracy: 0.8746
```

[30]:

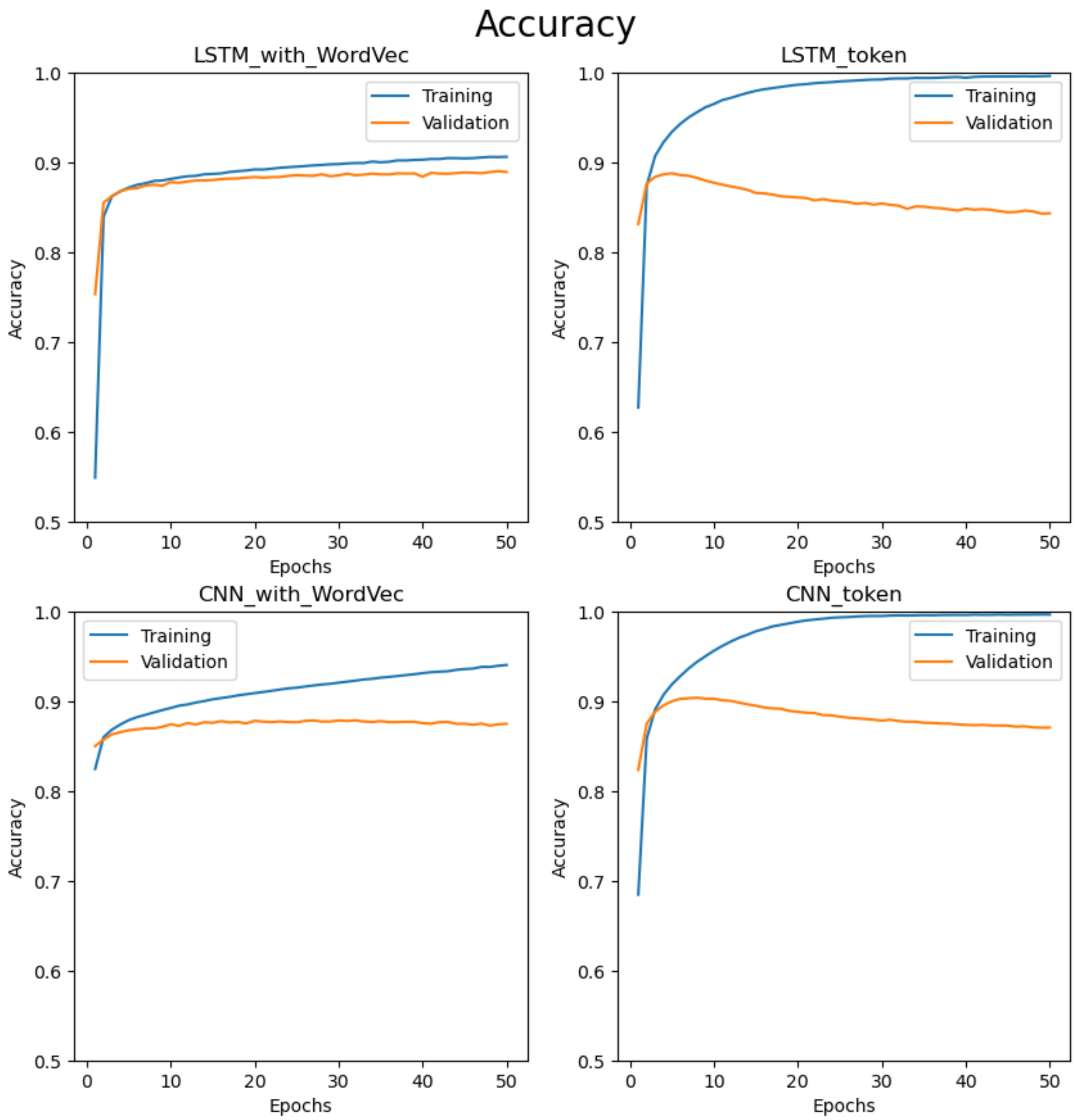
```
# Save the stats to disk incase the session ends
json_out(lstm_wordvec_model.stats|ltsm_1_token_model.stats |
 ↪cnn1_model_word_vec.stats | cnn1_model_token.stats, 'all_run1')
initial_stats = json_to_dict('all_run1.json')
```

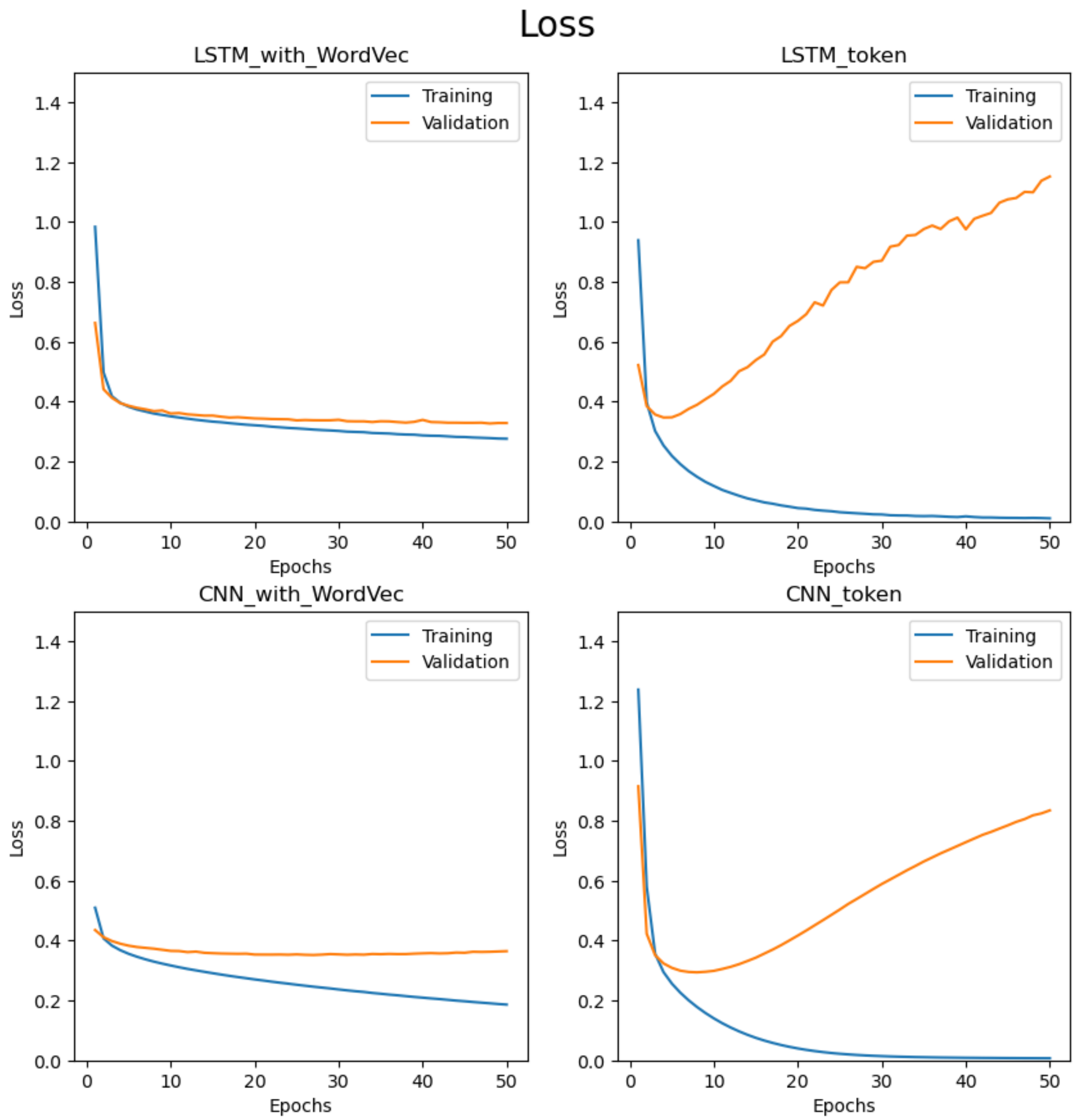
4.2.1 Results

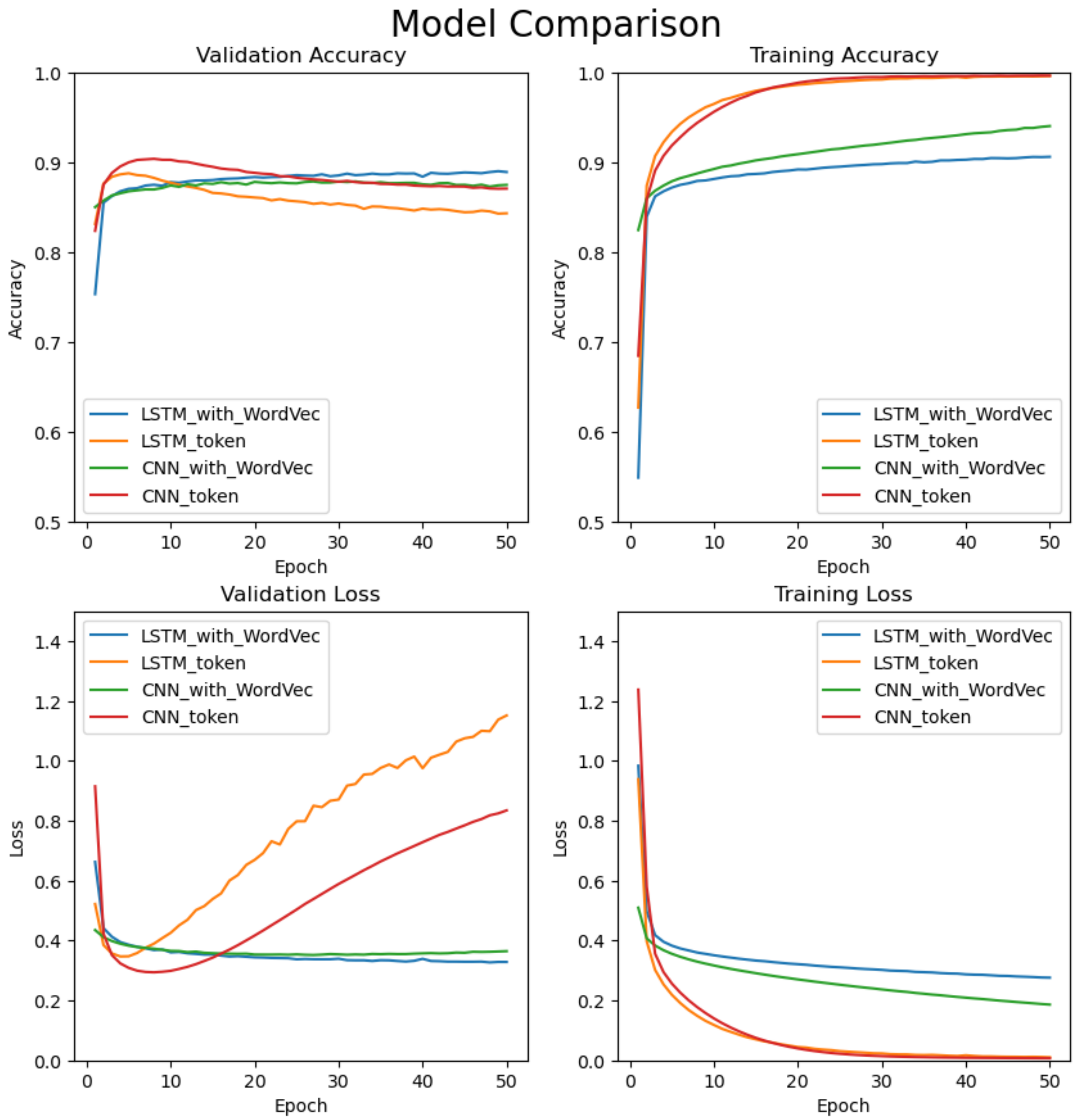
In looking at the graphs and the summary table, there is not a huge difference in the overall accuracy of the validation data between all four of the initial models. The most interesting thing about the WordVec models is that they don't begin to rapidly converge and overfit. Instead, the training loss slowly diverges from the validation loss. I suspect this is because since the emebeding layer not trainable when it based the WordVecModel, it is much harder to overfit the data, since it shouldn't be able to overfit one of the layers.

[22]:

```
plot_all_plots(initial_stats, plot_type = 'accuracy', cols = 2)
plot_all_plots(initial_stats, plot_type = 'loss', cols = 2)
plot_models_together(initial_stats)
create_summary_table(initial_stats)
```


Accuracy
LSTM_with_WordVec
LSTM_token
CNN_with_WordVec
CNN_token
Training
Validation
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50


Loss
LSTM_with_WordVec
LSTM_token
CNN_with_WordVec
CNN_token
Training
Validation
Loss
Epochs



	Model	Max Acc	Minnimum Loss	Max Val Acc	Min Val Loss	\
0	LSTM_with_WordVec	0.906	0.276	0.890	0.327	
1	LSTM_token	0.996	0.010	0.888	0.347	
2	CNN_with_WordVec	0.940	0.186	0.878	0.352	
3	CNN_token	0.996	0.007	0.904	0.294	

	Epoch of Min Val Loss
0	47
1	3
2	26
3	7

5 Hyperparemeter Tuning

After looking at the models, I decided to do some hyperparameter tuning on both the LSTM and CNN WordVec models. I chose the WordVec models to tune, since they didn't have the same propensity to rapidly overfit, compared to the more traditional text vectorization. I chose to use both the LSTM and CNN models because in the initial tests there wasn't a huge difference, and I wanted to see if I could determine if one was better than the other. ## LSTM Tuning For the LSTM tuning, I decided to try changing the number of units of the dense layers in the model to see if increasing them would have an effect on the overall accuracy of the model. I also tried adjusting the number of units of the LSTM layer to see if that would change anything. ### Dense Layer 128

[74]:
```
lstm_wordvec_model_hp1 = Model_Seq([vectorize_layer,
          layers.Embedding(nb_words,
                   embed_size,
                   weights=[embedding_matrix],
                   trainable=False),
          layers.LSTM(64),
        layers.Dense(128, activation='relu'),
        layers.Dense(4,  activation='softmax')
         ], name = 'LSTM_WordVec_Dense_128', X_train = X_train, y_train =␣
 ↪y_train, X_val = X_val, y_val = y_val, inital_lr=.0001)
```

[75]:
```
lstm_wordvec_model_hp1.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 10s 10ms/step - loss: 0.7200 -
accuracy: 0.7166 - val_loss: 0.4572 - val_accuracy: 0.8516
Epoch 2/50
704/704 [==============================] - 6s 9ms/step - loss: 0.4184 -
accuracy: 0.8616 - val_loss: 0.4047 - val_accuracy: 0.8652
Epoch 3/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3840 -
accuracy: 0.8705 - val_loss: 0.3843 - val_accuracy: 0.8708
Epoch 4/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3677 -
accuracy: 0.8759 - val_loss: 0.3787 - val_accuracy: 0.8716
Epoch 5/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3560 -
accuracy: 0.8791 - val_loss: 0.3640 - val_accuracy: 0.8775
Epoch 6/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3476 -
accuracy: 0.8822 - val_loss: 0.3559 - val_accuracy: 0.8792
Epoch 7/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3409 -
accuracy: 0.8840 - val_loss: 0.3549 - val_accuracy: 0.8807
Epoch 8/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3330 -
```

```
accuracy: 0.8867 - val_loss: 0.3497 - val_accuracy: 0.8818
Epoch 9/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3279 -
accuracy: 0.8884 - val_loss: 0.3452 - val_accuracy: 0.8835
Epoch 10/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3223 -
accuracy: 0.8894 - val_loss: 0.3398 - val_accuracy: 0.8855
Epoch 11/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3179 -
accuracy: 0.8914 - val_loss: 0.3408 - val_accuracy: 0.8832
Epoch 12/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3137 -
accuracy: 0.8924 - val_loss: 0.3347 - val_accuracy: 0.8861
Epoch 13/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3096 -
accuracy: 0.8935 - val_loss: 0.3362 - val_accuracy: 0.8860
Epoch 14/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3056 -
accuracy: 0.8946 - val_loss: 0.3317 - val_accuracy: 0.8864
Epoch 15/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3034 -
accuracy: 0.8943 - val_loss: 0.3305 - val_accuracy: 0.8868
Epoch 16/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2993 -
accuracy: 0.8961 - val_loss: 0.3275 - val_accuracy: 0.8881
Epoch 17/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2965 -
accuracy: 0.8979 - val_loss: 0.3278 - val_accuracy: 0.8878
Epoch 18/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2935 -
accuracy: 0.8987 - val_loss: 0.3291 - val_accuracy: 0.8863
Epoch 19/50
704/704 [==============================] - 7s 11ms/step - loss: 0.2908 -
accuracy: 0.8989 - val_loss: 0.3295 - val_accuracy: 0.8865
Epoch 20/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2884 -
accuracy: 0.9002 - val_loss: 0.3315 - val_accuracy: 0.8856
Epoch 21/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2875 -
accuracy: 0.9000 - val_loss: 0.3255 - val_accuracy: 0.8883
Epoch 22/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2840 -
accuracy: 0.9013 - val_loss: 0.3228 - val_accuracy: 0.8905
Epoch 23/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2822 -
accuracy: 0.9027 - val_loss: 0.3252 - val_accuracy: 0.8861
Epoch 24/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2793 -
```

```
accuracy: 0.9026 - val_loss: 0.3265 - val_accuracy: 0.8876
Epoch 25/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2788 -
accuracy: 0.9029 - val_loss: 0.3196 - val_accuracy: 0.8897
Epoch 26/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2765 -
accuracy: 0.9043 - val_loss: 0.3233 - val_accuracy: 0.8897
Epoch 27/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2743 -
accuracy: 0.9049 - val_loss: 0.3227 - val_accuracy: 0.8895
Epoch 28/50
704/704 [==============================] - 8s 12ms/step - loss: 0.2719 -
accuracy: 0.9051 - val_loss: 0.3321 - val_accuracy: 0.8865
Epoch 29/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2710 -
accuracy: 0.9057 - val_loss: 0.3244 - val_accuracy: 0.8873
Epoch 30/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2691 -
accuracy: 0.9065 - val_loss: 0.3296 - val_accuracy: 0.8865
Epoch 31/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2668 -
accuracy: 0.9086 - val_loss: 0.3198 - val_accuracy: 0.8901
Epoch 32/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2654 -
accuracy: 0.9082 - val_loss: 0.3238 - val_accuracy: 0.8887
Epoch 33/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2640 -
accuracy: 0.9081 - val_loss: 0.3233 - val_accuracy: 0.8899
Epoch 34/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2619 -
accuracy: 0.9095 - val_loss: 0.3212 - val_accuracy: 0.8902
Epoch 35/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2603 -
accuracy: 0.9100 - val_loss: 0.3215 - val_accuracy: 0.8890
Epoch 36/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2595 -
accuracy: 0.9100 - val_loss: 0.3289 - val_accuracy: 0.8865
Epoch 37/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2568 -
accuracy: 0.9111 - val_loss: 0.3266 - val_accuracy: 0.8882
Epoch 38/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2554 -
accuracy: 0.9117 - val_loss: 0.3207 - val_accuracy: 0.8899
Epoch 39/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2542 -
accuracy: 0.9119 - val_loss: 0.3230 - val_accuracy: 0.8902
Epoch 40/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2523 -
```

```
accuracy: 0.9131 - val_loss: 0.3338 - val_accuracy: 0.8862
Epoch 41/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2519 -
accuracy: 0.9131 - val_loss: 0.3267 - val_accuracy: 0.8888
Epoch 42/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2495 -
accuracy: 0.9140 - val_loss: 0.3303 - val_accuracy: 0.8886
Epoch 43/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2490 -
accuracy: 0.9140 - val_loss: 0.3280 - val_accuracy: 0.8878
Epoch 44/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2466 -
accuracy: 0.9150 - val_loss: 0.3295 - val_accuracy: 0.8884
Epoch 45/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2451 -
accuracy: 0.9150 - val_loss: 0.3302 - val_accuracy: 0.8877
Epoch 46/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2437 -
accuracy: 0.9162 - val_loss: 0.3309 - val_accuracy: 0.8886
Epoch 47/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2424 -
accuracy: 0.9160 - val_loss: 0.3285 - val_accuracy: 0.8908
Epoch 48/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2412 -
accuracy: 0.9160 - val_loss: 0.3292 - val_accuracy: 0.8880
Epoch 49/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2405 -
accuracy: 0.9174 - val_loss: 0.3286 - val_accuracy: 0.8885
Epoch 50/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2384 -
accuracy: 0.9175 - val_loss: 0.3278 - val_accuracy: 0.8907
```

5.0.1 Dense Layer 512

[76]:
```
lstm_wordvec_model_hp2 = Model_Seq([vectorize_layer,
          layers.Embedding(nb_words,
                   embed_size,
                   weights=[embedding_matrix],
                   trainable=False),
          layers.LSTM(64),
        layers.Dense(512, activation='relu'),
        layers.Dense(4,  activation='softmax')
         ], name = 'LSTM_WordVec_Dense_512', X_train = X_train, y_train =␣
 ↪y_train, X_val = X_val, y_val = y_val, inital_lr=.0001)
```

[77]:
```
lstm_wordvec_model_hp2.train_model(50, batch_size = 128)
```

```
Epoch 1/50
```

```
704/704 [==============================] - 9s 10ms/step - loss: 0.7273 -
accuracy: 0.6756 - val_loss: 0.4733 - val_accuracy: 0.8427
Epoch 2/50
704/704 [==============================] - 7s 10ms/step - loss: 0.4175 -
accuracy: 0.8614 - val_loss: 0.4061 - val_accuracy: 0.8656
Epoch 3/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3791 -
accuracy: 0.8737 - val_loss: 0.3812 - val_accuracy: 0.8714
Epoch 4/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3632 -
accuracy: 0.8788 - val_loss: 0.3737 - val_accuracy: 0.8741
Epoch 5/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3512 -
accuracy: 0.8825 - val_loss: 0.3618 - val_accuracy: 0.8778
Epoch 6/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3416 -
accuracy: 0.8848 - val_loss: 0.3505 - val_accuracy: 0.8806
Epoch 7/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3356 -
accuracy: 0.8859 - val_loss: 0.3495 - val_accuracy: 0.8816
Epoch 8/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3280 -
accuracy: 0.8889 - val_loss: 0.3432 - val_accuracy: 0.8828
Epoch 9/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3236 -
accuracy: 0.8894 - val_loss: 0.3403 - val_accuracy: 0.8835
Epoch 10/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3188 -
accuracy: 0.8913 - val_loss: 0.3362 - val_accuracy: 0.8844
Epoch 11/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3141 -
accuracy: 0.8926 - val_loss: 0.3377 - val_accuracy: 0.8845
Epoch 12/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3100 -
accuracy: 0.8937 - val_loss: 0.3322 - val_accuracy: 0.8857
Epoch 13/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3068 -
accuracy: 0.8938 - val_loss: 0.3364 - val_accuracy: 0.8833
Epoch 14/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3031 -
accuracy: 0.8957 - val_loss: 0.3331 - val_accuracy: 0.8861
Epoch 15/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3009 -
accuracy: 0.8955 - val_loss: 0.3286 - val_accuracy: 0.8887
Epoch 16/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2976 -
accuracy: 0.8971 - val_loss: 0.3287 - val_accuracy: 0.8876
Epoch 17/50
```

```
704/704 [==============================] - 7s 10ms/step - loss: 0.2953 -
accuracy: 0.8986 - val_loss: 0.3266 - val_accuracy: 0.8878
Epoch 18/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2922 -
accuracy: 0.8998 - val_loss: 0.3295 - val_accuracy: 0.8885
Epoch 19/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2889 -
accuracy: 0.9011 - val_loss: 0.3258 - val_accuracy: 0.8892
Epoch 20/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2872 -
accuracy: 0.9007 - val_loss: 0.3291 - val_accuracy: 0.8881
Epoch 21/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2848 -
accuracy: 0.9015 - val_loss: 0.3280 - val_accuracy: 0.8882
Epoch 22/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2823 -
accuracy: 0.9025 - val_loss: 0.3223 - val_accuracy: 0.8904
Epoch 23/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2806 -
accuracy: 0.9029 - val_loss: 0.3217 - val_accuracy: 0.8892
Epoch 24/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2768 -
accuracy: 0.9039 - val_loss: 0.3253 - val_accuracy: 0.8879
Epoch 25/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2778 -
accuracy: 0.9035 - val_loss: 0.3197 - val_accuracy: 0.8918
Epoch 26/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2746 -
accuracy: 0.9054 - val_loss: 0.3195 - val_accuracy: 0.8906
Epoch 27/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2723 -
accuracy: 0.9057 - val_loss: 0.3237 - val_accuracy: 0.8913
Epoch 28/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2701 -
accuracy: 0.9063 - val_loss: 0.3277 - val_accuracy: 0.8893
Epoch 29/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2686 -
accuracy: 0.9068 - val_loss: 0.3229 - val_accuracy: 0.8900
Epoch 30/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2675 -
accuracy: 0.9074 - val_loss: 0.3237 - val_accuracy: 0.8907
Epoch 31/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2648 -
accuracy: 0.9081 - val_loss: 0.3214 - val_accuracy: 0.8904
Epoch 32/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2632 -
accuracy: 0.9091 - val_loss: 0.3243 - val_accuracy: 0.8896
Epoch 33/50
```

```
704/704 [==============================] - 6s 9ms/step - loss: 0.2618 -
accuracy: 0.9091 - val_loss: 0.3241 - val_accuracy: 0.8899
Epoch 34/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2607 -
accuracy: 0.9092 - val_loss: 0.3200 - val_accuracy: 0.8914
Epoch 35/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2586 -
accuracy: 0.9098 - val_loss: 0.3191 - val_accuracy: 0.8922
Epoch 36/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2580 -
accuracy: 0.9107 - val_loss: 0.3263 - val_accuracy: 0.8888
Epoch 37/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2555 -
accuracy: 0.9109 - val_loss: 0.3259 - val_accuracy: 0.8892
Epoch 38/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2538 -
accuracy: 0.9118 - val_loss: 0.3210 - val_accuracy: 0.8909
Epoch 39/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2530 -
accuracy: 0.9122 - val_loss: 0.3250 - val_accuracy: 0.8905
Epoch 40/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2511 -
accuracy: 0.9129 - val_loss: 0.3344 - val_accuracy: 0.8892
Epoch 41/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2492 -
accuracy: 0.9134 - val_loss: 0.3229 - val_accuracy: 0.8899
Epoch 42/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2485 -
accuracy: 0.9135 - val_loss: 0.3259 - val_accuracy: 0.8913
Epoch 43/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2471 -
accuracy: 0.9142 - val_loss: 0.3246 - val_accuracy: 0.8900
Epoch 44/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2456 -
accuracy: 0.9147 - val_loss: 0.3264 - val_accuracy: 0.8911
Epoch 45/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2439 -
accuracy: 0.9152 - val_loss: 0.3258 - val_accuracy: 0.8901
Epoch 46/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2429 -
accuracy: 0.9158 - val_loss: 0.3254 - val_accuracy: 0.8898
Epoch 47/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2415 -
accuracy: 0.9158 - val_loss: 0.3243 - val_accuracy: 0.8925
Epoch 48/50
704/704 [==============================] - 7s 10ms/step - loss: 0.2401 -
accuracy: 0.9178 - val_loss: 0.3266 - val_accuracy: 0.8884
Epoch 49/50
```

```
704/704 [==============================] - 6s 9ms/step - loss: 0.2382 -
accuracy: 0.9178 - val_loss: 0.3248 - val_accuracy: 0.8914
Epoch 50/50
704/704 [==============================] - 7s 9ms/step - loss: 0.2374 -
accuracy: 0.9181 - val_loss: 0.3280 - val_accuracy: 0.8906
```

5.0.2 LSTM Units 256

[78]:
```
lstm_wordvec_model_hp3 = Model_Seq([vectorize_layer,
          layers.Embedding(nb_words,
                   embed_size,
                   weights=[embedding_matrix],
                   trainable=False),
          layers.LSTM(256),
        layers.Dense(128, activation='relu'),
        layers.Dense(4,  activation='softmax')
         ], name = 'LSTM_WordVec_LSTM_256', X_train = X_train, y_train =␣
 ↪y_train, X_val = X_val, y_val = y_val, inital_lr=.0001)
```

[79]:
```
lstm_wordvec_model_hp3.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 14s 16ms/step - loss: 0.5041 -
accuracy: 0.8197 - val_loss: 0.3860 - val_accuracy: 0.8685
Epoch 2/50
704/704 [==============================] - 11s 15ms/step - loss: 0.3703 -
accuracy: 0.8740 - val_loss: 0.3704 - val_accuracy: 0.8717
Epoch 3/50
704/704 [==============================] - 10s 14ms/step - loss: 0.3536 -
accuracy: 0.8790 - val_loss: 0.3662 - val_accuracy: 0.8729
Epoch 4/50
704/704 [==============================] - 10s 14ms/step - loss: 0.3417 -
accuracy: 0.8821 - val_loss: 0.3510 - val_accuracy: 0.8792
Epoch 5/50
704/704 [==============================] - 11s 15ms/step - loss: 0.3333 -
accuracy: 0.8832 - val_loss: 0.3405 - val_accuracy: 0.8823
Epoch 6/50
704/704 [==============================] - 11s 15ms/step - loss: 0.3234 -
accuracy: 0.8876 - val_loss: 0.3334 - val_accuracy: 0.8845
Epoch 7/50
704/704 [==============================] - 11s 16ms/step - loss: 0.3186 -
accuracy: 0.8891 - val_loss: 0.3426 - val_accuracy: 0.8835
Epoch 8/50
704/704 [==============================] - 10s 14ms/step - loss: 0.3102 -
accuracy: 0.8925 - val_loss: 0.3393 - val_accuracy: 0.8842
Epoch 9/50
704/704 [==============================] - 11s 15ms/step - loss: 0.3070 -
accuracy: 0.8928 - val_loss: 0.3483 - val_accuracy: 0.8836
```

```
Epoch 10/50
704/704 [==============================] - 11s 16ms/step - loss: 0.3019 -
accuracy: 0.8952 - val_loss: 0.3303 - val_accuracy: 0.8873
Epoch 11/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2957 -
accuracy: 0.8970 - val_loss: 0.3271 - val_accuracy: 0.8871
Epoch 12/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2899 -
accuracy: 0.8988 - val_loss: 0.3249 - val_accuracy: 0.8872
Epoch 13/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2847 -
accuracy: 0.9010 - val_loss: 0.3288 - val_accuracy: 0.8866
Epoch 14/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2791 -
accuracy: 0.9016 - val_loss: 0.3182 - val_accuracy: 0.8884
Epoch 15/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2753 -
accuracy: 0.9034 - val_loss: 0.3187 - val_accuracy: 0.8909
Epoch 16/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2702 -
accuracy: 0.9054 - val_loss: 0.3240 - val_accuracy: 0.8892
Epoch 17/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2649 -
accuracy: 0.9073 - val_loss: 0.3275 - val_accuracy: 0.8900
Epoch 18/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2602 -
accuracy: 0.9088 - val_loss: 0.3190 - val_accuracy: 0.8940
Epoch 19/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2560 -
accuracy: 0.9101 - val_loss: 0.3178 - val_accuracy: 0.8894
Epoch 20/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2507 -
accuracy: 0.9119 - val_loss: 0.3303 - val_accuracy: 0.8868
Epoch 21/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2452 -
accuracy: 0.9135 - val_loss: 0.3288 - val_accuracy: 0.8867
Epoch 22/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2415 -
accuracy: 0.9149 - val_loss: 0.3178 - val_accuracy: 0.8928
Epoch 23/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2363 -
accuracy: 0.9168 - val_loss: 0.3237 - val_accuracy: 0.8885
Epoch 24/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2300 -
accuracy: 0.9197 - val_loss: 0.3306 - val_accuracy: 0.8848
Epoch 25/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2273 -
accuracy: 0.9203 - val_loss: 0.3313 - val_accuracy: 0.8867
```

```
Epoch 26/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2218 -
accuracy: 0.9222 - val_loss: 0.3252 - val_accuracy: 0.8877
Epoch 27/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2167 -
accuracy: 0.9230 - val_loss: 0.3315 - val_accuracy: 0.8901
Epoch 28/50
704/704 [==============================] - 11s 15ms/step - loss: 0.2110 -
accuracy: 0.9265 - val_loss: 0.3447 - val_accuracy: 0.8880
Epoch 29/50
704/704 [==============================] - 11s 16ms/step - loss: 0.2073 -
accuracy: 0.9268 - val_loss: 0.3381 - val_accuracy: 0.8904
Epoch 30/50
704/704 [==============================] - 10s 14ms/step - loss: 0.2028 -
accuracy: 0.9288 - val_loss: 0.3451 - val_accuracy: 0.8869
Epoch 31/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1957 -
accuracy: 0.9306 - val_loss: 0.3381 - val_accuracy: 0.8895
Epoch 32/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1931 -
accuracy: 0.9319 - val_loss: 0.3629 - val_accuracy: 0.8872
Epoch 33/50
704/704 [==============================] - 10s 15ms/step - loss: 0.1892 -
accuracy: 0.9326 - val_loss: 0.3590 - val_accuracy: 0.8894
Epoch 34/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1826 -
accuracy: 0.9357 - val_loss: 0.3593 - val_accuracy: 0.8881
Epoch 35/50
704/704 [==============================] - 11s 16ms/step - loss: 0.1779 -
accuracy: 0.9375 - val_loss: 0.3774 - val_accuracy: 0.8851
Epoch 36/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1764 -
accuracy: 0.9380 - val_loss: 0.3719 - val_accuracy: 0.8877
Epoch 37/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1696 -
accuracy: 0.9405 - val_loss: 0.4123 - val_accuracy: 0.8804
Epoch 38/50
704/704 [==============================] - 11s 16ms/step - loss: 0.1648 -
accuracy: 0.9423 - val_loss: 0.3760 - val_accuracy: 0.8866
Epoch 39/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1648 -
accuracy: 0.9423 - val_loss: 0.3739 - val_accuracy: 0.8824
Epoch 40/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1567 -
accuracy: 0.9451 - val_loss: 0.4165 - val_accuracy: 0.8831
Epoch 41/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1541 -
accuracy: 0.9460 - val_loss: 0.3912 - val_accuracy: 0.8856
```

```
Epoch 42/50
704/704 [==============================] - 11s 15ms/step - loss: 0.1501 -
accuracy: 0.9473 - val_loss: 0.4021 - val_accuracy: 0.8863
Epoch 43/50
704/704 [==============================] - 11s 15ms/step - loss: 0.1474 -
accuracy: 0.9487 - val_loss: 0.3950 - val_accuracy: 0.8853
Epoch 44/50
704/704 [==============================] - 10s 15ms/step - loss: 0.1444 -
accuracy: 0.9493 - val_loss: 0.4251 - val_accuracy: 0.8845
Epoch 45/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1381 -
accuracy: 0.9509 - val_loss: 0.4326 - val_accuracy: 0.8847
Epoch 46/50
704/704 [==============================] - 11s 15ms/step - loss: 0.1363 -
accuracy: 0.9519 - val_loss: 0.4382 - val_accuracy: 0.8829
Epoch 47/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1300 -
accuracy: 0.9537 - val_loss: 0.4362 - val_accuracy: 0.8863
Epoch 48/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1277 -
accuracy: 0.9549 - val_loss: 0.4462 - val_accuracy: 0.8841
Epoch 49/50
704/704 [==============================] - 11s 15ms/step - loss: 0.1265 -
accuracy: 0.9550 - val_loss: 0.4802 - val_accuracy: 0.8807
Epoch 50/50
704/704 [==============================] - 10s 14ms/step - loss: 0.1233 -
accuracy: 0.9566 - val_loss: 0.4406 - val_accuracy: 0.8855
```

5.1 CNN

For the CNN model, I adjusted how many filters were used in the convolution layer, to see if more filters would have a positive impact on the model. ### Filters 256

[81]:
```
cnn_model_word_vec_hp1 = Model_Seq([vectorize_layer,
                                    layers.Embedding(nb_words,
                                        embed_size,
                                        weights=[embedding_matrix],
                                        trainable=False),
                                     layers.Conv1D(256, 3, activation = 'relu',␣
 ↪padding = 'valid'),
                                     layers.GlobalMaxPooling1D(),
                                     layers.Dense(4, activation='softmax')],␣
 ↪name = 'CNN_WordVec_filter_256', X_train = X_train, y_train = y_train, X_val␣
 ↪= X_val, y_val = y_val, inital_lr=.00001)

cnn_model_word_vec_hp1.summary()
```

```
Model: "CNN_WordVec_filter_256"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization_1 (TextV  (None, None)             0
 ectorization)

 embedding_8 (Embedding)     (None, None, 256)         13967360

 conv1d_3 (Conv1D)           (None, None, 256)         196864

 global_max_pooling1d_3 (Glo  (None, 256)              0
 balMaxPooling1D)

 dense_13 (Dense)            (None, 4)                 1028

=================================================================
Total params: 14,165,252
Trainable params: 197,892
Non-trainable params: 13,967,360
_________________________________________________________________
```

[82]:
```
cnn_model_word_vec_hp1.train_model(50, batch_size = 128)
```

```
Epoch 1/50
704/704 [==============================] - 6s 8ms/step - loss: 0.8038 -
accuracy: 0.7136 - val_loss: 0.5434 - val_accuracy: 0.8204
Epoch 2/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4986 -
accuracy: 0.8323 - val_loss: 0.4796 - val_accuracy: 0.8381
Epoch 3/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4586 -
accuracy: 0.8438 - val_loss: 0.4566 - val_accuracy: 0.8441
Epoch 4/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4392 -
accuracy: 0.8503 - val_loss: 0.4433 - val_accuracy: 0.8478
Epoch 5/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4260 -
accuracy: 0.8547 - val_loss: 0.4340 - val_accuracy: 0.8508
Epoch 6/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4157 -
accuracy: 0.8579 - val_loss: 0.4267 - val_accuracy: 0.8530
Epoch 7/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4074 -
accuracy: 0.8605 - val_loss: 0.4205 - val_accuracy: 0.8546
Epoch 8/50
704/704 [==============================] - 5s 7ms/step - loss: 0.4004 -
accuracy: 0.8631 - val_loss: 0.4158 - val_accuracy: 0.8560
```

```
Epoch 9/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3942 -
accuracy: 0.8654 - val_loss: 0.4118 - val_accuracy: 0.8571
Epoch 10/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3889 -
accuracy: 0.8667 - val_loss: 0.4077 - val_accuracy: 0.8582
Epoch 11/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3839 -
accuracy: 0.8684 - val_loss: 0.4049 - val_accuracy: 0.8594
Epoch 12/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3796 -
accuracy: 0.8697 - val_loss: 0.4016 - val_accuracy: 0.8596
Epoch 13/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3756 -
accuracy: 0.8716 - val_loss: 0.3988 - val_accuracy: 0.8607
Epoch 14/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3719 -
accuracy: 0.8725 - val_loss: 0.3968 - val_accuracy: 0.8613
Epoch 15/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3686 -
accuracy: 0.8735 - val_loss: 0.3946 - val_accuracy: 0.8625
Epoch 16/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3654 -
accuracy: 0.8747 - val_loss: 0.3926 - val_accuracy: 0.8627
Epoch 17/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3624 -
accuracy: 0.8757 - val_loss: 0.3908 - val_accuracy: 0.8636
Epoch 18/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3595 -
accuracy: 0.8766 - val_loss: 0.3893 - val_accuracy: 0.8642
Epoch 19/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3569 -
accuracy: 0.8776 - val_loss: 0.3880 - val_accuracy: 0.8640
Epoch 20/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3544 -
accuracy: 0.8789 - val_loss: 0.3863 - val_accuracy: 0.8647
Epoch 21/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3520 -
accuracy: 0.8793 - val_loss: 0.3851 - val_accuracy: 0.8656
Epoch 22/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3497 -
accuracy: 0.8807 - val_loss: 0.3838 - val_accuracy: 0.8662
Epoch 23/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3475 -
accuracy: 0.8814 - val_loss: 0.3823 - val_accuracy: 0.8669
Epoch 24/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3454 -
accuracy: 0.8820 - val_loss: 0.3813 - val_accuracy: 0.8673
```

```
Epoch 25/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3434 -
accuracy: 0.8826 - val_loss: 0.3800 - val_accuracy: 0.8677
Epoch 26/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3414 -
accuracy: 0.8835 - val_loss: 0.3793 - val_accuracy: 0.8677
Epoch 27/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3395 -
accuracy: 0.8842 - val_loss: 0.3780 - val_accuracy: 0.8683
Epoch 28/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3377 -
accuracy: 0.8845 - val_loss: 0.3773 - val_accuracy: 0.8685
Epoch 29/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3359 -
accuracy: 0.8854 - val_loss: 0.3768 - val_accuracy: 0.8684
Epoch 30/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3341 -
accuracy: 0.8862 - val_loss: 0.3754 - val_accuracy: 0.8697
Epoch 31/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3325 -
accuracy: 0.8870 - val_loss: 0.3748 - val_accuracy: 0.8696
Epoch 32/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3309 -
accuracy: 0.8874 - val_loss: 0.3742 - val_accuracy: 0.8694
Epoch 33/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3294 -
accuracy: 0.8878 - val_loss: 0.3733 - val_accuracy: 0.8699
Epoch 34/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3277 -
accuracy: 0.8884 - val_loss: 0.3724 - val_accuracy: 0.8699
Epoch 35/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3262 -
accuracy: 0.8893 - val_loss: 0.3719 - val_accuracy: 0.8700
Epoch 36/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3247 -
accuracy: 0.8897 - val_loss: 0.3713 - val_accuracy: 0.8701
Epoch 37/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3232 -
accuracy: 0.8902 - val_loss: 0.3706 - val_accuracy: 0.8712
Epoch 38/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3218 -
accuracy: 0.8907 - val_loss: 0.3699 - val_accuracy: 0.8714
Epoch 39/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3204 -
accuracy: 0.8911 - val_loss: 0.3692 - val_accuracy: 0.8709
Epoch 40/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3191 -
accuracy: 0.8918 - val_loss: 0.3687 - val_accuracy: 0.8715
```

```
Epoch 41/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3177 -
accuracy: 0.8921 - val_loss: 0.3684 - val_accuracy: 0.8711
Epoch 42/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3164 -
accuracy: 0.8926 - val_loss: 0.3675 - val_accuracy: 0.8719
Epoch 43/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3151 -
accuracy: 0.8931 - val_loss: 0.3669 - val_accuracy: 0.8720
Epoch 44/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3138 -
accuracy: 0.8935 - val_loss: 0.3666 - val_accuracy: 0.8717
Epoch 45/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3126 -
accuracy: 0.8937 - val_loss: 0.3660 - val_accuracy: 0.8718
Epoch 46/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3113 -
accuracy: 0.8949 - val_loss: 0.3658 - val_accuracy: 0.8722
Epoch 47/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3101 -
accuracy: 0.8951 - val_loss: 0.3651 - val_accuracy: 0.8727
Epoch 48/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3089 -
accuracy: 0.8952 - val_loss: 0.3648 - val_accuracy: 0.8729
Epoch 49/50
704/704 [==============================] - 5s 7ms/step - loss: 0.3077 -
accuracy: 0.8958 - val_loss: 0.3644 - val_accuracy: 0.8724
Epoch 50/50
704/704 [==============================] - 5s 8ms/step - loss: 0.3065 -
accuracy: 0.8964 - val_loss: 0.3637 - val_accuracy: 0.8730
```

5.1.1 Filters 512

[85]:
```
cnn_model_word_vec_hp2 = Model_Seq([vectorize_layer,
                                    layers.Embedding(nb_words,
                                        embed_size,
                                        weights=[embedding_matrix],
                                        trainable=False),
                                     layers.Conv1D(512, 3, activation = 'relu',␣
 ↪padding = 'valid'),
                                     layers.GlobalMaxPooling1D(),
                                     layers.Dense(4, activation='softmax')],␣
 ↪name = 'CNN_WordVec_filter_512', X_train = X_train, y_train = y_train, X_val␣
 ↪= X_val, y_val = y_val, inital_lr=.00001)
```

[86]:
```
cnn_model_word_vec_hp2.train_model(50, batch_size = 128)
```

```
Epoch 1/50
```

```
704/704 [==============================] - 13s 16ms/step - loss: 0.6839 -
accuracy: 0.7671 - val_loss: 0.4984 - val_accuracy: 0.8343
Epoch 2/50
704/704 [==============================] - 6s 9ms/step - loss: 0.4649 -
accuracy: 0.8435 - val_loss: 0.4551 - val_accuracy: 0.8438
Epoch 3/50
704/704 [==============================] - 6s 9ms/step - loss: 0.4341 -
accuracy: 0.8532 - val_loss: 0.4368 - val_accuracy: 0.8495
Epoch 4/50
704/704 [==============================] - 6s 8ms/step - loss: 0.4168 -
accuracy: 0.8579 - val_loss: 0.4251 - val_accuracy: 0.8526
Epoch 5/50
704/704 [==============================] - 6s 8ms/step - loss: 0.4044 -
accuracy: 0.8618 - val_loss: 0.4171 - val_accuracy: 0.8558
Epoch 6/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3946 -
accuracy: 0.8650 - val_loss: 0.4105 - val_accuracy: 0.8584
Epoch 7/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3865 -
accuracy: 0.8678 - val_loss: 0.4047 - val_accuracy: 0.8606
Epoch 8/50
704/704 [==============================] - 7s 10ms/step - loss: 0.3796 -
accuracy: 0.8698 - val_loss: 0.4008 - val_accuracy: 0.8610
Epoch 9/50
704/704 [==============================] - 8s 11ms/step - loss: 0.3736 -
accuracy: 0.8721 - val_loss: 0.3970 - val_accuracy: 0.8624
Epoch 10/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3682 -
accuracy: 0.8745 - val_loss: 0.3929 - val_accuracy: 0.8641
Epoch 11/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3632 -
accuracy: 0.8761 - val_loss: 0.3903 - val_accuracy: 0.8650
Epoch 12/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3587 -
accuracy: 0.8779 - val_loss: 0.3874 - val_accuracy: 0.8663
Epoch 13/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3547 -
accuracy: 0.8793 - val_loss: 0.3848 - val_accuracy: 0.8669
Epoch 14/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3509 -
accuracy: 0.8804 - val_loss: 0.3829 - val_accuracy: 0.8673
Epoch 15/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3474 -
accuracy: 0.8819 - val_loss: 0.3810 - val_accuracy: 0.8677
Epoch 16/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3440 -
accuracy: 0.8828 - val_loss: 0.3790 - val_accuracy: 0.8687
Epoch 17/50
```

```
704/704 [==============================] - 6s 9ms/step - loss: 0.3408 -
accuracy: 0.8841 - val_loss: 0.3773 - val_accuracy: 0.8686
Epoch 18/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3378 -
accuracy: 0.8848 - val_loss: 0.3760 - val_accuracy: 0.8693
Epoch 19/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3350 -
accuracy: 0.8859 - val_loss: 0.3748 - val_accuracy: 0.8689
Epoch 20/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3322 -
accuracy: 0.8871 - val_loss: 0.3731 - val_accuracy: 0.8699
Epoch 21/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3296 -
accuracy: 0.8880 - val_loss: 0.3720 - val_accuracy: 0.8701
Epoch 22/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3270 -
accuracy: 0.8890 - val_loss: 0.3706 - val_accuracy: 0.8706
Epoch 23/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3246 -
accuracy: 0.8899 - val_loss: 0.3695 - val_accuracy: 0.8711
Epoch 24/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3223 -
accuracy: 0.8903 - val_loss: 0.3686 - val_accuracy: 0.8718
Epoch 25/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3200 -
accuracy: 0.8915 - val_loss: 0.3673 - val_accuracy: 0.8718
Epoch 26/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3178 -
accuracy: 0.8923 - val_loss: 0.3665 - val_accuracy: 0.8719
Epoch 27/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3156 -
accuracy: 0.8930 - val_loss: 0.3653 - val_accuracy: 0.8733
Epoch 28/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3136 -
accuracy: 0.8941 - val_loss: 0.3649 - val_accuracy: 0.8731
Epoch 29/50
704/704 [==============================] - 7s 9ms/step - loss: 0.3116 -
accuracy: 0.8947 - val_loss: 0.3643 - val_accuracy: 0.8719
Epoch 30/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3094 -
accuracy: 0.8952 - val_loss: 0.3630 - val_accuracy: 0.8739
Epoch 31/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3075 -
accuracy: 0.8964 - val_loss: 0.3626 - val_accuracy: 0.8732
Epoch 32/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3057 -
accuracy: 0.8969 - val_loss: 0.3620 - val_accuracy: 0.8730
Epoch 33/50
```

```
704/704 [==============================] - 6s 9ms/step - loss: 0.3039 -
accuracy: 0.8977 - val_loss: 0.3613 - val_accuracy: 0.8739
Epoch 34/50
704/704 [==============================] - 6s 9ms/step - loss: 0.3020 -
accuracy: 0.8982 - val_loss: 0.3603 - val_accuracy: 0.8743
Epoch 35/50
704/704 [==============================] - 6s 8ms/step - loss: 0.3002 -
accuracy: 0.8990 - val_loss: 0.3598 - val_accuracy: 0.8741
Epoch 36/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2984 -
accuracy: 0.8996 - val_loss: 0.3594 - val_accuracy: 0.8741
Epoch 37/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2967 -
accuracy: 0.8998 - val_loss: 0.3586 - val_accuracy: 0.8748
Epoch 38/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2950 -
accuracy: 0.9006 - val_loss: 0.3580 - val_accuracy: 0.8750
Epoch 39/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2933 -
accuracy: 0.9015 - val_loss: 0.3574 - val_accuracy: 0.8750
Epoch 40/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2917 -
accuracy: 0.9021 - val_loss: 0.3569 - val_accuracy: 0.8752
Epoch 41/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2901 -
accuracy: 0.9024 - val_loss: 0.3566 - val_accuracy: 0.8750
Epoch 42/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2885 -
accuracy: 0.9033 - val_loss: 0.3556 - val_accuracy: 0.8753
Epoch 43/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2869 -
accuracy: 0.9039 - val_loss: 0.3554 - val_accuracy: 0.8751
Epoch 44/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2854 -
accuracy: 0.9046 - val_loss: 0.3550 - val_accuracy: 0.8756
Epoch 45/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2839 -
accuracy: 0.9049 - val_loss: 0.3546 - val_accuracy: 0.8760
Epoch 46/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2824 -
accuracy: 0.9056 - val_loss: 0.3545 - val_accuracy: 0.8763
Epoch 47/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2809 -
accuracy: 0.9060 - val_loss: 0.3540 - val_accuracy: 0.8767
Epoch 48/50
704/704 [==============================] - 6s 8ms/step - loss: 0.2794 -
accuracy: 0.9061 - val_loss: 0.3534 - val_accuracy: 0.8772
Epoch 49/50
```

```
704/704 [==============================] - 6s 8ms/step - loss: 0.2779 -
accuracy: 0.9070 - val_loss: 0.3535 - val_accuracy: 0.8766
Epoch 50/50
704/704 [==============================] - 6s 9ms/step - loss: 0.2764 -
accuracy: 0.9074 - val_loss: 0.3526 - val_accuracy: 0.8764
```

[12]:

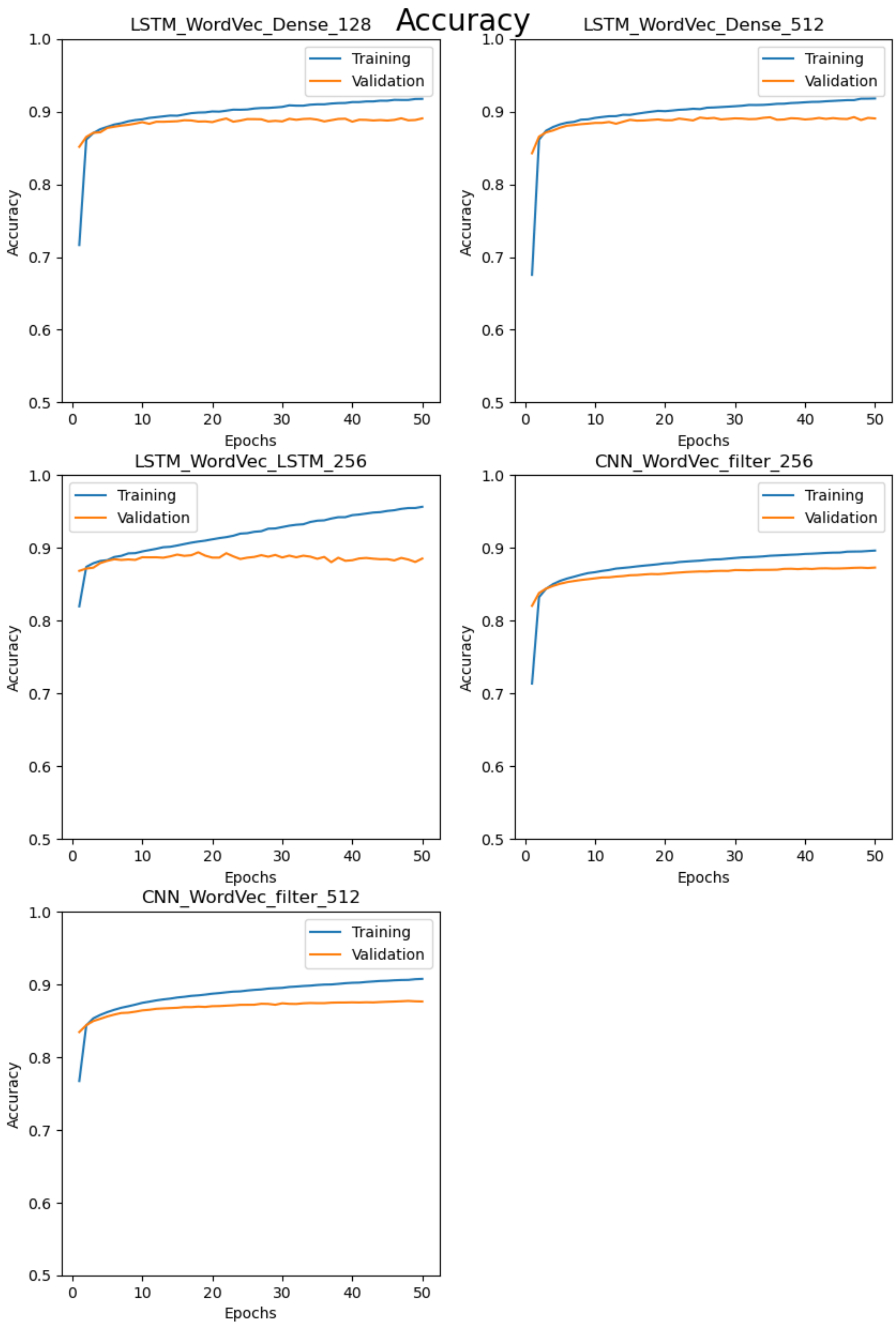
```
# Save the stats to disk incase the session ends
json_out(lstm_wordvec_model_hp1.stats| lstm_wordvec_model_hp2.stats␣
 ↪|lstm_wordvec_model_hp3.stats | cnn_model_word_vec_hp1.stats |␣
 ↪cnn_model_word_vec_hp2.stats, 'hp_all' )
hyperparameter_stats = json_to_dict('hp_all.json')
```

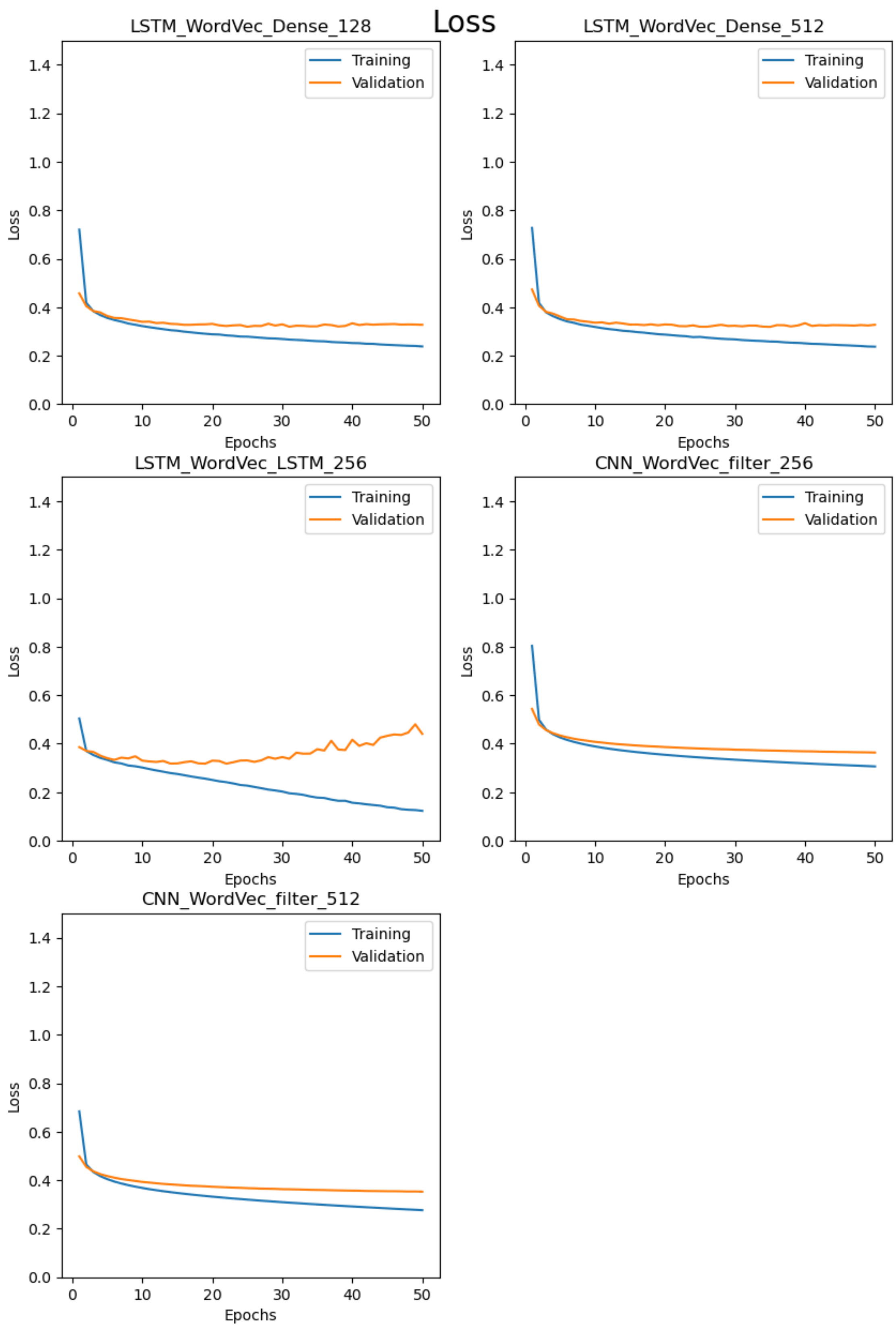
5.2 Hyperparameter Results

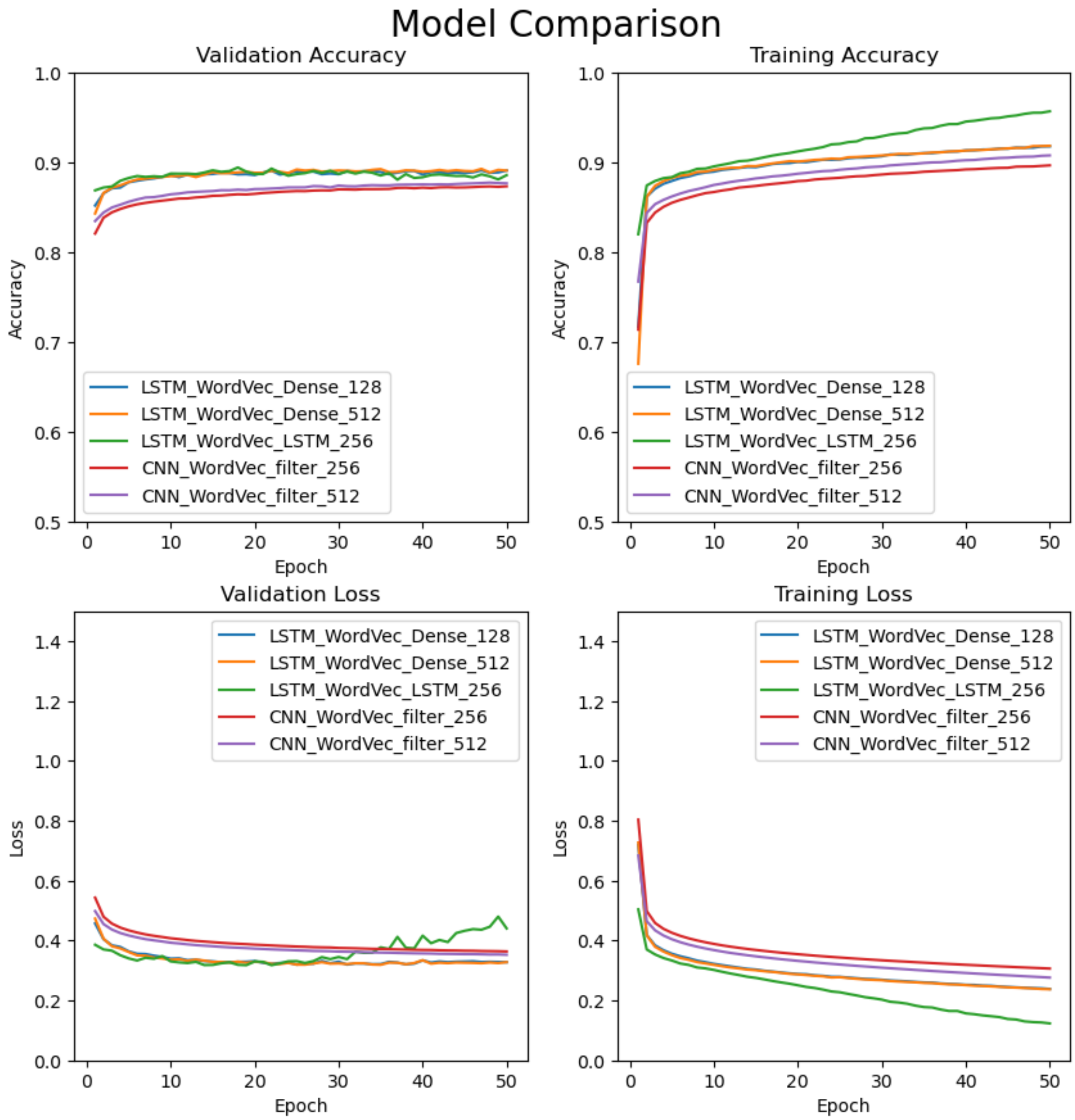
Upon initial glance, there does not appear to be a great change with the adjustment of any of the hyperparmeters. The CNN models did worse than the LSTM models, and increasing the layers in the CNN model appeared to make the model perform slightly worse.

[14]:

```
plot_all_plots(hyperparameter_stats, plot_type = 'accuracy', cols = 2)
plot_all_plots(hyperparameter_stats, plot_type = 'loss', cols = 2)
plot_models_together(hyperparameter_stats)
```


Accuracy
LSTM_WordVec_Dense_128
LSTM_WordVec_Dense_512
LSTM_WordVec_LSTM_256
CNN_WordVec_filter_256
CNN_WordVec_filter_512
Training
Validation
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50

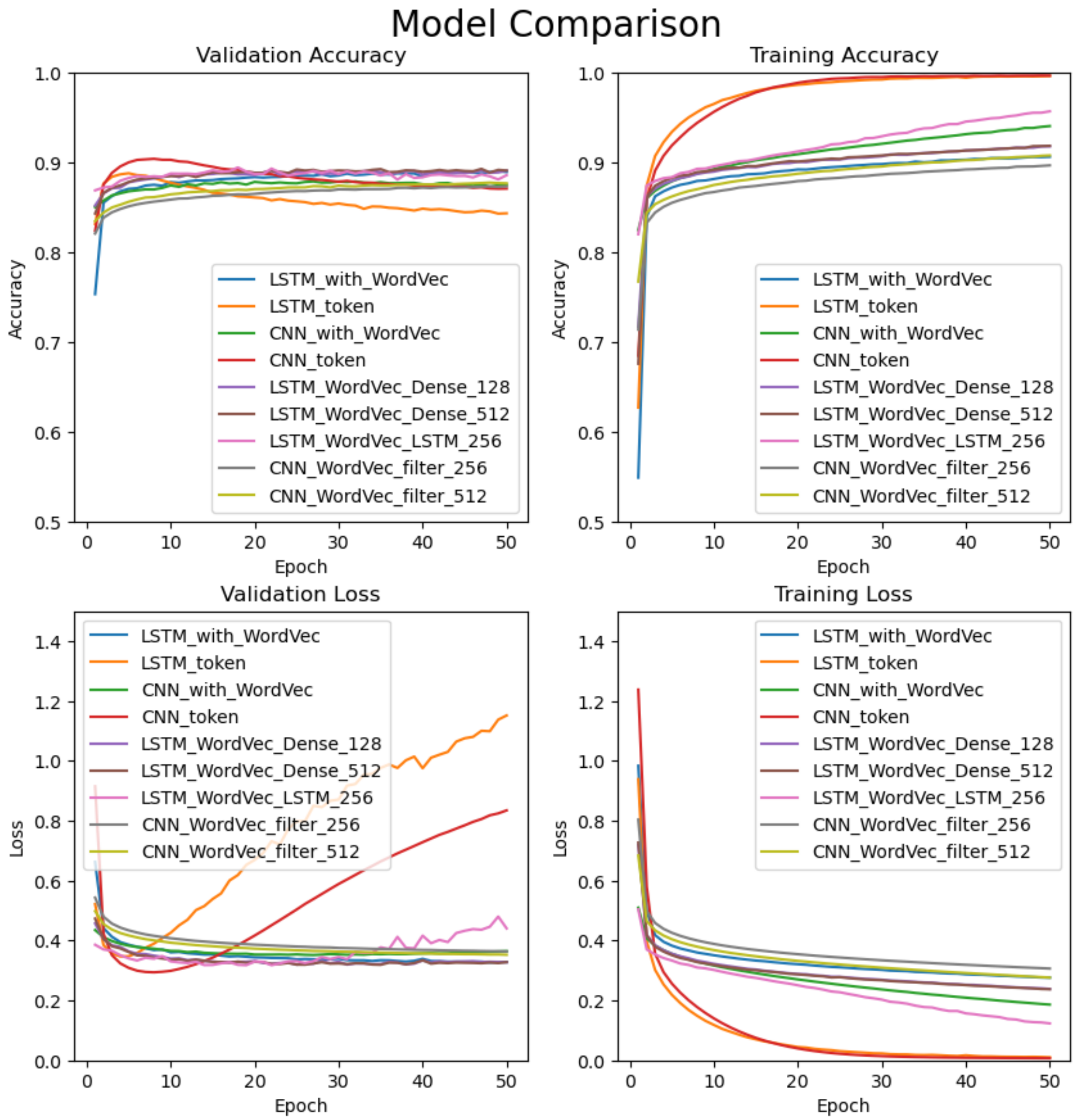

Loss
LSTM_WordVec_Dense_128
LSTM_WordVec_Dense_512
LSTM_WordVec_LSTM_256
CNN_WordVec_filter_256
CNN_WordVec_filter_512
Training
Validation
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40
50



5.3 Final Hyperparameter Comparisons

Comparing the hyperparameter adjustments to the previous models does show very little change after the hyperparameter adjustments. In fact, the the CNN models did worse with increased filters. One way in which the model might be tuned further would be to try and reducing filter size of the CNN models, and see if there is any improvement.

[33]:
```
plot_models_together(initial_stats | hyperparameter_stats)
create_summary_table(initial_stats| hyperparameter_stats)
```



	Model	Max Acc	Minnimum Loss	Max Val Acc	Min Val Loss \
0	LSTM_with_WordVec	0.906	0.276	0.890	0.327
1	LSTM_token	0.996	0.010	0.888	0.347
2	CNN_with_WordVec	0.940	0.186	0.878	0.352
3	CNN_token	0.996	0.007	0.904	0.294
4	LSTM_WordVec_Dense_128	0.918	0.238	0.891	0.320
5	LSTM_WordVec_Dense_512	0.918	0.237	0.892	0.319
6	LSTM_WordVec_LSTM_256	0.957	0.123	0.894	0.318
7	CNN_WordVec_filter_256	0.896	0.306	0.873	0.364
8	CNN_WordVec_filter_512	0.907	0.276	0.877	0.353

Epoch of Min Val Loss

```
0                    47
1                     3
2                    26
3                     7
4                    24
5                    34
6                    18
7                    49
8                    49
```

6 Final Model

6.1 Architecture

After looking at all of the data, I ultimately decided to use the CNN model without the WordVec model. All of the models were really close, but tuning hyperparameters didn't seem to help the WordVec model much, and this model still beat those models slightly, which makes me think that is has the most potential. It also had the potential to train with less runtime, since it needed significantly less epochs to get to its maximum validation accuracy.

[53]:
```
final_model = Model_Seq([vectorize_layer,
       layers.Embedding(
        input_dim=word_count,
        output_dim=256,
        mask_zero=True),
     layers.Conv1D(256, 3, activation = 'relu', padding = 'valid'),
     layers.GlobalMaxPooling1D(),
     layers.Dense(4, activation='softmax')], name = 'Final_Model', X_train =␣
 ↪X_train, y_train = y_train, X_val = X_val, y_val = y_val, inital_lr=.0001)
```

[55]:
```
final_model.train_model(4, batch_size = 128)
```

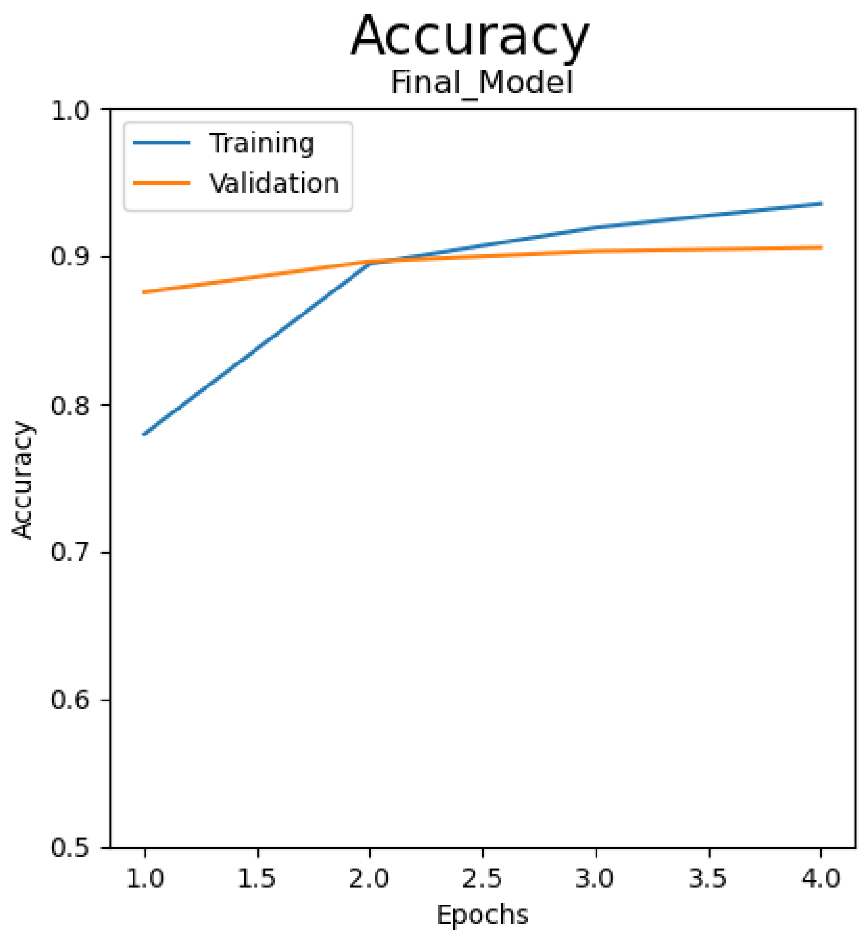
```
Epoch 1/4
704/704 [==============================] - 269s 382ms/step - loss: 0.8910 -
accuracy: 0.7792 - val_loss: 0.4157 - val_accuracy: 0.8751
Epoch 2/4
704/704 [==============================] - 242s 343ms/step - loss: 0.3357 -
accuracy: 0.8944 - val_loss: 0.3245 - val_accuracy: 0.8960
Epoch 3/4
704/704 [==============================] - 239s 340ms/step - loss: 0.2546 -
accuracy: 0.9188 - val_loss: 0.3006 - val_accuracy: 0.9029
Epoch 4/4
704/704 [==============================] - 239s 340ms/step - loss: 0.2047 -
accuracy: 0.9350 - val_loss: 0.2936 - val_accuracy: 0.9053
```

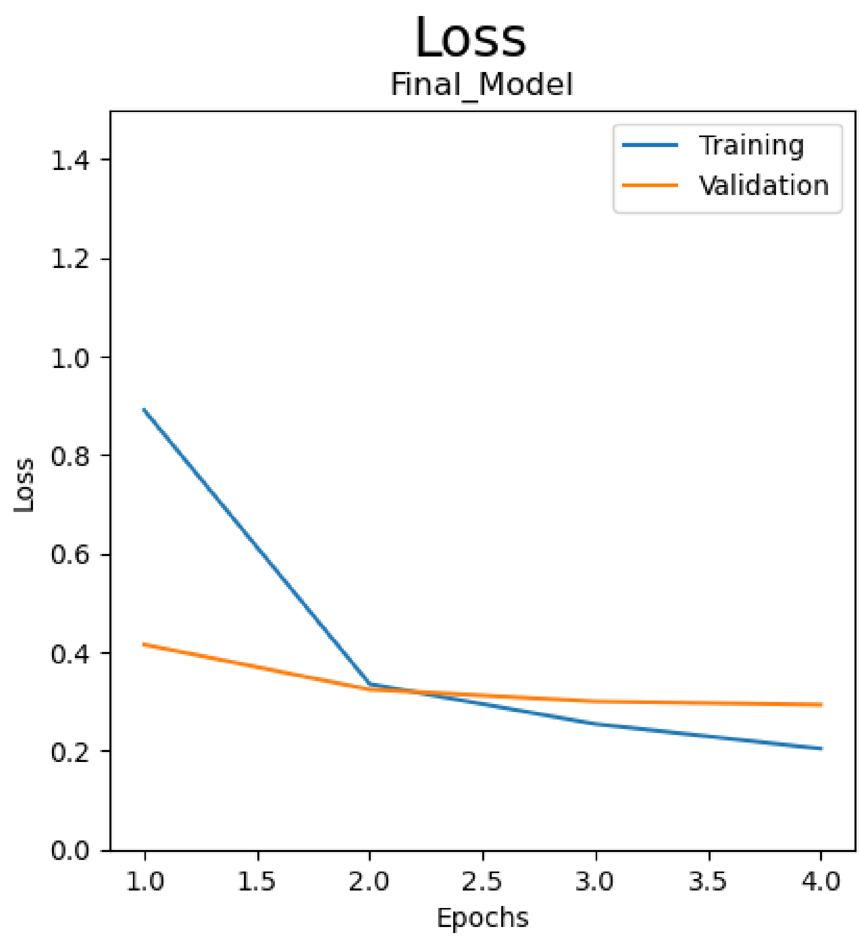
[58]:
```
#json_out(final_model.stats, 'final_model')
final_stats = json_to_dict('final_model.json')
```

6.1.1 Summary

After the model completed running, I verified that there was no evidence of overfitting, which would be seen with an increase of the validation loss, or a decrease of the validation accuracy.

[59]:
```
plot_all_plots(final_stats, plot_type = 'accuracy', cols = 1)
plot_all_plots(final_stats, plot_type = 'loss', cols = 1)
```





6.2 Test Data

6.2.1 Model

One important step with any model is to check it against unseen data to see how accurate it is. For this purpose, I used the test data from the dataset. I ran this data through the model and made a prediction about class labels.

[81]:
```
pred_test = predict_from_model(final_model, X_test)
```

```
238/238 [==============================] - 4s 18ms/step
```

[82]:
```
test_acc = accuracy_score(y_test, pred_test)
test_metrics = precision_recall_fscore_support(y_test, pred_test)
```

[83]:
```
test_metrics_df = pd.DataFrame(test_metrics, index = ['Precision', 'Recall',␣
↪'F-beta', 'Support'], columns = class_ids).T
test_metrics_df['Support'] =test_metrics_df['Support'].astype(int)
```

6.2.2 Results

Overall, the model did fairly well. Looking at the metrics and a confusion chart, it struggles the most differentiating between 'Business' and 'Sci/tech.' 'Business' had the worst metrics across the board, with the model often assuming 'Business' was other categories, especially the aforementioned 'Sci/Tech.'

The models did the best with 'Sports' with high recall and precision values. The place that it tended to struggle with 'Sports' is telling it apart from 'World'. It also had more of a tendency to produce false positives for 'Sports', compared to false negatives for 'Sports.' To me, it makes sense that it did the best with 'Sports,' since there are a lot of specific words, that are applied to sports. However, many sports terms are used as metaphors in other contexts, so it isn't surprising that this doesn't result in perfect results.

It had a final accuracy of .90, which is not that far off the training accuracy of the final model. This is a good place to be since, it shows that we probably didn't overfit the training data.

[85]:
```
cm = confusion_matrix(y_test, pred_test)
cm_display = ConfusionMatrixDisplay(cm, display_labels = class_ids).plot()
plt.title = 'Test'
plt.show()
display(test_metrics_df.round(3))
print(f"Overall Accuracy: {test_acc:.3f}")
```

	Precision	Recall	F-beta	Support
World	0.916	0.893	0.905	1900
Sports	0.953	0.972	0.962	1900
Business	0.864	0.863	0.864	1900
Sci/Tech	0.879	0.885	0.882	1900

```
Overall Accuracy: 0.903
```

6.3 Improvements Discussion

While the final model was in no way awful, there are lots of ways this model could be improved. I was surprised that both types of models ended with very similar accuracy and trying to change hyperparameters had very little effect on the overall outcome. Certainly, there are more hyperparameters that could be tuned in both the CNN and LTSM Models, to see if accuracy could be improved, or the speed of computation could be improved. However, there is no obvious hyperparameter that could be changed to improve the model.

There are many different text vectorization techniques that could be used with this model. I chose to naively vectorize the text, and I also used a WordVec model that I trained off the data. In the end, the best model did not use the WordVec model for its training. There are WordVec models that are pretrained that could possibly allow the creation of a more accurate matrix for the embedding layer.

Data augmentation is another way in which the model could be improved. This could be done by duplicating the news articles, but replacing all of the words with synonyms. This would have the effect of increasing the vocabulary of the model, so that future, unseen data, might be able to be categorized better.

6.4 Final Conclusion

The model had an overall accuracy of 90 percent on the test data, which is okay, but certainly not perfect. It struggled more with some categories than others, especially between 'Business' and 'Sci/Tech,' but still did an okay job on those. That brings up the question is the model "good enough?" That would depend on the application of the model. There are probably few to no life or death situations where a miscategorized news article could results in death or harm, so there is a good chance it is good enough. But at the same time, if you are using this in a new aggregator, customers probably aren't going to /be thrilled with a product that only categorizes 9 out of every ten articles correctly. In reality, the answer to the question "Is the model good enough?" is "it depends…"

7 References

Anand, Aman. "AG News Classification Dataset." *Kaggle.* https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset

Binhuraib, Taha. "NLP with CNNs." * Towrds Data Science.* October 13, 2020. https://towardsdatascience.com/nlp-with-cnns-a6aa743bdc1e*

Ganegedara, Thushan. "Intuitive Guide to Understanding GloVe Embeddings." *Towards Data Science.* May 5th, 2019. https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

"Generating WordClouds in Python Tutorial." *DataCamp.* February 2023. https://www.datacamp.com/tutorial/wordcloud-python

Gulli, Antonio. "AG's corpus of news articles." http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.htm

Řehůřek, Radim. "Word2Vec Model". *Gensim.* December 21, 2022. https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html

Saturn Cloud. "Using Pretrained Gensim Word2Vec Embeddings in Keras: A Comprehensive Guide." *Saturn Cloud Blog.* July 10th, 2023. https://saturncloud.io/blog/using-pretrained-gensim-word2vec-embeddings-in-keras-a-comprehensive-guide/

Zhang, Xiang and Zhao, Junbo and LeCun, Yann. "Character-level Convolutional Networks for Text Classification." *Arxiv.* 2016. https://arxiv.org/abs/1509.01626v3