CLASSIFICATION OF NEWS ARTICLES VIA DEEP LEARNING METHODS

Steve Desilets

MSDS 458: Artificial Intelligence and Deep Learning

November 5, 2023

1. Abstract

Natural language processing (NLP) algorithms, such as those sometimes used by chatbots, have transformed society by automating classification tasks so that people can focus their time on more strategically important (or more enjoyable) tasks. In this study, we leverage a subset of the AG News dataset to create ten such NLP models capable of classifying article test into four classes of article subjects. While creating these models, we discover which families of models may be most appropriate for NLP classification tasks and the impacts of architecture and regularization techniques on model performance. Through this study we build on the innovations of previous groundbreaking NLP researchers and find that, in particular, unidirectional long short-term memory (LSTM) models with proper regularization and architecture can serve as excellent tools for classifying news articles.

2. Introduction

From virtual assistant chatbots on company websites to voice recognition models in remote controls, natural language processing (NLP) models focused on text classification have begun positively impacting people's lives in many ways. In this study, we will construct and examine the effectiveness of 10 NLP algorithms similar to those leveraged by industry-leading tech companies to automate text classification processes in our daily lives. Specifically, we will leverage the publicly available subset of the AG_News dataset – a subset which contains 120,000 training and 7,600 testing observations collected from over 2,000 news organizations throughout a multi-year period (Villanova 2023). Each observation includes text from actual news articles along with their corresponding article classes. As summarized in Table 1 below, these articles each correspond to one of four article classes.

Table 1: Article Classes and Corresponding Labels in the AG_News Dataset

Class Label in AG_News Dataset	Class Description
0	World
1	Sports
2	Business
3	Science / Technology

Utilizing this dataset of articles, we create ten neural network models of varying model types so that we can gain discover which model types are best at text classification tasks. We also leverage various data wrangling, model architecture, and regularization techniques so that we can discover the impacts of these model design choices on model performance.

3. Literature Review

Advancements in natural language processing have held a special place in the public's fascination since Alan Turing first proposed the Turing Test in his 1950 paper "Computing Machinery and Intelligence." (Wikipedia 2023). This test focused on whether machines could engage in written conversation so well that humans would not be able to reliably distinguish whether they were conversing with a human or a machine. Since then, researchers have developed many groundbreaking chatbots, such as ELIZA – one of the earliest famous chatbots which Joseph Weizenbaum built in 1966 to mimic conversations with a psychotherapist (Ina 2022). Recent developments, like transformer-based models, have empowered data scientists (like those at Open AI) to construct far more advanced NLP models capable of writing in a highly sophisticated way. While in this paper, we focus on building models that focus on a much simpler task – text classification – researchers have also employed many of the same principles for text classification purposes in the past as those used for chatbots. For example, data scientists have demonstrated that 1-D convolutional neural networks can serve as powerful tools for classifying articles into their appropriate classes (Jang, Kim and Kim 2019). Perhaps even more exciting, researchers have even leveraged the exact same dataset that we'll use in this article – along with long short term memory (LSTM) models – to classify articles into classes as well (Dutta 2021). While these researchers reached accuracy levels as high as 93% and 91% respectively, we hope to build upon the knowledge of this long history of NLP thought leaders to construct deep learning-based text classification models that are as accurate as possible.

4. Methods

For this study, we constructed 10 neural network-based models throughout two phases of experimentation with AG_News NLP classification models. To begin this process, we first imported the

AG_News dataset and conducted exploratory data analysis, including examining the values in each column of our dataset and visualizing a histogram displaying the number of tokens per document. We also divide the dataset into three pieces: 114,000 training observations, 6,000 validation observations, and 7,600 testing observations. Last, we conducted our first round of data wrangling, including removal of stopwords and punctuation and vectorization using one hot coding.

In the first phase of experimentation, we constructed five different types of models with the objective of identifying which type of model would be best at performing article classification for the AG_News dataset. Accordingly, for the first five models, we constructed: 1) a long short term memory (LSTM) model, 2) a 1-dimensional convolutional neural network (CNN) model, 3) a gated recurrent unit (GRU) model, 4) a recurrent neural network (RNN) model, and 5) a dense artificial neural network model (ANN). Notably, even though these models all leveraged similar regularization techniques like early stopping and dropout layers, we did make one important change after the second model to address long fitting times, which was to reduce the number of tokens per document from 95 to 40. After constructing each of these models, we evaluated their performance by analyzing their confusion matrices, accuracy and loss curves throughout training epochs, accuracy and loss performance metrics, and time to fit. The goal of this first phase of experimentation was to identify which of these five model classes performed the best at article classification, so that we could build upon its success in the second phase of experimentation.

In the second phase of experimentation, the goal was to determine the impact of NLP model design choices - such as data wrangling, model architecture, and regularization - on model performance so that we could optimize the performance of the best model from phase one. For the sixth model, since the best model from phase one was our bidirectional LSTM model, we started off phase two by building a unidirectional LSTM to determine whether that change in how the model understands text context might impact model performance. For the seventh model, we aimed to build upon the success of the sixth model by examining whether deregularization would improve performance, so we reduced the dropout layer's dropout rate from 50% to 40% and modified the architecture by doubling the number of nodes in the unidirectional LSTM model's hidden layer. For the eighth model, we attempted to study whether we could improve the seventh model by increasing the vocabulary size from 1,000 to 2,000 and by reintroducing a bit of regularization by pushing the dropout rate back up to 50%. For the ninth model, we strived to improve the eighth model by determining whether reducing the number of tokens per document from 40 to 33 would improve the signal-to-noise ratio enough to improve model performance. For the tenth and final model, we aimed to build upon the success of the ninth model introducing further regularization, including reducing the dimensionality of the hidden layer's outputs from 32 nodes to 16 nodes and trimming the number of tokens per document down further from 33 tokens to 30 tokens. For each of these five unidirectional LSTM models, we evaluated model performance metrics (such as accuracy, loss, and time to fit) for the training, validation, and testing datasets, and we examined performance summaries (like confusion matrices and accuracy / loss plotted by epoch during training).

5. Results

After fitting the first text classification model, the primary takeaway is surprisingly not about the model performance but about the time to fit, which reached 3 hours, 31 minutes, and 51 seconds – by far the longest time to converge of all 10 models. Despite the long time to converge, this model actually achieved the best results of any of our five models throughout experimentation phase one. With a testing accuracy of 85.9%, this initial model definitely outperformed our initial expectations for model performance. Examination of the confusion matrix in Appendix B revealed that even though the classification accuracy was quite high, the biggest contributor to model misclassifications was distinguishing between "Business" and "Science / Technology" articles – a trend that would persist for all 10 models that we constructed. Notably, as the graphs that display loss and accuracy across epochs during training in Appendix C display, this model may have exhibited a tiny bit of overfitting to the training dataset. Still, given that this bidirectional LSTM model was the best overall model from phase 1, we strived to build on its success throughout the second phase of experimentation.

Our second model, the 1-D CNN model exhibited some nice attributes. This model achieved the second best testing accuracy of the phase one models – 84.6% and the second fastest training time of all ten models constructed – 13 minutes and 11 seconds. The fact that the 1-D CNN model didn't manage to outperform the bidirectional LSTM model is not surprising though given that LSTM models are specifically designed to handle sequential data and to take advantage of textual memory in a way that CNNs are not. Examination of the confusion matrix in Appendix B reveals that this model, like all ten models that we constructed, was most adept at identifying “Sports” articles – reaching an accuracy of 90.8% for identifying them. The plot of accuracy and loss across training epochs also highlights that this model was fit in just four epochs – the fewest epochs needed to fit any of our models – as well as no evidence of overfitting.

With a testing accuracy of 83.4%, the third model that we constructed – the GRU model – achieved the lowest testing accuracy of all ten models we constructed. Admittedly, the poor relative performance of this model was a bit surprising (since GRU models tend to handle sequential data well). However, this underperformance may be partially attributable to the fact that we shortened the number of tokens per document from 95 to 40 in order to avoid encountering extremely long times to fit (and to increase the signal-to-noise ratio in our data). Notably, the accuracy and loss trends across epochs in Appendix C reveal that implementing early stopping regularization was a great choice given that the validation accuracy and loss barely changed at all across epochs even though this model took 9 epochs to formally converge. For this reason, we maintained early stopping regularization in place for all ten models.

Achieving the median testing accuracy and loss values for our initial five models, our fourth model – the RNN – exhibited mediocre success. Furthermore, this model’s confusion matrix looked very similar to the confusion matrices for all the other 9 models, which further emphasized how unremarkable its results were compared to the other phase one models. Notably, given that RNNs were designed specifically for handling sequential data like text data, it’s a bit surprising that this model didn’t perform better than some other models (like the 1-D CNN model). However, with no strong evidence of overfitting in its Appendix C outputs and a reasonable time to fit of 27 minutes and 40 seconds, this model did have some positive attributes as well.

For our fifth and final model from experimentation phase one – the dense ANN model, the primary takeaway was that this was the fastest model of all to construct. Given the relative simplicity of ANNs compared to models designed for sequential data (like LSTMs, GRUs, and RNNs), it's not surprising that this model converged the most quickly (in just 11 minutes and 4 seconds). Still, this with a testing accuracy of 84%, this was the second worst performing model of all, which is not surprising given that ANNs have no ability to capture sequential correlations in data via attention / memory. Another notable finding from this model is that the testing accuracy range for the first phase of experimentation was quite narrow – ranging from 83.4% to 85.9%. While it was a bit surprising how narrow the range of testing accuracies turned out to be in phase one, we still advanced into phase two aiming to improve upon our best-performing phase one model – the LSTM model.

In the sixth model, we aimed to examine whether we could improve the first LSTM model's performance by changing its directionality from bidirectional to unidirectional. This change did increase the testing accuracy slightly from 85.9% to 86.2%, which was surprising because bidirectional LSTMs generally perform better than unidirectional LSTMs, since the former can take advantage of additional context. In this case, this counterintuitive phenomenon might be able to be explained by the fact that we were using one hot coding (not semantic embedding dimensions like Word2Vec), so the advantage of additional directionality may have been reduced. Given that this model resulted in the best validation and testing accuracy of the first 6 models and that it didn't exhibit strong evidence of overfitting, all the remaining models that we constructed (models 7 through 10) were unidirectional LSTM models.

For the seventh model, we aimed to research whether deregularization – driven by decreasing the dropout rate and doubling the nodes in the hidden layer – would improve model performance. In fact, these changes increased the testing accuracy by 0.3% and decreased the testing loss by 0.019 – making this the best performing model of the first seven constructed. Despite this success of this model, the results displayed some opportunities for improvement as well. The accuracy and loss curves across epochs suggested mild overfitting may have arisen, and this model took over 1 hour and 16 minutes to fit

– the second longest model fitting time overall. Given that this was our best performing model so far, we did maintain the new model architecture for most of the subsequent models.

For the eighth model, we researched the impact of vocabulary size on model performance. As displayed in Appendix A, this move to increase the vocabulary size from 1,000 tokens to 2,000 tokens yielded great results. Among the first eight models, this model achieved the highest accuracies and lowest losses for the training, validation, and testing datasets, which really underscores the value added by expanding the model vocabulary. Not only was this model a success because the testing accuracy reached 87.9%, but also this model was a success because the training time dropped by 33 minutes down to roughly 43 minutes. Fortunately, this model's accuracy and loss curves in Appendix C exhibited little evidence of overfitting, so we maintained this expanded vocabulary size of 2,000 as we moved into model 9.

For the ninth model, we strived to increase the signal-to-noise ratio in the documents and to improve model performance by reducing the number of tokens per document from 40 to 33. This move proved to be a success. Not only did the training time decrease by over 2 minutes compared to model eight, but also, this model resulted in the lowest training, validation, and testing losses of all ten models constructed: 0.304, 0.35, and 0.365, respectively. Unsurprisingly, this model also exhibited the highest testing accuracy of all 10 models constructed – 88.3%. Appendices B and C, which show similar confusion matrix patterns to all the former confusion matrices and little evidence of overfitting for this model, also reflect positive results for this model. Given all the positive findings surrounding model 9, if we had to pick one of these article classification models for a client, this would certainly be the one that we would recommend launching.

For our tenth and final model, we aimed to build upon the successes of the previous two models by further decreasing the number of tokens per model from 33 to 30 and by increasing the vocabulary from 2,000 to 2,500. While our goal was to further increase the signal-to-noise ratio, it appears that we may have been unsuccessful in achieving this goal. At 87.8%, the testing accuracy for this model was about half a percent lower than that of model 9. Still this model did exhibit some positive findings like it tying for lowest testing loss overall (0.365), its relatively short time to fit of 31 minutes and 48 seconds and its accuracy and loss charts in Appendix C that exhibit little evidence of overfitting.

6. Conclusions

The ten neural network-based models constructed throughout our two phases of research reveal that unidirectional LSTM models can serve as excellent tools for article classification. Furthermore, we find that data wrangling, model architecture, and regularization can have noticeable impacts on model performance as well. If I were the lead researcher responsible for advising a news organization whether to launch one of these NLP models, I would likely advise them to see whether additional research could lead to achieving a higher testing accuracy rate than 88%. Even though the stakes of the misclassification of an article category might be low, I think that simple tweaks could help them achieve higher classification accuracy rates. Specifically, I would recommend experimenting with using embedding techniques other than one hot coding and with building fine-tuned neural networks on top of successful, pre-trained NLP models like BERT, which have been used to achieve article classification accuracies as high as 93% (Khandelwal 2023). However, if the client news organization needed to move forward with launching one of these NLP models, I would advise that they select the ninth model since it achieved the lowest loss on the training, validation, and testing datasets – as well as the highest testing accuracy.

References

Dutta, Ishan. 2021. “ag-news-classification-lstm.” *Kaggle*. https://www.kaggle.com/code/ishandutta/ag-news-classification-lstm.

Ina. 2022. “The History Of Chatbots – From ELIZA to ChatGPT.” Onlim. https://onlim.com/en/the-history-of-chatbots/

Jang, Beakcheol, Inhwan Kim, and Jong Wook Kim. 2019. “Word2vec Convolutional Neural Networks for Classification of News Articles and Tweets.” *PloS One* 14, no. 8: e0220976–e0220976. https://doi.org/10.1371/journal.pone.0220976.

Khandelwal, Tanish. 2023. “Text-Classification-Ag-News.” *Github*. https://github.com/tknishh/Text-Classification-Ag-News/blob/master/notebook.ipynb

Villanova, Albert. 2023. “Datasets: ag_news.” *Hugging Face*. https://huggingface.co/datasets/ag_news

Wikipedia. 2023. “Chatbot.” 2023. https://en.wikipedia.org/wiki/Chatbot

Appendix A – Top-line Summary Of Experiment Results

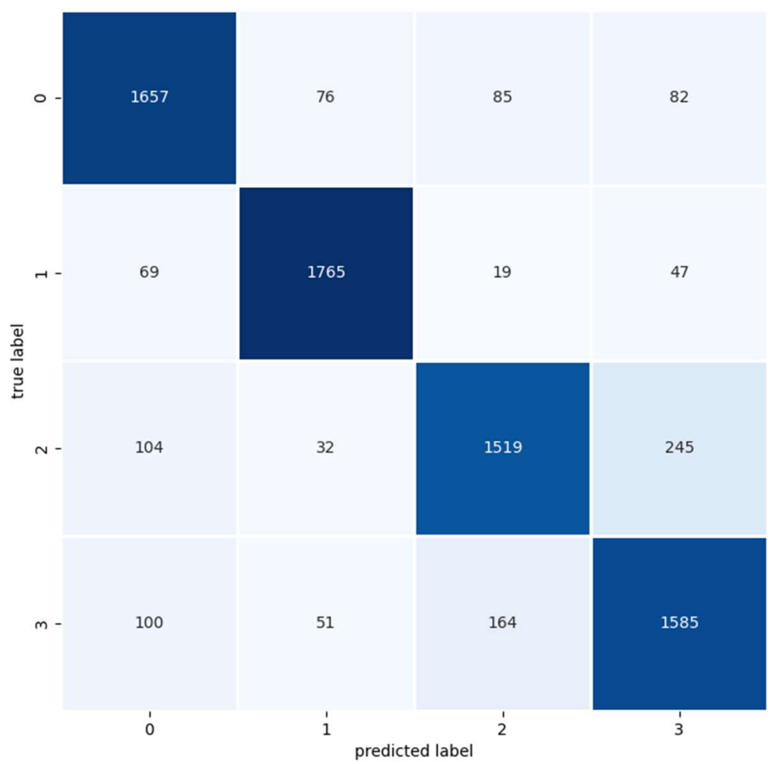
The table below displays the training, validation, and testing accuracy / loss of each of the 10 models developed for classification of articles.

Trial Number	Model Type	Text Data Wrangling and Vectorization	Model Architecture and Regularization	Training		Validation		Testing		Training Time	Recommendation
				Accuracy	Loss	Accuracy	Loss	Accuracy	Loss		
1	Bi-Directional LSTM	• 1,000 Maximum Vocabulary Tokens • One Hot Encoding	• 50% Dropout Layer • Early Stopping	87.6%	0.345	86.2%	0.388	85.9%	0.402	3:31:51	Do Not Implement
2	1-D CNN	• 1,000 Maximum Vocabulary Tokens • One Hot Encoding	• 50% Dropout Layer • Early Stopping	84.9%	0.455	84.8%	0.462	84.6%	0.468	0:13:11	Do Not Implement
3	GRU	• 256-Dimensional Embedding Vectors • 40-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping	84.5%	0.438	84.4%	0.444	83.4%	0.463	0:25:25	Do Not Implement
4	RNN	• 256-Dimensional Embedding Vectors • 40-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping	86.5%	0.396	85.0%	0.44	84.2%	0.459	0:27:40	Do Not Implement
5	Dense ANN	• 256-Dimensional Embedding Vectors • 40-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping	85.1%	0.434	84.7%	0.448	84.0%	0.457	0:11:04	Do Not Implement
6	Uni-Directional LSTM	• 256-Dimensional Embedding Vectors • 40-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping	87.6%	0.354	86.9%	0.396	86.2%	0.41	1:06:12	Do Not Implement
7	Uni-Directional LSTM	• 256-Dimensional Embedding Vectors • 40-Token Maximum Per Document	• 40% Dropout Layer • Early Stopping • Double the Nodes Per In the Hiden Layer than Model 6	87.8%	0.34	86.5%	0.391	86.5%	0.391	1:16:33	Do Not Implement
8	Uni-Directional LSTM	• 256-Dimensional Embedding Vectors • 40-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping • Increased Vocabulary Size to 2000	89.1%	0.324	87.8%	0.36	87.9%	0.37	0:42:58	Do Not Implement
9	Uni-Directional LSTM	• 256-Dimensional Embedding Vectors • 33-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping • Increased Vocabulary Size to 2000	89.6%	0.304	87.9%	0.35	88.3%	0.365	0:40:53	Implement
10	Uni-Directional LSTM	• Dimensionality of the LSTM's Layers Output Reduced by 50% • 30-Token Maximum Per Document	• 50% Dropout Layer • Early Stopping • Increased Vocabulary Size to 2500	89.4%	0.317	88.3%	0.35	87.8%	0.365	0:31:48	Do Not Implement

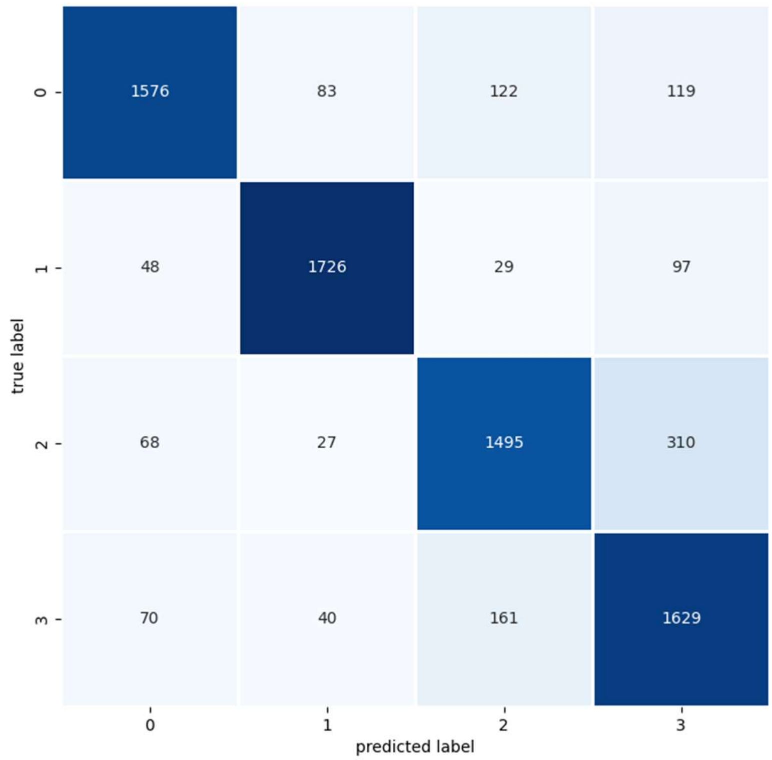
Appendix B – Confusion Matrices Resulting From Experiments

The images below display the confusion matrices resulting from the application of each classification model the testing dataset. For ease of interpretation, these confusion matrices are color coded as heat maps. Larger versions of these images and the code leveraged to generate these confusion matrices are available in Appendix D.

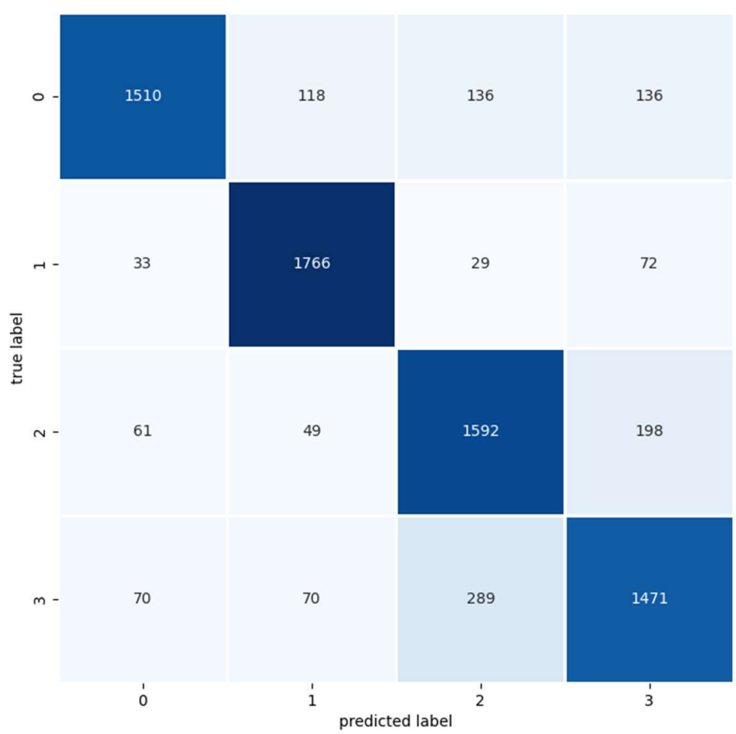
Experiment 1 – Confusion Matrix



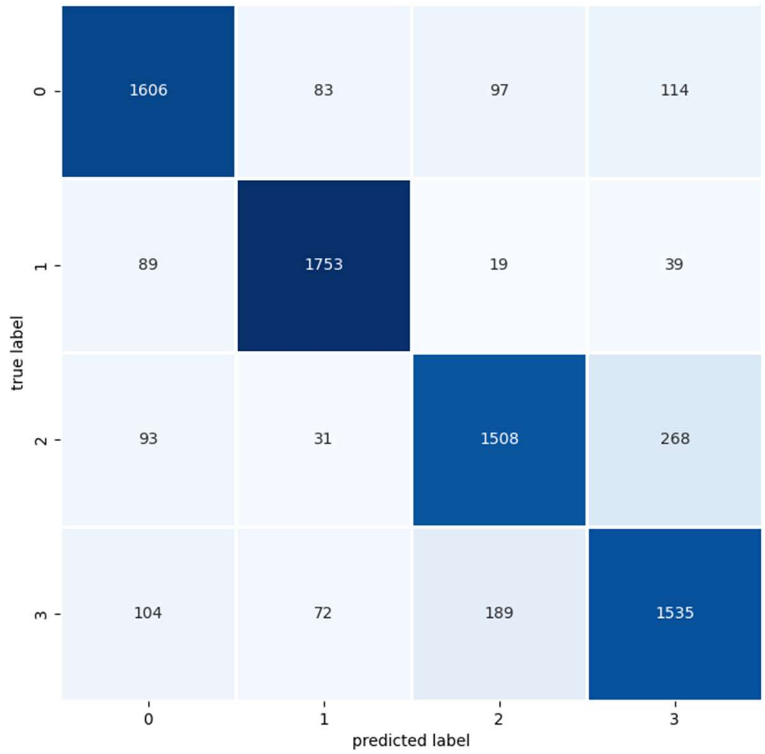
Experiment 2 – Confusion Matrix



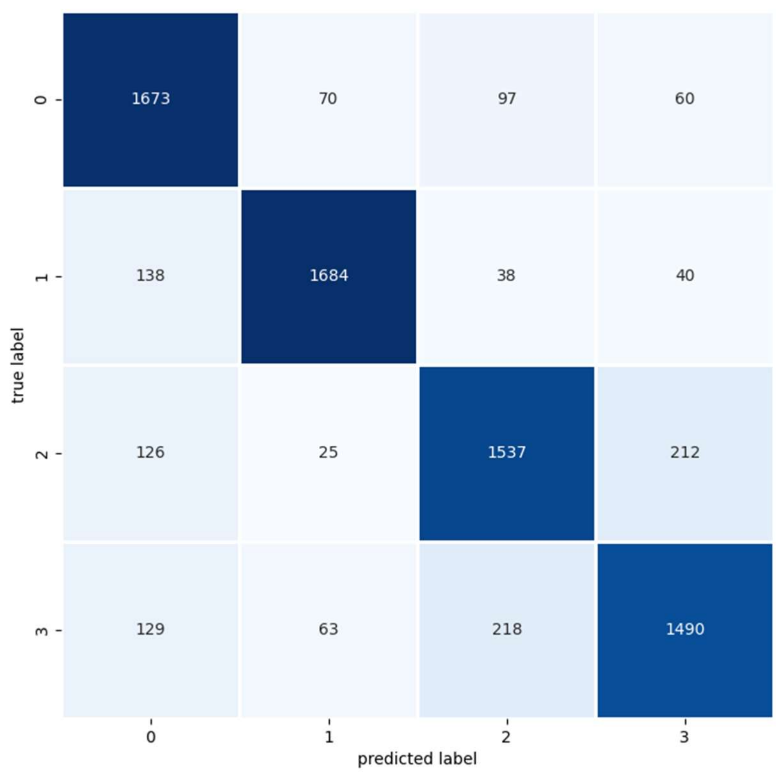
Experiment 3 – Confusion Matrix



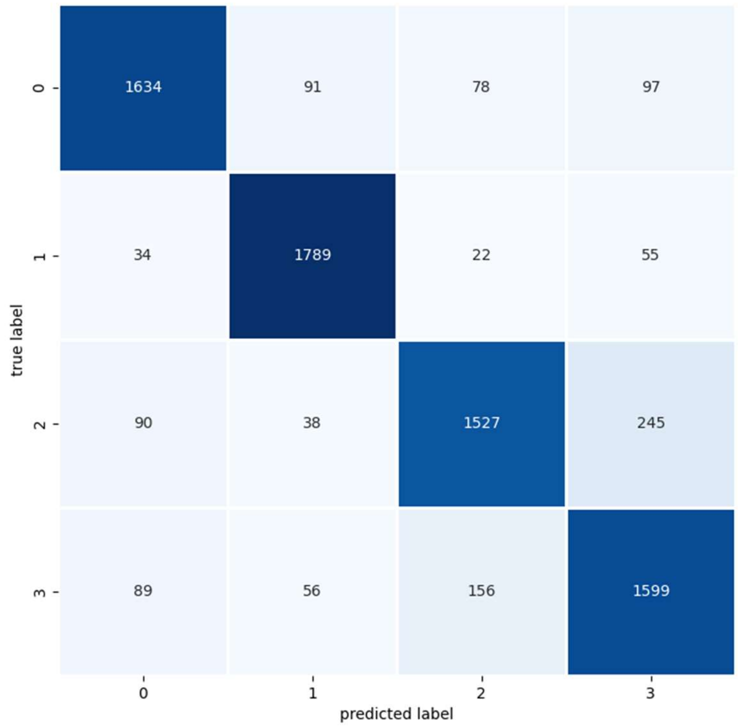
Experiment 4 – Confusion Matrix



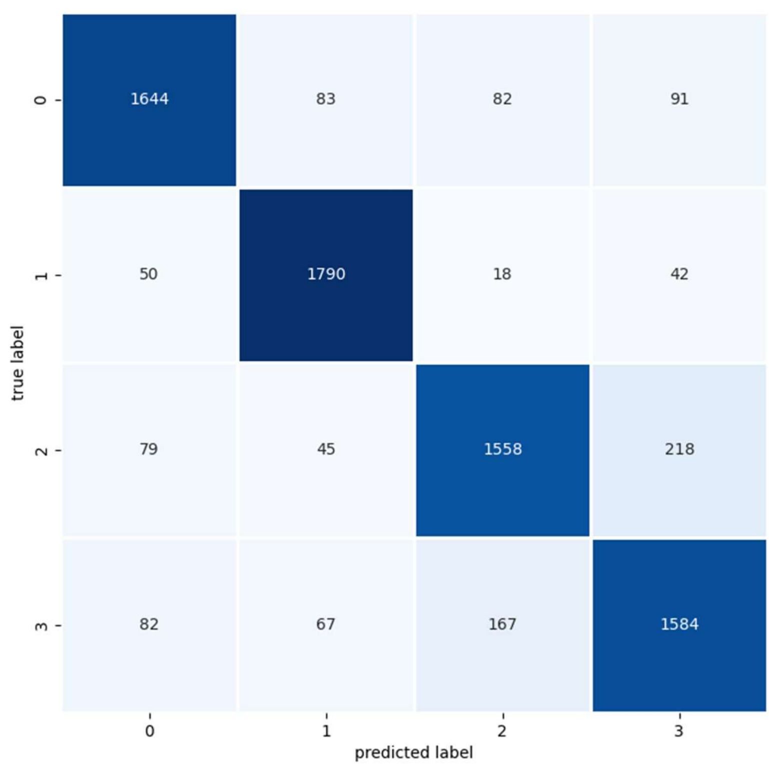
Experiment 5 – Confusion Matrix



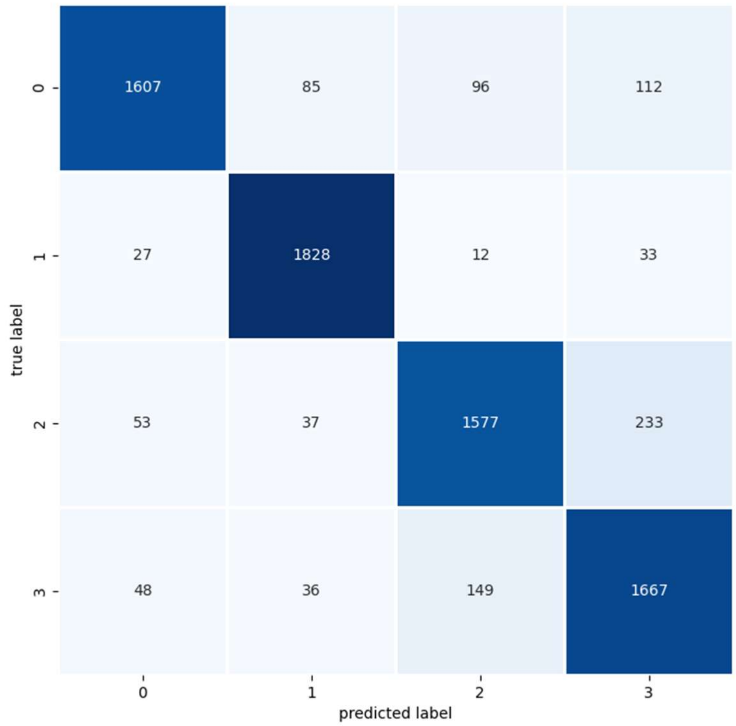
Experiment 6 – Confusion Matrix



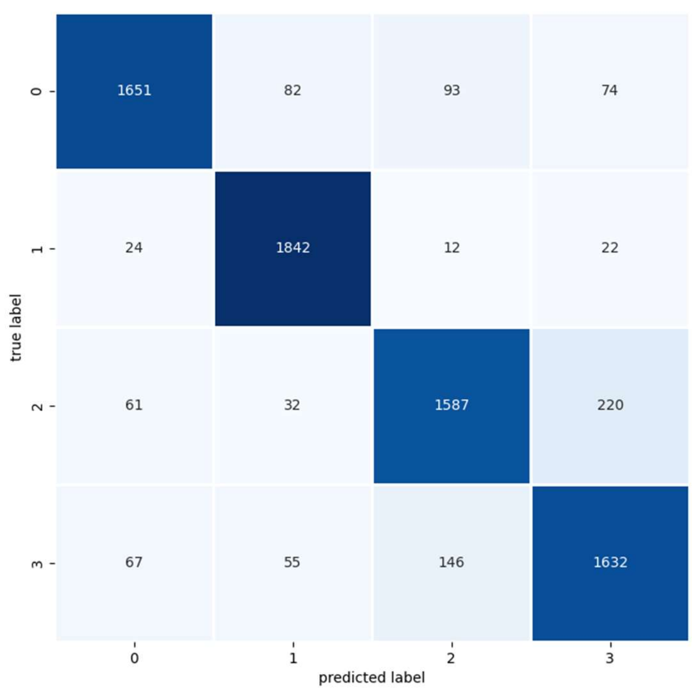
Experiment 7 – Confusion Matrix



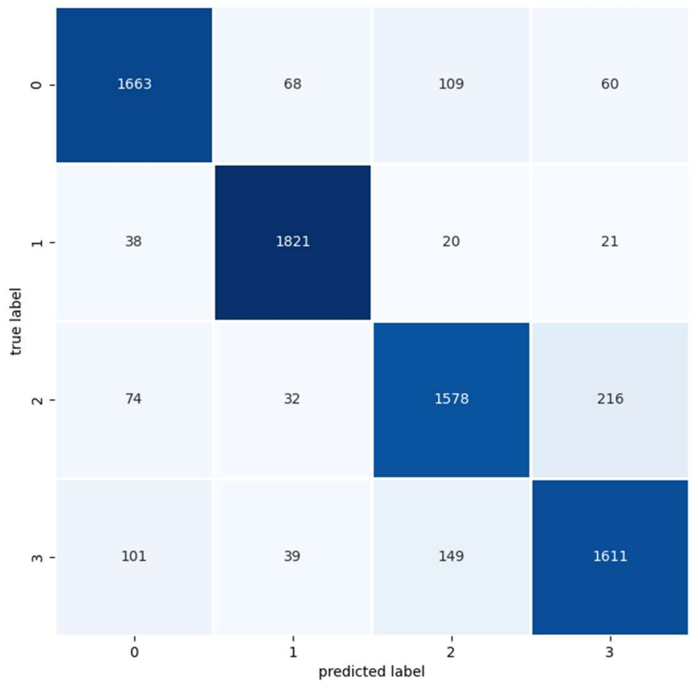
Experiment 8 – Confusion Matrix



Experiment 9 – Confusion Matrix



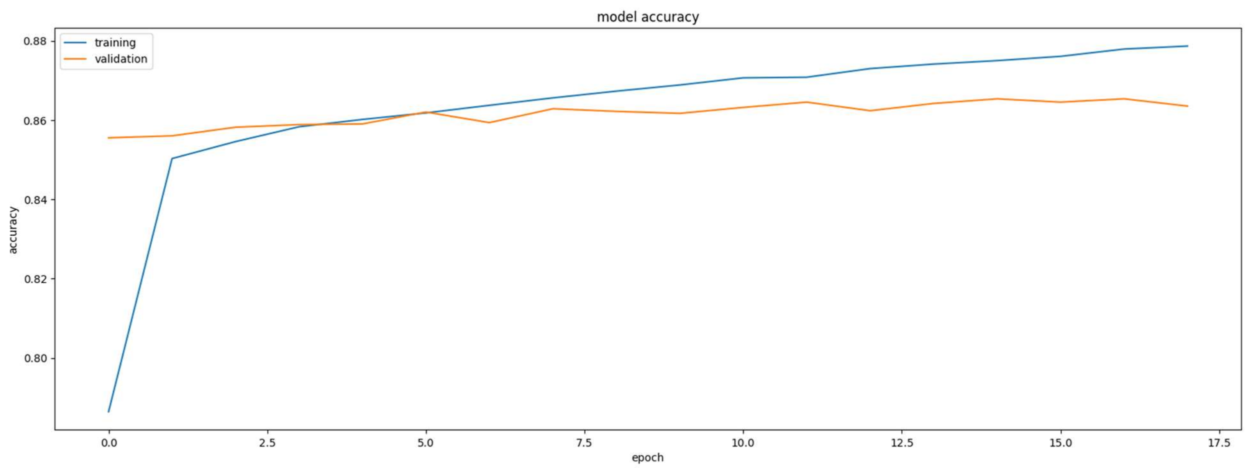
Experiment 10 – Confusion Matrix

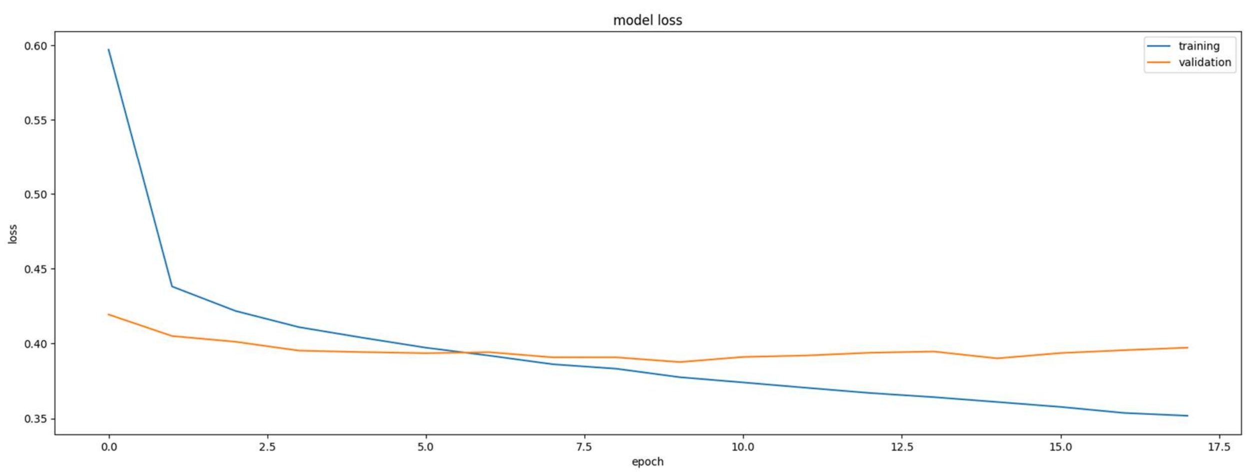


Appendix C – Accuracy and Loss Trends By Epoch Resulting From Experimental Model Fitting

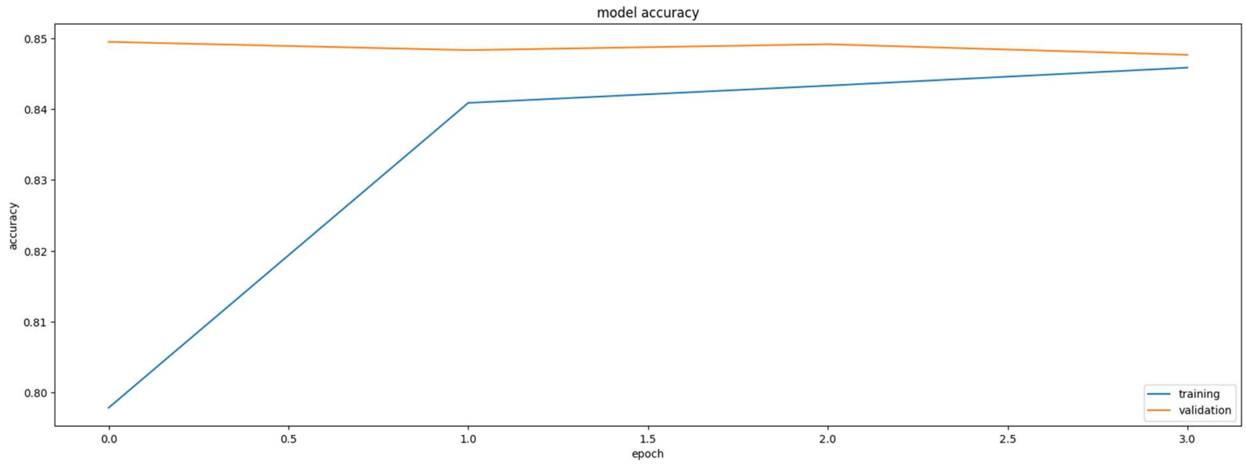
The graphs below display the training and validation accuracies generated throughout each epoch of training each of the classification models. Larger versions of these images and the code leveraged to generate these graphs are available in Appendix D.

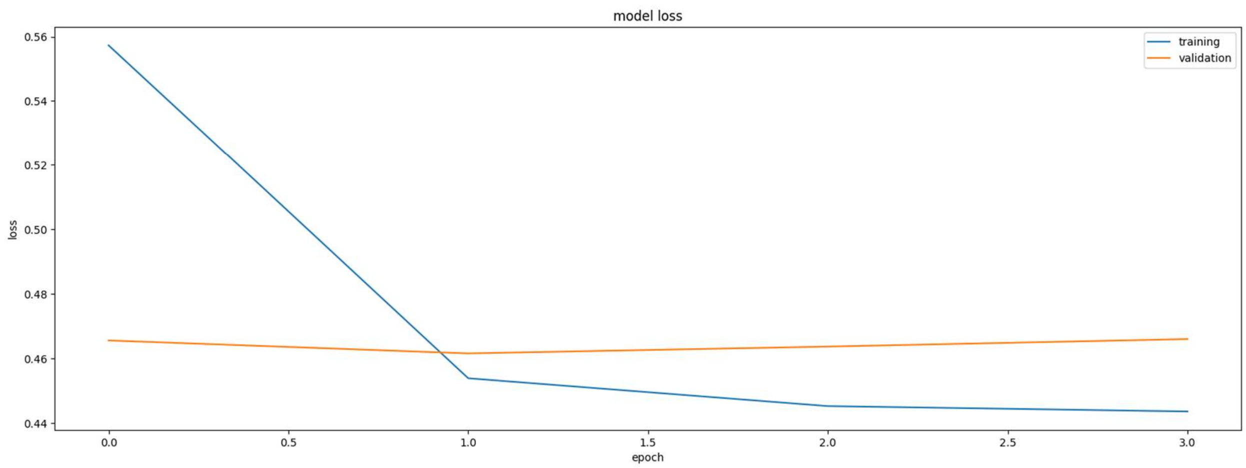
Experiment 1 – Accuracy and Loss Trends During Model Fitting



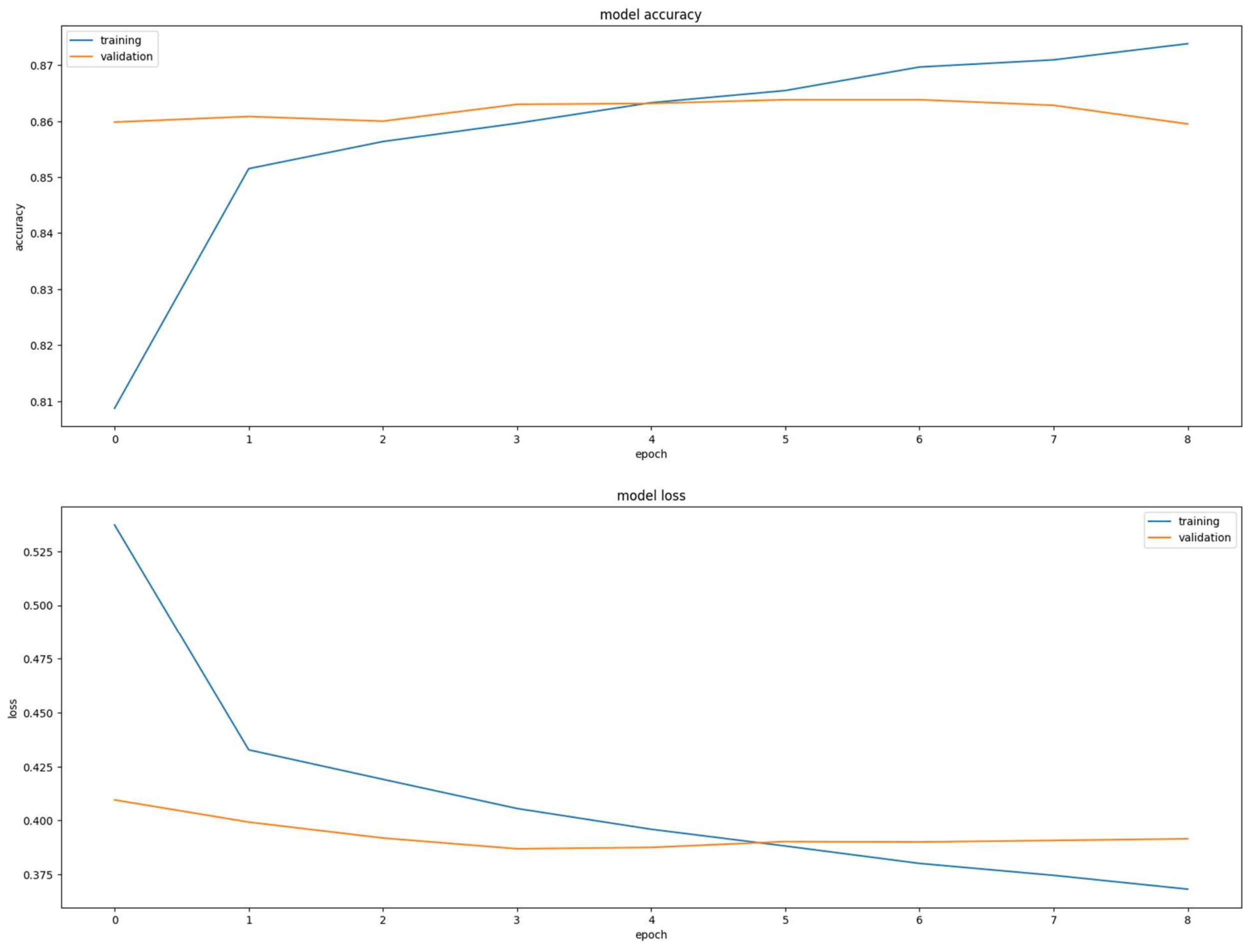


Experiment 2 – Accuracy and Loss Trends During Model Fitting

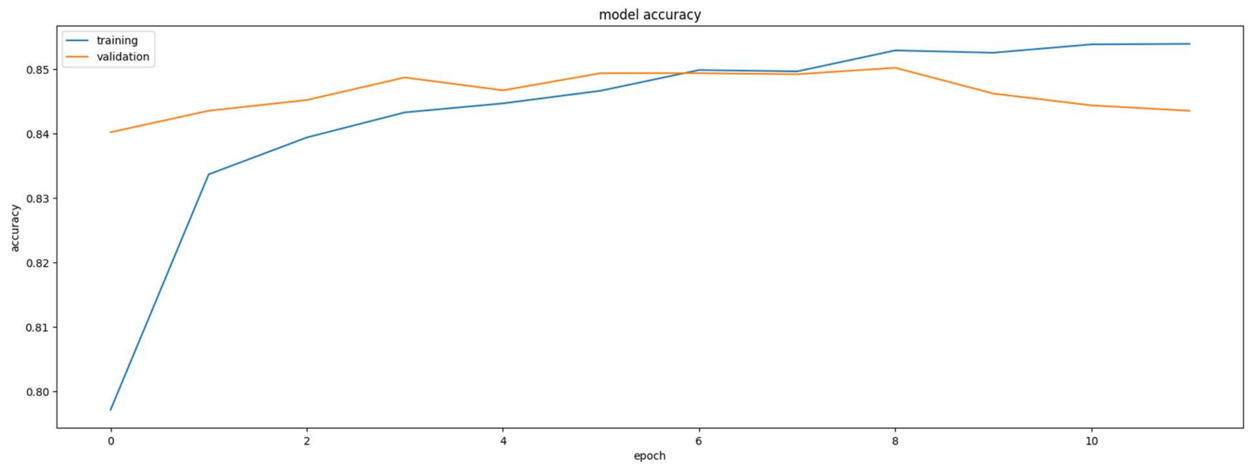


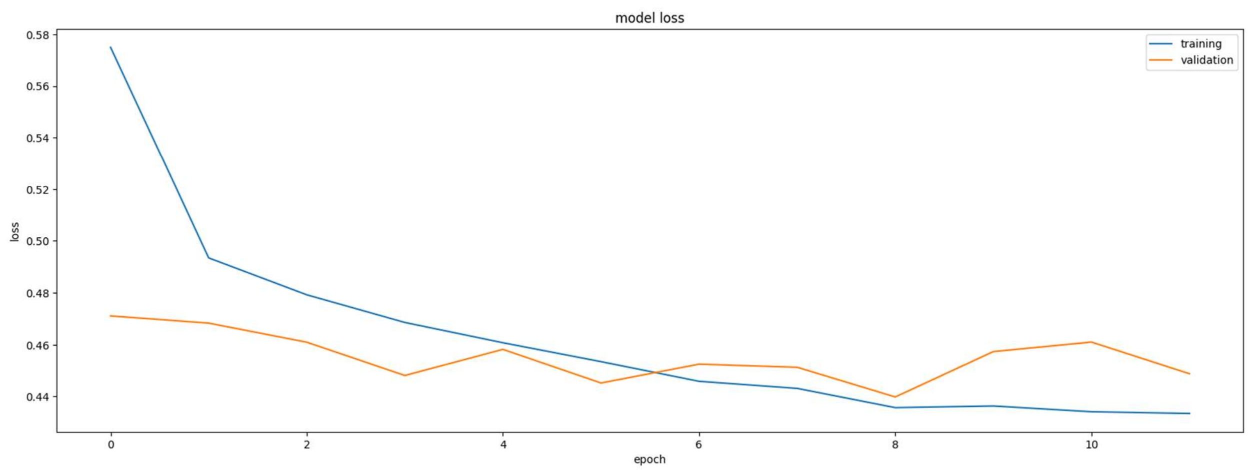


Experiment 3 – Accuracy and Loss Trends During Model Fitting

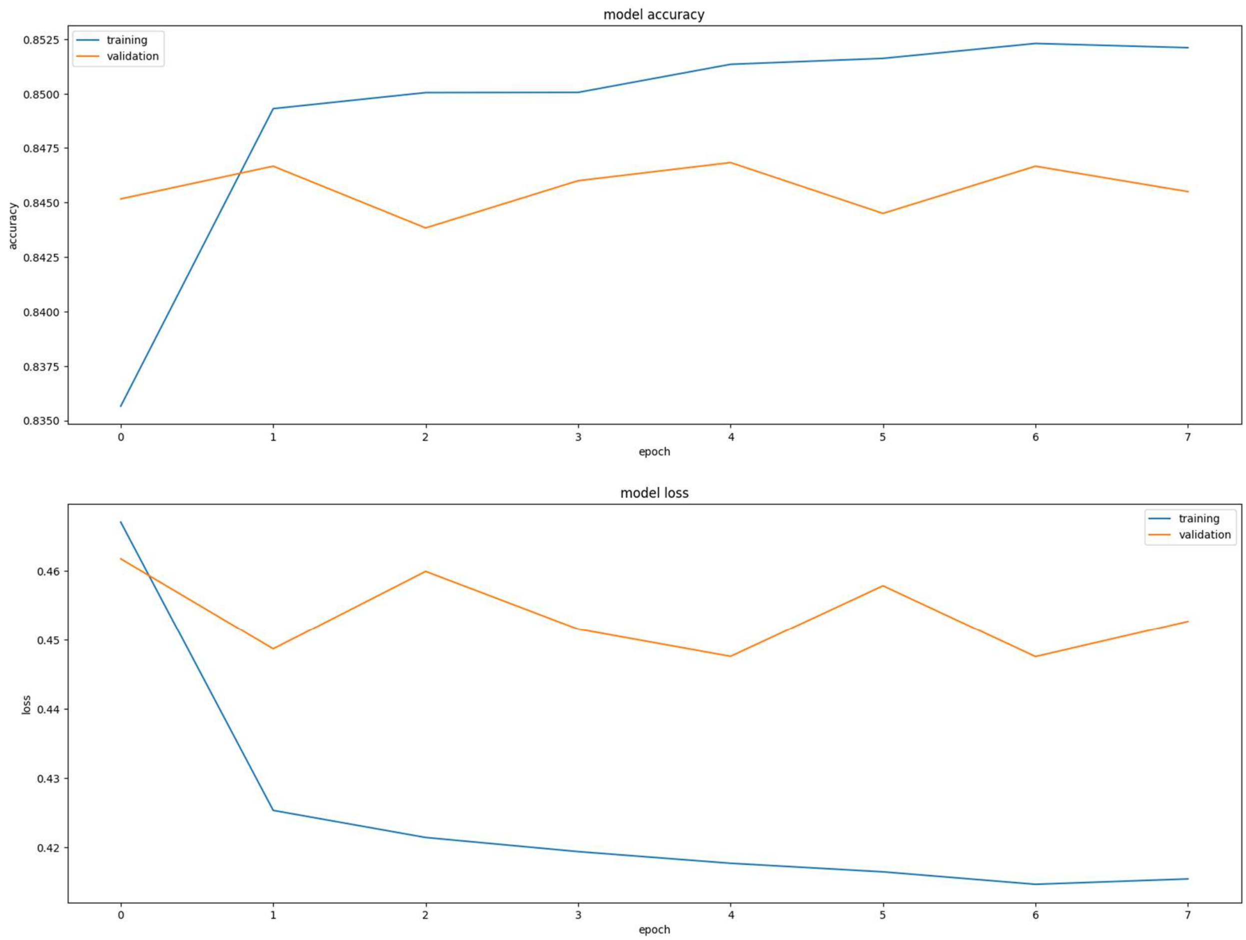


Experiment 4 – Accuracy and Loss Trends During Model Fitting

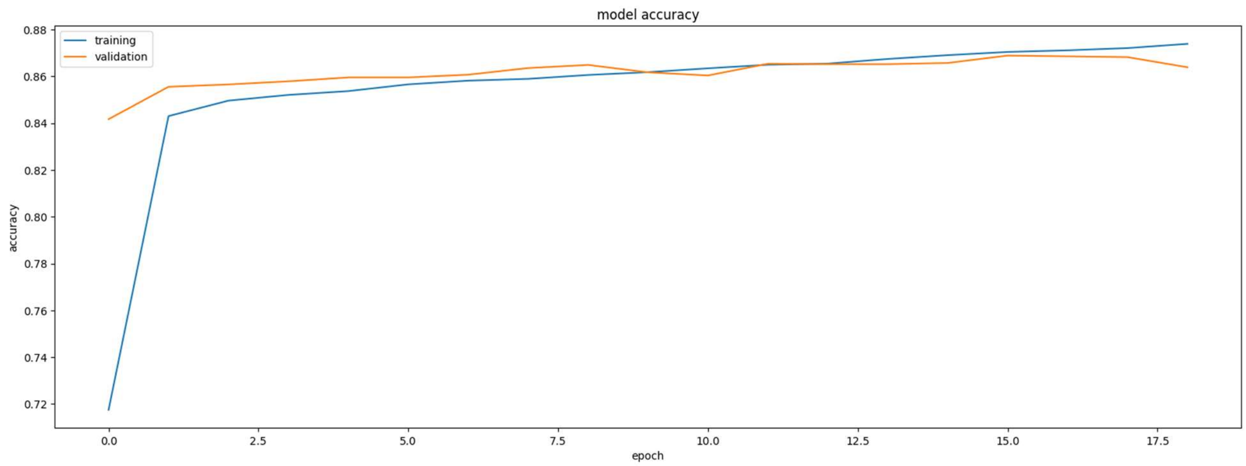


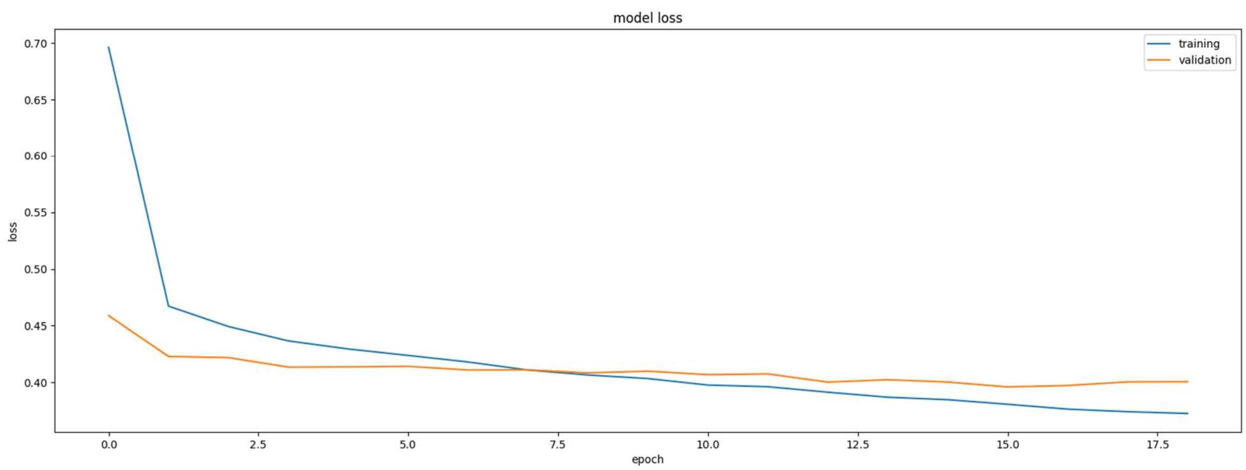


Experiment 5 – Accuracy and Loss Trends During Model Fitting

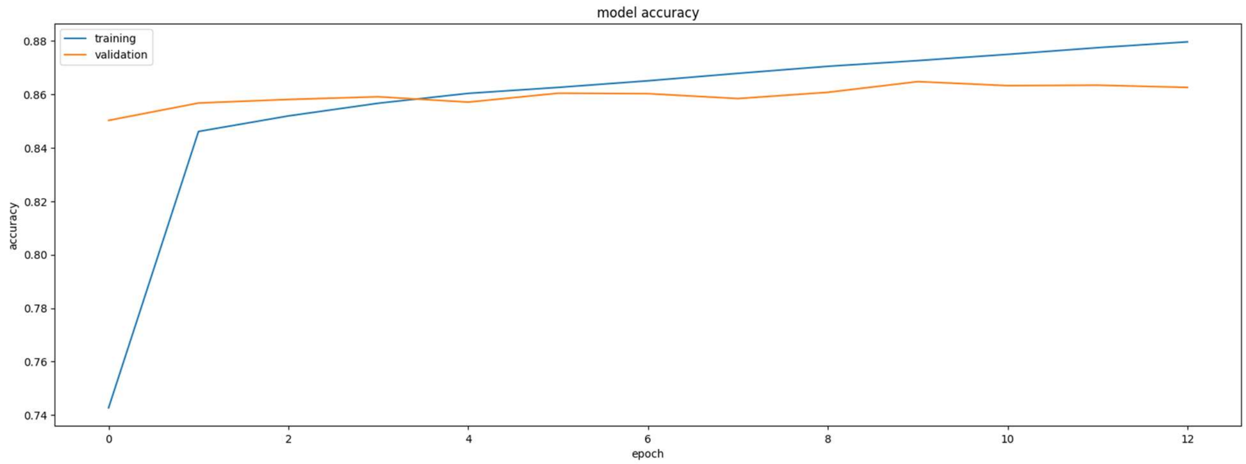


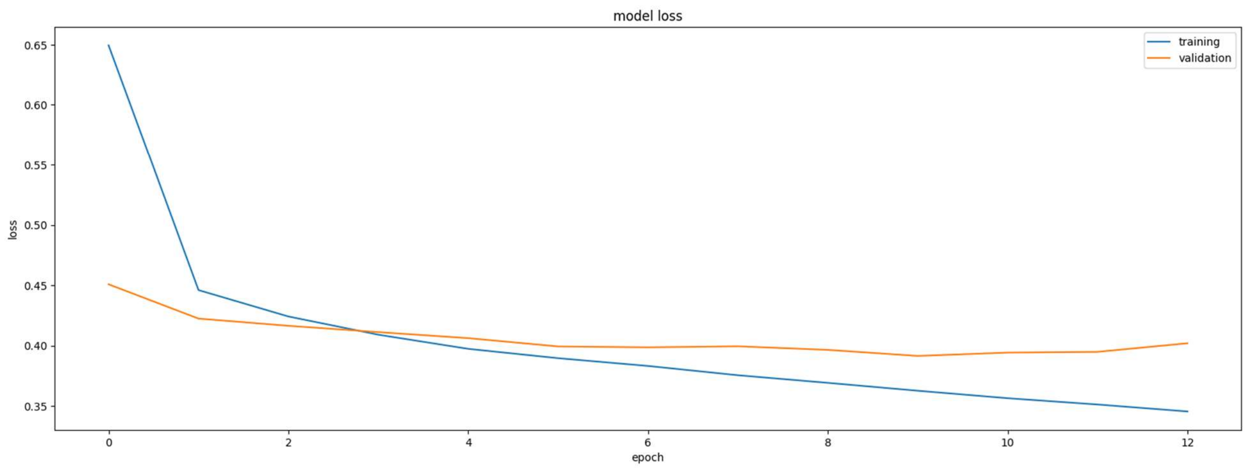
Experiment 6 – Accuracy and Loss Trends During Model Fitting



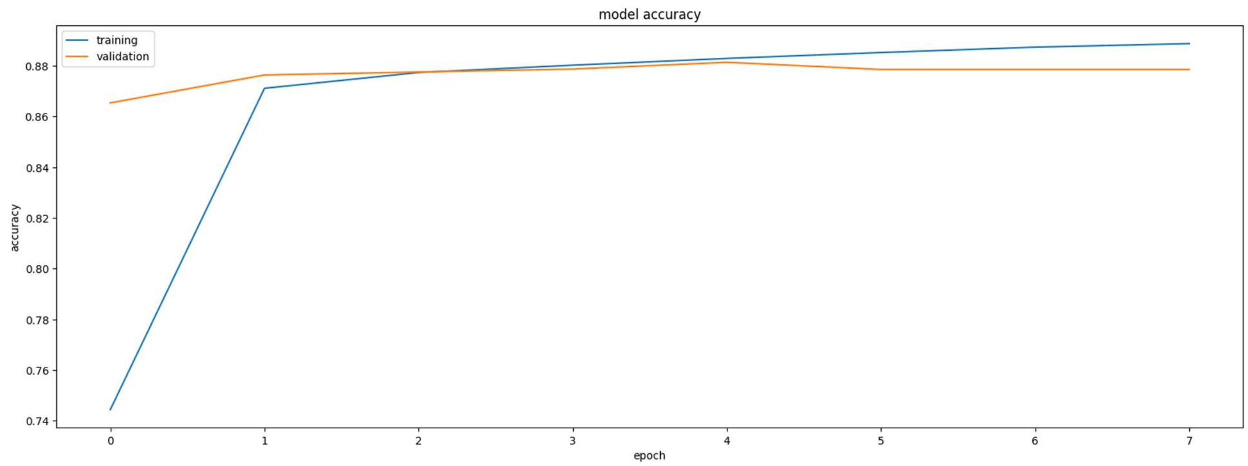


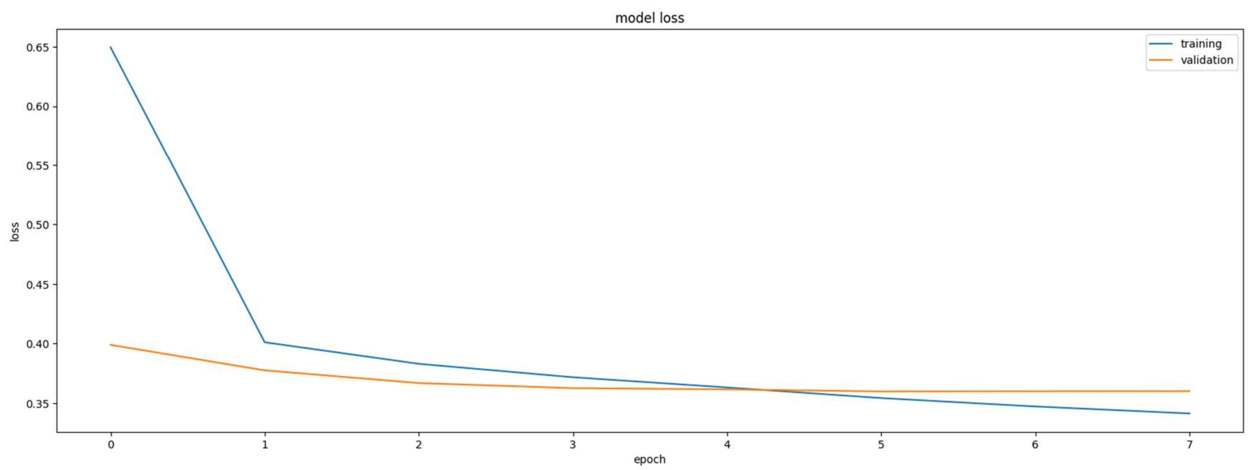
Experiment 7 – Accuracy and Loss Trends During Model Fitting



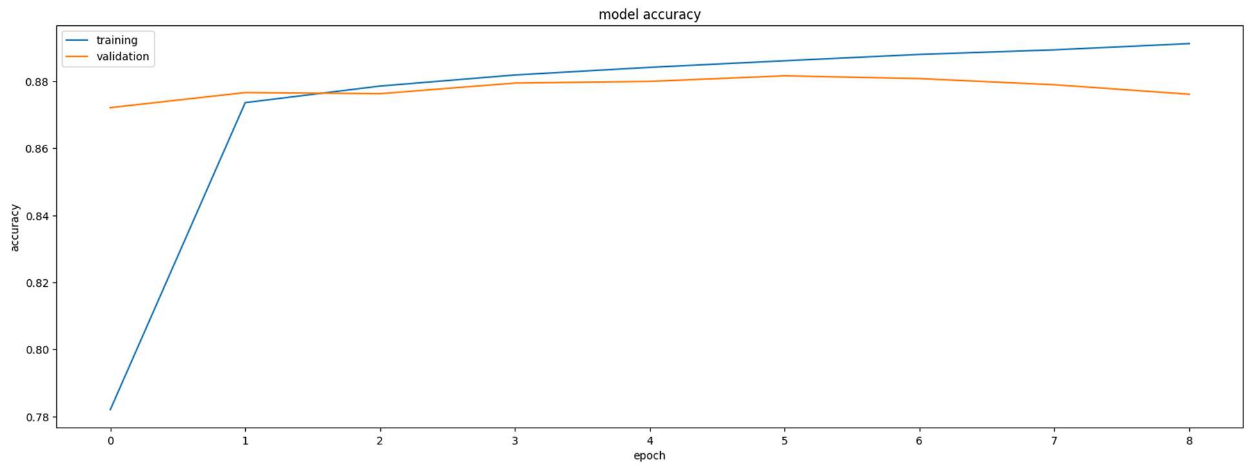


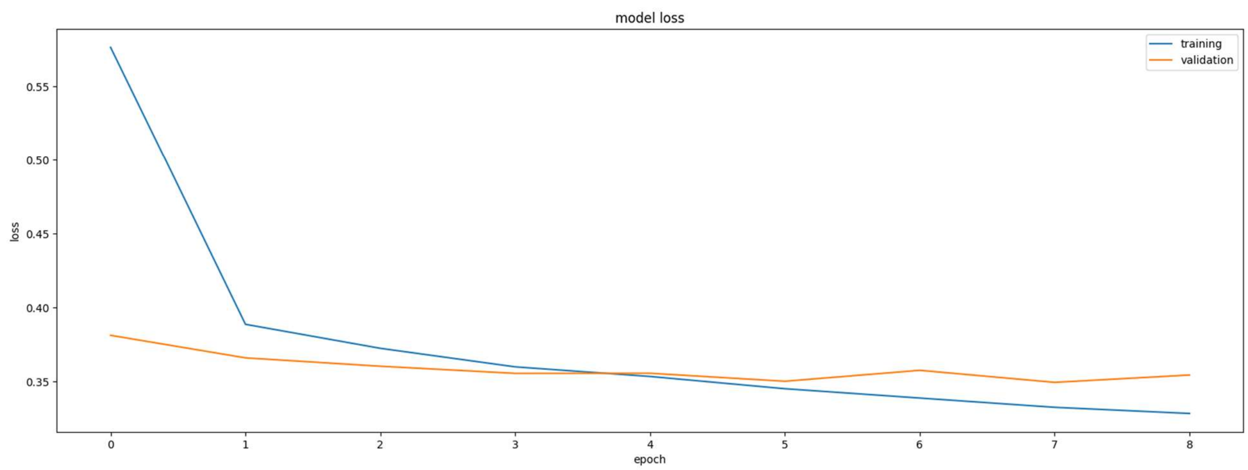
Experiment 8 – Accuracy and Loss Trends During Model Fitting



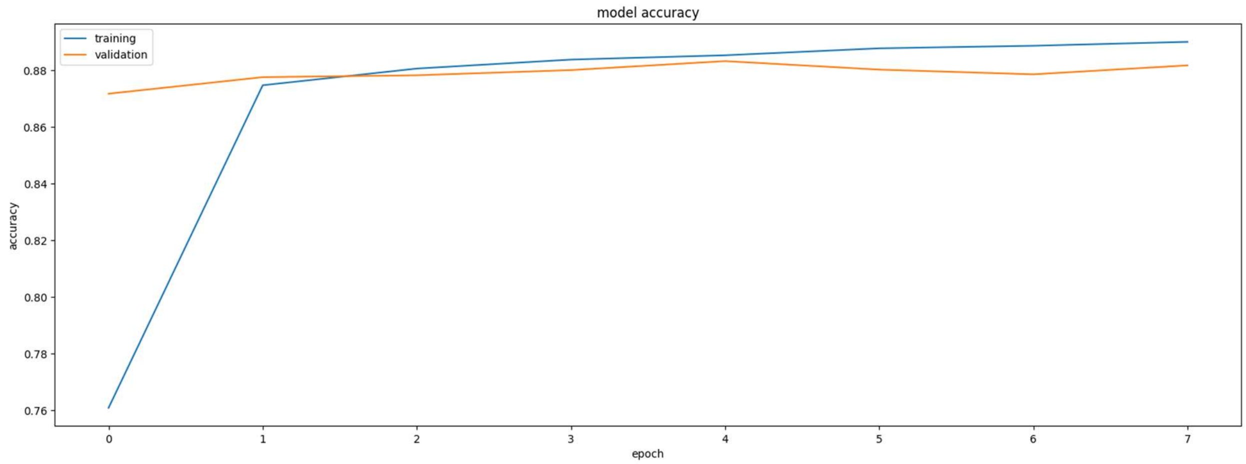


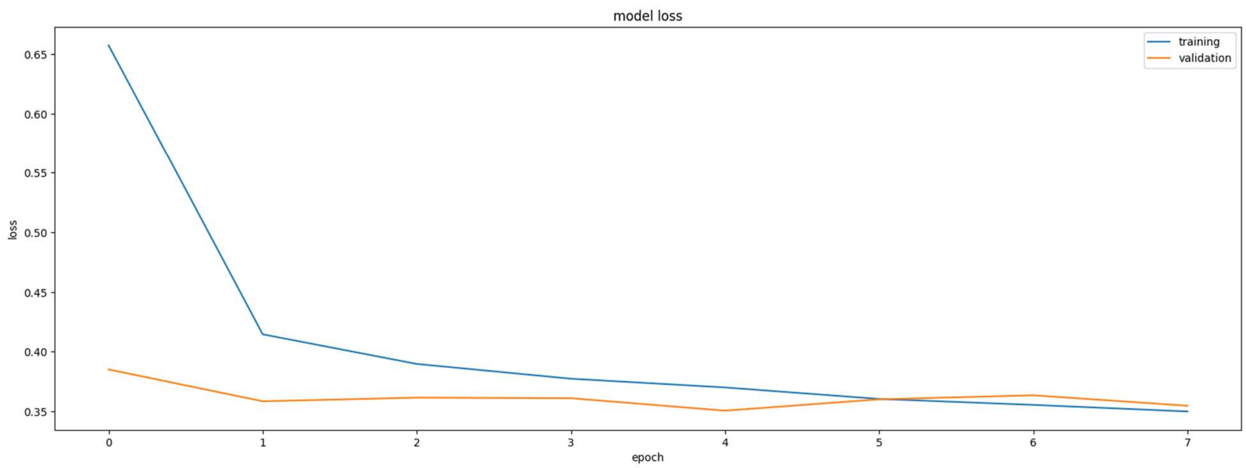
Experiment 9 – Accuracy and Loss Trends During Model Fitting





Experiment 10 – Accuracy and Loss Trends During Model Fitting





Appendix D - Supporting Python Code

Steve Desilets

November 5, 2023

1) Introduction

In this notebook, we aim to build natural language processing pipelines capable of effectively classifying text articles into their respective article categories. The underlying data that we leverage is the AG_News dataset, which includes over one million news articles corresponding to four categories. We aim to build a variety of models, including artificial neural networks, recurrent neural networks, long short term memory models, and transformer-based models to discover which methods are most effective for classifying articles into their respective categories.

1.1) Notebook Setup

First, let's set up this notebook by importing the relevant packages and by defining functions that we will use throughout our analysis.

In [5]:
```
import datetime
from packaging import version
from collections import Counter
import numpy as np
import pandas as pd
import time
import os
import re
import string

import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns

import nltk
from nltk.corpus import stopwords

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import accuracy_score

import tensorflow as tf
from tensorflow import keras
```

```
import tensorflow_datasets as tfds
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow.keras.backend as k
```

In [6]:
```
%matplotlib inline
np.set_printoptions(precision=3, suppress=True)
```

We can verify the version of TensorFlow in place.

In [7]:
```
print("This notebook requires TensorFlow 2.0 or above")
print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >=2
```

```
This notebook requires TensorFlow 2.0 or above
TensorFlow version:  2.14.0
```

Let's define visualization functions that we'll use throughout this analysis.

In [8]:
```
def print_validation_report(test_labels, predictions):
    print("Classification Report")
    print(classification_report(test_labels, predictions))
    print('Accuracy Score: {}'.format(accuracy_score(test_labels, predictions)))
    print('Root Mean Square Error: {}'.format(np.sqrt(MSE(test_labels, predictions))))

def plot_confusion_matrix(y_true, y_pred):
    mtx = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(8,8))
    sns.heatmap(mtx, annot=True, fmt='d', linewidths=.75,  cbar=False, ax=ax,cmap='Blu
    #  square=True,
    plt.ylabel('true label')
    plt.xlabel('predicted label')

def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])

def display_training_curves(training, validation, title, subplot):
  ax = plt.subplot(subplot)
  ax.plot(training)
  ax.plot(validation)
  ax.set_title('model '+ title)
  ax.set_ylabel(title)
  ax.set_xlabel('epoch')
  ax.legend(['training', 'validation'])
```

Let's mount to the Google Colab environment

In [9]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

1.2) Exploratory Data Analysis

Now that we've set up our notebook, let's load the subset of the AG news dataset.

Load AG News Subset Data

ag_news_subset
See https://www.tensorflow.org/datasets/catalog/ag_news_subset

Get all the words in the documents (as well as the number of words in each document) by using the encoder to get the indices associated with each token and then translating the indices to tokens. But first we need to get the "unpadded" new articles so that we can get their length.

In [10]:
```
#register  ag_news_subset so that tfds.load doesn't generate a checksum (mismatch) err
!python -m tensorflow_datasets.scripts.download_and_prepare \
        --register_checksums --datasets=ag_news_subset

# https://www.tensorflow.org/datasets/splits
# The full `train` and `test` splits, interleaved together.
ri = tfds.core.ReadInstruction('train') + tfds.core.ReadInstruction('test')
dataset_all, info = tfds.load('ag_news_subset', with_info=True,  split=ri, as_supervis
text_only_dataset_all=dataset_all.map(lambda x, y: x)
```

```
W1106 00:47:30.647041 134997017923584 download_and_prepare.py:46] ***`tfds build` sho
uld be used instead of `download_and_prepare`.***
INFO[build.py]: Loading dataset ag_news_subset from imports: tensorflow_datasets.data
sets.ag_news_subset.ag_news_subset_dataset_builder
2023-11-06 00:47:31.166595: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.c
c:9342] Unable to register cuDNN factory: Attempting to register factory for plugin c
uDNN when one has already been registered
2023-11-06 00:47:31.166668: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.c
c:609] Unable to register cuFFT factory: Attempting to register factory for plugin cu
FFT when one has already been registered
2023-11-06 00:47:31.166754: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.
cc:1518] Unable to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
2023-11-06 00:47:32.530617: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] T
F-TRT Warning: Could not find TensorRT
INFO[utils.py]: NumExpr defaulting to 2 threads.
2023-11-06 00:47:34.253586: W tensorflow/tsl/platform/cloud/google_auth_provider.cc:1
84] All attempts to get a Google authentication bearer token failed, returning an emp
ty token. Retrieving token from files failed with "NOT_FOUND: Could not locate the cr
edentials file.". Retrieving token from GCE failed with "NOT_FOUND: Error executing a
n HTTP request: HTTP response code 404".
INFO[dataset_info.py]: Load pre-computed DatasetInfo (eg: splits, num examples,...) f
rom GCS: ag_news_subset/1.0.0
INFO[dataset_info.py]: Load dataset info from /tmp/tmpgehqlqp0tfds
INFO[dataset_info.py]: For 'ag_news_subset/1.0.0': fields info.[description, splits,
supervised_keys, module_name] differ on disk and in the code. Keeping the one from co
de.
INFO[build.py]: download_and_prepare for dataset ag_news_subset/1.0.0...
INFO[dataset_builder.py]: Generating dataset ag_news_subset (/root/tensorflow_dataset
s/ag_news_subset/1.0.0)
Downloading and preparing dataset 11.24 MiB (download: 11.24 MiB, generated: 35.79 Mi
B, total: 47.03 MiB) to /root/tensorflow_datasets/ag_news_subset/1.0.0...
Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]




INFO[download_manager.py]: Downloading https://drive.google.com/uc?export=download&id
=0Bz8a_Dbh9QhbUDNpeUdjb0wxRms into /root/tensorflow_datasets/downloads/ucexport_downl
oad_id_0Bz8a_Dbh9QhbUDNpeUdjb0wxj4g1umFAV8OV-uDwxSJR0LdxO_k1jxMuFWwAfNX9jos.tmp.c5f5b
91113c947bcbbf2c94ccca05627...
Dl Completed...: 0 url [00:00, ? url/s]

Extraction completed...: 0 file [00:00, ? file/s]
Dl Completed...:   0% 0/1 [00:00<?, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:   0% 0/11 [00:09<?, ? MiB/s]

Extraction completed...: 0 file [00:09, ? file/s]
Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:   9% 1/11 [00:09<01:30,  9.05s/ MiB]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  18% 2/11 [00:09<01:21,  9.05s/ MiB]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  27% 3/11 [00:09<01:12,  9.05s/ MiB]
```

```
Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  36% 4/11 [00:09<01:03,  9.05s/ MiB]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  45% 5/11 [00:09<00:54,  9.05s/ MiB]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  55% 6/11 [00:09<00:45,  9.05s/ MiB]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  64% 7/11 [00:09<00:36,  9.05s/ MiB]

Extraction completed...: 0 file [00:09, ? file/s]
Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  73% 8/11 [00:09<00:02,  1.19 MiB/s]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  82% 9/11 [00:09<00:01,  1.19 MiB/s]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...:  91% 10/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...:   0% 0/1 [00:09<?, ? url/s]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Extraction completed...:   0% 0/4 [00:09<?, ? file/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Dl Completed...: 100% 1/1 [00:09<00:00,  9.21s/ url]
Dl Size...: 100% 11/11 [00:09<00:00,  1.19 MiB/s]

Extraction completed...: 100% 4/4 [00:09<00:00,  2.40s/ file]
Dl Size...: 100% 11/11 [00:09<00:00,  1.14 MiB/s]
Dl Completed...: 100% 1/1 [00:09<00:00,  9.61s/ url]
Generating splits...:   0% 0/2 [00:00<?, ? splits/s]
Generating train examples...:   0% 0/120000 [00:00<?, ? examples/s]
```

```
Generating train examples...:   8% 9857/120000 [00:01<00:11, 9856.57 examples/s]
Generating train examples...:  17% 20001/120000 [00:02<00:09, 10024.66 examples/s]
Generating train examples...:  25% 30276/120000 [00:03<00:08, 10138.67 examples/s]
Generating train examples...:  34% 40632/120000 [00:04<00:07, 10224.24 examples/s]
Generating train examples...:  42% 50888/120000 [00:05<00:06, 10235.55 examples/s]
Generating train examples...:  51% 61187/120000 [00:06<00:05, 10256.96 examples/s]
Generating train examples...:  60% 71444/120000 [00:07<00:04, 10254.23 examples/s]
Generating train examples...:  68% 81701/120000 [00:08<00:03, 10254.96 examples/s]
Generating train examples...:  77% 91956/120000 [00:09<00:02, 10236.66 examples/s]
Generating train examples...:  85% 102332/120000 [00:10<00:01, 10279.32 examples/s]
Generating train examples...:  94% 112612/120000 [00:11<00:00, 8607.86 examples/s]

Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-train.tfrecord*...:   0% 0/120000 [00:00<?, ? examples/s]
Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-train.tfrecord*...:   0% 1/120000 [00:00<6:09:09,  5.42 examples/s]
Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-train.tfrecord*...:  20% 24464/120000 [00:00<00:00, 106725.59 examples/s]
Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-train.tfrecord*...:  41% 49758/120000 [00:00<00:00, 162857.89 examples/s]
Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-train.tfrecord*...:  63% 75086/120000 [00:00<00:00, 194877.59 examples/s]
Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-train.tfrecord*...:  83% 99702/120000 [00:00<00:00, 212096.56 examples/s]
INFO[writer.py]: Done writing /root/tensorflow_datasets/ag_news_subset/1.0.0.incomple
teTOFU7L/ag_news_subset-train.tfrecord*. Number of examples: 120000 (shards: [12000
0])
Generating splits...:  50% 1/2 [00:13<00:13, 13.53s/ splits]
Generating test examples...:   0% 0/7600 [00:00<?, ? examples/s]
Generating test examples...:  87% 6594/7600 [00:01<00:00, 6593.58 examples/s]

Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incompleteTOFU7L/ag_news_sub
set-test.tfrecord*...:   0% 0/7600 [00:00<?, ? examples/s]
INFO[writer.py]: Done writing /root/tensorflow_datasets/ag_news_subset/1.0.0.incomple
teTOFU7L/ag_news_subset-test.tfrecord*. Number of examples: 7600 (shards: [7600])
Dataset ag_news_subset downloaded and prepared to /root/tensorflow_datasets/ag_news_s
ubset/1.0.0. Subsequent calls will reuse this data.
INFO[build.py]: Dataset generation complete...

tfds.core.DatasetInfo(
    name='ag_news_subset',
    full_name='ag_news_subset/1.0.0',
    description="""
    AG is a collection of more than 1 million news articles. News articles have been
    gathered from more than 2000 news sources by ComeToMyHead in more than 1 year of
    activity. ComeToMyHead is an academic news search engine which has been running
    since July, 2004. The dataset is provided by the academic comunity for research
    purposes in data mining (clustering, classification, etc), information retrieval
    (ranking, search, etc), xml, data compression, data streaming, and any other
    non-commercial activity. For more information, please refer to the link
    http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html .

    The AG's news topic classification dataset is constructed by Xiang Zhang
    (xiang.zhang@nyu.edu) from the dataset above. It is used as a text
    classification benchmark in the following paper: Xiang Zhang, Junbo Zhao, Yann
    LeCun. Character-level Convolutional Networks for Text Classification. Advances
    in Neural Information Processing Systems 28 (NIPS 2015).

    The AG's news topic classification dataset is constructed by choosing 4 largest
    classes from the original corpus. Each class contains 30,000 training samples
```

```
    and 1,900 testing samples. The total number of training samples is 120,000 and
    testing 7,600.
    """,
    homepage='https://arxiv.org/abs/1509.01626',
    data_dir=PosixGPath('/tmp/tmpgehqlqp0tfds'),
    file_format=tfrecord,
    download_size=11.24 MiB,
    dataset_size=35.79 MiB,
    features=FeaturesDict({
        'description': Text(shape=(), dtype=string),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=4),
        'title': Text(shape=(), dtype=string),
    }),
    supervised_keys=('description', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=7600, num_shards=1>,
        'train': <SplitInfo num_examples=120000, num_shards=1>,
    },
    citation="""@misc{zhang2015characterlevel,
        title={Character-level Convolutional Networks for Text Classification},
        author={Xiang Zhang and Junbo Zhao and Yann LeCun},
        year={2015},
        eprint={1509.01626},
        archivePrefix={arXiv},
        primaryClass={cs.LG}
    }""",
)
```

Let's conduct exploratory data analysis of this ag_news_subset dataset. We combined the training and test data for a total of 127,600 news articles. We can begin by observing the first 10 rows of this dataset.

In [11]:
```
tfds.as_dataframe(dataset_all.take(10),info)
```

Out[11]:

	description	label
0	AMD #39;s new dual-core Opteron chip is designed mainly for corporate computing applications, including databases, Web services, and financial transactions.	3 (Sci/Tech)
1	Reuters - Major League Baseball\Monday announced a decision on the appeal filed by Chicago Cubs\pitcher Kerry Wood regarding a suspension stemming from an\incident earlier this season.	1 (Sports)
2	President Bush #39;s quot;revenue-neutral quot; tax reform needs losers to balance its winners, and people claiming the federal deduction for state and local taxes may be in administration planners #39; sights, news reports say.	2 (Business)
3	Britain will run out of leading scientists unless science education is improved, says Professor Colin Pillinger.	3 (Sci/Tech)
4	London, England (Sports Network) - England midfielder Steven Gerrard injured his groin late in Thursday #39;s training session, but is hopeful he will be ready for Saturday #39;s World Cup qualifier against Austria.	1 (Sports)
5	TOKYO - Sony Corp. is banking on the $3 billion deal to acquire Hollywood studio Metro-Goldwyn-Mayer Inc...	0 (World)
6	Giant pandas may well prefer bamboo to laptops, but wireless technology is helping researchers in China in their efforts to protect the engandered animals living in the remote Wolong Nature Reserve.	3 (Sci/Tech)
7	VILNIUS, Lithuania - Lithuania #39;s main parties formed an alliance to try to keep a Russian-born tycoon and his populist promises out of the government in Sunday #39;s second round of parliamentary elections in this Baltic country.	0 (World)
8	Witnesses in the trial of a US soldier charged with abusing prisoners at Abu Ghraib have told the court that the CIA sometimes directed abuse and orders were received from military command to toughen interrogations.	0 (World)
9	Dan Olsen of Ponte Vedra Beach, Fla., shot a 7-under 65 Thursday to take a one-shot lead after two rounds of the PGA Tour qualifying tournament.	1 (Sports)

Let's review the labels for the categories of articles in this dataset.

In [12]:

```
categories =dict(enumerate(info.features["label"].names))
print(f'Dictionary: ',categories)
```

```
Dictionary:  {0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}
```

Let's observe the number of observations that correspond to each class in the dataset.

In [13]:

```
train_categories = [categories[label] for label in dataset_all.map(lambda text, label:
Counter(train_categories).most_common()
```

Out[13]:

```
[('Sci/Tech', 31900), ('Sports', 31900), ('Business', 31900), ('World', 31900)]
```

We see that the 127,600 articles are evenly distributed across the four classes.

Let's do a bit of data preprocessing to enable further exploratory data analysis.

We can start by making the corpus all lowercase, stripping punctuation, and removing stopwords.

In [14]:
```
def custom_stopwords(input_text):
    lowercase = tf.strings.lower(input_text)
    stripped_punct = tf.strings.regex_replace(lowercase
                                  ,'[%s]' % re.escape(string.punctuation)
                                  ,'')
    return tf.strings.regex_replace(stripped_punct, r'\b(' + r'|'.join(STOPWORDS) + r'
```

In [15]:
```
nltk.download('stopwords',quiet=True)
STOPWORDS = stopwords.words("english")
```

In [16]:
```
%%time
max_tokens = None
text_vectorization=layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    standardize=custom_stopwords
)
text_vectorization.adapt(text_only_dataset_all)
```

```
CPU times: user 2min 44s, sys: 11 s, total: 2min 55s
Wall time: 2min 38s
```

In [17]:
```
%%time
doc_sizes = []
corpus = []
for example, _ in dataset_all.as_numpy_iterator():
  enc_example = text_vectorization(example)
  doc_sizes.append(len(enc_example))
  corpus+=list(enc_example.numpy())
```

```
CPU times: user 14min 23s, sys: 8.14 s, total: 14min 31s
Wall time: 15min 21s
```

In [22]:
```
print(f"There are {len(corpus)} words in the corpus of {len(doc_sizes)} news articles.
print(f"Each news article has between {min(doc_sizes)} and {max(doc_sizes)} tokens in
```

```
There are 2579419 words in the corpus of 127600 news articles.
Each news article has between 2 and 95 tokens in it.
```

In [23]:
```
print(f"There are {len(text_vectorization.get_vocabulary())} vocabulary words in the c
```

```
There are 95827 vocabulary words in the corpus.
```

Let's observe the first 50 words in our vocabulary.

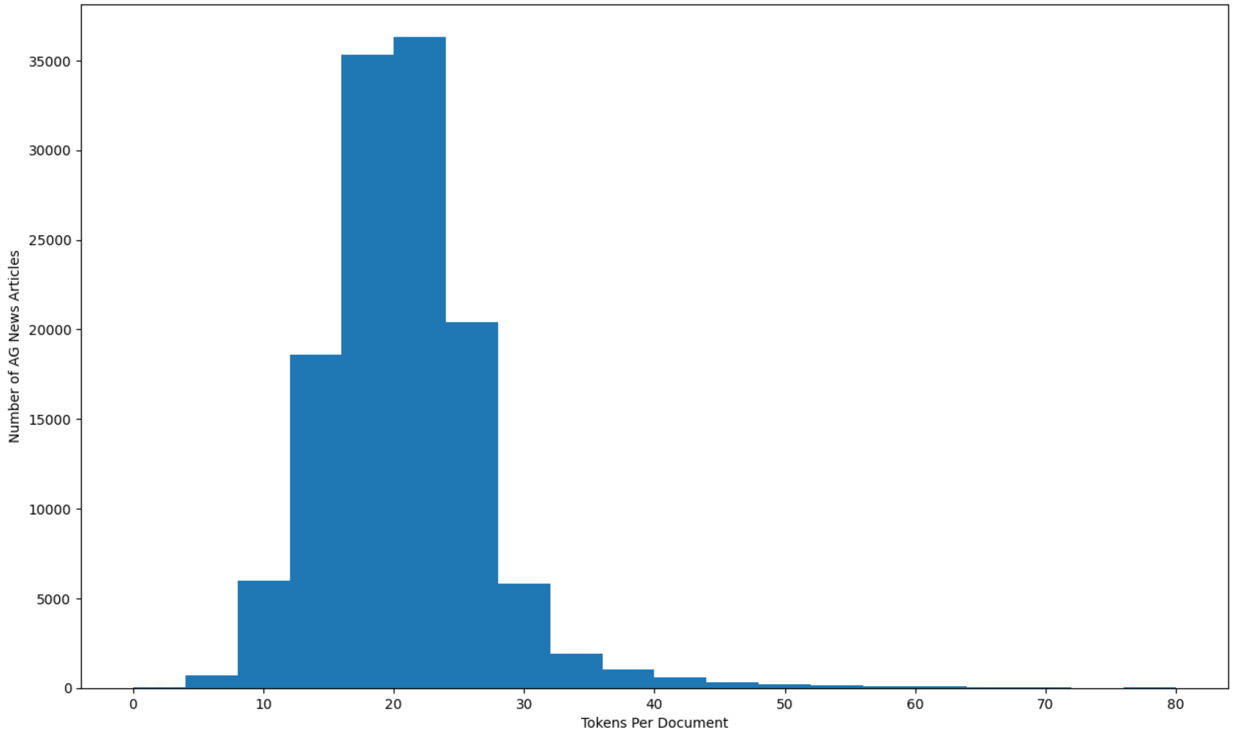
In [24]:
```
vocab = np.array(text_vectorization.get_vocabulary())
print(vocab[:50])
```

```
['' '[UNK]' '39s' 'said' 'new' 'us' 'reuters' 'ap' 'two' 'first' 'monday'
 'wednesday' 'tuesday' 'thursday' 'company' 'friday' 'inc' 'one' 'world'
 'yesterday' 'last' 'york' 'year' 'president' 'million' 'oil' 'corp'
 'united' 'would' 'sunday' 'years' 'week' 'people' 'today' 'three'
 'government' 'could' 'quot' 'group' 'time' 'percent' 'game' 'saturday'
 'software' 'night' 'next' 'prices' 'iraq' 'security' 'announced']
```

Let's examine the distribution of the number of tokens per document.

In [25]:
```
plt.figure(figsize=(15,9))
plt.hist(doc_sizes, bins=20,range = (0,80))
```

```
plt.xlabel("Tokens Per Document")
plt.ylabel("Number of AG News Articles");
```



1.3) Data Pre-Processing

Now that we've conducted exploratory data analysis, let's preprocess the text.

In [26]:
```
# register  ag_news_subset so that tfds.load doesn't generate a checksum (mismatch) er
!python -m tensorflow_datasets.scripts.download_and_prepare --register_checksums --dat

dataset,info=\
tfds.load('ag_news_subset', with_info=True,  split=['train[:95%]','train[95%:]', 'test
          , as_supervised=True)

train_ds, val_ds, test_ds = dataset
text_only_train_ds = train_ds.map(lambda x, y: x)
```

```
W1106 01:07:58.089279 135122770280448 download_and_prepare.py:46] ***`tfds build` sho
uld be used instead of `download_and_prepare`.***
INFO[build.py]: Loading dataset ag_news_subset from imports: tensorflow_datasets.data
sets.ag_news_subset.ag_news_subset_dataset_builder
INFO[dataset_info.py]: Load dataset info from /root/tensorflow_datasets/ag_news_subse
t/1.0.0
INFO[build.py]: download_and_prepare for dataset ag_news_subset/1.0.0...
INFO[dataset_builder.py]: Reusing dataset ag_news_subset (/root/tensorflow_datasets/a
g_news_subset/1.0.0)
INFO[build.py]: Dataset generation complete...

tfds.core.DatasetInfo(
    name='ag_news_subset',
    full_name='ag_news_subset/1.0.0',
    description="""
    AG is a collection of more than 1 million news articles. News articles have been
    gathered from more than 2000 news sources by ComeToMyHead in more than 1 year of
    activity. ComeToMyHead is an academic news search engine which has been running
    since July, 2004. The dataset is provided by the academic comunity for research
    purposes in data mining (clustering, classification, etc), information retrieval
    (ranking, search, etc), xml, data compression, data streaming, and any other
    non-commercial activity. For more information, please refer to the link
    http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html .

    The AG's news topic classification dataset is constructed by Xiang Zhang
    (xiang.zhang@nyu.edu) from the dataset above. It is used as a text
    classification benchmark in the following paper: Xiang Zhang, Junbo Zhao, Yann
    LeCun. Character-level Convolutional Networks for Text Classification. Advances
    in Neural Information Processing Systems 28 (NIPS 2015).

    The AG's news topic classification dataset is constructed by choosing 4 largest
    classes from the original corpus. Each class contains 30,000 training samples
    and 1,900 testing samples. The total number of training samples is 120,000 and
    testing 7,600.
    """,
    homepage='https://arxiv.org/abs/1509.01626',
    data_dir='/root/tensorflow_datasets/ag_news_subset/1.0.0',
    file_format=tfrecord,
    download_size=11.24 MiB,
    dataset_size=35.79 MiB,
    features=FeaturesDict({
        'description': Text(shape=(), dtype=string),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=4),
        'title': Text(shape=(), dtype=string),
    }),
    supervised_keys=('description', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=7600, num_shards=1>,
        'train': <SplitInfo num_examples=120000, num_shards=1>,
    },
    citation="""@misc{zhang2015characterlevel,
        title={Character-level Convolutional Networks for Text Classification},
        author={Xiang Zhang and Junbo Zhao and Yann LeCun},
        year={2015},
        eprint={1509.01626},
        archivePrefix={arXiv},
        primaryClass={cs.LG}
    }""",
```

```
)
```

In [27]:
```
max_length = 96
max_tokens = 1000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
    standardize=custom_stopwords
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

2) Model 1 - Bi-Directional Long Short Term Memory (LSTM) Model

2.1) Build The Model

In [29]:
```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_One",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]



start_time = datetime.datetime.now()

history=model.fit(int_train_ds, validation_data=int_val_ds, epochs=200, callbacks=call

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")
```

```
model = keras.models.load_model("Model_One")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 tf.one_hot (TFOpLambda)     (None, None, 1000)        0

 bidirectional (Bidirection  (None, 64)                264448
 al)

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 4)                 260

=================================================================
Total params: 264708 (1.01 MB)
Trainable params: 264708 (1.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 691s 193ms/step - loss: 0.5970 - accurac
y: 0.7864 - val_loss: 0.4194 - val_accuracy: 0.8555
Epoch 2/200
3563/3563 [==============================] - 690s 194ms/step - loss: 0.4381 - accurac
y: 0.8503 - val_loss: 0.4050 - val_accuracy: 0.8560
Epoch 3/200
3563/3563 [==============================] - 674s 189ms/step - loss: 0.4218 - accurac
y: 0.8545 - val_loss: 0.4012 - val_accuracy: 0.8582
Epoch 4/200
3563/3563 [==============================] - 686s 192ms/step - loss: 0.4110 - accurac
y: 0.8583 - val_loss: 0.3953 - val_accuracy: 0.8588
Epoch 5/200
3563/3563 [==============================] - 736s 206ms/step - loss: 0.4039 - accurac
y: 0.8601 - val_loss: 0.3943 - val_accuracy: 0.8590
Epoch 6/200
3563/3563 [==============================] - 726s 204ms/step - loss: 0.3972 - accurac
y: 0.8618 - val_loss: 0.3935 - val_accuracy: 0.8620
Epoch 7/200
3563/3563 [==============================] - 699s 196ms/step - loss: 0.3919 - accurac
y: 0.8637 - val_loss: 0.3942 - val_accuracy: 0.8593
Epoch 8/200
3563/3563 [==============================] - 714s 200ms/step - loss: 0.3862 - accurac
y: 0.8656 - val_loss: 0.3908 - val_accuracy: 0.8628
Epoch 9/200
3563/3563 [==============================] - 701s 197ms/step - loss: 0.3832 - accurac
y: 0.8673 - val_loss: 0.3908 - val_accuracy: 0.8622
Epoch 10/200
3563/3563 [==============================] - 728s 204ms/step - loss: 0.3775 - accurac
y: 0.8688 - val_loss: 0.3876 - val_accuracy: 0.8617
Epoch 11/200
3563/3563 [==============================] - 691s 194ms/step - loss: 0.3740 - accurac
y: 0.8706 - val_loss: 0.3910 - val_accuracy: 0.8632
Epoch 12/200
3563/3563 [==============================] - 685s 192ms/step - loss: 0.3704 - accurac
y: 0.8708 - val_loss: 0.3920 - val_accuracy: 0.8645
Epoch 13/200
3563/3563 [==============================] - 688s 193ms/step - loss: 0.3669 - accurac
y: 0.8730 - val_loss: 0.3939 - val_accuracy: 0.8623
Epoch 14/200
```

```
3563/3563 [==============================] - 689s 193ms/step - loss: 0.3642 - accurac
y: 0.8741 - val_loss: 0.3947 - val_accuracy: 0.8642
Epoch 15/200
3563/3563 [==============================] - 681s 191ms/step - loss: 0.3610 - accurac
y: 0.8750 - val_loss: 0.3901 - val_accuracy: 0.8653
Epoch 16/200
3563/3563 [==============================] - 682s 191ms/step - loss: 0.3577 - accurac
y: 0.8760 - val_loss: 0.3936 - val_accuracy: 0.8645
Epoch 17/200
3563/3563 [==============================] - 676s 190ms/step - loss: 0.3536 - accurac
y: 0.8779 - val_loss: 0.3956 - val_accuracy: 0.8653
Epoch 18/200
3563/3563 [==============================] - 687s 193ms/step - loss: 0.3518 - accurac
y: 0.8786 - val_loss: 0.3972 - val_accuracy: 0.8635
The runtime to fit this model was: 3:31:51.300028.
238/238 [==============================] - 19s 74ms/step - loss: 0.4018 - accuracy:
0.8587
Test acc: 0.859
```

2.2) Evaluate Model Performance

In [30]:

```
history_dict = history.history
history_dict.keys()
```

Out[30]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [31]:

```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

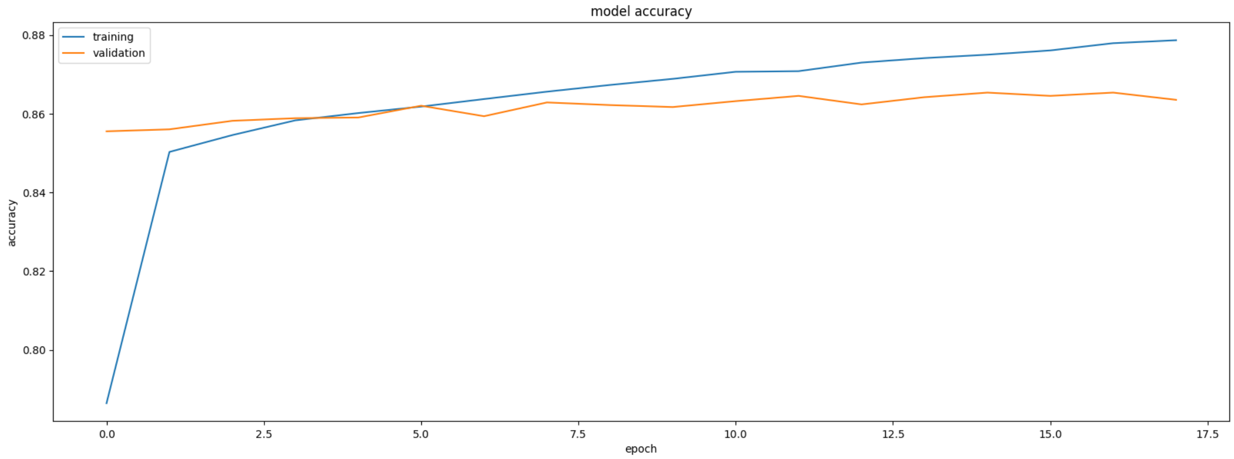
Out[31]:

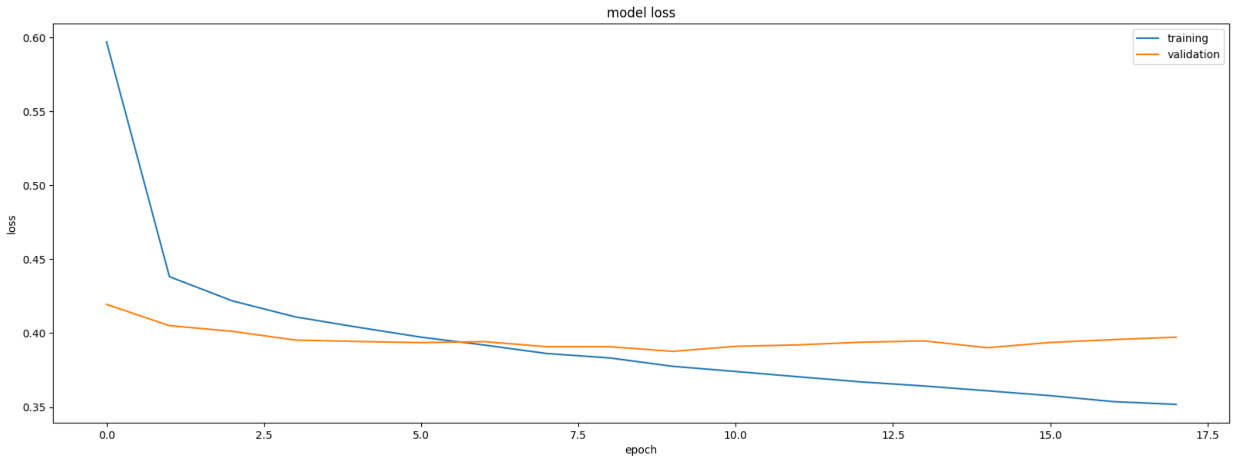
	loss	accuracy	val_loss	val_accuracy
13	0.364	0.874	0.395	0.864
14	0.361	0.875	0.390	0.865
15	0.358	0.876	0.394	0.864
16	0.354	0.878	0.396	0.865
17	0.352	0.879	0.397	0.863

In [32]:

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-6-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





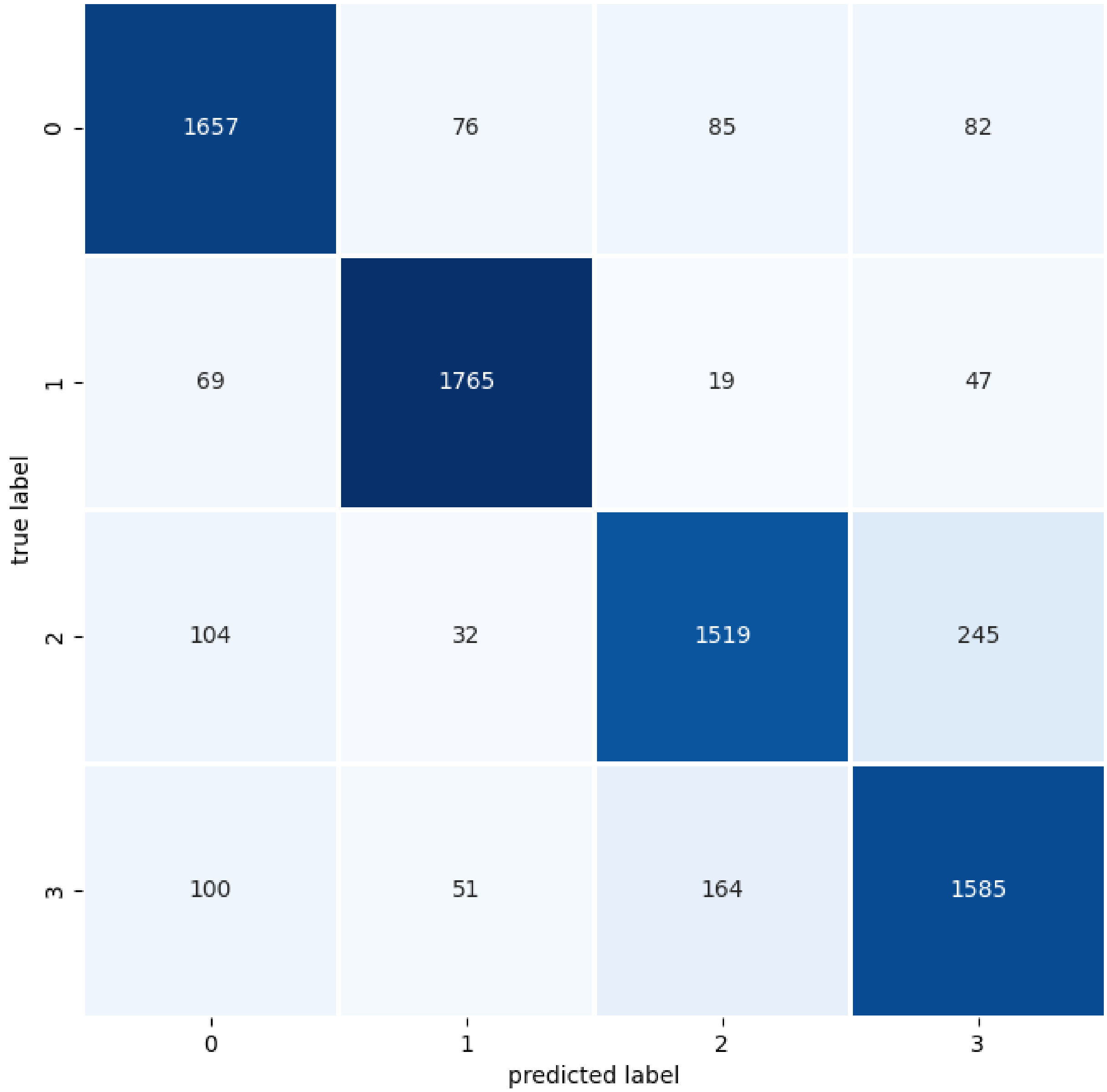
In [33]:
```
y_test = np.concatenate([y for x, y in int_test_ds], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds), axis=-1)
```

```
238/238 [==============================] - 29s 117ms/step
```

In [34]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.87      0.87      1900
           1       0.92      0.93      0.92      1900
           2       0.85      0.80      0.82      1900
           3       0.81      0.83      0.82      1900

    accuracy                           0.86      7600
   macro avg       0.86      0.86      0.86      7600
weighted avg       0.86      0.86      0.86      7600

Accuracy Score: 0.8586842105263158
Root Mean Square Error: 0.6679702167958657
```

In [35]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [ ]:
```
model = tf.keras.models.load_model("Model_One")
print(f"Training accuracy: {model.evaluate(x_train_norm, y_train_split)[1]:.3f}")
print(f"Validation accuracy: {model.evaluate(x_valid_norm, y_valid_split)[1]:.3f}")
print(f"Test accuracy: {model.evaluate(x_test_norm, y_test)[1]:.3f}")
```

In [36]:
```
model.evaluate(int_train_ds)
```

```
3563/3563 [==============================] - 283s 79ms/step - loss: 0.3454 - accurac
y: 0.8756
```

Out[36]:
```
[0.3453598916530609, 0.8756228089332581]
```

In [37]:
```
train_evaluation = model.evaluate(int_train_ds)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 283s 80ms/step - loss: 0.3454 - accurac
y: 0.8756
Training accuracy: 0.876
Training loss: 0.345
```

In [39]:
```
validation_evaluation = model.evaluate(int_val_ds)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
188/188 [==============================] - 15s 77ms/step - loss: 0.3876 - accuracy:
0.8617
Validation accuracy: 0.862
Validation loss: 0.388
238/238 [==============================] - 21s 89ms/step - loss: 0.4018 - accuracy:
0.8587
Testing accuracy: 0.859
Testing loss: 0.402
```

3) Model 2 - 1-Dimensional Convolutional Neural Network

3.1) Build The Model

In [ ]:
```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.Conv1D(filters=32, kernel_size=3, activation='relu')(embedded)
x = layers.Dropout(0.5)(x)
x = layers.MaxPooling1D(pool_size=2)(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(256, activation='relu')(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Two",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]



start_time = datetime.datetime.now()

history=model.fit(int_train_ds, validation_data=int_val_ds, epochs=200, callbacks=call

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")

model = keras.models.load_model("Model_Two")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 tf.one_hot (TFOpLambda)     (None, None, 1000)        0         
                                                                 
 conv1d (Conv1D)             (None, None, 32)          96032     
                                                                 
 dropout (Dropout)           (None, None, 32)          0         
                                                                 
 max_pooling1d (MaxPooling1  (None, None, 32)          0         
 D)                                                              
                                                                 
 global_max_pooling1d (Glob  (None, 32)                0         
 alMaxPooling1D)                                                 
                                                                 
 dense (Dense)               (None, 256)               8448      
                                                                 
 dense_1 (Dense)             (None, 4)                 1028      
                                                                 
=================================================================
Total params: 105508 (412.14 KB)
Trainable params: 105508 (412.14 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 190s 53ms/step - loss: 0.5573 - accurac
y: 0.7978 - val_loss: 0.4656 - val_accuracy: 0.8495
Epoch 2/200
3563/3563 [==============================] - 187s 52ms/step - loss: 0.4539 - accurac
y: 0.8409 - val_loss: 0.4616 - val_accuracy: 0.8483
Epoch 3/200
3563/3563 [==============================] - 193s 54ms/step - loss: 0.4453 - accurac
y: 0.8433 - val_loss: 0.4637 - val_accuracy: 0.8492
Epoch 4/200
3563/3563 [==============================] - 198s 55ms/step - loss: 0.4436 - accurac
y: 0.8459 - val_loss: 0.4660 - val_accuracy: 0.8477
The runtime to fit this model was: 0:13:11.366593.
238/238 [==============================] - 6s 26ms/step - loss: 0.4679 - accuracy: 0.
8455
Test acc: 0.846
```

3.2) Evaluate Model Performance

In [ ]:

```
history_dict = history.history
history_dict.keys()
```

Out[ ]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:

```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

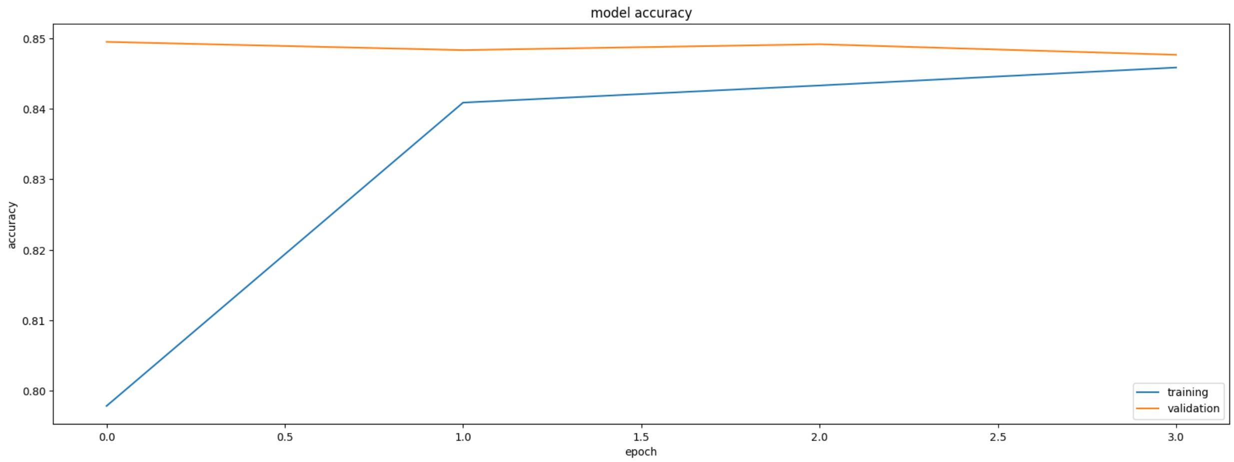
Out[ ]:

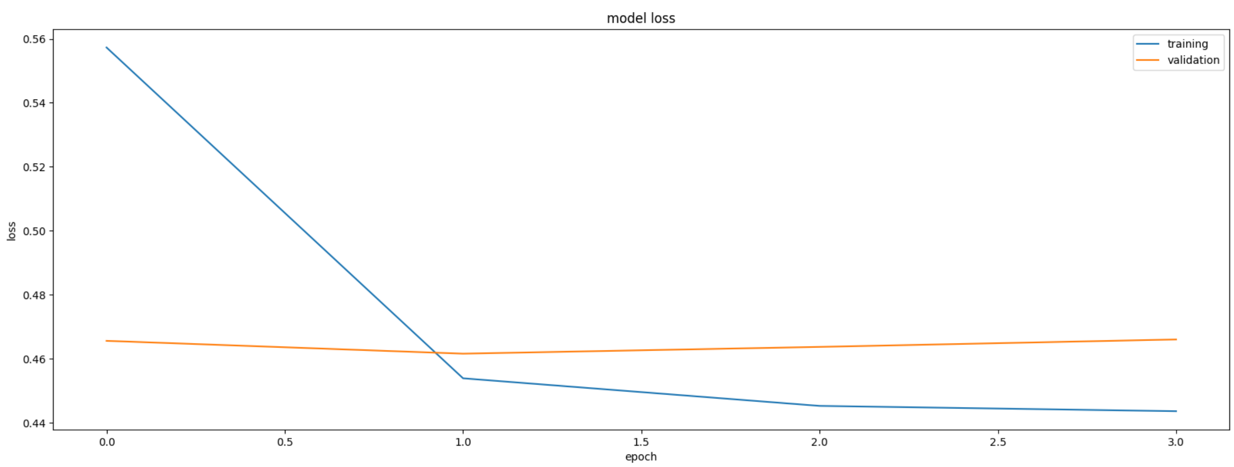
	loss	accuracy	val_loss	val_accuracy
0	0.557	0.798	0.466	0.850
1	0.454	0.841	0.462	0.848
2	0.445	0.843	0.464	0.849
3	0.444	0.846	0.466	0.848

In [ ]:

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-6-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





In [ ]:

```
y_test = np.concatenate([y for x, y in int_test_ds], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds), axis=-1)
```

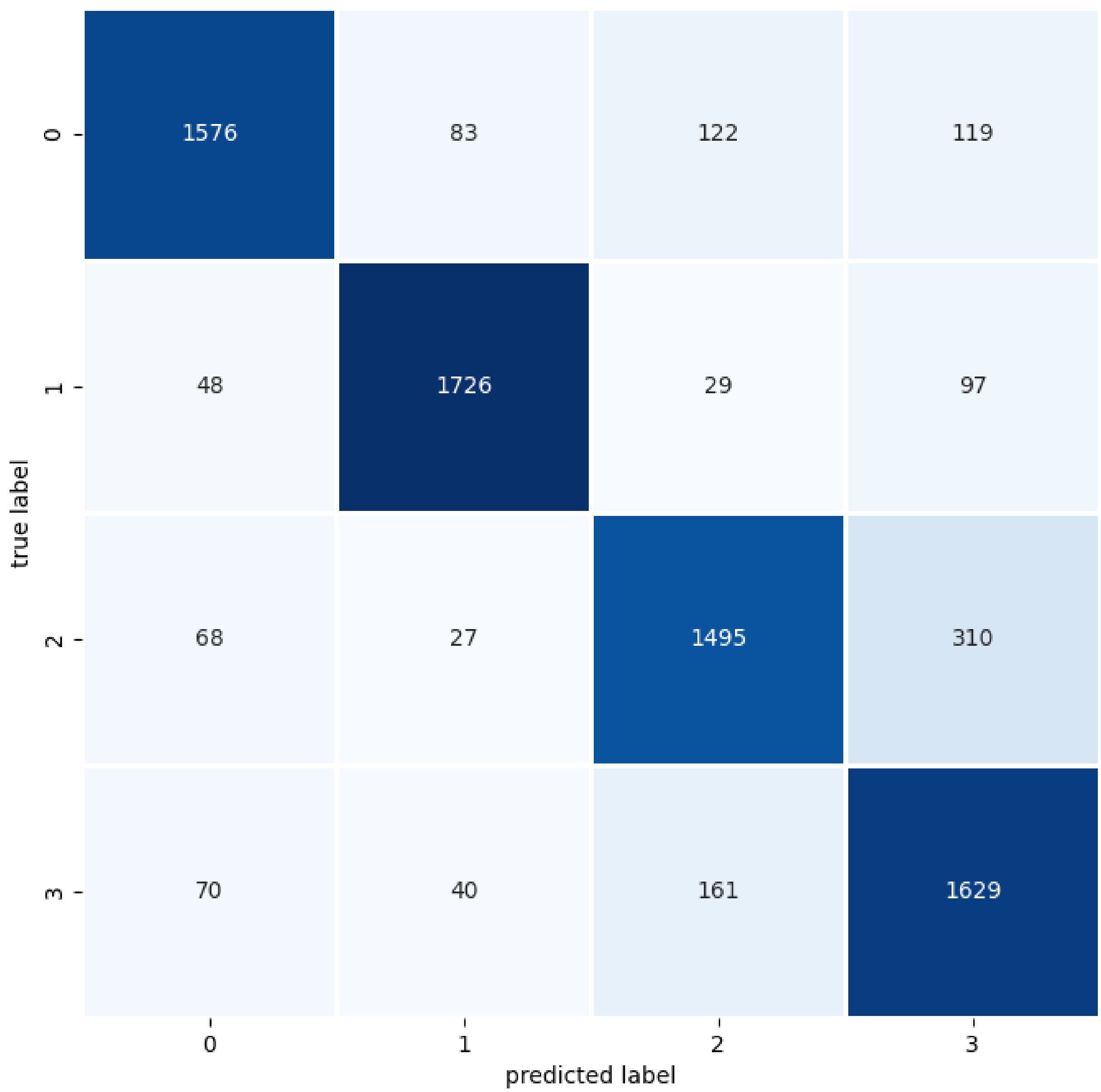
```
238/238 [==============================] - 5s 20ms/step
```

In [ ]:

```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.83      0.86      1900
           1       0.92      0.91      0.91      1900
           2       0.83      0.79      0.81      1900
           3       0.76      0.86      0.80      1900

    accuracy                           0.85      7600
   macro avg       0.85      0.85      0.85      7600
weighted avg       0.85      0.85      0.85      7600

Accuracy Score: 0.8455263157894737
Root Mean Square Error: 0.6946221994724903
```

In [ ]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [40]:
```
model = tf.keras.models.load_model("Model_Two")

train_evaluation = model.evaluate(int_train_ds)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
```

```
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 87s 24ms/step - loss: 0.4552 - accuracy:
0.8486
Training accuracy: 0.849
Training loss: 0.455
188/188 [==============================] - 4s 19ms/step - loss: 0.4616 - accuracy: 0.
8483
Validation accuracy: 0.848
Validation loss: 0.462
238/238 [==============================] - 6s 27ms/step - loss: 0.4679 - accuracy: 0.
8455
Testing accuracy: 0.846
Testing loss: 0.468
```

4) Model 3 - Gated Recurrent Unit (GRU) Model

4.1) Execute New Data Wrangling

In [28]:

```
max_length = 40
max_tokens = 1000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
    standardize=custom_stopwords
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds_two = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds_two = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds_two = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

4.2) Build the Model

In [51]:

```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
#embedded = tf.one_hot(inputs, depth=max_tokens)
embedded = layers.Embedding(input_dim=max_tokens
                            ,output_dim=256
                            ,mask_zero=True)(inputs)
x = layers.GRU(32)(embedded)
```

```
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Four",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_two, validation_data=int_val_ds_two, epochs=200, callba

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Four")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 256)         256000

 gru (GRU)                   (None, 32)                27840

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 4)                 132

=================================================================
Total params: 283972 (1.08 MB)
Trainable params: 283972 (1.08 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 170s 47ms/step - loss: 0.5372 - accurac
y: 0.8087 - val_loss: 0.4095 - val_accuracy: 0.8598
Epoch 2/200
3563/3563 [==============================] - 221s 62ms/step - loss: 0.4327 - accurac
y: 0.8515 - val_loss: 0.3992 - val_accuracy: 0.8608
Epoch 3/200
3563/3563 [==============================] - 186s 52ms/step - loss: 0.4191 - accurac
y: 0.8564 - val_loss: 0.3919 - val_accuracy: 0.8600
Epoch 4/200
3563/3563 [==============================] - 145s 41ms/step - loss: 0.4056 - accurac
y: 0.8596 - val_loss: 0.3869 - val_accuracy: 0.8630
Epoch 5/200
3563/3563 [==============================] - 135s 38ms/step - loss: 0.3959 - accurac
y: 0.8633 - val_loss: 0.3875 - val_accuracy: 0.8632
Epoch 6/200
3563/3563 [==============================] - 137s 39ms/step - loss: 0.3882 - accurac
y: 0.8655 - val_loss: 0.3902 - val_accuracy: 0.8638
Epoch 7/200
3563/3563 [==============================] - 138s 39ms/step - loss: 0.3801 - accurac
y: 0.8697 - val_loss: 0.3900 - val_accuracy: 0.8638
Epoch 8/200
3563/3563 [==============================] - 135s 38ms/step - loss: 0.3746 - accurac
y: 0.8709 - val_loss: 0.3908 - val_accuracy: 0.8628
Epoch 9/200
3563/3563 [==============================] - 138s 39ms/step - loss: 0.3682 - accurac
y: 0.8738 - val_loss: 0.3915 - val_accuracy: 0.8595
The runtime to fit this model was: 0:25:25.034429.
```

4.2) Evaluate Model Performance

In [53]:

```python
history_dict = history.history
history_dict.keys()
```

Out[53]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [54]:

```python
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
```

```
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

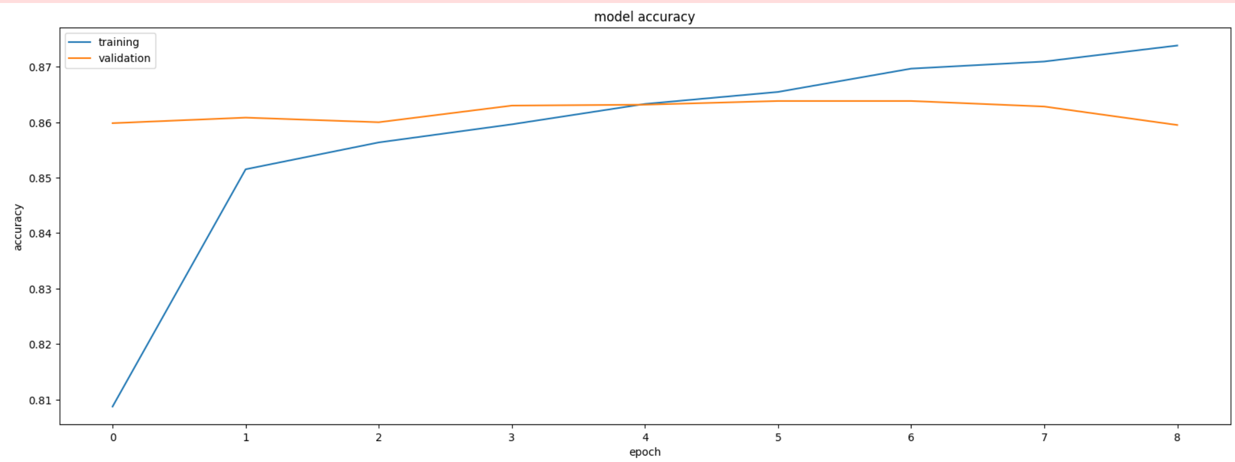
Out[54]:

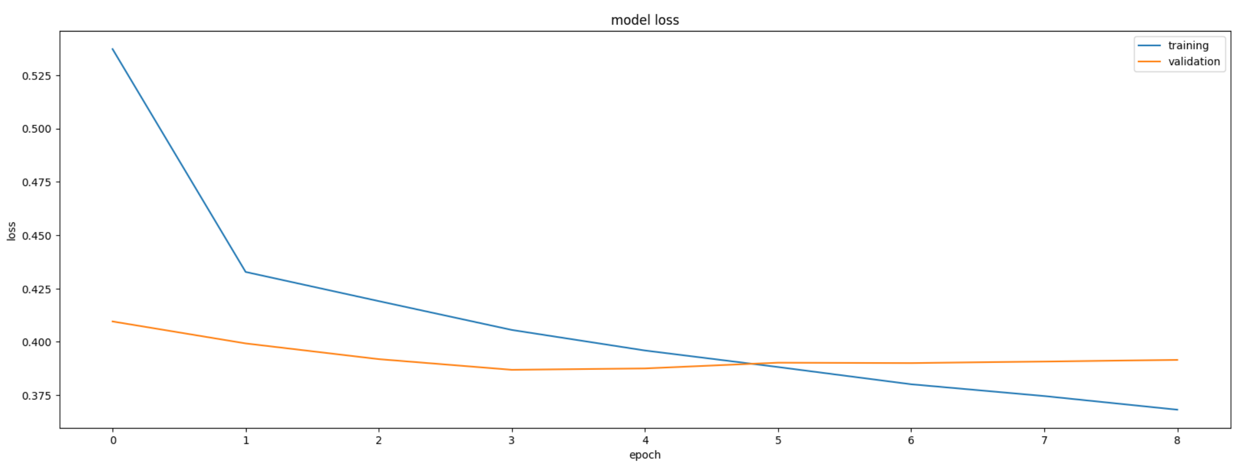
	loss	accuracy	val_loss	val_accuracy
4	0.396	0.863	0.388	0.863
5	0.388	0.865	0.390	0.864
6	0.380	0.870	0.390	0.864
7	0.375	0.871	0.391	0.863
8	0.368	0.874	0.392	0.859

In [55]:

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-6-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





In [56]:

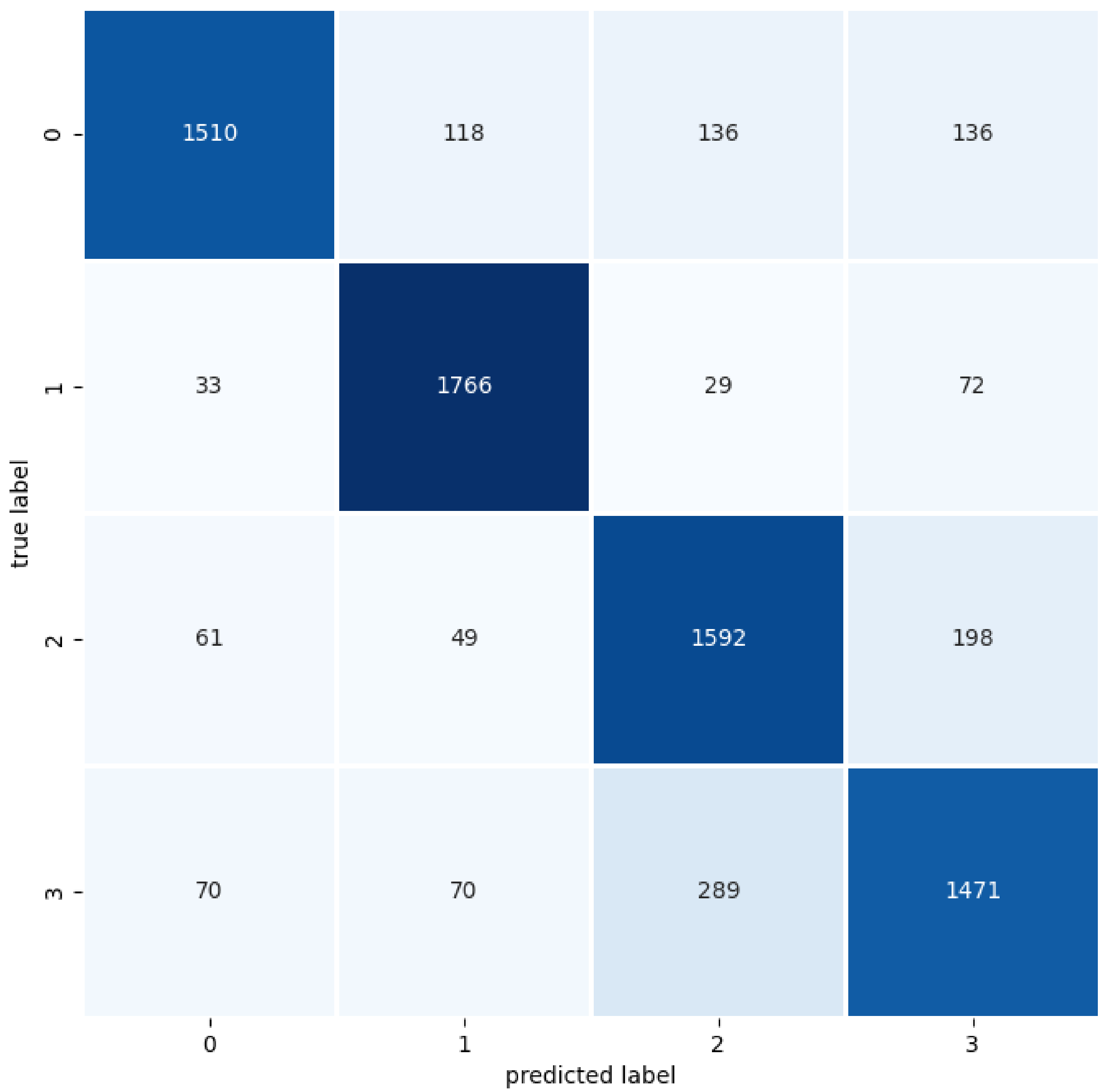
```
y_test = np.concatenate([y for x, y in int_test_ds_two], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_two), axis=-1)
```

```
238/238 [==============================] - 6s 12ms/step
```

In [57]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.90      0.79      0.84      1900
           1       0.88      0.93      0.90      1900
           2       0.78      0.84      0.81      1900
           3       0.78      0.77      0.78      1900

    accuracy                           0.83      7600
   macro avg       0.84      0.83      0.83      7600
weighted avg       0.84      0.83      0.83      7600

Accuracy Score: 0.834078947368421
Root Mean Square Error: 0.7187342675623731
```

In [58]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [59]:
```
train_evaluation = model.evaluate(int_train_ds_two)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_two)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_two)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 54s 14ms/step - loss: 0.4377 - accuracy:
0.8448
Training accuracy: 0.845
Training loss: 0.438
188/188 [==============================] - 2s 12ms/step - loss: 0.4444 - accuracy: 0.
8435
Validation accuracy: 0.844
Validation loss: 0.444
238/238 [==============================] - 4s 17ms/step - loss: 0.4635 - accuracy: 0.
8341
Testing accuracy: 0.834
Testing loss: 0.463
```

5) Model 4 - Recurrent Neural Network

5.1) Build The Model

In [60]:
```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
#embedded = tf.one_hot(inputs, depth=max_tokens)
embedded = layers.Embedding(input_dim=max_tokens
                            ,output_dim=256
                            ,mask_zero=True)(inputs)
x = layers.SimpleRNN(128)(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Three",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_two, validation_data=int_val_ds_two, epochs=200, callba

end_time = datetime.datetime.now()
runtime = end_time - start_time
```

```
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Three")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 embedding (Embedding)       (None, None, 256)         256000    
                                                                 
 simple_rnn (SimpleRNN)      (None, 128)               49280     
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense (Dense)               (None, 4)                 516       
                                                                 
=================================================================
Total params: 305796 (1.17 MB)
Trainable params: 305796 (1.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 137s 37ms/step - loss: 0.5750 - accurac
y: 0.7971 - val_loss: 0.4710 - val_accuracy: 0.8402
Epoch 2/200
3563/3563 [==============================] - 124s 35ms/step - loss: 0.4934 - accurac
y: 0.8336 - val_loss: 0.4683 - val_accuracy: 0.8435
Epoch 3/200
3563/3563 [==============================] - 122s 34ms/step - loss: 0.4792 - accurac
y: 0.8394 - val_loss: 0.4609 - val_accuracy: 0.8452
Epoch 4/200
3563/3563 [==============================] - 128s 36ms/step - loss: 0.4685 - accurac
y: 0.8432 - val_loss: 0.4481 - val_accuracy: 0.8487
Epoch 5/200
3563/3563 [==============================] - 124s 35ms/step - loss: 0.4607 - accurac
y: 0.8446 - val_loss: 0.4581 - val_accuracy: 0.8467
Epoch 6/200
3563/3563 [==============================] - 143s 40ms/step - loss: 0.4534 - accurac
y: 0.8466 - val_loss: 0.4452 - val_accuracy: 0.8493
Epoch 7/200
3563/3563 [==============================] - 126s 35ms/step - loss: 0.4458 - accurac
y: 0.8498 - val_loss: 0.4524 - val_accuracy: 0.8493
Epoch 8/200
3563/3563 [==============================] - 145s 41ms/step - loss: 0.4431 - accurac
y: 0.8496 - val_loss: 0.4512 - val_accuracy: 0.8492
Epoch 9/200
3563/3563 [==============================] - 125s 35ms/step - loss: 0.4357 - accurac
y: 0.8529 - val_loss: 0.4398 - val_accuracy: 0.8502
Epoch 10/200
3563/3563 [==============================] - 119s 33ms/step - loss: 0.4363 - accurac
y: 0.8525 - val_loss: 0.4573 - val_accuracy: 0.8462
Epoch 11/200
3563/3563 [==============================] - 121s 34ms/step - loss: 0.4341 - accurac
y: 0.8538 - val_loss: 0.4610 - val_accuracy: 0.8443
Epoch 12/200
3563/3563 [==============================] - 120s 34ms/step - loss: 0.4334 - accurac
y: 0.8539 - val_loss: 0.4487 - val_accuracy: 0.8435
The runtime to fit this model was: 0:27:40.145948.
```

5.2) Evaluate Model Performance

In [61]:
```
history_dict = history.history
history_dict.keys()
```

Out[61]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

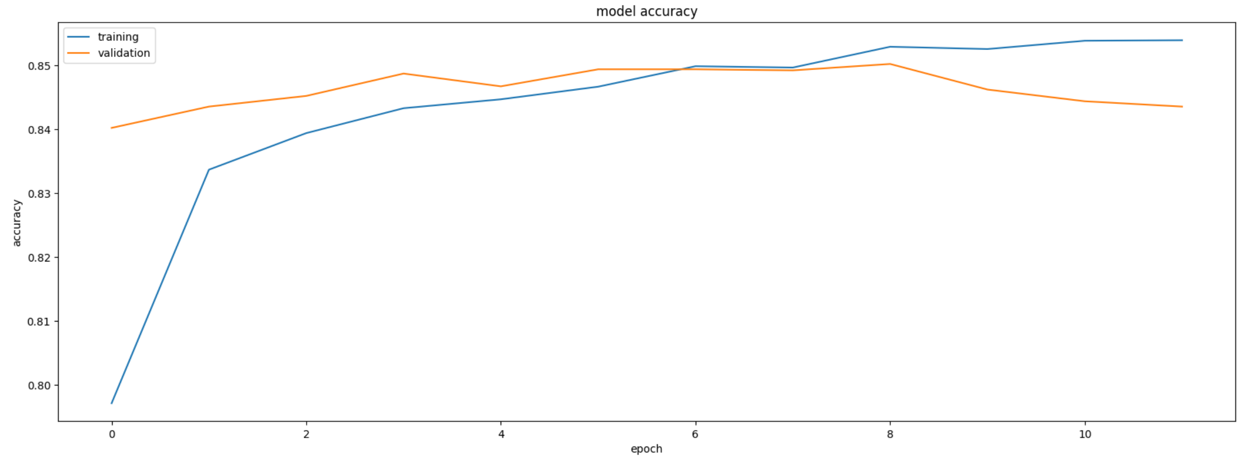
In [62]:
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

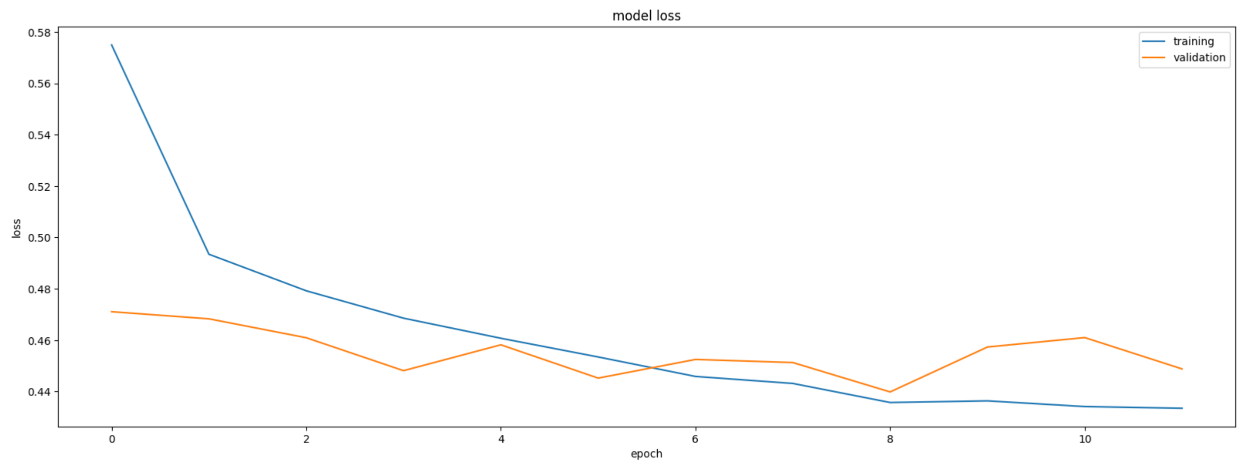
Out[62]:

	loss	accuracy	val_loss	val_accuracy
7	0.443	0.850	0.451	0.849
8	0.436	0.853	0.440	0.850
9	0.436	0.853	0.457	0.846
10	0.434	0.854	0.461	0.844
11	0.433	0.854	0.449	0.844

In [63]:
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-6-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





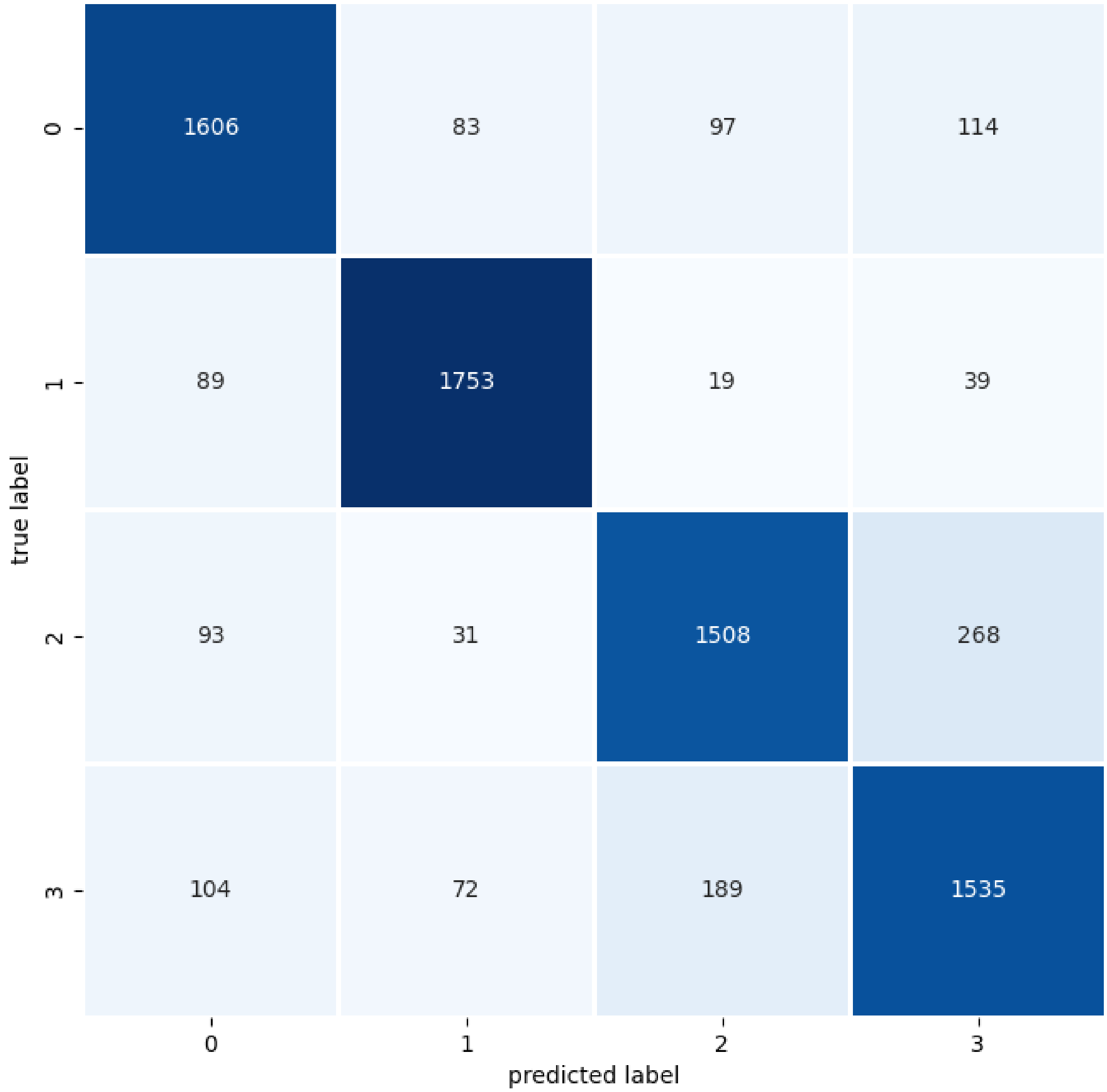
In [64]:
```
y_test = np.concatenate([y for x, y in int_test_ds_two], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_two), axis=-1)
```

```
238/238 [==============================] - 2s 9ms/step
```

In [65]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.85      0.85      1900
           1       0.90      0.92      0.91      1900
           2       0.83      0.79      0.81      1900
           3       0.78      0.81      0.80      1900

    accuracy                           0.84      7600
   macro avg       0.84      0.84      0.84      7600
weighted avg       0.84      0.84      0.84      7600

Accuracy Score: 0.8423684210526315
Root Mean Square Error: 0.711281275327545
```

In [66]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [67]:
```
train_evaluation = model.evaluate(int_train_ds_two)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_two)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_two)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 36s 10ms/step - loss: 0.3958 - accuracy:
0.8650
Training accuracy: 0.865
Training loss: 0.396
188/188 [==============================] - 2s 9ms/step - loss: 0.4398 - accuracy: 0.8
502
Validation accuracy: 0.850
Validation loss: 0.440
238/238 [==============================] - 2s 9ms/step - loss: 0.4595 - accuracy: 0.8
424
Testing accuracy: 0.842
Testing loss: 0.459
```

6) Model 5 - Dense Artificial Neural Network

6.1) Build The Model

In [81]:

```
from tensorflow.keras.layers import Flatten, GlobalMaxPooling1D

k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
#embedded = tf.one_hot(inputs, depth=max_tokens)
embedded = layers.Embedding(input_dim=max_tokens
                            ,output_dim=256
                            ,mask_zero=True)(inputs)
#x = Flatten()(embedded)
x = layers.Dense(128, activation = "relu")(embedded)
x = layers.Dropout(0.5)(x)
x = GlobalMaxPooling1D()(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

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

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Five",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_two, validation_data=int_val_ds_two, epochs=200, callba

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Five")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 256)         256000

 dense (Dense)               (None, None, 128)         32896

 dropout (Dropout)           (None, None, 128)         0

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 289412 (1.10 MB)
Trainable params: 289412 (1.10 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 59s 16ms/step - loss: 0.4670 - accuracy:
0.8357 - val_loss: 0.4617 - val_accuracy: 0.8452
Epoch 2/200
3563/3563 [==============================] - 55s 15ms/step - loss: 0.4253 - accuracy:
0.8493 - val_loss: 0.4487 - val_accuracy: 0.8467
Epoch 3/200
3563/3563 [==============================] - 52s 15ms/step - loss: 0.4214 - accuracy:
0.8500 - val_loss: 0.4599 - val_accuracy: 0.8438
Epoch 4/200
3563/3563 [==============================] - 55s 15ms/step - loss: 0.4193 - accuracy:
0.8501 - val_loss: 0.4515 - val_accuracy: 0.8460
Epoch 5/200
3563/3563 [==============================] - 131s 37ms/step - loss: 0.4176 - accurac
y: 0.8513 - val_loss: 0.4476 - val_accuracy: 0.8468
Epoch 6/200
3563/3563 [==============================] - 56s 16ms/step - loss: 0.4164 - accuracy:
0.8516 - val_loss: 0.4578 - val_accuracy: 0.8445
Epoch 7/200
3563/3563 [==============================] - 56s 16ms/step - loss: 0.4146 - accuracy:
0.8523 - val_loss: 0.4475 - val_accuracy: 0.8467
Epoch 8/200
3563/3563 [==============================] - 56s 16ms/step - loss: 0.4154 - accuracy:
0.8521 - val_loss: 0.4527 - val_accuracy: 0.8455
The runtime to fit this model was: 0:11:04.390397.
```

6.2) Evaluate Model Performance

In [82]:

```
history_dict = history.history
history_dict.keys()
```

Out[82]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [83]:

```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
```

```
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

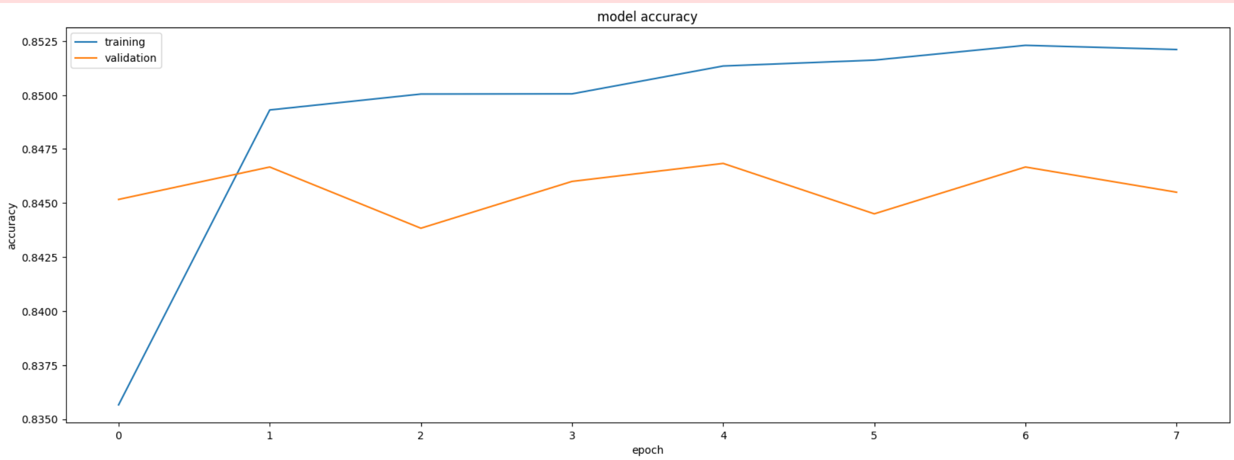
Out[83]:

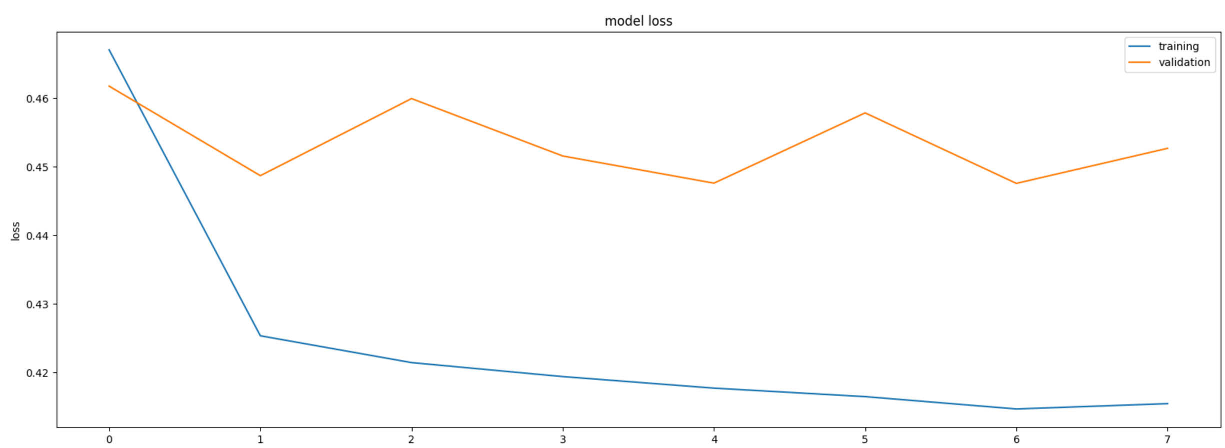
	loss	accuracy	val_loss	val_accuracy
3	0.419	0.850	0.452	0.846
4	0.418	0.851	0.448	0.847
5	0.416	0.852	0.458	0.845
6	0.415	0.852	0.448	0.847
7	0.415	0.852	0.453	0.845

In [84]:

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-6-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





In [85]:

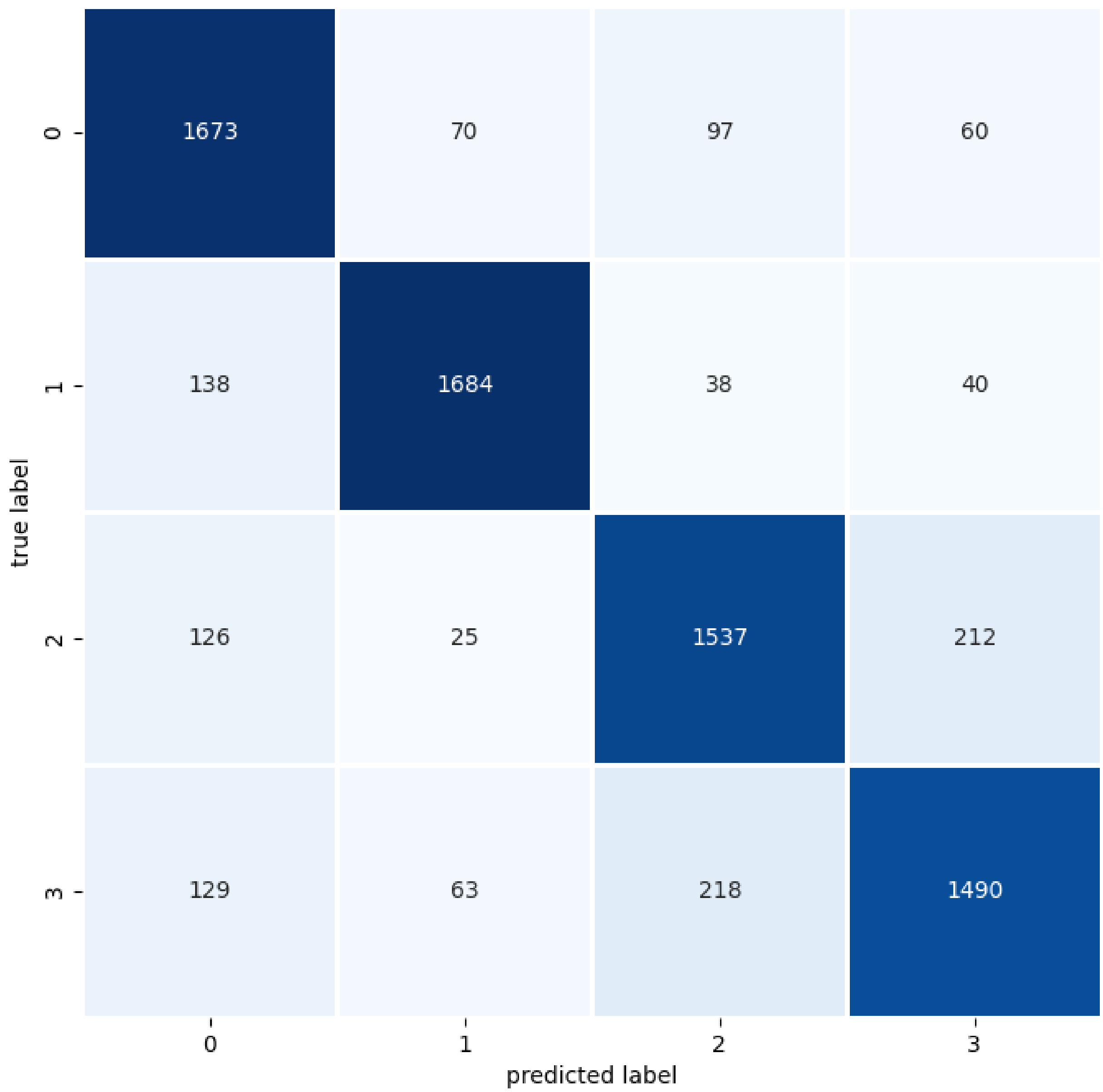
```
y_test = np.concatenate([y for x, y in int_test_ds_two], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_two), axis=-1)
```

```
238/238 [==============================] - 1s 5ms/step
```

In [86]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.88      0.84      1900
           1       0.91      0.89      0.90      1900
           2       0.81      0.81      0.81      1900
           3       0.83      0.78      0.80      1900

    accuracy                           0.84      7600
   macro avg       0.84      0.84      0.84      7600
weighted avg       0.84      0.84      0.84      7600

Accuracy Score: 0.84
Root Mean Square Error: 0.6983062214726204
```

In [87]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [88]:
```
train_evaluation = model.evaluate(int_train_ds_two)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
```

```
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_two)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_two)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 22s 6ms/step - loss: 0.4342 - accuracy:
0.8515
Training accuracy: 0.851
Training loss: 0.434
188/188 [==============================] - 1s 7ms/step - loss: 0.4475 - accuracy: 0.8
467
Validation accuracy: 0.847
Validation loss: 0.448
238/238 [==============================] - 2s 8ms/step - loss: 0.4568 - accuracy: 0.8
400
Testing accuracy: 0.840
Testing loss: 0.457
```

7) Model 6 - Uni-Directional Long Short Term Memory Model

7.1) Build The Model

In [89]:

```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.LSTM(32)(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Six",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]



start_time = datetime.datetime.now()

history=model.fit(int_train_ds_two, validation_data=int_val_ds_two, epochs=200, callba

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")
```

```
model = keras.models.load_model("Model_Six")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 tf.one_hot (TFOpLambda)     (None, None, 1000)        0         
                                                                 
 lstm (LSTM)                 (None, 32)                132224    
                                                                 
 dropout (Dropout)           (None, 32)                0         
                                                                 
 dense (Dense)               (None, 4)                 132       
                                                                 
=================================================================
Total params: 132356 (517.02 KB)
Trainable params: 132356 (517.02 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 216s 60ms/step - loss: 0.6958 - accurac
y: 0.7175 - val_loss: 0.4589 - val_accuracy: 0.8417
Epoch 2/200
3563/3563 [==============================] - 210s 59ms/step - loss: 0.4672 - accurac
y: 0.8430 - val_loss: 0.4229 - val_accuracy: 0.8555
Epoch 3/200
3563/3563 [==============================] - 200s 56ms/step - loss: 0.4493 - accurac
y: 0.8496 - val_loss: 0.4219 - val_accuracy: 0.8565
Epoch 4/200
3563/3563 [==============================] - 213s 60ms/step - loss: 0.4366 - accurac
y: 0.8520 - val_loss: 0.4135 - val_accuracy: 0.8578
Epoch 5/200
3563/3563 [==============================] - 208s 58ms/step - loss: 0.4295 - accurac
y: 0.8537 - val_loss: 0.4138 - val_accuracy: 0.8595
Epoch 6/200
3563/3563 [==============================] - 203s 57ms/step - loss: 0.4238 - accurac
y: 0.8565 - val_loss: 0.4142 - val_accuracy: 0.8595
Epoch 7/200
3563/3563 [==============================] - 209s 59ms/step - loss: 0.4181 - accurac
y: 0.8581 - val_loss: 0.4110 - val_accuracy: 0.8607
Epoch 8/200
3563/3563 [==============================] - 196s 55ms/step - loss: 0.4110 - accurac
y: 0.8589 - val_loss: 0.4111 - val_accuracy: 0.8635
Epoch 9/200
3563/3563 [==============================] - 191s 54ms/step - loss: 0.4066 - accurac
y: 0.8605 - val_loss: 0.4085 - val_accuracy: 0.8648
Epoch 10/200
3563/3563 [==============================] - 189s 53ms/step - loss: 0.4035 - accurac
y: 0.8617 - val_loss: 0.4100 - val_accuracy: 0.8617
Epoch 11/200
3563/3563 [==============================] - 187s 52ms/step - loss: 0.3978 - accurac
y: 0.8634 - val_loss: 0.4069 - val_accuracy: 0.8603
Epoch 12/200
3563/3563 [==============================] - 195s 55ms/step - loss: 0.3963 - accurac
y: 0.8649 - val_loss: 0.4076 - val_accuracy: 0.8653
Epoch 13/200
3563/3563 [==============================] - 197s 55ms/step - loss: 0.3915 - accurac
y: 0.8654 - val_loss: 0.4003 - val_accuracy: 0.8652
Epoch 14/200
3563/3563 [==============================] - 194s 54ms/step - loss: 0.3871 - accurac
```

```
y: 0.8674 - val_loss: 0.4025 - val_accuracy: 0.8652
Epoch 15/200
3563/3563 [==============================] - 196s 55ms/step - loss: 0.3849 - accurac
y: 0.8690 - val_loss: 0.4004 - val_accuracy: 0.8657
Epoch 16/200
3563/3563 [==============================] - 208s 58ms/step - loss: 0.3809 - accurac
y: 0.8704 - val_loss: 0.3961 - val_accuracy: 0.8688
Epoch 17/200
3563/3563 [==============================] - 190s 53ms/step - loss: 0.3766 - accurac
y: 0.8711 - val_loss: 0.3973 - val_accuracy: 0.8685
Epoch 18/200
3563/3563 [==============================] - 191s 54ms/step - loss: 0.3743 - accurac
y: 0.8720 - val_loss: 0.4006 - val_accuracy: 0.8682
Epoch 19/200
3563/3563 [==============================] - 191s 54ms/step - loss: 0.3728 - accurac
y: 0.8738 - val_loss: 0.4007 - val_accuracy: 0.8638
The runtime to fit this model was: 1:06:12.284051.
```

7.2) Evaluate Model Performance

In [90]:

```
history_dict = history.history
history_dict.keys()
```

Out[90]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [91]:

```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

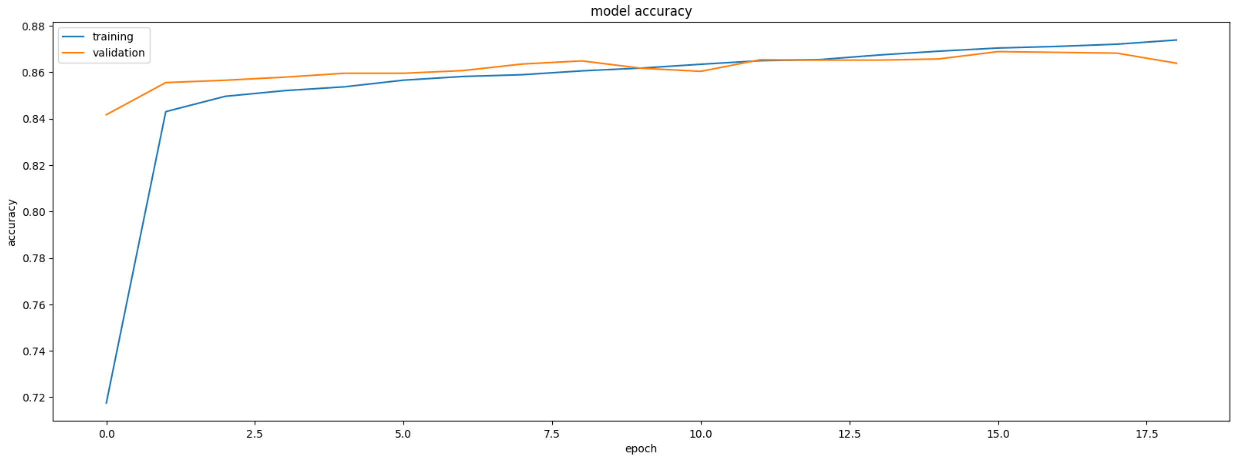
Out[91]:

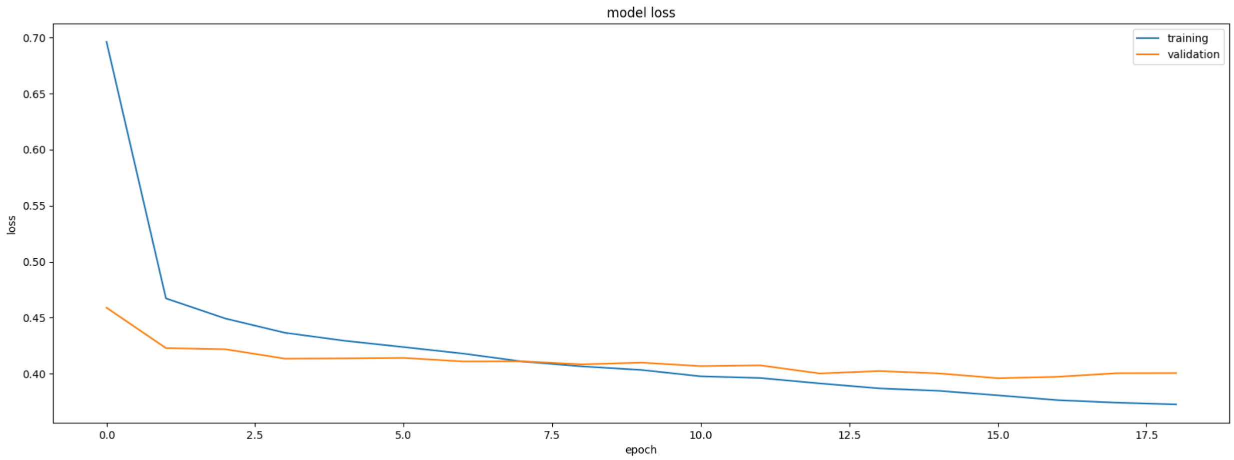
	loss	accuracy	val_loss	val_accuracy
14	0.385	0.869	0.400	0.866
15	0.381	0.870	0.396	0.869
16	0.377	0.871	0.397	0.868
17	0.374	0.872	0.401	0.868
18	0.373	0.874	0.401	0.864

In [92]:

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-6-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





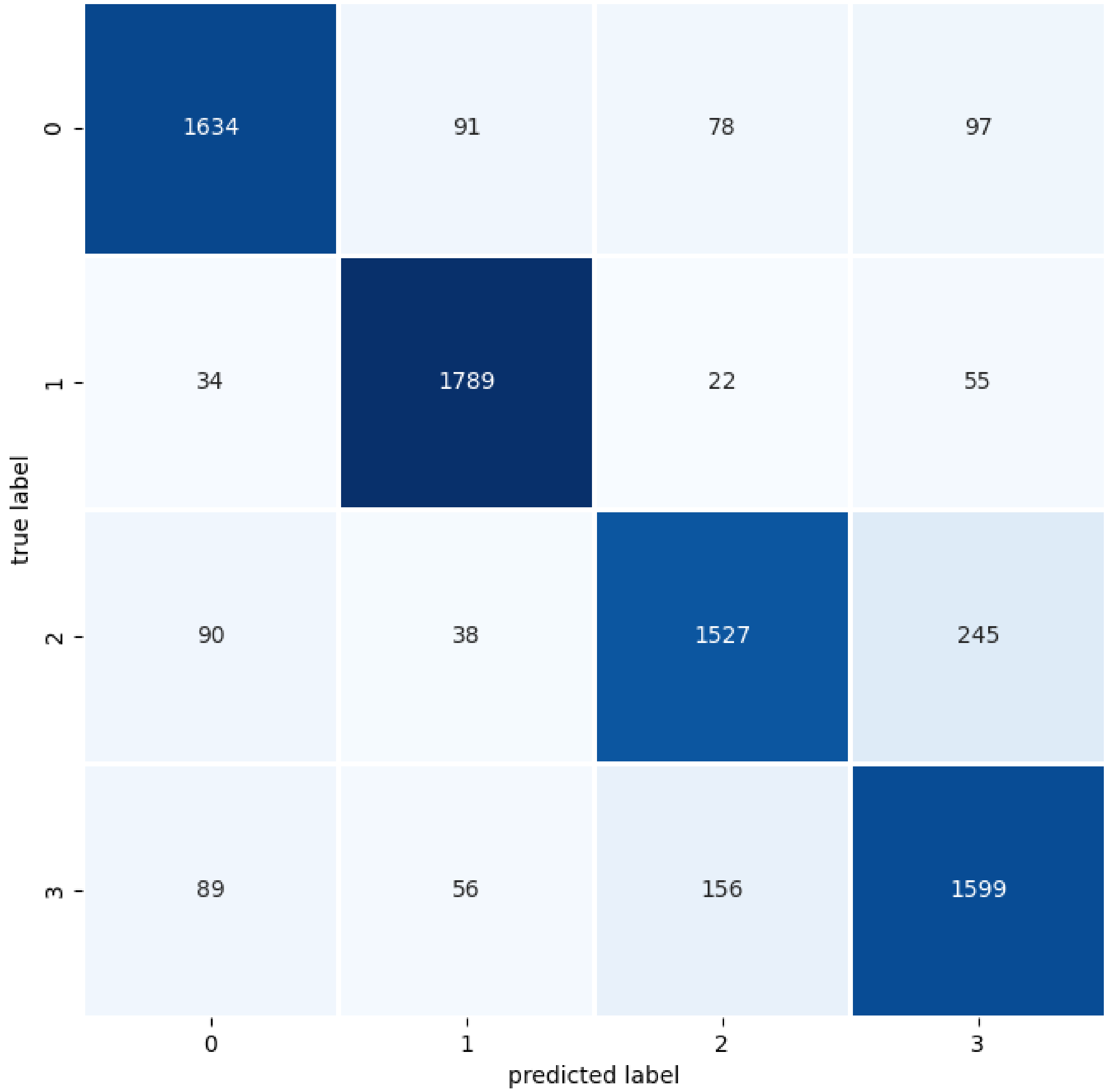
In [93]:
```
y_test = np.concatenate([y for x, y in int_test_ds_two], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_two), axis=-1)
```

```
238/238 [==============================] - 11s 35ms/step
```

In [94]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.86      0.87      1900
           1       0.91      0.94      0.92      1900
           2       0.86      0.80      0.83      1900
           3       0.80      0.84      0.82      1900

    accuracy                           0.86      7600
   macro avg       0.86      0.86      0.86      7600
weighted avg       0.86      0.86      0.86      7600

Accuracy Score: 0.8617105263157895
Root Mean Square Error: 0.6664912049800729
```

In [95]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [96]:
```
train_evaluation = model.evaluate(int_train_ds_two)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_two)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_two)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 85s 23ms/step - loss: 0.3539 - accuracy: 0.8764
Training accuracy: 0.876
Training loss: 0.354
188/188 [==============================] - 4s 19ms/step - loss: 0.3961 - accuracy: 0.8688
Validation accuracy: 0.869
Validation loss: 0.396
238/238 [==============================] - 6s 24ms/step - loss: 0.4105 - accuracy: 0.8617
Testing accuracy: 0.862
Testing loss: 0.410
```

8) Model 7 - Uni-Directional LSTM With Less Regularization

8.1) Build The Model

In [29]:

```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.LSTM(64)(embedded)
x = layers.Dropout(0.4)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Seven",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_two, validation_data=int_val_ds_two, epochs=200, callba

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Seven")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 tf.one_hot (TFOpLambda)     (None, None, 1000)        0

 lstm (LSTM)                 (None, 64)                272640

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 4)                 260

=================================================================
Total params: 272900 (1.04 MB)
Trainable params: 272900 (1.04 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 371s 103ms/step - loss: 0.6495 - accurac
y: 0.7427 - val_loss: 0.4508 - val_accuracy: 0.8502
Epoch 2/200
3563/3563 [==============================] - 356s 100ms/step - loss: 0.4460 - accurac
y: 0.8460 - val_loss: 0.4224 - val_accuracy: 0.8567
Epoch 3/200
3563/3563 [==============================] - 346s 97ms/step - loss: 0.4241 - accurac
y: 0.8518 - val_loss: 0.4164 - val_accuracy: 0.8580
Epoch 4/200
3563/3563 [==============================] - 313s 88ms/step - loss: 0.4090 - accurac
y: 0.8566 - val_loss: 0.4112 - val_accuracy: 0.8590
Epoch 5/200
3563/3563 [==============================] - 307s 86ms/step - loss: 0.3973 - accurac
y: 0.8603 - val_loss: 0.4062 - val_accuracy: 0.8570
Epoch 6/200
3563/3563 [==============================] - 308s 87ms/step - loss: 0.3896 - accurac
y: 0.8625 - val_loss: 0.3993 - val_accuracy: 0.8603
Epoch 7/200
3563/3563 [==============================] - 305s 86ms/step - loss: 0.3832 - accurac
y: 0.8650 - val_loss: 0.3986 - val_accuracy: 0.8602
Epoch 8/200
3563/3563 [==============================] - 297s 83ms/step - loss: 0.3755 - accurac
y: 0.8678 - val_loss: 0.3994 - val_accuracy: 0.8583
Epoch 9/200
3563/3563 [==============================] - 304s 85ms/step - loss: 0.3692 - accurac
y: 0.8704 - val_loss: 0.3965 - val_accuracy: 0.8607
Epoch 10/200
3563/3563 [==============================] - 453s 127ms/step - loss: 0.3626 - accurac
y: 0.8725 - val_loss: 0.3915 - val_accuracy: 0.8647
Epoch 11/200
3563/3563 [==============================] - 323s 91ms/step - loss: 0.3564 - accurac
y: 0.8749 - val_loss: 0.3942 - val_accuracy: 0.8632
Epoch 12/200
3563/3563 [==============================] - 333s 93ms/step - loss: 0.3512 - accurac
y: 0.8774 - val_loss: 0.3948 - val_accuracy: 0.8633
Epoch 13/200
3563/3563 [==============================] - 325s 91ms/step - loss: 0.3455 - accurac
y: 0.8795 - val_loss: 0.4020 - val_accuracy: 0.8625
The runtime to fit this model was: 1:16:33.501255.
```

8.2) Evaluate Model Performance

In [30]:
```
history_dict = history.history
history_dict.keys()
```

Out[30]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

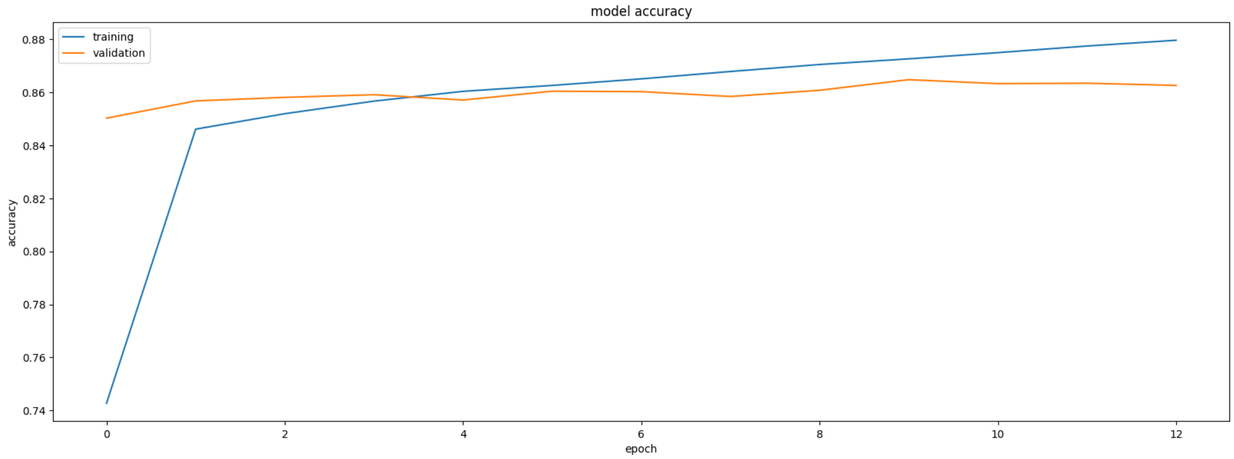
In [31]:
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

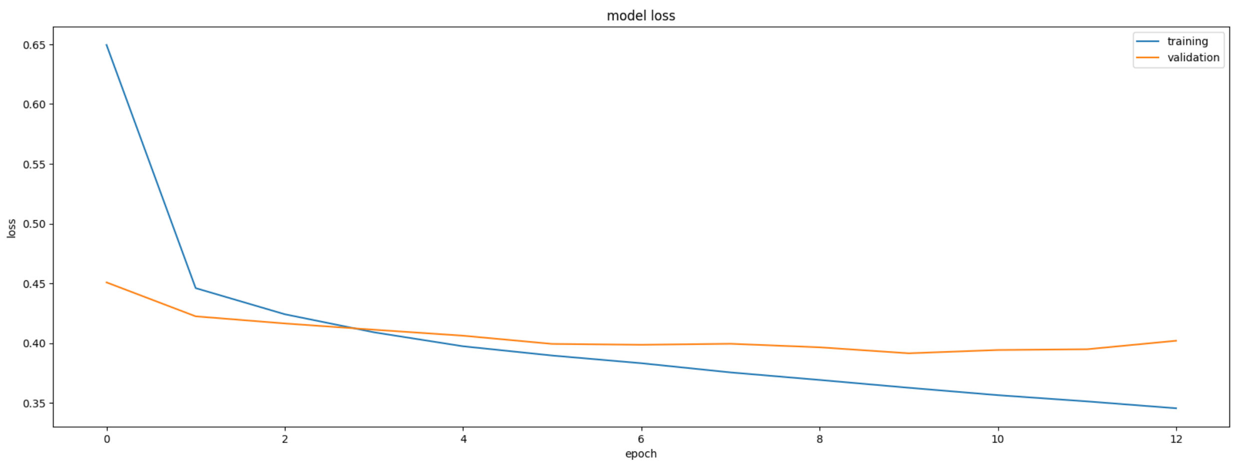
Out[31]:

	loss	accuracy	val_loss	val_accuracy
8	0.369	0.870	0.396	0.861
9	0.363	0.873	0.391	0.865
10	0.356	0.875	0.394	0.863
11	0.351	0.877	0.395	0.863
12	0.345	0.880	0.402	0.863

In [32]:
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-8-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





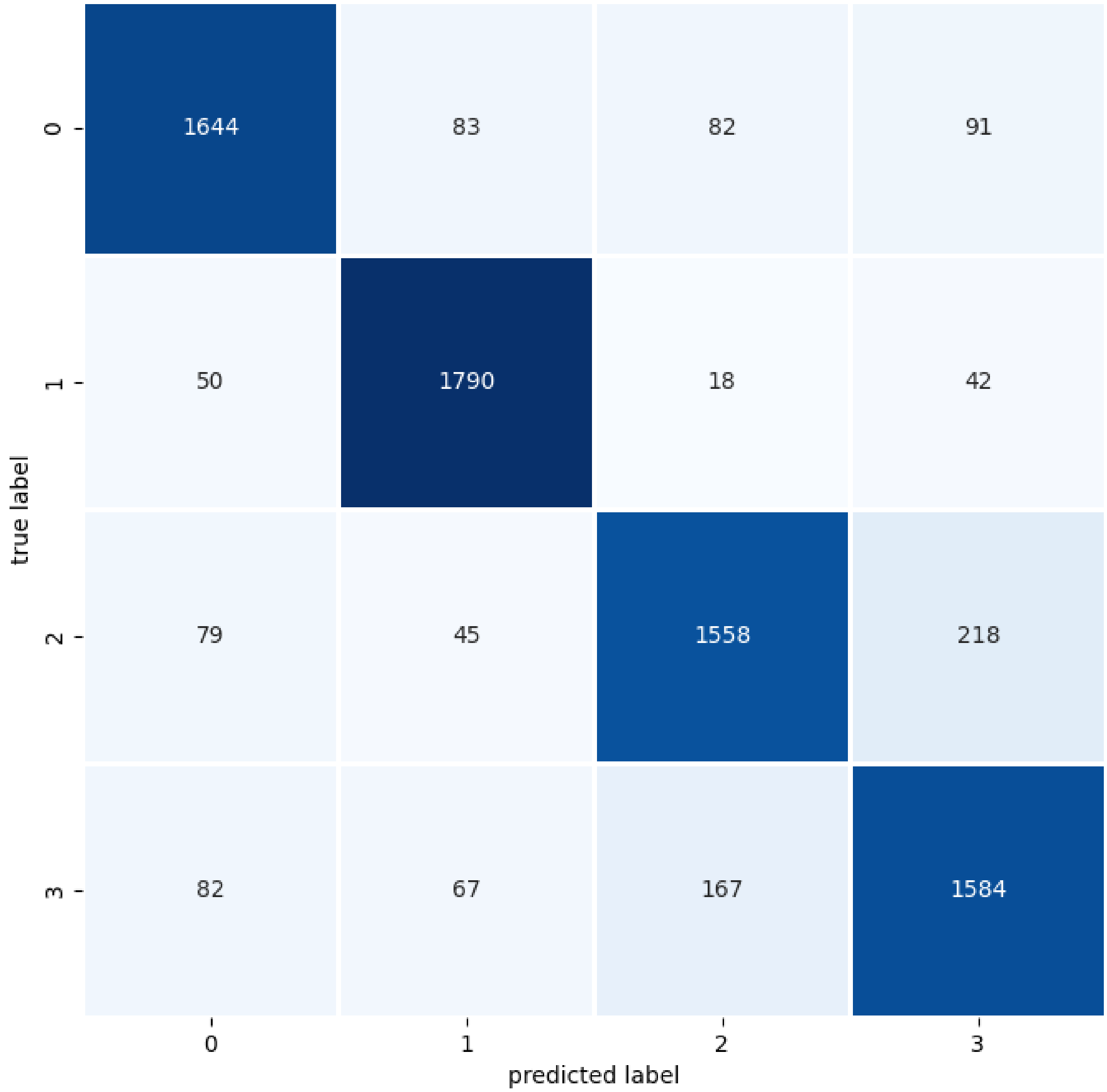
In [33]:
```
y_test = np.concatenate([y for x, y in int_test_ds_two], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_two), axis=-1)
```

```
238/238 [==============================] - 8s 30ms/step
```

In [34]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.87      0.88      1900
           1       0.90      0.94      0.92      1900
           2       0.85      0.82      0.84      1900
           3       0.82      0.83      0.83      1900

    accuracy                           0.87      7600
   macro avg       0.87      0.87      0.86      7600
weighted avg       0.87      0.87      0.86      7600

Accuracy Score: 0.8652631578947368
Root Mean Square Error: 0.6507081163099004
```

In [35]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [36]:

```
train_evaluation = model.evaluate(int_train_ds_two)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_two)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_two)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 120s 33ms/step - loss: 0.3400 - accurac
y: 0.8783
Training accuracy: 0.878
Training loss: 0.340
188/188 [==============================] - 7s 38ms/step - loss: 0.3915 - accuracy: 0.
8647
Validation accuracy: 0.865
Validation loss: 0.391
238/238 [==============================] - 7s 29ms/step - loss: 0.3913 - accuracy: 0.
8653
Testing accuracy: 0.865
Testing loss: 0.391
```

9) Model 8 - Uni-Directional LSTM Model with Larger Vocabulary

9.1) Data Wrangling and Vectorization

In [80]:

```
max_length = 40
max_tokens = 2000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
    standardize=custom_stopwords
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds_three = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds_three = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds_three = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

9.1) Build The Model

In [82]:

```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.LSTM(32)(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
```

```
    tf.keras.callbacks.ModelCheckpoint("Model_Eight",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_three, validation_data=int_val_ds_three, epochs=200, ca

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Eight")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 tf.one_hot (TFOpLambda)     (None, None, 2000)        0         
                                                                 
 lstm (LSTM)                 (None, 32)                260224    
                                                                 
 dropout (Dropout)           (None, 32)                0         
                                                                 
 dense (Dense)               (None, 4)                 132       
                                                                 
=================================================================
Total params: 260356 (1017.02 KB)
Trainable params: 260356 (1017.02 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 360s 99ms/step - loss: 0.6496 - accurac
y: 0.7444 - val_loss: 0.3989 - val_accuracy: 0.8653
Epoch 2/200
3563/3563 [==============================] - 306s 86ms/step - loss: 0.4010 - accurac
y: 0.8711 - val_loss: 0.3775 - val_accuracy: 0.8763
Epoch 3/200
3563/3563 [==============================] - 286s 80ms/step - loss: 0.3829 - accurac
y: 0.8773 - val_loss: 0.3668 - val_accuracy: 0.8775
Epoch 4/200
3563/3563 [==============================] - 291s 82ms/step - loss: 0.3717 - accurac
y: 0.8802 - val_loss: 0.3625 - val_accuracy: 0.8787
Epoch 5/200
3563/3563 [==============================] - 302s 85ms/step - loss: 0.3631 - accurac
y: 0.8829 - val_loss: 0.3615 - val_accuracy: 0.8813
Epoch 6/200
3563/3563 [==============================] - 315s 88ms/step - loss: 0.3542 - accurac
y: 0.8852 - val_loss: 0.3597 - val_accuracy: 0.8785
Epoch 7/200
3563/3563 [==============================] - 299s 84ms/step - loss: 0.3471 - accurac
y: 0.8873 - val_loss: 0.3599 - val_accuracy: 0.8785
Epoch 8/200
3563/3563 [==============================] - 298s 84ms/step - loss: 0.3412 - accurac
y: 0.8887 - val_loss: 0.3600 - val_accuracy: 0.8785
The runtime to fit this model was: 0:42:58.446300.
```

9.2) Evaluate Model Performance

In [83]:
```
history_dict = history.history
history_dict.keys()
```

Out[83]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

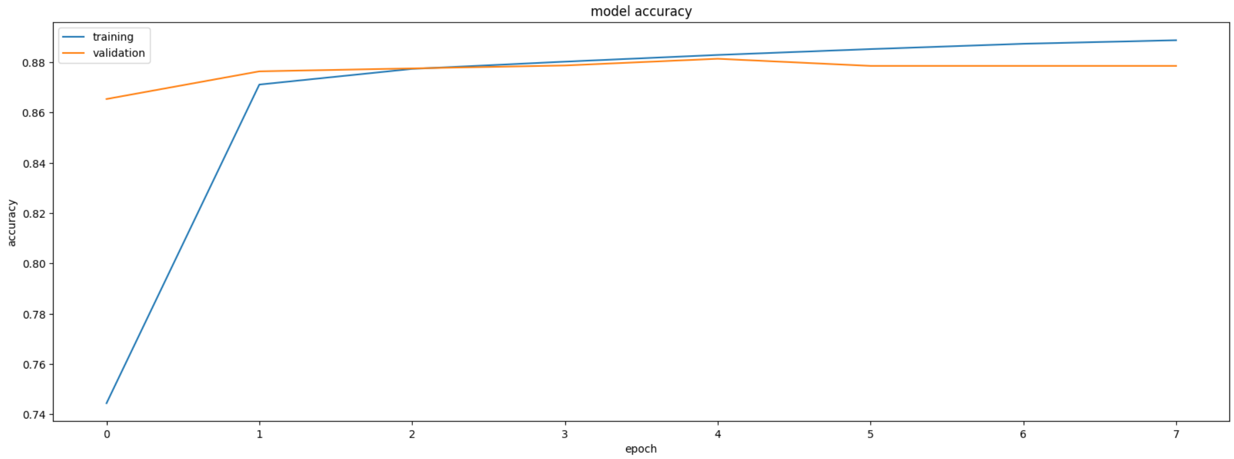
In [84]:
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

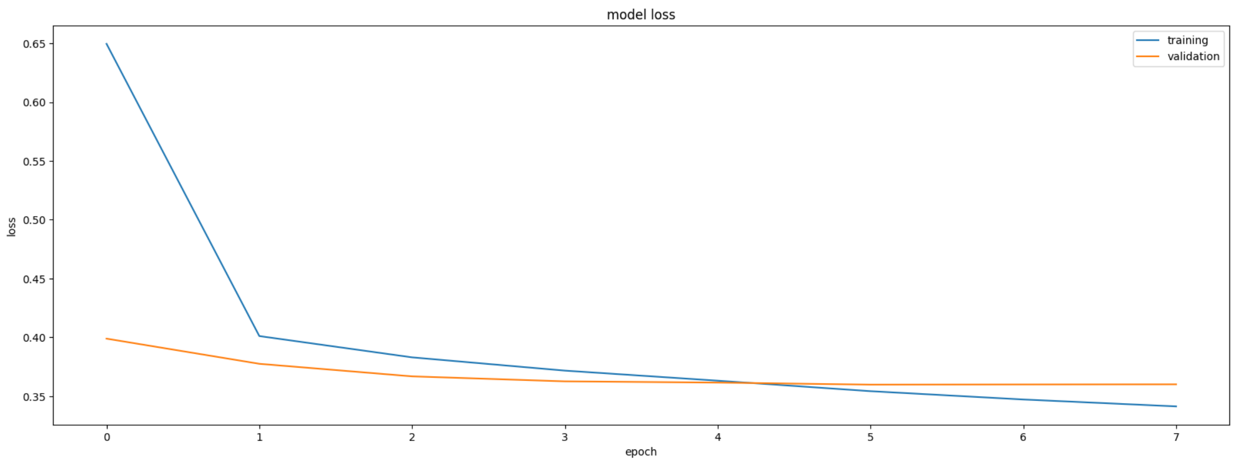
Out[84]:

	loss	accuracy	val_loss	val_accuracy
3	0.372	0.880	0.363	0.879
4	0.363	0.883	0.362	0.881
5	0.354	0.885	0.360	0.878
6	0.347	0.887	0.360	0.878
7	0.341	0.889	0.360	0.878

In [85]:
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-8-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





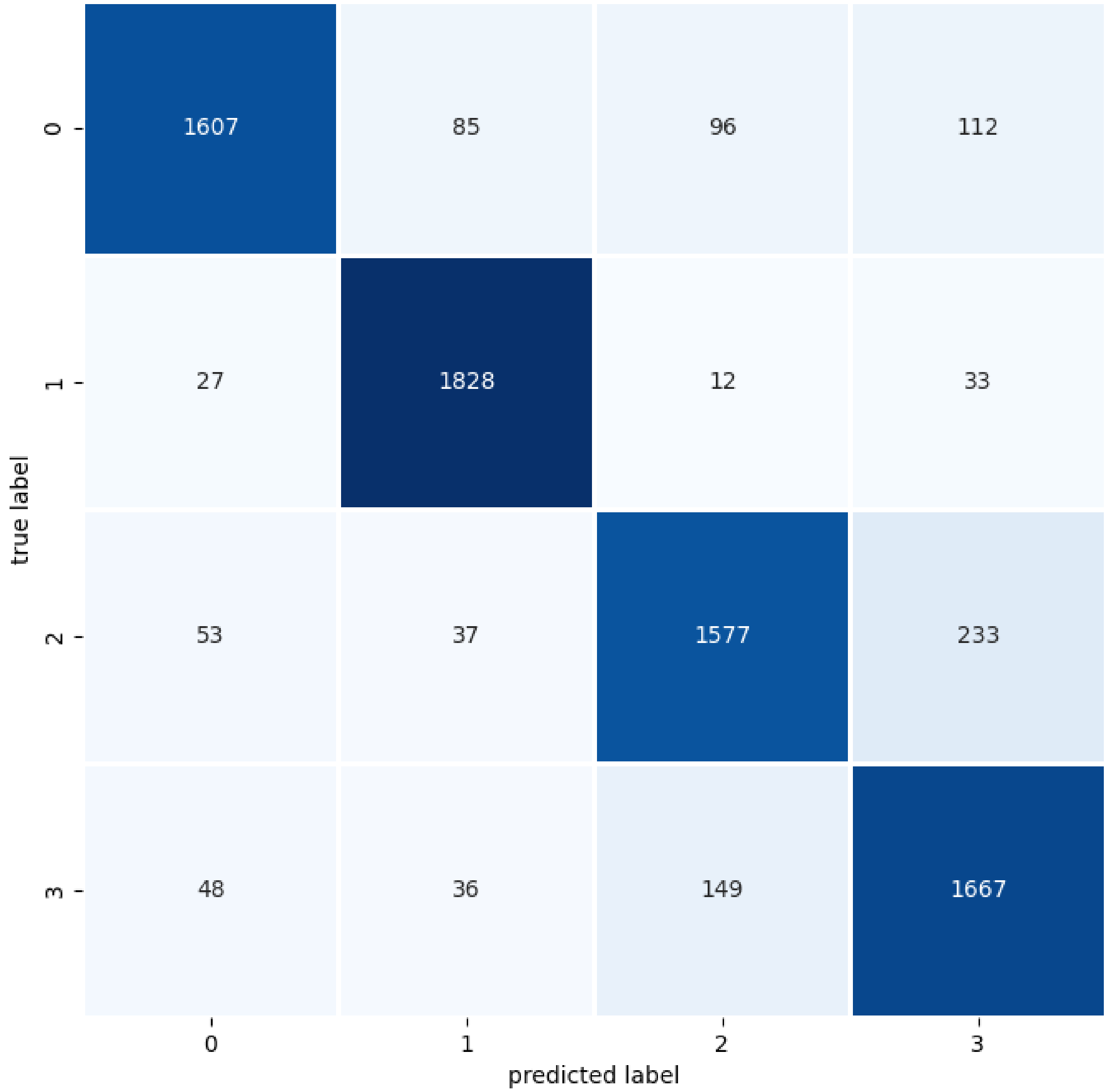
In [86]:
```
y_test = np.concatenate([y for x, y in int_test_ds_three], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_three), axis=-1)
```

```
238/238 [==============================] - 17s 66ms/step
```

In [87]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.93      0.85      0.88      1900
           1       0.92      0.96      0.94      1900
           2       0.86      0.83      0.84      1900
           3       0.82      0.88      0.85      1900

    accuracy                           0.88      7600
   macro avg       0.88      0.88      0.88      7600
weighted avg       0.88      0.88      0.88      7600

Accuracy Score: 0.8788157894736842
Root Mean Square Error: 0.6129093691051247
```

In [88]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [90]:
```
train_evaluation = model.evaluate(int_train_ds_three)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_three)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_three)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 154s 42ms/step - loss: 0.3240 - accurac
y: 0.8911
Training accuracy: 0.891
Training loss: 0.324
188/188 [==============================] - 8s 41ms/step - loss: 0.3597 - accuracy: 0.
8785
Validation accuracy: 0.878
Validation loss: 0.360
238/238 [==============================] - 7s 30ms/step - loss: 0.3695 - accuracy: 0.
8788
Testing accuracy: 0.879
Testing loss: 0.370
```

10) Model 9 - Uni-Directional LSTM Model with Fewer Tokens Per Document

10.1) Data Wrangling

In [91]:
```
max_length = 33
max_tokens = 2000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
    standardize=custom_stopwords
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds_four = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds_four = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds_four = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

10.2) Build The Model

In [93]:
```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.LSTM(32)(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
```

```
    tf.keras.callbacks.ModelCheckpoint("Model_Nine",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_four, validation_data=int_val_ds_four, epochs=200, call

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Nine")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 tf.one_hot (TFOpLambda)     (None, None, 2000)        0         
                                                                 
 lstm (LSTM)                 (None, 32)                260224    
                                                                 
 dropout (Dropout)           (None, 32)                0         
                                                                 
 dense (Dense)               (None, 4)                 132       
                                                                 
=================================================================
Total params: 260356 (1017.02 KB)
Trainable params: 260356 (1017.02 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 262s 73ms/step - loss: 0.5764 - accurac
y: 0.7820 - val_loss: 0.3813 - val_accuracy: 0.8722
Epoch 2/200
3563/3563 [==============================] - 308s 87ms/step - loss: 0.3887 - accurac
y: 0.8737 - val_loss: 0.3661 - val_accuracy: 0.8767
Epoch 3/200
3563/3563 [==============================] - 252s 71ms/step - loss: 0.3725 - accurac
y: 0.8786 - val_loss: 0.3604 - val_accuracy: 0.8763
Epoch 4/200
3563/3563 [==============================] - 269s 76ms/step - loss: 0.3600 - accurac
y: 0.8819 - val_loss: 0.3556 - val_accuracy: 0.8795
Epoch 5/200
3563/3563 [==============================] - 255s 72ms/step - loss: 0.3535 - accurac
y: 0.8842 - val_loss: 0.3557 - val_accuracy: 0.8800
Epoch 6/200
3563/3563 [==============================] - 262s 73ms/step - loss: 0.3452 - accurac
y: 0.8862 - val_loss: 0.3503 - val_accuracy: 0.8817
Epoch 7/200
3563/3563 [==============================] - 251s 70ms/step - loss: 0.3389 - accurac
y: 0.8881 - val_loss: 0.3577 - val_accuracy: 0.8808
Epoch 8/200
3563/3563 [==============================] - 259s 73ms/step - loss: 0.3327 - accurac
y: 0.8894 - val_loss: 0.3495 - val_accuracy: 0.8790
Epoch 9/200
3563/3563 [==============================] - 246s 69ms/step - loss: 0.3285 - accurac
y: 0.8913 - val_loss: 0.3545 - val_accuracy: 0.8762
The runtime to fit this model was: 0:40:53.093932.
```

10.3) Evaluate Model Performance

In [94]:

```
history_dict = history.history
history_dict.keys()
```

Out[94]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [95]:

```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
```

```
val_accs = history.history['val_accuracy']
epochs = len(losses)
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

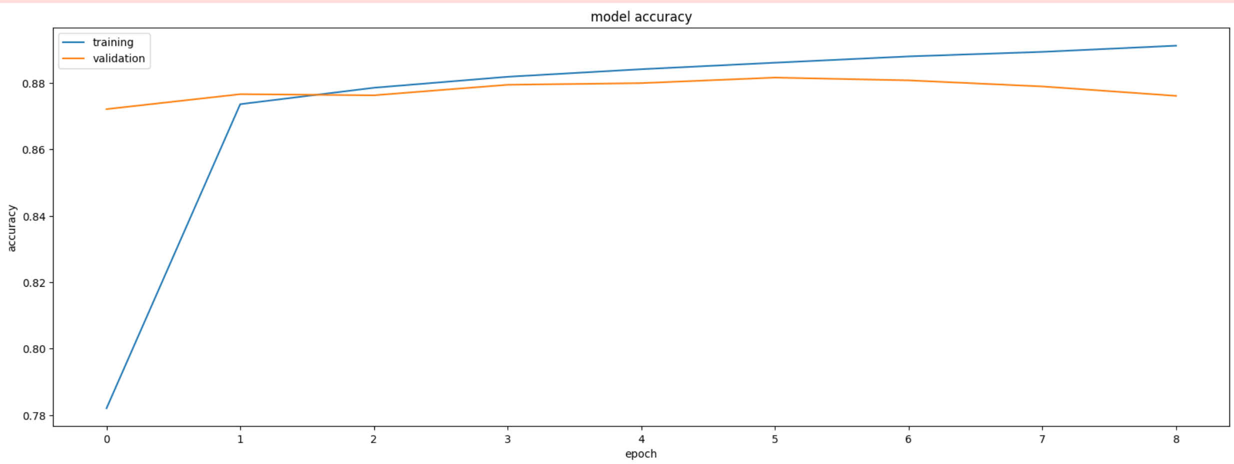
Out[95]:

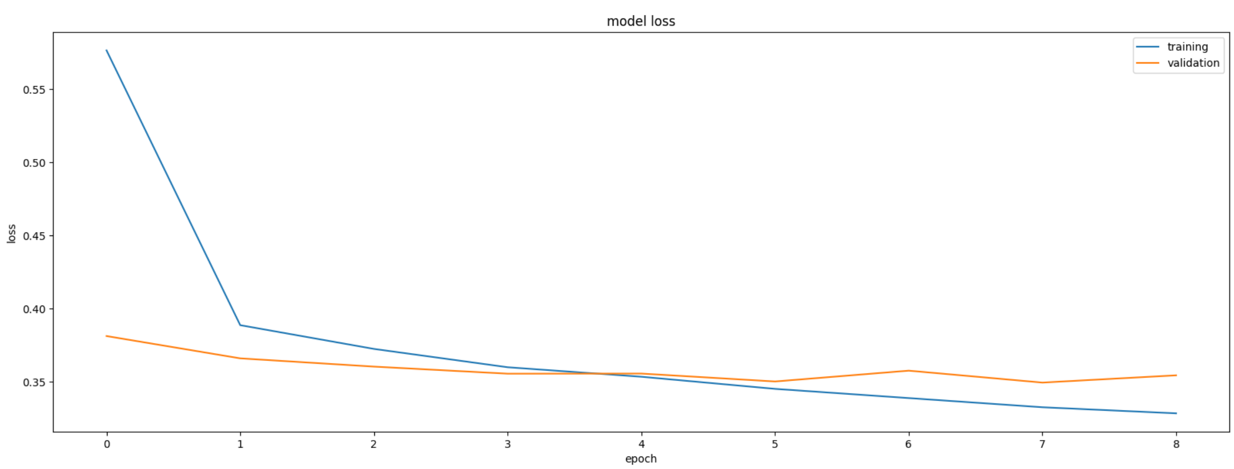
	loss	accuracy	val_loss	val_accuracy
4	0.354	0.884	0.356	0.880
5	0.345	0.886	0.350	0.882
6	0.339	0.888	0.358	0.881
7	0.333	0.889	0.350	0.879
8	0.329	0.891	0.354	0.876

In [96]:

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-8-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





In [97]:

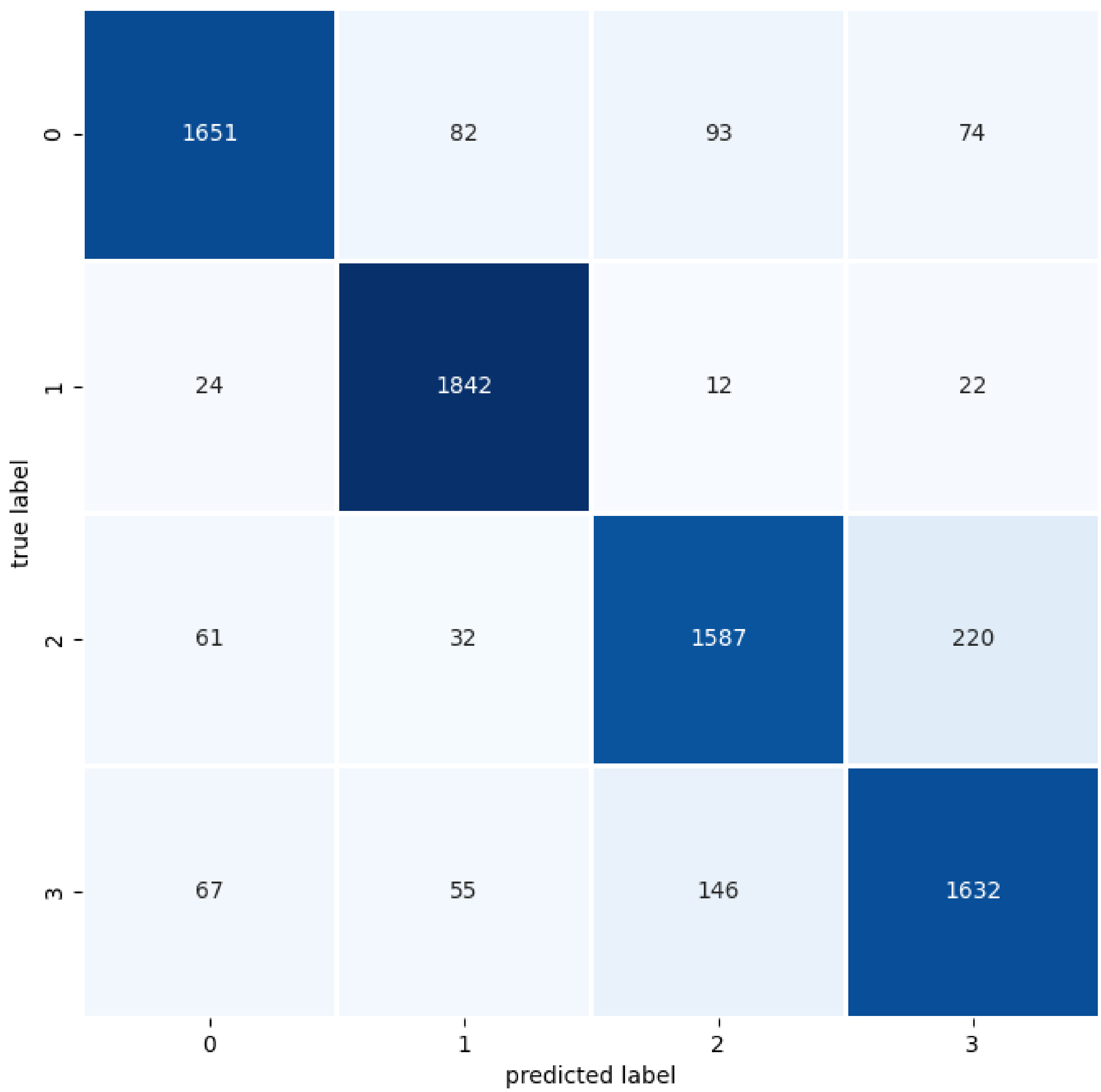
```
y_test = np.concatenate([y for x, y in int_test_ds_four], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_four), axis=-1)
```

```
238/238 [==============================] - 9s 37ms/step
```

In [98]:
```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.87      0.89      1900
           1       0.92      0.97      0.94      1900
           2       0.86      0.84      0.85      1900
           3       0.84      0.86      0.85      1900

    accuracy                           0.88      7600
   macro avg       0.88      0.88      0.88      7600
weighted avg       0.88      0.88      0.88      7600

Accuracy Score: 0.8831578947368421
Root Mean Square Error: 0.5970321334911988
```

In [99]:
```
plot_confusion_matrix(y_test,pred_classes)
```



In [100…

```
train_evaluation = model.evaluate(int_train_ds_four)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_four)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_four)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 101s 28ms/step - loss: 0.3042 - accurac
y: 0.8961
Training accuracy: 0.896
Training loss: 0.304
188/188 [==============================] - 4s 23ms/step - loss: 0.3495 - accuracy: 0.
8790
Validation accuracy: 0.879
Validation loss: 0.350
238/238 [==============================] - 8s 33ms/step - loss: 0.3647 - accuracy: 0.
8832
Testing accuracy: 0.883
Testing loss: 0.365
```

11) Model 10 - Uni-Directional LSTM Model with Different Regularization

11.1) Data Wrangling

In [153…

```
max_length = 30
max_tokens = 2500
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
    standardize=custom_stopwords
)
text_vectorization.adapt(text_only_train_ds)

int_train_ds_five = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_val_ds_five = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
int_test_ds_five = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

11.2) Build The Model

In [154…

```
k.clear_session()
inputs = tf.keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
```

```
x = layers.LSTM(16)(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(4, activation="softmax")(x)
model = tf.keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop",
              loss="SparseCategoricalCrossentropy",
              metrics=["accuracy"])

model.summary()

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Model_Ten",save_best_only=True)
    ,tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
]

start_time = datetime.datetime.now()

history=model.fit(int_train_ds_five, validation_data=int_val_ds_five, epochs=200, call

end_time = datetime.datetime.now()
runtime = end_time - start_time
print(f"The runtime to fit this model was: {runtime}.")


model = keras.models.load_model("Model_Ten")
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None)]            0         
                                                                 
 tf.one_hot (TFOpLambda)     (None, None, 2500)        0         
                                                                 
 lstm (LSTM)                 (None, 16)                161088    
                                                                 
 dropout (Dropout)           (None, 16)                0         
                                                                 
 dense (Dense)               (None, 4)                 68        
                                                                 
=================================================================
Total params: 161156 (629.52 KB)
Trainable params: 161156 (629.52 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
3563/3563 [==============================] - 261s 72ms/step - loss: 0.6574 - accurac
y: 0.7610 - val_loss: 0.3850 - val_accuracy: 0.8718
Epoch 2/200
3563/3563 [==============================] - 214s 60ms/step - loss: 0.4145 - accurac
y: 0.8748 - val_loss: 0.3584 - val_accuracy: 0.8777
Epoch 3/200
3563/3563 [==============================] - 202s 57ms/step - loss: 0.3896 - accurac
y: 0.8807 - val_loss: 0.3614 - val_accuracy: 0.8783
Epoch 4/200
3563/3563 [==============================] - 203s 57ms/step - loss: 0.3773 - accurac
y: 0.8839 - val_loss: 0.3609 - val_accuracy: 0.8802
Epoch 5/200
3563/3563 [==============================] - 214s 60ms/step - loss: 0.3699 - accurac
y: 0.8854 - val_loss: 0.3505 - val_accuracy: 0.8833
Epoch 6/200
3563/3563 [==============================] - 199s 56ms/step - loss: 0.3603 - accurac
y: 0.8879 - val_loss: 0.3600 - val_accuracy: 0.8803
Epoch 7/200
3563/3563 [==============================] - 202s 57ms/step - loss: 0.3553 - accurac
y: 0.8888 - val_loss: 0.3634 - val_accuracy: 0.8787
Epoch 8/200
3563/3563 [==============================] - 198s 55ms/step - loss: 0.3498 - accurac
y: 0.8901 - val_loss: 0.3546 - val_accuracy: 0.8818
The runtime to fit this model was: 0:31:48.128907.
```

11.3) Evaluate Model Performance

In [155…

```python
history_dict = history.history
history_dict.keys()
```

Out[155]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [156…

```python
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

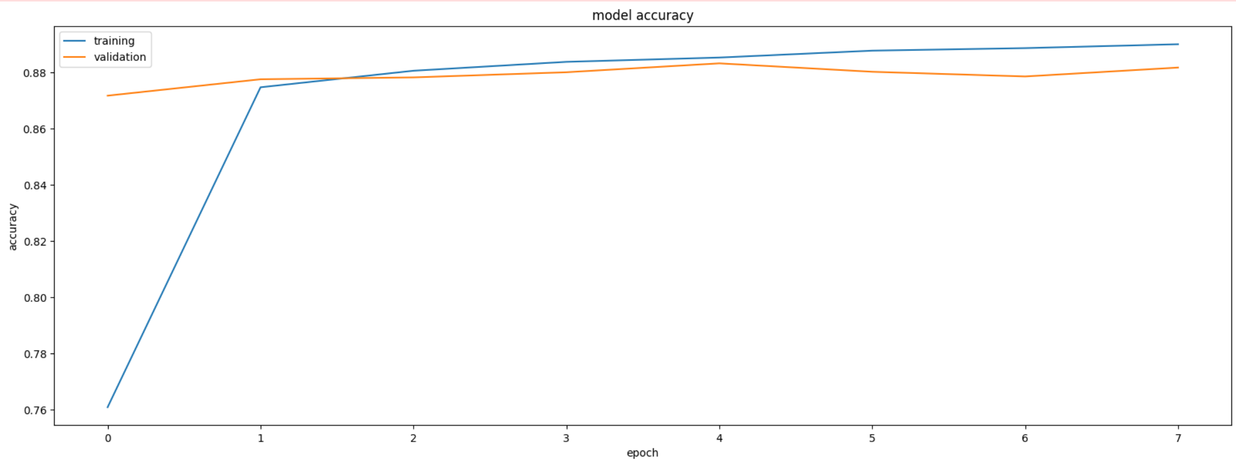
Out[156]:

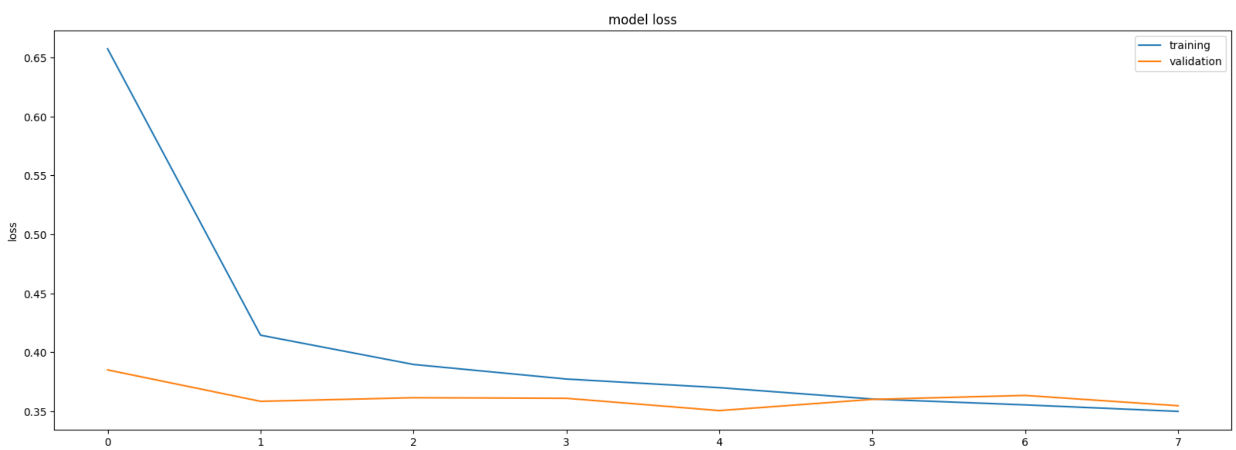
	loss	accuracy	val_loss	val_accuracy
3	0.377	0.884	0.361	0.880
4	0.370	0.885	0.350	0.883
5	0.360	0.888	0.360	0.880
6	0.355	0.889	0.363	0.879
7	0.350	0.890	0.355	0.882

In [157…

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```

```
<ipython-input-8-5294d8a6260d>:23: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```





In [158…

```
y_test = np.concatenate([y for x, y in int_test_ds_five], axis=0)
pred_classes = np.argmax(model.predict(int_test_ds_five), axis=-1)
```

```
238/238 [==============================] - 6s 24ms/step
```

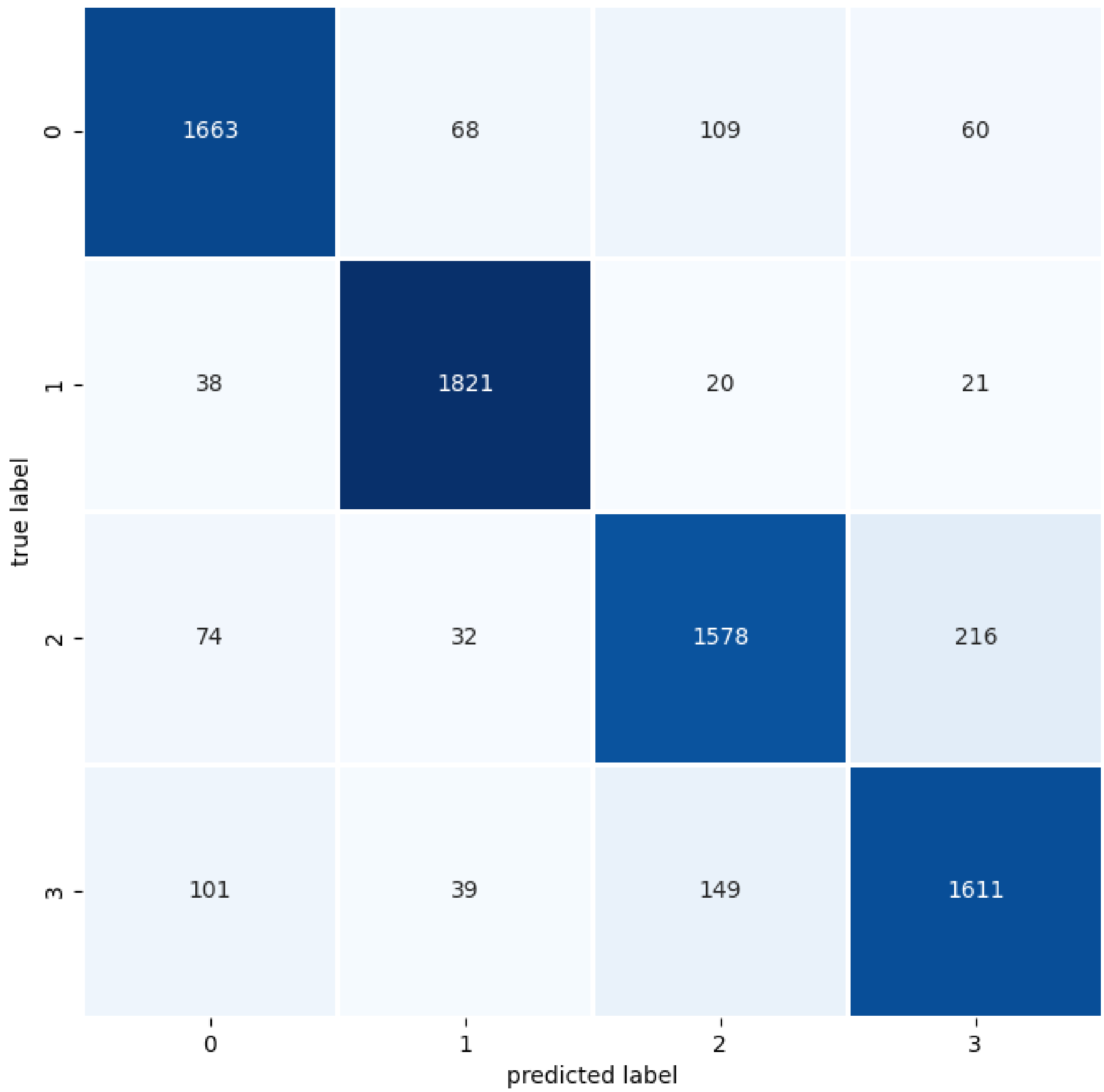
In [159…

```
print_validation_report(y_test, pred_classes)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.88      0.88      1900
           1       0.93      0.96      0.94      1900
           2       0.85      0.83      0.84      1900
           3       0.84      0.85      0.85      1900

    accuracy                           0.88      7600
   macro avg       0.88      0.88      0.88      7600
weighted avg       0.88      0.88      0.88      7600

Accuracy Score: 0.8780263157894737
Root Mean Square Error: 0.6223892841723992
```

In [160…

```
plot_confusion_matrix(y_test,pred_classes)
```



In [161…

```
train_evaluation = model.evaluate(int_train_ds_five)
print(f"Training accuracy: {train_evaluation[1]:.3f}")
```

```
print(f"Training loss: {train_evaluation[0]:.3f}")

validation_evaluation = model.evaluate(int_val_ds_five)
print(f"Validation accuracy: {validation_evaluation[1]:.3f}")
print(f"Validation loss: {validation_evaluation[0]:.3f}")

testing_evaluation = model.evaluate(int_test_ds_five)
print(f"Testing accuracy: {testing_evaluation[1]:.3f}")
print(f"Testing loss: {testing_evaluation[0]:.3f}")
```

```
3563/3563 [==============================] - 81s 22ms/step - loss: 0.3167 - accuracy:
0.8944
Training accuracy: 0.894
Training loss: 0.317
188/188 [==============================] - 5s 25ms/step - loss: 0.3505 - accuracy: 0.
8833
Validation accuracy: 0.883
Validation loss: 0.350
238/238 [==============================] - 5s 19ms/step - loss: 0.3650 - accuracy: 0.
8780
Testing accuracy: 0.878
Testing loss: 0.365
```