

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS5141—Intelligent Systems Laboratory

**Assignment #2 — Data Extraction and Preprocessing** **and Subjectivity Classification**

Prepared by: Miral Issa – 1200527

Instructor: Mohammad Jubran

Assistant: Hanan Awawdeh

Date: January 5, 2025

# Abstract:

This report presents a systematic approach to extracting and preprocessing data for subjectivity classification of English tweets. The methodology involves leveraging convolutional neural networks (CNNs) to process raw image data into text, followed by Long Short-Term Memory (LSTM) networks and transformer-based models for classifying subjectivity. The preprocessing steps include data transformation, tokenization, and embedding. Several models were implemented, including custom-trained CNNs, pretrained CNNs using transfer learning, LSTM models, and transformers fine-tuned on the task. Hyperparameter optimization was conducted for each model, with performance evaluated through metrics such as accuracy, precision, recall, and F1 score. Transformer models, benefiting from transfer learning, demonstrated superior performance compared to LSTMs, emphasizing the advantages of leveraging pretrained architectures. The report also discusses the challenges faced in selecting suitable pretrained models and highlights the importance of data size and hyperparameter tuning for effective model training.

Contents

[Abstract: 2](#_Toc187071662)

[1. Introduction 3](#_Toc187071663)

[CNN model: 3](#_Toc187071664)

[LSTM model: 4](#_Toc187071665)

[Transfer Learning 5](#_Toc187071666)

[2. Procedure and Discussion 6](#_Toc187071667)

[Task 1: Data Extraction and Preprocessing 6](#_Toc187071668)

[CNN model: 6](#_Toc187071669)

[Pre-trained CNN model: 7](#_Toc187071670)

[Task 2: Subjectivity Classification 7](#_Toc187071671)

[LSTM model 8](#_Toc187071672)

[Transfer learning in transformers models 9](#_Toc187071673)

[3. Conclusion 11](#_Toc187071674)

# Table of Figures:

[Figure 1: the work of a convolutional layer 5](#_Toc187071778)

[Figure 2: LSTM cell structure 6](#_Toc187071779)

[Figure 3: confusion matrix of the LSTM model with hyperparameters (100, 5, SGD, 0.01, 50) 10](#_Toc187071780)

[Figure 4: confusion matrix of the transformer model with hyperparameters (100, 4, 0.01, 15 and optimizer SGD) 11](#_Toc187071781)

# Table of Tables:

[Table 1:CNN hyperparameters tuning accuracy comparison 7](#_Toc187071825)

[Table 2: pre trained CNN model hyperparameter tuning accuracy comparison 8](#_Toc187071826)

[Table 3: LSTM models hyperparameters comparison 9](#_Toc187071827)

[Table 4: transformer model hyperparameters comparison 10](#_Toc187071828)

# Introduction

This assignment primary objective is to develop a model that is able to determine the subjectivity of 1,100 tweets in English. But closer to real life, the sentences are presented as a collection of handwritten characters with spaces between them, forming words that would be the body of the sentence.

From here it’s obvious that the raw data will need to go through multiple models, so it can be transformed into text format, and then to train a model that can perform the needed task.

## CNN model:

The first challenge is to extract the characters from the images. The most suitable way to do that is to use a CNN (Convolution Neural Network) to determine which character is written in which image.

A CNN is a type of Deep Learning neural network architecture commonly used in Computer Vision. That enables a computer to understand and interpret the image or visual data.

This model is different from normal neuron networks by the layers it contains, which are:

* **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters, also known as kernels, slide over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. As this layer extract the features from the raw image

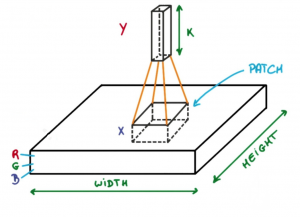


Figure 1: the work of a convolutional layer

* **Pooling layer:** This layer is periodically inserted in the model and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting.
* **Fully connected layers:** typically, the final layers of the CNN model are a normal fully connected layers, that takes the features extracted by the convolutional and pooling layers, and work on classifying them.

## LSTM model:

After extracting the characters from the images using a CNN model, and constructing the sentences, it’s time to build a model that can read these sentences and detect their subjectivity.

To do so our model must be capable of handling sequential connected input like text. Normal neuron networks won’t be able to connect past processed input with current input to conclude the sentence’s context meaning.

On the other hand, LSTM (Long Short Term Memory model) excels in sequence prediction tasks, capturing long-term dependencies. Ideal for time series, machine translation, and speech recognition due to order dependence.

The LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

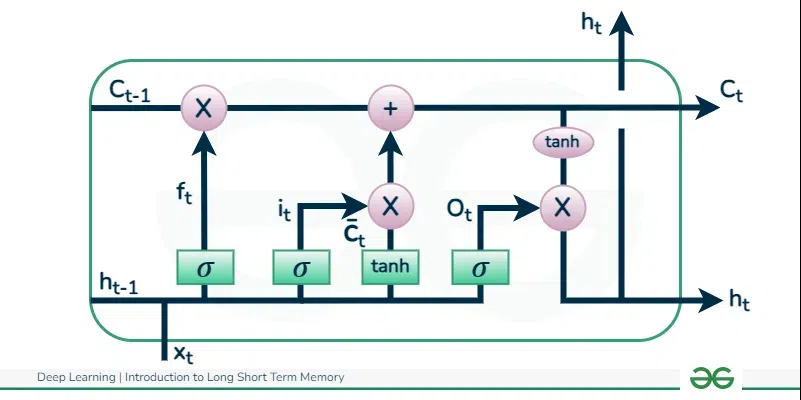


Figure 2: LSTM cell structure

## Transfer Learning

Machine learning models in general are data hungry, but what to do if the data available for training aren’t much?

Use the knowledge of a model trained on a much bigger dataset on a similar task to your advantage. Also known as transfer learning. A model trained on enough data and a similar task is used as the base to the new model.

Initialize the new model weights with the weights learned from the pre-trained model. Freeze the layers that hold generic information applicable to both the original and new task. Train only the other layers on the new data.

This way with the resources at hand, a model can achieve higher accuracy in a shorter training time.

# Procedure and Discussion

## Task 1: Data Extraction and Preprocessing

The file given for this task is a csv file of labels and pixels values of the image. To be precise: 785 columns where the first one is the label and the next 784 represents the pixels of the image. The first step I did was reading the file into a 2d numpy array.

Then I looped over every row of this array, extracted the first item and converted the rest of the array into a PIL image. After some debugging, I discovered that using the transpose of the pixels array to construct the image was the right choice. Finally, I saved the label and the corresponding image in the same index in different lists. Ending up with lists of the length 112800.

To map each label to a character, I displayed an image of each label, and saved the mapping between the label number and the character, in a dictionary.

I saved the data into a dataframe that I split into training and testing sets, then transformed them into a torch dataset, with a train and test data loaders, with batch size of 32. After of course transforming the images to be resized into 32x32 pixels, so I can feed them into the CNN model that I initialized later.

### CNN model:

The CNN had many hyperparameters to tune, such as the batch size, number of layers, optimizer used, learning rate and number of epochs. I ran some experiments with different values of these parameters, and monitored the test accuracy of each model.

Table 1:CNN hyperparameters tuning accuracy comparison

|  |  |
| --- | --- |
| Model (batch size, layers, optimizer, learning rate, epochs) | accuracy |
| 32, 2 conv, SGD, 0.001, 4 | 83.51063829787235% |
| 32, 3 conv, SGD, 0.001, 4 | 80.89095744680851% |
| 32, 3 conv, Adam, 0.001, 4 | 1.7863475177304964% |
| 32, 3 conv, SGD, 0.01, 4 | 83.89627659574468% |
| 100, 3 conv, SGD, 0.01, 6 | 85.80673758865248% |

### Pre-trained CNN model:

Another approach that was tested was using a pretrained model on the EMNIST dataset to also capture the characters. The model I choose to try was a model published on GitHub that reached an accuracy of 92.43%. After adding a new dense layer to it (that output the correct number of classes for my data set), I did a couple of edits to its hyperparameters, and monitored the test accuracy on my data.

Table 2: pre trained CNN model hyperparameter tuning accuracy comparison

|  |  |
| --- | --- |
| Model (batch size, optimizer, epochs) | accuracy |
| 32, Adam, 3 | 0.5301 |
| 100, SGD, 6 | 0.5409 |
| 100, SGD, 6, with learning rate = 0.01, and last 4 layers un-freezed | 0.5703 |
| 100, SGD, 10, with learning rate = 0.01 | 0.8193 |
| 100, SGD, 15, with learning rate = 0.01 | 0.8308 |

In this case surprisingly enough the pretrained model performed worse than the normal CNN model, this could due to choosing a pretrained model trained on a different task, or unsimilar data. or it could be because of the hyperparameters chosen for the model, or the added layer. All these could be reasons why the model ended worse than before.

I did try re-constructing the sentences using the CNN model that reached 83% accuracy, and put the constructed sentences into a spell checker. The processed data in the scentences\_ready.csv file. These are example of the sentences I got:

0 gone ARE THE DAYS WHEN THEr red the WORLD IN R...

1 tWE TREND IS EXPECTED TO reverse AS don is NEX...

2 BUT tHERE IS THE specious point again

3 HE ADDED HE WOVLhVT BE surprised TO SEE A NEW ...

4 NOT less government YOU SEE THE SAMI AMOUNT Of...

In the end I decided to continue task 2 with the tsv files provided.

## Task 2: Subjectivity Classification

Uploading the training data into a dataframe, and encoding the label column into a numerical data type, so it can be processed by the models later. Another preprocessing step applied was removing any special characters from the sentences (keeping only letters and numbers), removing and stripping any continuous white space if found, and finally converting the whole sentence into lowercase letters.

After these steps, the dataframe was transformed into a dataset, that tokenization, removal of step worlds and stemming were applied to. Alongside embedding tokens vectors, and indexing them. Finally, splitting the dataset into training, validation and testing sections. Which later a data loader for each was initialized.

### LSTM model

A number of LSTM models were trained on the data, each with different hyperparameters, and were compared by four metrics: accuracy, precision, Recall, f1\_score.

Table : LSTM models hyperparameters comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model (batch size, layers, optimizer, learning rate, epochs) | accuracy | precision | Recall | f1\_score. |
| 4, 2, Adam, 0.001, 50 | 0.6506024 | 0.6506024 | 0.6506024 | 0.6506024 |
| 100, 2, SGD, 0.001, 50 | 0.6506024 | 0.6506024 | 0.6506024 | 0.6506024 |
| 100, 5, SGD, 0.01, 50 | 0.3493975 | 0.3493975 | 0.3493975 | 0.3493975 |

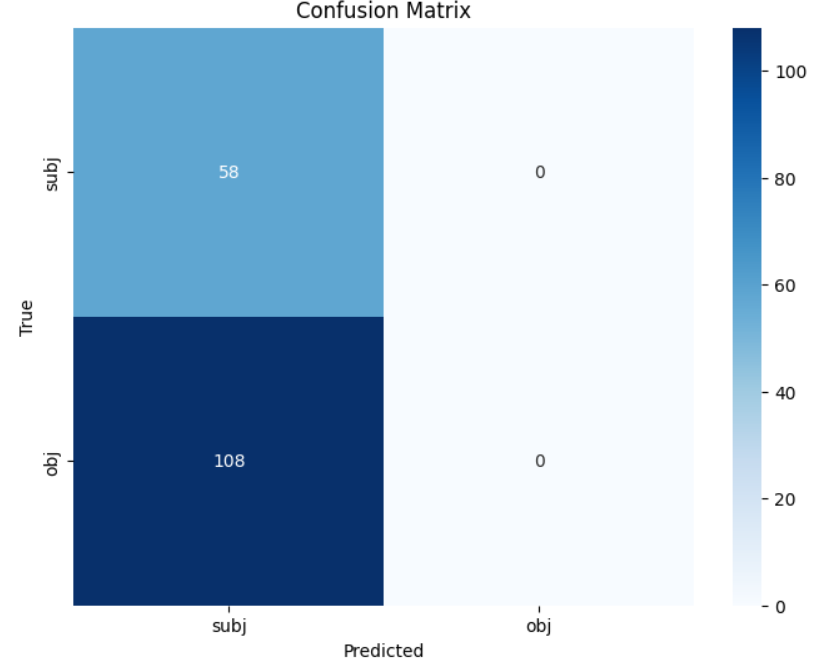


Figure : confusion matrix of the LSTM model with hyperparameters (100, 5, SGD, 0.01, 50)

### Transfer learning in transformers models

A transformer model pretrained on Sequence Classification was used as the base for my model, then the model was re-trained on my data using different hyperparameters.

Table : transformer model hyperparameters comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model (batch size, layers, weight decay ,epochs) | accuracy | precision | Recall | f1\_score. |
| 512, 2, 0.01, 10 | 0.7108433 | 0.7108433 | 0.7108433 | 0.7108433 |
| 100, 3, 0.01, 10 | 0.7289156 | 0.7289156 | 0.7289156 | 0.7289156 |
| 100, 4, 0.01, 15 and optimizer SGD | 0.7469879 | 0.7469879 | 0.7469879 | 0.7469879 |

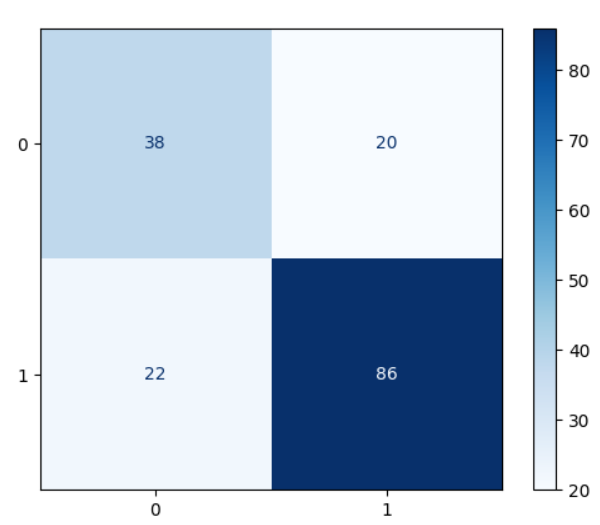


Figure : confusion matrix of the transformer model with hyperparameters (100, 4, 0.01, 15 and optimizer SGD)

In this case the pretrained transformer model performed much better that the LSTM models, due to it being trained previously on a similar task and a bigger data.

# Conclusion

CNN are a very strong machine learning model for computer vision applications, as it is made specifically to handle raw image data, learn to extract needed future from it automatically, and classify the image as needed. However, it suffers from too many parameters, wither they are hyperparameters the developer must tune, which will need too much training time and resources. Or learnable parameters, that need big data to train effectively.

In the scenario where big data isn’t available, transfer learning come to the rescue, still it’s not a magical solution perfect however it’s used, the developer must choose a suitable pretrained model trained on a similar data and task, in order to achieve good performance metrics on the new data and task.

Text processing could be challenging for machine learning as the true meaning of the text depend on its context, and long ago processed words could still influence newly inputted words classification. LSTM and transformers are great models for processing sequential data such as text. But again, they need tuning and too much data to train. And still transfer learning could prove to be a practical solution to such problems.

Some challenges I faced in this assignment was choosing a suitable pretrained models to use in the transfer learning, and the individuality of each model code to perform the transfer learning and train the model on the new data. if I would do the experiment again, I’ll try different models.